

Benjamin M. Abdel-Karim *Hrsg.*

Data Science

Best Practices mit Python

EBOOK INSIDE



Springer Vieweg

Data Science

Benjamin M. Abdel-Karim
(Hrsg.)

Data Science

Best Practices mit Python

Hrsg.
Benjamin M. Abdel-Karim
Frankfurt am Main, Hessen, Deutschland

ISBN 978-3-658-33459-8 ISBN 978-3-658-33460-4 (eBook)
<https://doi.org/10.1007/978-3-658-33460-4>

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

© Der/die Herausgeber bzw. der/die Autor(en), exklusiv lizenziert durch Springer Fachmedien Wiesbaden GmbH, ein Teil von Springer Nature 2022

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von allgemein beschreibenden Bezeichnungen, Marken, Unternehmensnamen etc. in diesem Werk bedeutet nicht, dass diese frei durch jedermann benutzt werden dürfen. Die Berechtigung zur Benutzung unterliegt, auch ohne gesonderten Hinweis hierzu, den Regeln des Markenrechts. Die Rechte des jeweiligen Zeicheninhabers sind zu beachten.

Der Verlag, die Autoren und die Herausgeber gehen davon aus, dass die Angaben und Informationen in diesem Werk zum Zeitpunkt der Veröffentlichung vollständig und korrekt sind. Weder der Verlag noch die Autoren oder die Herausgeber übernehmen, ausdrücklich oder implizit, Gewähr für den Inhalt des Werkes, etwaige Fehler oder Äußerungen. Der Verlag bleibt im Hinblick auf geografische Zuordnungen und Gebietsbezeichnungen in veröffentlichten Karten und Institutionsadressen neutral.

Planung: Petra Steinmueller

Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer Fachmedien Wiesbaden GmbH und ist ein Teil von Springer Nature.

Die Anschrift der Gesellschaft ist: Abraham-Lincoln-Str. 46, 65189 Wiesbaden, Germany

Vorwort

Im Rahmen meines Studiums der Wirtschaftsinformatik und innerhalb meiner Zeit als wissenschaftlicher Mitarbeiter und Forscher habe ich zahlreiche Probleme mithilfe der Programmierung lösen können, wie zum Beispiel die professionelle Datenanalyse oder das Implementieren von künstlichen Intelligenzen. Vor diesem Hintergrund kann ich gut nachvollziehen, welchen Herausforderungen Einsteiger¹ im Bereich der Programmierung gegenüberstehen. Ausgehend von den zahlreichen Lehrbüchern und Vorlesungen habe ich mir mit diesem Buch das persönliche Ziel gesetzt, ein Werk zu schaffen, das mit simplen Beispielen und einfachen Erklärungen den Zugang zur Programmierung, insbesondere im Bereich der Datenanalyse, ermöglicht. Hierbei möchte ich dem interessierten Leser ein Werk an die Hand geben, das meine wertvollsten Quellcode-artefakte beinhaltet, um somit einen Ansatzpunkt zu schaffen, die eigenen (alltäglichen) Datenanalyseherausforderungen zu lösen.

Ich habe dieses Buch aus der Motivation heraus geschaffen, eines der ersten deutschsprachigen Nachschlagewerke zu entwickeln, in dem relativ simple Quellcodebeispiele enthalten sind, um Lösungsansätze für die (wiederkehrenden) Programmierprobleme in der Datenanalyse weiterzugeben. Hinzu kommt, dass ich selbst wiederholt Lösungswege für ähnliche Probleme erarbeitet habe, die ich bereits in der Vergangenheit entwickelt hatte. Zweifellos gehört das Nachschlagen von Lösungsansätzen in Büchern oder im Internet zur normalen Arbeit eines Programmierers. Allerdings ist diese Suche in der Regel ein unstrukturierter und somit meist zeitaufwendiger Prozess. Entsprechend ist dieses Werk auch für mich eine Zusammenfassung wichtiger Lösungswege für die Grundprobleme des Programmierens.

Unabhängig davon, ob Sie das Buch als Studierender, Mitarbeiter, Gründer eines Start-ups oder als Unternehmer lesen, hoffe ich, dass Ihnen dieses Nachschlagewerk ein wertvoller Helfer für die ersten Anfänge sein wird. Ich gehe davon aus, dass jede Person

¹Aus Gründen der besseren Lesbarkeit wird im Folgenden in der Regel die männliche Form verwendet. Allerdings betonen alle Autoren, dass gleichermaßen alle Geschlechterformen angesprochen sind.

die Grundlagen der Datenanalyse mithilfe moderner Programmiersprachen erlernen kann. Der Erfolg der Programmiersprache Python zeigt, dass die Programmierung kein aufwendiges Unterfangen sein muss. In den letzten Jahren ist Python in Forschung und Praxis zu einer der beliebtesten Programmiersprachen avanciert. Ein zentraler Vorteil von Python ist seine Verständlichkeit, besonderes für Anfänger. Hinzu kommt, dass Python über eine große Entwicklergemeinschaft verfügt. Dies führt zur Bereitstellung zahlreicher und kostenfreier Erweiterungen. Durch meine Erfahrungen als Wissenschaftler im Bereich der künstlichen Intelligenz, Universitätsdozent und Redner, weiß ich, dass viele Menschen eine gewisse Einstiegshürde mit Blick auf die Programmierung verspüren, die sich aber schnell nach den ersten Erfolgserlebnissen legt. Genau jenes Erfolgserlebnis kann zu einem regelrechten Motivator werden, um immer tiefer in die Programmierung einzusteigen.

Ausgehend von meinen Erfahrungen, lernt man am besten durch einfache Beispiele. Das Lösen solcher Aufgaben ist aus meiner Perspektive besonderes wichtig, um ein Grundverständnis zu entwickeln. Vor diesem Hintergrund ist eine zentrale Prämisse des Buchs, dass alle Lösungsansätze so einfach wie möglich gehalten sind, was Sie als Leser befähigen soll, Ansätze für die Lösung Ihrer eigenen Probleme zu entwickeln. Sie benötigen für dieses Buch keine tiefgreifenden Vorkenntnisse in den Bereichen Programmierung oder Datenanalyse. Sofern Sie einen Computer bedienen können und in der Lage sind, Software zu installieren, können Sie mit dem Lesen des Buchs beginnen.

Dieses Werk grenzt sich von anderen Büchern ab, da es ein Nachschlagewerk für Quellcodeansätze im täglichen Data Science sein soll. Aus meiner Erfahrung sind viele einführende Lehrbücher in Python als universelle Bücher konzipiert, sodass sie viele Teilbereiche abdecken und ausführliche Erklärungen bieten. Damit sind diese Lehrbücher als Lernwerke im Sinn eines ausführlichen Studiums konzipiert, was dazu führt, dass der interessierte Leser einem solchen Lehrbuch ausreichend Zeit widmen muss. Im Gegensatz dazu soll mein Werk praktisch anwendbar sein und beim aktiven Quellcode schreiben helfen, sodass es einfach aus dem Regal genommen werden kann, um eine schnelle Hilfestellung bei aufkommenden Fragen zu liefern. Alternativ kann das Buch als Inspirationsgeber für die Umsetzung eigenständiger Projekte dienen. Mit diesem Buch möchte ich den Leser dazu befähigen, den Data-Science-Herausforderungen zu begegnen. Meine Co-Autoren und ich haben in diesem Werk möglichst auf Fachvokabular zugunsten von Alltagssprache verzichtet, um die Verständlichkeit zu erhöhen. Notwendige Fachbegriffe werden an geeigneten Stellen ausführlich erklärt. Durch das Stichwortverzeichnis lassen sich außerdem entsprechende Begrifflichkeiten schnell finden und das Lexikon bietet eine Übersicht mit Kurzerklärungen des Data-Science-Sprachgebrauchs.

An dieser Stelle möchte ich mich bei allen Personen bedanken, die sich unmittelbar oder indirekt an diesem Buchprojekt beteiligt haben. Mein Dank gilt besonders meinen Co-Autoren, die durch ihre unterschiedlichen Ideen und ihre Mitarbeit dieses Buch maßgeblich vorangebracht haben. Ich verzichte aus Platzgründen bewusst auf

die einzelne namentliche Nennung, möchte mich aber neben den Co-Autoren auch bei meinen Kollegen, Studenten und Familienmitgliedern bedanken.

Programmierung ist für mich vergleichbar mit dem Erstellen eines eigenen Kunstwerks. Jeder eigene Programmcode trägt die Handschrift des Schöpfers. Daher freue ich mich besonders darüber, dass der interessierte Leser in diesem Buch die Gelegenheit erhält, unterschiedliche Stile der Programmierung kennenzulernen. Daher wäre es schön, wenn Sie als Leser die unterschiedlichen Ideen und Quellcodes so betrachten, als würden Sie durch eine Galerie schlendern.

Sie finden alle Inhalte zu diesem Buch auf der Webseite des Buchs: <https://github.com/BenjaminMAK/DataScienceBestPractices>

Über Anregungen und Kritik zu diesem Buch, auch über Hinweise zu Entwicklungsmöglichkeiten oder Wünsche für zukünftige Auflagen, freue ich mich sehr. Gern können Sie mich über die Webseite zum Buch kontaktieren.

Nun wünsche ich allen Lesern einen interessanten Einstieg in die Programmierung und viel Freude beim Lesen und Ausprobieren.

Hochachtungsvoll im Namen aller Co-Autoren Ihr:
Benjamin M. Abdel-Karim

Herausgeber- und Autorenverzeichnis

Über den Herausgeber



Dr. Benjamin M. Abdel-Karim ist seit März 2018 als wissenschaftlicher Mitarbeiter an der Professur für Wirtschaftsinformatik und Informationsmanagement von Prof. Dr. Oliver Hinz tätig. Er hat Wirtschaftsinformatik an der Technischen Universität in Darmstadt (M.Sc.) studiert. Im Masterstudium lagen seine Schwerpunkte in den Bereichen Data Knowledge Engineering/Artificial Intelligence und Finanzierung. Die Masterarbeit befasst sich mit der Modellierung spezieller neuronaler Netze zur Abbildung komplexer Finanzmarktstrukturen zwecks der Prognose. Sein Bachelorstudium in Wirtschaftsinformatik (B.Sc.) absolvierte er an der Universität Bremen. Der Schwerpunkt seines Bachelorstudiums war Computational Finance. Praktische Erfahrungen konnte Benjamin M. Abdel-Karim unter anderem durch seine Bankausbildung und Mitarbeit in zahlreichen Banken, wie beispielsweise der Deutsche Asset & Wealth Management, sammeln. Zudem kommen zahlreiche Auftritte als Keynote Speaker im Bereich maschinelles Lernen und Data Science dazu.

Autorenverzeichnis



Dr. Kevin Bauer trat 2013 in das Doktorandenprogramm der GSEFM ein, nachdem er seinen Bachelor-Abschluss in Wirtschaftswissenschaften an der Goethe-Universität erworben hatte. Im Jahr 2017 schloss er seinen ersten Master-Abschluss in Law and Quantitative Economics ab. Im Jahr 2018 schloss er seine Promotion mit dem akademischen Titel Dr. rer. pol. mit einem summa cum laude ab. Seine Doktorarbeit *On the Economic Significance of Social Groups: New Evidence on Self-Selection, Social Identity and Social Preferences* konzentriert sich auf die empirische Analyse von Wirtschaftsbeziehungen. Während seiner Promotion begann er einen zweiten Masterstudiengang in Wirtschaftsinformatik an der Goethe-Universität mit den Schwerpunkten maschinelles Lernen und Künstliche Intelligenz. Nach seiner Promotion trat Kevin Bauer als KI-Spezialist dem TechQuartier bei, wo er weiterhin als externer Berater für KI-bezogene Themen tätig ist. Seit 2020 ist Kevin Bauer als Postdoc-Forscher am Leibniz SAFE in der Forschungsgruppe Digitalisierung in der Finanzindustrie (Leitung: Prof. Hinz) beschäftigt.



Daniel Franzmann erlangte seinen Bachelor in Wirtschaftswissenschaften mit dem Schwerpunkt in Finanzen und Accounting an der Goethe Universität Frankfurt. Nach einem Auslandssemester an der York University in Großbritannien und erster Berufserfahrung bei PwC und Senacor Technologies AG absolvierte er im Jahr 2016 seinen Master in Wirtschaftsinformatik mit dem Fokus auf Data Science an der Goethe Universität Frankfurt. Von 2016 bis 2020 war Daniel Franzmann am Lehrstuhl für Information Systems Engineering der Goethe Universität Frankfurt als Data Scientist angestellt und promovierte über das Thema *Software Updates*. Seit Oktober 2020 arbeitet Daniel Franzmann im Data Analytics Team der Deutschen Bank. Privat interessiert er sich fürs Laufen, Kochen und surft gerne auf r/dataisbeautiful.



Hendrik Jöntgen absolvierte seinen Bachelor und Master in Wirtschaftsinformatik an der Technischen Universität Darmstadt. Im Masterstudium lag sein Schwerpunkt in dem Bereich Informationsvisualisierung und durch langjährige Arbeitserfahrung als studentische Hilfskraft an verschiedenen Lehrstühlen konnte er bereits früh Erfahrungen mit dem Erheben und Auswerten von Social Media Daten sammeln. In seiner Masterarbeit untersuchte er Vertrauensbildung innerhalb der Sharing Economy und die Effekte von unterschiedlichen Profil-Charakteristiken auf die Wahrnehmung der Plattform-Nutzer. Seit März 2019 ist Hendrik Jöntgen als wissenschaftlicher Mitarbeiter am Lehrstuhl für Wirtschaftsinformatik und Informationsmanagement von Prof. Dr. Oliver Hinz tätig und erforscht Vertrauen und soziales Kapital auf Social Media Plattformen. Seine Freizeit verbringt er gerne mit Brettspielen, am Schlagzeug oder auf Waldwegen.



Katharina Keller studierte Wirtschaftsinformatik an der Goethe-Universität Frankfurt (M.Sc.). Ihr Bachelorstudium absolvierte sie in Wirtschaftswissenschaften mit Schwerpunkt Marketing/Management. Während ihres Bachelorstudiums verbrachte sie ein Auslandssemester an der Católica Lisbon School of Business and Economics. Im Juni 2017 schloss sie ihr Studium mit dem akademischen Grad Master of Science ab. Ihre Masterthesis schrieb sie zum Thema „Technology Choice Behavior in Post-Adoption IS Usage“. Seit Juli 2017 ist Katharina Keller als wissenschaftliche Mitarbeiterin an der Professur von Prof. Dr. Oliver Hinz tätig. Zuerst am Lehrstuhl für Wirtschaftsinformatik | Electronic Markets an der TU Darmstadt und seit September 2017 an der Professur für Wirtschaftsinformatik und Informationsmanagement an der Goethe-Universität. Sie ist Teil des Sonderforschungsbereichs 1053 MAKI (Multi-Mechanismen-Adaption für das künftige Internet), der sich mit Fragen zum Thema künftiges Internet und dessen Auswirkung auf das Kommunikationsverhalten befasst. Außerdem betrachtet der SFB 1053 die Mechanismen in Kommunikationssystemen sowie die daraus resultierenden

Anforderungen an die Infrastruktur. Im Speziellen ist Katharina Keller im Teilprojekt B3 beteiligt, das die Transitionen in Kommunikationssystemen aus ökonomischer Perspektive untersucht und dezentrale Analyse- und Planungslösungen für autonome Endgeräte erstellt. Ziel ist eine effiziente und realistische Anwendbarkeit proaktiver Transitionen für koexistente Multi-Mechanismen in drahtlosen Netzen software-definierter autonomer Knoten.



Marcel Zeuch studierte Mathematik an der Technischen Universität in Darmstadt (M.Sc.). Sein erstes Mastersemester absolvierte er dabei 2016 als Erasmus-Stipendiat an der Politecnico di Milano in Mailand, Italien. Seither sammelte er Erfahrungen in einer Managementberatung, einer Top Tier Investmentbank sowie seit seinem Abschluss im Jahr 2018 in einer datengetriebenen Boutiqueberatung im Capital Markets Umfeld. In seiner aktuellen Position im Bereich Corporates & Markets bei einer deutsch-französischen Investmentbank beschäftigt er sich mit Themen der internen Prozessautomatisierung und -weiterentwicklung. In seiner Freizeit verfasste er bereits mehrfach Artikel für einen deutschen Finanzblog und interessiert sich für Themen aus den Bereichen Capital Markets, Data Science und Entrepreneurship.

Inhaltsverzeichnis

Teil I Grundlagen der Programmierung

1	Einleitung	3
	Benjamin M. Abdel-Karim	
1.1	Gegenstandsbereich dieses Buchs	3
1.2	Aufbau und Zielsetzung	4
1.3	Warum Python?	5
2	Python: Installation und Einstieg	7
	Benjamin M. Abdel-Karim	
2.1	Python	7
2.2	Installation	8
2.3	Entwicklungsumgebung	9
2.4	Formalitäten für ein geeignetes Skript	13
3	Primitive Datentypen	17
	Benjamin M. Abdel-Karim	
3.1	Grundlagen zum Verständnis von Programmiersprachen	18
3.2	Ganze Zahlen	19
3.3	Gleitkommazahlen	20
3.4	Zeichen und Zeichenketten in der Programmierung	21
3.5	Boolean	21
4	Datenstruktur	25
	Benjamin M. Abdel-Karim	
4.1	Listen in Python	25
4.1.1	Liste als Stapelspeicher (Stack)	28
4.1.2	Listen als Warteschlange (Queue)	29
4.2	Dictionaries in Python	30
4.3	Mengen (Sets) in Python	32

5	Kontrollstrukturen	35
	Benjamin M. Abdel-Karim	
5.1	Verzweigungen	36
5.1.1	Die If-Else-Bedingungen	36
5.1.2	Verschachtelte If-else-Bedingungen	39
5.2	Schleifen	41
5.2.1	Die For-Schleife	41
5.2.2	Die While-Schleife	44
5.3	Try-except-Bedingung	45
6	Funktionen	49
	Benjamin M. Abdel-Karim	
6.1	Built-in-Funktionen	49
6.2	Funktionen	50
6.3	Bibliotheken (Module) in Python	53
Teil II Data Science		
7	Data Science	57
	Benjamin M. Abdel-Karim	
7.1	Einordnung Data Science	57
7.2	Data-Science-Prozess	60
7.3	Data-Science-Projekte für dieses Buch	61
8	Data Science und Maschinelles Lernen	63
	Benjamin M. Abdel-Karim	
8.1	Definitionen des maschinellen Lernens	63
8.2	Herausforderungen des Maschinellen Lernens im Kontext von Data Science	66
Teil III Produktanalyse		
9	Anwendungsbeispiel: Meine besten Videospiele	73
	Benjamin M. Abdel-Karim	
9.1	Videospiel: Datensatz und Fragestellung	74
9.2	Videospiel: Preprocessing	75
9.3	Videospiel: Explore the Data	80
9.4	Videospiel: Model the Data (Regression)	88
9.4.1	Theoretische Modellgrundlagen	88
9.4.2	Implementierung des Modells	90
9.5	Videospiel: Interpretation der Ergebnisse	92

10 Anwendungsbeispiel: Conjoint-Analyse – Mehr als die Summe seiner Teile	93
Katharina Keller	
10.1 Conjoint-Analyse: Einführung in die Methodik	94
10.2 Conjoint-Analyse: Datensatz und Fragestellung	96
10.3 Conjoint-Analyse: Preprocessing	98
10.4 Conjoint-Analyse: Explore the Data	101
10.5 Conjoint-Analyse: Model the Data	103
10.6 Conjoint-Analyse: Interpretation der Ergebnisse	106
 Teil IV Kunden- und soziale Medienanalyse	
11 Anwendungsbeispiel: Game of Social Networks	113
Benjamin M. Abdel-Karim	
11.1 Soziales Netzwerk: Datensatz und Fragestellung	114
11.2 Soziales Netzwerk: Pre-processing	115
11.3 Soziales Netzwerk: Explore the Data	117
11.4 Soziales Netzwerk: Model the Data (Logistische Regression)	126
11.4.1 Theoretische Grundlage des Modells	126
11.4.2 Implementierung des Modells	128
11.5 Soziales Netzwerk: Interpretation der Ergebnisse	129
 12 Erhebung und Auswertung von Social-Media-Daten	131
Hendrik Jöntgen	
12.1 Social Media: Datensatz und Fragestellung	132
12.2 Social Media: Pre-processing	135
12.3 Social Media: Explore the Data	138
12.4 Social Media: Model the Data	144
12.5 Social Media: Interpretation der Ergebnisse	148
 13 Anwendungsbeispiel: Cloud Web Services	151
Daniel Franzmann	
13.1 Cloud Web Services: Fragestellung	152
13.2 Cloud Web Services: Pre-processing	152
13.3 Cloud Web Services: Explore the Data and More	161
13.4 Cloud Web Services: Zusammenfassung	169

Teil V Mitarbeiteranalyse

14 Anwendungsbeispiel: Mitarbeiterabwanderung	173
Kevin Bauer	
14.1 Mitarbeiterabwanderung: Vorbereitung und Definition des Problems. . .	174
14.2 Mitarbeiterabwanderung: Auf-/Vorbereitung der Daten	177
14.2.1 Mitarbeiterabwanderung: Sampling und Erzeugung von Trainings- und Testdaten	192
14.2.2 Explorative Analysen auf Trainingsdaten	198
14.2.3 Auswahl und Training von ML Modellen	203
15 Anwendungsbeispiel: Get Your Things Done – Modernes Zeitmanagement	225
Benjamin M. Abdel-Karim	
15.1 Zeitmanagement: Datensatz und Fragestellung	225
15.2 Zeitmanagement: Pre-processing	228
15.3 Zeitmanagement: Explore the Data	232
15.4 Zeitmanagement: Model the data.	240
15.4.1 Theoretische Modellgrundlagen	244
15.4.2 Implementierung des Modells	247
15.5 Zeitmanagement: Interpretation der Ergebnisse	248

Teil VI Finanzanalyse

16 Anwendungsbeispiel: Portfolioanalyse	253
Marcel Zeuch	
16.1 Portfolioanalyse: Datensatz und Fragestellung	254
16.2 Bereitstellung einer Datenbank	255
16.3 Portfolioanalyse: Pre-processing	258
16.4 Portfolioanalyse: Explore the Data	262
16.5 Portfolioanalyse: Model the Data	266
16.5.1 Datenaggregation und Speicherung in der Datenbank.	267
16.5.2 Verarbeitung der Daten und Implementierung der Benutzeroberfläche.	273
16.6 Portfolioanalyse: Interpretation der Ergebnisse.	293
Thematisches Lexikon	295
Literatur	301
Stichwortverzeichnis	307

Abbildungsverzeichnis

Abb. 2.1	Das Python-Installationsfenster	9
Abb. 2.2	Willkommenfenster PyCharm	10
Abb. 2.3	Projektkonfiguration unter PyCharm	11
Abb. 2.4	Das erste Skript im Projektordner	12
Abb. 2.5	Die IDE und Hello World.	13
Abb. 3.1	Ausgehend von den übergeordneten Klassen der primitiven Datentypen zeigt die Abbildung eine Auswahl der wichtigsten Datentypen, die in Python zum Einsatz kommen können	18
Abb. 4.1	Ausgehend von der übergeordneten Klasse der Datenstruktur wird eine Auswahl der wichtigsten Datenstrukturen gezeigt, die in Python zum Einsatz kommen können	26
Abb. 5.1	Die Kontrollstrukturen in der Übersicht. Die Abbildung visualisiert ausgewählte Kontrollstrukturen	36
Abb. 5.2	Die If-else-Bedingung prüft, ob eine Bedingung zutrifft. Sofern die deklarierte Bedingung erfüllt ist, wird der nächste Programmabschnitt ausgeführt. Trifft die Bedingung nicht zu, wird ein anderer Programmabschnitt ausgeführt	37
Abb. 5.3	Die verschachtelten If-else-Bedingungen. Zunächst erfolgt eine If-Bedingungsprüfung. Ist diese Bedingung erfüllt, erfolgt die Ausführung des nächsten Programmabschnitts. Ist diese Bedingung nicht erfüllt, wird im Else-Zweig eine weitere If-Bedingungsprüfung ausgeführt	40
Abb. 5.4	Die For-Schleife. Der Schleifenkopf legt durch die Deklaration der Bedingung die Lauflänge fest. Solange die Bedingung Gültigkeit besitzt, wird der Quellcode im Schleifenrumpf ausgeführt. Sobald die maximale Lauflänge erreicht und damit die Bedingung nicht mehr erfüllt ist, wird die Schleife ihren Dienst einstellen und der nächste Quellcode außerhalb der For-Schleife ausgeführt	42

Abb. 5.5	Die While-Schleife. Der Schleifenkopf der While-Schleife enthält eine Bedingung. Solange diese erfüllt ist, wird die Schleife ihren Betrieb fortsetzen. Diese Bedingung wird bei jedem Schleifendurchlauf geprüft. Erst wenn die Bedingung ihre Gültigkeit verloren hat, stellt die Schleife ihren Betrieb ein, was dazu führt, dass der nächste Codeabschnitt außerhalb der Schleife ausgeführt wird.	44
Abb. 5.6	Die Try-except-Kontrollstruktur. Zunächst wird Python versuchen, den Try-Block auszuführen. Dieser umfasst das Codesegment, das ausgeführt werden soll. Der Except-Block greift dann, wenn der Try-Block zu einem Ausnahmefall führt und der Code eigentlich abstürzen würde.	46
Abb. 7.1	Data Science – ein Versuch der Einordnung	59
Abb. 7.2	Data-Science-Prozess	61
Abb. 8.1	Definitionsbaum	64
Abb. 8.2	Maschinelles Lernen – Taxonomie.	65
Abb. 9.1	Der Variableninspektor in PyCharm zur Visualisierung der Wertebelegung eines DataFrame	78
Abb. 9.2	Der Pairplot für die Videospiele	83
Abb. 9.3	Der Barplot für die Videospieleinahmen im Zeitverlauf	85
Abb. 9.4	Der horizontale Barplot für die Top-Ten-Videospiel-Publisher gemessen an den Einnahmen	87
Abb. 10.1	Beispielhaftes Choice Set.	98
Abb. 10.2	Plot Bedeutungsgewichte	109
Abb. 11.1	Das soziale Game-of-Thrones-Netzwerk in einer ersten Übersicht.	119
Abb. 11.2	Game-of-Thrones-Netzwerkstruktur auf Basis des Degree	126
Abb. 12.1	Zeitverlauf der Tweets pro Stunde	139
Abb. 12.2	Zeitverlauf der Tweets pro Minute	140
Abb. 12.3	Zeitverlauf des Sentiment.	141
Abb. 12.4	Zeitverlauf des Sentiment mit Einbettung der Timestamps	142
Abb. 12.5	Verteilung der Tweets auf die unterschiedlichen Videospiele	144
Abb. 12.6	Sentiment-Boxplots der einzelnen Videospiele	145
Abb. 12.7	Grafische Überprüfung der Normalverteilung des Sentiment	147
Abb. 13.1	Der FileZilla Server Manager in der Übersicht	159
Abb. 13.2	Erfolgreicher Verbindungsaufbau zur EC2-Instanz	160
Abb. 13.3	Fake- (rot) vs. Non-fake-Kommentar-Sentiments (blau)	164
Abb. 13.4	Non-fake vs. fake word clouds	166
Abb. 13.5	Das neu erstellte S3-Bucket	167
Abb. 13.6	Erfolgreiche Visualisierung auf dem S3-Bucket	169
Abb. 14.1	Der Pairplot des Trainsets.	201

Abb. 14.2	(a) Der Relplot auf Basis der Daten. (b) Rel plot: Label = 0. (c) Rel plot: Label = 1. (d) Catplot der Variante ‚bar‘. (e) Catplot der Variante ‚box‘.	203
Abb. 14.3	(a) Konfusionsmatrix Trainingsdaten. (b) Konfusionsmatrix Testdaten.	208
Abb. 14.4	Optimaler Hyperparameter des KNN Modells.	212
Abb. 14.5	(a) Konfusionsmatrix Trainingsdaten. (b) Konfusionsmatrix Testdaten.	213
Abb. 14.6	(a) Konfusionsmatrix Trainingsdaten. (b) Konfusionsmatrix Testdaten.	216
Abb. 14.7	(a) Konfusionsmatrix Trainingsdaten. (b) Konfusionsmatrix Trainingsdaten	221
Abb. 14.8	Feature importances	222
Abb. 15.1	Grundfunktionalität des Time Tracker Wearable	226
Abb. 15.2	Zeitliche Verteilung der Aktivitäten in Form von Boxplots	235
Abb. 15.3	Zeitliche Verteilung der Aktivitäten in Form eines Ringdiagramms	238
Abb. 15.4	Gegenüberstellung der Daten aus der Datentransformation von Minuten zu Stunden.	243
Abb. 15.5	Der Informationsverarbeitungsprozess eines Neurons.	246
Abb. 16.1	Implementierung des vorgestellten Portfolioanalysetools	255
Abb. 16.2	Darstellung der zu implementierenden Benutzeroberfläche	274
Abb. 16.3	Darstellung der unterschiedlichen HTML-Komponenten der zu implementierenden Benutzeroberfläche.	286

Tabellenverzeichnis

Tab. 3.1	Übersicht nützlicher arithmetischer Operatoren mit Integer. $iX =$ Integer-Variable; $iY =$ weitere Integer-Variable.	20
Tab. 3.2	Übersicht nützlicher arithmetischer Operatoren mit Float. $dX =$ Float-Variable; $dY =$ weitere Float-Variable	20
Tab. 3.3	Übersicht der Vergleichsoperatoren.	23
Tab. 6.1	Auswahl einiger Built-in-Funktionen in Python.	50
Tab. 9.1	Übersicht nützlicher Funktionen des pandas-Moduls für die Videospielanalyse im ersten Pre-Processing	80
Tab. 9.2	Übersicht nützlicher Funktionen für die Datenexploration.	89
Tab. 9.3	Übersicht nützlicher Funktionen für die Modellkonzeption.	91
Tab. 10.1	Übersicht Attribute und Attributlevel	97
Tab. 11.1	Top-Ten-Charaktere in „Game of Thrones“ basierend auf ihrem Degree, Betweenness Centrality und der Degree Centrality. Betweenness steht für Betweenness Centrality und Centrality steht für Degree Centrality	124
Tab. 11.2	Übersicht nützlicher Funktionen für die Datenexploration.	127
Tab. 11.3	Übersicht nützlicher Funktionen für die Modellkonzeption.	130
Tab. 15.1	Abrechenbare Stunden in Euro für das Arbeitsjahr.	239
Tab. 16.1	Auszug der Eingabeparameter des Musterportfolios	255

Listings

Listing 2.1	Professionelle Quellcodedokumentation	15
Listing 3.1	Deklaration eines	21
Listing 3.2	Operationen mit Boolean in Python	22
Listing 3.3	Verknüpfungsoperatoren mit Logicals in Python	22
Listing 4.1	Eine Liste in Python anlegen	26
Listing 4.2	Ein Element in eine Liste aufnehmen	27
Listing 4.3	Zugriff auf das erste Listenelement	27
Listing 4.4	Liste in Liste	28
Listing 4.5	Liste als Stapelspeicher	29
Listing 4.6	Liste als Warteschlange	30
Listing 4.7	Dictionary	31
Listing 4.8	Mengen	32
Listing 5.1	If-Else in Python	37
Listing 5.2	If-Else mit Konjunktion in Python.	38
Listing 5.3	If-Else mit oder Operation in Python.	39
Listing 5.4	Verschachtelte If-else-Bedingung mit Oder-Operation in Python.	39
Listing 5.5	For-Schleife in Python	41
Listing 5.6	For-Schleifen Ergebnis	42
Listing 5.7	For-Schleife in Python	43
Listing 5.8	For-Schleifen-Ergebnis für Listeniteration	43
Listing 5.9	While-Schleife in Python.	44
Listing 5.10	While-Schleifen Ergebnis	45
Listing 5.11	Try-exception-Kontrollstruktur in Python	46
Listing 5.12	Try-Exception mit genauer Exception Spezifikation in Python	47
Listing 6.1	Eine eigene Funktion in Python.	52
Listing 6.2	Nutzung des random-Moduls in Python	54
Listing 9.1	Import des Moduls Pandas.	76
Listing 9.2	Import einer .CSV-Datei als pandas DataFrame	77
Listing 9.3	Aufruf der df.head()-Funktion	77

Listing 9.4	Konsolenausgabe: <code>df.head()</code>	77
Listing 9.5	Aufruf der <code>df.info()</code> -Funktion	78
Listing 9.6	Konsolenausgabe: <code>df.info()</code>	79
Listing 9.7	Aufruf der Funktion <code>df.describe()</code>	80
Listing 9.8	Konsolenausgabe: <code>df.describe()</code>	80
Listing 9.9	Erstellung eines Pairplot für die Videospiele	82
Listing 9.10	Erstellung eines Barplot für die Videospiele im zeitlichen Verlauf	83
Listing 9.11	Erstellung eines Barplot für die Top-Ten-Publisher	86
Listing 9.12	Top-Ten-Videospiele	87
Listing 9.13	Konsolenausgabe: Top-Ten-Videospiele	88
Listing 9.14	Konsolenausgabe: Lineare Regression	91
Listing 10.1	Import der benötigten Module	98
Listing 10.2	Einlesen der Daten	99
Listing 10.3	Zusammenführen der Daten	99
Listing 10.4	Vorbereitung der Daten	100
Listing 10.5	Endogene und exogene Variable trennen	101
Listing 10.6	Dummycodierung	101
Listing 10.7	Effektcodierung	102
Listing 10.8	Schätzung der Parameterwerte	104
Listing 10.9	Speichern der Ergebnisse	104
Listing 10.10	Iteration	105
Listing 10.11	Berechnung der fehlenden Werte	105
Listing 10.12	Berechnung der Spannweiten	106
Listing 10.13	Berechnung der Bedeutungsgewichte	107
Listing 10.14	Konsolenausgabe: Die Bedeutungsgewichte	107
Listing 10.15	Plotten der Ergebnisse	107
Listing 11.1	Import der Module für die Soziale Netzwerk Analyse	115
Listing 11.2	Import der Daten für die Soziale Netzwerk Analyse	115
Listing 11.3	Konsolenausgabe: <code>len(df.columns)</code>	116
Listing 11.4	Konsolenausgabe: <code>df.columns.to_list()</code>	116
Listing 11.5	Konsolenausgabe: <code>df.info()</code>	116
Listing 11.6	Konsolenausgabe: <code>df.isnull().sum()</code>	117
Listing 11.7	Nutzung von <code>df.describe()</code>	117
Listing 11.8	Konsolenausgabe: <code>df.describe()</code>	118
Listing 11.9	Nutzung von <code>nx.from_pandas_edgelist</code>	118
Listing 11.10	Abbildung eines Graphen durch die eigene Funktion <code>fGeneratePlot</code>	119
Listing 11.11	Durchführung einer ersten Sozialen Netzwerk Analyse	122
Listing 11.12	Verteilungswahrscheinlichkeit der Degrees auf Basis des Logarithmus	124
Listing 11.13	Logistische Regressionsanalyse	128

Listing 11.14	Konsolenausgabe: Logistische Regression	129
Listing 12.1	Initialisierung der Twitter API	133
Listing 12.2	Abfrage der Twitter-Query	134
Listing 12.3	Umwandelung der Query-Ergebnisse in ein DataFrame	134
Listing 12.4	Speicherung der Twitter-Daten in eine MongoDB	135
Listing 12.5	Laden der Twitter-Daten von einer MongoDB	135
Listing 12.6	Aufbereitung der Tweet-Zeitdaten	135
Listing 12.7	Aufbereitung der Tweet-Zeitdaten	136
Listing 12.8	Definition von Regex	136
Listing 12.9	Filtern der Tweet-Texte	137
Listing 12.10	Berechnung der Tweet-Sentiments	138
Listing 12.11	Zeitverlauf der Tweets pro Stunde	138
Listing 12.12	Zeitverlauf der Tweets pro Minute	139
Listing 12.13	Zeitverlauf des Sentiment	140
Listing 12.14	Einbettung der Timestamps	141
Listing 12.15	Zuteilung der Tweets	142
Listing 12.16	Erstellung eines Bar-Plots	143
Listing 12.17	Erstellung von Boxplots für jedes Videospiel	145
Listing 12.18	Durchführung einer ANOVA zum Testen auf Unterschieden im Sentiment zwischen Videospielen	146
Listing 12.19	Shapiro-Wilk-Test zum Überprüfen der Normalverteilung des Sentiment für jedes Videospiel	146
Listing 12.20	Plot der Residuen	146
Listing 12.21	Durchführung eines Kruskal-Willis Tests auf Unterschiede im Sentiment zwischen Videospielen	148
Listing 13.1	Benötigte Libraries und Initialisierung des API-Schlüssels	156
Listing 13.2	Suche nach dem Begriff „corona“ und Speichern der Video-Links	156
Listing 13.3	Initialisierung der Ergebnislisten	157
Listing 13.4	YouTube-Kommentar Web Crawler	157
Listing 13.5	Speichern der Ergebnisse	158
Listing 13.6	Benötigte Libraries	162
Listing 13.7	Import der CSV-Datei und Filtern nach englischen Kommentaren	162
Listing 13.8	Generierung der Sentiments und Transformation des Datum-/Zeit-Formats	163
Listing 13.9	Aufteilung in zwei Gruppen und Aggregation der Durchschnitts-Sentiment-Werte	163
Listing 13.10	Visualisierung der Resultate	164
Listing 13.11	Verknüpfung der täglichen Kommentare durch Gruppierung nach Datum	165
Listing 13.12	Visualisierung der beiden Gruppen in täglichen Word Clouds	165

Listing 13.13	index.html	168
Listing 14.1	Import pandas as pd	174
Listing 14.2	Import numpy as np	175
Listing 14.3	Import matplotlib	175
Listing 14.4	Import seaborn	175
Listing 14.5	Import Sklearn	175
Listing 14.6	Import der Daten	178
Listing 14.7	DataFrame ansicht	178
Listing 14.8	Konsolenausgabe	179
Listing 14.9	DataFrame dtypes	179
Listing 14.10	Konsolenausgabe	179
Listing 14.11	DataFrame columns	179
Listing 14.12	Konsolenausgabe: Datentypen im DataFrame	180
Listing 14.13	Teil des DataFrame	180
Listing 14.14	Teil des DataFrame	180
Listing 14.15	Slice-Indexierer	181
Listing 14.16	Slice-Indexierer mit Bedingung	181
Listing 14.17	Konsolenausgabe	182
Listing 14.18	DataFrame weitere Datenmanipulationen	182
Listing 14.19	DataFrame auf null nan Werte überprüfen	183
Listing 14.20	Konsolenausgabe	183
Listing 14.21	Optionen zur Handhabung fehlender Werte	184
Listing 14.22	Verwendung von value_counts	185
Listing 14.23	Konsolenausgabe	185
Listing 14.24	Verwendung des one-hot-encoders	186
Listing 14.25	Schleife für Dummy Variablen	186
Listing 14.26	Matrix to DataFrame	187
Listing 14.27	Konsolenausgabe	187
Listing 14.28	Matrix to DataFrame	187
Listing 14.29	Implementierung einer Methode zum automatisierten One-Hot-Encoding	188
Listing 14.30	Reskalierung von Variablen	190
Listing 14.31	Implementierung einer Funktion zur Reskalierung von Variablen	191
Listing 14.32	Verwendung der Funktion sample_without_replacement	193
Listing 14.33	Identifikation von Imbalances	194
Listing 14.34	Konsolenausgabe	194
Listing 14.35	Verwendung der Funktion Undersampling	194
Listing 14.36	Implementierung einer Methode zum Ausbalancieren von Daten	195
Listing 14.37	Konsolenausgabe	197
Listing 14.38	Verwendung der Funktion train_test_split	197

Listing 14.39	Konkatenieren der Features und Labels und Korrelationsberechnung	199
Listing 14.40	Konsolenausgabe	199
Listing 14.41	Pairplot der ausgewählten Variablen	200
Listing 14.42	Weitere Abbildungen	202
Listing 14.43	Partition der Daten in Trainings- und Testdaten und Training des Modells	205
Listing 14.44	Implementierung einer Methode zur Überprüfung der Modell Performance	206
Listing 14.45	Konsolenausgabe	208
Listing 14.46	Implementierung der Hyperparameter Optimierung des KNN Modells	210
Listing 14.47	Konsolenausgabe	212
Listing 14.48	Implementierung des optimierten KNN Modells	213
Listing 14.49	Konsolenausgabe	213
Listing 14.50	Abpeichern und Laden eines trainierten Modells	214
Listing 14.51	Implementierung eines naiven Random Forest Modells	217
Listing 14.52	Konsolenausgabe	217
Listing 14.53	Optimierung der Hyperparameter des Random Forests	218
Listing 14.54	Konsolenausgabe	220
Listing 14.55	Implementierung des optimierten Modells	220
Listing 14.56	Konsolenausgabe	220
Listing 14.57	Implementierung der Ausgabe der Feature Importances	223
Listing 15.1	Datenimport	228
Listing 15.2	Konsolenausgabe: df.columns.tolist() und df.shape()	229
Listing 15.3	Spalten entfernen aus einem DataFrame	230
Listing 15.4	Konsolenausgabe: print(df.info())	230
Listing 15.5	Von Strings zu DateTime Objekten	231
Listing 15.6	Nutzung von unique()	232
Listing 15.7	Konsolenausgabe: LActivity aus unique()	233
Listing 15.8	Nutzung von describe()	233
Listing 15.9	Konsolenausgabe: Describe	233
Listing 15.10	Boxplot für die Aktivitäten entsprechend ihres zeitlichen Anspruchs	234
Listing 15.11	Finde Aufgabe im Date Frame mit loc	234
Listing 15.12	Verwendung von Groupby für die Aktivitäten	236
Listing 15.13	Anteil der Aktivitäten an der genutzten Arbeitszeit	236
Listing 15.14	Werteüberprüfung in DataFrame zur Klassifikation von Werten mit loc	237
Listing 15.15	Berechnung der abrechenbaren Stunden	239
Listing 15.16	Datentransformation von Minuten zu Stunden	242
Listing 15.17	Umsetzung des One Hot Encoding	243

Listing 15.18	Umsetzung des One Hot Encoding	244
Listing 15.19	Implementierung eines Perceptron-Netzwerks	247
Listing 16.1	Anlegen des Schemas data_science in MySQL.	257
Listing 16.2	Anlegen der Tabelle statics in MySQL	257
Listing 16.3	Anlegen der Tabelle portfolio_transactions in MySQL.	257
Listing 16.4	Anlegen der Tabelle marketdata_daily in MySQL	257
Listing 16.5	Anlegen der Tabelle marketdata_fx_daily in MySQL.	257
Listing 16.6	Upload einer .csv Datei in eine Tabelle der Datenbank.	258
Listing 16.7	Umsetzung der Anfrage des Tickers zu einer spezifizierten ISIN	260
Listing 16.8	Konsolenausgabe: print(out) zur Funktion SYMBOL_SEARCH.	260
Listing 16.9	Umsetzung der Datenspeicherung aus dem Dictionary.	261
Listing 16.10	Umsetzung der Anfrage von Marktdaten zu einem gegebenen Ticker.	261
Listing 16.11	Speicherung der Daten in die Datenbank	261
Listing 16.12	Konsolenausgabe: print(out) zur Funktion TIME_SERIES_DAILY	262
Listing 16.13	Überführung der bereinigten Marktdaten in einen DataFrame	263
Listing 16.14	Konsolenausgabe: print(df) der Marktdaten als DataFrame	263
Listing 16.15	Erweiterung des DataFrame df	263
Listing 16.16	Upload der Marktdaten in die Datenbank über eine .csv Datei	264
Listing 16.17	Eingabeparameter zur Ansprache der Funktion FX_DAILY.	264
Listing 16.18	Eingabeparameter zur Ansprache der Funktion OVERVIEW.	265
Listing 16.19	Abfrage an die Datenbank und Speicherung in einem DataFrame.	266
Listing 16.20	Abfrage aller Security ID, zu denen noch keine Statics angelegt wurden.	268
Listing 16.21	Anfrage fehlender Ticker und Speicherung in der Datenbank	268
Listing 16.22	Anfrage der gesamten historischen Marktdaten und Upload in die Datenbank	269
Listing 16.23	Abfrage an die Datenbank zur Bestimmung fehlender Marktdaten.	271
Listing 16.24	Verarbeitung der angefragten Marktdaten und Upload in die Datenbank	272
Listing 16.25	Abfrage und Speicherung der ersten Zeile der Tabelle marketdata_fx_daily	273
Listing 16.26	Initialisierung notwendiger Bibliotheken und Komponenten	273
Listing 16.27	Darstellung des aktuellen Portfolios in MySQL	275

Listing 16.28	Wert des gesamten Portfolios zum letzten Handelstag	275
Listing 16.29	Darstellung der Portfolioallokation nach Land	276
Listing 16.30	Zusammenfassen übriger Länder mit dem entsprechenden Volumen	277
Listing 16.31	Bereitstellung des kumulierten Nominals in einem DataFrame.	277
Listing 16.32	Ausgabe aller jemals gehandelten Wertpapiere	278
Listing 16.33	Definition der Listen var und initial_date	278
Listing 16.34	Initialisierung der Berechnung der historischen Portfolioentwicklung	279
Listing 16.35	Initialisierung der Berechnung des kumulierten Portfoliowerts zu jedem Handelstag	280
Listing 16.36	Speicherung des kumulierten Handelspreises	280
Listing 16.37	Bereitstellung der Marktdaten der zu iterierenden Security ID ab dem ersten Handelstag.	280
Listing 16.38	Initialisierung und Berechnung des Portfoliowerts, vom ersten bis zum nächsten Handelstag	281
Listing 16.39	Initialisierung und Berechnung des letzten Depotwerts, vom ersten bis zum nächsten Handelstag	281
Listing 16.40	Berechnung des kumulierten Handelspreises im Fall eines Zukaufs des Wertpapiers	282
Listing 16.41	Berechnung des Portfoliowerts und des letzten Depotwerts im Fall des letzten Handelstags des Wertpapiers.	282
Listing 16.42	Speicherung des DataFrame df_portfolio	283
Listing 16.43	Initialisierung und Speicherung der berechneten Portfolio- und letzten Depotwerte	283
Listing 16.44	Datengruppierung im DataFrame df_dash.	284
Listing 16.45	Vorbereitung der Daten zur grafischen Darstellung innerhalb der App	284
Listing 16.46	App in dash – Header	286
Listing 16.47	App in dash – Grafik Portfolioentwicklung.	287
Listing 16.48	App in dash – Tabelle Portfolio	287
Listing 16.49	App in dash - Portfolio nach Land/Industrie	288
Listing 16.50	Initialisierung des app.callback	289
Listing 16.51	Darstellung des Gesamtportfolios innerhalb des app.callback	290
Listing 16.52	Initialisierung der Darstellung einzelner, selektierter Wertpapiere innerhalb des app.callback	291
Listing 16.53	Konsolenausgabe: print(df_portfolio.loc[df_portfolio ['security_id'] == i, 'Ticker'])	292
Listing 16.54	Fortsetzung der Darstellung einzelner, ausgewählter Wertpapiere innerhalb des app.callback	292

Teil I

Grundlagen der Programmierung



Einleitung

1

Benjamin M. Abdel-Karim

Dieses Kapitel gibt einen Überblick über den Inhalt, den Aufbau und die Zielsetzung des Buchs. Mit dem wachsenden Angebot an Datenquellen wächst das Interesse, diese entsprechend zu monetarisieren. Dies gilt nicht nur für digitale Geschäftsprozesse, sondern auch für die Forschung und Entwicklung. Damit sind das Verständnis und die Fähigkeit zur Datenanalyse gewinnbringende Vermögenswerte, um die gemeinschaftliche Wohlfahrt des Staats voranzubringen. Im ersten Teil des Kapitels wird der Gegenstand dieses Buchs beschrieben. Daran knüpft der Aufbau des Buchs an und es wird eine Erklärung geboten, weshalb sich Python besonders für Einsteiger eignet.

1.1 Gegenstandsbereich dieses Buchs

Die stets zunehmende Menge an Daten und die implizit entstehende Komplexität dieser Daten führen dazu, dass neue Geschäftsbereiche und Stellenbeschreibungen entstehen, sodass die Fähigkeit, diese Daten zu verarbeiten und geeignet zu analysieren, immer gefragter wird. Das Anwendungsfeld der Datenanalyse war bis vor wenigen Jahren nur im Bereich Forschung und Entwicklung zu finden. Allerdings entdecken zunehmend Start-ups, der Mittelstand und die großen Technologiekonzerne diesen Bereich für sich, um neue Geschäftsmodelle zu erschließen. Vor dem Hintergrund der Wettbewerbssicherung führt dieser Umstand dazu, dass Studierende, Berufsanfänger und Unternehmer sich frühzeitig mit den Grundlagen der Datenanalyse befassen sollten. Damit richtet sich dieses Buch an eben jene interessierten Leser, die sich mit den Grundlagen der Datenanalyse befassen

B. M. Abdel-Karim (✉)
Frankfurt am Main, Hessen, Deutschland
E-mail: BenjaminM.Abel-Karim@gmx.de

möchten, insbesondere an Leser, die sich frühzeitig den neuen Wettbewerbsanforderungen stellen wollen.

Dieses Werk ist nicht als reines Einführungswerk zu verstehen, sondern als Nachschlagewerk zum Lösen der klassischen Problemstellungen in der Datenanalyse, die in der Regel zu Beginn der Datenanalyse auftreten. Basierend auf den Erkenntnissen zahlreicher Vorlesungen, Übungen, Mentorings und Speaker Events fasst dieses Buch einige Quellcodeauschnitte in strukturierter Art und Weise zusammen. Das Buch beginnt mit den essenziellen Grundlagen der Programmierung in Python. Diese werden durch praktische Implementierungen von Anwendungsbeispielen vertieft.

1.2 Aufbau und Zielsetzung

Als Grundlagen- und Nachschlagewerk hat dieses Buch den primären Anspruch, die Umsetzung von wesentlichen Methoden im Umgang der Datenanalyse im Kontext der Programmiersprache Python darzulegen. Der erste Teil des Buchs vermittelt die essenziellen Grundkenntnisse, um die Data-Science-Projekte im zweiten Teil des Buchs nachvollziehen zu können. Die Besonderheit in diesem Buch sind eben jene Projekte (Case Studies). Jede Case Study behandelt eine Data-Science-Herausforderung aus der Perspektive einer meiner Co-Autoren, sodass Sie als Leser in diesem Buch die Gelegenheit erhalten werden, unterschiedliche Programmierstile und -strategien kennenzulernen. Daraus ergibt sich der folgende Aufbau dieses Buchs:

- Grundlagen
 - Python Installation und Einstieg
 - Grundlegende Datentypen
 - Datenstrukturen und Indexierung
 - Kontrollstrukturen
- Data-Science-Projekte
 - Data Science und Data-Science-Prozess
 - Überblick bedeutender Bibliotheken
 - Datenvorverarbeitung
 - Datenvisualisierungen
 - Entwicklung eigener Modelle

Die Zielsetzung des Buchs ist damit, die gesammelten Erkenntnisse und Erfahrungen zu archivieren und weiterzugeben, damit der Leser aus vorangegangenen Erfahrungen der Autoren und dem entstandenen Lernprozess profitieren kann. Im Allgemeinen richtet sich das Buch mit dieser Zielsetzung an jeden interessierten und motivierten Leser, sich die Grundlagen des Programmierens in Python und von Data Science anzueignen. Außerdem ist das Buch für Studierende im Bachelor- und Masterstudium konzipiert, die ihren

Wissenshorizont erweitern möchten und durch die erworbenen Fähigkeiten imstande sind, die klassischen Aufgaben des wissenschaftlichen Arbeitens im Rahmen der Datenanalyse zu bewerkstelligen. Darüber hinaus ist das Buch für Berufspraktiker geschrieben, die im Rahmen ihrer Berufstätigkeit eine flexible und moderne Programmiersprache nutzen wollen, um die anfallenden Daten zu analysieren. In vielen Firmen werden bisher einfache Tabellenverarbeitungsprogramme eingesetzt, die allerdings schnell an ihre Verarbeitungskapazitäten stoßen. Hierzu können die vorgestellten Anwendungsfälle einen Eindruck zu den möglichen Alternativen mithilfe der Programmiersprache Python aufzeigen. Zusätzlich richtet sich dieses Buch an junge Unternehmer, die den Wert ihrer Daten erkannt haben und nun auf der Suche nach einer Inspiration sind, diese Daten geeignet auszuwerten.

Demnach möchte dieses Buch einen Beitrag zur Wissensdiffusion moderner Technologien liefern, wie beispielsweise Hardware und Programmiersprachen, und durch anschauliche und praxisnahe Beispiele zum selbstständigen Ausprobieren anregen. Vor dem Hintergrund des exponentiellen Wissenszuwachses der menschlichen Gesellschaft und der damit verbunden Entwicklung einer hochspezialisierten Wissensgesellschaft (Stehr, 2006; Bittlingmayer & Bauer, 2006), ist die private Weiterbildung ein zentraler Bestandteil der fachlichen Weiterentwicklung.

1.3 Warum Python?

In der Welt der Programmierung existieren zahlreiche Programmiersprachen. Jede einzelne von ihnen besitzt Stärken und Schwächen. Für Einsteiger bietet sich besonders die Programmiersprache Python an. An dieser Stelle möchte ich kurz auf einige gute Gründe zur Nutzung von Python eingehen, ohne den Details aus den kommenden Kapiteln vorzugreifen:

- Python ist schnell zu erlernen, da diese Programmiersprache im Vergleich zu anderen Programmiersprachen auf wenige Syntaxsymbole setzt.
- Zudem werden Variablen dynamisch deklariert, sodass das Schreiben des Codes zu Beginn vereinfacht wird.
- Python wird im Bereich Forschung und Entwicklung sowie in vielen Praxiskontexten eingesetzt, sodass ihr eine hohe praktische Relevanz beizumessen ist.
- Python wird von einem großen Nutzerkreis durch umfassende Module (Bibliotheken) unterstützt, sodass viele Programmcode Teile schon verwendet werden können. Außerdem existieren zahlreiche Onlinekurse und Literaturbeiträge.
- Python lässt sich mit anderen Programmiersprachen wie C++ oder HTML verknüpfen, sodass komplexere Programme geschrieben werden können.

Die folgenden Kapitel gehen auf die Grundlagen der Programmiersprache ein.



Python: Installation und Einstieg

2

Benjamin M. Abdel-Karim

Dieses Kapitel bildet den Grundstein für die Verwendung von Python, um im späteren Verlauf des Buchs mit der Datenanalyse zu beginnen. Zuerst wird Python als Programmiersprache vorgestellt (Abschn. 2.1). Anschließend wird die Installation in einzelnen Schritten beschrieben (Abschn. 2.2). Darauf aufbauend wird eine passende Entwicklungsumgebung vorgestellt (Abschn. 2.3) und das Bedienkonzept dargestellt, um die ersten Programme schreiben zu können.

2.1 Python

Python ist eine verhältnismäßig junge Programmiersprache. Sie erschien 1991 und wurde von Guido van Rossum entwickelt. Der Name Python ist eine Anspielung auf die englischsprachige Komikergruppe Monty Python. Seit der ersten Version von Python wurde diese Programmiersprache fortlaufend weiterentwickelt. Inzwischen ist Python in der Version 3.8+ (stand 12. Februar 2022) verfügbar und ist imstande, verschiedene Paradigmen der Informatik zu unterstützen, wie z. B. die funktionale oder objektorientierte Programmierung.

Pythons Erfolg kann auf zwei grundlegende Aspekte zurückgeführt werden. Erstens zeichnet sich diese Programmiersprache durch ihre Simplifizierung aus. Der geschriebene Quellcode verzichtet auf zusätzliche Klammerung und Zeichen, was bei vielen anderen Programmiersprachen wie z. B. Java der Fall ist; die Strukturierung des Quellcodes findet über die Einrückung statt, was den Quellcode für Menschen gut lesbar macht. Also: Python ist einfach zu lesen und zu schreiben im Vergleich zu anderen Programmiersprachen. Ein Beispiel ist, dass Python auf die Deklaration durch den Nutzer verzichtet. Das bedeutet,

B. M. Abdel-Karim (✉)
Frankfurt am Main, Hessen, Deutschland
E-mail: BenjaminM.Abdel-Karim@gmx.de

dass sich der Programmierer durch die dynamische Typisierung, im Gegensatz zu anderen Programmiersprachen, keine Gedanken über initiale Datentypen machen muss. Der zweite Aspekt ist, dass Python kostenlos ist. Dieser Umstand hat vermutlich dazu beigetragen, dass Python in der Welt der Programmierung weit verbreitet ist und sich einer großen Community erfreuen kann, die durch das Anfertigen zahlreicher zusätzlicher und ebenfalls kostenfreier Bibliotheken zur Verbreitung von Python beiträgt.

2.2 Installation

Die Installation von Python gelingt für die geläufigen Betriebssysteme relativ einfach, da Python über eine Installationsdatei verfügt, die über die Webseite <https://www.python.org/downloads/> heruntergeladen werden kann. Dabei steht die Installationsdatei für die Betriebssysteme Windows, Linux und Mac OS zur Verfügung. Für Einsteiger empfiehlt sich der Download der aktuellsten Version. Die Version zum Zeitpunkt der Bucherstellung ist die Version 3.8.2. (stand 12. Februar 2022). Allerdings sind die meisten Quellcodebeispiele dieses Buchs abwärtskompatibel, sodass der interessierte Leser, unabhängig von der aktuellen Python Version, die Quellcodebeispiele benutzen kann.

Nachdem der Download der ungefähr 30 Megabyte großen Datei abgeschlossen ist, kann die Installation einfach ausgeführt werden. Analog zur Installation herkömmlicher Software mithilfe einer Installationsdatei wird der Benutzer durch verschiedene Dialogfenster (Abb. 2.1) geführt. Die Darstellung zeigt exemplarisch die Installation auf einem MacBook mit macOS-Betriebssystem.

Für die Installation von Python reserviert die Installationsdatei etwa 160 Megabyte auf dem Computer des Benutzers. Die Installation ist nach dem Dialogmenü und dem Installationsprozess abgeschlossen. Danach kann Python auf dem Computer des Benutzers verwendet werden.

Durch die zunehmende Verbreitung von mobilen Endgeräten wie Smartphones und Tablets gibt es zahlreiche Applikationen zur Programmierung mit Python auf diesen Geräten. Aufgrund des umfangreichen Angebots und des dynamischen Wandels können diese Lösungen in diesem Buch nicht dargestellt werden. Dieses Buch geht aus Gründen der besseren Nachvollziehbarkeit von dem Standardfall aus, dass der Leser einen handelsüblichen Laptop zur Programmierung verwendet. Sollte der Leser dennoch das Programmieren auf anderen mobilen Endgeräten, wie z. B. Smartphones und Tablets, bevorzugen, empfiehlt sich hier der Blick in den entsprechenden App-Store und natürlich ins Internet, um gegebenenfalls schon vor dem Download einer App herauszufinden, ob die App geeignet ist oder ein anderes Produkt gewählt werden sollte.

Ein kurzer Test mithilfe des sogenannten Terminals (bzw. Konsole in Windows) zeigt die installierte Python-Version. Durch den Aufruf des Terminals auf dem Computer und der Eingabe `python -v` wird die aktuelle Python-Version ausgegeben. Durch die Eingabe von `python` können Python-Codes direkt eingegeben werden. An dieser Stelle verweist dieses Buch jedoch darauf, dass das Schreiben von Quellcodes von Einsteigern über eine



Abb. 2.1 Das Python-Installationsfenster

Entwicklungsumgebung geeigneter sein kann. Vor diesem Hintergrund befasst sich der anschließende Abschnitt mit den Entwicklungsumgebungen von Python.

2.3 Entwicklungsumgebung

Zur Entwicklung von Quellcodes in Python empfiehlt sich die Verwendung einer professionellen Entwicklungsumgebung (IDE)¹. Für Python gibt es zahlreiche IDE; damit hat ein Einsteiger sprichwörtlich „die Qual der Wahl“. Grundsätzlich ist anzunehmen, dass eine Vielzahl von IDE über dieselben Grundfunktionalitäten verfügen. Auf der Webseite <https://wiki.python.org/moin/IntegratedDevelopmentEnvironments> gibt es eine Übersicht zahlreicher IDE. Neben den professionellen IDE ist es denkbar, Python-Quellcode in einem herkömmlichen Texteditor, wie z.B. Sublime Text² zu verfassen und über das Terminal auszuführen.

¹ Integrated Development Environment bezeichnet eine integrierte Entwicklungsumgebung, die eine wichtige Werkzeugkomponente darstellt, um von Funktionen wie einer Autovervollständigung und Programmcodeanalysen zu profitieren.

² Sublime Text findet sich auf der Webseite des Herstellers <https://www.sublimetext.com>.



Abb. 2.2 Willkommenfenster PyCharm

Im Rahmen dieses Buchs wurde die IDE PyCharm³ ausgewählt. Primär ist die Wahl auf diese IDE aufgrund der persönlichen Präferenzen gefallen. Allerdings ist es für den Leser in der Regel belanglos, welche IDE zum Einsatz kommt, da die Quellcodes unabhängig von der bevorzugten IDE funktionieren sollten. Vielmehr ist die gewählte IDE als Beispiel zu sehen, um den Installationsprozess und die grundlegenden Handgriffe mit einer IDE zu zeigen. Demnach kann also jede beliebige IDE vom Leser herangezogen werden.

Zur Installation von PyCharm kann die entsprechende Installationsdatei wie .exe oder .dmg über die Webseite heruntergeladen werden. Hierbei muss der potenzielle Nutzer zwischen zwei Versionen entscheiden. Die erste Variante ist die Community-Fassung, die kostenlos ist. Die andere Version ist die professionelle Fassung. Im Fall der reinen Python-Programmierung reicht die kostenlose Community-Fassung völlig aus. Nach der simplen Installation wird der Benutzer von der IDE nach einigen Benutzerpräferenzen gefragt. Hier empfiehlt es sich zunächst, die Standardeinstellung zu belassen. Diese Einstellungen lassen sich nachträglich noch ändern. Für die Darstellungen in diesem Buch kommt die PyCharm Version 2019.2.6 zum Einsatz (Abb. 2.2).

Das Willkommensfenster hat drei Menüpunkte zur Auswahl. Hierbei empfiehlt es sich zunächst mit einem neuen Projekt zu starten. In diesem Fenster sollte zuvor ein Projektordner auf dem Schreibtisch (Desktop) des Benutzercomputers angelegt werden. Dieser Ordner dient dann als Speicherplatz für alle Projektdateien. In diesem Fallbeispiel dient der Ordner

³ PYCharm findet sich unter <https://www.jetbrains.com/de-de/pycharm/>.

„Fallbeispiel“ als Demonstration. Im Menüpunkt „Project Interpreter“ sollte unter „Existing Interpreter“ die auf dem System installierte Python-Version ausgewählt werden. PyCharm bietet zahlreiche Möglichkeiten, Python als Programmiersprache in ein Projekt einzubinden. Für die Zwecke des Buchs sollte jedoch die installierte Python-Version des Systems ausreichen. Die Abb. 2.3 zeigt die Projektkonfiguration unter PyCharm, um ein Python-Projekt zu beginnen. Diese Auswahl wird anschließend mit der Bestätigung auf „Create“ ausgewählt, sodass ein Projekt angelegt wird.

Ausgehend von dem Startfenster, das aus der Darstellung des Projektordners und dem Auswahlmenü auf der linken Seite besteht, empfiehlt es sich, zunächst ein neues Python-Skript anzulegen. Dieses Skript soll in die Datei, in der der Quellcode geschrieben werden soll, abgelegt werden. Hierbei kann über einen Doppelklick auf den Ordner „Fallbeispiel“ ein neues „Python File“ als ein neues Skript angelegt werden. Python-Skripte sind einfache Folgen von Python-Ausdrücken, die in einer Textdatei zusammengefasst werden. Die Realisierung eines Skripts ist in Python leicht nachvollziehbar, da in PyCharm von oben nach unten gearbeitet wird. Zum besseren Verständnis hilft eine Analogie: Stellen Sie sich vor, Sie müssen in regelmäßigen Abständen einen Kuchen backen. Dieser Kuchen besteht aus einer klar definierten Menge und Zusammensetzung von Zutaten. Die Erstellung dieses Kuchens unterliegt immer den gleichen Schritten. Sie entscheiden sich dazu, dass Sie in

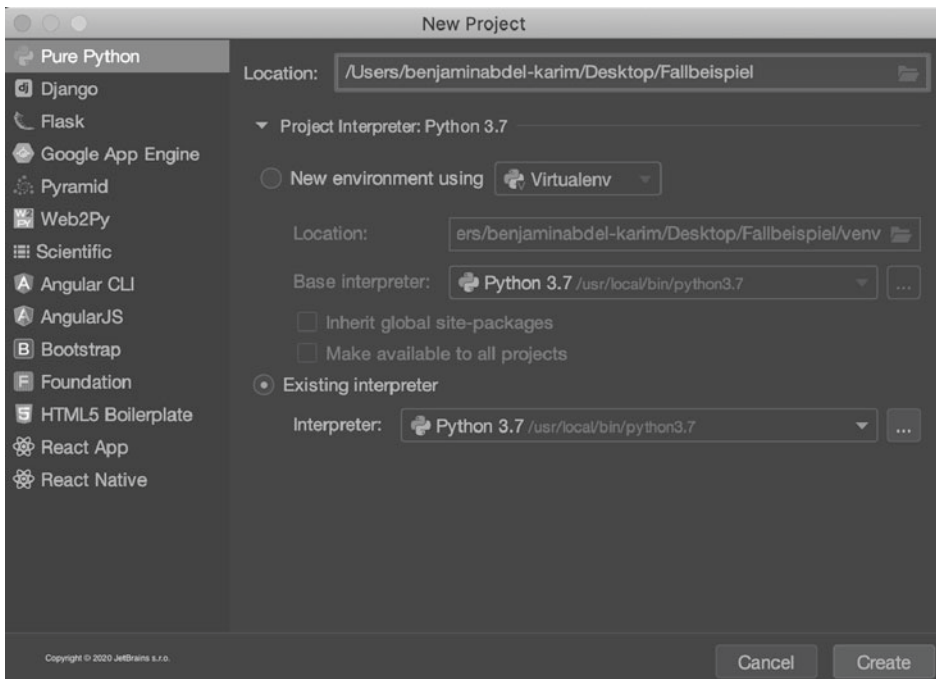


Abb. 2.3 Projektkonfiguration unter PyCharm

Zukunft diesen Kuchen nicht mehr selbst backen wollen, sondern Python dazu einsetzen wollen. Dabei soll Python den Kuchen identisch zu Ihrer Vorlage erstellen. Sie nutzen das Editorfenster zum Erstellen des Skripts oder die bereits angeführten Alternativen. Hierzu erstellen Sie ein Backrezept, das genau die Menge der Zutaten und die Reihenfolge angibt, in der Python für Sie den Kuchen auf Ihre Anweisung hin backen soll. Dieses Rezept ist das Skript. Bezogen auf unser Anwendungsbeispiel nennen wir das Skript für unsere Zwecke „b_HelloWorld.py“. Die Abb. 2.4 illustriert die Vorgehensweise.

Bezogen auf das Erstellen des Backrezepts nutzen wir eine Zutat in Form der ersten Eingabe. Nach Bestätigung der Eingabe öffnet sich das Skriptfenster, indem wir zum Testen die Funktion `print('Hello World')` eingeben. Anschließend wird das Skript über die linke Menüseite der Benutzeroberfläche (Graphical User Interface, GUI) mithilfe eines Rechtsklicks auf den Befehl „Run“ ausgeführt. `print` ist eine sogenannte Built-in-Funktion. Das sind Funktionen, die bei Aufruf in Python ausgeführt werden. Im späteren Verlauf folgt eine detailliertere Beschreibung von Funktionen, da hierzu einige Grundkenntnisse nötig sind. Durch die Auswahl dieses Menüpunkts wird unser Skript ausgeführt, sodass Python den Quellcode einliest und diesen entsprechend auf unserem Computer ausführt. Auf der sogenannten Konsole erscheint die Ausgabe `Hello World`. Die Abb. 2.5 zeigt das entsprechende Ergebnis. Damit haben wir gezeigt, dass unsere erste Einrichtung auf dem Computer von Python und PyCharm funktioniert hat.

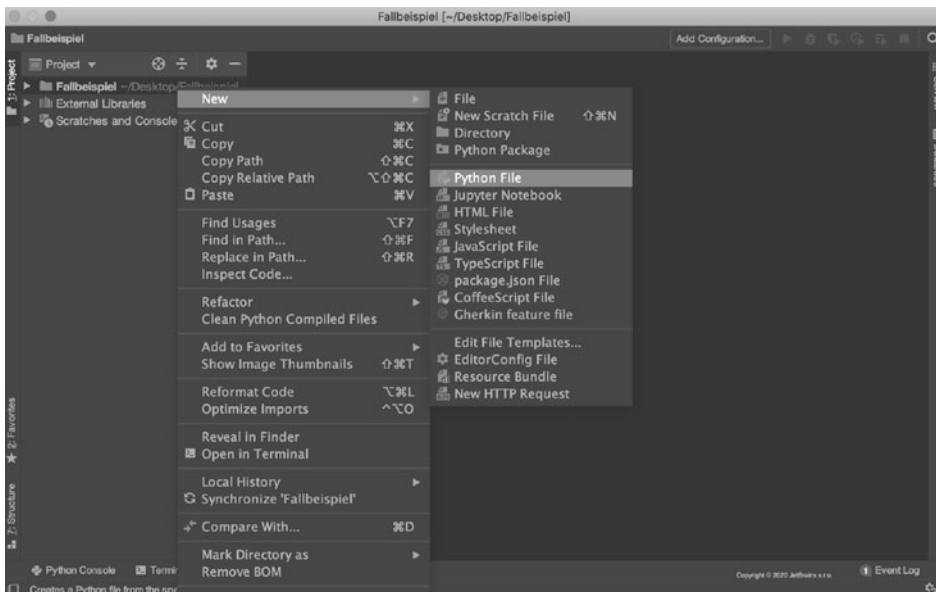


Abb. 2.4 Das erste Skript im Projektordner

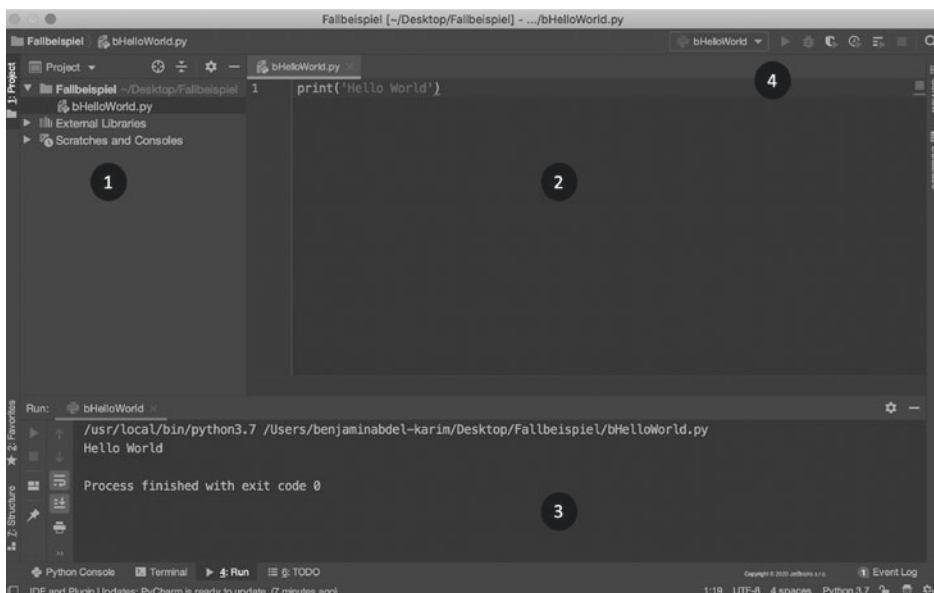


Abb. 2.5 Die IDE und Hello World

Die Annotationen der Abb. 2.5 veranschaulichen die zentralen Fenster nach der Skriptausführung. In 1 ist die Projektstruktur zu sehen, die im Wesentlichen die Struktur des Ordners widerspiegelt. Hier finden sich die angelegten Python-Skripte. In Punkt 2 ist das eigentliche Skript zu sehen. Hier können Anweisungen und Kommentare verfasst werden. Dieses Skript wird von Python von oben nach unten abgearbeitet. Die Konsolenausgabe findet sich in Punkt 3. Hier werden die Ergebnisse der Skriptausführung angezeigt. Durch die `print`-Funktionen werden Texte auf der Konsole ausgegeben. In diesem Fall wird „Hello World“ ausgegeben, weil dieser Aufruf so in der Funktion deklariert wurde. Nachdem das Skript das erste Mal ausgeführt wurde, schaltet PyCharm die Schnellausführungsfunktion frei. Diese in Punkt 4 dargestellte Funktion erlaubt die Ausführung der Skripte durch einen einfachen Klick. Im späteren Verlauf stellt sich heraus, dass dies eine sehr nützliche Funktion sein kann, wenn Skripte im Rahmen der Fehlerkorrektur mehrfach ausgeführt werden müssen.

2.4 Formalitäten für ein geeignetes Skript

Zum Verfassen von Skripten empfiehlt es sich, auf einige Dinge als Programmierer zu achten. Diese sollen in diesem Abschnitt kurz dargestellt werden. Jedes Skript sollte mit einer sogenannten Quellcodedokumentation beginnen. Hierbei handelt es sich um wichtige Kommentierungen, um die Quellcodezeilen mit lesbaren Hinweisen zu versehen. Zwar ist

die Verfassung solcher Kommentare zeitaufwendig, aber die Vorteile einer Quellcodedokumentation ergeben sich aus vielen Überlegungen. Ein dokumentierter Quellcode erlaubt es einer anderen Person, den verfassten Quellcode zu verstehen, mögliche Fehler zu erkennen und diese entsprechend zu korrigieren. Zum Zeitpunkt der Entstehung von Quellcodes werden Gedanken und Erkenntnisse aggregiert und durch den Code formalisiert. Dies ist in der Regel ein anspruchsvoller kognitiver Prozess. Im Zeitverlauf können diese aggregierten Informationen verblasen und der Entwickler oder eine andere Person hat große Mühe, den Quellcode nachzuvollziehen. Diesem Umstand kann eine ausführliche Dokumentation vorbeugen, indem sie die Gedanken bündelt. Allerdings ist das Verfassen einer geeigneten Dokumentation kein triviales Unterfangen. Eine ausführlichere Dokumentation ist sicherlich hilfreicher als keine Dokumentation. Allerdings ist eine triviale Aneinanderreihung von bekannten Kommentaren vielmehr Platzverschwendung. Vor diesem Hintergrund ist ein geeignetes Mittelmaß an Dokumentation eine gute Wahl. Die Grundsätze der Quellcodedokumentation werden in diesem Buch berücksichtigt, um den Leser die Nachvollziehbarkeit der Quellcodes zu ermöglichen. Zudem werden zusätzliche textuelle Beschreibungen geliefert. Die folgende Übersicht der Quellcodedokumentation soll eine Hilfestellung bieten:

- Name des Programmierers
- Datum der Skripterstellung
- Datum von Revision und Updates
- Name der jeweiligen Funktion
- Name der Anweisungen oder Funktion, gegebenenfalls Versionsnummer und Datum der Darstellung des Funktionsaufrufs
- Kurzbeschreibung der Aufgabe, die die Funktion erfüllt
- Beschreibung aller Inputparameter hinsichtlich Art, Dimension, Bedeutung usw., insbesondere sind notwendige und optionale Inputparameter klar zu kennzeichnen; alle Inputparameter, die nicht als optional kenntlich gemacht werden, müssen zwingende Inputparameter sein
- Beschreibung aller Outputparameter (Rückgabewerte der Funktion) hinsichtlich Art, Dimension und Bedeutung, insbesondere sind die Fehlercodes genau zu dokumentieren, falls eine Funktion solche zurückliefert

Unklare Begrifflichkeiten, wie beispielsweise „Funktion“, werden an geeigneter Stelle im Buch ausführlicher dargestellt. Sofern das Skript ausreichend Platz bietet, sollte die Dokumentation auch eine exemplarische Anwendung zeigen, die die Art und Weise des Funktionsaufrufs und die erwarteten Ergebnisse darstellt. Umfangreichere Programmcodeabschnitte sollten an den entsprechenden Stellen ausführlich beschrieben sein, damit eine andere Person die Möglichkeit hat, deutlich zu erkennen, auf welchen Abschnitt sich die Quellcodedokumentation bezieht. Das Beispiel im Code 2.1 illustriert das Verfassen einer geeigneten Quellcodedokumentation zu Beginn des Skripts.

Listing 2.1 Professionelle Quellcodedokumentation

```
1 # bSkript_Dokumentation
2 # Dieses Skript gibt einen Eindruck, wie sich eine professionelle Quellcode
3 # Dokumentation in Python erstellen laesst.
4 # @author: Benjamin M. Abdel-Karim
5 # @since: 2020-04-20
6 # @version: 2020-04-20 V.1
7
8 # Aufruf der @code: print() Funktion, um eine Konsolenausgabe zu erzeugen.
9 print('Hello World')
```

Durch die # werden Kommentare in der Programmiersprache Python gesetzt, sodass diese Zeilen von Python nicht weiter betrachtet werden. Alle Anweisungen, die nach einem # Symbol folgen, sind damit Textstellen, die als Kommunikation in Form einer Dokumentation eingesetzt werden können. Eine Besonderheit bildet das Sonderzeichen @ in der Quellcodedokumentation. Es dient dem Autor als Hashtag, um schnell nach wesentlichen Informationen in der Quellcodedokumentation zu suchen. Der Ursprung dieser Annotation liegt in der Programmiersprache Java und wurde vom Autor für Python übernommen. Der Leser kann von dieser Art und Weise der Schlüsselwortsetzung ebenfalls Gebrauch machen, um seine Quellcodes entsprechend zu strukturieren.

Ein weiterer Aspekt für ein geeignetes Skript ist das Benennen der Quellcodebestandteile. Diese sogenannten Benennungskonventionen werden an dieser Stelle des Buchs eingeführt, um als Hilfestellung zu dienen. Auch wenn Sie als Leser zu diesem Zeitpunkt mit den Begrifflichkeiten nicht vertraut sind, sollten Sie einen kurzen Blick auf diesen Teil werfen. Im späteren Verlauf des Werks werden die Texte auf diese Stelle zurückgreifen und dadurch zum Gesamtverständnis beitragen. Die Benennung von Quellcodeteilen in einer einheitlichen Art und Weise trägt zur Lesbarkeit des eigenen Quellcodes bei und soll damit die Codingeffizienz und Wartbarkeit steigern. Im Kontext des professionellen Software-Engineerings ist eine einheitliche Benennung unerlässlich, sodass eine konsistente und strukturierte Codeentwicklung durchgeführt werden kann. Damit steht die Standardisierung des Quellcodes im Fokus (vgl. Pomberger & Pree, 2004, S. 3 ff.). Bei der Benennung von Quellcodebestandteilen im Sinn einer Dokumentation und der besseren Lesbarkeit für andere Personen werden Benennungskonventionen für dieses Buch vereinbart. Dabei werden diese Quellcodebestandteile mit der CamelCase-Notation formuliert. Das ist eine Benennungskonvention der Informatik für Programmabschnitte (vgl. Roden, 2008, S. 33), die besagt, dass Bestandteile, die sich aus mehreren Wörtern zusammensetzen, ohne Leerzeichen verwendet werden. Der Anfangsbuchstabe jedes neuen Worts wird dabei groß geschrieben. Die konsequente Anwendung der Schreibweisenempfehlung zeigt die nachfolgende Auflistung:

- bSkriptName: Skripte beginnen mit einem kleinen b
- iNum: Variablen mit ganzen Zahlen beginnen mit einem kleinen i

- dNum: Variablen mit Gleitkommazahlen
- LList: Listen beginnen mit einem L
- fAddNum(): Funktionen beginnen mit einem f
- CPerson(): Klassen beginnen mit einem C

Die hier genannten Benennungskonventionen sollen für die Zwecke der ersten Kapitel in diesem Buch ausreichen, um die strukturierte Erstellung von Quellcodes zu veranschaulichen. Natürlich obliegt es dem Leser, diesen Vorschlag zur Benennungskonvention anzunehmen. Im Lauf der Case Studies zeigen die Co-Autoren weitere Alternativen auf.



Primitive Datentypen

3

Benjamin M. Abdel-Karim

Primitive Datentypen sind in der Informatik elementare Bausteine, um Daten effizient zu speichern und zu verarbeiten. Im Wesentlichen lassen sich die folgenden Grundformen unterscheiden: Zahlendatentypen (Abschn. 3.2 und 3.3), Zeichendatentypen (Abschn. 3.4) und Wahrheitswerte (Abschn. 3.5). In diesem Kapitel wird das nötige Grundlagenwissen der Programmierung vermittelt, das für die Anwendung von Python erforderlich ist. Der Fokus dieses Kapitels liegt auf den Variablen und den dazugehörigen primitiven Datentypen. Hierbei werden die relevanten und die Python-spezifischen Datentypen betrachtet. Ziel dieses Kapitels ist es demnach, ein Grundverständnis der Programmierung zu vermitteln und dabei Variablen und Datentypen kurz zu erläutern. Vor diesem Hintergrund ergeben sich die folgenden Lernziele des Kapitels:

- Grundlagen zum Verständnis von Programmiersprachen
- Grundverständnis für primitive Datentypen
- Zahlendatentypen
- String-Datentypen
- Verständnis von Wahrheitswerten

Ausgehend von den beschriebenen Lernzielen gibt Abb. 3.1 zunächst einen Überblick über die relevanten primitiven Datentypen, die für die Programmierung in Python relevant sein können.

B. M. Abdel-Karim (✉)
Frankfurt am Main, Hessen, Deutschland
E-mail: BenjaminM.Abdel-Karim@gmx.de

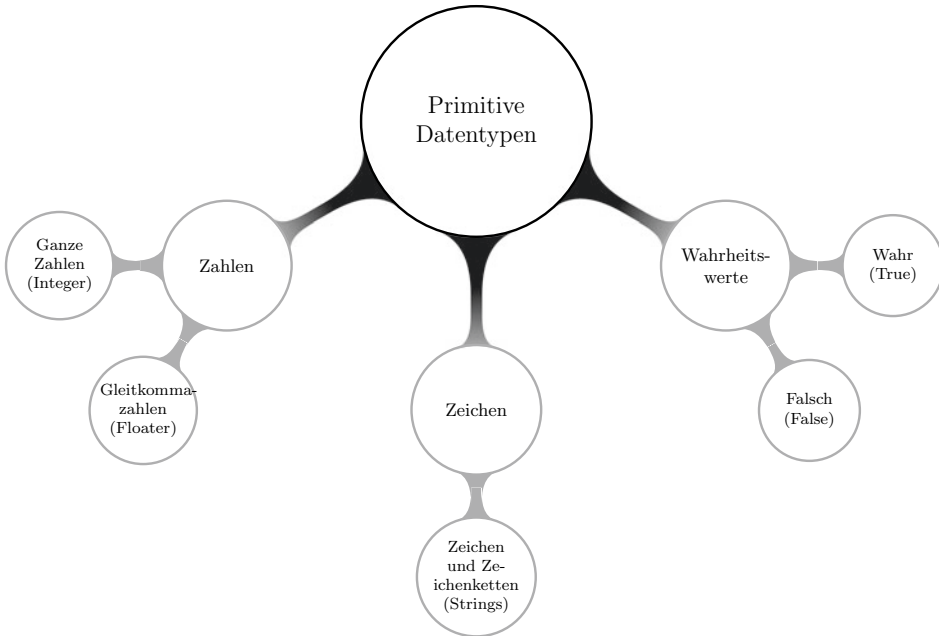


Abb.3.1 Ausgehend von den übergeordneten Klassen der primitiven Datentypen zeigt die Abbildung eine Auswahl der wichtigsten Datentypen, die in Python zum Einsatz kommen können

3.1 Grundlagen zum Verständnis von Programmiersprachen

Die Verwendung von Zahlen zur Quantifizierung der Welt ist vermutlich ebenso alt wie die Menschheit selbst. Im Lauf der Evolution hat sich die Verwendung von Zahlen im Prinzip nicht verändert. Die Fähigkeit, mit ihnen eine Ordnung (Ordinalität) zu schaffen sowie Objekten eine Wertigkeit zu verleihen (Kardinalität), ist heute von äußerster Relevanz. Im Lauf der Zeit wurden aus Strichlisten komplexe mathematische Konzepte, die als Instrument der Abstraktion Zugang zu neuen Welten ermöglichen. Vor diesem Hintergrund ist die Informatik die technische Implementierung der Mathematik. Diese Verwandtschaft ist allgegenwärtig und begegnet dem interessierten Leser unmittelbar bei den ersten Gehversuchen in dieser Welt. Denn die binäre Welt aus Einsen und Nullen in der Informatik ist der Versuch, die moderne Mathematik auf die Ebene der kleinsten technischen Einheit, dem Transistor in der Informatik, abzubilden. Die Thematisierung der Grundmechanismen der Zahlendarstellung ist also notwendig, um ein Verständnis für die Relevanz der Datentypen zu erlangen. Transistoren ermöglichen es, Stromimpulse weiterzuleiten oder zu unterbrechen, sodass durch ihre Schaltung komplexere Abfolgen von Stromimpulsen möglich sind. In modernen Computern finden auf den *Central Processing Units* (CPU) mehrere Millionen

Transistoren Platz. Die CPU als Einheit dient dazu, alle logischen, arithmetischen und sonstigen Steuerungsoperationen auszuführen. Damit fußen alle auf dem Computer angewiesenen komplexeren Befehle letzten Endes auf einem binären System aus Einsen und Nullen (Strom ein oder Strom aus).

Die Grundüberlegung in der Programmierung ist also die Zuweisung von Daten zu Variablen. Hierbei können Variablen als Behälter aufgefasst werden, um Informationen zu speichern. Viele Programmiersprachen wie Java erwarten allerdings, dass die Variablen vor der Initialisierung, also der Wertebelegung, einen Datentyp zugewiesen bekommen. Dies ist in Python jedoch anders. Die Variablen werden erst im Moment der Zuweisung mit einem Datentyp deklariert, sodass es in Python keine Variablendeklaration gibt. Dies ist für den Programmierer insofern eine Erleichterung, weil er sich nicht über die Zustände der Variablen zur Programmausführung (Laufzeit) Gedanken machen muss. Dieser einsteigerfreundliche Umstand wird aber von einigen Anhängern anderer Programmiersprachen als die größte Schwäche von Python gesehen, denn die dynamische Variablendeklaration führt dazu, dass Python verhältnismäßig langsam arbeitet. Sofern Sie aber in Python keine Luft- und Raumfahrtanwendung schreiben oder versuchen, Ihren eigenen Satelliten zu bauen, sollte dieser Umstand Sie nicht weiter beschäftigen, weil dieser zusätzliche Laufzeitaufwand, bedingt durch die dynamische Variablendeklaration in Python, für diese Anwendungsfälle zu vernachlässigen ist, vor allem auf den modernen handelsüblichen Computersystemen.

Damit sind die primitiven Datentypen in der Programmierung ein zentrales Konzept, bei dem verschiedene Datentypen in Variablen gespeichert werden, um unterschiedliche Operationen auf ihnen auszuführen. Standardmäßig unterscheidet Python im Kontext der primitiven Datentypen die folgenden Arten:

- Zahlen: Integer, Float, Complex
- Zeichen: Strings
- Wahrheitswerte: Boolean

Diese Datentypen werden in den folgenden Abschnitten näher betrachtet und durch kleine Beispiele illustriert.

3.2 Ganze Zahlen

Ausgehend von der obigen Argumentation bedeutet die Zuweisung einer ganzen Zahl zu einer Variable, dass Python für diese Zahl Speicherplatz reservieren muss. Ferner wird eine ganze Zahl im internen System des Computers in einer binären Darstellung aus Nullen und Einsen dargestellt. Die Darstellung erfolgt durch die Transistoren und ihren jeweiligen Zuständen, sodass diese Zahl in Form einer binären Darstellung gespeichert wird. Ganze

Tab. 3.1 Übersicht nützlicher arithmetischer Operatoren mit Integer. *iX* = Integer-Variable; *iY* = weitere Integer-Variable

Funktion	Auswertung zu:
<i>iX</i> + <i>iY</i>	Addition
<i>iX</i> − <i>iY</i>	Subtraktion
<i>iX</i> * <i>iY</i>	Multiplikation
<i>iX</i> / <i>iY</i>	Division
<i>iX</i> % <i>iY</i>	Rest der ganzzahligen Division
<i>iX</i> ** <i>iY</i>	Potenzieren

Zahlen bekommen in Python den Datentyp integer zugewiesen. Integer (kurz: int) lassen sich definieren als ganze Zahlen, die mit Vorzeichen versehen sein können, sodass sie sowohl positiv oder negativ sein können. Zentrales Merkmal ist, dass es Zahlen ohne Dezimalstelle sind. In Python sind für die ganzzahligen numerischen Datentypen beispielsweise die in Tab.3.1 dargestellten arithmetischen Operatoren möglich.

3.3 Gleitkommazahlen

Zu den ganzen Zahlen bringt Python standardmäßig auch Gleitkommazahlen mit. Der Zahlendatentyp Float dient zur Darstellung von Gleitkommazahlen. Diese lassen sich auffassen als Zahlen mit einem Vorzeichen und einem Komma. In Python sind für die Gleitkommazahlen beispielsweise die dargestellten arithmetischen Operatoren möglich (Tab. 3.2)

Tab.3.2 Übersicht nützlicher arithmetischer Operatoren mit Float. *dX* = Float-Variable; *dY* = weitere Float-Variable

Funktion	Auswertung zu:
<i>dX</i> + <i>dY</i>	Addition
<i>dX</i> − <i>dY</i>	Subtraktion
<i>dX</i> * <i>dY</i>	Multiplikation
<i>dX</i> / <i>dY</i>	Division
<i>dX</i> % <i>dY</i>	Rest der ganzzahligen Division
<i>dX</i> ** <i>dY</i>	Potenzieren

3.4 Zeichen und Zeichenketten in der Programmierung

Von den Zahlendatentypen sind die alphanumerischen Datentypen abzugrenzen. In Python werden als Zeichen und als Zeichenketten die sogenannten Strings als primitive Datentypen eingesetzt. Der Begriff leitet sich aus dem Englischen ab und definiert im Kontext der Programmierung die sequenzielle Abfolge von Zeichen. Jede Zeichenkette besteht damit aus einzelnen Zeichen. In Analogie zur menschlichen Sprachen bilden die einzelnen Zeichen Zeichenketten und können zu komplexeren Strukturen zusammengesetzt werden. In Python können Strings sowohl mit `' '` als auch mit `" "` deklariert werden. Der Quellcode 3.1 zeigt die Variablendeklaration mit einem String.

Listing 3.1 Deklaration eines String

```
1 # Variablen Deklaration mit einem String.
2 # Hierbei erfolgt die Zuweisung (=) des Namens Benny als String zur Variable sName
3
4 sName = 'Benny'
5 print(sName)
6
7 >> Benny
```

Der Ausdruck `sName = 'Benny'` erzeugt ein Python-Objekt vom Typ String, das den Wert „Benny“ zugewiesen bekommen hat. Dieser Wert wird nun auf Prozessorebene als Abfolge von Zahlen dargestellt. Damit ist jedes Zeichen innerhalb der Zeichenketten als Zahl codiert. Die Zeichenkette besteht aus fünf einzelnen Zeichen. Der Ausdruck `sName = 'Benny'` deklariert zunächst eine Zeichenkette aus den fünf Buchstaben und kann dann über die Variable `sName` auf der Konsole ausgegeben werden. Hierbei sollten die Benennungskonventionen aus Abschn. 2.4 berücksichtigt werden. Die *Strings* werden mit dem vorangestellten Kleinbuchstaben `s` (für *String*), gefolgt von einem Großbuchstaben, dann gefolgt von einer beliebigen Zeichenkette aus Groß- und Kleinbuchstaben benannt.

3.5 Boolean

Der primitive Datentyp *Boolean* definiert die klassischen Wahrheitswerte im Sinn der Aussagenlogik, wie sie in der Mathematik verwendet werden. Sie können zunächst zwei Zustände annehmen:

- `true`
- `false`

Die genauere Verwendung von Wahrheitswerten wird anhand eines Beispiels (Code: 3.2) verdeutlicht:

Listing 3.2 Operationen mit Boolean in Python

```
1 # Python soll pruefen, ob 10 groesser 5 ist
2 bBoolean = 10 > 5
3 print(bBoolean)
4
5 >> True
6
7 # Das Gegenbeispiel:
8 bBoolean = 10 < 5
9 print(bBoolean)
10
11 >> False
```

Im ersten Fall wird auf der Konsole des Benutzers **True** ausgegeben. Das bedeutet, die Aussage trifft zu, da die zu überprüfende Aussage wahr ist. Beim Gegenbeispiel zeigt die Konsolenausgabe **False** an, da die Aussage falsch ist.

In den modernen Programmiersprachen werden solche Objekte, die sich aus Vergleichen bilden, wie die aus dem Codebeispiel 3.2, in der Regel als *boolesche Variablen* bezeichnet. Python hat dabei die gängigen Vergleichsoperatoren wie **>** (größer), **<** (kleiner), **==** (gleich), **>=** (größer-gleich) oder **<=** (kleiner-gleich) implementiert. Durch den Einsatz von Verknüpfungsoperatoren können diese Bedingungsprüfungen miteinander verbunden werden, damit kombinierte logische Ausdrücke gebildet werden können. Dabei sind die beiden Verknüpfungen **and**- und **or**-Operator zu unterscheiden. Eine Verallgemeinerung der Verknüpfung lässt sich wie folgt definieren: *Bedingung Nr. 1 and Bedingung Nr. 2*. Dabei sollten in dieser Vorgehensweise beide zutreffen. Im nächsten Anwendungsbeispiel gilt: *Bedingung Nr. 1 or Bedingung Nr. 2*. In diesem Anwendungsfall reicht die Erfüllung einer der beiden Bedingungen aus, um als wahr klassifiziert zu werden.

Die Umsetzung in Python zeigt das folgende Codebeispiel 3.3.

Listing 3.3 Verknüpfungsoperatoren mit Logicals in Python

```
1 # Python soll pruefen, ob 10 groesser ist als 9 und ob 10 kleiner ist als 100
2 bConditionA = 10 > 9 and 10 < 100
3 print(bConditionA)
4
5 >> True
6
7 # Das Ergebnis ist True, weil die beide Bedingungen erfuehlt worden sind und es gilt:
  9 < 10 und 10 < 100.
8
9 # 24 > 100 || 24 <= 24 Hier reicht aus, wenn eine der beiden Bedingungen zutrifft,
10
11 bConditionB = 24 > 100 or 24 <= 24
12 print(bConditionB)
13 >> True
14
15 # Zwar ist die erste Bedingung unwahr, jedoch erfuehlt die zweite Anweisung die
  Bedingung, sodass Wahr zurueckgeliefert wird.
```

Tab. 3.3 Übersicht der Vergleichsoperatoren

Ausdruck	Auswertung zu:
$X > Y$	Wertet zu true aus, wenn X größer Y ist
$X < Y$	Wertet zu true aus, wenn X kleiner Y ist
$X == Y$	Wertet zu true aus, wenn X und Y gleich sind
$X >= Y$	Wertet zu true aus, wenn X größer oder gleich Y ist
$X <= Y$	Wertet zu true aus, wenn X kleiner oder gleich Y ist

Der mögliche Verknüpfungsspielraum geht gegen unendlich, jedoch sollte über die Länge einer Kette an Bedingungen kritisch nachgedacht werden. Die [Tab. 3.3](#) zeigt einige Vergleichsoperatoren.



Benjamin M. Abdel-Karim

Ausgehend von primitiven Datentypen in Kap. 3, gibt dieses Kapitel einen Überblick über die Datenstrukturen. Die Datenstrukturen dienen in der Regel dazu, die primitiven Datentypen in Strukturen zu organisieren. Dieses Kapitel stellt dabei die wesentlichen Datenstrukturen der Programmiersprache Python vor. Den Anfang machen die Listen (in Abschn. 4.1). Daran schließen die sogenannten Dictionaries an (Abschn. 4.2). Abschließend geht dieses Kapitel auf die Mengen ein (Abschn. 4.3). Aus den genannten Themenbereichen dieses Kapitels ergeben sich die folgenden Lernziele, die für einen geeigneten Umgang mit Python wichtig sind:

- Grundlagen zum Verständnis von Listen erhalten
- Ein Verständnis über die Dictionaries gewinnen
- Kurzer Einblick zu Mengen

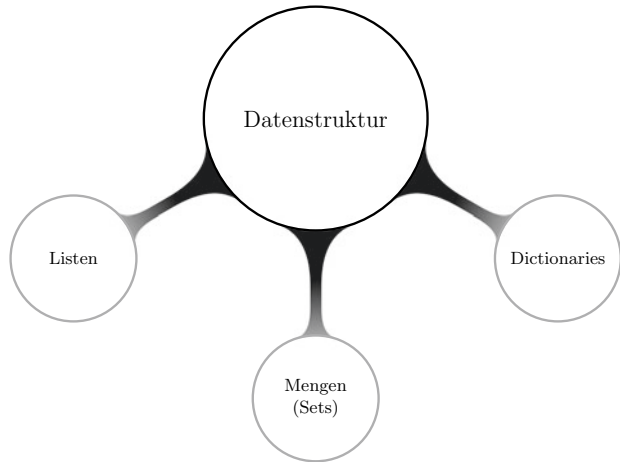
Ausgehend von den beschriebenen Lernzielen gibt die Abb. 4.1 zunächst einen Überblick über die relevanten Datenstrukturen, die für die Programmierung in Python relevant sein können.

4.1 Listen in Python

Listen sind nützlich, um primitive Datentypen zu speichern, und bieten damit die Möglichkeit, effizient auf die primitiven Datentypen zuzugreifen. Damit sind Listen prädestiniert für eine effiziente Datenverwaltung. Der Fokus dieses Kapitels ist die Darstellung des

B. M. Abdel-Karim (✉)
Frankfurt am Main, Hessen, Deutschland
E-mail: BenjaminM.Abdel-Karim@gmx.de

Abb. 4.1 Ausgehend von der übergeordneten Klasse der Datenstruktur wird eine Auswahl der wichtigsten Datenstrukturen gezeigt, die in Python zum Einsatz kommen können



grundlegenden Konzepts hinter den Listen in Python. Listen sind äußerst relevant, da sie in der praktischen Anwendung allgegenwärtig sind. Sie ermöglichen dem Programmierer eine geeignete Datenverwaltung und Verarbeitung. Die Stärke der Listen liegt vermutlich in ihrer Einfachheit. Speziell in Python können Listen relativ einfach implementiert und verwaltet werden. Aus einer formalen Perspektive ergibt sich die Schlussfolgerung, dass eine Liste eine dynamisch veränderbare Sammlung von Elementen ist. Jedes Element in einer beliebigen Liste verfügt über einen eigenen Index. Die Indexierung von Elementen in einer Liste ermöglicht den Zugriff auf eben jedes Element über seinen Index. Die Formalisierung 4.1 zeigt mögliche Interpretationen der Liste, bestehend aus Indizes und ihren Elementen. Das bedeutet, jedes Element besitzt seinen eigenen Index.

$$\text{Liste} = \left[\begin{array}{l} \text{index :} \quad 0 \quad 1 \quad 2 \quad \dots \quad n \\ \text{element :} \text{element}_0 \text{ element}_1 \text{ element}_2 \dots \text{element}_n \end{array} \right] \quad (4.1)$$

Weiterhin stellen Listen Datencontainer dar. Die einzelnen Güter in einem Container entsprechen den eingelagerten Elementen. Damit ein Element aus dem Container geholt werden kann, wird der Index des Elements genutzt. Der Index entspricht in dieser Analogie einem Lagerverzeichnis. Das Anlegen einer Liste und die Aufnahme eines Elements zeigt das Skript 4.1.

Listing 4.1 Eine Liste in Python anlegen

```

1 # Die Verwendung von Listen in Python.
2 # @author: Benjamin M. Abdel-Karim
3 # @since: 2020-04-24
4 # @version: 2020-04-24 V1
5
6 # Eine Liste erstellen.
7 LNames = []
  
```

Das Anlegen einer Liste gelingt in Python mit der Benennung einer neuen Variable. Analog zur vorgestellten Benennungskonvention (Abschn. 2.4) wird die Variable für die Liste mit einem großen L eingeleitet. Nachfolgend erfolgt die Bezeichnung für den Inhalt der Liste. Für dieses Fallbeispiel wird die Liste mit Zeichenketten (Strings) in Form von Namen gefüllt. Es folgt `Names` als Bezeichner. Damit ergibt sich als Variablenname `LNames`, angeführt von einem Gleichheitszeichen und zwei eckigen Klammern.

In Python verfügen Listen über eigene und vorimplementierte Funktionen, die es ermöglichen, Operationen auf den Daten durchzuführen. Eine zentrale Funktion, die Listen naturgemäß mitbringen, ist die `append()`-Funktion, um neue Elemente in eine Liste aufzunehmen. Dabei wird das aufzunehmende Element an das Ende der Liste angehängt. In diesem Fallbeispiel soll eine Liste mit Namen angelegt werden. Das Codebeispiel 4.2 zeigt die Verwendung der `append()`-Funktion, um einen Namen in der neu angelegten Liste 4.1 anzulegen. Damit soll „Julia“ der Liste hinzugefügt werden.

Listing 4.2 Ein Element in eine Liste aufnehmen

```
9 # Einen Namen in die Namensliste aufnehmen.  
10 LNames.append('Julia')
```

Damit nun ein Zugriff auf den ersten Namen oder auf ein anderes beliebiges Element der Liste möglich ist, kann die Indexierung der Liste verwendet werden. Der Indexzugriff auf eine Liste erfolgt mithilfe von eckigen Klammern am Ende der jeweiligen Liste. Vor diesem Hintergrund wird eine neue Variable angelegt, die das erste Listenelement übergeben bekommt. An dieser Stelle ist wichtig zu wissen, dass die Indexierung einer Liste mit 0 beginnt. Das Codebeispiel 4.3 zeigt exemplarisch eine mögliche Implementierung in Python.

Listing 4.3 Zugriff auf das erste Listenelement

```
12 # Zugriff auf das erste Element der Liste, um an den Namen zu gelangen.  
13 sFirstNameFromList = LNames[0]  
14 print(sFirstNameFromList)
```

Dabei wird auf die Indexierung der Liste über die eckigen Klammern zugegriffen, sodass mit der Übergabe der 0 das erste Element der Liste zurückgegeben wird. In diesem Codebeispiel ist es das erste Element an der Stelle 0.

Die Rückgabe der Liste über den Indexzugriff wird anschließend der neu angelegten Variable `sFirstNameFromList` zugewiesen. Daraufhin wird die Variable `sFirstNameFromList` mithilfe der `print`-Funktion auf der Konsole ausgegeben. Das Ergebnis ist der Name `Julia`, da dieser in der Liste auf der ersten Position steht. Die Anwendungsmöglichkeiten der Listen in Python sind zahlreich. Zum besseren Verständnis der Anwendung von Listen sollen im Folgenden Beispiele die Nutzung verdeutlicht werden.

Ausgehend vom Beispiel der primitiven Datentypen in Listen, sollte vollständigkeitshalber erwähnt werden, dass Listen ebenfalls andere Datenstrukturen wie weitere Listen beinhalten können. Vor diesem Hintergrund kann eine Liste weitere Listen an den Indexstel-

len bereithalten. Dies führt zu einer zusätzlichen Komplexitätsstufe hinsichtlich der Dimension. Die Formalisierung 4.2 zeigt die Systematik hinter dem Konzept der verschachtelten Listen.

$$\text{Liste in Liste} = \left[\begin{array}{lcl} \text{index :} & 0 & \dots n \\ \text{element :} & \text{List}_0 & \dots n \\ & \text{element}_0 & \\ & \text{element}_1 & \\ & \dots & \\ & \text{element}_n & \end{array} \right] \quad (4.2)$$

Vor dem Hintergrund, dass die erste Liste an einem beliebigen Index eine weitere Liste beinhaltet, entsteht unweigerlich eine Dimensionerweiterung für den Index. Damit könnte eine Liste aus fünf Elementen bestehen und eines dieser Elemente wäre eine weitere Liste. Denn durch die Einführung einer weiteren Liste und ihrem Index, kommt es zu einer Verschachtelung. Auf der ersten Ebene der ersten Liste besteht damit ein Index, exemplarisch *i*. Auf der Ebene der Liste, die ein Element der ersten Liste ist, existiert ein weiterer Index, exemplarisch *j*. Für den hypothetischen Zugriff auf das erste Element der zweiten Liste, analog zur Formalisierung 4.2 also das Element element_0 , werden also die Indices (*i*, *j*) benötigt. Damit ein Zugriff auf element_0 möglich ist, müssen die Werte (0, 0) der ersten Liste übergeben werden. Durch $i=0$ wird auf die erste Position der ersten Liste zugegriffen, durch $j=0$, auf das erste Element der Liste an der Stelle 0.

Zur Illustration des Beispiels soll der Code dienen. Hierbei wird eine Liste `LPerson` erstellt, die direkt mit einer weiteren Liste initialisiert wird. Diese weitere Liste besteht aus Strings mit Vornamen. Das Ziel ist, den Zugriff auf das erste Listenelement der Vornamenliste (hier Index 0) zu realisieren.

Listing 4.4 Liste in Liste

```

16 # Die Listen in der Liste
17 LPerson = [['Lisa', 'Max', 'Marlene']]
18 print(LPerson[0][0])

```

Analog zur Formalisierung 4.2 zeigt das Codebeispiel 4.4 die Umsetzung in Python. Durch die Übergabe der Indices [*i*, *j*] mit den Werten 0 und 0 wird das erste Element `Lisa` auf der Konsole ausgegeben.

4.1.1 Liste als Stapelspeicher (Stack)

Eine besondere Art und Weise in der Verwendung von Listen ist der Anwendungsfall, Listen als Stapelspeicher (Stack) zu verwenden. Hierbei werden neue Elemente in der Liste entsprechend des Last-in- und First-out-Prinzips so gespeichert, dass ein neues Element an erster Stelle steht. Die Implementierung des Stack in Python (und auch anderen Program-

miersprachen) lässt sich mit einem Stapel von Dokumenten vergleichen. In dieser Analogie werden neue Dokumente oben auf dem Stapel abgelegt, sodass das oberste Dokument das letzte ist, was hinzugekommen ist. Das Codebeispiel 4.5 zeigt die Implementierung einer Liste als Stapelspeicher.

Listing 4.5 Liste als Stapelspeicher

```
1  # Die Verwendung von Listen als Stack in Python.
2  # @author: Benjamin M. Abdel-Karim
3  # @since: 2020-04-24
4  # @version: 2020-04-24 V1
5
6  # Ein neuer Stack wird angelegt.
7  LStack = []
8
9  # Drei neue Rechnungen kommen hinzu.
10 LStack.append('Rechnung_1')
11 LStack.append('Rechnung_2')
12 LStack.append('Rechnung_3')
13 print(LStack)
14
15 # Die oberste Rechnung wird aus dem Stack genommen.
16 LStack.pop()
17 print(LStack)
```

Zur besseren Illustration bedient sich das Codebeispiel an der getroffenen Analogie. Hierbei wird zunächst eine Liste angelegt und entsprechend mit der `append()`-Funktion mit drei Zeichenketten `'Rechnung_1'`, `'Rechnung_2'`, `'Rechnung_3'` gefüllt. Anschließend soll entsprechend des Prinzips eines Stack, die zuletzt hinzugefügte Zeichenkette `'Rechnung_3'` entfernt werden, sodass im Stack nur noch die Zeichenketten `'Rechnung_1'`, `'Rechnung_2'` verbleiben. Das Entfernen der obersten Zeichenkette, also dem Element, was zuletzt einer Liste hinzugefügt wurde, übernimmt die `pop()`-Funktion. Sie realisiert die Entfernung des letzten Elements.

4.1.2 Listen als Warteschlange (Queue)

Ein weiterer Anwendungsfall von Listen sind sogenannte Warteschlangen (Queue). Bei einer Warteschlange wird nach dem First-in- und dem First-out-Prinzip das erste eingefügte Element zuerst bearbeitet. Für ein besseres Verständnis der Funktionen wird dieselbe Analogie wie im Abschn. 4.1.1 genutzt. Das folgende Codebeispiel zeigt die Vorgehensweise in der Realisierung der Warteschlange. Hierbei wird angenommen, dass die Rechnungen entsprechend nacheinander abgelegt werden, sodass die letzte Rechnung am Schluss der Warteschlange liegt.

Listing 4.6 Liste als Warteschlange

```
1  # Die Verwendung von Listen als Stack in Queue
2  # @author: Benjamin M. Abdel-Karim
3  # @since: 2020-04-24
4  # @version: 2020-04-24 V1
5  # Ein neuer Stack wird angelegt.
6  LQueue = []
7
8  # Drei neue Rechnungen kommen hinzu.
9  LQueue.append('Rechnung_1')
10 LQueue.append('Rechnung_2')
11 LQueue.append('Rechnung_3')
12 print(LQueue)
13
14 # Die letzte Rechnung soll entfernt werden.
15 LQueue.remove(LQueue[0])
16
17 print(LQueue)
```

Zunächst ist festzuhalten, dass die Aufnahme der Rechnungen in derselben Reihenfolge wie aus dem vorherigen Codebeispiel 4.5 erfolgt. Die Aufnahme ist also identisch zum obigen Beispiel, aber nicht die Ausgabe. Zum Entfernen eines Elements aus der Warteschlange wird die `remove()`-Funktion genutzt. Diese Funktion erwartet als Eingabewert das Element, das entfernt werden soll. Mithilfe der Indexierung und der `remove()`-Funktion ist es möglich, beispielsweise das erste Element der Liste zu entfernen. Durch diesen Funktionsaufruf wird in Python veranlasst, das erste Element an Stelle 0 zu entfernen. Dies führt dazu, dass die Indizes der verbleibenden Elemente durch Python um eine Stelle verschoben werden müssen.

Die oben genannten Beispiele werden als String-Elemente deklariert. An dieser Stelle muss aber gesagt werden, dass Listen ebenfalls Zahlen in Form von Integers und Floats aufnehmen sowie unterschiedliche primitive Datentypen verwalten können. Zudem bieten Listen in der Standardvariante weitere Methoden, die sich aus der Python-Dokumentation ergeben. Auf eine weiterführende Darstellung der Python-Dokumentation verzichten wir an dieser Stelle, da dies die Zielsetzung des Buchs verfehlen würde.

4.2 Dictionaries in Python

In Python existiert eine weitere hilfreiche Datenstruktur mit dem Namen Dictionary. Hierbei handelt es sich um eine Datenstruktur, die die Datenverwaltung mithilfe von sogenannten Assoziationen realisiert. Dabei bekommt, in Abgrenzung zu einer Liste, jedes Element einen Schlüssel zugewiesen. Das Dictionary speichert entsprechend ein Paar, bestehend aus einem Schlüsselwort (Key) und einem dazugehörigen Wert (Value). Ferner ist ein Dictionary eine

unsortierte Menge bestehend aus Elementen und ihren dazugehörigen Schlüsseln. In diesem Kontext bedeutet unsortiert, dass es keine festgelegten Plätze gibt. Dies ist die Abgrenzung zu Listen, in denen es festgelegte Plätze gibt. Das Besondere hierbei ist, dass der Datentyp des Schlüssels nicht dem primitiven Datentyp des zu speichernden Werts entsprechen muss, sondern davon abweichen kann. Die Formalisierung 4.3 des Dictionary kann wie hier gezeigt erfolgen:

$$\mathbf{Dic} = [key_i : element_i \ key_{i+1} : element_{i+1} \ ... \ key_n : element_n] \quad (4.3)$$

In gewisser Weise kann geschlussfolgert werden, dass die Indexierung aus den eingangs erklärten Listen hier durch die Verwendung von Schlüsseln ersetzt wird. Mit dieser Schlussfolgerung folgt ebenfalls, dass die Schlüsselwörter innerhalb eines Dictionary eindeutig sein müssen. Die Verwendung eines Dictionary zeigt das folgende Quellcodebeispiel 4.7. In diesem Beispiel wird zunächst ein Dictionary angelegt. Bei der Initialisierung des Dictionary wird es entsprechend mit Werten belegt, um dann auf die Werte über den Key zuzugreifen und das Dictionary anschließend um neue Werte zu aktualisieren. Am Schluss erfolgt eine Übersicht aller Schlüssel.

Listing 4.7 Dictionary

```

1  # Die Verwendung von Dictionaries in Python.
2  # @author: Benjamin M. Abdel-Karim
3  # @since: 2020-04-24
4  # @version: 2020-04-24 V1
5
6  # Anlegen eines Dictionary.
7  Dic = {'vorname': 'Lisa', 'nachname': 'Mustermann', 'alter': 25, 'weiblich': True}
8
9  # Zugriff auf einen Wert ueber den Schluessel
10 sVorname = Dic.get('vorname')
11 print(sVorname)
12
13 sAlter = Dic.get('Alter')
14 print(sAlter)
15
16 # Weitere Werte hinzufuegen
17 Dic.update({'Einkommen': 1000})

```

Im Quellcodebeispiel 4.3 wird das Dictionary dazu genutzt, um Werte einer Person zu speichern. Diese Werte werden, wie oben angeführt, jeweils als Tupel aus Schlüsselwort und Wert hinzugefügt. In diesem Fall sind es die Schlüssel und die Werte für `vorname`, `nachname`, `alter` und `geschlecht`. Das hier angeführte Beispiel zeigt, dass die Schlüsselwörter Zeichenketten sind. Die Werte bestehen allerdings aus verschiedenen primitiven Datentypen. Einmal werden zur Codierung der Vor- und Nachnamen Strings verwendet. Für das Alter wird ein Integer und für die Geschlechtscodierung wird ein boolean benutzt,

um anzuzeigen, ob die Person weiblich ist (wahr bzw. true) oder nicht (unwahr bzw. false). Zur Demonstration des Zugriffs auf den Wert des Vornamens über den Schlüssel wird die `get()`-Funktion genutzt. Diese bekommt den gewünschten Schlüssel übergeben und weist diesem Wert eine neue Variable `sVorname` zu. Anschließend wird das Ergebnis auf der Konsole ausgegeben, sodass auf der Konsole `Lisa` erscheint. Analog dazu führt das Codebeispiel dies auch mit dem Schlüsselwort `alter` durch.

Zur Aktualisierung des Dictionary wird ein weiteres Tupel, bestehend aus dem Schlüsselwort und dem Wert, mithilfe der `update()`-Funktion hinzugefügt. Jedes neue Tupel kann, analog zur Formalisierung 4.3, als Paar aus Schlüsselwort und dem eigentlichen Wert, getrennt durch Doppelpunkte, einem Dictionary hinzugefügt werden.

4.3 Mengen (Sets) in Python

Eine spezielle Form der Datenstrukturen in Python sind die Mengen (Sets.). Eine Menge zeichnet sich durch die unsortierte Aufnahme von einzigartigen Elementen aus. Eine Menge ist damit eine Ansammlung von Elementen, die nicht doppelt vorkommen. Dies impliziert die Möglichkeit, Operationen aus der Mengenlehre mit Mengen in Python durchzuführen, wie beispielsweise die Bestimmung der Differenz, der Schnittmenge oder der Vereinigungsmenge. Der Beispielcode 4.8 zeigt einige der möglichen Operationen zu Mengen.

Listing 4.8 Mengen

```
1 # Die Verwendung von Listen in Python.
2 # @author: Benjamin M. Abdel-Karim
3 # @since: 2020-04-24
4 # @version: 2020-04-24 V1
5
6 # Zwei Sets erstellen.
7 SA = set('ABCD')
8 SB = set('DEFG')
9
10 # Alle Elemente aus A ausgeben lassen.
11 print(SA)
12
13 # Alle Elemente, die in A oder B zu finden sind.
14 print(SA ^ SB)
15
16 # Alle Elemente finden, die in A und nicht in B sind.
17 print(SA-SB)
18
19 # Alle Elemente, die in A oder B zu finden sind.
20 print(SA | SB)
21
22 # Alle Elemente finden, die sowohl in A als auch in B sind.
23 print(SA & SB)
```

Auf Basis des Beispielcodes 4.8 wird zunächst ein Set `SA` mit den Elementen `ABCD` und ein weiteres Set mit den Elementen `SB` mit den Elementen `DEFG` verwendet. Damit die einzelnen Elemente einer Menge angezeigt werden, wird die `print()`-Funktion mit dem übergebenen Set aufgerufen. Die Ausgabe zeigt alle Elemente des Set `'B'`, `'D'`, `'A'`, `'C'`. Hierbei ist zu beachten, dass die Aussage abweichen kann. Der Hintergrund dafür ist, dass ein Set eine unsortierte Menge ist. Damit gilt, dass die Elemente in einem Set keiner Ordnung unterliegen, die Anzeige der Elemente somit zufällig erfolgt. Daher kann die Anzeige auf dem Computer des Lesers variieren. Im nächsten Schritt wird mit `SA ^ SB` geprüft, welche Elemente entweder in `SA` oder in `SB` sind. Das Ergebnis dieser Operation ergibt unter anderem `'E'`, `'C'`, `'B'`, `'F'`, `'G'`, `'A'`. In dieser Aufzählung fehlt das `D`. Dies liegt daran, dass das Element `D` in beiden Mengen zu finden ist. Mit der Operation `SA - SB` werden alle Elemente ausgegeben, die in `SA` und nicht in `SB` zu finden sind. Das führt zu dem Ergebnis `'A'`, `'B'`, `'C'`, denn alle andere Elemente sind auch Teil der Menge `SB`. Die letzte Operation `SA | SB` gibt als Rückgabe alle Elemente, die in `SA` oder `SB` zu finden sind. Ausgehend von dieser Operation wird `D` auf der Konsole ausgegeben, weil dieses Element das einzige ist, das in beiden Mengen vorhanden ist.

Ausgehend von diesen ersten Grundlagen zu Datenstrukturen ist zu erwähnen, dass in Python weitere Datenstrukturen, wie beispielsweise Sequenzen, unterschieden werden können. Diese werden jedoch in diesem Buch aus Gründen der Praktikabilität nicht weiter betrachtet. Mit Blick auf die Zielsetzung sind die bisher kurz vorgestellten Datenstrukturen für eine Vielzahl von Anwendungsfällen ausreichend.



Benjamin M. Abdel-Karim

Dieses Buch hat bisher die zentralen Grundlagen für die ersten Programmiererfahrungen mit primitiven Datentypen und Datenstrukturen gelegt. Der nächste logische Schritt besteht in der Einführung in die Kontrollstrukturen. Für das Schreiben geeigneter Skripte spielen Kontrollstrukturen eine entscheidende Rolle, um Quellcodes zu implementieren, die imstande sind, auf unterschiedliche Ereignisse zu reagieren und Fehlerfälle zu verhindern. Vor diesem Hintergrund ist die Auseinandersetzung mit Kontrollstrukturen sinnvoll. Zuerst befasst sich dieses Kapitel mit Verzweigungen (Abschn. 5.1). Ausgehend von den Verzweigungen folgt die Auseinandersetzung mit dem Themenfeld der Schleifen (Abschn. 5.2). Im Anschluss daran folgt die Darstellung der Try-Except-Bedingungen (Abschn. 5.3). Eine Besonderheit dieses Kapitels ist die Darstellung der Kontrollstrukturen mithilfe der Unified Modeling Language (UML)¹. Mithilfe der grafischen Darstellungen der UML sollen die Konzepte der Kontrollstrukturen veranschaulicht werden und das Verständnis erhöhen. Ausgehend von den Inhalten werden die folgenden Lernziele in diesem Kapitel behandelt:

- Einführung in das Konzept der Kontrollstrukturen
- Verständnis über die Funktionen von If-else-Bedingungen erhalten
- Schleifentypen kennenlernen und implementieren können

Ausgehend von den beschriebenen Lernzielen gibt die Abb. 5.1 zunächst einen Überblick über die relevanten Datenstrukturen, die für die Programmierung in Python relevant sein können.

¹ Die Unified Modeling Language ist eine formalisierte Sprache zur Erstellung von Diagrammen unter Bezugnahme einheitlicher Standards (Kecher, 2011, S. 13 ff.).

B. M. Abdel-Karim (✉)
Frankfurt am Main, Hessen, Deutschland
E-mail: BenjaminM.Abel-Karim@gmx.de

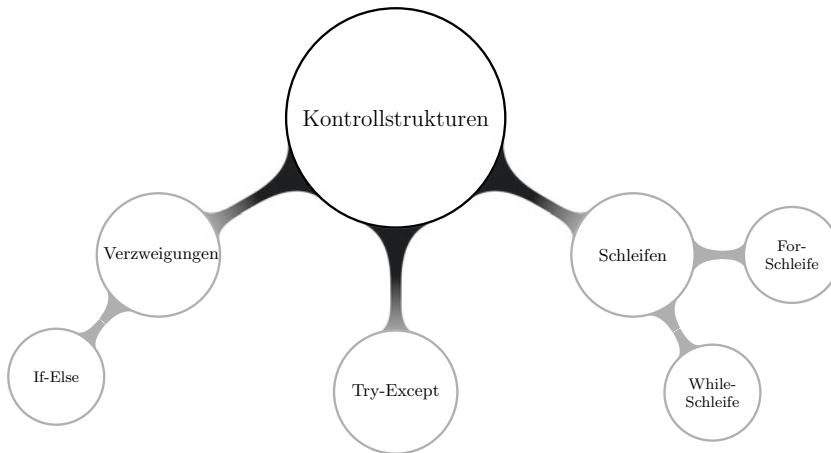


Abb. 5.1 Die Kontrollstrukturen in der Übersicht. Die Abbildung visualisiert ausgewählte Kontrollstrukturen

5.1 Verzweigungen

Die Verzweigungen in Python können definiert werden als Steuerelemente, die die Ausführung eines Programmzweigs in Abhängigkeit von einer Bedingung steuern. Je nachdem welche Bedingung erfüllt ist, wird der entsprechend definierte Programmzweig ausgeführt. Die simpelste Form dieser Verzweigung in Python ist die If-else-Bedingung.

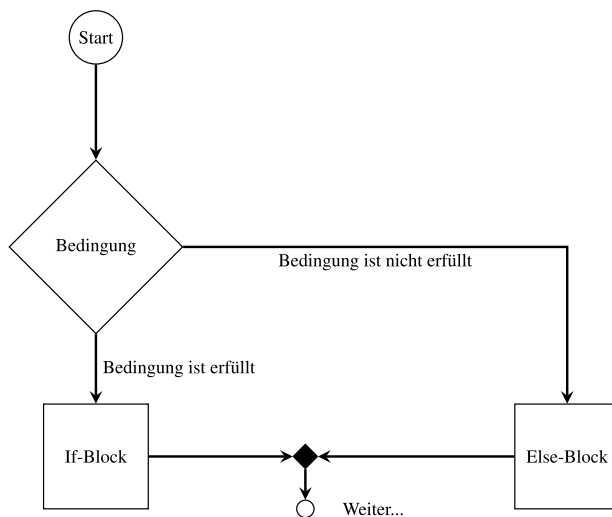
5.1.1 Die If-Else-Bedingungen

Im Allgemeinen überprüft die If-else-Bedingung, ob eine vorgegebene Bedingung erfüllt ist. Das Kernprinzip dieser Kontrollstruktur besteht darin, dass der sogenannte If-Block prüft, ob ein Quellcodeabschnitt eine Bedingung erfüllt und entsprechend die dazugehörigen Quellcodeabschnitte ausführt, sofern die Bedingung erfüllt ist. Trifft die Bedingung nicht zu, wird durch den Else-Block ein anderer Quellcodeabschnitt ausgeführt. Ferner werden zunächst zwei Zustände durch den If-Block unterschieden. Die Abb. 5.2 illustriert das grundlegende Funktionsprinzip der If-else-Kontrollstruktur.

Die Verwendung einer If-else-Bedingung kann in Python relativ einfach umgesetzt werden. Durch den Aufruf des Schlüsselworts `if` wird der If-Block in Python deklariert. Die an das Schlüsselwort angrenzenden Befehle werden damit zur Bedingungsprüfung. Der Abschluss der Bedingung in Python erfolgt durch zwei Doppelpunkte `:`. Durch die nächste und eingerückte Zeile wird der Programmblock definiert, der ausgeführt werden soll, wenn die If-Bedingung zutrifft. Der alternative Zweig wird durch das Schlüsselwort `else` in Python deklariert und muss entsprechend der Einrückungslogik auf derselben Ebene wie

Abb. 5.2 Die

If-else-Bedingung prüft, ob eine Bedingung zutrifft. Sofern die deklarierte Bedingung erfüllt ist, wird der nächste Programmabschnitt ausgeführt. Trifft die Bedingung nicht zu, wird ein anderer Programmabschnitt ausgeführt



der Else-Zweig stehen. Dabei wird `else` direkt mit einem Doppelpunkt abgeschlossen. Eine Einrückungsebene tiefer erfolgt dann der Codeabschnitt für den Else-Zweig. Das Vorgehen wird anhand des Codebeispiels 5.1 illustriert.

Listing 5.1 If-Else in Python

```

1  # Die If-else-Bedingung in Python.
2  # @author: Benjamin M. Abdel-Karim
3  # @since: 2020-04-25
4  # @version: 2020-04-25 V1
5
6  iA = 500
7  iB = 100
8
9  if iA > iB:
10     print('iA ist groesser als iB')
11 else:
12     print('iA ist nicht groesser als iB')
```

Im Codebeispiel 5.1 werden zunächst zwei Integer `iA` und `iB` angelegt. Beide Variablen werden jeweils mit zwei unterschiedlichen Integer-Werten initialisiert. Durch die nachfolgende `if`-Anweisung wird geprüft, ob die erste Variable `iA` größer ist als `iB`. Der Abschluss der If-Bedingung erfolgt durch die Doppelpunkte `:`, die zugleich den Anfang der Codezeile einleiten. Diese sollen ausgeführt werden, wenn die Bedingung zutrifft. Hierbei erfolgt die eingerückte Deklaration durch eine Konsolenausgabe mit der entsprechenden Aussage, dass `iA` größer ist als `iB`. Im alternativen Fall wird der Codeabschnitt nach der `else`-Deklaration ausgeführt. Das Ausführen des Quellcodes 5.1 führt zu folgendem Ergebnis: Die Variable

`iA` ist grösser als `iB`. Dieses nachvollziehbare Beispiel zeigt, wie einfach sich in Python eine If-else-Bedingung implementieren lässt.

Ausgehend von dieser ersten Verzweigung sollte an dieser Stelle kurz auf die möglichen Vergleichsoperatoren der Bedingungsprüfung eingegangen werden. In Python lassen sich zunächst, am Beispiel von zwei zu vergleichenden Bedingungen `A` und `B`, die folgenden Vergleichsoperatoren zur Bedingungsprüfung unterscheiden:

- `A == B` `A` und `B` sind gleich
- `A != B` `A` und `B` sind ungleich
- `A < B` `A` ist kleiner als `B`
- `A > B` `A` ist größer als `B`
- `A <= B` `A` ist kleiner oder gleich `B`
- `A >= B` `A` ist größer oder gleich `B`

Im Gegensatz zu einer Zuweisung mit einem Gleichheitszeichen (`=`) erfolgt die Bedingungsprüfung in Python (bis auf `<` und `>`) durch zwei Operatoren. Die Prüfung auf Gleichheit durch (`==`) prüft, ob zwei Bedingungen gleich sind. Die Notation (`!=`) prüft, ob zwei Bedingungen ungleich sind. Die Prüfung, ob eine Bedingungen gleich oder kleiner bzw. gleich oder größer ist, erfolgt durch das Hinzufügen des Gleichheitssymbol zu einem der Vergleichsoperatoren (`<`) bzw. (`>`).

Die Vergleichsoperatoren werten die Bindungen zu boolean, also den bereits kennengelernten Wahrheitswerten, aus. Dieser Umstand ermöglicht es, entsprechend der Aussagenlogik die Bedingungsprüfung beliebig komplex zu gestalten. Durch `and` oder `or` lassen sich mehrere Bedingungen verknüpfen. Im Codebeispiel 5.2 wird das Schlüsselwort `and` dazu genutzt, um zwei Bedingungen zu prüfen.

Listing 5.2 If-Else mit Konjunktion in Python

```

14 # Konjunktion von Bedingungen
15 if iA > 499 and iA < 999:
16     print('iA liegt im Wertebereich zwischen 499 und 999')
```

Durch die erste Bedingung im Codebeispiel 5.2 wird geprüft, ob `iA` größer ist als 499 und mit der zweiten Bedingung wird ermittelt, ob `iA` kleiner ist als 999. Durch die Konjunktion `and` müssen beide Bedingungen erfüllt sein, damit der Gesamtausdruck wahr ist. In diesem Fall trifft die Bedingung zu. Zur Erinnerung: Die Variable `iA` hat den Wert 500.

Eine Alternative stellt die Prüfung mehrerer Bedingungen dar, wobei das Erfüllen einer Bedingung ausreichen soll, um mit dem nächsten Codeabschnitt weiter zu machen. Dies kann mithilfe des `or` Operators bewerkstelligt werden. Dieser Operator prüft eine Abfolge von Bedingungen. Sobald einer der deklarierten Bedingungen erfüllt ist, wird der nächste Quellcodeabschnitt 5.3 ausgeführt.

Listing 5.3 If-Else mit Oder-Operation in Python

```
18 if iA == 499 or iA >= 500:  
19     print('iA ist entweder 499 oder groesser bzw. gleich 500')
```

Im Codebeispiel 5.3 wird zunächst geprüft, ob `iA` genau 499 ist. Danach erfolgt die Prüfung, ob `iA` größer oder gleich 500 ist. Offensichtlich ist dies der Fall. Damit wird die entsprechende Konsolenausgabe auf der Konsole ausgegeben.

Im Verlauf der Programmierung kann es ebenfalls erforderlich sein, die Bedingungsprüfung in sequenzieller Abfolge durchzuführen, sodass die Bedingungen in gewisser Weise verschachtelt werden müssen. Die Durchführung einer solchen Verschachtelung von Bedingungen mithilfe aufeinanderfolgender Bedingungsprüfungen zeigt das nächste Kapitel.

5.1.2 Verschachtelte If-else-Bedingungen

Im Kontext der If-else-Bedingungen können diese beliebig komplex verschachtelt werden. Dies gelingt, indem der Else-Zweig durch eine weitere If-Bedingung erweitert wird. Die Abb. 5.3 illustriert diesen Umstand.

In Python ist die Realisierung von verschachtelten If-else-Bedingungen mithilfe des `elif` Schlüsselworts möglich. Das Codebeispiel 5.4 illustriert den Einsatz des `elif` Schlüsselworts.

Listing 5.4 Verschachtelte If-else-Bedingung mit Oder-Operation in Python

```
1 # Die If-else-Bedingung in Python.  
2 # @author: Benjamin M. Abdel-Karim  
3 # @since: 2020-04-25  
4 # @version: 2020-04-25 V1  
5  
6 iA = 500  
7  
8  
9 if iA <= 100:  
10     print('iA ist kleiner gleich 100')  
11  
12 elif iA <= 200:  
13     print('iA ist kleiner gleich 200')  
14  
15 elif iA <= 300:  
16     print('iA ist kleiner gleich 300')  
17  
18 elif iA <= 400:  
19     print('iA ist kleiner gleich 400')  
20  
21 elif iA <= 500:  
22     print('iA ist kleiner gleich 500')
```

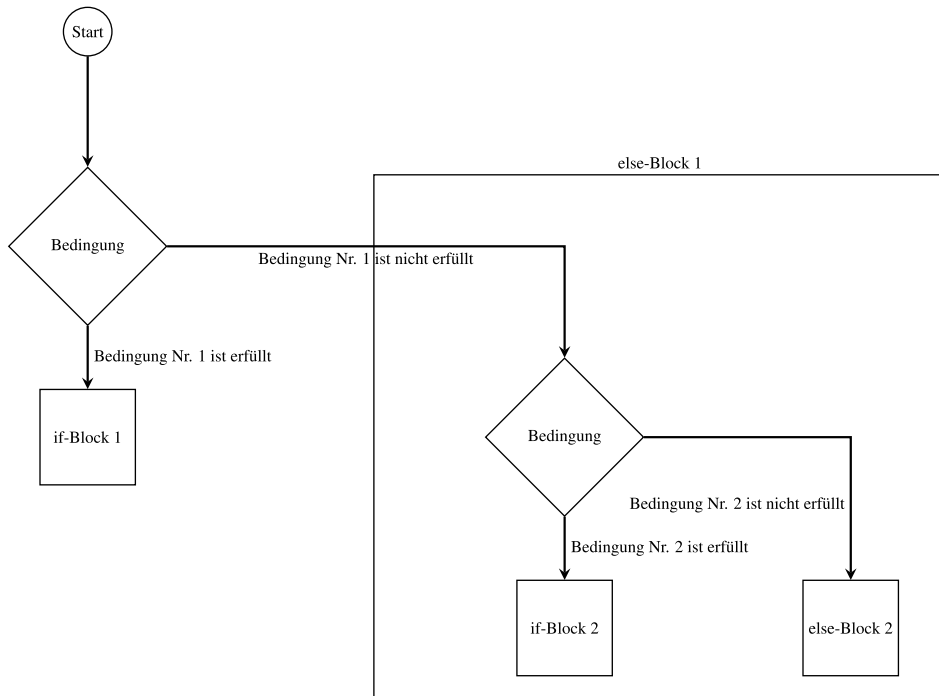


Abb. 5.3 Die verschachtelten If-else-Bedingungen. Zunächst erfolgt eine If-Bedingungsprüfung. Ist diese Bedingung erfüllt, erfolgt die Ausführung des nächsten Programmabschnitts. Ist diese Bedingung nicht erfüllt, wird im Else-Zweig eine weitere If-Bedingungsprüfung ausgeführt

Das neue Codebeispiel 5.4 initialisiert die Variable `iA` mit einem Integer, um sich speziell den Aspekt der verschachtelten Strukturen anzusehen. Anschließend wird die Variable zunächst darauf geprüft, ob sie kleiner gleich 100 ist. Sofern diese Bedingung nicht erfüllt ist, wird anschließend geprüft, ob die Variable kleiner gleich 200 ist. Diese Schachtelung wird bis zur Zahl 500 fortgesetzt. Der Else-Zweig wird dann ausgelöst, wenn die vorherigen Prüfungen nicht erfüllt worden sind. Sollte `iA` größer als 500 sein, wird keiner der Zweige ausgeführt, weil in keinem Fall einer der Bedingungen zutreffen würde. Durch den Einsatz verschachtelter If-else-Kontrollstrukturen lassen sich sequenziell verschiedene Fälle prüfen. Allerdings sollte der Einsatz von verschachtelten If-else-Konstruktionen wohl überlegt sein. Durch die Verschaltung steigt die Komplexität und damit das Fehlerrisiko. Der Hintergrund dafür ist, dass die Implementierung mit der Hinzunahme jeder weiteren Bedingungsprüfung unübersichtlicher und damit unleserlicher wird.

5.2 Schleifen

Im Rahmen der Programmierung werden die sogenannten Schleifen für die Wiederholung von Codezeilen eingesetzt, wobei die Anzahl der Wiederholungen von einer Bedingung abhängig ist (vgl. Bauer, 2009, S. 57). Vor diesem Hintergrund werden Schleifen aus zwei Komponenten zusammengesetzt. Dies gilt in der Regel unabhängig von der Programmiersprache. Der Schleifenkopf definiert die Anzahl der Wiederholungen auf Basis einer vorgegebenen Bedingung. Der Schleifenrumpf umfasst den Codeabschnitt, der entsprechend wiederholt werden soll. Solange die Bedingung im Schleifenkopf Gültigkeit hat, wird der Code im Schleifenrumpf ausgeführt. Auf Basis einer Bedingungsklassifikation lassen sich zwei grundlegende Schleifenformen unterscheiden: die For-Schleife und die While-Schleife. Diese beiden Formen werden in den folgenden Abschnitten behandelt.

5.2.1 Die For-Schleife

Die For-Schleife besteht dadurch, dass ihre Bedingung in einer vorab definierten Anzahl an Wiederholungen liegt. Das bedeutet, die Anzahl der Wiederholungen wird im Schleifenkopf festgelegt und der Quellcode solange im Schleifenrumpf ausgeführt, bis die vorgegebene Anzahl an Wiederholungen erreicht ist. Der Schleifenkopf definiert nicht nur die Anzahl der Wiederholungen, sondern stellt zusätzlich eine Variable bereit, die die aktuelle Anzahl an Wiederholungen der Schleife speichert. Diese Laufvariable (auch Schleifenindex genannt und nicht zu verwechseln mit dem Index der Listen) beinhaltet die aktuelle Position aus der Anzahl der Durchläufe. Die Abb. 5.4 illustriert das Prinzip der For-Schleife.

Zum besseren Verständnis einer For-Schleife soll das folgende Codebeispiel 5.5 herangezogen werden. Hierbei wird eine For-Schleife angelegt, die zehnmal den Code im Schleifenrumpf ausführen soll. Dabei wird die aktuelle Position des Laufindexes auf der Konsole ausgegeben. Die Anweisung `range(0, 10)` definiert den Indexbereich des Laufindex der For-Schleife. Vor dem Komma wird mit 0 der Startwert initialisiert, sodass die For-Schleife mit 0 startet. Nach dem Komma steht die Anzahl der Wiederholungen.

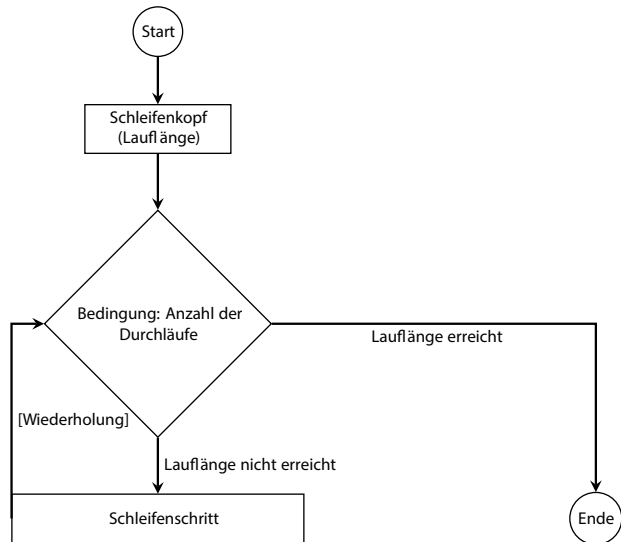
Listing 5.5 For-Schleife in Python

```
1 # Die For-Schleife in Python.  
2 # @author: Benjamin M. Abdel-Karim  
3 # @since: 2020-04-25  
4 # @version: 2020-04-25 V1  
5  
6 for iIndex in range(0, 10):  
7     print('Position der Schleife', iIndex)
```

Das Codebeispiel 5.5 zeigt die Umsetzung einer For-Schleife in Python. Hierbei wird mit dem Schlüsselwort `for` der Schleifenkopf deklariert. Dieser Schleifenkopf erwartet die Benennung des Laufindexes, in diesem Fall `iIndex`, sowie die Festlegung der Lauflänge.

Abb. 5.4 Die For-Schleife.

Der Schleifenkopf legt durch die Deklaration der Bedingung die Lauflänge fest. Solange die Bedingung Gültigkeit besitzt, wird der Quellcode im Schleifenrumpf ausgeführt. Sobald die maximale Lauflänge erreicht und damit die Bedingung nicht mehr erfüllt ist, wird die Schleife ihren Dienst einstellen und der nächste Quellcode außerhalb der For-Schleife ausgeführt



Die Lauflänge wird mithilfe der Funktion `range` festgelegt. In diesem Codebeispiel soll die Schleife zehnmal den Quellcode im Schleifenrumpf ausführen. Damit wird die Lauflänge von `0`, `10` definiert. Die Print-Funktion bekommt hierbei einmal einen String als Text übergeben sowie den Laufindex der Schleife. Der Auszug der Konsolenausgabe [5.6](#) zeigt das Ergebnis.

Listing 5.6 For-Schleifen-Ergebnis

```

1 >> Position der Schleife 0
2 >> Position der Schleife 1
3 >> Position der Schleife 2
4 >> Position der Schleife 3
5 >> Position der Schleife 4
6 >> Position der Schleife 5
7 >> Position der Schleife 6
8 >> Position der Schleife 7
9 >> Position der Schleife 8
10 >> Position der Schleife 9
  
```

In der Praxis wird die For-Schleife beispielsweise für das Iterieren über Listen genutzt. Das bedeutet, dass For-Schleifen Listen durchgehen, um auf die einzelnen Elemente in einer Liste zuzugreifen und Operationen auf ihnen auszuführen. Aufgrund der hohen Praxisrelevanz und des thematischen Bezugs wird das Vorgehen der Listeniteration an dieser Stelle kurz behandelt. Bei dem Vorgehen nutzt die For-Schleife das Vorhandensein des

Listenindexes, um mithilfe der Indexierung und des Laufindexes der For-Schleife auf die Elemente zuzugreifen. Das Codebeispiel 5.7 illustriert das Vorgehen der Listeniteration.

Listing 5.7 For-Schleife in Python

```
9  # Iterieren ueber Listen
10 LInteger = [1, 10, 100, 1000, 10000,
11             100000, 1000000, 10000000]
12
13 for iIndex in range(0, len(LInteger)):
14     print(LInteger[iIndex])
```

Das Codebeispiel 5.7 initialisiert zunächst eine Liste mit Integer-Werten. Anschließend wird der Schleifenkopf definiert, indem der `iIndex` als Laufindex deklariert wird. Die maximale Lauflänge wird hierbei durch die `range()`-Funktion festgelegt. Sie bekommt dabei den Startwert `0`. Als Ende wird mithilfe der Python eigenen `len()`-Funktion die Anzahl der Elemente in der Liste ermittelt und entsprechend läuft die For-Schleife so viele Wiederholungen durch, wie es Listenelemente gibt. Anschließend werden die Werte über die `print()`-Funktion an ihrer Indexposition in der Liste ausgegeben. Dies geschieht, indem der Laufindex `iIndex` der Liste in der `print()`-Funktion als Listenindex übergeben wird.

Listing 5.8 For-Schleifen-Ergebnis für Listeniteration

```
1  >>> 1
2  >>> 10
3  >>> 100
4  >>> 1000
5  >>> 10000
6  >>> 100000
7  >>> 1000000
8  >>> 10000000
```

Durch die Anwendung der For-Schleife lässt sich nun ihre Schönheit für Programmierer erkennen. Die For-Schleife ermöglicht es, Codes effizienter zu schreiben, weil sie die Wiederholungen durch eine kompakte Schreibweise ausführen kann. Dies spart im alltäglichen Programmieren Coderedundanzen und damit Zeit ein. Allerdings muss an dieser Stelle gesagt werden, dass Schleifen in anderen Programmiersprachen deutlich effizienter arbeiten und damit schneller sind. Hintergrund ist die dynamische Variablendeklaration. Die Schleife zeigt erst zur Laufzeit auf, um welche Datentypen es sich handelt. Damit muss für die dynamische Deklaration erheblich mehr Speicher allokiert werden. Jedoch muss ebenso erwähnt werden, dass dieser Geschwindigkeitsnachteil in Python angesichts der zunehmenden Rechenleistung von privaten Computern kaum ins Gewicht fallen dürfte.

5.2.2 Die While-Schleife

Die While-Schleife wird im Gegensatz zur For-Schleife nicht durch die Anzahl der Durchläufe bei ihrer Initialisierung determiniert. Das bedeutet, die While-Schleife beendet ihren Dienst erst dann, wenn die Bedingung im Schleifenkopf in Abhängigkeit zur Laufbedingung nicht mehr erfüllt wird. Das Besondere hierbei ist, dass die While-Bedingung im Schleifenkopf beliebig ausgestaltet werden kann. Aus diesem Umstand ergibt sich der Einsatz von While-Schleifen an Stellen im Code, die zwar wiederholt werden müssen, aber nur solange, bis ein Ereignis auftritt. Die Abb. 5.5 illustriert den Einsatz der While-Schleife in Python.

Das Codebeispiel 5.9 hilft beim Verständnis der Funktionsweise einer While-Schleife. In diesem Beispiel soll eine Zahl solange um einen Wert erhöht werden, solange sie kleiner als zehn ist. Entsprechend ist die Bedingung für die While-Schleife, dass sie solange weiterarbeiten soll, bis die gewünschte Zahl die Grenze (kleiner als zehn) erreicht hat.

Listing 5.9 While-Schleife in Python

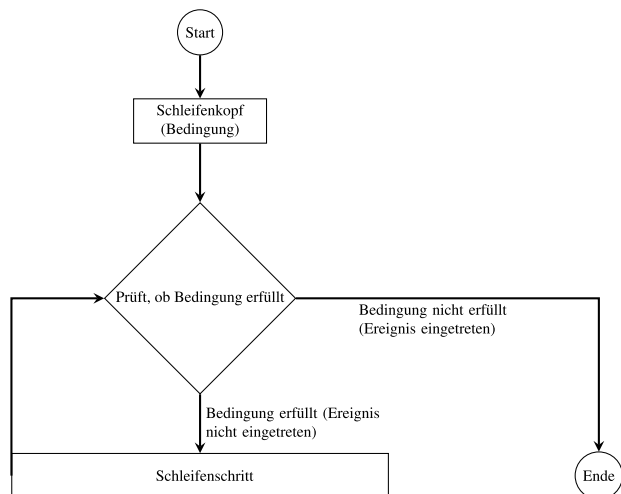
```

1  # Die While-Schleife in Python.
2  # @author: Benjamin M. Abdel-Karim
3  # @since: 2020-04-25
4  # @version: 2020-04-25 V1
5
6
7  iZahl = 1
8  while iZahl < 10:
9      print('Aktueller Zahlenwert:', iZahl)
10     iZahl += 1

```

Abb. 5.5 Die While-Schleife.

Der Schleifenkopf der While-Schleife enthält eine Bedingung. Solange diese erfüllt ist, wird die Schleife ihren Betrieb fortsetzen. Diese Bedingung wird bei jedem Schleifendurchlauf geprüft. Erst wenn die Bedingung ihre Gültigkeit verloren hat, stellt die Schleife ihren Betrieb ein, was dazu führt, dass der nächste Codeabschnitt außerhalb der Schleife ausgeführt wird



Für das Vorhaben wird im Codebeispiel 5.9 die zu Beginn initialisierte Variable `iZahl` solange um einen Wert erhöht, wie die Bedingung `iZahl < 10` erfüllt ist. Der aktuelle Zwischenschritt wird auf der Konsole ausgegeben. Die Konsolenausgabe 5.10 zeigt das Ergebnis. Hierbei ist zu beachten, dass ab `iZahl = 9` die While-Schleife den Zähler von 9 auf 10 erhöht, sodass bei der nächsten Prüfung die Schleife abbricht.

Listing 5.10 While-Schleifen-Ergebnis

```
1 >> Aktueller Zahlenwert: 1
2 >> Aktueller Zahlenwert: 2
3 >> Aktueller Zahlenwert: 3
4 >> Aktueller Zahlenwert: 4
5 >> Aktueller Zahlenwert: 5
6 >> Aktueller Zahlenwert: 6
7 >> Aktueller Zahlenwert: 7
8 >> Aktueller Zahlenwert: 8
9 >> Aktueller Zahlenwert: 9
```

5.3 Try-except-Bedingung

Eine zentraler Baustein in der Programmierung ist die Fehlerbehandlung. Hierbei kommen sogenannte Try-except-Kontrollstrukturen zum Einsatz. Die Deklaration von Try-Except ist ein passender Name für die Funktionsweise. Durch den Try-Block wird versucht, einen Codeabschnitt auszuführen. Sofern diese Ausführung misslingt, soll das Programm nicht stehen bleiben, wie es Python normalerweise vorsieht. Es soll stattdessen durch den Except-Block in eine andere Codezeile springen, um den Betrieb des Quellcodes zu sichern. Diese Kontrollstruktur ist insofern hilfreich, um Fehlerfälle abzufangen. Durch das Schreiben komplexer Programme steigt die Wahrscheinlichkeit potenzieller Ausnahmefälle (Exceptions), die zum Absturz des Programms führen. Der Einsatz von Try-Except kann somit aufkommende Fehlerfälle abfangen und versuchen, in einen anderen (stabilen) Codeabschnitt zu springen. Die Funktionsweise dieser Kontrollstruktur zeigt die Abb. 5.6.

Das folgende Codebeispiel 5.11 soll den Einsatz von Try-exception-Kontrollstrukturen veranschaulichen. In diesem Beispiel wird versucht, auf ein Listenelement in einer Liste zuzugreifen, das nicht existiert. Dies führt in Python zu einer `IndexError`-Exception, sodass das Programm aufgrund dieses Ausnahmезustands vorzeitig beendet werden sollte. Zur Beweisführung kann der Quellcodeabschnitt in Codezeile sieben kommentiert werden.

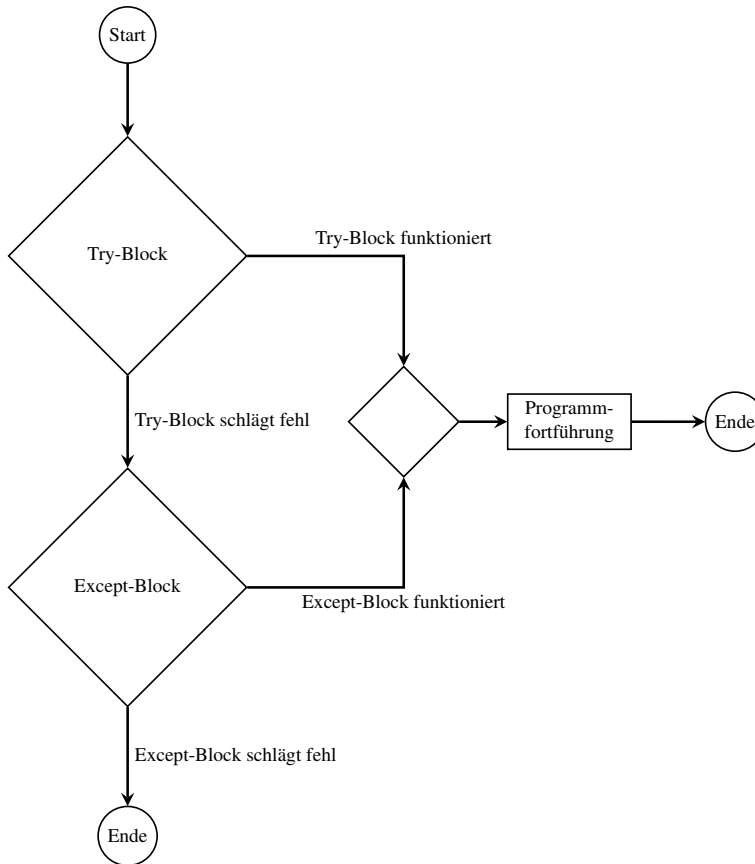


Abb. 5.6 Die Try-except-Kontrollstruktur. Zunächst wird Python versuchen, den Try-Block auszuführen. Dieser umfasst das Codesegment, das ausgeführt werden soll. Der Except-Block greift dann, wenn der Try-Block zu einem Ausnahmefall führt und der Code eigentlich abstürzen würde

Listing 5.11 Try-exception-Kontrollstruktur in Python

```

6 LNames = ['Marlene', 'Mareike', 'Nadine']
7 # Versuch des Zugriffs auf LNames[4]
8 try:
9     LNames[4]
10 except:
11     print('Die Liste hat kein Element an dem Index')
12
13 # Genauere Spezifikation
14 try:
15     LNames[4]
16 except IndexError as error:
17     print('Die Liste hat kein Element an dem Index')
  
```


Durch die Anwendung der Try-exception-Struktur im Codebeispiel 5.11 kann die Ausnahme verhindert werden. Zunächst versucht Python innerhalb des Quellcodes mit dem Indexzugriff auf Indexposition 4 in der Liste zuzugreifen, also auf das Listenelement an der Stelle 5 (ohne Einbezug der Null). Allerdings existiert kein Listenelement an dieser Stelle, weil die Liste nur die drei Elemente 'Marlene', 'Mareike', 'Nadine' besitzt. Python stellt diesen Umstand im Try-Block fest. Durch die Funktion `except` werden alle Ausnahmefälle abgefangen und es wird die Konsolenausgabe ausgeführt. Die `except`-Funktion lässt sich in Python genauer spezifizieren. Zum Beispiel führt die Verwendung von `except IndexError as error:` dazu, dass nur Fehler der Art `IndexError` `as error` abgefangen werden. Ein passendes Beispiel hierzu zeigt das Codebeispiel 5.12. Sollten andere Ausnahmefälle entstehen, führen diese zum Absturz des Programms.

Listing 5.12 Try-Exception mit genauer Exception-Spezifikation in Python

```
13 # Genauere Spezifikation
14 try:
15     LNames[4]
16 except IndexError as error:
17     print('Die Liste hat kein Element an dem Index')
```

Die dedizierte Spezifikation von Fehlern ist wahrscheinlich die sinnvollste Art und Weise, um seinen Quellcode zu schreiben. Die Verwendung von Try-exception-Kontrollstrukturen sollte mit Bedacht eingesetzt werden. Das Aufkeimen von Fehlern im Quellcode ist eine gute Möglichkeit, den eigenen Quellcode zu reparieren und auf seine Sinnhaftigkeit zu überprüfen. Zudem wäre es wahrscheinlich viel zu aufwendig, den Quellcode auf jeden erdenklichen Ausnahmefall vorzubereiten, statt diesen sorgsam zu schreiben. Die Verwendung sollte daher nur an kritischen Quellcodestellen erfolgen.



Benjamin M. Abdel-Karim

Python Funktionen realisieren die Möglichkeit, Codesegmente, die häufiger benötigt werden, zu modularisieren und über den gesamten Code hinweg zugänglich zu machen. Diese modularen Codesegmente können dabei als Eingaben verwendet werden, um gegebenenfalls anschließend eine Rückgabe zu liefern, die durch das modularisierte Codesegment generiert worden ist. In diesem Kapitel sollen zwei grundlegende Arten von Funktionen vorgestellt werden. Zum einen die sogenannten Built-in-Funktionen (Abschn. 6.1), also Funktionen, die Python bereits für den Nutzer bereitstellt, und Funktionen, die von dem Nutzer definiert werden (Abschn. 6.2). Zum anderen wird das Konzept der Bibliothek als Funktionssammlung in Python (Abschn. 6.3) vorgestellt. Somit ergeben sich die folgenden Lernziele für dieses Kapitel:

- Built-in-Funktionen kennenlernen
- Eigene Funktionen kennenlernen
- Bibliothek als Funktionssammlung verstehen

6.1 Built-in-Funktionen

Durch die Installation von Python auf dem eigenen Computer werden zusätzlich sogenannte Built-in-Funktionen für den Nutzer zugänglich gemacht. Diese Built-in-Funktionen führen Operationen im Code auf, sodass der Nutzer durch ihren Aufruf bequem Anweisungen ausführen kann. Beispielsweise ist die in diesem Buch verwendete `print()`-Funktion eine Built-in-Funktion. Sie erwartet als Inputvariable einen String und kann zusätzlich noch

B. M. Abdel-Karim (✉)
Frankfurt am Main, Deutschland
E-mail: BenjaminM.Abdel-Karim@gmx.de

Tab. 6.1 Auswahl einiger Built-in-Funktionen in Python

Built-in-Funktion	Input-Argument	Rückgabewert
<code>print()</code>	Beliebiger Wert	Ausgabe des Werts auf der Konsole
<code>len()</code>	Objekt	Gibt die Anzahl der Elemente zurück
<code>max()</code>	Objekt	Gibt das grösste Elemente im Objekt zurück
<code>min()</code>	Objekt	Gibt das kleinste Elemente im Objekt zurück
<code>type()</code>	Objekt	Gibt den Datentyp des Objekts zurück
<code>float()</code>	Int oder String	Konvertiert die Variable zu einem Float
<code>int()</code>	Float oder String	Konvertiert die Variable zu einem Int
<code>str()</code>	Int, Float	Konvertiert die Variable zu einem String

Variablen aufnehmen, um diese auf der Konsole auszugeben. Ebenfalls verfügen die Datenstrukturen über Built-in-Funktionen. Zum Beispiel hat dieses Buch die Aufnahme von neuen Elementen durch `append()` gezeigt. Die Tab. 6.1 zeigt einige nützliche Built-in-Funktionen in Python.

Grundsätzlich können Funktionen Eingabewerte erwarten. Diese werden in runden Klammern eingegeben und „Übergabe“ genannt. Dabei führt die Funktion aus dieser Eingabe eine vorher definierte Operation aus und liefert das Ergebnis beispielsweise in Form des sogenannten Rückgabewerts zurück.

6.2 Funktionen

In Python haben die Nutzer die Möglichkeit, eigene Funktionen zu definieren. Damit sind Funktionen Codesegmente, analog zum Skript, die zeilenweise vom Nutzer geschrieben werden. Dabei können sie in einer Extradatei (.py) oder im oberen Bereich des Skripts, in dem sie verwendet werden sollen, definiert werden. In Abgrenzung zu einem Skript ist eine Funktion jedoch nicht ohne die zentralen Informationen ausführbar. Klassischerweise erwarten selbstdefinierte Funktionen, wie die Built-in-Funktion, Inputvariablen, also Eingabewerte

in Form von Variablen, womit die Funktion Berechnungen ausführen kann. Innerhalb der Funktion werden die Inputvariablen entsprechend der Anweisungen des Nutzers verarbeitet und führen zu einer Outputvariable, dem Rückgabewert. Innerhalb der Funktion besteht eine abgeschlossene Welt. Alle Variablen, die innerhalb der Funktion definiert werden, existieren nur für die Funktion und innerhalb der Laufzeit. Damit werden alle Wertebelegungen der Funktion nach der Ausführung wieder gelöscht und stehen dem System nach Ausführung nicht mehr zur Verfügung. Ausgehend von diesen ersten Kenntnissen ergibt sich die Schlussfolgerung, dass Funktionen für den Nutzer eine erhebliche Erleichterung darstellen können. Im Unterschied zu einem Skript können Funktionen wiederholende Anweisungen kapseln und sparen damit Schreibarbeit. Eine Berechnung, die an mehreren Stellen in einem Skript ausgeführt werden müsste, kann so einmal im Skript als Funktion definiert werden. An den entsprechenden Stellen reicht der Funktionsaufruf aus, um die Berechnung auszuführen. Daraus ergibt sich, dass Funktionen eine weitere Abstraktionsebene bilden. Codesegmente können durch Funktionen abstrahiert werden, sodass sie erst zur Laufzeit mit konkreten Inputvariablen belegt werden müssen. In Abgrenzung hierzu sind Skripte Monolithen aus Codesegmenten.

Das Anlegen einer Funktion ist in Python mithilfe des Schlüsselworts `def` möglich. Hiermit wird Python signalisiert, eine Funktion zu definieren. Der sogenannte Funktionskopf (analog zum Schleifenkopf aus Abschn. 5.2) definiert den Funktionsnamen und ihre Inputvariablen. Vor dem Hintergrund, dass Python zunächst nicht zwischen Built-in-Funktionen und eigenen Funktionen unterscheidet, sollte darauf geachtet werden, dass die eigenen Funktionsnamen sich von den Built-in-Funktionen in Python unterscheiden. Hierzu empfiehlt sich der Verweis auf die Benennungskonventionen aus diesem Buch. Funktionen werden hier mit einem vorangestellten `f` deklariert, um zu signalisieren, dass es sich um eine Funktion handelt. Der Funktionsname ergibt sich aus der Aufgabe der Funktion und wird entsprechend der CamelCase-Notation an das `f` angeschlossen. Die Anzahl der Input-Parameter kann beliebig gewählt werden, sollte aber auf das Nötigste beschränkt werden, um die Komplexität möglichst gering zu halten. Der sogenannte Funktionsrumpf wird mit einem Doppelpunkt `:` eingeleitet. Durch die Einrückebene bildet Python die logische Abgrenzung zum restlichen Skript. Innerhalb des Funktionsrumpfs erfolgt die Implementierung der Funktionsaufgaben. Funktionen können einen Rückgabewert besitzen. Dieser wird in Python mit dem Schlüsselwort `return` benannt. Das Codebeispiel 6.1 zeigt die Implementierung einer Funktion, die eine gegebene Zahl mit sich selbst multiplizieren soll.

Listing 6.1 Eine eigene Funktion in Python

```
1  # Die Verwendung von Funktionen in Python.
2  # @author: Benjamin M. Abdel-Karim
3  # @since: 2020-04-24
4  # @version: 2020-04-24 V1
5
6  # Funktion zur Multiplikation einer Zahl mit sich selbst.
7  # @input: Ein int oder float
8  # @output: die Multiplikation
9  def fMultiplikator(dInput):
10     if isinstance(dInput, float) or isinstance(dInput, int):
11         dOutPut = dInput * dInput
12         return dOutPut
13     else:
14         raise ValueError('Falscher Datentyp der Inputvariable')
15     return dOutPut
16
17 dErgebnis = fMultiplikator(10)
18 print(dErgebnis)
```

Das Codebeispiel 6.1 zeigt die Implementierung der Funktion `fMultiplikator` als eigene Funktion in Python. Diese Funktion erwartet eine Inputvariable `dInput`. Aus der Quellcodedokumentation ergibt sich, dass diese Inputvariable ein Int oder Float sein soll. Allerdings macht die hier genannte Funktion sich die Eigenschaft der If-else-Bedingungsprüfung (Abschn. 5.1.1) zunutze, um den Ausnahmefall abzudecken. Ein solcher Fall wäre zum Beispiel ein Nutzer, der einen falschen Datentyp übergibt und bei der Ausführung des Codes einen entsprechenden Hinweis erhält. Dies gelingt durch die Verwendung der Datentypprüfung mithilfe der Built-in-Funktion `isinstance()`. Diese erwartet eine Inputvariable und den entsprechenden Datentyp. Sofern die Eingabe mit dem Datentyp übereinstimmt, gibt diese Funktion `True` zurück. Im anderen Fall wird `False` zurückgegeben. Die If-Bedingung erwartet ein `True`, um den Input mit sich selbst zu multiplizieren. Sollte der Rückgabewert von `isinstance()` jedoch `False` sein, wird der Else-Zweig ausgelöst und eine Fehlermeldung mithilfe von `raise ValueError('Falscher Datentyp der Inputvariable')` auf der Konsole für den Benutzer zurückgegeben, sodass er seine Eingabe überprüfen kann.

Der Aufruf der Funktion `fMultiplikator` gelingt, indem eine Variable für die Rückgabe definiert wird. In diesem Fall wird das `dErgebnis` über das Gleichheitszeichen als Variable zur Speicherung des Ergebnisses aus der Funktion `fMultiplikator` definiert. Die Funktion `fMultiplikator` bekommt als Inputvariable den Wert 10 übergeben. Dieses Beispiel zeigt zum einen, dass eigene Funktionen von weiteren Funktionen innerhalb des Funktionsrumpfs profitieren können, und zum anderen, wie in Python eigene Funktionen definiert werden können.

Allerdings ist das Schreiben von eigenen Funktionen in Python nicht immer erforderlich, weil Python aufgrund seines Open-Source-Gedankens zahlreiche sogenannte Bibliotheken zur Verfügung stellt, die im nächsten Abschnitt vorgestellt werden.

6.3 Bibliotheken (Module) in Python

Bibliotheken in Python stellen Sammlungen von Funktionen bereit. Einige dieser Bibliotheken bringt Python schon mit der Installation mit. Im Sprachgebrauch von Python wird in Abgrenzung zu anderen Programmiersprachen von Modulen gesprochen. Die Standardmodule in Python ermöglichen es, beispielsweise neue Ordner aus einem Skript heraus (`os modul`) oder ein Log-Datei (`log modul`) anzulegen. Von den Standardmodulen sind die sogenannten öffentlichen Module abzugrenzen. Hierbei handelt es sich um Module, die von anderen Nutzern verfasst und allgemein zur Verfügung gestellt werden. Vermutlich kann Python seinen Erfolg eben jenen zahlreichen kostenfreien Modulen verdanken, die von einer großen Anzahl von Nutzern verfasst wurden und anderen Nutzern mit entsprechendem Quellcode zur Verfügung gestellt werden.

Die Einbindung von Modulen in Python gelingt mit dem Schlüsselwort `import`, gefolgt vom Modulnamen. Module, die von Python bereitgestellt werden, müssen nicht extra heruntergeladen werden. Andere Module müssen in der Regel vorab durch Installation in das Verzeichnis von Python hinzugefügt werden. Moderne IDE wie PyCharm sind in der Lage, den Importversuch von gängigen Modulen zu erkennen und die Installation im Hintergrund durchzuführen. An dieser Stelle sei darauf verwiesen, dass entsprechende Module zumeist mit einer ausführlichen Anleitung aufwarten, sodass auf eben jene Anleitungen verwiesen sei. Eine ausführliche Darstellung sämtlicher Bibliotheken kann das Buch aufgrund der Zielsetzung nicht leisten.

Das folgende Codebeispiel 6.2 zeigt den exemplarischen Umgang mit Bibliotheken in Python. Im diesem Codebeispiel sollen Pseudozufallszahlen¹ erstellt werden. Die Erstellung der Zufallszahlen erfolgt dabei mithilfe einer For-Schleife. Innerhalb der Schleife soll erfasst werden, bei welchem Zug die Zahl von möglichen Pseudozufallszahlen im Wertebereich von 0 bis 10 gezogen wird. Wir wollen also wissen, wie oft die 5 innerhalb von 100 zufälligen (`random`) Wahlgängen gezogen wurde.

¹ Als Pseudozufallszahlen werden Zahlenreihen beschrieben, die in ihrer Verteilung Zufallszahlen aus natürlichen Prozessen ähneln. Ein solcher Prozess kann beispielsweise der Zerfall von Atomen sein. Der Unterschied zwischen natürlichen Zahlen und Pseudozufallszahlen ist, dass die Pseudozufallszahlen durch einen deterministischen Prozess entstanden sind. Vor diesem Hintergrund handelt es sich um keine echten Zufallszahlen, sondern um sogenannte Pseudozufallszahlen.

Listing 6.2 Nutzung des random-Moduls in Python

```
1  # Die Verwendung eines Moduls in Python.
2  # @author: Benjamin M. Abdel-Karim
3  # @since: 2020-04-24
4  # @version: 2020-04-24 V1
5
6  from random import *
7  # Erstellen von Pseudozufallszahlen
8  for iIndex in range(0, 100):
9      iRandom = randint(0, 10)
10     if iRandom == 5:
11         sOutput = 'In Runde' + ' ' + str(iIndex) + ' ' + 'wurde eine 5 gezogen'
```

Hierzu wird die Funktion `randint()` mithilfe des Standardmoduls `random` zugänglich gemacht. Durch den Befehl `from random import *` wird das Standardmodul in das aktuelle Skript importiert. Durch die Anweisung `*` im `Import`-Befehl werden alle Funktionen dieses Standardmoduls geladen. Alternativ kann die genaue Funktion spezifiziert werden. Die For-Schleife in diesem Skript zieht einhundertmal Pseudozufallszahlen in Form von Integer-Werten mithilfe der Funktion `randint()` aus dem Wertebereich von 0 bis 10. Sofern die gezogene Pseudozufallszahl eine 5 ist, wird eine Zeichenkette mithilfe der String-Konkatenation erstellt. Diese besteht aus den Zeichenkettenteilen `'In Runde'` und `'wurde eine fünf gezogen'` sowie den als String umgewandelten Laufindex der Schleife. Zusätzlich werden die Zeichenkettenteile durch leere Strings ergänzt, um den `sOutput` leserlicher zu gestalten. Der Einsatz von zusätzlichen Modulen wird vor dem Hintergrund von Data Science noch eine zentrale Rolle einnehmen. An dieser Stelle reichen jedoch erst einmal diese grundlegenden Informationen.

Teil II

Data Science



Benjamin M. Abdel-Karim

In diesem Kapitel wird auf den Begriff Data Science (Abschn. 7.1) und die dahinterstehenden Forschungsdisziplinen eingegangen. Hierbei werden zentrale Aspekte beleuchtet und eine Abgrenzung zu anderen Bereichen getroffen. Vor diesem Hintergrund wird der Data-Science-Prozess (Abschn. 7.2) vorgestellt, um ein systematisches Vorgehen für die eigenen Projekte darzustellen. Damit dieses Buch seinem übergeordneten Ziel, ein kleines und nützliches Handbuch zu sein, um alltägliche Aufgaben der Datenanalyse zu meistern, gerecht werden kann, wird zum Abschluss dieses Kapitels auf die konzeptionelle Gestaltung dieses Teils eingegangen (Abschn. 7.3). Daraus ergeben sich die folgenden Lernziele für dieses Kapitel:

- Einordnung Data Science
- Data-Science-Prozess kennenlernen
- Datensatz für die folgenden Kapitel kennenlernen

7.1 Einordnung Data Science

Zweifellos ist der Begriff Data Science allgegenwärtig. Zahlreiche Stellenanzeigen werben mit Stellen und suchen nach Persönlichkeiten mit entsprechenden Fähigkeiten. Allerdings ist zum Verständnis des Begriffs eine genauere Einordnung notwendig. Ein möglicher Ansatz liefert die Publikation von Donoho (2017). Hierbei kann Data Science als Wissenschaft der Datenanalyse begriffen werden, bei der es darum geht, aus Daten durch den Einsatz von modernen Technologien zu lernen (vgl. Donoho, 2017, S. 763).

B. M. Abdel-Karim (✉)
Frankfurt am Main, Hessen, Deutschland
E-mail: BenjaminM.Abdel-Karim@gmx.de

Auf der Suche nach dem Ursprung des Begriffs Data Science stößt der interessierte Leser auf unterschiedliche Aussagen hinsichtlich der Etymologie. Eine scharfe Eingrenzung des Entstehungszeitpunkts ist aufgrund der unterschiedlichen Aussagen in der Literatur, wie beispielsweise bei Cao (2017) oder Murtagh und Devlin (2018), nur schwer möglich. Fest steht allerdings, dass Data Science in seiner heutigen Form eine große Schnittmenge zu anderen Forschungsdisziplinen besitzt. In Bezug auf die Definition von Donoho (2017) steht die Wissensextraktion aus Daten mithilfe von moderner Technologie im Fokus. Vor diesem Hintergrund stellt Data Science die Verbindung aus mehreren Forschungsrichtungen im Allgemeinen dar. Im Speziellen greift dieses Forschungsfeld auf Methoden der Mathematik und Statistik zurück. Aufgrund der Relevanz für die praktische Anwendung sind ebenfalls die Perspektiven der Anwendungsfelder zu berücksichtigen. Ausgehend vom Einsatz moderner Technologien bedient sich Data Science entsprechend dem Feld der Informatik. In Bezug zu diesen drei übergeordneten Teildisziplinen bestehen in den meisten Data-Science-Vorhaben Schnittmengen mit anderen Forschungsdisziplinen, wie der Softwareentwicklung, den Themenbereichen des maschinellen Lernens und der Datenanalyse. Die Abb. 7.1 versucht eine Verortung entsprechend der angrenzenden Fachrichtungen.

Ausgehend von Abb. 7.1 kann die Verortung von Data Science im Rahmen einer wissenschaftlichen Disziplin-Betrachtung als Schnittmenge der drei großen Wissenschaften Informatik (Computer Science), Mathematik und Statistik (Mathematics and Statistics) sowie der jeweiligen Anwendungsdomain (Domain Knowledge) aufgefasst werden. Damit werden unweigerlich zentrale Forschungsgebiete dieser Schnittmengen angesprochen, wie beispielsweise die Datenanalyse (Data Analytics), die sich aus zentralen Methoden der Mathematik und Statistik in Kombination mit dem Domainverständnis der Zieldomain ergibt. Hinzu kommt im Speziellen das Feld des maschinellen Lernens als Teildisziplin, die ihre Verortung in der Informatik und der Mathematik findet. Die Umsetzung von Data-Science-Projekten bildet in der Regel die Grundlage für Softwareanwendungen, sodass auch die Softwareentwicklung (Software Engineering) zum Einsatz kommen kann.

Ausgehend von den vielseitigen Verortungen zu verschiedenen Forschungsdisziplinen ergibt sich der Umstand, dass verschiedene personelle Rollen innerhalb von Data Science vertreten sein können. Der Datenwissenschaftler (Data Scientist) ist in der Regel eine Person mit akademischem Hintergrund und versucht, mithilfe seiner wissenschaftlichen Kenntnisse passende Fragestellungen zu entwickeln und entsprechende Interpretationen der Ergebnisse vorzunehmen. Dem Gegenüber steht der Dateningenieur (Data Engineer), dessen Aufgabe die Bereitstellung der entsprechenden Daten ist. Hierbei umfasst sein Aktivitätsfeld sowohl die Gewinnung der Daten (Data Mining, Data Crawling and Data Acquisition) als auch die Vorbearbeitung der Daten, sodass diese für spätere Auswertungen geeignet sind. Im Zuge dessen hat der Umgang mit entsprechenden Daten eine zentrale Rolle. Der Datenanalyst (Data Analyst) ist imstande, die Daten zu analysieren und entsprechende Auswertung vorzunehmen. Hierbei kommen klassische statistische Auswertungen zum Einsatz. Ebenfalls spielen Visualisierungsverfahren und Dokumentationstechniken eine wichtige Rolle. Eine besondere Rolle nimmt hierbei der Experte für maschinelles Lernen ein. Seine Aufgabe

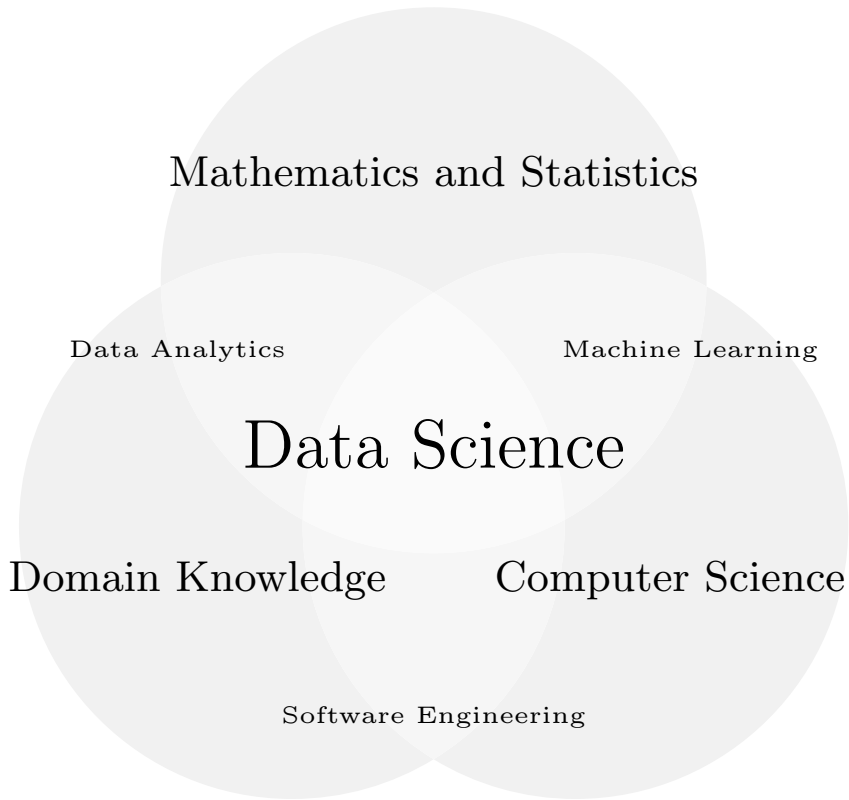


Abb. 7.1 Data Science – ein Versuch der Einordnung

besteht in der Regel darin, komplexe Modelle auf Basis der Daten zu trainieren, entsprechend zu validieren und zu testen. Diese Modelle können in der Praxis dazu eingesetzt werden, um Prognosen für die Zukunft zu erstellen. Im Kontext der praktischen Umsetzung ist zu antizipieren, dass die Rollenbilder nicht immer klar voneinander abgegrenzt werden können. Der Data Scientist beispielsweise kann für alle Aufgabenteile, wie die Beschaffung der Daten, Analyse und Visualisierung, verantwortlich sein. Vor diesem Hintergrund wundert es kaum, dass der aktuelle Blick in den Stellenmarkt zur Erkenntnis führt, dass zwar häufig ein Data Scientist in der Stellenausschreibung gesucht wird, jedoch andere Teilaufgaben vom Bewerber verlangt werden. Vermutlich liegt es in der Natur von Data Science, dass eine genaue Verortung kaum möglich ist. Allerdings lässt sich der Prozess von Data Science in genauere Phasen systematisieren. Diese Systematisierung ist Gegenstand des folgenden Abschnitts.

7.2 Data-Science-Prozess

Ausgehend von der Definition von Data Science und den implizit vorhanden Rollenbildern ergeben sich für eine praktische Umsetzung in Form eines Data-Science-Projekts zahlreiche Herausforderungen. Vor diesem Hintergrund ist eine gezielte Data-Science-Projektdurchführung, analog zu anderen Projekten, für den Erfolg notwendig. Dies gilt insbesondere dann, wenn das Vorhaben den typischen Spannungsfeldern aus Zeit, Kosten und Effizienz unterworfen ist. Daher ist es notwendig, die unterschiedlichen Teile eines Data-Science-Projekts in übergeordnete Projektteile zu gliedern, um so eine Systematik zu generieren. Dieser Projektprozess wird auch Data-Science-Prozess genannt. Ausgehend von der vorgestellten Definition wagt dieses Buch eine Gliederung, in der die Zielsetzung von Data Science, Erkenntnisse aus den Daten zu gewinnen, an den Anfang der Überlegungen gestellt wird. Damit steht zu Beginn jedes Vorhabens eine geeignete Fragestellung, um zielgerichtet die Daten zu analysieren. Ausgehend von der erarbeiteten Fragestellung ergibt sich die Beschaffung der Daten aus entsprechenden Datenquellen sowie deren Bereinigung. In der Praxis liegen Daten selten in der passenden Form bereit. Beispielsweise müssen die Datenbanken um Einträge mit fehlenden Werten bereinigt bzw. diese fehlenden Werte interpoliert werden. Diese Schritte können im Rahmen des Data-Science-Projekts einen erheblichen Teil an Zeit in Anspruch nehmen, weil der Umgang mit unvollständigen bzw. fehlerhaften Daten kein triviales Unterfangen ist. Sind die Herausforderungen der Datenvorverarbeitung (Data Preprocessing) gemeistert, folgt die Erkundung der Daten (Explore the data).

Dieser Analyseschritt dient dazu, ein besseres Verständnis über die bereinigten Daten zu bekommen. Hierbei eignen sich erste Visualisierungen und passende statistische Analysen. Entsprechend der eingangs entwickelten Fragen bietet der nächste Schritt ein passendes Modell für die Modellierung der Daten (Model the Data). Zumeist werden für diesen Schritt Modelle aus dem Bereich des maschinellen Lernens implementiert. Dieses Modellieren dient dazu, ein Modell bereitzustellen, das die Daten in Abhängigkeit der Fragestellungen schätzen oder Prognosen durchführen kann. Hierbei werden in der Regel die Modelle auf Teilen der bereinigten Originaldaten trainiert und validiert. Anschließend erfolgt der Test der Modelle auf einem für das jeweilige Modell unbekannten Teil der Datensätze. Abschließend folgt die Interpretation der Ergebnisse. Dieser Schritt geht in der praktischen Umsetzung mit der adressatenorientierten Kommunikation der Ergebnisse und deren Schlussfolgerungen einher. Diese einzelnen Projektschritte lassen sich in einer systematischen Prozessprojektstruktur zusammenfassen, sodass sich hieraus der Data-Science-Prozess ergibt. Die deduktive Betrachtung der Systematisierung des Data-Science-Prozesses zeigt die Abb. 7.2.

Aus Abb. 7.2 ergibt sich, dass der Data-Science-Prozess aus mehreren Teilschritten besteht. Jeder dieser Teilschritte umfasst seine eigenen spezifischen Aufgaben. Allerdings ist an dieser Stelle zu erwähnen, dass sowohl in der Forschung als auch in der Praxis diese Schritte zwar zunächst linear vollzogen werden, allerdings kann es notwendig sein, zu einem

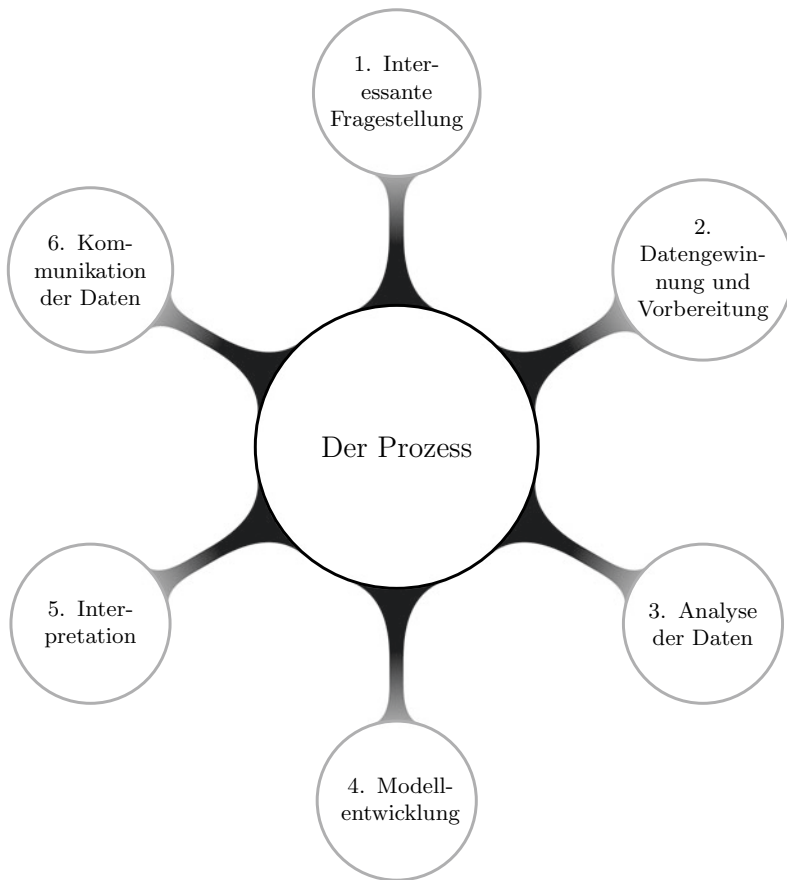


Abb. 7.2 Data-Science-Prozess

vorherigen Schritt zurückzukehren, um die Aktivitäten zu überdenken. Insofern darf die lineare Reihenfolge aus Gründen der Praktikabilität nicht zu streng ausgelegt werden.

7.3 Data-Science-Projekte für dieses Buch

Aus der Zielsetzung dieses Buchs ergibt sich der Sinnhaftigkeit wegen, das Feld von Data Science anhand von verschiedenen Projekten mit der Programmiersprache Python durchzuführen. Hierbei werden in den nachfolgenden Kapiteln verschiedene Vorhaben vorgestellt und mittels der Methoden aus dem Bereich von Data Science und seinen verwandten Teildisziplinen entlang des Data-Science-Prozesses bearbeitet, siehe Abschn. [7.2](#).

Damit richten sich die ausgewählten Projekte prinzipiell an interessierte Leser aus Forschung oder Praxis. Diese Projekte versuchen, ein möglichst großes Themenfeld abzudecken, in dem Data-Science-Projekte mit verschiedensten Datensätzen als Anschauungsbeispiele dienen. Grundsätzlich gilt, dass diese nachfolgenden Buchkapitel die Arbeitsweisen und Möglichkeiten mit Python möglichst anschaulich beschreiben sollen. Im Rahmen einer ersten Einführung setzt das Werk auf nachvollziehbare Problemstellung und Datensätze, sodass der Grad an Komplexität einen überschaubaren Rahmen haben sollte. In diesem Kontext liegt der Fokus darauf, essenzielle Handgriffe im Rahmen der Programmierung ausführlich zu beschreiben. Es soll erreicht werden, dass der Leser für eine Problemabstraktion sensibilisiert wird und die Lösungswege auf andere Sachverhalte übertragen kann. Der Einsatz von Python wird als Instrument zur Datenanalyse im Umgang mit den Herausforderungen für Forschung und Praxis dargestellt.

Ausgehend vom Anspruch dieses Buchs werden die folgenden Projekte innerhalb der einzelnen Kapitel entlang des Data-Science-Prozesses strukturiert und die Arbeitspakete Schritt für Schritt erklärt. Auf den Aspekt der Datengewinnung kann in diesem Buch nur am Rande eingegangen werden. Dieser Umstand ergibt sich aus lizenzrechtlichen Aspekten sowie der teilweise verbundenen Komplexität, die mit der Datengewinnung (Data Mining) einhergeht. Damit setzt das Buch auf eigene Datensätze, die für die Zwecke dieses Buchs bereitgestellt werden.



Benjamin M. Abdel-Karim

Ausgehend von den Grundlagen von Data Science im Kap. 7 fokussiert sich dieses Kapitel auf einen Teil des Data-Science-Prozesses (Abschn. 7.2). Das maschinelle Lernen wird Hand in Hand mit Data Science erwähnt, weil es eine zentrale Komponente des Data-Science-Prozesses ist. Vor diesem Hintergrund bildet dieses Kapitel eine Kurzeinführung in das Themenfeld des maschinellen Lernens. Lernziele dieses Kapitels sind:

- Definition des maschinellen Lernens
- Einordnung des maschinellen Lernens

8.1 Definitionen des maschinellen Lernens

Es gibt verschiedene Definitionen, die das Forschungsgebiet und den Begriff künstliche Intelligenz (KI) abdecken. Im Allgemeinen kann sie als die Wissenschaft beschrieben werden, die sich mit intelligenten Agenten befasst, die sich Ziele gesetzt haben und mit ihrer Umgebung interagieren, indem sie ihre Sensoren und Aktoren einsetzen, um diese Ziele zu erreichen (Russell & Norvig, 2016). Durch den Einsatz ihrer Sensoren nehmen diese Agenten die Umwelt wahr und aktualisieren ihren inneren Zustand, ähnlich wie Menschen ihre Organe benutzen, um ihre Umwelt wahrzunehmen. Im Gegenzug nutzen die künstlichen Agenten ihre Aktoren, um einen Einfluss auf ihre Umwelt auszuüben (Russell & Norvig, 2016).

In den letzten 70 Jahren hat die Forschungsdisziplin der KI verschiedene Methoden und Forschungsrichtungen entwickelt, die sich mit der KI in funktionellen Bereichen befassen.

B. M. Abdel-Karim (✉)
Frankfurt am Main, Hessen, Deutschland
E-mail: BenjaminM.Abdel-Karim@gmx.de

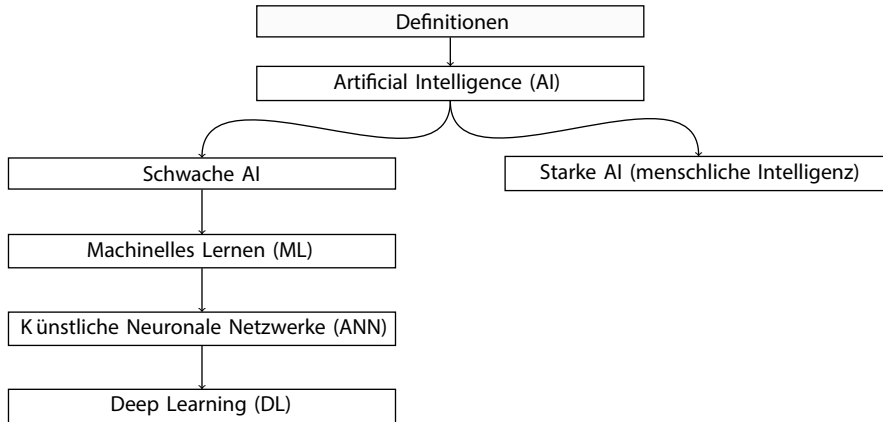


Abb. 8.1 Definitionsbaum

Da das Umfeld, in dem der Mensch agiert, eine Fülle von verschiedenen Einflussfaktoren sowie Daten mit unterschiedlichen und sich verändernden Eigenschaften enthält, hat die Forschung das Umfeld der KI-Algorithmen so eingegrenzt, dass sie in einem bestimmten Umfeld eine bestimmte Aufgabe übernehmen. Daher wird die KI in der aktuellen Forschung manchmal in zwei Kategorien eingeteilt: starke KI und schwache KI (Wayne & Pasternack, 2011). Die Abb. 8.1 gibt einen Definitionsüberblick.

Starke KI steht für eine Form der maschinellen Intelligenz, die heute als Utopie gelten würde, da sie der menschlichen Intelligenz gleichgestellt werden müsste. Daraus lässt sich ableiten, dass alle existierenden Ansätze mit KI als schwache KI klassifiziert werden können. Diese Unterscheidung ist jedoch nicht ganz unumstritten. Aus Sicht der Forschung stellt die starke KI nur einen theoretischen Bezugspunkt dar. Dies macht es schwierig, eine klar definierte Linie zu ziehen. Eine Alternative ist die Unterscheidung zwischen allgemeiner KI und funktionaler KI. Wie der aktuelle Stand der Forschung nahe legt, zeichnet sich maschinelles Lernen (ML) durch seinen statistischen Ansatz zur Problemlösung in diesen engen, funktionalen Bereichen aus.

Der Bereich des ML befasst sich mit der Entwicklung und Erforschung von lernfähigen computergestützten Algorithmen zur Verbesserung ihrer Aufgabenleistung (Jordan & Mitchell, 2015; Mjolsness & DeCoste, 2001). Es wurde eine Vielzahl von Algorithmen im Bereich ML entwickelt, um verschiedene Problemtypen abzudecken, die durch die Formalisierung von Problemen aus der realen Welt in rechnergestützte Probleme aufgedeckt wurden (Jordan & Mitchell, 2015; Shafique & Qaiser, 2014). Dieser Umstand führt zu dem Problem, dass selbst die Auswahl geeigneter Modelle ein intimes Verständnis der Daten und Kenntnisse über die verfügbaren ML-Werkzeuge, ihrer Eigenschaften und der Berechnungsprobleme, die sie (annähernd) lösen können, erfordert (Reicha & Barai, 1999). Ein Anwender von ML-Methoden benötigt daher zunächst einen Überblick über die zentralen

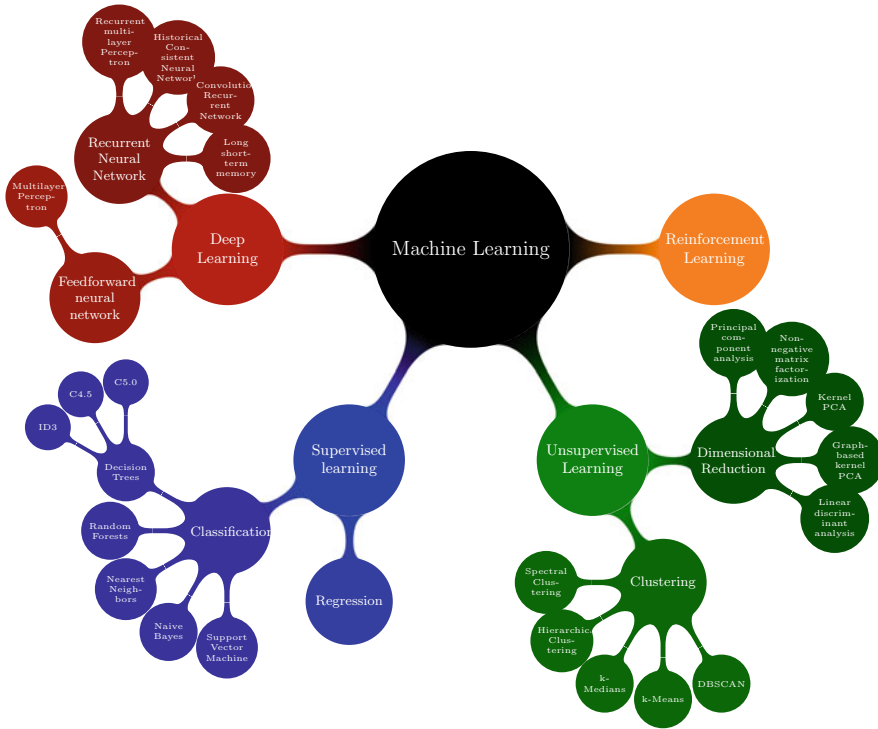


Abb. 8.2 Maschinelles Lernen – Taxonomie

Modelle des ML. Die Abb. 8.2 stellt zunächst eine Auswahl an möglichen ML-Modellen in Formen einer Taxonomie dar.

Im Allgemeinen kann ML in zwei hochrangige Kategorien eingeteilt werden: überwachtes Lernen und unbeaufsichtigtes Lernen. Die Abb. 8.2 zeigt eine Taxonomie der Methoden des ML und entsprechende Beispiele von Algorithmen. Das überwachte Lernen vergleicht seine Ergebnisse mit den korrekten Ergebnissen während der Ausbildung (Hutson, 2017). Modelle des überwachten Lernens werden im Allgemeinen auf etikettierten Daten trainiert und machen statistische Rückschlüsse auf Grundlage eines trainierten Modells. In dieser Hinsicht können beaufsichtigte Lernmodelle der Klassifizierung oder der Vorhersage dienen.

Im Bereich des unüberwachten Lernens lassen sich beispielsweise die folgenden ML-Modelle anführen: 1) DBScan oder formales Density-based Spatial Clustering von Anwendungen mit Rauschen wird von Ester et al. (1996) vorgeschlagen. Als nicht-parametrisches Verfahren versucht dieses Modell, Gruppen von Datenpunkten herauszufinden. 2) Hierarchisches Clustering ist ein Modell, bei dem Daten durch den Aufbau einer Hierarchie von Clustern geclustert werden. 3) k-means ist ein Modell, bei dem eine vordefinierte Anzahl von Klassen aus einer Menge ähnlicher Datenpunkte gebildet wird, die durch die Arbeit

von Lloyd (1982) entwickelt wurde. 4) k-medians ist eine Variante des k-means Clustering, die auf Lloyd (1982) basiert. 5) Spectral Clustering transformiert das Datenobjekt als Knoten eines Graphen. Die Abstände (Unähnlichkeiten) zwischen den Objekten werden durch die gewichteten Kanten zwischen den Knoten des Graphen dargestellt (Donath & Hoffman, 1973). In der Unterkategorie der Dimensionalitätsreduktion finden wir die folgenden Modelle: 1) Die lineare Diskriminanzanalyse ist ein Modell, das jeder Beobachtung in einer Diskriminanzanalyse einen Score-Wert zuordnet. Der Score-Wert wird für die Berechnung der Gruppe für jede Beobachtung verwendet (Fisher, 1973). 2) Nicht-Negativ-Matrix oder mehr formale Nicht-Negativ-Matrix-Faktorisierung (NMF) ist eine Technik zur Reduktion der Dimensionalität, die an das Clustering angepasst ist (Aggarwal, 2015). 3) Das Hauptkomponentenanalysemodellverfahren transformiert Datensätze durch Vereinfachung der Zählung von Variablen mit mathematischer Transformation, die das Ziel der Linearkombination (Hauptkomponenten) haben (Pearson, 1901). 4) Kernel Principal Component Analysis (PCA) ist eine Erweiterung der Hauptkomponentenanalyse.

8.2 Herausforderungen des Maschinellen Lernens im Kontext von Data Science

Der Einsatz von modernen Verfahren aus dem Bereich des ML, wie in Abschn. 8.1 vorgestellt, geht in der Regel mit zahlreichen Herausforderungen einher. Daher ist die Zielsetzung dieses Abschnitts, die Leser für diese Herausforderungen zu sensibilisieren und ihr Verständnis zu erhöhen. Am Anfang stehen die Daten, sodass mit ihnen die ersten Herausforderungen für Nutzer, Programmierer und Entscheidungsträger auftreten. Bei der Verwendung von ML sind riesige Datensätze sinnvoll. Die Arbeit von Baier et al. (2019) hat herausgefunden, dass die Datenqualität für die meisten ML-Nutzer sowohl in der Praxis als auch in der Wissenschaft eine Herausforderung darstellt (Lee-Post & Pakath, 2019). Im Allgemeinen wird die Datenqualität vor Beginn eines neuen Projekts überprüft. Aber die Analyse der Daten ist ein aufwendiger Prozess, weshalb eine realistische Verifizierung möglich ist, wenn das ML-Projekt läuft. Die Verifizierung von Daten ist jedoch ohne Domänenkenntnisse schwierig (Baier et al., 2019). Die Datenqualität wirft zudem andere Fragen auf, die neben der Anwendungsebene auch auf die Ebenen der Zugriffsrechte und der organisatorischen Verantwortung führen. Wie können beispielsweise Daten in einer Multistakeholderumgebung sicher ausgetauscht werden? Wer sind die Entscheidungsträger in Bezug auf die Daten in einer Organisation? All dies sind Fragen, die in Zukunft durch weitere Forschung und durch Entscheider in den Unternehmungen geklärt werden müssen.

Im Kontext der Datenvorbereitung entstehen die nächsten Herausforderungen. ML-Methoden stützen sich stark auf qualitativ hochwertige Trainingsdaten. Sie hängen auch von den Eigenschaften, der Struktur und der Komplexität der Datenproben ab (Piri et al., 2018, S. 23). Auch der Zeitstempel der Datenerhebung könnte einen Einfluss auf die Ergebnisse haben, beispielsweise durch fehlende oder unvollständige Daten. Aus diesem Grund

müssen Daten gescreent, bereinigt und vorverarbeitet werden, bevor ML-Algorithmen diese Daten als Input verwenden können (Basti et al., 2015, S. 22). Ein Hauptproblem, das sich aus der Datenbereinigung ergibt, ist die Möglichkeit, wichtige Informationen durch Entfernen bestimmter Teile der Daten zu verlieren (Rittgen et al., 2009). Infolgedessen könnten die aus Data-Mining-Modellen generierten Variablen zu Verzerrungen und Fehlklassifikationen führen (Yang et al., 2018, S. 4). Daher ist es wichtig zu erkennen, dass es gängige Praxis ist, etwa 80 % von Arbeit und Zeit der Datenaufbereitung zu widmen, wie eine repräsentative Umfrage eines populärwissenschaftlichen Datenportals ergab¹. Dieser Umstand zeigt, dass es großes Potenzial für weitere Forschung im Bereich der Datenaufbereitung gibt, um den Zeitaufwand der Datenaufbereitung zu reduzieren. Dieser Umstand hängt mit der Qualität der Daten zusammen.

Darüber hinaus ist die Stichprobengröße für ML-Algorithmen ebenso wichtig wie für traditionelle quantitative Analysemethoden, da die Genauigkeit von ML-Algorithmen in den meisten Fällen nur durch Training an großen Stichprobendatensätzen gewährleistet werden kann (Goodhue et al., 2012, S. 983). Da die verfügbaren Daten nicht nur zum Training des Algorithmus, sondern auch zum Testen und zur Validierung des Algorithmus verwendet werden müssen, scheint es eine ebenso wichtige Aufgabe zu sein, die verfügbaren Stichprobendaten in Trainings- und Testsätze aufzuteilen.

Die Komplexität nimmt weiter zu, wenn es um textuelle Analysen oder andere Aufgaben des Neuro-Linguistischen Programmierens (NLP) geht, was das Problem der unstrukturierten textuellen Stichprobendaten aufwirft (Wang et al., 2015, S. 90). Im Fall von NLP-Aufgaben müssen die Daten in der Vorverarbeitungsphase gründlich bereinigt werden. Beispielsweise sollten Satzzeichen, Zahlen und abstrakte Strukturen wie Hyperlinks vor der eigentlichen Analyse entfernt werden, da Hyperlinks und Satzzeichen selbst keine neuen Informationen hinzufügen, aber dennoch Rauschen erzeugen können (See-To & Yang, 2017). Außerdem müssen die Daten kontinuierlich analysiert werden, um zu verstehen, welche Implikationen die Ergebnisse einer Analyse im Allgemeinen haben können (Cnudde & Martens, 2015, S. 83). Im Widerspruch zu diesem Ansatz steht die Tatsache, dass viele reale Geschäftsdaten, die im Rahmen von Forschungsstudien zur Verfügung gestellt und analysiert werden, nur in verschlüsselter Form abgerufen werden können, sodass die Personen, die diese Daten auswerten sollen, kein tieferes Verständnis der Daten entwickeln, aber ihre Algorithmen damit füttern können (Martens & Provost, 2014, S. 884).

Es ist auch zu beachten, dass der Prozess der Datenaufbereitung für einen Gutachter äußerst schwierig zu überprüfen ist. Dieser Umstand schafft zusätzliche Unsicherheit, was wiederum die Wahrscheinlichkeit einer Ablehnung erhöhen kann.

Ein zusätzlicher Aspekt im Zusammenhang mit der Datenaufbereitung ist der Zugang zu Quellen mit relevanten und interessanten Daten, was die hohe Anzahl von Publikationen mit Industriekooperationen erklärt. Beispiele hierfür sind Kooperationen mit Unternehmen wie Google, Deloitte Consulting und IBM (Lozano et al., 2017; Fu et al., 2017; Pai et al., 2014).

¹ <https://bit.ly/2WwVPho>

Die Verwendung von ML-Algorithmen erfordert häufig eine sorgfältige Abstimmung der Lernparameter und Modellhyperparameter (Snoek et al., 2012, S. 2951). In Zusammenhang mit der Aufgabe der Datenaufbereitung steht die Herausforderung der Modellwahl. Einige Wissenschaftler haben diese Frage in ihren Arbeiten untersucht (siehe Gao et al., 2017; Evermann et al., 2017). Ob unüberwachte oder überwachte Lernmethoden gewählt werden, hängt von der Problemumgebung sowie von den Eigenschaften der verfügbaren Beispieldaten ab (Lau et al., 2012, S. 1245).

Nur wenige Arbeiten verwenden eine Kombination oder Schichtung mehrerer Algorithmen, um die Robustheit ihrer Ergebnisse zu testen (Martens et al., 2016, S. 75). Dieser Ansatz kann für solche Fälle nützlich und angemessen sein, in denen Daten in verschiedenen Phasen verarbeitet werden müssen, um aus den Daten Wissen abzuleiten. Dies impliziert eine höhere Komplexität, was ein Grund dafür ist, dass ein solcher Multimethodenansatz nur selten verwendet wird.

Ein wichtiges Problem der Modelloptimierung ist die Gefahr der Überanpassung im Zusammenhang mit prädiktiven Modellen, was zu überoptimistischen Ergebnissen führt (Siering et al., 2018). Um das Problem der Überanpassung zu verringern, kann die Wahl eines einfachen Modells eine mögliche Lösung sein. Gelehrte und Praktiker scheinen gleichermaßen der Meinung zu sein, dass die Vorhersagekraft und Robustheit eines ML-Modells umso höher sein kann, je komplexer es ist. Dennoch kann dies ein Trugschluss sein, wie von Cresci et al. (2015) hervorgehoben wird. Es scheint in der Tat so zu sein, dass die Wahrscheinlichkeit einer Überanpassung umso größer ist, je komplexer ein Modell ist. Dies sollte uns dazu veranlassen, dieses allgemeine Paradigma komplexer Architekturen zu überdenken oder uns vielleicht einfacheren, aber wirksameren Modellen zuzuwenden Cresci et al. (2015).

Mit Blick auf die zahlreichen ML-Methoden hängt die Wahl der ML-Lernmethode und des ML-Algorithmus nicht nur von der Problemumgebung, sondern auch von den Eigenschaften und der Struktur der verfügbaren Daten ab. Beispielsweise können die verfügbaren Daten entweder numerisch oder alphanumerisch, diskret oder kontinuierlich, strukturiert oder unstrukturiert sein. Ein weiterer wichtiger Punkt ist die Wahl der richtigen Modellparametrisierung – z. B. die Wahl des richtigen k für ein Modell des k -Mittelwert-Algorithmus oder die Wahl der Anzahl der Schichten, Iterationen und der Batch-Größe der Trainingsdaten für ein ANN (Li et al., 2017, S. 83). Die Parametrisierung kann die Genauigkeit und Erklärungskraft der Modelle stark beeinflussen ((Walczak & Velanovich, 2018, S. 117). Darüber hinaus ist auch die Auswahl der Trainingsdaten eine schwierige Aufgabe, für die Verfahren wie die k -fache Kreuzvalidierung versuchen, das geeignete Trainingsset (Topuz et al., 2018; Singh & Tucker, 2017) oder die Auswahl der richtigen Parameter für Support Vector Machines aus der verfügbaren Datenstichprobe zu berechnen (Huang et al., 2016, S. 22).

Die Modelloptimierung und Parametrisierung von ML-Modellen ist daher eine sehr anspruchsvolle Aufgabe. Darüber hinaus müssen Wissenschaftler und Manager weitere Probleme lösen, wenn sie ML-Methoden anwenden wollen. Diese Herausforderungen sind auf die mangelnde Standardisierung im Sinn von theoretischen Richtlinien für die Anwendung

von ML-Methoden auf ökonomische Daten zurückzuführen. Mögliche Fragen, die sich stellen können, sind unter anderem: Wie sollen die Daten aufbereitet und verarbeitet werden? Welche Methoden sind für die Analyse der zugrundeliegenden Daten am erfolgversprechendsten? Wie sollten die Parameter gesetzt werden, damit das Modell dauerhaft überprüfbare und robuste Ergebnisse erzielt? Wissenschaftler und Praktiker sollten gleichermaßen ermutigt werden, nicht nur ihre Problem im Fall der Anwendung von ML-Methoden gründlich zu analysieren, sondern auch Robustheitsanalysen durchzuführen und ihre Ansätze in Bezug auf die Parametrisierung ihrer Modelle offenzulegen, um die Transparenz und das Vertrauen in die Zuverlässigkeit und Robustheit ihrer Analysen zu erhöhen.

Eine der größten Herausforderungen für die groß angelegte Einführung von ML in weiteren Lebensbereichen ist das Blackbox-Phänomen der meisten Methoden zur Laufzeit (Müller et al., 2016, S. 294). ML-Methoden, insbesondere Künstliche Neuronale Netzwerke (KNN), werden oft als Blackbox-Lösungen verstanden und etikettiert, weil sie wenig erklärende Einblicke in den Einfluss der unabhängigen Variablen geben. Darüber hinaus kann eine Frage sein, warum die Berechnungen innerhalb des KNN zu bestimmten Ergebnissen führen. Mit dem Ziel, zu verstehen, warum bestimmte Ergebnisse gegeben sind, kann es erforderlich sein, bei der Anwendung von ML-Methoden tief in die Trainingsalgorithmen der ML-Algorithmen selbst einzutauchen. Vor allem mit dem Aufkommen hochgradig nicht-linearer komplexer Inferenzsysteme in Form von Deep Learning (DL) wurde dieses Problem für Wissenschaftler vieler Disziplinen offensichtlich, sodass die Aufgabe, die Blackbox zu entschlüsseln, zentraler Forschungsgegenstand ist.

Teil III

Produktanalyse



Anwendungsbeispiel: Meine besten Videospiele

9

Benjamin M. Abdel-Karim

Der Erfolg des Verkaufs von Videospielen zeigt, dass Videospiele keine Randprodukte mehr sind. Durch den Einzug moderner Spieltechnologien, wie beispielsweise Googles neuer Cloud-Gaming-Plattform „Google Stadia“, Apples „Apple Arcade“ oder „Steam“ werden Videospiele für die unterschiedlichen Zielgruppen interessant. Dieser Umstand wird durch den Erfolg der Aktienkurse der größten, an der Börse notierten Spieleproduzenten, wie beispielsweise Activision Blizzard, Nintendo oder Ubisoft belegt. Zudem profitieren Firmen wie Nvidia und AMD durch die Produktion von entsprechender Hardware ebenfalls von dem Trend. Allerdings stellt sich auch Investoren und Entwicklern die Frage, welche Spieletitel und welche Genres in der Vergangenheit erfolgreich waren, um daraus Schlussfolgerungen für die Zukunft ableiten zu können. Eine solche erste Analyse wird der Gegenstand dieses Kapitels. Hierbei werden die im vorangegangenen (Abschn. 7.2) vorgestellten Schritte des Data-Science-Processes durchlaufen. Über die Vorstellung des Datensatzes (Abschn. 9.1) und den abgeleiteten Fragestellungen hinaus werden die ersten Bereinigungsschritte (Abschn. 9.2) vorgestellt. Daran knüpft die Analyse der Daten (Abschn. 9.3) sowie die Entwicklung eines passenden Modells an (Abschn. 9.4). Abschließend werden entsprechende Interpretationen (Abschn. 9.5) geliefert. Im Rahmen dieser Vorgehensweise wird ein Skript entwickelt und dessen essenzielle Teile werden entsprechend vorgestellt. Das komplette Skript findet sich im Onlineangebot.

B. M. Abdel-Karim (✉)
Frankfurt am Main, Hessen, Deutschland
E-mail: abdel-karim@wiwi.uni-frankfurt.de

9.1 Videospiel: Datensatz und Fragestellung

Für dieses Anwendungsbeispiel werden Daten in Anlehnung an die Originaldaten der Videospielleplattform `vgchartz.com` verwendet. Vor dem Hintergrund, dass dieses Buch die Datenschutz- und Inhaberrechte Dritter schützen muss, wurden die originalen Videospieldaten mithilfe eines sogenannten Crawler, einem Programmcode zur Extraktion von Webseiteninformationen, entnommen und als Ausgangsbasis für die Erstellung eines synthetischen Datensatzes genutzt. Das bedeutet, dass der hier zur Verfügung gestellte Datensatz zwar in Grundstruktur und Eigenschaften den Originaldaten gleicht, genaugenommen beinhaltet er allerdings andere Daten und ist somit ein eigener und damit neuer Datensatz, sodass entsprechende Urheberrechte gewahrt bleiben. Der Einsatz der Daten in diesem Buch dient als Demonstrationsbeispiel, um das Vorgehen zu veranschaulichen, sodass es hierbei nicht auf die exakte Replizierung von Ergebnissen aus vorangegangenen Arbeiten ankommt. Vor diesem Hintergrund betrachtet dieses Buch die hier vorgelegten Daten als eigenständigen Ausgangspunkt für genauere Analysen in diesem Anwendungsbeispiel und im Kontext des Data-Science-Prozesses.

Ausgehend von den bereits getroffenen Aussagen dient der bereitgestellte Datensatz `dataset_gamesales_final.csv` als Datenbank für die kommenden Arbeitsschritte. Der Datensatz ist daher eine sogenannte Comma-Separated-Values-Datei. Die Dateierendung `.CSV` beschreibt in diesem Fall den Aufbau dieser Datei zur Speicherung einfach strukturierter Daten. Hierbei werden die Werte in diesem Dateiformat durch ein Trennzeichen separiert. Dieser Datentyp ist in der Praxis weit verbreitet. Daher ergibt es aus Sicht dieses Buchs Sinn, zunächst dieses Dateiformat als Basis für die weiteren Aktivitäten zu betrachten. Die Anordnung der Daten ist formal nicht festgelegt, wird aber in der Regel so durchgeführt, dass die einzelnen Einträge einer Zeile einer zeilenweisen Zuordnung unterliegen. Vor diesem Hintergrund ergibt sich eine tabellarische Struktur aus Zeilen und Spalten. Dabei wird jedes Zeilenelement durch ein entsprechendes Trennzeichen vom nächsten Element getrennt. Der Name `.CSV` suggeriert zwar die Trennung durch ein Komma, allerdings können auch andere Trennzeichen spezifiziert werden, wie beispielsweise Semikolon, Doppelpunkt, Tabulatorzeichen oder Leerzeichen. Daher sollte beim Austausch solcher Dateien für den Empfänger spezifiziert werden, welches Trennzeichen zum Einsatz kommt, sodass keine ungewollten Datenmanipulationen beim Einlesen der Dateien durch den Empfänger entstehen. Die Datei `dataset_gamesales_final.csv` ist hierbei mittels „`,`“ separiert worden.

Die Datendatei `dataset_gamesales_final.csv` besteht aus elf Spalten und 15.500 Zeilen mit Werten sowie einer Kopfzeile am Anfang. Jede Spalte repräsentiert eine Variable und jede Zeile eine verknüpfte Datenreihe zu einem Videospiel. Dabei sind die Variablen wie folgt zu interpretieren:

- `'index'`: Das ist eine fortlaufende Nummer (eine ID)
- `'year'`: Das Erscheinungsjahr des Videospiels

- 'game_name': Der Name des Videospieles
- 'publisher': Der Herausgeber des Videospieles
- 'hardware': Die Plattform, auf der das Videospiel betrieben werden kann
- 'genre': Das Genre des Videospieles
- 'sales_EU': Die Einnahmen innerhalb der Europäischen Union in Millionen US\$
- 'sales_USA': Die Einnahmen innerhalb von Nordamerika in Millionen US\$
- 'sales_JP': Die Einnahmen innerhalb von Japan in Millionen US\$
- 'sales_GLOBAL': Die weltweiten Einnahmen in US\$
- 'sales_OTHER': Die Einnahmen im Rest der Welt in Millionen US\$

Auf Basis dieser Informationen zum Datensatz zu den Videospiele lassen sich interessante Fragestellungen ableiten:

- Wie haben sich die Videospieleinnahmen im zeitlichen Verlauf entwickelt?
- Welche Publisher erzielen die meisten Einnahmen?
- Gibt es einen Zusammenhang zwischen den europäischen und den globalen Umsätzen in den Videospieleinnahmen?

Ausgehend von den gewonnenen Erkenntnissen zum Datensatz und den abgeleiteten Fragestellungen kann dieses Data-Science-Anwendungsbeispiel mit dem folgenden Abschnitt beginnen. Hierbei sollten `dataset_gamesales_final.csv` und das neue Skript in einem gemeinsamen Ordner liegen. Für die Ordnerbezeichnung ist ein prägnanter Name sinnvoll, um das Projekt im späteren Verlauf der Arbeiten schnell in der Ordnerstruktur wiederzufinden. Vor diesem Hintergrund wird in diesem Buch der Ordner `Videospiele` genannt. Dieser Ordner beinhaltet die Datei `dataset_gamesales_final.csv` und das Skript `b_Videospiele.py` als Skript für die folgenden Codes. Mithilfe von Pycharm wird der Ordner über `New Projekt` als Projektordner ausgewählt. Damit stehen alle Dokumente in dem Projekt in Pycharm für die Implementierung zur Verfügung.

9.2 Videospiel: Preprocessing

Für das weitere Vorgehen empfiehlt sich die Nutzung des Moduls `pandas`. Hierbei handelt es sich um eine externe Funktionsbibliothek. Diese stammt von einem Entwicklerteam unter der Leitung von Wes McKinney. Die ausführliche Dokumentation zu `pandas` findet sich auf der Dokumentationswebseite <https://pandas.pydata.org>. Der Name leitet sich, im Gegensatz zum häufig verwendeten Panda-Logo, vom englischen Begriff „panel data“ ab. Kernstück des Moduls ist das sogenannte „DataFrame“. Hierbei handelt es sich um eine in der Regel zweidimensionale Datenstruktur (analog zu einer Tabelle), mit größenveränderbaren Zeilen und Spalten. Dabei können die Daten heterogen sein. Zudem besitzt das `DataFrame` entsprechende Achsenbeschriftungen. Die besondere Stärke des `DataFrame` sind zahlreiche

vordefinierte Funktionen, die speziell auf die Verarbeitung von unterschiedlichen Datentypen ausgelegt sind. Somit kann das DataFrame in Python ein mächtiges und effizientes Werkzeug für die eigenen Datenanalysen sein.

Das Modul kann in ein Python-Skript über den Befehl `import pandas as pd` importiert werden. Durch die Anweisung `as pd` in diesem Befehl wird der gesamte Inhalt des Moduls verfügbar und durch `pd` als abgekürztes Schlüsselwort zugänglich. Vor dem Hintergrund, dass pandas ein externes Modul ist, muss es zusätzlich installiert werden. Mit PyCharm gibt es hier zwei Wege. Der erste Weg ist, PyCharm selbstständig erkennen zu lassen, dass das gewünschte Modul nicht existiert und nicht verfügbar ist. Dabei wird die Anweisung von PyCharm rot unterstrichen. Durch einen Doppelklick auf die rote Markierung wird eine rote Lampe mit Ausrufezeichen sichtbar. Diese kann durch einen Rechtsklick dazu aufgefordert werden, „install pandas“ auszuführen. Damit wird PyCharm versuchen, pandas zu installieren. Diese Schnellfunktion in PyCharm ist sehr nützlich, um gängige Module bequem nachzuinstallieren. Alternativ kann über „Preferences“ im Kontextmenü „Projekt:...“ (auf der linken Seite) die Installation im unteren Menübereich über das „+“-Symbol, die Installation von weiteren Modulen veranlasst werden. Professionelle Anwender können als zusätzliche Alternative die Installation über das Terminal veranlassen. Die Beschreibung dieses Wegs bedarf allerdings weiterführende Kenntnisse über das jeweilige Betriebssystem und würde für die Ziele dieses Buchs zu weit führen. Das Codebeispiel 9.1 zeigt den Import des pandas-Moduls in das eigene Python-Skript.

Listing 9.1 Import des Moduls Pandas

```
1 # Anwendungsbeispiel Videospiele
2 # @author: Benjamin M. Abdel-Karim
3 # @since: 2020-04-29
4 # @version: 2020-04-29 – VI
5 # Imports
6 import pandas as pd
7 import matplotlib.pyplot as plt
```

Der Datenimport aus der .CSV-Datei und die Transformation in ein pandas-DataFrame gelingt mithilfe des Modulbefehls `df = pd.read_csv()`. Die Funktion erwartet dabei den Dateinamen. Der Dateipfad wird nur erwartet, wenn die .CSV-Datei und das Skript nicht in einem Ordner liegen. In diesem vorgestellten Anwendungsbeispiel reicht dementsprechend der Dateiname für die Übergabe an die Funktion aus, weil die Datei im selben Projektordner liegt. Als optionalen Parameter erwartet `df = pd.read_csv()` noch das Separationszeichen der .CSV-Datei. Hierfür wird das Separationszeichen an die Funktion in Form eines Strings und dem Funktionsparameternamen `sep=' , '` übergeben. Die Funktion liest mit diesen Parametern die .CSV-Datei ein. In unserem Fall werden die Zeilen der .CSV-Datei mit einem Komma getrennt, sodass die Verwendung des optionalen

Separationsparameters Sinn ergibt. Damit wird ein richtiger Import der Daten sichergestellt. Das Codebeispiel 9.2 zeigt die Umsetzung.

Listing 9.2 Import einer .CSV-Datei als pandas-DataFrame

```

10 # -----
11 # Pre-Processing
12 # -----
13 # Datenimport
14 df = pd.read_csv('dataset_gamesales_final.csv', sep=',')
```

Analog zum Codebeispiel 9.2 bietet diese Funktion zahlreiche optionale Parameter. Eine ausführliche Darstellung findet sich in der offiziellen Dokumentation¹. Wird das Codebeispiel ausgeführt, so ist das Ergebnis die Erstellung der Variable `df`, die das DataFrame beinhaltet. Das Buch verwendet hierbei die in der Community geläufige Benennungskonvention `df`, um eine Variable als DataFrame zu benennen.

Über die Funktion `df.head()` in einer `print()`-Anweisung lassen sich die ersten fünf Zeilen des Datensatzes zurückgeben. Das Codebeispiel 9.3 zeigt den Aufruf dieser Funktion.

Listing 9.3 Aufruf der `df.head()`-Funktion

```

16 # Die ersten fuenf Zeilen anzeigen lassen.
17 print(df.head())
```

In der Standardbelegung gibt die Funktion aus dem Codebeispiel 9.3 die ersten fünf Zeilen aus.

Listing 9.4 Konsolenausgabe: `df.head()`

	index	year	game_name	...	sales_JP	sales_GLOBAL	sales_OTHER
2	0	1980	Asteroids	...	0.0	1.093598	0.021345
3	1	1980	Bridge	...	0.0	0.062658	0.000000
4	2	1980	Defender	...	0.0	0.240872	0.002410
5	3	1980	Freeway	...	0.0	0.048173	0.000000
6	4	1980	Kaboom!	...	0.0	0.413267	0.000041

Die Konsolenausgabe 9.4 zeigt eine verkürzte Darstellung (...) der ersten fünf Einträge. Als Alternative kann in PyCharm der „Debugger“ eingesetzt werden, um mithilfe eines zusätzlichen Variableninspektors das DataFrame zu betrachten. Hierzu muss an einer beliebigen Stelle im Quellcode ein „Breakpoint“ gesetzt werden und der Quellcode mithilfe der „Debug“-Funktion in PyCharm bis zu diesem Punkt ausgeführt werden. Der Begriff Debugging leitet sich aus dem Englischen ab und bedeutet sinngemäß „Entwanzen des Programmcodes“. An dieser Stelle bietet der Debugger in Python zusätzliche Werkzeuge, wie den

¹ https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html, zuletzt abgerufen 12. Februar 2022.

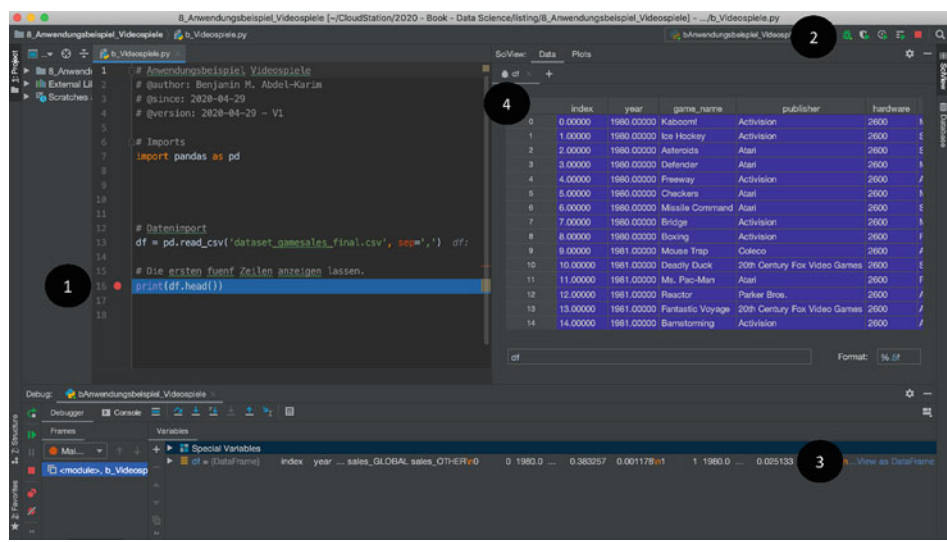


Abb.9.1 Der Variableninspektor in PyCharm zur Visualisierung der Wertebelegung eines DataFrame

Variableninspektor, der imstande ist, die genauen Variablenbelegungen zur Laufzeit anzuzeigen. In PyCharm muss allerdings zur Anzeige des Variableninspektors, in Abgrenzung zu anderen IDE wie beispielsweise in Spyder, der Umweg über die Debug-Funktion gegangen werden. Die Abb. 9.1 zeigt die Vorgehensweise.

Zur Nutzung des PyCharm-Variableninspektors wird zwischen der Codeline 16 und dem Editor mit einem Klick auf die Freifläche ein Breakpoint (Abb. 9.1 Nr. 1) gesetzt. Anschließend erfolgt der Start des Debugger durch einen Klick auf das Debugger-Symbol (Abb. 9.1 Nr. 2). Der Debugger führt den Quellcode nun bis zum Breakpoint aus und unterbricht dann den Fortlauf des Quellcodes. PyCharm öffnet anschließend eine zusätzliche Menübar am unteren Bildschirmrand der IDE, sodass alle aktuell verfügbaren Variablen und die entsprechenden Wertebelegungen angezeigt werden. Bei der DataFrame-Variable stellt PyCharm eine zusätzliche Ansicht bereit (Abb. 9.1 Nr. 3). Durch einen Klick auf diese zusätzliche Ansicht kann das gesamte DataFrame betrachtet werden (Abb. 9.1 Nr. 4). Diese Art der Wertbetrachtung ist insbesondere im Rahmen der Datenanalyse im Data-Science-Prozess eine wertvolle Hilfestellung. Um ein noch besseres Verständnis für den Datensatz zu erhalten, eignet sich eine weitere Funktion des DataFrame. Mit `df.info()` lassen sich weitere Informationen zum Datensatz auf der Konsole ausgeben. Die Eingabe zeigt das Codebeispiel 9.5.

Listing 9.5 Aufruf der `df.info()`-Funktion

```
19 # Aufruf der Info-Funktion.  
20 print(df.info())
```

Diese Funktion gibt Informationen über das DataFrame, einschließlich der vorhandenen Datentypen und deren Speicherverbrauch, übersichtlich aus. Die Ausführung der Funktion auf diesen Datensatz ergibt die folgende Konsolenausgabe 9.6.

Listing 9.6 Konsolenausgabe: df.info()

```

1 <class 'pandas.core.frame.DataFrame'> RangeIndex: 16291 entries, 0
2 to 16290 Data columns (total 11 columns):
3 #   Column      Non-Null Count  Dtype
4 ---  ---
5 0   index       16291 non-null  int64
6 1   year        16291 non-null  int64
7 2   game_name   16291 non-null  object
8 3   publisher   16291 non-null  object
9 4   hardware    16291 non-null  object
10 5   genre       16291 non-null  object
11 6   sales_EU    16291 non-null  float64
12 7   sales_USA   16291 non-null  float64
13 8   sales_JP    16291 non-null  float64
14 9   sales_GLOBAL 16291 non-null  float64
15 10  sales_OTHER 16291 non-null  float64
16 dtypes: float64(5), int64(2), object(4)
17 memory usage: 1.4+ MB None

```

Die Konsolenausgabe 9.6 zeigt, dass der Datensatz aus elf Variablen besteht und die Anzahl der einzelnen Datenpunkte keine Nullwerte sind. Es handelt sich demnach um Werte, die tatsächlich belegt worden sind. Außerdem verrät die Spalte Dtype die überwiegenden Datentypen. Ausgehend von der Ausführung der Funktion `df.info()` ergibt sich die Erkenntnis, dass keine Einträge im Datensatz vorhanden sind, die in irgendeiner Art behandelt werden müssten. Eine entsprechende Bereinigung der Daten könnte beispielsweise durch das Ersetzen der Nullwerte durch 0 sein oder die Werte könnten interpoliert, d.h. synthetisch berechnet werden. Dieses erste Fallbeispiel ist bewusst so gewählt worden, damit sich der Leser auf die Kernschritte konzentrieren kann. Die Bereinigung von Daten ist sicherlich ein eigenes Forschungsfeld. Vor diesem Hintergrund werden in diesem Kapitel zunächst die essenziellen Schritte im Data-Science-Prozess beschrieben und auf die Datenbereinigung wird ausführlicher in den folgenden Kapiteln eingegangen.

Damit wurde der Datensatz in diesem ersten Schritt als DataFrame eingelesen und es wurde ein Verständnis dafür geliefert, wie der Debugger dabei helfen kann, die Prüfung der Daten auf Vollständigkeit zu vereinfachen. Die Tab. 9.1 gibt einen Überblick über die ersten Funktionen zum Datenimport mithilfe des pandas-Moduls in Python.

Ausgehend von diesen ersten Schritten baut der nächste Abschnitt auf den bisherigen Codebeispielen auf und vertieft das Thema Datenanalyse. So wurden im obigen Abschnitt die ersten notwendigen Grundbausteine durch den Import und die Überprüfung der Daten gelegt.

Tab. 9.1 Übersicht nützlicher Funktionen des pandas-Moduls für die Videospielanalyse im ersten Pre-Processing

Funktion	Auswertung zu:
<code>pd.read_csv()</code>	Import der Daten aus einer .CSV-Datei in ein DataFrame
<code>df.head()</code>	Ausgabe der ersten fünf Zeilen des DataFrame auf der Konsole
<code>df.info()</code>	Ausgabe der Variablen, deren Nicht-Null-Werteanzahl und den Datentypen

9.3 Videospiel: Explore the Data

Ausgehend von dem Data Preprocessing geht es im nächsten Schritt primär darum, ein Gefühl für die Daten zu bekommen. Das bedeutet ein besseres Verständnis über die Datenpunkte und ihre Beziehungen untereinander zu erhalten. Eine einfache Zusammenfassung der Daten bietet dabei beispielsweise die Funktion `df.describe()`. Diese Funktion gibt zunächst eine erste deskriptive Statistik zu den Variablen aus, indem sie die zentrale Tendenz, Streuung und die Form der Verteilung eines Datensatzes zusammenfasst. Hierbei werden potenzielle NaN-Werte nicht berücksichtigt. Das Bemerkenswerte ist, dass diese Funktion die Heterogenität von Datensätzen berücksichtigt. Das Codebeispiel 9.7 zeigt die Anwendung der Funktion auf den hier vorgestellten Datensatz.

Listing 9.7 Aufruf der Funktion `df.describe()`

```
23 # -----
24 # Explore the data
25 # -----
26 # Explore the data
27 print(df.describe())
28
29 # Fuer mehr Informationen auf der Konsole
30 with pd.option_context('display.max_columns', 11):
```

Der Aufruf dieser Funktion führt zu der dargestellten Konsolenausgabe 9.8.

Listing 9.8 Konsolenausgabe: `df.describe()`

1	index	year	...	sales_GLOBAL	sales_OTHER
2	count	16291.000000	16291.000000	...	16291.000000 16291.000000
3	mean	8145.000000	2006.405561	...	0.134592 0.012256
4	std	4702.950953	5.832412	...	0.383525 0.057249
5	min	0.000000	1980.000000	...	0.000000 0.000000
6	25%	4072.500000	2003.000000	...	0.012210 0.000000
7	50%	8145.000000	2007.000000	...	0.038556 0.001836
8	75%	12217.500000	2010.000000	...	0.115028 0.007783
9	max	16290.000000	2020.000000	...	15.544977 3.599725

Die obige Konsolenausgabe 9.8 gibt die entsprechenden deskriptiven Statistiken für die einzelnen Variablen aus. In der Standardeinstellung der Funktion sind es die Anzahl der einzelnen Werte (count), der Mittelwert (mean), die dazugehörige Standardabweichung (std), das Minimum (min) sowie das 25-Prozent-, 50-Prozent- und 75-Prozent-Quantil. Wie schon im `df.head()`-Codebeispiel gezeigt, verkürzt Python die Ausgabe auf der Konsole. Das Quellcodebeispiel 9.7 für die Funktion `df.describe()` zeigt die erweiterte Darstellung der Konsolenausgabe. Weitere Informationen zur Funktion `df.describe()` finden sich auf der Dokumentationsseite².

Neben dieser ersten deskriptiven Statistik kann die Visualisierung der Daten ein nützliches Element sein. Hierzu sollen zwei zusätzliche Module dienen. Es wird in diesem Buch das Modul `matplotlib` vorgestellt. Dieses Python-Modul eignet sich zur Erstellung animierter, statistischer und interaktiver Visualisierungen. Auf diesem Modell baut das Modul `seaborn` auf. Dieses Modul bietet primär eine übergeordnete Schnittstelle zwischen dem Datenmodul `pandas` und `matplotlib`. Mithilfe des Moduls `seaborn` können Visualisierungen für `DataFrames` relativ einfach erstellt werden. Sowohl für `matplotlib` als auch für `seaborn` finden sich ausführliche Gallerien^{3,4}.

Vor dem Hintergrund der verfügbaren Module ergibt es Sinn, diese für das Vorhaben zu nutzen. Zur Nutzung der Module werden sie entsprechend dem `pandas`-Modul im oberen Teil des Skripts, direkt unter der Anweisung `import pandas as pd` durch die Befehle `import matplotlib.pyplot as plt` und `import seaborn as sns` aktiviert, nachdem sie gegebenenfalls installiert worden sind (siehe hierzu die Beschreibung in Abschn. 9.2).

Für eine deduktive Exploration der Daten eignet sich die Erstellung eines sogenannten `Pairplot`. Hierbei handelt es sich um eine Darstellungsart, die die paarweisen Beziehungen der Daten durch sogenannte `Scatterplots` in einer Abbildung realisiert. Die grundlegenden `Scatterplots` tragen die Beziehungen zwischen zwei numerischen Variablen ab. Damit bestehen die X- und Y-Koordinate eines jeden Punkts eines `Scatterplots` aus jeweils einer Variablen. Diese grafische Darstellung ermöglicht das Erkennen von Beziehungen. Der `Pairplot` übernimmt damit die Darstellung der Variablenkombinationen aus dem `DataFrame`, wobei auf der Diagonalen dieser Darstellung die Verteilungsdiagramme abgetragen werden. Die Umsetzung dieser Darstellung zeigt der Beispielcode 9.9.

² <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html>, zuletzt abgerufen am 12. Februar 2022.

³ <https://matplotlib.org/gallery.html>, zuletzt abgerufen am 12. Februar 2022.

⁴ <https://seaborn.pydata.org/examples/index.html>, zuletzt abgerufen am 12. Februar 2022.

Listing 9.9 Erstellung eines Pairplot für die Videospiele

```
33 # Pairplot
34 plt.figure()
35 sns.pairplot(df, vars=['year', 'sales_EU', 'sales_USA', 'sales_JP',
36                       'sales_GLOBAL', 'sales_OTHER'])
37 plt.tight_layout()
38 plt.savefig('Pairplot.png')
```

Die Funktion `plt.figure` erstellt eine Abbildungsumgebung, in der Pairplot mithilfe der Funktion `sns.pairplot` eingesetzt werden kann. Damit kommt die Funktion `plt.figure` aus dem `matplotlib`-Modul (`plt.`) und der Pairplot aus dem `seaborn`-Modul (`sns.`). Der Pairplot erwartet dabei das `DataFrame` als Datenbasis. Der optionale Parameter `vars` ermöglicht dabei, die Erstellung des Pairplot auf die Darstellung einzelner Parameter zu beschränken.

Dieser Parameter wird an dieser Stelle eingesetzt und alle Parameter bis auf den Datensatzindex werden übergeben. Die Erstellung der Pairplot-Paare erfolgt anschließend auf der Y-Achse in der Abfolge von oben nach unten durch die Variablenabtragung 'year', 'sales_EU', 'sales_USA', 'sales_JP', 'sales_GLOBAL', 'sales_OTHER' sowie auf der X-Achse in derselben Anordnung. Das Ergebnis dieser Operation zeigt die Abb. 9.2.

Der Pairplot ermöglicht es, relativ leicht und nur mittels einer visuellen Inspektion der Daten, multiple Beziehungen in unserem Fall zu erkennen. Beispielsweise ist zu erkennen, dass 'sales_EU' und 'year' (Abb. 9.2 zweiter Scatterplot von oben auf der linken Seite) im Zeitablauf tendenziell ansteigen. Demgegenüber scheinen 'sales_JP' und 'year' (Abb. 9.2 vierter Scatterplot von oben auf der linken Seite) relativ unkorreliert im Zeitverlauf.

Der Pairplot hat allerdings auch einige Limitationen. Ein Kritikpunkt bei dieser Darstellungsform ist, dass mit jeder hinzugefügten Variable die Komplexität in der Darstellung zunimmt. Damit wird die Darstellung zunehmend unübersichtlicher. Zudem nimmt die benötigte Rechenleistung zur Erstellung des Pairplot zu. Vor diesem Hintergrund muss der Einsatz dieser Darstellung wohlüberlegt sein und es muss gegebenenfalls auf andere Visualisierungen zurückgegriffen werden.

Ausgehend von dem gebauten Pairplot bietet sich eine detaillierte Betrachtung der Einnahmen im zeitlichen Verlauf an. Für solch eine Betrachtung ist ein Barplot eine hilfreiche Darstellung. Im Deutschen auch Balkendiagramm genannt, bildet er die Anzahl von Ereignissen mithilfe der Höhe von entsprechenden Balken ab. Dies macht für das menschliche Auge Unterschiede im zeitlichen Verlauf schnell erfassbar. Das Codebeispiel 9.10 zeigt die Umsetzung für die Abbildung der kumulierten Einnahmen im zeitlichen Verlauf.

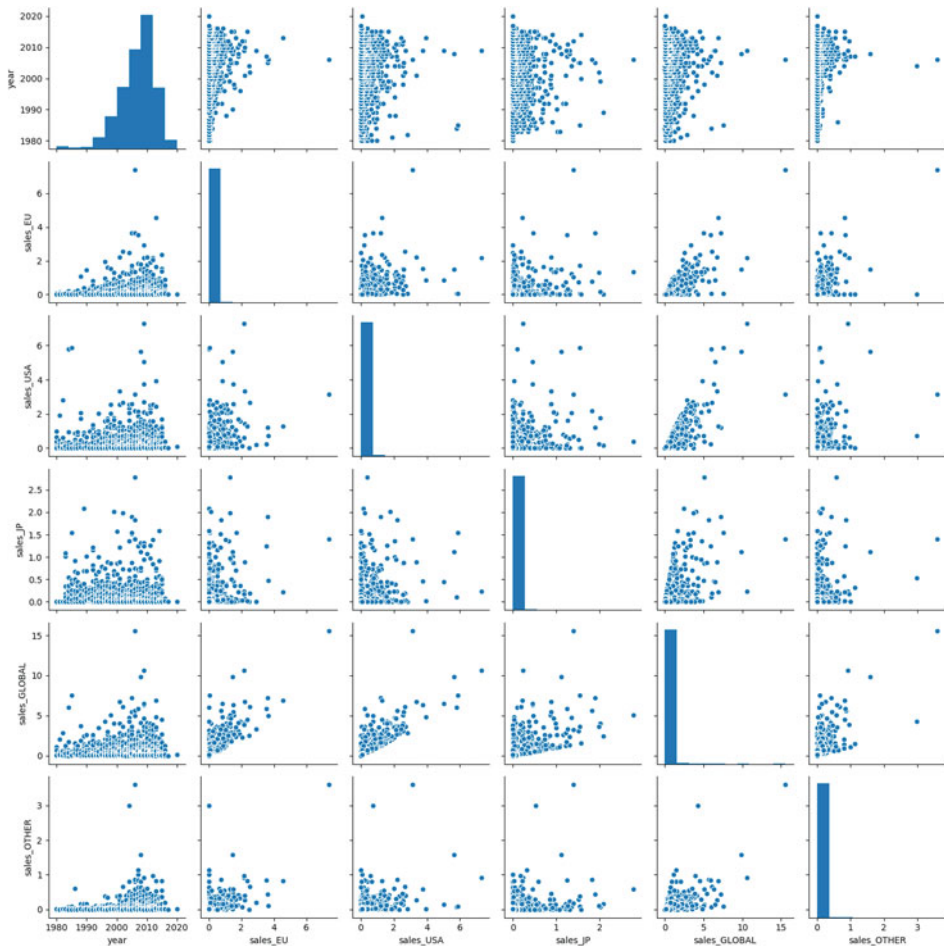


Abb. 9.2 Der Pairplot für die Videospiele

Listing 9.10 Erstellung eines Barplot für die Videospiele im zeitlichen Verlauf

```

42 # — Verkaufs pro Jahr —————
43 # Verkaufs pro Jahr mit Hilfe von Groupby
44 df_GroupYearSum = df.groupby(['year']).sum()
45
46 # Abbildung erstellen
47 fig, ax = plt.subplots()
48 ax.spines['top'].set_visible(False)
49 ax.spines['right'].set_visible(False)
50
51 # Add counts above the two bar graphs

```

```

52 ax = sns.barplot(y=df_GroupYearSum['sales_GLOBAL'], x=df_GroupYearSum.index.astype
    (int), color='white', edgecolor='black')
53 ax.set_xlabel(xlabel='Jahr', fontsize=12)
54 ax.set_xticklabels(labels=df_GroupYearSum.index.astype(int), fontsize=12, rotation
    =45)
55 ax.set_ylabel(ylabel='Verkauefe', fontsize=12)
56 ax.set_title(label='Videospieleinnahmen in Millionen Dollar pro Jahr', fontsize=20)
57 fig.set_size_inches(12, 8, forward=True)
58 plt.savefig('YearAndSales.pdf', bbox_inches='tight')
59 plt.show()

```

Das Codebeispiel zeigt zunächst die Gruppierung nach Jahren (Variable 'year') mithilfe von `df.groupby(['year'])`. Hierbei werden die Datenmengen zu Gruppen auf Basis des jeweiligen Jahres zusammengefasst. Durch die angehängte Operation `sum()` werden die numerischen Variablen entsprechend auf Jahresebene kumuliert. Das Ergebnis dieses Funktionsaufrufs ist ein neues DataFrame, das in dieser Variable `df_GroupYearSum` deklariert worden ist. `df_GroupYearSum` bildet nun die Grundlage für die Visualisierung. Zur Visualisierung wird durch `fig, ax = plt.subplots()` ein Figure-Objekt und ein Tupel mit Achsenobjekten erstellt, sodass die Achsen im späteren Verlauf moduliert werden können. In diesem Fall bildet `fig` das Figure-Objekt und `ax` das Achsenobjekt. Das Achsenobjekt wird direkt genutzt, um die obere und rechte Achse auszublenden. `ax.spines['top'].set_visible(False)` und `ax.spines['left'].set_visible(False)` führen dazu, dass die beiden unerwünschten Achsen unsichtbar werden. Sie bilden normalerweise den Rahmen um eine Abbildung. Im nächsten Schritt geht es an die eigentliche Erstellung der Abbildung. Hierzu wird mithilfe von `ax = sns.barplot()` ein seaborn-Barplot erstellt und dem Achsenobjekt zugewiesen. Dieser Barplot bekommt direkt die Spalte des DataFrames mit den kumulierten globalen Einnahmen durch die Zuweisung und den Y-Parameter durch `y= df_GroupYearSum['sales_GLOBAL']` übergeben. Damit ist es möglich, auf dem neu erstellten DataFrame ebenfalls über den Namensaufruf der Spalte, die Daten dieser Spalte in eine neue Variable zu allokalieren. In diesem Fall bekommt also der Y-Parameter der seaborn-Barplot-Funktion die Werte der Spalte `['sales_GLOBAL']` des gruppierten DataFrames `y=df_GroupYearSum` direkt übergeben. Der X-Parameter des Barplot erwartet die Positionen der einzelnen Balken. Dies geschieht durch den Zugriff auf die Indexwerte des neuen DataFrames `df_GroupYearSum`. Hierbei handelt es sich um Indexwerte, die durch die Erstellung des DataFrames automatisch erzeugt wurden. Deshalb handelt es sich hierbei also um andere Indexwerte als die im Originaldatensatz vorhandenen. Diese neuen Indexwerte entsprechen den Jahren und entstehen durch die `df.groupby(['year'])`. Das DataFrame verwendet bei der Gruppierung der Werte den Gruppierungsparameter als Grundlage für den Index, also dem Identifikator der Werte. Durch den Befehl `index.astype(int)` werden die Indexwerte durch einen Typcast zu einem Integer umgewandelt. Diese Integer-Werte werden nun für die X-Achse zur Positionsallokation genutzt. Die Belegung der optio-

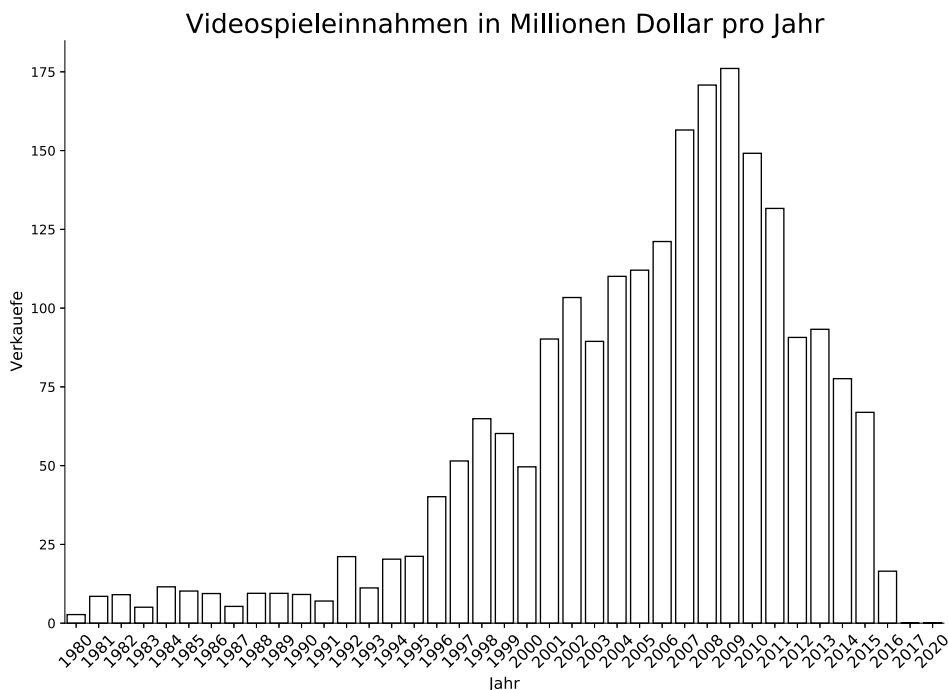


Abb. 9.3 Der Barplot für die Videospieleinnahmen im Zeitverlauf

nalen Parameter `color='white'` und `edgecolor='black'` setzt die Farbgebung der Balken fest. Hier erhalten diese einen weißen Hintergrund und schwarze Kanten. Die anschließenden Anweisungen, die durch `ax.set_` eingeleitet werden, dienen der Achsenbeschriftung und der Festlegung der Schriftgröße (`fontsize=12`). Anschließend wird durch `fig.set_size_inches` die Standardgröße der zu erstellenden Abbildung auf die Maße zwölf zu acht erweitert. Der optionale Parameter `forward=True` bewirkt, dass die Leinwandgröße automatisch skaliert wird. Zahlreiche weitere Einstellungsmöglichkeiten von Abbildungen finden sich auf der Dokumentationsseite⁵. Am Schluss wird die Abbildung als Portable Document Format (.PDF) abgespeichert. Das PDF besitzt den Vorteil, dass ein .PDF-Dokument ein Dokument im Vektorformat ist. Damit ist es frei skalierbar, sodass die Darstellungen auch in der größten Zoom-Stufe scharf bleiben. Das Ergebnis des Codebeispiels zeigt die Abb. 9.3.

Die Betrachtung der Abb. 9.3 zeigt, dass der Verkauf zwischen 1980 und 1995 relativ stetig und horizontal verläuft. Ab 1996 beginnt ein bemerkenswerter Anstieg der Spieleinnahmen, der bis 2009 anhält. Nach diesem Verkaufsjahr fallen die Einnahmen der Videospiele wieder ab.

⁵ https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.figure.Figure.html, zuletzt abgerufen am 12. Februar 2022.

Eine weitere interessante Datenerkundung kann z. B. die Analyse der Top-Ten-Publisher sein. Die Umsetzung zeigt hierzu das folgende Codebeispiel 9.11.

Listing 9.11 Erstellung eines Barplot für die Top-Ten-Publisher

```

42 # — Verkaufte pro Jahr —————
43 # Verkaufte pro Jahr mit Hilfe von Groupby
44 df_GroupYearSum = df.groupby(['year']).sum()
45
46 # Abbildung erstellen
47 fig, ax = plt.subplots()
48 ax.spines['top'].set_visible(False)
49 ax.spines['right'].set_visible(False)
50
51 # Add counts above the two bar graphs
52 ax = sns.barplot(y=df_GroupYearSum['sales_GLOBAL'], x=df_GroupYearSum.index.astype
    (int), color='white', edgecolor='black')
53 ax.set_xlabel(xlabel='Jahr', fontsize=12)
54 ax.set_xticklabels(labels=df_GroupYearSum.index.astype(int), fontsize=12, rotation
    =45)
55 ax.set_ylabel(ylabel='Verkaufe', fontsize=12)
56 ax.set_title(label='Videospieleinnahmen in Millionen Dollar pro Jahr', fontsize=20)
57 fig.set_size_inches(12, 8, forward=True)
58 plt.savefig('YearAndSales.pdf', bbox_inches='tight')
59 plt.show()

```

Im Rahmen dieser Datenanalyse sollen die Top-Ten-Publisher, basierend auf den Einnahmen, ermittelt werden. Das Ergebnis soll ein horizontaler Barplot sein, bei dem der Publisher mit dem größten Umsatz oben zu finden ist, sodass die Top-Ten in absteigender Reihenfolge, gemessen an den Einnahmen, dargestellt werden. Das Codebeispiel zeigt, dass erneut eine DataFrame-Gruppierung durchgeführt wird. Hierbei werden diesmal die Variable 'publisher' als Gruppierung herangezogen und die Anzahl der globalen Einnahmen mit `.sum()['sales_GLOBAL']` aufsummiert. Für die sortierte Reihenfolge wird auf dem neuen DataFrame `df_SalesByPublisher` die Funktion `sort_values()` aufgerufen. Sie sortiert das DataFrame. Diese Funktion befindet sich innerhalb der Funktion `pd.DataFrame`, die dazu dient, ein neues DataFrame zu erstellen. Sie besteht aus den ersten zehn Positionen des `df_SalesByPublisher`, indem der Zeilenindex von Null bis Zehn (`[0:10]`) übergeben wird. Das Ergebnis dieses verschachtelten Funktionsaufrufs liefert ein neues DataFrame zurück, das in der Variable `df_SalesByPublisherTopTen` gespeichert wird. Dieses DataFrame bildet die Datengrundlage für den horizontalen Barplot. Bei dieser Abbildung werden erneut die seitlichen Achsen auf transparent geschaltet. Für die Färbung der Balken soll dieses Mal eine abstufende Farbpalette genutzt werden. Diese kann durch das `seaborn`-Modul ermöglicht werden, indem die Funktion `sns.color_palette()` mit einer entsprechenden Farbpalette und der Anzahl an Farbabstufungen aufgerufen wird. Für dieses Beispiel soll entsprechend des Buchlayouts

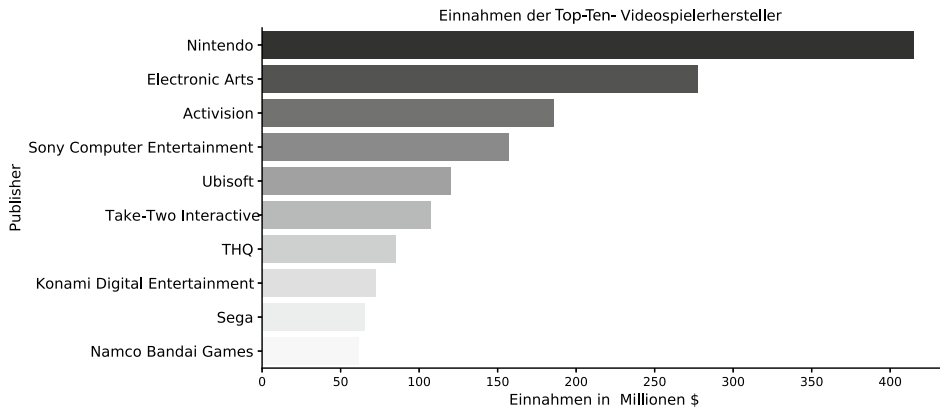


Abb.9.4 Der horizontale Barplot für die Top-Ten-Videospiel-Publisher gemessen an den Einnahmen

eine Abstufung aus Grautönen verwendet werden. Dabei stellt das Modul `matplotlib`, auf das sich das `seaborn`-Modul stützt, eine große Auswahl an Farben und Paletten auf der Dokumentationsseite⁶ bereit. Für die Anzahl der Farbtöne dient die Anzahl der Publisher, die über `len()` und dem `DataFrame` ermittelt wird. Durch den Aufruf der Funktion `reversed()` wird die Farbabstufung invertiert, sodass die Farbelemente in umgekehrter Reihenfolge vorliegen und damit der höchste Wert, die dunkelste Farbe, anstatt die hellste Farbe, zugeordnet bekommt. Die Farben werden dann in die Variable `LColors` gespeichert. Wie die Benennung verrät, handelt es sich hierbei um eine Liste bestehend aus den entsprechenden Farbkodierungen. Der nachfolgende Code im Codebeispiel 9.11 setzt zusätzlich die Achsenbeschriftungen und die Formatierung der Größe, sodass abschließend die Abbildung als .PDF-Dokument gespeichert wird. Das Ergebnis zeigt die Abb. 9.4.

Die nähere Betrachtung der Ergebnisse aus Abb. 9.4 zeigt, dass der Publisher Nintendo mit etwa 415 Mio. US\$ die Liste der Top-Ten-Einnahmen anführt. Auf dem zweiten Platz folgt Electronic Arts mit etwa 277 Mio. US\$, gefolgt von Activision mit etwa 185 Mio. US\$.

Basierend auf dieser Analyse drängt sich die Frage auf, welches die erfolgreichsten Videospiele gemessen am globalen Verkauf sind. Die Beantwortung der Frage gelingt durch die Verwendung des oben erstellten Codebeispiels und ein paar kleinen Modifikationen. Das Codebeispiel 9.12 zeigt die Umsetzung.

Listing 9.12 Top-Ten-Videospiele

```

82
83 # ----- Top-10 Games -----
84 # Top-10 Analyse fuer die Games.
85 df_SalesByGame = df.groupby(['game_name']).sum()['sales_GLOBAL']
86 df_SalesByGameTopTen = pd.DataFrame(df_SalesByGame.sort_values(ascending=False))
87 [0:10]
88 print(df_SalesByGameTopTen)

```

⁶ <https://matplotlib.org/tutorials/colors/colormaps.html>, zuletzt abgerufen am 12. Februar 2022.

Das Codebeispiel 9.11 für die Top-Ten-Publisher kann für die Fragestellung hinsichtlich der Top-Ten-Videospiele kopiert und entsprechend modifiziert werden. Hierzu wird im kopierten Code die Variable `df_SalesByPublisher` zu `df_SalesByGame` und `df_SalesByPublisherTopTen` zu `df_SalesByGameTopTen` umgeschrieben. Die `groupby()` bekommt statt der Publisher-Variable die Variable für die Videospielnamen `'game_name'` übergeben. Das Vorgehen zeigt, wie einfach es sein kann, Analysen zu erweitern, sofern der Basiscode erst einmal geschrieben ist. Das Ergebnis wird anstelle einer Abbildung durch den Aufruf `print()` auf der Konsole ausgegeben und führt zu dem Ergebnis in 9.13.

Listing 9.13 Konsolenausgabe: Top-Ten-Videospiele

	sales_GLOBAL
1	
2	game_name
3	Grand Theft Auto V 15.628081
4	Wii Sports 15.544977
5	New Super Mario Bros. Wii 10.614181
6	Mario Kart Wii 9.817236
7	Super Mario Bros. 9.439853
8	Call of Duty: Black Ops 3 7.665491
9	Call of Duty: Black Ops II 7.456758
10	Call of Duty: Modern Warfare 3 7.420088
11	Mario Kart DS 7.183404
12	Call of Duty: Modern Warfare 2 6.850581

Abschließend zu diesem Abschnitt wird in der Tab.9.2 eine Übersicht der bis hierhin verwendeten Funktionen im Kontext der Datenanalyse gegeben.

9.4 Videospiel: Model the Data (Regression)

Ausgehend von den ersten Analysen drängt sich die Frage auf, ob es einen Trend der Videospieleinnahmen im zeitlichen Verlauf gibt. Für diesen Zweck bietet sich beispielsweise eine Regression an. Hierzu werden in Abschn.9.4.1 die theoretischen Grundlagen zu diesem Modell geliefert sowie in Abschn.9.4.2 eine mögliche Implementierung gezeigt.

9.4.1 Theoretische Modellgrundlagen

Die Lineare Regression kann als mathematisches Modell eingesetzt werden, um lineare Zusammenhänge zwischen verschiedenen unabhängigen (auch exogenen) Variablen x_i und einer hierzu abhängigen (auch endogenen) Variable \hat{y} zu identifizieren. Die Formel 9.1 zeigt eine klassische lineare Regression:

$$\hat{y} = \alpha + \beta_1 \cdot x_1 + \dots + \beta_n \cdot x_n + \epsilon \tag{9.1}$$

Tab. 9.2 Übersicht nützlicher Funktionen für die Datenexploration

Funktion	Auswertung zu:
<code>plt.figure()</code>	Erstellung einer Abbildungsumgebung (Modul: matplotlib)
<code>sns.pairplot()</code>	Pairplot für einen Daten-Scatter-Plot (Modul: seaborn)
<code>sns.barplot</code>	Balkendiagramm erstellen (Modul: seaborn)
<code>plt.savefig('FORMAT')</code>	Speicherung der Abbildung im gewünschten Format (.png, .pdf, .svg)
<code>ax.set_ylabel('TEXT')</code>	Y-Achsen-Beschriftung als String 'TEXT' setzen (Modul: matplotlib)
<code>ax.set_xlabel('TEXT')</code>	X-Achsen-Beschriftung als String 'TEXT' setzen (Modul: matplotlib)
<code>fig, ax=plt.subplots()</code>	Erstellung einer Subplot-Umgebung mit Abbildungsumgebung (fig) und Achsenobjekten (ax) in (Modul: matplotlib)
<code>ax.spines['AX'] .set_visible(False)</code>	Einstellung der Achsensichtbarkeit mithilfe von fig, ax=plt.subplots(), um gewünschte Achsen unsichtbar zu schalten. In diesem Buch werden 'top' und 'right' unsichtbar gestellt

- \hat{y} Endogene Variable (analoge Bezeichnungen sind abhängige Variable, Regressor oder zu erklärende Variable)
- x_i Beliebige exogene Variable (analoge Bezeichnungen sind unabhängige Variable, Regressand oder erklärende Variable)
- α Konstante der Regressionsgleichung
- β_i Sensitivität oder auch Regressionskoeffizient genannt. Dieser beschreibt die Stärke, mit der \hat{y} auf Änderungen von x_i reagiert
- ϵ Beschreibt die unsystematische und zufällige Störgröße auf das Modell, also den Anteil, der nicht durch die Linearkombination erklärt werden kann

Im Kontext der linearen Regression können zwei grundsätzliche Regressionen unterschieden werden. Im Rahmen einer univariaten Regression hängt der Regressor nur von einer exogenen Variable ab. Bei einer multivariaten Regression dahingegen fließen bis zu n verschiedene exogene Variablen in das Modell ein.

Im Wesentlichen wird bei einer klassischen linearen Regression ein linearer Zusammenhang zwischen der endogenen und den exogenen Variablen unterstellt. Hierbei wird angenommen, dass es sich um ein Optimierungsproblem handelt, das sich mittels Kleinste-Quadrate-Schätzung (Ordinary Least Squares (OLS)) lösen lässt. Dieses Optimierungsproblem lässt sich wie folgt formalisieren:

$$\min_{x_i \in X} RQS = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (9.2)$$

- RQS Residuenquadratsumme oder auch SSR für Sum of Squared Residuals
- x_i i -te exogene Variable
- y_i Beobachteter Wert zur entsprechenden exogenen Variable x_i
- \hat{y}_i Wert, der durch die Regressionsgleichung zur entsprechenden exogenen Variable x_i geschätzt wird

Ausgehend von diesen Überlegungen sollen hier für die Regressionsgleichung drei Annahmen getroffen werden, die bei der Umsetzung dieses Modells berücksichtigt werden müssen. Erstens, das Modell erwartet einen linearen Zusammenhang der endogenen und exogenen Variablen. Zweitens, die Störgröße ϵ ist zufällig und normalverteilt. Das bedeutet insbesondere, dass sie einen Erwartungs- oder hier Mittelwert von Null besitzt. Drittens, die exogenen Variablen sind linear unabhängig. Mit diesen Grundlagen kann die erste einfache Regression auf dem Datensatz geschätzt werden. Zweifellos ist das Themenfeld der Regression weitreichend. Vor diesem Hintergrund kann das Buch nur einen kleinen Eindruck leisten. Daher sei an dieser Stelle auf die weiterführende Grundlagenliteratur wie beispielsweise Poddig et al., (2008, 2015) oder Backhaus et al., (2018) verwiesen.

9.4.2 Implementierung des Modells

Die Implementierung von Modellen aus der Familie der Regression kann mithilfe des Moduls `statsmodels` umgesetzt werden. Dieses Modul bietet zahlreiche vordefinierte Funktionen zum Schätzen von Modellen. Im Zuge dieser Analyse wird das Modul durch den Aufruf in der oberen Importsektion des Skripts durch die Anweisung `import statsmodels.formula.api as smf` aktiviert. Ausgehend von der exemplarischen Fragestellung ergibt es Sinn, zunächst einen Zusammenhang zwischen den Videospielumsätzen in der EU und den globalen Videospielumsätzen zu analysieren. Hierzu wird zunächst für die Regression ein String `sFormula` definiert. Dieser bekommt entsprechend der Dokumentation⁷ des Moduls die exogene Variable (`'sales_EU'`) übergeben. Diese Variable soll mit der exogenen Variable (`'sales_GLOBAL'`) geschätzt werden. Damit bildet die Variable `sFormula`

⁷ <https://www.statsmodels.org/stable/index.html>, zuletzt abgerufen am 12. Februar 2022.

das formalisierte Modell. Durch den Aufruf von `smf.ols` wird vom importierten Modul das Modell zur Regression aufgerufen. Hierbei wird sowohl das formulierte Modell als auch das `DataFrame` übergeben. Durch den Aufruf `.fit()` wird das Modell entsprechend geschätzt. Das Modul verwendet damit die Formalisierung des Nutzers sowie das `DataFrame`. Die Fähigkeit zu solch einer abstrahierten Lösung, die nur das formalisierte Modell als String sowie ein `DataFrame` erwartet, ergibt ein mächtiges Werkzeug. Der abschließende Aufruf `print(OLSModel.summary())` gibt die geschätzten Modellergebnisse auf der Konsole aus. Das Ergebnis zeigt der Auszug aus der Konsolenausgabe 9.14.

Listing 9.14 Konsolenausgabe: Lineare Regression

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

OLS Regression Results						
Dep. Variable:	sales_EU	R-squared:	0.681			
Model:	OLS	Adj. R-squared:	0.681			
Method:	Least Squares	F-statistic:	3.485e+04			
Date:	Thu, 30 Apr 2020	Prob (F-statistic):	0.00			
Time:	14:46:06	Log-Likelihood:	17462.			
No. Observations:	16291	AIC:	-3.492e+04			
Df Residuals:	16289	BIC:	-3.490e+04			
Df Model:	1					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
Intercept	-0.0053	0.001	-7.743	0.000	-0.007	-0.004
sales_GLOBAL	0.3159	0.002	186.674	0.000	0.313	0.319
Omnibus:	16356.476	Durbin-Watson:	1.954			
Prob(Omnibus):	0.000	Jarque-Bera (JB):	44840082.720			
Skew:	3.693	Prob(JB):	0.00			
Kurtosis:	259.913	Cond. No.	2.66			

Ausgehend von den ersten Modellerstellungsansätzen stellt die Tab. 9.3 zentrale Funktionen zusammen.

Tab. 9.3 Übersicht nützlicher Funktionen für die Modellkonzeption

Funktion	Auswertung zu:
<code>statsmodels.formula.api as smf</code>	Import des Moduls statsmodels in das Python-Skript
<code>sFormula = 'Gleichung'</code>	Erstellung der Regressionsgleichung mittels String, um das Modell zu spezifizieren
<code>OLSModel = smf.ols(formula=sFormula, data=df).fit()</code>	Aufruf einer Regression mittels OLS. Mit <code>sFormula</code> wird das Gleichungssystem übergebenen. Durch <code>.fit()</code> wird das Modell auf Basis der Daten geschätzt
<code>OLSModel.summary()</code>	Erstellung der Modellzusammenfassung analog zur Konsolenausgabe in 9.14

9.5 Videospiel: Interpretation der Ergebnisse

Analog zum Data-Science-Prozess soll an dieser Stelle eine Interpretation der Ergebnisse erfolgen. Die durchgeführte Regression zeigt einen linearen Zusammenhang zwischen den Einnahmen aus dem europäischen und globalen Wirtschaftsraum. Dies ergibt sich aus dem Bestimmtheitsmaß $R - Squared \approx 0,681$. Diese Kennzahl beschreibt die Ausprägung des linearen Zusammenhangs. Das Bestimmtheitsmaß ist normiert und liegt zwischen -1 und 1 . In diesem Fall sagt das Bestimmtheitsmaß aus, dass sich etwa $68,1\%$ der Schätzer durch das Modell erklären lassen. Zudem zeigt der Korrelationskoeffizient β an, dass eine Änderung um eine Einheit bei den globalen Verkäufen den Wert der Schätzer, also den europäischen Einnahmen, mit einer Sensitivität von $0,3159$ beeinflusst. Die Prüfung der T-Statistik für den Korrelationskoeffizient β mit $\approx 0,000^{***}$ zeigt, vor dem Hintergrund der üblichen Signifikanzniveaus ($* p < ,01$; $** p < ,005$; $*** p < ,001$), dass dieser Korrelationskoeffizient signifikant ist. Darüber hinaus bietet die Modellzusammenfassung weitere Kennzahlen, für die bei weiterführendem Interesse auf die angeführte Literatur verwiesen sein soll, um den Rahmen dieses Buchs einzuhalten.

Dieses Kapitel hat gezeigt, wie sich mit nur wenigen Zeilen Quellcode komplexe Transformationen und Analysen an den Daten durchführen lassen und wie sich ein erstes einfaches statistisches Modell auf die Daten anwenden lässt, um tiefere Zusammenhänge zu begreifen. Im Speziellen kommt das `DataFrame` zum Einsatz, um die Daten nützlich zu organisieren. Innerhalb dieses Datenrahmens lassen sich durch die vordefinierten Funktionen zahlreiche Operationen ausführen, um die Daten schnell in die entsprechenden Strukturen zu bringen. Mithilfe von `seaborn` und `matplotlib` werden anschauliche Diagramme erstellt, die eine erste Analyse der Daten ermöglichen. Zudem dient das Modul `statsmodels` zur Implementierung einer einfachen Regression.



Anwendungsbeispiel: Conjoint-Analyse – Mehr als die Summe seiner Teile

10

Katharina Keller

Die Entwicklung von neuen Produkten ist wichtig für Unternehmen, da die erfolgreiche Etablierung von Innovationen zum finanziellen Erfolg und zur Stärkung gegen den Wettbewerb beitragen. Innovative Produkte und Technologien haben es aber meist besonders zu Beginn recht schwer. Für gewöhnlich durchlaufen sie nach Markteinführung einen langen Diffusionsprozess, bis sie die breite Masse erreichen (Rogers, 1962). Viele bekannte Innovationen, die sich inzwischen durchgesetzt haben, brauchten hierfür mehrere Jahrzehnte und erlebten bis zur Marktdurchdringung Höhen und Tiefen. Ein Beispiel sind E-Bikes, die bereits vor 1900 entwickelt wurden, aber erst kürzlich den Massenmarkt erreicht haben.

Die Gründe für diesen beschwerlichen Weg sind vielfältig. Ein Blick auf die bestehende Forschung zeigt, dass unter anderem ein genereller anfänglicher Widerstand der Verbraucher gegen Neuerungen im Produktsortiment ein Grund für die hohe Misserfolgsrate von Innovationen ist (Heidenreich & Kraemer, 2016). Dieser anfängliche Widerstand muss entsprechend überwunden werden, um einen erfolgreichen Weg für ein neues Produkt zu bahnen. Hierbei können Marketinginstrumente, wie die simulative Darstellung einer Nutzungsumgebung, helfen, die den Nutzern die Funktionalität eines neuen Produkts und seine Kompatibilität mit bestehenden Routinen verständlich machen (Zhao et al., 2011). Außerdem lassen sich durch Vergleiche zu bekannten Produkten die Vorteile einer Innovation hervorheben (Heidenreich & Kraemer, 2016). Nachdem die Verbraucher ihren anfänglichen Widerstand überwunden haben, bilden sie sich auf Grundlage ihres ersten Verständnisses des Produkts eine positive oder negative Meinung über die Innovation (Rogers, 1962). Produktverständnis und Bewertung des Produkts sind somit wichtige Voraussetzungen für seine Akzeptanz (Reinders et al., 2010).

K. Keller (✉)
Frankfurt am Main, Deutschland
E-mail: kakeller@wiwi.uni-frankfurt.de

Aber auch nach Erreichen einer gewissen Verbraucherakzeptanz verbreiten sich Innovationen nicht über Nacht. Stattdessen durchlaufen sie einen Diffusionsprozess, der bei frühen versus späteren Käufern unterschiedliche Adoptionszeitpunkte beinhaltet (Rogers, 1962).

Entsprechend der Diffusionstheorie stützt die Mehrheit eines Sozialsystems ihre Adoptionsentscheidungen auf die bestehende Netzwerkgröße (Song et al., 2009). Daher müssen Innovationen eine kritische Masse an Adoptierenden erreichen, um den Diffusionsprozess zu fördern. Demnach hängt der Erfolg einer Innovation von der Anzahl an Käufern, insbesondere von frühen Käufern, ab (Hinz et al., 2014). Mithilfe von externen Maßnahmen wie adäquater Infrastruktur (z. B. Ladesäulen für Elektroautos) oder finanzieller Unterstützung (z. B. staatliche Anreize wie die Umweltprämie bei Elektroautos) lässt sich die Menge an frühen Käufern erhöhen (Heidenreich et al., 2017).

Die Bewertung eines neuen Produkts und die damit einhergehende Verbraucherakzeptanz werden weitgehend von den Präferenzen der Verbraucher beeinflusst. Für den erfolgreichen Vertrieb von neuen Produkten ist es somit wichtig zu verstehen, welche Komponenten im Produkt essenziell für die potenziellen Kunden sind. Besonders im internationalen Wettbewerb mit zahlreichen ähnlichen Konkurrenzprodukten können einzelne Attribute den Unterschied zwischen Erfolg und Misserfolg ausmachen. In diesem Kontext kann die wissenschaftliche Analyse von Produktbestandteilen einen Mehrwert für die zielgerichtete Entwicklung und Produktion von neuen Produkten geben.

Dieser Umstand lässt sich exemplarisch an den Vertrieb von Elektroautos skizzieren. Der Hersteller Tesla hat durch die Fokussierung auf Batterie, Laufzeit und Design frühzeitig seine Vormachtstellung auf dem Elektroautomobilmarkt ausgebaut, sodass die etablierten Hersteller bemüht sind, den Anschluss nicht zu verlieren. Aber wie hat es Tesla geschafft, den richtigen Fokus auf die passenden Produktmerkmale zu setzen, die jetzt den entscheidenden Unterschied und somit Teslas Erfolg ausmachen? In diesem Kapitel möchten wir eine wissenschaftliche Methode zur Analyse von Nutzerpräferenzen präsentieren, mit deren Hilfe Sie beispielsweise die Wichtigkeit von unterschiedlichen Merkmalen eines Produkts oder einer Technologie analysieren können. Im Folgenden wird eine Einführung in ausgewählte Methoden gegeben 10.1. Entlang des vorgestellten Data-Science-Prozesses (Abschn. 7.2) geht dieses Kapitel zunächst auf den Datensatz (Abschn. 10.2) ein. Darauf aufbauend werden Vorverarbeitungsschritte vorgestellt (Abschn. 10.3). Im Anschluss erfolgt die Exploration der Daten (Abschn. 10.4), um das Auswertungsmodell zu präsentieren (Abschn. 10.5). Das Kapitel endet mit der Interpretation der Ergebnisse (Abschn. 10.6).

10.1 Conjoint-Analyse: Einführung in die Methodik

Für die Bestimmung und Analyse von Nutzerpräferenzen entwickelten Luce und Tukey (1964) die Conjoint-Analyse, die inzwischen eine beliebte Methode in der Psychologie und im Marketing ist. Sie erlaubt Einblicke in die Präferenzen potenzieller Nutzer, selbst wenn das betrachtete Produkt oder einzelne Merkmale noch nicht auf dem Markt existieren. Der

Conjoint-Analyse liegt die Annahme zugrunde, dass ein Produkt oder eine Technologie durch verschiedene Eigenschaften, auch Attribute genannt, charakterisiert ist, von denen jede unterschiedliche Vorteile oder Kosten für die Nutzer hat. Die einzelnen Attribute wiederum spiegeln sich in verschiedenen Merkmalsausprägungen wider. Basierend auf Skiera und Gensler (2002) lässt sich die Durchführung einer Conjoint-Analyse in fünf Schritte unterteilen: 1) Auswahl der Attribute und ihrer Merkmalsausprägungen, 2) Festlegung des Umfragedesigns, 3) Auswertung der Stimuli durch die Teilnehmer, 4) Schätzung der Nutzenfunktion und 5) Interpretation der Ergebnisse.

Die zunehmende Akzeptanz der Conjoint-Analyse lässt sich auf das breite Anwendungsspektrum der Methode zurückführen (Böhler & Scigliano, 2009). Heutzutage findet diese überwiegend im Bereich der Markt- und Konsumentenforschung Anwendung (Fiedler et al., 2017). Die Conjoint-Analyse ermöglicht die Ermittlung einer Nutzenfunktion, die die spezifischen Merkmalsausprägungen eines Produkts in Form von Konsumentenpräferenzen darstellt. Daher stellt die Analyse eine Grundlage für eine Vielzahl von Marketinganwendungen dar (Eggers et al., 2018). Zum Beispiel:

- Marktsegmentierung, z. B. Segmentierung auf der Grundlage von bevorzugten Produkteigenschaften (Teichert, 2001)
- Neuproduktplanung, z. B. Identifizierung eines Produktkonzepts, dass von den Konsumenten bevorzugt wird (Page & Rosenbaum, 1992)
- Preisgestaltung, z. B. Bestimmung der Zahlungsbereitschaft für eine Verbesserung der Produkteigenschaften (Miller et al., 2011)
- Markenbildung, z. B. Ermittlung des Werts einer Produktmarke (Sattler, 2005)
- Marktszenarien, z. B. Bestimmung der erwarteten Marktanteile für ein neues Produkt (Burmester et al., 2016)

Allgemein kann die Conjoint-Analyse angewendet werden, wenn Individuen eine Entscheidung bezüglich multiattributiver Objekte treffen müssen. Die Methode ist daher nicht auf die genannten Einsatzgebiete beschränkt und wird auch in anderen Bereichen angewandt, z. B. im Transportwesen, bei Rechtsstreitigkeiten, in der Landwirtschaft oder in der Gesundheitsökonomie.

In allen conjoint-analytischen Verfahren verbinden die Probanden einzelne Produkteigenschaften mit Nutzenerwartungen. Der Gesamtnutzen der Eigenschaften führt zu einem Gesamturteil, das sich zunächst in der Präferenz und schließlich in der Wahlentscheidung widerspiegelt (Bichler & Trommsdorff, 2009). Die Ermittlung des Gesamtnutzens erfolgt je nach Verfahren unterschiedlich.

Die klassische Conjoint-Analyse ist rankingbasiert, sodass die Präferenzen der Nutzer für jedes Attribut aus einem Rangfolgenprozess abgeleitet werden (Ranking-based Conjoint). Diese Parameterwerte können dann analysiert werden, um somit die beliebteste Kombination von Attributausprägungen und somit das beliebteste Produkt bestimmen zu können. Weiterhin existiert die Rating-based Conjoint-Analyse, die auf der Bewertung einzelner Pro-

duktkonzepte oder Produktpaare basiert. Ein großer Nachteil dieser Conjoint-Verfahren ist die starke Limitation hinsichtlich der Attributanzahl. Da Probanden lediglich eine begrenzte Menge an Informationen verarbeiten können, ohne sich überfordert zu fühlen (Raghavarao et al., 2010), werden diese Verfahren zumeist nur auf Technologien mit einer geringen Anzahl von Attributen angewandt. Darüber hinaus können mittels der Rating-based Conjoint- und der Ranking-based Conjoint-Analyse lediglich Präferenzstrukturen abgebildet werden, sie liefern jedoch keinerlei Informationen über die tatsächliche Produktauswahl der Probanden, geschweige denn über Nutzerakzeptanz oder Kaufbereitschaft. Um diese Schwachstellen zu überwinden, wurden die Conjoint-Analyseverfahren im Lauf der Zeit weiterentwickelt (Eggers et al., 2018). Beispielsweise wurden hybride Conjoint-Techniken entwickelt, die Rating- oder Rankingfragen mit Conjoint-Aufgaben kombinieren. Die verbreitetste Form ist die Adaptive Conjoint-Analyse von Johnson et al. (1987) (siehe auch Green et al., 1991).

Einen weiteren Fortschritt zur Überwindung der Schwachstellen traditioneller Conjoint-Analysen stellt die Choice-Based Conjoint-Analyse (CBC-Analyse) dar. Diese basiert im Gegensatz zu den anderen Methoden nicht auf ordinalen (Rankings) oder metrischen (Ratings) Präferenzurteilen, sondern auf Auswahlentscheidungen (Eggers et al., 2018). Die CBC ist die am häufigsten eingesetzte Variante der traditionellen Conjoint-Analyse (vgl. Hartmann & Sattler, 2002). Die auf Louviere und Woodworth (1983) zurückgehende Methode wurde entwickelt, um die externe Validität der Untersuchungsergebnisse zu verbessern. In einer CBC werden die Probanden wiederholt aufgefordert, eine hypothetische Wahl zwischen einer Reihe von Produktalternativen zu treffen, die durch ihre Attribute und die entsprechenden Attributmerkmale beschrieben werden. Dabei treffen die Befragten Abwägungsentscheidungen zwischen der Attraktivität dieser Alternativen, die wertvolle Erkenntnisse über den Beitrag jedes einzelnen Attributs zur Entscheidungsfindung liefern. Basierend auf den Auswahlentscheidungen innerhalb der Choice Sets (Auswahlmengen) können Rückschlüsse auf die Nutzenbeiträge der Produkteigenschaften abgeleitet werden, die wiederum Informationen bezüglich der Konsumentenpräferenzen liefern (Louviere & Woodworth, 1983).

10.2 Conjoint-Analyse: Datensatz und Fragestellung

Im spannenden Kontext von Elektrofahrzeugen gibt es bereits einige Präferenzanalysen für unterschiedliche Produktmerkmale. Für die Auswahl der Produktmerkmale für unsere Studie orientieren wir uns am Studiendesign von Hinz et al. (2015), wobei wir uns auf die technischen Attribute beschränken. Unser Datensatz enthält somit die folgenden Attribute: Reichweite pro Ladung, Kaufpreis, Ladedauer, Stromkosten pro 100km und Motorleistung. Die entsprechenden Attributausprägungen sind in Tab. 10.1 dargestellt. Der gesamte Datensatz ist simuliert und somit fiktiv.

Generell ist zu erwähnen, dass vor allem bei neuartigen Technologien, mit denen die Probanden bislang keinen oder nur wenig Kontakt hatten, die Anzahl der Attribute auch bei

Tab. 10.1 Übersicht
Attribute und Attributlevel

Attribut	Attributlevel
Reichweite pro Ladung	100, 175, 250, 325 km
Kaufpreis	15.000, 20.000, 25.000, 30.000 €
Ladedauer	1, 4 h
Stromkosten pro 100 km	1, 3, 5, 7 €
Motorleistung	40, 80 kW

einer CBC nicht zu groß sein sollte. Durch eine Erhöhung der Attributanzahl steigt die Komplexität der Conjoint-Analyse, sodass die Befragten höhere kognitive Leistung erbringen müssen (Eggers et al., 2018). Zu viele Informationen können zudem zu Verwirrung bzw. Überlastung der Befragten führen, da diese die Gesamtheit der Informationen nicht verarbeiten können (Raghavarao et al., 2010). In der Literatur findet sich keine einheitliche Grenzangabe, jedoch legen CBC-Experten nahe, dass die Attributanzahl ein Maximum von acht nicht überschreiten sollte. Eggers et al. (2018) setzen das Limit sogar auf sieben Attribute, um ein überschaubares Studiendesign zu gewährleisten.

Bei der Auswahl der Attribute ist darauf zu achten, dass die Attribute relevant sind für die Konsumenten, aber nicht miteinander in Beziehung stehen (Eggers et al., 2018). Außerdem sollten keine Attribute gewählt werden, die für manche Probanden ein Ausschlusskriterium darstellen, sodass einige Probanden aufgrund eines speziellen Attributs das Produkt generell nicht kaufen würden (Skiera & Gensler, 2002).

Sind nun die Attribute und die zugehörigen Attributlevel ausgewählt, so kann dieses Studiendesign in eine Online-Umfrage übertragen werden. Hierzu empfiehlt sich beispielsweise die Online-Plattform DISE (Schlereth & Skiera, 2012), die sich auf die Durchführung von Conjoint-Analysen spezialisiert hat. Im „Demonstration Survey“ geben die Autoren unter dem Punkt „Advanced data collection methods“ einen Überblick über verschiedene Datenerfassungsmethoden. Hier finden sich auch verschiedene Conjoint-Analysen. Aber auch viele andere Online-Plattformen wie beispielsweise LimeSurvey bieten sich an, eine Onlinestudie aufzusetzen.

Die Studie sollte nach einem Willkommenstext zunächst einige allgemeine Informationen über die betrachtete Technologie enthalten. Anschließend sollten alle ausgewählten Attribute und die entsprechenden Attributlevel vorgestellt und erklärt werden. Eine anschließende tabellarische Übersicht eignet sich außerdem gut, um diese Informationen für die Probanden zusammenzufassen. Je nach Komplexität des zu untersuchenden Produkts oder der Technologie, ist auch eine Verständnisfrage sinnvoll, um sicherzustellen, dass die Studienteilnehmer ausreichend vertraut sind mit der vorgestellten Technologie. Das folgende Conjoint-Experiment sollte mit einer kurzen Erklärung der Methode und einem exemplarischen Choice Set starten. Anschließend sollen die Studienteilnehmer alle Choice Sets hintereinander ausfüllen. Am Ende der Umfrage können zusätzlich Informationen zu

demografischen Daten wie Alter, Geschlecht, Wohnort etc. gesammelt werden, um weitere Einblicke in mögliche Käufergruppen zu erhalten.

Die Abb. 10.1 zeigt ein beispielhaftes Choice Set unserer Studie. In jedem Choice Set hatten unsere Befragten die Auswahl zwischen drei Optionen. Insgesamt umfasst unsere Studie 14 Choice Sets mit unterschiedlichen Produktalternativen.

10.3 Conjoint-Analyse: Preprocessing

Nach der Durchführung der Studie steht die Schätzung der Parameterwerte im Vordergrund, um die Umfrage analysieren zu können. Auf den folgenden Seiten werden wir den simulierten Datensatz in Python aufbereiten und analysieren, um Erkenntnisse über die Nutzerpräferenzen hinsichtlich der von uns gewählten Attribute im Studiendesign zu erlangen.

Hierzu werden im ersten Schritt die zwei Bibliotheken 'pandas' und 'numpy' importiert, die für die Auswertung benötigt werden. Das Codebeispiel 10.1 zeigt die Vorgehensweise im Skript.

Listing 10.1 Import der benötigten Module

```
1 # Die Libraries pandas und numpy importieren, die fuer die Auswertung benoetigt
  werden
2 import pandas as pd
3 import numpy as np
```

Nun werden die Daten aus einer Excel-Datei eingelesen (siehe Listing 10.2). Der simulierte Datensatz enthält im ersten Tabellenblatt die Auswahlentscheidungen auf individueller Ebene. Für jeden Teilnehmer gibt es somit eine Zeile, in deren Spalten die entsprechenden Auswahlentscheidungen in den einzelnen Choice Sets aufgelistet sind.

Das zweite Tabellenblatt enthält das Studiendesign, in dem die Ausprägungen der verschiedenen Attribute der jeweiligen Produktalternativen festgehalten sind. Hierbei wird pro Zeile eine Produktalternative mit ihren entsprechenden Attributausprägungen spezifiziert, sodass die ersten drei untereinander stehenden Zeilen die drei Auswahlmöglichkeiten des

1 Bitte wählen Sie das Elektrofahrzeug, für das Sie die größte Präferenz aufweisen.			
Reichweite pro Ladung	250 km	100 km	175 km
Kaufpreis	20.000 €	25.000 €	15.000 €
Ladedauer	1 Stunde	4 Stunden	1 Stunde
Stromkosten pro 100 km	5 €	3 €	1 €
Motorleistung	40 kW (~54 PS)	80 kW (~109 PS)	80 kW (~109 PS)
	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Abb. 10.1 Beispielhaftes Choice Set

ersten Choice Set bilden. In einer CBC mit drei Optionen pro Choice Set repräsentieren somit jeweils drei Zeilen die Auswahlmöglichkeiten innerhalb eines Choice Set.

Die Daten beider Tabellenblätter werden eingelesen und die ersten fünf Zeilen der jeweiligen Datensätze ausgegeben, um einen Eindruck über die entsprechenden Daten zu erhalten. Das erste Tabellenblatt mit den Auswahlentscheidungen wird als 'df_choices' gespeichert. Das zweite Tabellenblatt mit dem Studiendesign wird als 'df_study_design' gespeichert.

Listing 10.2 Einlesen der Daten

```

4  # Einlesen der Daten aus einer Excel-Datei mit zwei Tabellenblaettern
5
6  # Einlesen des Tabellenblattes 1 (Auswahlentscheidungen)
7  df_choices = pd.read_excel("Datensatz_Electric Vehicles_simuliert.xlsx", sheet_name="
    Choices")
8  # Anzeigen der ersten 5 Zeilen des ersten Tabellenblattes (Auswahlentscheidungen)
9  print(df_choices.head())
10
11 # Einlesen des Tabellenblattes 2 (Studiendesign)
12 df_study_design = pd.read_excel("Datensatz_Electric Vehicles_simuliert.xlsx",
    sheet_name="Studiendesign Choice Sets")
13 # Anzeigen der ersten 5 Zeilen des zweiten Tabellenblattes (Studiendesign)
14 print(df_study_design.head())
  
```

Für die weitere Auswertung müssen die Informationen aus den beiden Tabellenblättern zusammengeführt werden. Hierzu werden für jeden Teilnehmer pro Choice Set drei Zeilen gebildet. Bei 14 Choice Sets pro Teilnehmer ergibt das $14 * 3 = 42$ Zeilen. Jeweils drei Zeilen bilden ein Choice Set ab und sind somit hinsichtlich der Auswahlentscheidung gemeinsam zu betrachten. Aus diesem Grund wird eine Liste mit 42 Zeilen erstellt, wobei jeweils drei Zeilen als Choice Set zusammengehören. Dies zeigen auch die Werte von 'new_ChoiceSet'. Diese Liste wird als neue Spalte im zweiten Tabellenblatt 'study_design' hinzugefügt. Unser beispielhafter Datensatz umfasst 250 Teilnehmer für die jeweils 42 Zeilen angelegt werden. Hierzu wird die aktuelle Tabelle mit 42 Zeilen für weitere 249 Teilnehmer kopiert, sodass der neue Datensatz insgesamt $250 * 42 = 10.500$ Zeilen enthält. Nachfolgend wird eine weitere Spalte in den Datensatz eingefügt, die die Teilnehmer durchnummeriert und ihnen somit eine aufsteigende ID vergibt. Bei Betrachtung des Data Set wird deutlich, dass pro ID (Teilnehmer) 42 Zeilen untereinander aufgelistet sind, die die 14 Choice Sets mit ihren drei Auswahlmöglichkeiten entsprechend der Attributausprägungen spezifizieren. Die Umsetzung im Pythoncode wird im nachfolgenden Listing 10.3 gezeigt.

Listing 10.3 Zusammenführen der Daten

```

15 # Zusammenfuehrung der Daten zu einem Datensatz
16
17 # Erstellen einer Liste mit 42 Zeilen, wobei jeweils 3 Zeilen als Choice Set
    zusammengehoeeren
18 iX_ChoiceSet = range(1, 15)
19 iN_ChoiceSet = 3
  
```

```

20 new_ChoiceSet = [item for item in iX_ChoiceSet for i in range(iN_ChoiceSet)]
21 print(new_ChoiceSet)
22
23 # Hinzufuegen der Liste als neue Spalte in study_design
24 df_study_design.insert(0, "ChoiceSet", new_ChoiceSet)
25
26 # Kopieren der Tabelle, um 250 Teilnehmer abzubilden
27 df_dataset = df_study_design.append([df_study_design] * 249)
28
29 # Einfuegen einer ID Spalte
30 x_ID = range(1,251)
31 n_ID = 42
32 new_ID = [item for item in x_ID for i in range(n_ID)]
33 df_dataset.insert(0, "ID", new_ID)

```

Nachfolgend wird das erste Tabellenblatt 'choices' für die weitere Analyse vorbereitet. Hierzu wird die Spalte UserID entfernt, damit eine Liste aus den Daten erstellt werden kann. Um schließlich die Auswahlentscheidung aus dem ersten Tabellenblatt zu berücksichtigen, wird eine neue Spalte „Selected“ in den Datensatz eingefügt. Wurde eine Option innerhalb eines Choice Set ausgewählt, so wird eine 1 in diese Zeile eingefügt. Die Optionen, die in den jeweiligen Choice Sets nicht ausgewählt wurden, werden mit 0 markiert – siehe [10.4](#).

Listing 10.4 Vorbereitung der Daten

```

34 # Entfernen der Spalte UserID
35 df_choices_new = df_choices.drop(columns=['UserID'])
36 df_choices_new.head()
37
38 # Erstellen einer Liste aus dem Datensatz 'choices'
39 LdfRows = df_choices_new.to_numpy().tolist()
40
41 # Erstellen einer flachen Liste aus der Liste der Listen
42 import operator
43 from functools import reduce
44 LRows2 = reduce(operator.concat, LdfRows)
45
46 # Liste jeweils 3mal wiederholen
47 S_repeated = pd.Series(LRows2)
48 S_repeated = S_repeated.repeat(3)
49
50 # Serie in Liste konvertieren
51 LRepeatedList = S_repeated.to_numpy().tolist()
52 df_dataset.insert(0, "Choice", LRepeatedList)
53
54 # Liste mit drei Optionen (pro Choice Set) und 3500 mal wieder holen (3*3500 =10500)
55 def duplicate(testList, n):
56     return testList*n
57 LChoiceOptions = [1, 2, 3]
58 LChoice_Option = duplicate(LChoiceOptions, 3500)

```

```

59
60 # Neue Spalte im Datensatz fuer ChoiceOption
61 df_dataset.insert(0, "ChoiceOption", LChoiceOption)
62
63 # Neue Spalte fuer: Auswahl (selected=1), die anderen beiden Optionen wurden nicht
    ausgewaehlt (selected=0)
64 df_dataset['selected'] = np.where((df_dataset['ChoiceOption']
65                                   == df_dataset['Choice']), 1, 0)
66
67 # Uebersicht Datensatz mit gewaehlter Anordnung der Spalten
68 df_dataset = df_dataset[["ID", "ChoiceSet", "ChoiceOption", "Choice", "selected", "
    Reichweite", "Kaufpreis", "Ladedauer", "Stromkosten", "Motorleistung"]]
69 # Ausgabe Datensatz
70 print(df_dataset)

```

10.4 Conjoint-Analyse: Explore the Data

Für die folgende Analyse extrahieren wir die Spalte 'selected', die die Auswahlentscheidungen der Teilnehmer beinhaltet und die in unserer Auswertung die endogene Variable darstellt.

Außerdem erstellen wir einen Datensatz, der nur die Attributspalten und somit die Produktausprägungen der Studie beinhaltet.

Der Skriptabschnitt 10.5 zeigt die zugehörige Umsetzung im Code.

Listing 10.5 Endogene und exogene Variable trennen

```

71 # Spalte 'selected' in 'endog' Dataset abspeichern
72 S_endog = df_dataset[["selected"]]
73
74 # Die Spalten der Attribute in neuem Datensatz abspeichern
75 df_x = df_dataset[["Reichweite", "Kaufpreis", "Ladedauer", "Stromkosten", "
    Motorleistung"]]

```

Nun wird für jedes Attributlevel eine Spalte erstellt und mit binärer Dummy-Codierung kenntlich gemacht, ob dieses Attributlevel in der entsprechenden Produktalternative vorhanden war (1) oder nicht (0), siehe 10.6.

Listing 10.6 Dummycodierung

```

76 # Dummy-Codierung
77 df_xdum = pd.get_dummies(df_x, columns=[c for c in df_x.columns])
78
79 # Anzeigen der ersten 5 Zeilen von xdum
80 print(df_xdum.head())

```

Im folgenden Skriptabschnitt 10.7 wandeln wir die dummy-codierten Attribute in eine Effektcodierung um. Hierfür wird jeweils ein Attributlevel (in unserem Fall jeweils das

letzte Attributlevel) durch die anderen Attributlevel dargestellt und deshalb aus dem Datensatz entfernt. Es dient somit als Referenzlevel und wird nicht geschätzt. Wenn eines der anderen Attributlevel in einer Produktalternative gezeigt wird, so wird die jeweilige Dummyvariable auf 1 gesetzt, die anderen auf 0. Wird das Referenzlevel gezeigt, so werden alle Dummyvariablen dieses Attributs auf -1 gesetzt. Der effektcodierte Datensatz dient als exogener Input für die anstehende Schätzung der Parameterwerte.

Listing 10.7 Effektcodierung

```

81 # Effektkodierung
82 df_effect_dum = pd.DataFrame()
83 for row in df_xdum.to_dict(orient="records"):
84     if row["Reichweite_4"] == 1:
85         row["Reichweite_1"] = -1
86         row["Reichweite_2"] = -1
87         row["Reichweite_3"] = -1
88     row.pop("Reichweite_4", None)
89     if row["Kaufpreis_4"] == 1:
90         row["Kaufpreis_1"] = -1
91         row["Kaufpreis_2"] = -1
92         row["Kaufpreis_3"] = -1
93     row.pop("Kaufpreis_4", None)
94     if row["Ladedauer_2"] == 1:
95         row["Ladedauer_1"] = -1
96     row.pop("Ladedauer_2", None)
97     if row["Stromkosten_4"] == 1:
98         row["Stromkosten_1"] = -1
99         row["Stromkosten_2"] = -1
100        row["Stromkosten_3"] = -1
101    row.pop("Stromkosten_4", None)
102    if row["Motorleistung_2"] == 1:
103        row["Motorleistung_1"] = -1
104    row.pop("Motorleistung_2", None)
105    df_effect_dum = df_effect_dum.append(row, ignore_index=True)
106
107 # Umbenennung aus Ausgabe der exogenen Variablen
108 exog = df_effect_dum
109 print(exog)

```

Im Fall des Attributs Kaufpreis dient das letzte Attributlevel Kaufpreis_4 als Referenzlevel. Somit wird die Effektcodierung durch die Attributlevel Kaufpreis_1, Kaufpreis_2 und Kaufpreis_3 dargestellt. Enthält eine Produktalternative den ersten Kaufpreis, so wird die Dummy-Variable Kaufpreis_1 auf 1 gesetzt, die anderen beiden Dummy-Variablen Kaufpreis_2 und Kaufpreis_3 auf 0. Enthält die Alternative jedoch das vierte Level des Kaufpreises, so werden alle Dummy-Variablen Kaufpreis_1, Kaufpreis_2 und Kaufpreis_3 auf -1 gesetzt. In der folgenden Schätzung werden somit nur die Parameterwerte für die Dummy-

Variablen in der Effektcodierung geschätzt. Der Parameterwert des Referenzlevels ergibt sich dann aus der negativen Summe aller Parameterwerte des entsprechenden Attributs.

10.5 Conjoint-Analyse: Model the Data

Die Conjoint-Analyse geht von einem nutzenmaximierenden Befragten aus und stützt sich auf die Random Utility Theory (Thurstone, 1927). Sie nimmt an, dass der Befragte i die Alternative j wählt, die seinen Nutzen maximiert.

Dabei setzt sich der Nutzen $u_{i,j}$ des Befragten i für die Produktalternative j aus einem deterministischen Teil $v_{i,j}$ und einem stochastischen Teil $\epsilon_{i,j}$ zusammen: $u_{i,j} = v_{i,j} + \epsilon_{i,j}$. Der deterministische Teil $v_{i,j}$ enthält beobachtbare Informationen, wie z. B. die in den Choice Sets gezeigten Attribute und Attributlevel, die experimentell manipuliert werden. Der stochastische Teil $\epsilon_{i,j}$ enthält unbeobachtbares Verhalten, für das eine Extremwertverteilung angenommen wird, die eine ähnliche funktionale Form wie die Normalverteilung hat und die zu einer geschlossenen Berechnung der Auswahlwahrscheinlichkeiten für konkrete Attribute und Ebenen eines Elektrofahrzeugs j führt (Train, 2009).

Der Nutzen eines Produkts setzt sich aus den Teilnutzenwerten der entsprechenden Konfiguration zusammen. Entsprechend wird ein additives Modell für den Gesamtnutzen einer Produktalternative verwendet, d. h. $v_{i,j} = \beta_i' X_j$, wobei β_i ein Vektor der Präferenzen des Befragten i für alle Attribute und daher konstant ist, während der Vektor X_j die Attributlevel jedes Attributs im Produkt j angibt.

Weitergehende Informationen zur Erstellung der Nutzenfunktion und vor allem zur Schätzung der Nutzenparameter finden sich in Gensler (2002). In diesem Beitrag werden ausführlich und nachvollziehbar sowohl das multinomiale Logit-Modell als auch die Likelihood-Funktion hergeleitet und beschrieben.

Für die Schätzung der Parameterwerte nutzen wir ein frei zugängliches Open-Source-Modul namens `'statsmodels.discrete.discrete_model.MNLogit'`.

Die Modellierung der Auswahlwahrscheinlichkeit der einzelnen Produktalternativen innerhalb eines Choice Set basiert auf dem multinomialen Logit-Modell (MNL) und wird im Sourcecode wie folgt angegeben:

$$\frac{\exp(\beta_j' x_i)}{\sum_{k=0}^J \exp(\beta_k' x_i)} \quad (10.1)$$

- j die Anzahl der Wahlmöglichkeiten für die endogene Variable
- k die tatsächliche Anzahl der Parameter für das exogene Design
- β_j' Parameterwert der j -ten Wahlmöglichkeit
- x_i Ausprägungen der Attributlevel für Produkt i

Da die Schätzung der Parameterwerte mithilfe der OLS-Schätzung ineffizient wäre (siehe Gensler, 2006), wird zur Modellierung die Maximum-Likelihood-Schätzung verwendet.

In dem von uns verwendeten Modul wird diese Modellierung wie folgt beschrieben:

$$\ln L = \sum_{i=1}^n \sum_{j=0}^J d_{ij} \ln \left(\frac{\exp(\beta'_j x_i)}{\sum_{k=0}^J \exp(\beta'_k x_i)} \right) \quad (10.2)$$

wobei

- $d_{ij} = 1$, wenn der Teilnehmer i die Alternative j auswählt, ansonsten 0

Mit dem Modul `'statsmodels.discrete.discrete_model.MNLogit'` schätzen wir nun die Parameterwerte auf Basis der Auswahlentscheidungen als endogene Variable und den Attributausprägungen der einzelnen Auswahloptionen als exogene Variablen, siehe 10.8. In der Ausgabe erscheint die Zusammenfassung der Schätzung. Neben den Parameternamen und den Parameterwerten (als `'coef'` bezeichnet), kann man in der fünften Spalte auch die p-Werte ablesen und somit Rückschlüsse auf die Signifikanz der Schätzung ziehen.

Listing 10.8 Schätzung der Parameterwerte

```

110 # Schätzung der Parameterwerte
111 import statsmodels.discrete.discrete_model
112 S_endog = S_endog.reset_index(drop=True)
113 res = statsmodels.discrete.discrete_model.MNLogit(S_endog, exog).fit(maxiter=10000)
114 print(res.summary())

```

Im nächsten Schritt werden die Parameternamen und die zugehörigen Parameterwerte aus der Schätzung in einem Datensatz gespeichert. Die Parameter werden zudem in `ndarrays` gespeichert, die zu Listen konvertiert werden und dann mit der Funktion `.flatten()` auf eine 1D-Dimension reduziert werden. Die entsprechende Umsetzung findet sich im Skript 10.9.

Listing 10.9 Speichern der Ergebnisse

```

115 # Speichern der Parameternamen und -werte in einem Datensatz
116 df_modelValues = pd.DataFrame(columns=['param_name', 'param_w'])
117
118 # Speichern der Parameter in ndarrays
119 LVariables = res.params.index.values
120 LParamValues = res.params.values
121
122 # ndarrays in Listen konvertieren
123 # Mit .flatten() wird die Dimension von 2D auf 1D reduziert
124 LVariables = LVariables.tolist()
125 LParamValues = LParamValues.flatten()
126
127 dValue = LParamValues[0]

```

Danach wird, wie in 10.10 gezeigt, über die Modelldaten iteriert und die Werte im Datensatz werden hinzugefügt.

Listing 10.10 Iteration

```

128 # Iteration ueber die Modelldaten
129 for iIndex in range(len(LVariables)):
130     sVariable = LVariables[iIndex]
131     dValue = LParamValues[iIndex]
132     print(sVariable, str(dValue))
133
134 # Hinzufuegen der Werte im Datensatz
135 df_modelValues = df_modelValues.append({
136     'param_name': sVariable,
137     'param_w': dValue}, ignore_index=True)
138
139 df_res = df_modelValues
  
```

Nachfolgend werden die fehlenden Parameterwerte berechnet und in einem Dataframe abgespeichert – siehe 10.11. Die Parameterwerte der Referenzlevel ergeben sich aus der negativen Summe aller Parameterwerte des entsprechenden Attributs. Alle Parameterwerte werden gemeinsam in einem DataFrame gespeichert und entsprechend der Parameternamen sortiert. Die Summe der jeweiligen Parameterwerte eines Attributs ergibt entsprechend der Effektcodierung 0.

Listing 10.11 Berechnung der fehlenden Werte

```

133 # Berechnung der fehlenden Werte (Referenzlevel)
134 df_ommitted_params = pd.DataFrame({'param_name':
135     ['Kaufpreis_4', 'Ladedauer_2', 'Motorleistung_2',
136     'Reichweite_4', 'Stromkosten_4'],
137     'param_w':
138     [-(df_res.param_w[0] + df_res.param_w[1] + df_res.param_w[2]),
139     -(df_res.param_w[3]),
140     -(df_res.param_w[4]),
141     -(df_res.param_w[5] + df_res.param_w[6] + df_res.param_w[7]),
142     -(df_res.param_w[8] + df_res.param_w[9] + df_res.param_w[10])]
143     },
144     columns=['param_name', 'param_w'])
145
146 # Erstellen eines gemeinsamen Dataframes fuer alle Parameterwerte
147 df_all = pd.concat([df_res, df_ommitted_params], ignore_index=True)
148
149 # Sortierung des Dataframes entsprechend der Parameternamen
150 df_all = df_all.sort_values(by='param_name', ascending=True)
151
152 print(df_all)
  
```

10.6 Conjoint-Analyse: Interpretation der Ergebnisse

Für die weitere Analyse wird die Spalte mit dem Parameternamen als Index gesetzt, um entsprechend des Namens die einzelnen Parameterwerte in einem Array speichern zu können. Die Parameterwerte werden gruppenweise gespeichert, sodass pro Attribut eine Liste mit den jeweiligen Attributlevels erstellt wird.

Mit `np.max` und `np.min` wird der Maximalwert bzw. der Minimalwert der jeweiligen Liste bestimmt. Die Spannweite eines Attributs wird über die Differenz dieser beiden Werte berechnet und ausgegeben. Außerdem wird die Summe aller Spannweiten berechnet und ausgegeben. Der zugehörige Code wird in [10.12](#) dargestellt.

Listing 10.12 Berechnung der Spannweiten

```

153 # Spalte Param_name als Index setzen
154 df_all.index = df_all['param_name']
155
156 # Die Parameterwerte der Attribute werden gruppenweise in einem Array gespeichert
157 ndarray_A = np.array([[df_all["param_w"]["Kaufpreis_1"],
158                        df_all["param_w"]["Kaufpreis_2"],
159                        df_all["param_w"]["Kaufpreis_3"],
160                        df_all["param_w"]["Kaufpreis_4"]],
161                       [df_all["param_w"]["Ladedauer_1"],
162                        df_all["param_w"]["Ladedauer_2"]],
163                       [df_all["param_w"]["Motorleistung_1"],
164                        df_all["param_w"]["Motorleistung_2"]],
165                       [df_all["param_w"]["Reichweite_1"],
166                        df_all["param_w"]["Reichweite_2"],
167                        df_all["param_w"]["Reichweite_3"],
168                        df_all["param_w"]["Reichweite_4"]],
169                       [df_all["param_w"]["Stromkosten_1"],
170                        df_all["param_w"]["Stromkosten_2"],
171                        df_all["param_w"]["Stromkosten_3"],
172                        df_all["param_w"]["Stromkosten_4"]]]
173                       ])
174
175 # Berechnung der Spannweiten der einzelnen Attribute
176 dSpannw_kaufpreis = np.max(ndarray_A[0]) - np.min(ndarray_A[0])
177 dSpannw_ladedauer = np.max(ndarray_A[1]) - np.min(ndarray_A[1])
178 dSpannw_motorleistung = np.max(ndarray_A[2]) - np.min(ndarray_A[2])
179 dSpannw_reichweite = np.max(ndarray_A[3]) - np.min(ndarray_A[3])
180 dSpannw_stromkosten = np.max(ndarray_A[4]) - np.min(ndarray_A[4])
181
182 # Ausgabe der Spannweiten
183 print(dSpannw_kaufpreis)
184 print(dSpannw_ladedauer)
185 print(dSpannw_motorleistung)
186 print(dSpannw_reichweite)
187 print(dSpannw_stromkosten)

```



```

188
189 # Berechnung der Summe alle Spannweiten
190 dSpannw_summe = dSpannw_kaufpreis + dSpannw_ladedauer + dSpannw_motorleistung +
    dSpannw_reichweite + dSpannw_stromkosten
191 print(dSpannw_summe)

```

Zuletzt werden in 10.13 die Bedeutungsgewichte der einzelnen Attribute kalkuliert. Hierzu wird für jedes Attribut die Spannweite durch die Summe der Spannweiten aller Attribute geteilt. Durch das Multiplizieren mit 100 erhalten wir den Wert als Prozentzahl, die wir auf zwei Dezimalstellen runden und ausgeben.

Listing 10.13 Berechnung der Bedeutungsgewichte

```

141 # Bestimmung der Bedeutungsgewichte (in %) aus den Spannweiten
142 dIW_kaufpreis = (round(100 * (dSpannw_kaufpreis / dSpannw_summe), 2))
143 dIW_ladedauer = (round(100 * (dSpannw_ladedauer / dSpannw_summe), 2))
144 dIW_motorleistung = (round(100 * (dSpannw_motorleistung / dSpannw_summe), 2))
145 dIW_reichweite = (round(100 * (dSpannw_reichweite / dSpannw_summe), 2))
146 dIW_stromkosten = (round(100 * (dSpannw_stromkosten / dSpannw_summe), 2))
147
148 # Ausgabe der Bedeutungsgewichte
149 print('Kaufpreis: ' + str(dIW_kaufpreis) + ' %')
150 print('Ladedauer: ' + str(dIW_ladedauer) + ' %')
151 print('Motorleistung: ' + str(dIW_motorleistung) + ' %')
152 print('Reichweite: ' + str(dIW_reichweite) + ' %')
153 print('Stromkosten: ' + str(dIW_stromkosten) + ' %')

```

Listing 10.14 zeigt die Ausgabe der Bedeutungsgewichte und somit die Ergebnisse unserer Analyse.

Listing 10.14 Konsolenausgabe: Die Bedeutungsgewichte

```

1      Kaufpreis : 17.1 %
2      Ladedauer : 9.09 %
3      Motorleistung : 16.26 %
4      Reichweite : 30.62 %
5      Stromkosten : 26.92 %

```

Um die unterschiedlichen Bedeutungsgewichte zu visualisieren, plotten wir abschließend noch die Ergebnisse, wie in Listing 10.15 gezeigt. Hierzu importieren wir das Modul `matplotlib.pyplot` und erstellen ein Säulendiagramm mit den Attributnamen auf der x-Achse und den Bedeutungsgewichten auf der y-Achse.

Listing 10.15 Plotten der Ergebnisse

```

154 # Plotten der Bedeutungsgewichte als Saeulendiagramm
155 # Library matplotlib importieren
156 import matplotlib.pyplot as plt
157
158 # Saeulendiagramm erstellen

```

```

159 fig, ax = plt.subplots(figsize=(10,9))
160 BarChart = plt.bar(['Kaufpreis', 'Ladedauer', 'Motorleistung', 'Reichweite', 'Stromkosten'
161                    ],
162                    [dIW_kaufpreis, dIW_ladedauer, dIW_motorleistung, dIW_reichweite, dIW_stromkosten
163                    ])
164 plt.title('Bedeutungsgewichte')
165 y_value=['{:,.2f}'.format(x) + '%' for x in ax.get_yticks()]
166 ax.set_yticklabels(y_value)
167
168 for bar in BarChart:
169     percentage = '{:.2f}%'.format(bar.get_height())
170     x = bar.get_x() + bar.get_width() - 0.57
171     y = bar.get_y() + bar.get_height() + 0.3
172     ax.annotate(percentage, (x, y))
173
174 # Ausgabe des Saulendiagramms
175 plt.show()

```

Das Ergebnis unseres Plottens zeigt sich in Abb. 10.2. Anhand der vertikalen Säulen lässt sich bereits gut erkennen, welche Attribute einen großen Einfluss auf die Nutzerakzeptanz haben.

Die Ergebnisse unserer Auswertung zeigen, dass die Reichweite mit einem Bedeutungsgewicht von 30.63 % der wichtigste Einflussfaktor beim Kauf eines Elektrofahrzeugs ist, dicht gefolgt von den Stromkosten mit immerhin 26.92 %. Diese beiden Attribute sind somit wichtige Treiber hinsichtlich der Nutzerakzeptanz und beeinflussen somit auch stark die Kaufabsicht der potenziellen Nutzer. Die Autohersteller in der Elektromobilbranche sollten also vor allem die technologischen Voraussetzungen für eine größere Reichweite schaffen. Die Stromkosten sind aus Herstellersicht eine exogene Variable und somit nicht beeinflussbar. Interessant ist aber auch, dass Kaufpreis und Motorleistung wichtiger sind als die Ladedauer. Vermutlich planen die meisten potenziellen Nutzer, ihre Elektrofahrzeuge zu Hause (z. B. über Nacht) oder bei der Arbeit zu laden – also überall, wo sie sich mehrere Stunden aufhalten und somit nicht zu sehr auf kurze Ladezeiten angewiesen sind.

Das Ziel der hier dargestellten Auswertung ist die Vorstellung der CBC-Methode zur Erhebung und Analyse von Nutzerpräferenzen, insbesondere für innovative Produkte und Technologien. Anhand des frei zugänglichen Moduls 'statsmodels.discrete.discrete_model.MNLogit' und mittels eines simulierten Datensatzes wurde exemplarisch gezeigt, wie eine solche CBC unter Verwendung von Python ausgewertet werden kann.

An dieser Stelle ist zu erwähnen, dass die hier gezeigte Auswertungsmethodik einen ersten Eindruck über die zugrunde liegenden Nutzerpräferenzen bieten kann. Nichtsdestotrotz bergen solche Open-Source-Module die Gefahr, dass die dem Modul zugrunde liegende Schätzung nicht exakt ist oder Fehler in der Implementierung vorliegen. Entsprechend sollten im industriellen Betrieb die Module sowie die eigene Implementierung sorgfältig zum

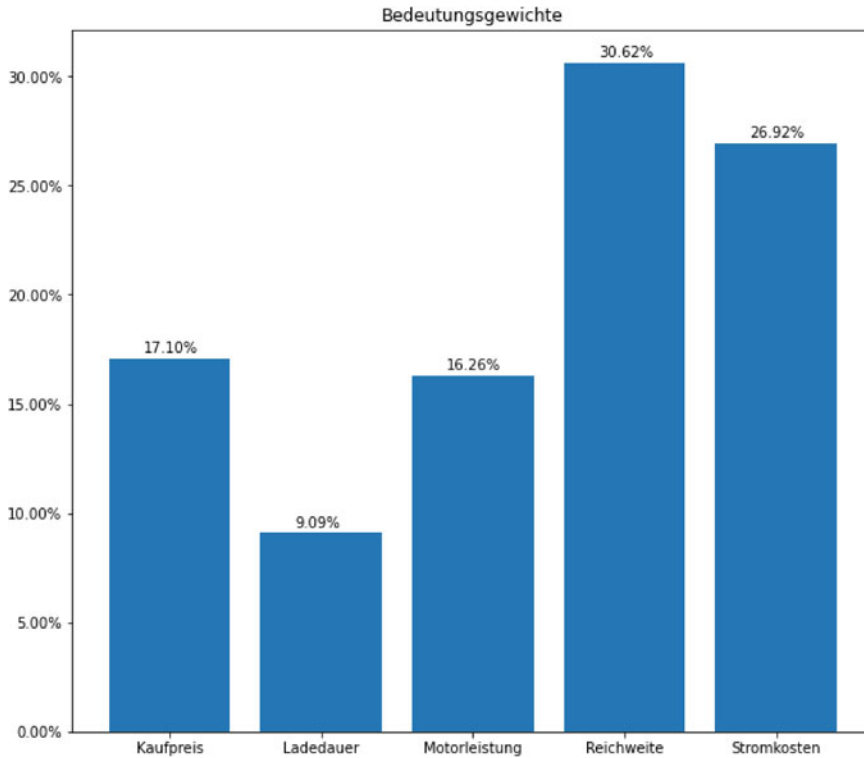


Abb. 10.2 Plot Bedeutungsgewichte

Zweck der Qualitätssicherung überprüft werden, um solche Fehlerquellen auszuschließen. Hierzu ist es ratsam, den Sourcecode eingehend zu studieren und zu kontrollieren. Diese detaillierte Kontrolle können wir für das von uns verwendete Open-Source-Modul nicht gewährleisten.

Die Auswertung von CBC-Analysen ist nicht trivial, vor allem wenn man noch eine Nichtkaufoption in die Choice Sets implementiert, um die Kaufentscheidung realistischer zu machen. Aus diesem Grund bieten verschiedene Unternehmen kommerzielle Software zur Durchführung und Auswertung von Conjoint-Analysen an. Besonders bekannt in diesem Zusammenhang ist das Computer-Software-Unternehmen Sawtooth, das Softwarelösungen für verschiedene Conjoint-Methoden im Portfolio hat. Die Schätzung erfolgt über Hierarchical Bayes und folgt somit der Vorgehensweise der aktuellen Literatur in diesem Bereich. Mit dieser Schätzmethode können die Parameterwerte mit einer höheren Qualität geschätzt werden. Außerdem ist eine Schätzung auf individueller Ebene möglich, sodass für jeden Befragten einzeln die Nutzenwerte bestimmt werden. Dies ermöglicht eine spezifischere Analyse der Käufersegmente, die für Unternehmen von großem Interesse sein kann. Beispielsweise sind Präferenzunterschiede für unterschiedliche Altersgruppen, Wohnorte

oder andere demografische Charakteristiken denkbar, auf die Unternehmen mit maßgeschneiderten Produktoptionen reagieren und somit ihre Produktpalette strategisch erweitern können. Hinzu kommt, dass auch Kaufbereitschaften mittels CBC ermittelt werden können, die wiederum für unterschiedliche Käufergruppen variieren können. Somit lassen sich auch Preisstrategien mittels CBC entwickeln. Die kommerziell angebotenen Softwares sind jedoch recht teuer. Die Kosten hierfür lohnen sich, wenn vorher bereits Vorstudien durchgeführt wurden und beispielsweise Managemententscheidungen auf den Studienergebnissen getroffen werden sollen. Eine Implementierung der Schätzung mit Hierarchical Bayes ist bislang in Python noch nicht verfügbar. Für die Auswertung der Vorstudien bietet sich aber die in diesem Kapitel präsentierte kostenfreie Auswertung mit Python an. Diese erlaubt ein schnelles Experimentieren und eine flexible Anpassung an neue Fragestellungen. Außerdem ist die Auswertung durch den einsehbaren Quellcode von Open-Source-Modulen gut nachvollziehbar und entsprechend auch individualisierbar.

Teil IV

Kunden- und soziale Medienanalyse



Benjamin M. Abdel-Karim

Die Soziale-Netzwerk-Analyse (SNA) ist heute ein Teil der zentralen Geschäftsfelder von Facebook, Instagram und Twitter. Der Ursprung dieser Analysetechnik hat seinen Ursprung in verschiedenen Forschungsrichtungen und ist damit ein interdisziplinäres Forschungsfeld, das sich im Kern auf die Bereiche Informatik und Sozial- und Geisteswissenschaften stützt. Die Daten der modernen sozialen Medien sind insbesondere für Forscher ideal, um die theoretischen und praktischen Ansätze der ersten Experimente, wie beispielsweise das Small-world-Experiment von Milgram (1967), zu prüfen. Im Rahmen dieses Anwendungsbeispiels wird ein synthetischer Datensatz der Buchreihe „Game of Thrones“ verwendet, die als gleichnamige Serie zu einem berühmten Fantasy-Epos avanciert ist. Vor diesem Hintergrund werden zur Analyse dieser Sozialen-Netzwerk-Daten die vorgestellten Prozessschritte von Data Science (Abschn. 7.2) durchgeführt. Hierbei werden zunächst der Datensatz (Abschn. 11.1) und die abgeleiteten Fragestellungen vorgestellt. Daran schließt der Datenbereinigungsschritt (Abschn. 11.2) an. Ein besonderer Teil in diesem Anwendungsbeispiel nimmt die Datenanalyse (Abschn. 11.3) ein. Analog dazu wird ein einfaches logistisches Regressionsmodell konzipiert und angewendet. Abschließend erfolgt die Interpretation (Abschn. 11.5) der Ergebnisse. Beim Erarbeiten der einzelnen Schritte werden das Python-Skript implementiert und zentrale Aspekte dieser Implementierungslösung besprochen. Das komplette Skript findet sich, wie gewohnt, in den Online-Listings des Buchs.

B. M. Abdel-Karim (✉)
Frankfurt am Main, Deutschland
E-mail: BenjaminM.Abdel-Karim@gmx.de

11.1 Soziales Netzwerk: Datensatz und Fragestellung

Für das vorliegende Anwendungsbeispiel kommen, analog zu einigen anderen Anwendungsbeispielen, synthetisch generierte Daten zum Einsatz. Dies liegt in den oftmals strengen Datenschutz- und Inhaberrechten begründet. Vor diesem Hintergrund sind die hier vorgestellten Daten künstlich erstellt und lassen sich nur bedingt mit Datensätzen aus der realen Welt vergleichen. Die hier verwendeten Daten dienen als Hilfe zur Veranschaulichung der Vorgehensweise in diesem Anwendungsbeispiel.

Der Datensatz für dieses Anwendungsbeispiel ist in der Datei `dataset_GoT.csv` zu finden. Diese Datei ist eine .CSV-Datei. Für eine detaillierte Darstellung des .CSV-Formats sei auf das vorangegangene Kapitel (Abschn. 9.1) verwiesen. Im Kern erfasst der Datensatz die Kontakte zwischen unterschiedlichen Charakteren in der Buchreihe „Game of Thrones“. Hierbei wird erfasst, wenn eine beliebige Person A mit einer anderen beliebigen Person B in einen Dialog getreten ist. Der vorliegende Datensatz besteht aus vier Spalten. Diese Spalten lassen sich wie folgt kurz beschreiben:

- `'book_id'`: Die Buchnummer, in dem die Kontakte gemessen worden sind
- `'source'`: Der Ausgangspunkt des Kontakts in Form der Person
- `'target'`: Das Ziel der Kontaktaufnahme in Form der Person
- `'contact'`: Die Häufigkeit der Kontakte im Buch

Auf Basis dieser Informationen über den Datensatz lassen sich folgende exemplarische Fragestellungen für eine tiefere Analyse ableiten.

- Welche Person hat die meisten messbaren Kontakte auf der Grundlage der Bücher?
- Welche Person ist für den Informationsfluss besonders wichtig?
- Lässt sich mit den Daten ein Modell entwickeln, das in der Lage ist, zu identifizieren, welche Person eher eine gute Rolle einnimmt und welcher Charakter eher eine böse Rolle einnimmt?

Wie angesprochen dient der Datensatz zur Illustration der grundlegenden technischen Möglichkeiten. Übertragen auf eine Unternehmung könnten vergleichbare Daten beispielsweise die Anzahl der E-Mail- oder Telefonkontakte von Mitarbeitern und Kunden sein. Eine solche Betrachtung kann dabei helfen, herauszufinden, welche Personen maßgebliche Informationsknoten sind, und damit eine bedeutende Rolle für den Informationsfluss einnehmen.

Ausgehend von diesen ersten Erkenntnissen über die Daten und den gebildeten Fragestellungen kann das Data-Science-Vorhaben starten. Zu diesem Zweck wird in einem neuen Ordner auf dem Computer mit dem Namen `bGameOfNetwork` die .CSV-Datei `dataset_GoT.csv` abgelegt und das Projekt über PyCharm gestartet. In dem Ordner wird ein neues Python-File mit der Benennung `b_bGameOfNetwork.py` erstellt. Vor diesem

Hintergrund stehen alle Dokumente in einem Projektordner zur Verfügung. Mit dieser Vorbereitung geht es an den nächsten Arbeitsschritt.

11.2 Soziales Netzwerk: Pre-processing

Im Zuge des weiteren Vorgehens sollte zunächst eine passende Quellcodedokumentation angelegt werden. Nach dem Erstellen der Erklärungen erfolgt der Import notwendiger Module. In Anlehnung an das vorherige Projekt aus Abschn. 9.1 und mit Ausblick auf das Projekt werden die Module `pandas`, `matplotlib`, `networkx` durch entsprechende Importbefehle für das Skript zugänglich gemacht. Das Codebeispiel 11.1 zeigt die Vorgehensweise im Skript.

Listing 11.1 Import der Module für die Soziale-Netzwerk-Analyse

```

1 # Anwendungsbeispiel Videospiele
2 # @author: Benjamin M. Abdel-Karim
3 # @since: 2020-05-01
4 # @version: 2020-05-201 - V1
5 # Imports
6 import pandas as pd
7 import matplotlib.pyplot as plt
8 import networkx as nx

```

Auf Basis des Codebeispiels 11.1 im aktuellen Skript `b_bGameOfNetwork.py` zeigt sich die Implementierung. Neben den bereits kennengelernten Modulen `pandas` und `matplotlib` bietet das Modul `networkx` notwendige vordefinierte Funktionen zur Erstellung von Graphen und entsprechende Analysewerkzeuge. Hierbei handelt es sich um ein Python-Modul für die Generierung, Manipulation und Analyse von Netzwerken und ihren Strukturen. Das Modul wird durch den Aufruf `import networkx` und der Zugriffsabkürzung `as nx` skriptweit aktiviert. In Abhängigkeit von den Spezifikationen des Benutzercomputers muss das Paket nachinstalliert werden. Die Installation von zusätzlichen Modulen wurde in den vorangegangenen Kapiteln bereits erläutert.

Im Anschluss an die Modulvorbereitung folgen der Datenimport sowie die Analyse der Daten. Das Codebeispiel 11.2 illustriert dabei die Vorgehensweise.

Listing 11.2 Import der Daten für die Soziale-Netzwerk-Analyse

```

11 # -----
12 # Import und Pre-Processing
13 # -----
14 # Import der .CSV Datei als DataFrame.
15 df = pd.read_csv('dataset_GoT.csv', sep=',')
16

```

```

17 # Einige Statistiken ueber die Daten.
18 # Anzahl der Spalten und Informationen ueber die
19 # Spaltenbezeichnung
20 # Infos zu den dtypes, Nicht-Null-Werte und Speicherverbrauch.
21 print('Anzahl der Spalten:' + ' ', len(df.columns))
22 print('Variablen:' + ' ', df.columns.to_list())
23 print(df.info())
24
25 # Summe der Not-a-Number Werte im Datensatz.

```

Auf Basis des Codebeispiels [11.2](#) wird mithilfe des pandas-Moduls aus der .CSV-Datei `dataset_GoT.csv` ein DataFrame erstellt und in die Variable `df` gespeichert. Durch den optionalen Parameter `sep=', '` wird das Komma als Trennzeichen zwischen den einzelnen Spalten definiert. Anschließend erfolgt ein erster Überblick über das erstellte DataFrame und seinen Daten. Die Funktion `len(df.columns)` gibt die Anzahl der vorhandenen Spalten auf der Konsole aus. Die Abweichung zu den oben angegebenen Variablen liegt darin begründet, dass das DataFrame über einen eigenen Index verfügt, sodass zu den eigentlichen Variablen noch eine Indexvariable hinzukommt. Die `print`-Funktion gibt die Informationen ergänzt um einen Erklärungsstring auf der Konsole aus. Hierbei wird zum Erklärungsstring `'Anzahl der Spalten:'` ein zusätzlicher leerer String durch `'+'` konkateniert, um auf der Konsole ein zusätzliches Leerzeichen anzuzeigen. Das Ergebnis zeigt die Konsolenausgabe [11.3](#).

Listing 11.3 Konsolenausgabe: `len(df.columns)`

```

1 Anzahl der Spalten : 5

```

Mithilfe von `df.columns` und der angehängten Funktion `to_list()` werden die Spaltenbeschriftungen ausgegeben und als Liste konvertiert, sodass sie sich auf der Konsole mit einer entsprechenden `print`-Funktion auf der Konsole ausgeben lassen. Das Ergebnis zeigt die Konsolenausgabe [11.4](#).

Listing 11.4 Konsolenausgabe: `df.columns.to_list()`

```

1 Variablen:  ['index', 'book_id', 'source', 'target', 'contact']

```

Mithilfe der Funktion `df.info()` werden detaillierte Informationen zu den einzelnen Variablen und Datentypen ausgegeben. Das Ergebnis zeigt die Konsolenausgabe [11.5](#).

Listing 11.5 Konsolenausgabe: `df.info()`

```

1 RangeIndex: 5238 entries, 0 to 5237
2 Data columns (total 5 columns):
3 #   Column  Non-Null Count  Dtype
4 ---  ---

```

```

5  0  index    5238 non-null  int64
6  1  book_id  5238 non-null  int64
7  2  source   5238 non-null  object
8  3  target   5238 non-null  object
9  4  contact  5238 non-null  int64
10 dtypes: int64(3), object(2)
11 memory usage: 204.7+ KB

```

Die Konsolenausgabe 11.5 zeigt, dass der Datensatz 5238 Zeilen aufweist. Hierbei handelt es sich um fünf Variablen, die alle über 5238 Einträge verfügen und alle die Eigenschaft non-null verfügen, also keine Nullwerte sind. Drei der Variablen sind als Integer gespeichert, die anderen zwei werden als Objekte ausgewiesen. In diesem Fall sind das String-Objekte. Der Datensatz ist insgesamt etwa 204.7 KB groß und damit relativ klein.

Alternativ kann das Vorkommen auf NaN-Werte auch mit dem verschachtelten Funktionsaufruf `df.isnull().sum()` erfolgen. Die Ausgabe dieser Anweisung mithilfe der entsprechenden `print` Funktion zeigt die Konsolenausgabe 11.6.

Listing 11.6 Konsolenausgabe: `df.isnull().sum()`

```

1  Anzahl NaN:  index      0
2  book_id      0
3  source       0
4  target       0
5  contact      0

```

Ausgehend von diesen ersten Analysen zeigt sich, dass in diesem Datensatz keine unerwünschten oder problematischen Einträge vorhanden sind, sodass der nächste Schritt erfolgen kann.

11.3 Soziales Netzwerk: Explore the Data

Auf Basis des Data Pre-processing ergibt es Sinn, ein Gefühl für die Verteilung der Daten zu erhalten. Vor diesem Hintergrund ist die Funktion `df.describe()` ein nützliches Instrument. Das Codebeispiel 11.7 zeigt die Ausführung der Funktion.

Listing 11.7 Nutzung von `df.describe()`

```

29  # _____
30  # Explore the data
31  # _____
32  # Erste uebersicht der Daten.
33  print(df.describe())

```

Die Konsolenausgabe 11.8 zeigt das Ergebnis des Funktionsaufrufs als Übergabe in der `print`-Funktion.

Listing 11.8 Konsolenausgabe: `df.describe()`

	index	book_id	contact
count	5238.000000	5238.000000	5238.000000
mean	2618.500000	1.216877	12.579420
std	1512.224686	0.412157	19.880906
min	0.000000	1.000000	2.000000
25%	1309.250000	1.000000	4.000000
50%	2618.500000	1.000000	7.000000
75%	3927.750000	1.000000	12.000000
max	5237.000000	2.000000	442.000000

Auf Basis der numerischen Werte zeigt die Funktion `df.describe()` in der Konsolenausgabe 11.8, dass es eine Zeile mit 442.0 Kontakten ('contact') gibt. Die kleinste Anzahl an Kontakten beträgt 2,0. Das 75-Prozent-Quantil umfasst zwölf Kontakte, wobei das 50-Prozent-Quantil gerade mal auf sieben kommt und das 25-Prozent-Quantil gerade einmal auf vier Kontakte verweist. Diese Verteilung ist ein Indikator, dass die Daten nicht normal verteilt zu sein scheinen. Die Betrachtung der Werte in den Spalten 'Index' und 'book_ID' aus der Konsolenausgabe 11.8 ergibt nicht viel Sinn, weil der Index eine fortlaufende Nummer ist und die 'book_ID' nur auf das jeweilige Buch in der Reihe verweist, in dem die Kontaktaufnahme erfasst worden ist.

Damit aus den Daten ein Graph mit Knoten und Kanten erstellt werden kann, bietet sich die Funktion `nx.from_pandas_edgelist` an. Diese Funktion stammt aus dem `networkx`-Modul und kann auf Basis eines `DataFrame` einen Graphen herstellen. Hierzu benötigt die Funktion den Anfangsknoten 'source' und einen Zielknoten 'target' sowie die optionale Information über die Stärke einer Kante zwischen den Knoten 'edge_attr' und den Zielgraphen. Der Zielgraph kann durch den Funktionsaufruf `Graph = nx.Graph()` bequem erstellt werden. Die Ausführung zeigt das Codebeispiel 11.9.

Listing 11.9 Nutzung von `nx.from_pandas_edgelist`

```

35 # Erstellen eines Graphen mit Hilfe des Networks Moduls
36 # aus einem DataFrame Objekt.
37 Graph = nx.Graph()
38 Graph = nx.from_pandas_edgelist(df, source='source', target='target', edge_attr='
    contact', create_using=Graph)

```

Das Ergebnis des Codebeispiels 11.9 ist ein Graphobjekt, das für die weitere Analyse der Kommunikationsstruktur in der Game-of-Thrones-Buchreihe verwendet werden kann.

Die Datenvisualisierung ist für die Exploration von Netzwerkdaten besonderes wichtig, um einen Eindruck über die Netzwerkstruktur zu erhalten. Daher wird im Folgenden eine eigene Funktion definiert, um auf Basis der Daten das Netzwerk zu visualisieren. Die Abb. 11.1 zeigt das Ziel des Vorhabens.

Die Abb. 11.1 zeigt, dass es einige wenige große Knoten und zahlreiche kleinere Knoten gibt. An dieser Stelle sei vorweggegriffen, dass die größeren Knoten auf ihren Grad projiziert

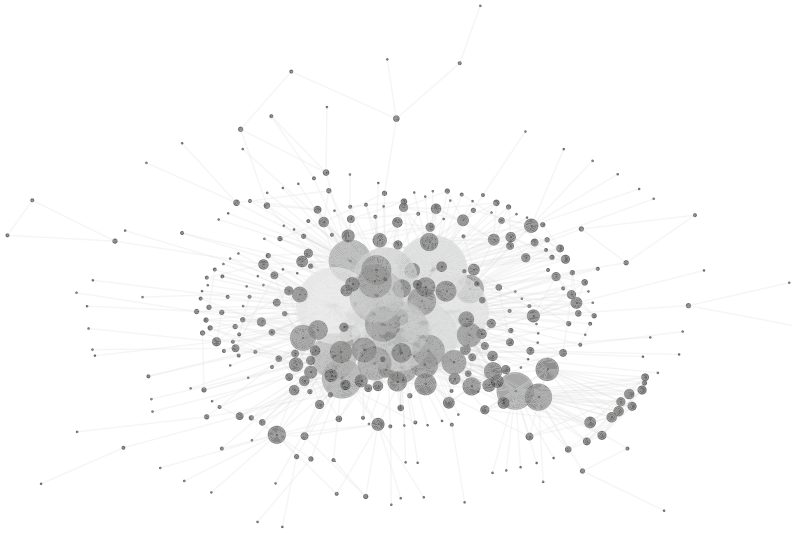


Abb. 11.1 Das soziale Game-of-Thrones-Netzwerk in einer ersten Übersicht

wurde. Der Grad (Degree) beschreibt die Anzahl der ausgehenden und eingehenden Kontakten eines Knotens. Je größer der Knoten, desto mehr Kontakte besitzt er. Dies zeigt, dass im Datensatz einige wenige Knoten maßgeblich für die Kommunikation in der Buchreihe verantwortlich sind.

Zu Erstellung dieses Graphen in der Abb. 11.1 bietet sich die Erstellung einer Funktion an, die im späteren Verlauf oder in anderen Skripten für die Erstellung von weiteren Abbildungen genutzt werden kann. Die eigene Funktion zur Erstellung der Graphen für die Abbildung zeigt das Codebeispiel 11.10.

Listing 11.10 Abbildung eines Graphen durch die eigene Funktion fGeneratePlot

```

40 # Funktion fuer die Erstellung einer Abbildung
41 # @param: Graph Objekt, Labels, Bezeichnung, Format
42 # @return: none, speichert die Abbildung im gewuensten Format.
43 def fGeneratePlot(Graph, bLabes, sFigureName, sFormat):
44     LNodeDegree = []
45     LNodeSize = []
46     LNames = []
47
48     # Liste fuer die Berechnung des Grades.
49     for node in Graph:
50         LNames.append(node)
51         dDegree = Graph.degree(node)
52         LNodeDegree.append(float(dDegree))
53
54     # Potenzierung des Grades fuer eine bessere Knoten groesse.

```

```

55         LNodeSize.append(dDegree ** 2)
56
57     # Positionierung der Knoten
58     DicPos = nx.kamada_kawai_layout(Graph)
59
60     # Abbildung erstellen
61     plt.figure(figsize=(12, 8))
62     nx.draw_networkx_edges(Graph, DicPos, alpha=0.1, edge_size=1, edge_color='grey')
63     nx.draw_networkx_nodes(Graph, DicPos, node_size=LNodeSize, cmap=plt.cm.binary_r,
64                             node_color=LNodeDegree, alpha=0.7)
65
66     if bLabes == True:
67         nx.draw_networkx_labels(Graph, pos=DicPos, font_size=5, alpha=0.5)
68     plt.axis('off')
69
70     sFigureNameAsFormat = sFigureName + sFormat
71     plt.savefig(sFigureNameAsFormat, bbox_inches='tight')
72     plt.show()

```

Die Funktion aus dem Codebeispiel 11.10 erwartet vier Eingabeparameter. Das Graphobjekt, einen Boolean für die Knotenbeschriftung, einen Namen für die Abbildung als String sowie das Format der Abbildung als String. Der Methodenkopf `fGeneratePlot` erwartet somit vier Eingaben vom Benutzer. Innerhalb der Funktion werden drei Listen initialisiert `LNodeDegree`, `LNodeSize` und `LNames`. Diese Listen dienen als Informationsspeicher für Berechnungen, die innerhalb der eigenen Funktion im Codebeispiel 11.10 notwendig sind, um die Abbildung entsprechend zu erstellen.

Das nächste wichtige Element der eigenen Funktion im Codebeispiel 11.10 ist eine `for`-Schleife. In diesem Fall iteriert die Schleife nicht über einen Index, sondern nutzt eine Objektvariable. Das bedeutet, die Indizes der Schleife sind zur Laufzeit einzelne Objekte. Genauer gesagt verwendet die Schleife als Index die einzelnen Knoten des Graphen. Dies ergibt sich aus der Anweisung `for node in Graph`. Damit wird die Schleife veranlasst, über alle Knoten innerhalb des Graphen zu iterieren. Innerhalb der Schleife werden zunächst die Namen der Knoten in die Liste `LNodeDegree` gespeichert. Diese Namen entsprechen den Namen für die Charaktere aus den Büchern. Durch den Funktionsaufruf `Graph.degree()` werden die einzelnen Degrees der Knoten berechnet. Der Degree wird anschließend in der Variable `dDegree` gespeichert und als Float in die Liste `LNodeDegree` aufgenommen. Für die Graphenvisualisierung scheint es sinnvoll, eine weitere Liste mit der Größe der Knoten anzulegen und die Werte entsprechend zu speichern. Diese Aufgabe übernimmt die vorher angelegte Liste `LNodeSize`. Diese Liste erhält den Degree jedes Knotens, der mit dem Faktor zwei potenziert wird. Diese Potenzierung soll bei der Unterscheidung der Größe helfen und so der Flächenvisualisierung gerecht werden.

Nach dem Durchlauf der Schleife erfolgt die Positionierung der einzelnen Knoten im Graphen. Dies übernimmt der Funktionsaufruf `nx.kamada_kawai_layout()`, der die Positionen der einzelnen Knoten im Graphen für die Visualisierung zurückgibt. Im Modul `net-`

workx gibt es zahlreiche sogenannte Layoutalgorithmen¹. Allerdings kann die Berechnung der Positionen in Abhängigkeit vom gewählten Algorithmus, der verfügbaren Rechenleistung und der gegebenen Komplexität des Graphen einige Zeit in Anspruch nehmen. Aber für diesen Graphen ist das Rechenzeitproblem zu vernachlässigen. Die Werte der Knotenpositionierung werden in der Variable `DicPos` gespeichert. Diese Variable ist ein Python Dictionary und beinhaltet die x- und y-Koordinaten für einen entsprechenden Knoten im Graphen.

An dieser Stelle sind alle wichtigen Berechnungen für die Erstellung der Abbildung abgeschlossen, sodass die eigentliche Erstellung der Abbildung erfolgen kann. Hierzu wird eine Abbildungsumgebung aus dem matplotlib-Modul mit `plt.figure()` erstellt, das zusätzlich noch mit dem optionalen Parameter `figsize=(12,8)` die Größe der Abbildung spezifiziert. Die Funktion `nx.draw_networkx_edges` zeichnet die Kanten des Graphen aus dem Graphenobjekt `Graph` und den Positionen `DicPos` der Knoten. Dabei werden zusätzlich die optionalen Parameter genutzt, um die Kanten stark durchsichtig mit `alpha=0.1` zu gestalten. Mit `edge_size` wird die Kantendicke auf eins gesetzt. Außerdem sollen die Kanten grau erscheinen, was mit dem Aufruf `'grey'` realisiert wird. Die Knoten des Graphen werden entsprechend mit dem Aufruf `nx.draw_networkx_nodes` veranlasst. Hierbei werden ebenfalls der Graph `Graph` und die Positionen `DicPos` der Knoten sowie die Größe der einzelnen Knoten über die Liste `LNodeSize` und den optionalen Parameter `node_size` übergeben. Die Farben für die Knoten werden mithilfe einer `cmap colormap`, oder auch Farbtafel genannt, erstellt. Dieser optionale Parameter `cmap` erwartet eine definierte Farbpalette mit passenden Farben. Diese kann über das matplotlib-Modul mit dem Aufruf `plt.cm.` aufgerufen werden. In diesem Fall wird die Farbpalette `binary` verwendet. Durch das angehängte `_r` (reverse) wird die Farbpalette invertiert. Zusätzlich werden die Knotenfarbe mit `node_color` und die Liste `LNodeDegree` auf die Größe der Knoten projiziert. Die Knoten sollen dabei einen transparenten Schimmer erhalten, was mit `alpha=0.7` gelingt.

Die Funktion sieht zusätzlich vor, dass auf Wunsch die Knotennamen zur Abbildung hinzugefügt werden. Dazu wird geprüft, ob der entsprechende Eingabeparameter `bLabels` der eigenen Funktion `fGeneratePlot` auf `True` steht. Sollte dem so sein, werden mithilfe von `nx.draw_networkx_labels` die Knotennamen des Graphen `Graph` an den Positionen `pos=DicPos` in einer Schriftgröße von `font_size=5` mit einer Durchsichtigkeit von 50 % `alpha=0.5` in die Abbildung aufgenommen.

Die Beschriftung für das zu speichernde File der Abbildung wird aus dem Inputparameter der eigenen Funktion in die Variable `sFigureNameAsFormat` und dem aus Inputparameter für das File Format über eine String Konkatenation zusammengesetzt. Die Variable wird dann der `plt.savefig()`-Funktion zur Speicherung der Abbildung mit einem kleineren weißen Rahmen durch `bbox_inches=` und `'tight'` übergeben, um die Abbildung zu speichern. Am Schluss der eigenen Funktionen `fGeneratePlot` wird die Abbildung durch

¹ Eine Übersicht möglicher Layoutalgorithmen zur Erstellung von Graphen im Modul `networkx` findet sich in der Dokumentation des Moduls unter: <https://networkx.github.io/documentation/stable/reference/drawing.html>, zuletzt abgerufen am 1. Dezember 2020.

`plt.show()` in der Entwicklungsumgebung angezeigt. Der Aufruf der eigenen Funktion im Skript gelingt durch `fGeneratePlot()` mit den entsprechenden Eingabeparametern `Graph, False, 'GameOfThrones', '.pdf')`.

Nach der Erstellung der ersten Übersichtsabbildung 11.1 mithilfe der Funktion `fGeneratePlot` und dem dadurch gewonnenen Eindruck für die Daten scheint eine erste detaillierte Analyse der Daten sinnvoll. Zu diesem Zweck bietet es sich an, auf Basis der Sozialen-Netzwerk-Analyse einige Kennzahlen zu berechnen und in einem neuen `DataFrame` für die Weiterverarbeitung zusammenzufassen. Hierzu werden im aktuellen Skript `b_bGameOfNetwork.py` ein weiteres `DataFrame` angelegt, entsprechende Metriken berechnet und in einem `DataFrame` zusammengefasst. Anschließend wird das neue `DataFrame` als `LATEX`-Tabelle zu Darstellung in diesem Buch exportiert. Das Codebeispiel 11.11 zeigt die Vorgehensweise.

Listing 11.11 Durchführung einer ersten Sozialen-Netzwerk-Analyse

```

78 # ----- SNA Metriken -----
79 # Erstelle DataFrame zum Speichern der Informationen
80 df_Nodes = pd.DataFrame(columns=['node', 'degree', 'betweenness', 'centrality'])
81
82 DicBetweenness = nx.betweenness_centrality(Graph)
83 DicDegreeCentrality = nx.degree_centrality(Graph)
84
85 for node in Graph:
86     # Zugriff auf die Werte
87     dDegree = Graph.degree(node)
88     dBetweenness = DicBetweenness.get(node)
89     dCentrality = DicDegreeCentrality.get(node)
90
91     df_Nodes = df_Nodes.append({'node': node,
92                                'degree': float(dDegree),
93                                'betweenness': dBetweenness,
94                                'centrality': dCentrality
95                                }, ignore_index=True)
96
97 df_Nodes = df_Nodes.sort_values(by='degree', ascending=False)
98
99 df_NodesTopTen = df_Nodes[0:10]
100 df_NodesTopTen.to_latex('tab-df_NodesTopTen.tex', float_format="%0.2f", index=False,
    index_names=False)

```

Das Codebeispiel 11.11 zeigt, das zunächst die Betweenness Centrality mit der entsprechenden Funktion berechnet wird. Die Ergebnisse werden als Dictionary der Variable `DicBetweenness` gespeichert. Die Betweenness Centrality kann definiert werden als die Anzahl der kürzesten Wege (p for path) zwischen zwei Punkten (n nach m), die über den jeweils betrachteten Knoten (beliebiger Knoten i) gehen. Formaler ergibt sich damit die Gl. 11.1.

$$p_{n,m}(i) \quad (11.1)$$

Die Betweenness Centrality ist damit ein Indikator für die Anzahl der kürzesten Kommunikationswege, die über einen Knoten i gehen. Je höher die Anzahl an kürzesten Kommunikationswege, die über den Knoten gehen, desto höher ist die soziale Stellung. Knoten mit einer hohen Betweenness Centrality fungieren als Intermediäre und können die erhaltenen Informationen für sich nutzen.

Zusätzlich wird mithilfe der vordefinierten Funktion aus dem `networkx`-Modul die Degree Centrality für jeden einzelnen Knoten berechnet und in die Variable `DicDegreeCentrality` gespeichert. Die Degree Centrality kann definiert werden als der Grad des Knoten i ($K(i)$) geteilt durch die Anzahl aller Knoten $- 1$ ($N - 1$), analog zur Formalisierung 11.2.

$$\frac{K_i}{N - 1} \quad (11.2)$$

Damit beschreibt die Degree Centrality den relativen Anteil an direkten Kontakten aus dem gesamten Netzwerk.

Zur Bestimmung der Werte für jeden einzelnen Knoten wird wieder erneut das Konzept der For-Schleife auf Objektebene genutzt. Dabei werden die Knoten genutzt, um in der For-Schleife den Degree zu bestimmen `Graph.degree()`. Ebenso wird der dazugehörige Wert für die Betweenness Centrality und für die Degree Centrality berechnet, indem auf den jeweiligen Dictionaries durch den Aufruf `.get(node)` die spezifischen Werte für die einzelnen Knoten zurückgegeben werden. Die einzelnen Metriken werden innerhalb der Schleife knotenweise bestimmt und in das neue DataFrame geschrieben. Durch die Zuweisung von `df_Nodes` mit `df_Nodes.append` werden die neuen Werte bestimmt und an das neue DataFrame angehängt. Dieses DataFrame `df_Nodes` überschreibt das vorherige DataFrame mit der neuen Wertebelegung.

Nach Abschluss der For-Schleife wird das DataFrame `df_Nodes` durch den Funktionsaufruf `.sort_values()` entsprechend nach dem Degree durch `by='degree'` in absteigender Reihenfolge mit `ascending=False` sortiert. Dieses sortierte DataFrame wird dann der bisherigen Data-Frame-Variable zugewiesen, sodass das sortierte DataFrame das unsortierte DataFrame überschreibt. Anschließend wird mittels Indexierung auf dem DataFrame `df_Nodes[0:10]` eine neue Variable `df_NodesTopTen` erstellt, die entsprechend der vorherigen Sortierung die zehn größten Knoten (Charakter im Buch) gemessen an der Anzahl ihrer Kontakte (Degree) beinhaltet. Das `df_NodesTopTen` wird mithilfe des Funktionsaufrufs `to_latex()` als $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}^2$ -Tabelle exportiert, um diese Tabelle in diesem Buch darstellen zu können. Das Ergebnis des Exports zeigt die Tab. 11.1.

Die Tab. 11.1 zeigt die Ergebnisse der ersten Analysen. Die Ergebnisse zeigen, dass Tyrion Lannister (in der gleichnamigen Serie gespielt von Peter Dinklage) eine zentrale Rolle einnimmt, auf Basis seiner Verbindungen zu anderen Personen. Zudem hat er neben

² $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ ist ein Textverarbeitungsprogramm was in den Naturwissenschaften und technischen Disziplinen zum Einsatz kommt. Dieses Buch ist beispielsweise in $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ verfasst.

Tab. 11.1 Top-Ten-Charaktere in „Game of Thrones“ basierend auf ihrem Degree, Betweenness Centrality und der Degree Centrality. Betweenness steht für Betweenness Centrality und Centrality steht für Degree Centrality

Node	Degree	Betweenness	Centrality
Tyrion Lannister	71	0.17	0.20
Eddard Stark	68	0.17	0.20
Catelyn Stark	58	0.13	0.17
Arya Stark	56	0.10	0.16
Jon Snow	53	0.17	0.15
Joffrey Baratheon	53	0.06	0.15
Robert Baratheon	53	0.13	0.15
Cersei Lannister	51	0.06	0.15
Bran Stark	49	0.13	0.14
Sansa Stark	43	0.04	0.12

Catelyn Stark (in der Serie gespielt von Michelle Fairley), die höchste Betweenness (0.17) und Degree Centrality (0.20). Insgesamt zeigt die Liste, dass die Familie Stark mit fünf Familienmitgliedern sehr einflussreich zu sein scheint.

Ausgehend vom Degree fällt auf, dass die Verteilung des Degree nicht normal verteilt zu sein scheint. In realen sozialen Netzwerken ist es häufig so, dass es sehr viele Knoten mit sehr wenigen Kontakten gibt und einige wenige Personen, die über sehr viele Kontakte verfügen. Diese Eigenschaft von sozialen Netzwerken wird als skalenfreie Struktur bezeichnet. Für diesen Datensatz bietet es sich an, diesen Sachverhalt durch eine einfache Analyse zu überprüfen. Hierzu wird die Verteilung des Degree mittels Verteilungsdiagramm visualisiert. Allerdings wird in der Forschung häufig, zur besseren Darstellung, der Logarithmus für die Achsenskalierung verwendet. Damit repräsentieren die Achsenwerte beispielsweise $10^0 = 1$, für $10^{-1} = 0.1$ und $10^{-2} = 0.01$. Die Umsetzung dieses Vorhabens zeigt das Codebeispiel 11.12.

Listing 11.12 Verteilungswahrscheinlichkeit der Degrees auf Basis des Logarithmus

```

103 # ----- Degree Verteilung -----
104 # Berechne den Anteil jedes Degree auf Basis der Gesamtverteilung.
105 # Lege hierzu eine neue Spalte im DataFrame an um den Anteil entsprechend
106 # zu speichern. Ueber Unique werden alle möglichen Degrees erfasst.
107 # Der relativer Anteil wird durch die Division bestimmt.
108 # Anschliessend werden die Datenpunkte fuer die Darstellung sortiert.
109 df_Nodes[ 'perc' ] = df_Nodes[ 'degree' ]/len(df_Nodes[ 'degree' ])
110 LDegreePerc = sorted(df_Nodes[ 'perc' ], reverse=True)
111
112 # Darstellung erstellen
113 fig, ax = plt.subplots()
114 ax.spines[ 'top' ].set_visible(False)

```

```
115 ax.spines['right'].set_visible(False)
116 plt.loglog(LDegreePerc, "ko")
117 ax.set_xlabel(xlabel='Degree')
118 ax.set_ylabel(ylabel='Probability')
119 plt.savefig('LogLogDegree.pdf', bbox_inches='tight')
120 plt.show()
```

Im Codebeispiel werden zunächst die relativen Anteile der Datenpunkte in der Spalte 'degree' bestimmt. Hierbei wird der betreffende Wert `df_Nodes['degree']` durch die Anzahl der vorkommenden Werte dividiert `len()`. Das Ergebnis in Form des relativen Anteil jedes Degree wird dann in eine neue Spalte 'perc' mit `df_Nodes['perc']` des DataFrame angelegt. Damit wird also für jeden Degree bestimmt, welchen relativen Anteil dieser Datenpunkt (und die anderen mit dieser Größe) an der Gesamtverteilung besitzen. Der Vorteil des DataFrame besteht nun darin, dass alle Werte passend zu jedem ursprünglichen Datenpunkt in eine zusätzliche Informationsspalte geschrieben werden, indem die Ergebnisvariable `df_Nodes['perc']` die Operation zugewiesen bekommt. Das DataFrame erkennt jetzt, dass keine neue Variable angelegt werden soll, sondern nur eine neue Spalte in der bestehenden Variable `df_Nodes`. Anschließend werden die Prozentwerte durch `sorted()` sortiert und in eine neue Variable `LDegreePerc` geschrieben, um die Werte für die spätere Abbildung zu speichern. Die Erstellung der Abbildung erfolgt analog zu den bisherigen Abbildung, bis auf die Tatsache, dass als Plot der `plot.loglog` aus dem Modul `matplotlib` verwendet wird. Dieser Plot plottet eine normale Diagrammabbildung mit logarithmuskalierten Achsen. Neben den üblichen Formatierungsanweisungen bekommt die Abbildungsumgebung den Dateinamen und Achsenbeschriftungen übergeben. Der Plot in der Abbildungsumgebung bekommt die sortierten Daten `LDegreePerc` übergeben. Das Ergebnis der Skriptanweisungen zeigt die Abb. 11.2.

Die Abb. 11.2 zeigt, dass die Wahrscheinlichkeit von Knoten mit einer kleinen Degree-Anzahl (etwa ein bis zehn Kontakte) mit (etwa um die 20%) relativ hoch ist. Knoten mit einer großen Anzahl an Kontakten sind aber relativ gering, sodass in der Abbildung der klassische lineare Zusammenhang von realen Netzwerken (Facebook, Flickr, Twitter, E-Mail Telefonnetzwerken) zu sehen ist. Diese Struktur unterliegt dem Power law (auch genannt Long tail, Heavy tail, Zipf's law, Pareto's law). Damit lassen sich die Strukturen von sozialen Netzwerken nicht über einen zufälligen Prozess erklären, sondern folgen einer Gesetzmäßigkeit.

Ausgehend von diesen ersten Arbeiten im Hinblick auf die Datenexploration fasst die Tab. 11.2 einige nützliche Funktionen übersichtlich zusammen.

Ausgehend von diesen Operationen bietet es sich an, ein Modell zu konzipieren, das auf den gegebenen Daten in der Lage ist festzustellen, welche Teilnehmer besonders relevant sind.

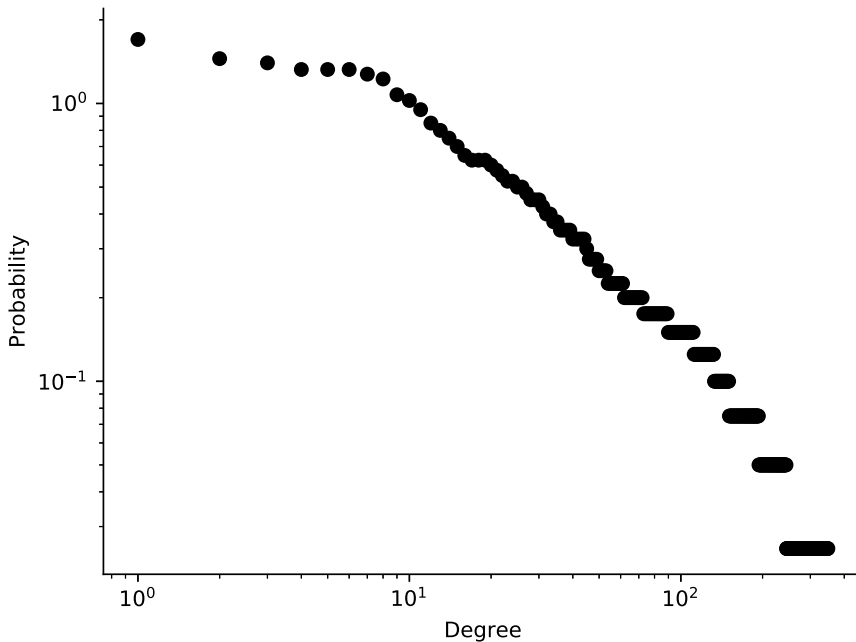


Abb. 11.2 Game-of-Thrones-Netzwerkstruktur auf Basis des Degree

11.4 Soziales Netzwerk: Model the Data (Logistische Regression)

Ausgehend von der ersten Datenexploration drängt sich die Frage auf, ob eine Klassifikation der relevanten Charaktere in der Game-of-Thrones-Buchreihe mithilfe einer logistischen Regression möglich ist. Hierzu werden im Abschn. 11.4.1 die theoretischen Grundlagen zu diesem Modell geliefert sowie in Abschn. 11.4.2 eine mögliche Implementierung gezeigt.

11.4.1 Theoretische Grundlage des Modells

Zur Modellierung der Daten wird zu Illustrationszwecke eine logistische Regression verwendet. Die Begründung für die Verwendung dieses Logit-Modells statt einer klassischen linearen Regression liegt in der geplanten abhängigen Variable begründet. Dabei soll eine sogenannte Dummy-Variable verwendet werden, die eine Aussage darüber gibt, ob ein Charakter in dem Datensatz eine wichtige Rolle einnimmt oder nicht. Diese Variable ist somit binär codiert, also mit 0 oder 1. Dabei ist 0 die Dummy-Codierung für einen Charakter, der nicht gut ist, bzw. 1 ist ein Charakter mit guten Absichten. Zur Umsetzung dieser Regressionsart wird die nachstehende Formel 11.3 verwendet.

Tab. 11.2 Übersicht nützlicher Funktionen für die Datenexploration

Funktion	Auswertung zu:
<code>nx.from_pandas_edgelist()</code>	Erstellt aus einem DataFrame auf Basis der Spaltennamen für Start und Zielknoten ein networkx-Graphenobjekt
<code>nx.draw_networkx_edges()</code>	Zeichnet die Kanten in eine Abbildungsumgebung aus dem matplotlib-Modul. Benötigt das networkx-Modul
<code>nx.draw_networkx_nodes()</code>	Zeichnet die Knoten in eine Abbildungsumgebung aus dem matplotlib-Modul. Benötigt das networkx-Modul
<code>nx.draw_networkx_labels()</code>	Zeichnet die Knotennamen in eine Abbildungsumgebung aus dem matplotlib-Modul. Benötigt das networkx-Modul
<code>df_AUSWAHL = df[START:ENDE]</code>	Ein Teil-Data-Frame aus dem ursprünglichen DataFrame erstellen
<code>df['RELATIVE_ANTEILE'] = df['SPALTE']\len(df['SPALTE'])</code>	Berechnet die relativen Anteile für alle Einträge in einer Spalte und speichert diese in einer neuen Spalte im ursprünglichen DataFrame

$$Logit(y_{1|0}) = \alpha + \beta_1 + x_1 + \cdots + \beta_i + x_i + \epsilon \tag{11.3}$$

- y Abhängige Variable
- α Regressionskonstante
- β_1 Regressionskoeffizient bzw. Steigungskoeffizient der ersten unabhängigen Variable
- x_1 Ausprägung des Merkmals Nr. 1
- β_i Regressionskoeffizient bzw. Steigungskoeffizient der i -ten unabhängigen Variable
- x_i Ausprägung des i -ten Merkmals
- ϵ Nicht durch das Modell erklärbare Ausprägungen

Für diese multivariate logistische Regressionsanalyse werden die unabhängigen Variablen aus dem Datensatz `dataset_GoT.csv` verwendet. Im Rahmen der multivariaten logistischen Regression wird die Odds Ratio betrachtet. Über das Einsetzen der jeweiligen β -Werte in die Exponentialfunktion ($exp^{(\beta)}$) wird diese Kennzahl bestimmt. Dabei gilt für dieses Buch, dass die Odds Ratio eine prozentuale Eintrittswahrscheinlichkeit ist. Ist der Wert größer als 1, steigt die Wahrscheinlichkeit, dass es sich um einen guten Charakter handelt. Analog dazu gilt: Ist die Odds Ratio kleiner als 1, nimmt die Wahrscheinlichkeit entsprechend ab. Die Umsetzung in Python zeigt der folgende Abschn. [11.4.2](#).

11.4.2 Implementierung des Modells

Die Implementierung des Modells gelingt ebenfalls mit dem umfangreichen statsmodels-Modul in Python. Das Codebeispiel 11.13 zeigt die Umsetzung.

Listing 11.13 Logistische Regressionsanalyse

```

123 # -----
124 # Model the data
125 # -----
126 df_Nodes.loc[df_Nodes['node'].str.contains('Stark'), 'good'] = 1
127 df_Nodes.loc[df_Nodes['good'] != 1, 'good'] = 0
128
129 # Erstellung des Modell
130 sFormula = 'good ~ degree + betweenness + centrality'
131 LogitModel = smf.logit(formula=sFormula, data=df_Nodes).fit()
132 LogitModel.summary()
133 print(LogitModel.summary())
134
135 df_model_odds = pd.DataFrame(np.exp(LogitModel.params), columns= ['Odds-Ratio' ])
136 df_model_odds['z-value'] = LogitModel.pvalues
137 print(df_model_odds)

```

Zunächst wird im aktuellen DataFrame mithilfe der `.loc`-Funktion ein zeilenweiser Vergleich durchgeführt, um herauszufinden, ob ein Knotennamen den Familiennamen 'Stark' beinhaltet. Sofern die Beendigung zutrifft, wird in die neue Spalte 'good' eine 1 eingesetzt. Die Funktion `.loc` des DataFrame ermöglicht den Zugriff auf Gruppen von Zeilen und Spalten über Label(s), sodass eine Bedingungsabfrage, wie im Codebeispiel 11.13 zur Einteilung der Variablen mithilfe einer Dummy-Variable gezeigt, möglich wird.

In allen anderen Fällen wird eine 0 in die Zeilen geschrieben. Die Betrachtung der Gegenfälle übernimmt die untenstehende Codezeile. Sie prüft, ob in der Zeile der Spalte eine 1 steht. Wenn das nicht der Fall ist, wird eine 0 eingetragen. Damit werden die durch die Initialisierung der Spalte entstandenen NaN-Werte mit einer 0 überschrieben.

Die Modellerstellung erfolgt analog zur Nutzung der linearen Regression aus dem vorherigen Abschnitt. An dieser Stelle wird ebenfalls eine Regressionsgleichung definiert, indem ein entsprechender String aus den Variablen der Regression gebildet wird. Diese Gleichung wird in die Variable `sFormula` gespeichert. Das formalisierte Modell wird entsprechend der logistischen Regression übergeben, die durch den Aufruf des Moduls `smf`. und das gewünschte Modell `logit` der Variable `LogitModel` zugewiesen wird. Hierbei werden das Modellgleichungssystem `sFormula` sowie die Daten `df_Nodes` mitübergeben. Ebenfalls erfolgt die Ausgabe der Modellstatistiken.

Allerdings ist bei logistischen Regressionen zu beachten, dass erst durch den Einsatz der Odds Ratio die Werte sinnvoll interpretierbar sind. Vor diesem Hintergrund wird zunächst ein neues DataFrame für das Modellergebnis angelegt. Dieses DataFrame `df_model_odds`

bekommt die Modellparameter als exponentielle Transformation, die für die Bestimmung der Odds Ratio notwendig sind, durch `np.exp()` übergeben. `np.exp` ist im Modul `numpy` verfügbar. Daher wird das Modul zu Beginn des Skripts durch `numpy as np` verfügbar gemacht. `numpy` ist ein Modul für Python, das speziell für Skalar-, Vektor- und Matrizenoperationen konzipiert wurde. Daher bietet es zahlreiche vorimplementierte Funktionen. Die Ergebnisse in Form der Odds Ratio und der sogenannten z-Values (vergleichbar mit den p-Values) werden dem Data Fame `df_model_odds` übergeben. Die kurze Betrachtung der Ergebnisse erfolgt im Anschluss in Abschn. 11.5.

11.5 Soziales Netzwerk: Interpretation der Ergebnisse

Im vorherigen Abschn. 11.4.2 wurde die logistische Regression implementiert und auf den generierten Daten geschätzt. Die Ergebnisse aus dem Codebeispiel 11.13 zeigt die Konsolenausgabe 11.14.

Listing 11.14 Konsolenausgabe: Logistische Regression

	Odds-Ratio	z-value
1 Intercept	0.008696	7.523881e-18
2 Degree	1.108670	9.999998e-01
3 Betweenness	0.000092	5.205556e-01
4 Centrality	1.000296	1.000000e+00

Auf Basis der Odds-Ratio-Ergebnisse und den z-Values ergibt sich, dass mit dem Anstieg des Degree um eine Einheit die Wahrscheinlichkeit um 11,0 % (1,108670), ein guter Charakter im Film zu sein, anzusteigen scheint. Die Betweenness und die Degree Centrality haben scheinbar mit 0,0 % (0,000092) und 1,0 % (1,000296) keinen Einfluss. Allerdings führt die Betrachtung der z-Values zu dem Schluss, dass die Ergebnisse nicht signifikant sind. Analog zur Regression müssen für stichhaltige Aussagen die Signifikanzniveaus genau geprüft werden. Dieser Umstand kann mehrere Ursachen haben. Beispielsweise können nicht ausreichend Datenpunkte vorhanden sein, um einen entsprechenden Nachweis zu liefern, oder die Annahme, dass der Degree einen Einfluss hat, ist nicht zielführend. Ausgehend von den ersten Modellerstellungsansätzen stellt die Tab. 11.3 zentrale Funktionen zusammen.

An dieser Stelle müssten also weitere Analysen durchgeführt oder der Datensatz entsprechend um weitere Variablen ergänzt werden. Eine genauere Diskussion an dieser Stelle würde den Rahmen einer ersten Einführung sprengen. Daher soll diese erste Einführung in die Soziale-Netzwerk-Analyse ausreichen.

Tab. 11.3 Übersicht nützlicher Funktionen für die Modellkonzeption

Funktion	Auswertung zu:
<code>statsmodels.formula.api as smf</code>	Import des statsmodels-Moduls in das Python-Skript
<code>sFormula = 'Gleichung'</code>	Erstellung der Regressionsgleichung mittels String, um das Modell zu spezifizieren
<code>LogitModel = smf.logit(formula y=sFormula, data=df).fit()</code>	Aufruf einer logistischen Regression, wobei <code>sFormula</code> das übergebende Gleichungssystem ist und durch <code>.fit()</code> das Modell auf Basis der Daten geschätzt wird
<code>np.exp(LogitModel.params)</code>	Berechnung der Odds Ratio auf Basis der Modellwerte
<code>LogitModel.pvalues</code>	Zugriff auf die p-Values, die im Rahmen der Odds Ratio auch z-Values genannt werden



Erhebung und Auswertung von Social-Media-Daten

12

Hendrik Jöntgen

Social-Media-Plattformen, wie beispielsweise Facebook, Instagram oder Twitter, sind mittlerweile ein essenzieller Bestandteil unserer Gesellschaft. So gaben 89 % aller Deutschen zwischen 16 und 24 Jahren an, dass sie soziale Netzwerke nutzen und durchschnittlich 79 min täglich auf diesen Plattformen verbringen.¹

Nutzende von Social-Media-Plattformen haben die Möglichkeit, ein öffentliches oder privates Profil zu erstellen, Verbindungen mit anderen Nutzenden herzustellen und können sich die Verbindungen von anderen Nutzenden ansehen (Boyd & Ellison, 2007). Ebenfalls sind die Nutzenden dazu in der Lage, Meinung mit ihren (Online-)Freunden oder mit der gesamten Plattform zu teilen.

Auf diesem Weg entstehen tagtäglich riesige Datenmengen aus Texten, Bildern und Videos. Aus diesem Grund ist es nicht überraschend, dass diese Plattformen oftmals für die Erhebung von Daten verwendet werden, um beispielsweise das Meinungsbild über ein Unternehmen oder ein Produkt zu erfassen. Dies soll in diesem Kapitel exemplarisch geschehen. Der Aufbau des vorliegenden Kapitels folgt dabei dem vorgestellten Data-Science-Prozess (siehe Abschn. 8.2).

In Abschn. 12.1 wird die Fragestellung aufgezeigt und erläutert, wie die hierfür notwendigen Daten erhoben werden können. Anschließend werden die erhobenen Daten in Abschn. 12.2 aufbereitet, gefiltert und schließlich explorativ untersucht (Abschn. 12.3). Die gewonnenen Ergebnisse werden in 12.5 interpretiert.

¹ Quelle: <https://de.statista.com/themen/1842/soziale-netzwerke/>, zuletzt abgerufen am 1. Dezember 2020.

H. Jöntgen (✉)
Darmstadt, Deutschland
E-mail: datascience@hendrikjoentgen.de

12.1 Social Media: Datensatz und Fragestellung

Am 11.06.2020 enthüllte die Sony Corporation die ersten Informationen über ihre neuste geplante Spielekonsole: die PlayStation 5.² Im Rahmen eines 75-min Livestreams kündigte Sony über 25 Spiele für die Konsole an und enthüllte auch das Aussehen, das Zubehör und die verschiedenen Varianten der Konsole.

Dieser Livestream wurde von Millionen von potenziellen Käufern der Konsole gleichzeitig verfolgt und auf Social-Media-Plattformen öffentlich diskutiert. Da davon auszugehen ist, dass der Livestream vom Großteil der Zuschauer live, also ohne zwischenzeitliche Pausierung, geschaut wurde, ist es möglich, die Beiträge auf den Social-Media-Plattformen den einzelnen Enthüllungen während des Livestreams zuzuordnen.

Das Ziel unserer Analyse ist die detaillierte Untersuchung dieser Beiträge, sodass wir herausfinden können, welche Ankündigungen positiv und welche Ankündigungen negativ wahrgenommen wurden. Um dies zu untersuchen, sollen Beiträge auf der Plattform Twitter untersucht werden.

Die Plattform Twitter wurde 2006 gegründet und verbindet einen Großteil der gängigen Social-Media-Features, die zuvor beschrieben wurden, mit der Möglichkeit, kurze Blog-Einträge, sogenannte Tweets, zu verschicken. Diese Tweets können anschließend von der Öffentlichkeit gelesen werden, aber werden primär den eigenen Followern angezeigt. Andere Nutzende können die Tweets favorisieren, auf sie antworten und sie retweeten, was der Weiterleitung des Tweets an (eine Selektion) eigene(r) Follower entspricht.

Zwei Besonderheiten der Plattform sind i) die Möglichkeit der Verwendung von sogenannten Hashtags und ii) Mentions. Mittels Hashtag kann man seinen Tweet einem oder mehreren Themen zuordnen und andere Nutzende können anschließend gezielt nach diesen Themen suchen. Wenn ein Hashtag innerhalb von kurzer Zeit besonders häufig von besonders vielen Nutzenden verwendet wird, wird es zu einem „Trending Topic“ auf Twitter deklariert und sämtlichen Nutzenden prominent angezeigt. Mit Mentions können Nutzende der Plattform ihre Tweets an bestimmte Accounts richten, sodass die Tweets dieses Accounts prominent angezeigt werden, oder zeigen, dass sie in ihren Tweets über andere Accounts schreiben.

Wie die meisten anderen Social-Media-Plattformen, bietet auch Twitter eine API für den Zugriff auf die Daten der Plattform an.³ Die notwendigen Schritte, um Zugang zu dieser API zu erhalten, werden in dem oben angegebenen Link detailliert beschrieben. Aus diesem Grund wird darauf an dieser Stelle verzichtet. Kurz zusammengefasst sind die folgenden Schritte notwendig:

² <https://www.youtube.com/watch?v=8apW-UFhGZg>, zuletzt abgerufen am 1. Dezember 2020.

³ Siehe <https://developer.twitter.com/en/docs/basics/getting-started>, zuletzt abgerufen am 1. Dezember 2020.

1. Erstellen eines Twitter-Developer-Accounts
2. Erstellen einer Twitter-Developer-App
3. Erstellung von API-Schlüssel und Access Tokens

Anschließend kann die API zum einen dazu genutzt werden, um einen eigenen Twitter-Account automatisch zu betreiben und Tweets der Plattform abzufragen. Für Python existiert hierfür ein spezielles Interface, mit dem die Twitter-API auf einfache Art und Weise genutzt werden kann.⁴ Um die API zu nutzen, muss sie nun zunächst in Python wie folgt initialisiert werden:

Listing 12.1 Initialisierung der Twitter-API

```
1 import twitter
2
3 # Zugangsdaten fuer die Twitter-API muessen hier gesetzt werden
4 # Anschliessend kann die Twitter-API genutzt werden
5 api = twitter.Api(consumer_key="",
6                   consumer_secret="",
7                   access_token_key="",
8                   access_token_secret="")
```

In diesem Code-Snippet müssen die jeweiligen Schlüssel, die zuvor erstellt wurden, eingesetzt werden. Ebenfalls ist hier anzumerken, dass nicht das Modul `twitter`, sondern das Modul `python-twitter` installiert werden sollte.

Nachdem die API nun initialisiert wurde, können wir sie nutzen, um auf die Suchfunktion von Twitter zuzugreifen. Diese erlaubt die Suche nach bestimmten Wörtern, Sätzen sowie Hashtags und/oder Tweets, die von bestimmten Twitter-Accounts stammen, diese erwähnen oder an sie gerichtet sind.⁵

Für unseren Anwendungsfall betrachten wir alle Tweets, die entweder an die Twitter-Accounts `playstation` oder `@PS5Console` gerichtet waren oder die Hashtags `#playstation5` oder `#ps5` verwendet haben. Außerdem interessieren uns nur die Tweets vom Tag des Reveal-Livestreams, also vom 11.06.2020.

Die hierfür notwendige Query lautet demnach:

`„#ps5 OR #playstation5 OR playstation OR ps5console until:2020-08-12 since:2020-08-11,,`

⁴ Siehe: <https://github.com/bear/python-twitter>, zuletzt abgerufen am 1. Dezember 2020.

⁵ Siehe: <https://twitter.com/search-advanced> zuletzt abgerufen am 1. Dezember 2020.

Um diesen Query in unserem Python-Code abfragen zu können, müssen wir sie zunächst noch encodieren, da die Twitter-API nicht mit Sonderzeichen, wie Leerzeichen, Doppelpunkte oder # und umgehen kann. Diese werden jeweils durch Uniform Resource Identifier ersetzt.

Listing 12.2 Abfrage der Twitter-Query

```
11 # Twitter-Query fuer alle Tweets zum PS5-Reveal Event
12 results = api.GetSearch(
13     raw_query="q=%23ps5%20OR%20%23playstation5%20OR%20%40playstation%20OR
    %20%40ps5console until%3A2020-06-12 since%3A2020-06-11")
```

Basierend auf den Ergebnissen des Queries können wir unser benötigtes DataFrame erstellen.

Listing 12.3 Umwandlung der Query-Ergebnisse in ein DataFrame

```
15 import pandas as pd
16 # Ergebnisse der Twitter-Query in ein DataFrame umwandeln
17 # Jeder Tweet wird in ein Dictionary umgewandelt und
18 # die Liste der Dictionaries wird schliesslich zum DataFrame
19 tweet_data = [tweet.AsDict() for tweet in results]
20 df = pd.DataFrame(tweet_data)
```

An dieser Stelle ist jedoch zu bemerken, dass die Twitter-API in ihrer kostenlosen Form lediglich das Erfassen von Tweets, die innerhalb der letzten 30 Tage entstanden sind, ermöglicht. Aus diesem Grund ist es leider dem geneigten Leser nicht möglich, die hier aufgezeigte Query selbst abzufragen. Daher wird dazu eingeladen, die bereitgestellten Twitter-Daten auf der Webseite zum Buch zu verwenden.

Nachdem nun die Daten erhoben wurden, sollten sie anschließend für die weitere Verwendung in eine Datenbank geschrieben werden. Für die Verarbeitung von Social-Media-Daten bietet sich hierfür oftmals die Verwendung einer sogenannten NoSQL-Datenbank an, da hier die Struktur der zu speichernden Dokumente nicht im Vorfeld vorgegeben werden muss und sie auch ohne Probleme im weiteren Bearbeitungsverlauf verändert werden kann, ohne dass die Datenbank hierfür angepasst werden muss.

Es existieren viele verschiedene NoSQL-Datenbanken, die an dieser Stelle verwendet werden können, beispielsweise die MongoDB (von humongous, also riesig), die mit einem dazugehörigen Python-Modul pymongo⁶ einhergeht.

⁶ Siehe: <https://pymongo.readthedocs.io/en/stable/>, zuletzt abgerufen am 1. Dezember 2020.

Listing 12.4 Speicherung der Twitter-Daten in eine MongoDB

```

22 from pymongo import MongoClient
23 # Einstellungen der MongoDB
24 client = MongoClient("ClientAddress", 27017)
25 db = client["ClientName"]
26 # Einfuegen der Tweets in die Datenbank "rawData"
27 db.rawData.insert_many(df.to_dict("records"))

```

Nachdem die Daten erhoben und zwischengespeichert wurden, können wir nun mit dem Pre-processing beginnen.

12.2 Social Media: Pre-processing

Für das Pre-processing der Daten müssen sie zunächst aus der Datenbank geladen werden. Falls hierfür MongoDB verwendet wurde, ist dies wie folgt möglich.

Listing 12.5 Laden der Twitter-Daten von einer MongoDB

```

29 # Einstellungen der MongoDB
30 client = MongoClient("ClientAddress", 27017)
31 db = client["ClientName"]
32 collection = db["rawData"]
33 # Extrahierung der Tweets aus der Datenbank
34 tweets = collection.find({})
35 # Umwandlung in ein DataFrame
36 df = pd.DataFrame(list(tweets))

```

Im nächsten Schritt werden die Zeitdaten der Tweets aufbereitet. Dies ist notwendig, da in unserer Datenbank die Information über das Veröffentlichungsdatum der Tweets lediglich als Strings vorliegt. Mithilfe des Moduls `datetime` können wir diesen String jedoch in ein `Datetime`-Objekt umwandeln. Hierfür verwenden wir die `.strptime()`-Funktion des Moduls `datetime`. Diese erwartet neben der Datumsangabe als String ebenfalls die Angabe des Datumformats. Ebenfalls interessiert uns das Alter der Tweets gemessen am Startzeitpunkt des Reveal Livestreams. Wie bereits vorhin beschrieben, begann dieser um 20:00 Uhr. Hierfür verwenden wir zum einen `lambda`-Funktionen, da wir so Funktionen nicht im Vorfeld extra definieren müssen, und zum anderen die `df.apply()`-Funktion, die die übergebene Funktion auf allen Zeilen des `DataFrame` ausführt.

Listing 12.6 Aufbereitung der Tweet-Zeitdaten

```

38 from datetime import datetime
39 # Zeitstempel der Tweets wird ins Datetime-Format umgewandelt
40 df["datetime"] = df.apply(lambda x: datetime.strptime(x["datetime"], "%Y-%m-%d %H:%M:%S"), axis = "columns")

```

Das Alter kann durch eine simple Subtraktion des Startzeitpunkts vom Datum des Tweets berechnet werden. Anschließend wandeln wir das Alter des Tweets in Sekunden um und berechnen das Alter des Tweets in Minuten und Stunden.

Listing 12.7 Aufbereitung der Tweet-Zeitdaten

```

42 # Funktion zur Berechnung des Tweetalters gemessen vom Startzeitpunkt des Livestreams
43 def calc_tweet_age(tweetdate):
44     start = datetime.strptime("11-06-2020 20:00:00", "%d-%m-%Y %H:%M:%S")
45     return tweetdate - start
46
47 # Erstellung der DataFrame-Spalten fuer das Tweet-Alter (in Sekunden, Minuten und Stunden)
48 df["tweetAge"] = df.apply(lambda x: calc_tweet_age(x["datetime"]), axis="columns")
49 df["tweetAgeSeconds"] = df.apply(lambda x: x["tweetAge"].total_seconds(), axis="columns")
50 df["tweetAgeMinutes"] = df.apply(lambda x: round(x["tweetAgeSeconds"]/60), axis="columns")
51 df["tweetAgeHours"] = df.apply(lambda x: round(x["tweetAgeMinutes"]/60), axis="columns")

```

Nun widmen wir uns den Inhalten der Tweets. Wie in Abschn. 12.1 beschrieben, gehören Hashtags und Mentions zu den wichtigsten Features der Plattform Twitter. Für unsere weitere Analyse sind diese jedoch nicht notwendig und es ist daher sinnvoll, sämtliche Hashtags, Mentions, aber auch URL, die in Tweets verwendet wurden, aus diesen zu löschen. Zusätzlich sollen unnötige Whitespaces, also beispielsweise Absätze oder Einschübeungen, entfernt werden. So haben URL und unnötiger Whitespace zwar keinen Informationsgehalt; jedoch erschweren und verlängern sie den Prozess der Sentiment-Bestimmung für die Tweets. Wir definieren zunächst die zu verwendenden Regex.

Listing 12.8 Definition von Regex

```

53 # Regex fuer Erkennung von Hashtags, Mentions und URLs
54 hashtag_regex = r"#(\w+)"
55 mention_regex = r"@(\w+)"
56 url_regex = r"(http[s]?://(?:[a-zA-Z]|[0-9]|[$_-@.&+]|!*\(\)\)|(?:%[0-9a-fA-F][0-9a-fA-F]))+)"

```

An dieser Stelle wird der Hashtag-Regex exemplarisch erklärt. Das erste # in der Variable `hashtag_regex` sucht zunächst das Zeichen # in einem String. Anschließend sucht \w nach dem nächsten Buchstaben und durch + wird schließlich definiert, dass nach beliebig vielen Buchstaben und Zahlen gesucht werden soll. In anderen Worten: Der Regex sucht also nach einem Hashtag, das von beliebig vielen Buchstaben oder Zahlen gefolgt wird. Sobald ein Leerzeichen oder eine andere Form von Whitespace auftaucht, ist die Suche beendet. Für weitere Informationen über Regex sei an dieser Stelle auf die Dokumentation zum Modul `re`

verwiesen.⁷ Anschließend suchen wir nach diesen Regex in den Tweet-Inhalten und ersetzen sie (mit `re.sub()`) durch nichts (Mentions) was einer Löschung gleicht.

Listing 12.9 Filtern der Tweet-Texte

```

58 # Entfernung von allen Hashtags, Mentions, URLs und Whitespace aus den Tweets
59 df["filtered_text"] = df.apply(lambda x: re.sub(hashtag_regex, "", x["text"]), axis =
    "columns")
60 df["filtered_text"] = df.apply(lambda x: re.sub(mention_regex, "", x["filtered_text"
    ]), axis = "columns")
61 df["filtered_text"] = df.apply(lambda x: re.sub(url_regex, "", x["filtered_text"]),
    axis = "columns")
62 whitespace_regex = r"[\t\r\n]"
63 df["filtered_text"] = df.apply(lambda x: re.sub(whitespace_regex, "", x["
    filtered_text"]), axis="columns")

```

Um das Stimmungsbild innerhalb dieser Tweets feststellen zu können, ist es notwendig, deren Sentiments zu berechnen. Eine solche Sentiment-Analyse ist ein Teilbereich des Text-Minings und erlaubt uns die Klassifizierung der Tweets in positive, neutrale und negative Tweets.

Hierfür existieren etliche Python-Module, die je nach Art der zu analysierenden Texte zu präferieren sind. Ebenfalls ist hier zu erwähnen, dass ein Großteil der Sentiment-Analysis-Module auf Basis von englischen Texten erstellt worden sind und daher ausschließlich mit dieser Sprache umgehen können. Es existieren zwar einige deutsche Alternativen zu diesen Modulen, jedoch bieten diese leider in vielen Fällen weniger Performance als ihre englischen Äquivalente.

Für unser Vorhaben verwenden wir Valence Aware Dictionary and sEntiment Reasoner (VADER).⁸ VADER verfolgt einen Lexikon- und Regelansatz zur Bestimmung von Sentiments und wurde speziell für Social-Media-Texte trainiert, weshalb es sich für unsere Zwecke ausgezeichnet eignet. VADER bietet ein Wörterbuch mit bereits klassifizierten Wörtern. In diesem Wörterbuch wurde jedem Wort ein bestimmter Sentiment-Score zugeteilt, und mit dem automatischen Erkennen von Satzbauteilen kombiniert, die zum Verstärken der eigenen Meinung dienen, wie beispielsweise die Verwendung von Ausrufezeichen und das Großschreiben von Adjektiven (Hutto & Gilbert, 2014).

Mithilfe von VADER können wir unser DataFrame mit den Spalten „compound“, „pos“, „neu“ und „neg“ erweitern. Bei „compound“ handelt es sich um die Summe des Sentiment-Scores aller Wörter innerhalb eines Texts, die anschließend zwischen -1 (extrem negativ) und $+1$ (extrem positiv) normalisiert wird. „Pos“, „neu“ und „neg“ geben jeweils den Anteil der positiven, neutralen und negativen Wörter innerhalb eines Texts an (Hutto & Gilbert, 2014).

⁷ Siehe: <https://docs.python.org/3/library/re.html>, zuletzt abgerufen am 1. Dezember 2020.

⁸ Siehe: <https://github.com/cjhutto/vaderSentiment>, zuletzt abgerufen am 1. Dezember 2020.

Listing 12.10 Berechnung der Tweet-Sentiments

```

65 import requests
66 from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
67 # Bestimmung des Tweet-Sentiment
68 analyzer = SentimentIntensityAnalyzer()
69 df["compound"] = df.apply(lambda x: analyzer.polarity_scores(x["filtered_text"])["
    compound"], axis="columns")

```

Nachdem das Pre-processing der Daten abgeschlossen ist, können wir nun damit anfangen, die ersten Blicke auf die Daten zu werfen. Falls gewünscht, können die Daten zunächst erneut in einer Datenbank zwischengespeichert werden, da man sich somit die Durchführung der in diesem Kapitel beschriebenen Schritte für die Zukunft sparen kann.

12.3 Social Media: Explore the Data

Nachdem wir uns die Twitter-Daten über den Reveal-Stream der PlayStation 5 besorgt haben und aufbereitet haben, werden wir diese in einem nächsten Schritt explorativ untersuchen. Im Nachgang an die explorative Analyse werden wir die Daten systematisch untersuchen.

Für die Visualisierungen der Daten verwenden wir hierbei die Python-Module matplotlib und seaborn.

Zunächst werden wir den Zeitverlauf der Tweets über den erfassten Zeitraum betrachten. Hierfür verwenden wir die .hist()-Funktion von matplotlib. Diese erlaubt uns, die einzelnen Tweets in beliebig viele Zeitabschnitte einzuteilen (Abb. 12.1).

Listing 12.11 Zeitverlauf der Tweets pro Stunde

```

86 import matplotlib.pyplot as plt
87 # Plotten der Anzahl der Tweets pro Stunde
88 plt.hist(df["tweetAgeSeconds"]/3600, bins = 24, cumulative=False, histtype="step")
89 plt.xlabel("Zeit seit Beginn (in Stunden)")
90 plt.ylabel("Anz. der Tweets")

```

In der resultierenden Grafik ist zu erkennen, dass, wie zu erwarten, der Großteil der Tweets um den Zeitpunkt des Livestreams entstanden ist. Da im Vorfeld des Livestreams nahezu keine Tweets zum Thema Playstation5 veröffentlicht wurden, werden diese nun aus unserem Datensatz entfernt. Ebenfalls filtern wir die letzten zwei Stunden der Tweets, da wir diese nicht mehr den verschiedenen Veröffentlichungen innerhalb des Livestreams zuordnen können.

Anschließend betrachten wir erneut den Zeitverlauf der Tweets. Jedoch ist es uns jetzt möglich, diesen detaillierter zu gestalten (Abb. 12.2).

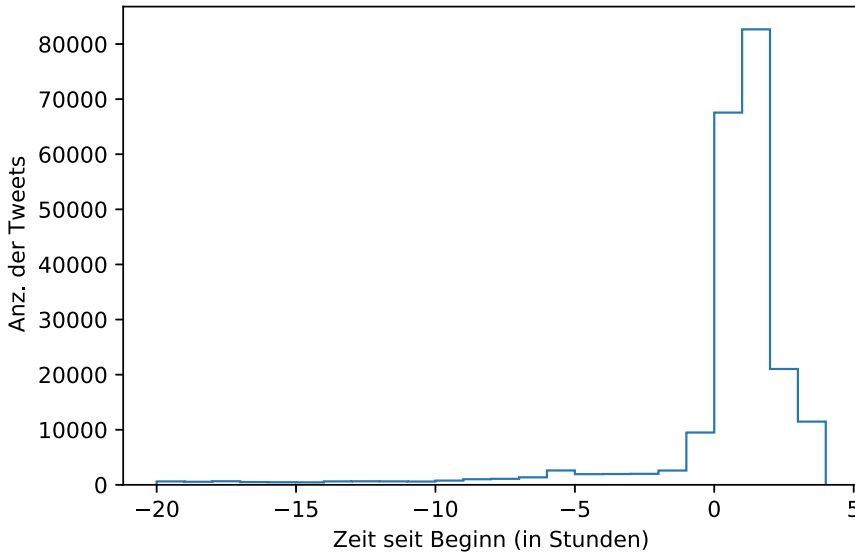


Abb. 12.1 Zeitverlauf der Tweets pro Stunde

Listing 12.12 Zeitverlauf der Tweets pro Minute

```

92 #Filtern der Tweets, sodass sie nur von Beginn bis zwei Stunden nach dem Livestream
    erfasst werden
93 df = df[df["tweetAgeSeconds"]/3600>=2]
94
95 # Plotten der Anzahl der Tweets pro Stunde (mit gefilterten Daten)
96 plt.hist(df["tweetAgeSeconds"]/3600, bins = 6*60, cumulative=False, histtype="step")
97 plt.xlabel("Zeit seit Beginn (in Stunden)")
98 plt.ylabel("Anz. der Tweets")

```

Wir können erkennen, dass zu Beginn des Livestreams die Anzahl der Kommentare stetig gestiegen ist und für die Dauer des Streams zwischen 1000 und 1500 Tweets pro Minute schwankte. Nach Beendigung des Livestreams schossen die Anzahl der Kommentare auf 4000 pro Minute in die Höhe. Eine mögliche Erklärung dieser Beobachtung ist, dass einige Zuschauer den Stream ungestört schauen wollten und daher erst im Anschluss daran getwittert haben. Ebenfalls wurde erst am Ende des Livestreams das Design der Playstation enthüllt, was als Highlight des Streams verstanden werden kann und somit eine weitere Erklärung für den Kurvenverlauf bietet.

Im nächsten Schritt betrachten wir den Meinungsverlauf der Tweets. Hierfür gruppieren wir zunächst die Tweets auf Minutenebene, um die starken Schwankungen innerhalb der einzelnen Tweets durch die Aggregation zu reduzieren. Den Mittelwert jeder Minute stellen wir mit der `.plot()`-Funktion dar und die Varianz des Sentiment mit der `.fill_between()`-Funktion. Schließlich fügen wir noch die Nulllinie in die Grafik und eine Trendlinie ein (Abb. 12.3).

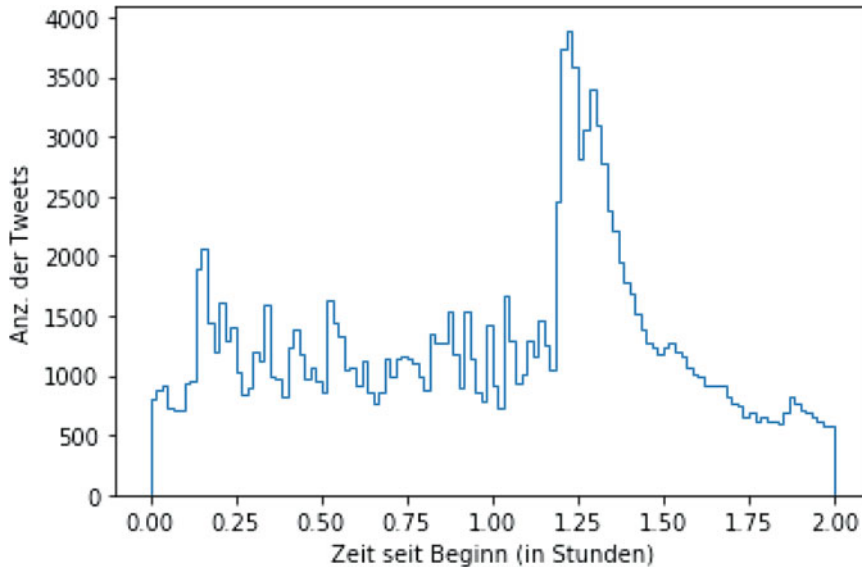


Abb. 12.2 Zeitverlauf der Tweets pro Minute

Listing 12.13 Zeitverlauf des Sentiment

```

100 import numpy as np
101 # Gruppieren der Tweets nach Minute
102 tweets = df.groupby("tweetAgeMinutes", as_index=False)
103 # Plotten des Durchschnitt-Sentiment
104 plt.plot(tweets.mean()["tweetAgeMinutes"], tweets.mean()["compound"], label = "Tweets
    ", color= "red")
105 # Erstellung des Varianzbereichs durch Addition und Subtraktion der Varianz vom
    Durchschnitt
106 plt.fill_between(tweets.mean()["tweetAgeMinutes"], tweets.mean()["compound"]-tweets.
    var()["compound"], tweets.mean()["compound"]+tweets.var()["compound"], alpha
    =0.5, facecolor='orange', label="Variance")
107 # Erstellung der Trendlinie
108 z = np.polyfit(tweets.mean()["tweetAgeMinutes"], tweets.mean()["compound"], 1)
109 p = np.poly1d(z)
110 plt.plot(tweets.mean()["tweetAgeMinutes"],p(tweets.mean()["tweetAgeMinutes"]), "r—",
    color = "black", label="Trend")
111 plt.hlines(0, -120, 180, linewidth = 1, linestyle = "dotted")
112 plt.legend(loc="upper left")
113 plt.ylabel("Durchschnittliches Sentiment")
114 plt.ylim(-0.5, 0.5)

```

In der resultierenden Grafik ist zu erkennen, das im Vorfeld des Livestreams die Stimmung positiv und nur wenig gespalten war. Während des Livestreams wurde sie jedoch schlagartig

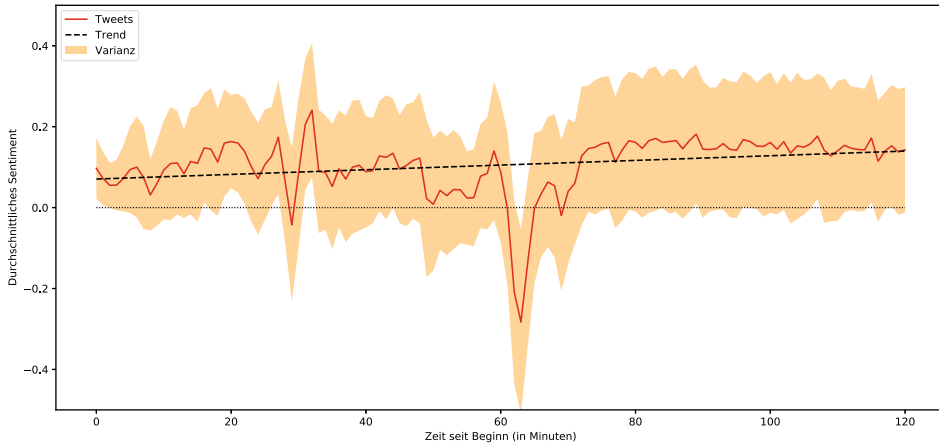


Abb. 12.3 Zeitverlauf des Sentiment

extrem negativ. Im Anschluss wurde jedoch das Sentiment wieder schnell positiv, wobei jedoch die Meinung sehr gespalten zu sein scheint. Dies ist an der relativ hohen und vor allem gestiegenen Varianz in der Sentiment-Variable zu erkennen. Hier stellt sich die Frage, wie das sehr negative Sentiment während des Livestreams zu erklären sein könnte und welche Videospielankündigungen dazu geführt haben.

Um diese Frage zu beleuchten, benötigen wir jedoch noch die genauen Abschnitte des Livestreams, in denen die unterschiedlichen Videospiele enthüllt worden sind. Hierfür schauen wir uns die Aufzeichnung des Livestreams⁹ an und erfassen die Timestamps für jede einzelne Veröffentlichung. Hieraus wird anschließend eine Liste von Dictionaries namens timestamps erstellt. Die Timestamps der Spiele können anschließend in die oben erstellte Grafik eingefügt werden.

Listing 12.14 Einbettung der Timestamps

```

170 # Fuer jeden Timestamp wird eine vertikale Linie erstellt
171 for timestamp in timestamps:
172     # Dabei wird das Timedelta als Abstand zur y-Achse genommen
173     plt.axvline(x=timestamp["timestamp"].total_seconds()/60, linestyle=":", color="
        grey")
174     # Und der Text um 90 Grad gedreht
175     plt.text(x=timestamp["timestamp"].total_seconds()/60, y=-0.48, s=timestamp["game"
        ], rotation=90)

```

Die resultierende Grafik beinhaltet nun die Timestamps der Veröffentlichungen, sodass die Schwankungen im Meinungsbild diesen zugeordnet werden können (Abb. 12.4).

Hierbei sind mehrere Faktoren zu bedenken, die diese Betrachtung und eine Interpretation der Ergebnisse erschweren.

⁹ <https://www.youtube.com/watch?v=8apW-UFhGZg>


```

182     return "GTA V"
183 elif any(word in text for word in ["spider", "miles morales"]):
184     return "Spider-Man: Miles Morales"
185 elif any(word in text for word in ["gt7", "gran turismo", "turismo"]):
186     return "Gran Turismo 7"
187 elif any(word in text for word in ["ratchet", "clank", "rift apart"]):
188     return "Ratchet & Clank: Rift Apart"
189 # Und so weiter fuer die restlichen Spiele

```

Auch an dieser Stelle ist zu erwähnen, dass dieses Vorgehen ebenfalls nicht perfekt ist und nicht garantiert, dass alle Tweets tatsächlich korrekt zugeordnet werden. So ist es beispielsweise möglich, dass in einem Tweet mehrere Spiele erwähnt werden. In diesem Fall würde der Tweet dem Spiel zugeordnet werden, das in der verwendeten If-else-Bedingung zuerst erwähnt wird. Ebenfalls ist es denkbar, dass gewisse Wörter, wie beispielsweise Rockstar, in einem Tweet verwendet wurden, obwohl in diesem Tweet nicht über das Spiel GTA V geschrieben wurde. Im Rahmen dieser Analyse sind diese Fehlerquellen jedoch zu vernachlässigen, unter anderem weil die Wörter so gewählt wurden, dass diese möglichst eindeutig einem Spiel zugeordnet werden können (Abb. 12.5).

Im nächsten Schritt visualisieren wir die Verteilung der Spieleerwähnungen mittels eines Bar-Plots. Hierfür gruppieren wir die Tweets zunächst nach den zugeordneten Spielen und zählen anschließend die Vorkommen der einzelnen Nennungen. Im nächsten Schritt entfernen wir aus der entstandenen Liste alle Tweets, die nicht zugeordnet werden konnten, wandeln die Liste in ein DataFrame um, sortieren dieses absteigend und erstellen schließlich die Grafik mit `df.plot.bar()`.

Listing 12.16 Erstellung eines Bar-Plots

```

243 # Gruppierung der Tweets je nach Videospiel
244 game_distribution = df.groupby("game")['_id'].nunique()
245 # Tweets, die keinem Spiel zugeordnet werden konnten, werden entfernt
246 game_distribution.pop("Other")
247 # Umwandlung in DataFrame
248 game_distribution = game_distribution.to_frame("Tweets")
249 # Sortieren nach der Anzahl der Tweets pro Videospiel
250 game_distribution = game_distribution.sort_values("Tweets", ascending=False)
251 # Erstellung eines Barplots
252 game_distribution.plot.bar(legend=None)
253 plt.xlabel("Spielname")
254 plt.ylabel("Anz. der Tweets")

```

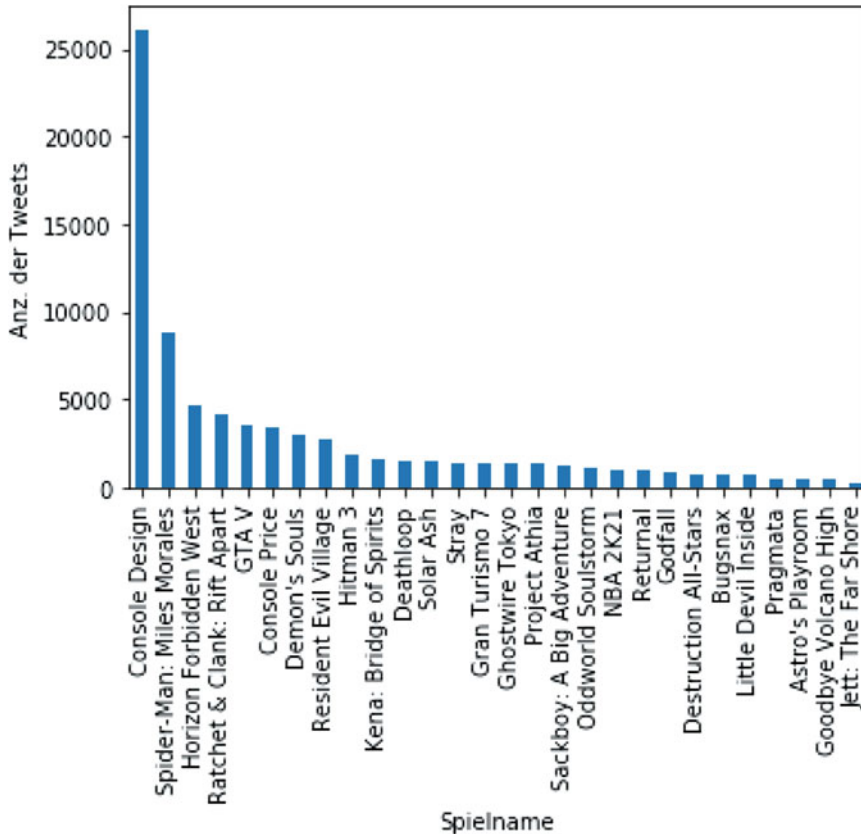


Abb. 12.5 Verteilung der Tweets auf die unterschiedlichen Videospiele

Hier ist zu erkennen, dass der Großteil der Tweets (wobei die nicht zugeordneten Tweets hierbei außen vor gelassen wurden) das Design der Konsole kommentiert. In Bezug auf die Videospiele wurden Spiderman, Horizon, Ratchet & Clank und GTA V am meisten kommentiert, während Pragmata, Astro's Playroom, Goodbye Volcano High und Jett scheinbar wenig Beachtung fanden.

12.4 Social Media: Model the Data

Bis hierher konnten wir uns einen explorativen Überblick über die Daten verschaffen, diese weiter aufbereiten und unsere Fragestellung herauskristallisieren. Nun wollen wir die von uns aufgestellten Fragen beantworten:

Welche Enthüllungen wurden besonders negativ oder besonders positiv wahrgenommen?

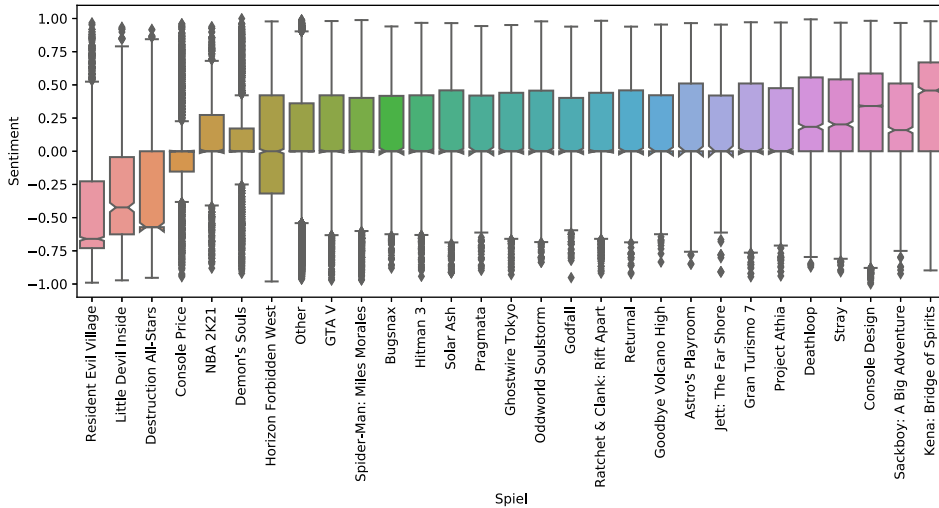


Abb. 12.6 Sentiment-Boxplots der einzelnen Videospiele

Um diese Frage zu beantworten, ist es notwendig, dass wir die Gesamtmenge der Tweets zunächst nach den erwähnten Videospiele aufteilen. Anschließend können wir uns Unterschiede im Meinungsbild zu diesen Spielen anschauen und diese Unterschiede mithilfe einer sogenannten Analysis of Variance (ANOVA) auf statistische Signifikanz prüfen.

Anschließend können wir die `boxplot()`-Funktion des Python-Moduls `seaborn` verwenden, um für jedes Videospiel einen einzelnen Boxplot zu erstellen. Normalerweise wären die Namen der Videospiele auf der x-Achse alphabetisch sortiert, jedoch ändern wir dies, indem wir die Variable `order` anlegen, die die Indizes der nach Sentiment-Mittelwert sortierten Tweet-Gruppen enthält (Abb. 12.6)

Listing 12.17 Erstellung von Boxplots für jedes Videospiel

```

256 import seaborn as sns
257 # Sortierung nach dem Durchschnitt-Sentiment
258 order = df.groupby("game")["compound"].mean().sort_values().index
259 plt.figure(figsize=(29*0.4,4))
260 # Erstellung von Boxplots fuer jedes Videospiel
261 ax = sns.boxplot(x="game", y='compound', data=df, notch=True, order=order)
262 ax.set_xticklabels(ax.get_xticklabels(), rotation=90)
263 plt.xlabel("Spiel")
264 plt.ylabel("Sentiment")

```

Die resultierende Grafik ist nun aufsteigend nach dem Sentiment-Mittelwert sortiert. In dieser ist bereits erkennbar, dass die Enthüllung der Spiele Resident Evil, Little Devil Inside

sowie Destruction All-Stars sehr negativ kommentiert wurden, während die Enthüllung des Konsolendesigns sowie der Spiele Sackboy und Kena sehr positiv aufgenommen wurden.

Nun wollen wir diese Unterschiede noch auf ihre statistische Signifikanz überprüfen. Hierfür verwenden wir eine sogenannte ANOVA, die testet, ob sich die Mittelwerte mehrerer unabhängiger Gruppen unterscheiden, die durch eine unabhängige Variable definiert werden.

Das Python-Modul statsmodels.api ermöglicht die Durchführung einer solchen ANOVA wie folgt:

Listing 12.18 Durchführung einer ANOVA zum Testen auf Unterschieden im Sentiment zwischen Videospielen

```

266 import statsmodels.api as sm
267 from statsmodels.formula.api import ols
268 # Durchführung einer ANOVA fuer das Durchschnitts-Sentiment der Videospiele
269 model = ols('compound ~ C(game)', data=df).fit()
270 aov_table = sm.stats.anova_lm(model, typ=2)
271 print(aov_table)

```

Bevor wir jedoch die Ergebnisse der ANOVA auswerten können, müssen wir zunächst überprüfen, ob die Voraussetzungen für ihre Anwendung erfüllt sind. Diese sind:

1. Normalverteilung der abhängigen Variable in jeder der Gruppen
2. Homogenität der Varianzen in jeder der Gruppen

Die Normalverteilung der abhängigen Variablen können wir mit einem Shapiro-Wilk-Test überprüfen.

Listing 12.19 Shapiro-Wilk-Test zum Überprüfen der Normalverteilung des Sentiment für jedes Videospiel

```

273 # Shapiro-Wilk-Test auf Normalverteilung der Residuen
274 print(stats.shapiro(model.resid))
275 #(0.9651668667793274, 0.0)

```

Der Test ist signifikant ($W = 0.965$, $p = 0.000$), was darauf hindeutet, dass keine Normalverteilung in der Sentiment-Variable vorliegt. Da jedoch der Shapiro-Wilk-Test bei sehr großen Stichproben nicht mehr akkurat ist, überprüfen wir die Normalverteilung nochmal visuell, indem wir die Residuen plotten (Abb. 12.7).

Listing 12.20 Plot der Residuen

```

277 # Plotten der Residuen zur Ueberpruefung der Normalverteilung
278 normality_plot, stat = stats.probplot(model.resid, plot= plt, rvalue= True)
279 plt.xlabel("Theoretische Quantile")
280 plt.ylabel("Geordnete Werte")

```

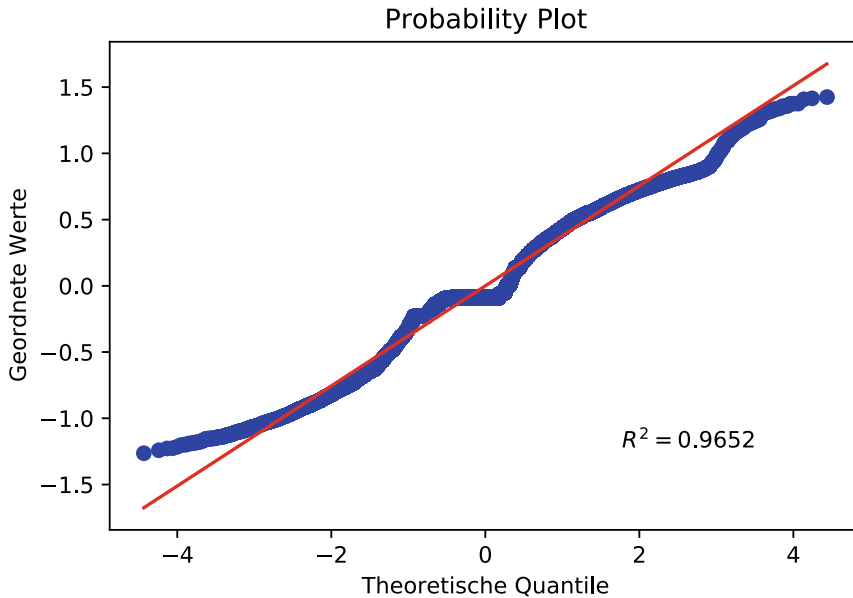


Abb. 12.7 Grafische Überprüfung der Normalverteilung des Sentiment

Im Fall einer Normalverteilung der Variable würden die einzelnen Datenpunkte zu einem großen Teil auf der hier vorliegenden Diagonale liegen und kein anderes Muster erkennbar sein. In unserem Fall folgen die Daten keiner Geraden, sondern folgen anscheinend einer anderen Verteilung. Daher können wir auch mit der visuellen Überprüfung nicht auf eine Normalverteilung schließen und folglich keine ANOVA durchführen. Die Verteilung der Daten ist durch die Verwendung von VADER begründet, da hier kurze Tweets als sehr negativ oder sehr positiv klassifiziert werden und Tweets ohne entsprechende Schlüsselwörter, die Informationen über das Sentiment geben, mit einem Sentiment von 0 klassifiziert werden.

Die zweite Voraussetzung einer ANOVA, die Homogenität der Varianzen in jeder der Gruppen, kann mit einem Levene-Test oder auch visuell überprüft werden. Auf den Levene-Test wird an dieser Stelle verzichtet, da sich die Anwendung einer ANOVA aufgrund der Verletzung der ersten Voraussetzung nicht anbietet. Alternativ können wir Boxplots für jede Gruppe erstellen und die Größe der Boxen miteinander vergleichen. Dies haben wir bereits in Abb. 12.6 getan. Hier ist erkennbar, dass sich auch die Varianzen der Gruppen unterscheiden. So hat beispielsweise Demon's Souls ein sehr geringe Varianz, während Kena eine sehr hohe Varianz aufweist.

Bei einer Verletzung einer der Voraussetzungen der ANOVA existiert jedoch die Möglichkeit, stattdessen einen Kruskal-Wallis-Test durchzuführen, der überprüft, ob sich die zentralen Tendenzen mehrerer unabhängiger Stichproben unterscheiden.

Hierfür verwenden wir die `kruskal()`-Methode aus dem `scipy.stats`-Modul.

Listing 12.21 Durchführung eines Kruskal-Willis-Tests auf Unterschiede im Sentiment zwischen Videospielen

```

282 from scipy.stats import kruskal
283 # Erstellung einer Liste der Sentiment-Werte der Videospiele
284 games = df.groupby("game")
285 games_list = [games.get_group(g)["compound"] for g in games.groups]
286
287 # Durchfuehrung eines Kruskal-Wallis-Tests
288 print(kruskal(*games_list))
289 #KruskalResult(statistic=9768.108707789559, pvalue=0.0)

```

Die `kruskal()`-Methode erwartet mehrere Listen als Parameter, sodass wir zunächst die Tweets erneut nach den erwähnten Spielen gruppieren und anschließend für jedes Spiel eine Liste der Sentiments erstellen.

Auf Basis dieser Listen können wir den Kruskal-Willis-Test durchführen und erkennen, dass die Sentiments zu den unterschiedlichen Videospielen sich statistisch signifikant unterscheiden, sodass die beobachteten Unterschiede in Abb. 12.6 interpretiert werden können.

12.5 Social Media: Interpretation der Ergebnisse

Das Ziel unserer Analyse war es, anhand von Twitter-Daten, die während eines Livestreams entstanden, herauszufinden, welche Enthüllungen innerhalb des Livestreams besonders gut bzw. besonders schlecht bei den Zuschauern ankamen.

Um diese Frage zu beantworten, haben wir zunächst den Zeitverlauf des Sentiment während des Livestreams betrachtet und hier die Timestamps der einzelnen Reveals eingezeichnet (siehe Abb. 12.4). Hier hat sich bereits angedeutet, dass enorme Schwankungen durch die Reveals gewisser Spiele entstanden sind. So stieg das durchschnittliche Sentiment durch die Enthüllung von Kena: Bridge of Spirits stark an, während es durch die Enthüllung von Destruction All-Stars und Resident Evil: Village stark sank.

Im nächsten Schritt haben wir die Gesamtmenge der Tweets basierend auf mehrere Schlüsselwörter den einzelnen Spielen zugeordnet und das Sentiment der Tweets über die einzelnen Spiele mittels Boxplots visualisiert. In der hier resultierenden Abb. 12.6 ist ebenfalls erkennbar, dass Resident Evil: Village, Little Devil Inside und Destruction All-Stars ein sehr schlechtes Sentiment erfahren haben und das Design der Konsole, Sackboy und Kena sehr positiv aufgenommen wurden. Im letzten Schritt haben wir die Unterschiede im Sentiment auf ihre statistische Signifikanz überprüft. Hierfür haben wir einen Kruskal-Willis-Test verwendet, da die Voraussetzung einer ANOVA nicht erfüllt waren.

Dieser unterstützt unsere Vermutung, dass die verschiedenen Spiele tatsächlich sehr unterschiedlich wahrgenommen wurden. Da die Erkenntnisse aus den beiden Abb. 12.4 und 12.6 übereinstimmen, können wir ebenfalls die diskutierten Probleme dieser beiden Vorgehensweisen vernachlässigen.

Insgesamt konnten wir durch unsere Analyse zeigen, dass der Enthüllungs-Livestream der Playstation 5 zwar im Durchschnitt positiv diskutiert wurde, jedoch die Spiele Resident Evil: Village, Little Devil Inside, Destruction All-Stars negativ aufgenommen wurden. Ebenfalls negativ aufgenommen wurde die (fehlende) Enthüllung des Preises der Playstation 5. Der Großteil der übrigen Spiele wurden relativ neutral, aber mit einer positiven Tendenz wahrgenommen. Ausnahmen hierzu bilden schließlich Deathloop, Stray, Sackboy: A Big Adventure sowie Kena: Bridge of Spirits, die allesamt sehr positiv von der Twitter-Community aufgenommen wurden. Ebenfalls wurde das enthüllte Design der Playstation 5 sehr positiv wahrgenommen.

Zusammenfassend haben wir in diesem Kapitel gelernt, wie man mithilfe von Python und der Twitter-API Tweets zu einem bestimmten Thema oder Produkt erheben kann, diese anschließend aufbereitet, ihr Sentiment bestimmt und dieses schließlich verwenden kann, um Rückschlüsse auf das Meinungsbild zu einem Thema bzw. einem Produkt ziehen zu können.



Daniel Franzmann

Cloud Computing und speziell Cloud-basierte Webservices sind für Unternehmen und Privatpersonen eine wartungsfreundliche und kostensparende Alternative zu komplexen und teuren Server-Hosting-Projekten. Praktisch alle großen Technologiekonzerne wie Microsoft, Google oder Amazon haben in den letzten Jahren solche Cloud-Computing-Dienste gegründet, wobei die Amazon Web Services (AWS) aktuell den Marktführer darstellen. AWS umfasst (Stand September 2020) mehr als 175 Dienste, die von Endkunden im *Pay-as-you-go*-Verfahren genutzt werden können. Am bekanntesten sind hierbei die Amazon-Elastic-Compute-Cloud(EC2)-Instanzen. Hierbei handelt es sich um skalierbare und von Amazon gewartete Server. Zudem gibt es noch den Simple Storage Service (S3) von Amazon. Dabei handelt es sich um eine Cloud-basierte Lösung zur Datenspeicherung. Das folgende Kapitel demonstriert, wie mit einfachen Mitteln und für Neukunden völlig kostenlos, eine EC2-Instanz gestartet wird, um einen in Python geschriebenen Web Crawler zum täglichen Extrahieren von YouTube-Kommentaren zu nutzen. Hierbei werden z. B. die Kommentare von YouTube-Videos mit Bezug auf den SARS-CoV2-Virus extrahiert und selbige in einem zweiten Schritt mit Natural Language Processing ausgewertet und die Ergebnisse durch Graphen und Word Clouds visualisiert. Um die Auswertungen interessanter zu gestalten, werden die Kommentare grob in zwei Gruppen unterteilt: 1) Kommentare, die das Wort *fake* enthalten, und 2) Kommentare, die das Wort *fake* **nicht** enthalten. Zuletzt werden diese Ergebnisse auf einer statischen HTML-Seite, in einem S3-Bucket, als öffentlich verfügbare Webseite gehostet.

D. Franzmann (✉)
Frankfurt am Main, Deutschland

13.1 Cloud Web Services: Fragestellung

Damit die Nutzung von AWS im Allgemeinen, die Implementierung eines Web Crawler auf einer EC2-Instanz und die Visualisierung der extrahierten Daten in einem S3-Bucket verständlicher gestaltet werden, werden diese Schritte anhand eines konkreten Fallbeispiels konzipiert. Das Ziel des folgenden Kapitels ist es, die täglichen Sentiments¹ von YouTube-Kommentaren zum Thema SARS-CoV2 zu messen, Unterschiede zwischen *fake*- und *non-fake*-Kommentaren zu visualisieren und diese Ergebnisse in einem S3-Bucket als Webseite zu hosten und öffentlich zur Verfügung zu stellen. Es werden also folgende Schritte durchgeführt: 1) AWS-Account anlegen, 2) eine EC2-Instanz und einen YouTube API-Zugriff erstellen, 3) einen in Python geschriebenen Web Crawler erstellen, der automatisiert und täglich auf der EC2-Instanz ausgeführt wird, 4) die Ergebnisse von der EC2-Instanz herunterladen sowie lokal zu analysieren und 5) die Visualisierungen und Analysen in einem S3-Bucket öffentlich verfügbar machen. Selbstverständlich kann die hier gezeigte Vorgehensweise jederzeit abgewandelt und für andere Data-Science-Projekte verwendet werden. Wichtig ist hierbei noch, dass die Code Snippets in den folgenden Abschnitten nicht den gesamten Code darstellen. Der vollständige Code, der zur Ausführung benötigt wird, kann auf der Webseite des Buchs heruntergeladen werden.

13.2 Cloud Web Services: Pre-processing

Im ersten Schritt geht es darum, einen neuen AWS-Account anzulegen. Dies gelingt durch einen Klick auf den Button *AWS-Konto erstellen* auf der deutschen Seite von AWS.² Die nachfolgenden Informationen zur Anmeldung basieren auf dem Stand vom 01.09.2020, aber es ist davon auszugehen, dass der Umfang der Services auch in Zukunft unverändert bleiben wird oder sich nur marginal ändert. Trotzdem übernehme ich keine Haftung für etwaige Kosten, die durch Änderungen in den Servicebedingungen von AWS entstehen. Die Sprache der Webseite kann im unteren Teil der Webseite auf Deutsch geändert werden, trotzdem verbleiben einige Schritte weiterhin auf Englisch.

Schon im ersten Registrierungsschritt wird darauf hingewiesen, dass neuen AWS-Konten zwölf Monate kostenloser Zugriff auf die wichtigsten Services, wie z.B. EC2 und S3, gewährt wird. Die Registrierung erfolgt relativ unkompliziert via Name, E-Mail-Adresse, Passwort, AWS-Kontobezeichnung, Telefonnummer und Postanschrift. Im nächsten Schritt wird eine Kreditkarte benötigt, um die Identität des Kontoerstellers zu überprüfen und mögliche Kosten abzurechnen. Solange aber die Schritte dieses Kapitels befolgt werden, entstehen keine Kosten. Generell ist das Kostenmodell von AWS sehr einsteigerfreundlich und weist auf potenzielle Kosten transparent hin. Im nächsten Schritt muss die Identität per SMS/Te-

¹ Eine Sentiment-Analyse beschreibt die Textauswertung mit dem Ziel, die Stimmung bzw. geäußerte Haltung (positiv oder negativ) zu detektieren.

² Siehe: <https://aws.amazon.com/de>, zuletzt abgerufen am 1. September 2020.

lefon bestätigt werden und es kann abschließend einer von drei Support-Plänen gewählt werden. Hierbei ist der *Basic Plan* kostenlos und dazu noch vollkommen ausreichend für die folgende Implementierung.

Nach der erfolgreichen Erstellung eines AWS-Accounts ist es nun möglich, verschiedene Tutorials bzw. Anleitungen auszuprobieren und so die Dienste von AWS näher kennenzulernen. Dies kann durch das Ausfüllen der Felder *My Role is* und *I am interested in* gestartet werden.

Zunächst werden wir uns bei der *AWS-Konsole* anmelden, um im nächsten Schritt eine EC2-Linux-Instanz zu starten, auf der der Web Crawler gestartet und ausgeführt werden soll. Hierbei stellt die EC2-Instanz einen virtuellen Server dar, auf dem Programme oder Scripts ausgeführt werden können. Die Erstellung und der Zugriff werden in den nachfolgenden Schritten erläutert.

Nach dem erfolgreichen Login (als Stammbenutzer mit E-Mail-Adresse und Passwort) wird man zum Dashboard der sogenannten *AWS-Managementkonsole*³ weitergeleitet, die es ermöglicht, schnell und unkompliziert auf die verschiedensten AWS-Dienste zuzugreifen. Dies kann im Dashboard entweder per Suchfunktion, über die zuletzt besuchten Services oder über die komplette Übersicht an Services erfolgen. Am einfachsten ist es, direkt *EC2* in die Suche einzugeben, um zu einem Dashboard der EC2-Instanzen weitergeleitet zu werden. Um eine EC2-Instanz zu starten, ist es nun notwendig, den orangefarbenen Button *Instance starten* auszuwählen und dies erneut im Drop-down-Menü zu bestätigen. Nun wird man zum ersten von mehreren Schritten weitergeleitet. Alle Informationen und Schritte des Startprozesses sind nach dem Stand des 01.09.2020 dokumentiert, werden aber auch zukünftig vermutlich nicht sehr abweichen.

1. *Step 1: Choose an Amazon Machine Image (AMI)*: In diesem ersten und wichtigsten Schritt kann das Betriebssystem bzw. die Version der EC2-Instanz ausgewählt werden. Zusätzlich wird dem Nutzer, neben generellen Informationen zu den Betriebssystemen, auch ein Banner angezeigt, das zeigt, welche der Betriebssysteme in der kostenlosen zwölfmonatigen Einführungsphase kostenlos verfügbar sind. Meine Empfehlung ist die Nutzung eines Ubuntu-Servers der Version 20.04 und einer SSD als Speichermedium. Als Architektur wird die Standardeinstellung 64-bit (x86) gewählt. Weiterhin kann auf der linken Seite auch eine Checkbox aktiviert werden, die nach Aktivierung nur die kostenlos verfügbaren EC2-Instanzen auswählt. Anschließend wird der Button *Select* ausgewählt, um zum zweiten Schritt zu wechseln.
2. *Step 2: Choose an Instance Type*: Nun kann die Hardwarekonfiguration der EC2-Instanz ausgewählt werden. Standardmäßig ist die kostenlose *Free-tier-eligible*-Version ausgewählt, die eine virtuelle CPU und 1 GiB RAM aufweist. Diese Konfiguration ist für unsere Zwecke völlig ausreichend. Wichtig ist im nächsten Schritt, **nicht** direkt auf den blauen Button *Review and Launch* zu klicken, sondern noch die weiteren Konfigurations-

³ Siehe: <https://aws.amazon.com/de/console>, zuletzt abgerufen am 1. September 2020.

möglichkeiten zu prüfen. Dies geschieht durch den Klick auf den Button *Next: Configure Instance Details*.

3. *Step 3: Configure Instance Details*: Im dritten Schritt werden erst einmal keine Veränderungen an den Einstellungen vorgenommen. Theoretisch könnte hier aber festgelegt werden, wie z. B. das *Shutdown behavior* funktionieren soll: das heißt, ob eine EC2-Instanz nur temporär gestoppt (sie kann über das EC2-Dashboard mit denselben Einstellungen erneut gestartet werden) oder gänzlich terminiert wird (die EC2-Instanz müsste komplett neu initialisiert werden). Als nächstes wird der Button *Next: Add Storage* ausgewählt, um zum nächsten Schritt zu gelangen.
4. *Step 4: Add Storage*: In diesem vierten Schritt wird die Speicherart und -größe bestimmt. Dieser Schritt ist relevant, gerade wenn ein Web Crawler genutzt werden soll, der große Datenmengen extrahiert und die Speicherung dieser Daten in erster Linie auf der EC2-Instanz erfolgen soll. Die Größe des Festplattenspeichers der EC2-Instanz (Size GiB) kann hier auf bis zu 30 GiB im kostenlosen Zwölf-Monats-Modus erhöht werden. Sollen größere Datenmengen gespeichert werden, ist es sinnvoll, dies in einer speziellen Datenbank vorzunehmen, der entsprechende AWS-Service heißt dabei *AWS RDS*. Diese Art der Speicherung und Implementierung ist aber komplexer und wird an dieser Stelle nicht behandelt, da eine lokale Speicherung auf der EC2-Instanz für die Zwecke der Veranschaulichung völlig ausreichend ist. Nach der Änderung der Speichergöße auf z. B. 10 GiB geht es weiter mit einem Klick auf *Next: Add Tags*.
5. *Step 5: Add Tags*: In diesem Schritt kann die EC2-Instanz nun mit Tags versehen werden; dies ist dann besonders hilfreich, wenn es notwendig ist, den Überblick über eine Vielzahl von EC2-Instanzen zu behalten und mögliche Kosten der einzelnen EC2-Instanzen transparenter und speziellen Kostenstellen zuschlüsseln zu können. In unserem Beispiel wird diese Funktionalität aber nicht genutzt und es werden keine Tags erstellt. Erneut wird zum nächsten Schritt durch einen Klick auf *Next: Configure Security Group* übergeleitet.
6. *Step 6: Configure Security Group*: Hier wird der externe Zugriff auf die EC2-Instanz konfiguriert. Wichtig ist, dass ein Zugriff mittels SSH über den Port 22 zulässig ist. Standardmäßig ist der Zugriff von jeder IP-Adresse möglich (*Source: Custom: 0.0.0/0*); dies kann und sollte bei sensiblen Daten aber dringend angepasst werden, sodass nur der Zugriff über die eigene IP-Adresse oder den IP-Bereich zugelassen wird. Aktuell werden die Einstellungen aber nicht verändert. Durch die Auswahl des Buttons *Review and Launch* geht es zum finalen Schritt.
7. *Step 7: Review*: Der letzte Schritt beinhaltet eine erneute Überprüfung aller gewählten Einstellungen. Abschließend kann die EC2-Instanz mit der Auswahl des blauen Buttons *Launch* gestartet werden. Ein neues Fenster erscheint und es muss zuletzt noch ein sogenanntes *Key Pair* erstellt werden. Dies ist die Bezeichnung für einen Kryptographieschlüssel, der zur späteren Authentifizierung (z. B. über SSH oder SFTP) genutzt wird. Die Auswahl im Drop-down-Menü ist deshalb *Create a new key pair*, wobei der zu vergebende Name nebensächlich ist. Das *Key Pair* sollte aber anschließend sicher lokal abgespeichert werden. Nach dem Speichern kann nun der Button *Launch Instances* ausgewählt werden.

Nach dem erfolgreichen Starten der EC2-Instanz (dies kann einige Minuten dauern) kann über das EC2-Dashboard und einen Klick auf *Laufende Instances* der Status der EC2-Instanz eingesehen werden. Hier können auch durch einen Rechtsklick auf den Instanzennamen (*Name*) oder die *Instance-ID*-Parameter und -Einstellungen verändert, die EC2-Instanz gestoppt bzw. terminiert oder der Zugriff per Web-Interface gestartet werden (*Verbinden*).

Anlegen des YouTube-API-Zugriffs

Im nächsten Schritt werden die Erstellung eines YouTube-Developer-Accounts und die Generierung eines API-Schlüssels beschrieben. Dies ist notwendig, um die YouTube-Kommentare abzurufen, die im Rahmen des Projekts analysiert werden sollen. Als erstes muss deshalb ein Google-Account erstellt werden.⁴ Hierzu müssen Vorname, Nachname, Benutzername und Passwort in das Webformular eingegeben werden. Im Anschluss muss nur noch das Geburtsdatum angegeben und den Datenschutzbedingungen zugestimmt werden. Im nächsten Schritt wird der YouTube-API-Schlüssel über die *Google Cloud Console*⁵ erstellt. Dazu muss initial den Nutzungsbedingungen der *Google Cloud Platform* zugestimmt werden.

1. *Neues Projekt erstellen*: Zuerst wird ein neues Projekt über die Auswahl des Buttons *Projekt auswählen* in der oberen Navigationsleiste der Webseite erstellt. Nun kann durch die Auswahl von *Neues Projekt* ein neues Projekt erstellt werden, wobei ein beliebiger Projektname vom Nutzer ausgewählt werden kann. Der Speicherort des Projekts ist hierbei nicht weiter von Bedeutung.
2. *YouTube-API-Schlüssel erstellen*: Im nächsten Schritt wird das neu erstellte Projekt ausgewählt und anschließend über das Suchfeld in der oberen Navigationsleiste nach *YouTube Data API v3* gesucht. Diese von Google bereitgestellte API ermöglicht es dem Web Crawler, auf die jeweiligen YouTube-Videos, dazugehörige Metadaten und die zugehörigen Kommentare zuzugreifen. Im nächsten Schritt wird die API nun durch die Auswahl der blauen Schaltfläche *Aktivieren* aktiviert.
3. *Anmeldedaten erstellen*: Nach der erfolgreichen Aktivierung der API, muss nur noch ein sogenannter API-Schlüssel erstellt werden, damit dieser später vom Web Crawler als Authentifizierungsmethode genutzt werden kann. Dies erfolgt durch die Auswahl des Buttons *Anmeldedaten Erstellen*. Hierzu wird erneut die *YouTube Data API v3* ausgewählt und als Plattform der *Webbrowser (Javascript)* selektiert. Weiterhin soll über die API nur auf *öffentliche Daten* zugegriffen werden. Nach der Auswahl von *Welche Anmeldedaten brauche ich?* wird der API-Schlüssel generiert und kann anschließend in die Zwischenablage kopiert und abgespeichert werden.

Nach diesen drei Schritten ist es nun möglich, einen Web Crawler in Python zu erstellen, der nach bestimmten Schlagworten in YouTube sucht und die jeweiligen Kommentare der

⁴ Siehe: <https://accounts.google.com>, zuletzt abgerufen am 1. September 2020.

⁵ Siehe: <https://console.cloud.google.com>, zuletzt abgerufen am 1. September 2020.

Videos extrahiert und speichert.

Entwicklung des Python Web Crawlers

In diesem Abschnitt wird ein simpler Web Crawler in Python implementiert und seine Funktionen jeweils kurz beschrieben. Der Web Crawler soll hierbei eine Stichprobe von YouTube-Kommentaren nach einem bestimmten Suchbegriff extrahieren. Im ersten Schritt wird der Web Crawler zuerst lokal auf dem Rechner getestet, bevor er anschließend auf den EC2-Server hochgeladen und automatisiert wird.

Listing 13.1 Benötigte Libraries und Initialisierung des API-Schlüssels

```

8 from youtube_api import YouTubeDataAPI
9 import pandas as pd
10 import os
11 from langdetect import detect
12 auth_key = 'XXXXXXXXXXXXXXXXXXXXXXXXX'
13 yt = YouTubeDataAPI(auth_key)
```

Im ersten Schritt werden die benötigten Libraries importiert. Diese sind vor allem die 1) *YouTubeDataAPI*, die notwendig für den Zugriff auf die YouTube-Kommentare ist, 2) *pandas*, das für die Erstellung des DataFrame genutzt wird, 3) *as*, das für das Setzen des Working Directory genutzt wird und 4) *langdetect*, eine Bibliothek, die die Sprache des jeweiligen YouTube-Kommentars erkennen soll. Weiterhin wird der im vorherigen Kapitel erstellte YouTube-API-Schlüssel im String-Format als Variable *auth_key* definiert und initialisiert. Dieser muss an die Stelle der Platzhalter-Zeichen in Zeile 12 des Codes kopiert werden.

Listing 13.2 Suche nach dem Begriff „corona“ und Speichern der Video-Links

```

16 video_search = yt.search('corona',max_results=100)
17 video_links = []
18 video_titles = []
19 video_publish_date = []
20 for i in video_search:
21     video_links.append(i['video_id'])
22     video_titles.append(i['video_title'])
23     video_publish_date.append(i['video_publish_date'])
```

Zuerst werden der Suchbegriff „corona“ und die Anzahl der Ergebnisse begrenzt (Zeile 16). Dies ist notwendig, da YouTube die Anzahl der Anfragen, die über einen einzelnen API-Schlüssel generiert werden, täglich limitiert. Im nächsten Schritt werden drei leere Listen definiert, die für die Speicherung der Video-Daten (vor allem der Video-Links) verwendet werden. Die For-Schleife (siehe Abschn. 6.2.1 für die Grundlagen) von Zeile 20 bis Zeile 23 durchläuft die Ergebnisse der YouTube-Video-Suche und fügt diese den drei Listen hinzu.

Listing 13.3 Initialisierung der Ergebnislisten

```
26 video_comments_text = []
27 video_comments_likes = []
28 video_comments_date = []
29 video_id = []
30 video_comment_language = []
```

Da im nächsten Schritt nun die jeweiligen Video-Links aufgerufen werden sollen, um die YouTube-Kommentare zu extrahieren, werden fünf leere Listen definiert, in denen die Ergebnisse gespeichert werden.

Listing 13.4 YouTube-Kommentar Web Crawler

```
33 for i in video_links:
34     print('extracting video comments from link ', i)
35     try:
36         temp = yt.get_video_comments(i,max_results=50)
37         for i in temp:
38             video_comments_text.append(i['text'])
39             video_comments_likes.append(i['comment_like_count'])
40             video_comments_date.append(i['collection_date'])
41             video_id.append(i['video_id'])
42         try:
43             lang = detect(i['text'])
44             video_comment_language.append(lang)
45         except:
46             video_comment_language.append('no language detected')
47     except:
48         print('no comments for video', i)
```

Die For-Schleife durchläuft hierbei alle Einträge der Liste *video_links*, die die jeweiligen Links zu den YouTube-Videos beinhaltet. Für jedes Video werden dann maximal 50 Kommentare durch die Funktion *yt.get_video_comments* abgerufen und in der Liste *temp* gespeichert. Im nächsten Schritt wird jeder Kommentar in der Liste *temp* durchlaufen und der jeweilige Kommentar (*text*), die Anzahl an Likes (*comment_like_count*), Datum und Uhrzeit (*collection_date*) und die Video-ID (*video_id*) in den vorher erstellten Listen gespeichert. Im nächsten Schritt (ab Zeile 42) wird versucht, mithilfe der Python Library *langdetect* die Sprache des Kommentars zu ermitteln. Wenn dies gelingt, wird das Ergebnis der Liste *video_comment_language* hinzugefügt, anders nur der Wert *no language detected*. Falls keine Kommentare für das jeweilige Video vorhanden sind, wird ein Hinweis in der Konsole ausgegeben (Zeile 48) und die Iteration durch die Liste *video_links* fortgesetzt.

Im finalen Schritt sollen die extrahierten Daten noch in einem DataFrame gespeichert werden:

Listing 13.5 Speichern der Ergebnisse

```

51 results = pd.DataFrame(
52     {'video_id': video_id,
53      'comment_date': video_comments_date,
54      'video_comments_likes': video_comments_likes,
55      'video_comments_text': video_comments_text,
56      'language': video_comment_language})
57
58 if not os.path.isfile('/home/ubuntu/youtube_results.csv'):
59     print('no csv file found – creating new one')
60     results.to_csv('youtube_results.csv', header='column_names', index=False)
61 else:
62     print('csv file found – appending')
63     results.to_csv('/home/ubuntu/youtube_results.csv', mode='a', header=False, index=
        False)
64 print('the csv has been saved / extended')

```

Zuerst wird aus den verschiedenen Ergebnislisten ein gemeinsamer DataFrame erstellt (Zeile 51–56). Im nächsten Schritt muss die Speicherung der extrahierten Daten implementiert werden. Wichtig hierbei ist, dass der Web Crawler täglich gestartet wird und die Ergebnisse der CSV-Datei hinzugefügt werden. Deshalb wird im ersten Schritt (Zeile 58) geprüft, ob bereits eine CSV-Datei mit dem Namen *youtube_results.csv* existiert. Falls dies nicht der Fall ist, wird diese Datei erstellt und mit dem DataFrame *results* befüllt. Hierzu wird auch ein Header initialisiert, der die Spaltennamen des DataFrame aufweist. Falls die CSV-Datei schon existiert, soll der Inhalt des DataFrame *results* hinzugefügt werden, aber ohne eine erneute Nennung der Spaltennamen (*header=False*). Das Python-Skript wird nun als *youtube_crawler.py* gespeichert und muss im nächsten Schritt auf die EC2-Instanz hochgeladen werden.

Zugriff auf EC2 und Einbinden des Python Web Crawlers

Um den Web Crawler nun in die EC2-Instanz einzubinden, muss im ersten Schritt das Python-Skript hochgeladen und dann per Kommandozeile gestartet bzw. ein automatischer Start eingerichtet werden. Es gibt mehrere Möglichkeiten, auf die Dateien der EC2-Instanz zuzugreifen und neue Dateien hochzuladen: Die gängigsten Methoden hierbei sind entweder das Hochladen per Kommandozeile oder mithilfe eines FTP-Programms. Die zweite Methode wird in diesem Fallbeispiel genutzt und vorgestellt, wobei FileZilla⁶ als FTP-Client dient. FileZilla ist ein populäres, kostenloses und als Open Source verfügbares Tool, das zusätzlich noch ein einfaches und übersichtliches Interface aufweist. Nach der Installation von FileZilla wird in einem ersten Schritt ein neuer Server (die EC2-Instanz) über *Datei* → *Servermanager* angelegt. Hierbei wird zuerst als Protokoll *SFTP – SSH File Transfer Protocol* ausgewählt und die *Public DNS (IPv4)* aus dem EC2-Dashboard in das Feld *Server* eingetragen. Die Verbindungsart wird auf *Schlüsseldatei* gestellt und das heruntergeladene

⁶ Siehe: <https://filezilla-project.org>, zuletzt abgerufen am 1. September 2020.

Key Pair ausgewählt (wichtig ist die Dateiendung im Windows Explorer auf *.pem* zu stellen). Als letzter Schritt muss noch als Benutzername *ubuntu* eingetragen werden (der Standardbenutzername für Ubuntu-EC2-Instanzen). Danach kann mit einem Klick auf *Verbinden* die Verbindung zum Dateisystem der EC2-Instanz durchgeführt werden. Die Meldung *Unbekannter Server-Schlüssel* muss akzeptiert werden. Nach dem erfolgreichen Verbinden kann nun per Drag-and-Drop das Python-Skript (links – lokal) in das ausgewählte Verzeichnis (rechts – EC2) hochgeladen werden (Abb. 13.1 und 13.2).

In einem nächsten Schritt muss nun per Kommandozeile auf die EC2-Instanz zugegriffen werden. Hierbei werden noch benötigte Python Libraries auf der EC2-Instanz installiert, Sicherheitsupdates durchgeführt und ein sogenannter Cron-Daemon erstellt. Ein Cron-Daemon oder Cron-Job automatisiert wiederkehrende Aufgaben in einem UNIX/LINUX-Betriebssystem. Da eine tägliche Ausführung des Web Crawler gewünscht wird, ist dies notwendig, da ein manuelles Starten des Python-Skripts zeitaufwendig und fehleranfällig wäre. Um diese Schritte durchzuführen, wird erneut mit dem Webbrowser das *EC2-Dashboard* aufgerufen und mit der rechten Maustaste auf die aktive EC2-Instanz geklickt, um dann den Button *Verbinden* auszuwählen. Ein neues Fenster öffnet sich und wir wählen – die für die aktuellen Zwecke völlig ausreichende - *EC2-Instance-Verbindung* aus. Nun sollte sich ein schwarzes Konsolenfenster geöffnet haben (unsere Shell auf der EC2-Instanz), in die Befehle eingegeben werden können. Es bietet sich jetzt an, erst einmal wichtige verfügbare Sicherheits-Updates zu installieren und eventuell die EC2-Instanz nach der Installation neu zu starten. Dies wird durch die folgenden Eingaben erreicht:

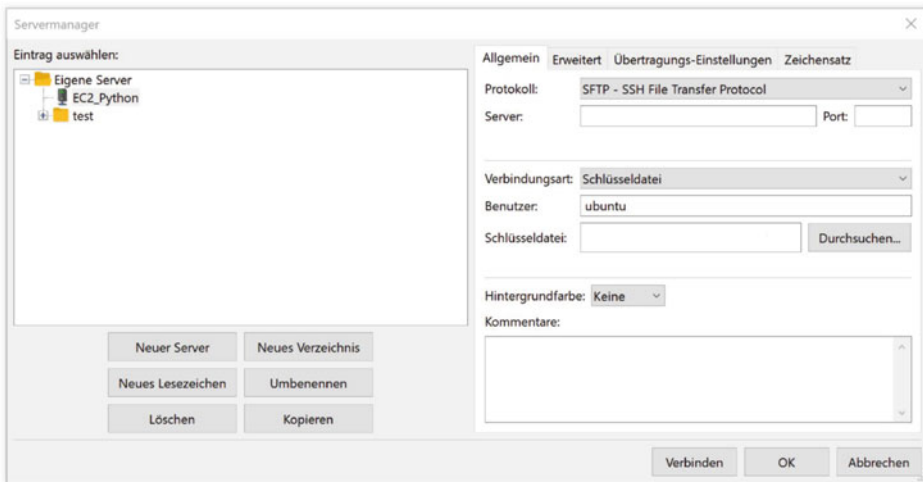


Abb. 13.1 Der FileZilla Server Manager in der Übersicht

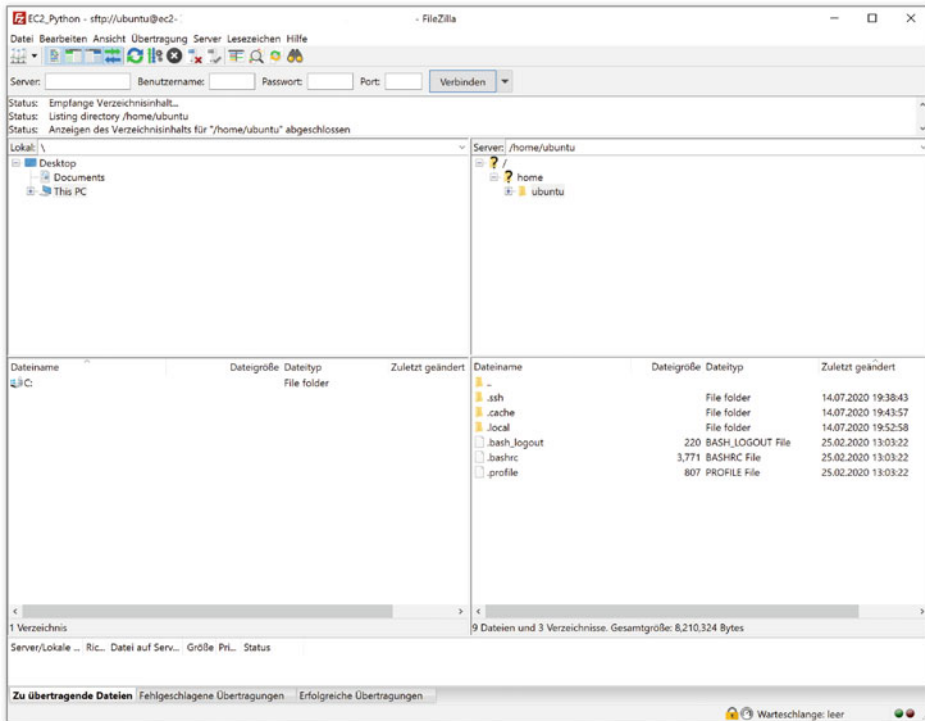


Abb. 13.2 Erfolgreicher Verbindungsaufbau zur EC2-Instanz

1. `sudo apt-get update`
2. `sudo apt full-upgrade` //dieser Befehl muss mit *Y* bestätigt werden. Daraufhin werden alle verfügbaren Software-Updates installiert
3. `sudo reboot` //dieser Befehl startet die EC2-Instanz neu

Nun muss ein paar Minuten gewartet werden, bis die EC2-Instanz wieder online und erreichbar ist. Anschließend werden die obigen Schritte zum Verbinden wiederholt und die Konsole geöffnet.

Sind alle Software-Updates installiert und der Server neu gestartet, muss noch sichergestellt werden, dass die im Python-Skript verwendeten Pakete installiert sind. Dies ist notwendig, da der Web Crawler auf die Funktionalitäten dieser Pakete aufbaut. Eine Installation erfolgt hierbei über den Python Package Installer mit den folgenden Befehlen, die in der Konsole eingegeben werden:

1. `sudo apt install python3-pip` //der Python Package Installer wird installiert (muss mit *Y* bestätigt werden)

2. `pip3 install --user youtube-data-api` //die Library wird als `--user` installiert, da ein virtueller Server verwendet wird
3. `pip3 install --user pandas`
4. `pip3 install --user langdetect`

Sind die Libraries erfolgreich installiert, muss nur noch das Python-Skript als täglicher Cron-Daemon registriert werden. Schließlich soll der Web Crawler selbstständig und automatisiert täglich gestartet werden und die YouTube-Kommentare extrahieren. Hierzu werden wieder die folgenden Befehle in der Konsole eingegeben:

1. `nano youtube_crawler.cron` //eine Datei `youtube_crawler.cron` wird erstellt und bekommt folgenden Inhalt (siehe nächste Zeile)
2. `0 13 * * * python3 /home/ubuntu/youtube_crawler.py` //an jedem Tag um 13 Uhr soll das Python-Skript gestartet werden
3. Die Tastenkombinationen `STRG+O` und dann die Eingabetaste und danach `STRG+X` drücken, um den Inhalt in der Datei zu speichern und den Texteditor zu schließen
4. `crontab youtube_crawler.cron` //die Verlinkung als Cron-Daemon wird hergestellt
5. `crontab -l`

Nach diesen Eingaben kann die Konsole wieder geschlossen werden und der Web Crawler sollte täglich um 13:00 Uhr Systemzeit der EC2-Instanz ausgeführt werden, die YouTube-Daten extrahieren und in der angegebenen CSV-Datei speichern. Wichtig hierbei ist noch, auf die Cron-Daemon-Syntax hinzuweisen (siehe Befehl 2.), die immer wie folgt ist: A B C D E Befehl. A, B, C, D und E stehen hier für den Ausführungszeitpunkt des Befehls. A beschreibt die Minute, in der der Befehl ausgeführt werden soll und kann Werte zwischen 0 und 59 aufweisen. B definiert die Stunde und kann Werte zwischen 0 und 23 annehmen. C definiert den Tag und ist als 0–31 festgelegt. D beschreibt den Ausführungsmonat, also 1–12. Zuletzt steht E für den Wochentag und kann Werte zwischen 0 und 7 annehmen, wobei 0 und 7 beide für Sonntag stehen. Ist der Buchstabe durch ein `*` ersetzt, so erfolgt der Befehl immer zu der jeweiligen Zeitkomponente. Für das obere Beispiel heißt dies: Der Befehl zum Aufruf des Web Crawlers wird an jedem Tag der Woche um 13:00 Uhr Systemzeit der EC2-Instanz ausgeführt.

13.3 Cloud Web Services: Explore the Data and More

Nach einigen Tagen kann dann die mit täglich neuen YouTube-Kommentaren befüllte CSV-Datei mit FileZilla heruntergeladen werden. Obwohl die Durchführung der Analysen auch ohne Probleme auf der EC2-Instanz möglich wären, rate ich bei diesem Fallbeispiel davon ab. In der kostenlosen Variante hat die EC2-Instanz einfach zu wenig RAM, CPU Power

und außerdem ist das Debugging von möglichen Fehlern deutlich erschwert. In den nächsten Schritten wird anhand der extrahierten Daten auf zwei relativ simple Analysen eingegangen, diese werden detailliert beschrieben und die Ergebnisse visualisiert.

Zuerst wird wir eine Sentiment-Analyse aller englischen YouTube-Kommentare vorgenommen. Hierzu wird die Python Library *vaderSentiment*, die im Kapitel *Social Media* detailliert beschrieben wurde, genutzt, um die YouTube-Kommentare, die das Wort *fake* beinhalten, mit denen ohne das Wort *fake* zu vergleichen.⁷ Im zweiten Schritt werden anhand dieser Trennung Word Clouds mit der Python Library *wordcloud* visualisiert und gegenübergestellt. Abschließend werden die Ergebnisse im nächsten Kapitel in einem S3-Bucket als Webseite visualisiert und veröffentlicht.

Listing 13.6 Benötigte Libraries

```

7 import pandas as pd
8 import os
9 import re
10 from wordcloud import WordCloud, STOPWORDS
11 from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
12 import matplotlib.pyplot as plt
13 plt.ioff() #requires plt.show()
14 analyzer = SentimentIntensityAnalyzer()

```

Neben den Standard-Python-Libraries wie *pandas*, *os* und *re* wird für die Visualisierung noch die Library *matplotlib* genutzt. Zeile 13 definiert zusätzlich, dass Plots nicht automatisch visualisiert werden sollen, während Zeile 14 den Vader-Sentiment-Analyzer initialisiert.

Im nächsten Abschnitt wird nun die von der EC2-Instanz exportierte CSV-Datei eingelesen. Zeile 32 spezifiziert die Anzahl an Spalten und Zeile 33 das Zeilen-End-Zeichen (`\n`). Zusätzlich werden nur als englisch identifizierte Kommentare im Datensatz vorgehalten (Zeile 35):

Listing 13.7 Import der CSV-Datei und Filtern nach englischen Kommentaren

```

30 #load csv file
31 youtube_df = pd.read_csv('youtube_results.csv',
32                          usecols=range(5),
33                          lineterminator='\n')
34 #keep only comments in english
35 youtube_df2 = youtube_df[youtube_df['language']=='en']
36 youtube_df2.info()

```

⁷ Natürlich ist diese Art der Teilung in zwei Gruppen sehr ungenau. Normalerweise würden Supervised-Learning- oder Topic-Modeling-Techniken genutzt werden, um die Kommentare von Verschwörungsmystikern zu kennzeichnen. Dies überstiege aber den Umfang dieses Fallbeispiels, das den Fokus auf AWS setzt.

Nachdem so der Datensatz gefiltert wurde, wird im nächsten Schritt die Vader Library zur Generierung der Sentiment-Werte verwendet. Zusätzlich wird das Datum-/Zeit-Format angepasst und transformiert, sodass die Daten in einem späteren Schritt auf Tageslevel aggregiert werden können. Hierzu werden neue Spalten im DataFrame erstellt und der in Zeile 14 initialisierte Analyzer der Bibliothek Vader zur Analyse des Kommentartexts genutzt:

Listing 13.8 Generierung der Sentiments und Transformation des Datum-/Zeit-Formats

```

39 #sentiment analysis on the whole data set
40 youtube_df2['compound'] = [analyzer.polarity_scores(x)['compound'] for x in youtube_df2['
    video_comments_text']]
41 youtube_df2['neg'] = [analyzer.polarity_scores(x)['neg'] for x in youtube_df2['
    video_comments_text']]
42 youtube_df2['neu'] = [analyzer.polarity_scores(x)['neu'] for x in youtube_df2['
    video_comments_text']]
43 youtube_df2['pos'] = [analyzer.polarity_scores(x)['pos'] for x in youtube_df2['
    video_comments_text']]
44 youtube_df2['date'] = pd.to_datetime(youtube_df2['comment_date'],format="%Y-%m-%d")
45 youtube_df2['date'] = youtube_df2['date'].dt.date

```

Im nächsten Schritt wird der Datensatz in zwei Gruppen aufgeteilt: Die erste Gruppe enthält nur Kommentare, die das Wort *fake* enthalten, während die zweite Gruppe nur Kommentare enthält, die das Wort *fake* nicht enthalten. Außerdem werden auf Tagesbasis die durchschnittlichen Sentiment-Werte gebildet:

Listing 13.9 Aufteilung in zwei Gruppen und Aggregation der Durchschnitts-Sentiment-Werte

```

52 #filter non fake posts
53 youtube_nofake_comm = youtube_df2[~youtube_df2['video_comments_text'].str.contains("
    fake")]
54 youtube_nofake_avg = youtube_nofake_comm.groupby(['date']).mean().rename(columns={'
    num': 'Averaged'}).reset_index()
55 #filter fake posts
56 youtube_fake_comm = youtube_df2[youtube_df2['video_comments_text'].str.contains("fake
    ")]
57 youtube_fake_avg = youtube_fake_comm.groupby(['date']).mean().rename(columns={'num': '
    Averaged'}).reset_index()

```

Nun werden die täglichen Durchschnittswerte der jeweiligen Sentiments der beiden Gruppen verglichen und visualisiert.

Die folgenden Code-Zeilen zeigen, wie die Bibliothek *matplotlib* (siehe für weitere Informationen Abschn. 10.3) genutzt wird, um einen neuen Graphen im Format (16:8) zu erstellen (Zeile 62). Die Beschriftungen der X- und Y-Achsen finden in den Zeilen 63 und 64 statt. Da sowohl die *non-fake*- als auch die *fake*-Graphen gezeichnet werden sollen, ist es notwendig, zweimal den *plot*-Befehl aufzurufen (Zeilen 65 und 66). Abschließend wird noch eine

horizontale Linie mit dem Wert 0 eingefügt, die ein neutrales Sentiment darstellt (Zeile 67). Der Graph wird dann, in Zeile 71, als PNG-Datei gespeichert:

Listing 13.10 Visualisierung der Resultate

```

60 #plot together
61 plt.close()
62 plt.figure(figsize=(16,8), dpi=150)
63 plt.xlabel('Days',fontsize=18)
64 plt.ylabel('Overall Sentiment',fontsize=18)
65 plt.plot(youtube_nofake_avg.index+1, youtube_nofake_avg['compound'], linewidth=4,
        color = 'blue')
66 plt.plot(youtube_fake_avg.index+1, youtube_fake_avg['compound'], linewidth=4, color
        ='red')
67 plt.axhline(y=0, color='black', linestyle=':')
68 plt.legend(loc='best')
69 plt.tight_layout()
70 #plt.show()
71 plt.savefig('Sentiment_comparison.png')

```

Das beispielhafte Ergebnis der Visualisierung kann in der nächsten Abb. 13.3 betrachtet werden. Ein Sentiment-Wert von 0 stellt hierbei einen ausgeglichenen YouTube-Kommentar dar, der weder positiv noch negativ konnotiert ist. Es ist klar zu sehen, dass die YouTube-Kommentare, die das Wort *fake* beinhalten, deutlich negativer sind und eine höhere Varianz aufweisen.

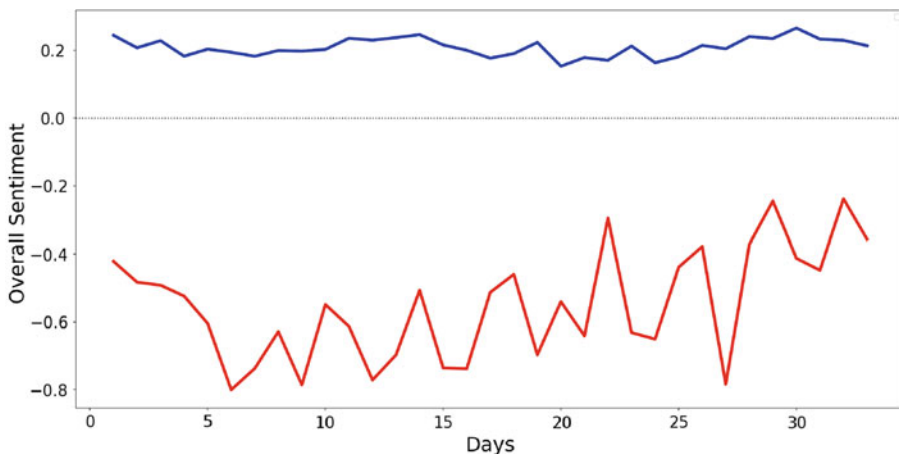


Abb. 13.3 Fake- (rot) vs. Non-fake-Kommentar-Sentiments (blau)

Im zweiten Schritt werden Word Clouds für die YouTube-Kommentare gebildet: Analog zum ersten Vorgehen werden hierfür wieder die beiden Gruppen (*fake* vs. *non-fake*) gebildet (Zeile 75 und 76) und alle Kommentartexte in die jeweilige Liste geschrieben. Weiterhin wird eine Liste mit dem jeweiligen Datum erstellt, das für die spätere automatische Generierung des Dateinamens verwendet wird (Zeile 79):

Listing 13.11 Verknüpfung der täglichen Kommentare durch Gruppierung nach Datum

```

74 #build word clouds of fake vs non fake
75 nofake_comm_for_wordcloud = youtube_nofake_comm.groupby(['date'])['
    video_comments_text'].apply(', '.join).reset_index()
76 fake_comm_for_wordcloud = youtube_fake_comm.groupby(['date'])['video_comments_text'].
    apply(', '.join).reset_index()
77 nofake_comm_per_date = nofake_comm_for_wordcloud['video_comments_text']
78 fake_comm_per_date = fake_comm_for_wordcloud['video_comments_text']
79 dates = nofake_comm_for_wordcloud['date'].dt.date

```

Anschließend wird eine For-Schleife genutzt, um für beide Gruppen jeweils eine eigene Word Cloud pro Datum zu erzeugen, beide nebeneinander in einem Graphen zu visualisieren und als PNG-Datei abzuspeichern (Zeile 104):

Listing 13.12 Visualisierung der beiden Gruppen in täglichen Word Clouds

```

82 for i in range(len(fake_comm_per_date)):
83     wordcloud_nofake = WordCloud(width = 800, height = 800,
84                                 background_color = 'white',
85                                 stopwords = stopwords,
86                                 min_font_size = 10).generate(nofake_comm_per_date[i])
87     wordcloud_fake = WordCloud(width = 800, height = 800,
88                               background_color = 'white',
89                               stopwords = stopwords,
90                               min_font_size = 10).generate(fake_comm_per_date[i])
91     plt.close()
92     plt.figure(figsize=(16,8))
93     plt.subplot(1, 2, 1)
94     plt.imshow(wordcloud_nofake)
95     plt.title('Comments that DO NOT contain "fake"')
96     plt.axis("off")
97
98     plt.subplot(1, 2, 2)
99     plt.imshow(wordcloud_fake)
100    plt.axis("off")
101    plt.title('Comments that do contain "fake"')
102    plt.tight_layout()
103    #plt.show()
104    plt.savefig("wordcloud_day_"+str(dates[i])+".png")

```

Abb. 13.4 Non-fake vs. fake word clouds



Als Ergebnis wird im Working Directory des Python-Skripts pro Tag eine PNG-Datei mit der Gegenüberstellung der Word Cloud gespeichert. Ein exemplarisches Beispiel ist hier als Abb. 13.4 eingebettet. Es ist klar ersichtlich, dass die YouTube-Kommentare der *Fake-Posts* bei Impfkritikern und Verschwörungsmystikern beliebte Wörter und Themen wie z. B. *nanobot*, *harmful* oder *poison* enthalten.

S3-Bucket und Konfiguration als Webseite

S3 ist ein Service von AWS, der das Speichern von jeglichen Daten in sogenannten Buckets ermöglicht, wobei S3 für *Simple Storage Service* steht. Der Endnutzer muss sich praktisch keine Gedanken über die dahinterliegende Hardware und Konfigurationsmöglichkeiten machen, sondern kann diesen skalierbaren Service einfach per Browser (wie in unserem Fall) oder per API mit Daten füllen, darauf zugreifen, diese zu modifizieren oder zu löschen. Zuerst wird erneut die *AWS Management Console* aufgerufen, um über die Suchfunktion S3 zu suchen. Im nächsten Schritt wird eine Übersicht über mögliche aktive S3-Buckets angezeigt und ein neues S3-Bucket durch den Klick auf den Button *Bucket erstellen* erstellt.

1. Name und Region: Hier muss ein eindeutiger und noch nicht genutzter Name für das S3-Bucket verwendet werden. Weiterhin muss eine Region spezifiziert werden, in der das S3-Bucket zur Verfügung gestellt werden soll. Ich empfehle die voreingestellte Region erst einmal nicht zu verändern, es gibt aber auch bei unserem Beispiel keine bedeutsamen Nachteile (außer vielleicht minimal höhere Zugriffszeiten).
2. Hier werden die Haken abgewählt und der Warnhinweis bestätigt. Normalerweise ist es nicht nötig, dass ein S3-Bucket öffentlich zugänglich ist, aber da die Auswertungsgrafiken unserer Analysen als Webseite öffentlich gehostet werden sollen, wird dies in unserem Fall benötigt.
3. Die Bucket-Versioning kann auf *Deaktivieren* eingestellt bleiben.
4. Optional kann das S3-Bucket noch Tags zugewiesen bekommen, dies ist in unserem Fall aber nicht nötig.
5. Eine Verschlüsselung muss nicht aktiviert werden.
6. Nach einer Überprüfung der Einstellungen wird die S3-Bucket-Erstellung durch einen Klick auf *Bucket erstellen* bestätigt.

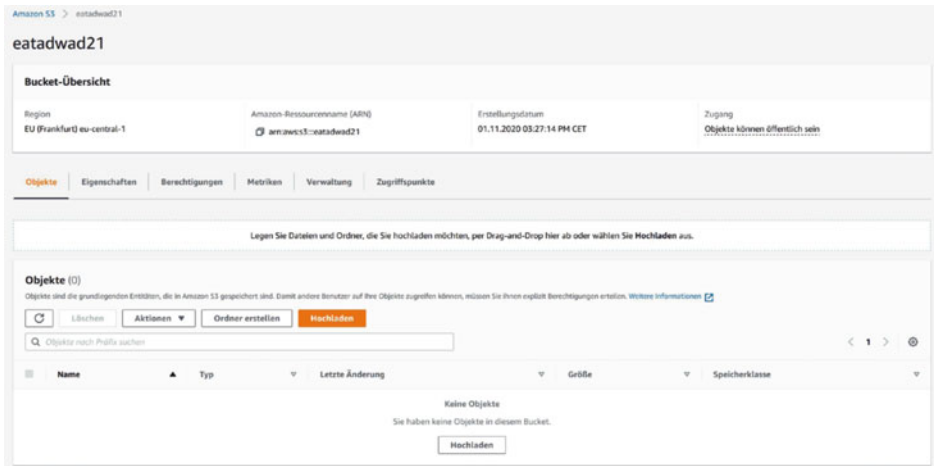


Abb. 13.5 Das neu erstellte S3-Bucket

Nun sollte das S3-Bucket bereits in der Liste auftauchen und kann angewählt werden. Damit das S3-Bucket auch von außen erreichbar ist, muss im Optionsmenü *Eigenschaften* noch die Einstellung *Hosten einer statischen Webseite* aktiviert werden. Dazu wird die Option *Diesen Bucket zum Hosten einer Website verwenden* aktiviert und unter *Indextdokument* die *index.html* eingetragen. Als *Fehlerdokument* kann entweder eine eigene HTML-Datei hochgeladen werden oder für unseren einfachen Testfall, auch die *index.html* eingetragen werden. Abschließend werden die Einstellungsänderungen noch durch einen Klick auf *Speichern* bestätigt. Der *Endpunkt* stellt die Web-Adresse der Webseite dar, deshalb sollte diese Adresse gespeichert oder als Bookmark gesetzt werden. Eine Übersicht über das neu erstellte S3-Bucket findet sich in Abb. 13.5.

Erstellung der Webseite und Upload ins S3-Bucket

Nachdem das S3-Bucket erfolgreich als Webserver konfiguriert wurde, müssen nun noch die vom Python-Analyseskript erstellten Bilder in eine HTML-Datei eingebunden werden. Zuerst wird mit einem einfachen Texteditor (z. B. Notepad++⁸) eine neue Textdatei erstellt und als HTML-Datei gespeichert.

⁸ Siehe: <https://notepad-plus-plus.org>, zuletzt abgerufen am 10. September 2020.

Listing 13.13 index.html

```

1 <!doctype html>
2 <html>
3   <head>
4     <title>Visualization of YouTube Data Regarding Covid</title>
5   </head>
6   <body>
7     <p>In a first step, we extracted YouTube comments from videos that included the
      term "covid".</p>
8     <p>Next, we filtered the comments into two sets: (1) comments that included
      the word "fake" and (2) comments that did not have the word "fake".</p>
9     <p><strong>Analysis of the sentiments:</strong></p>
10    
11    <p><strong>Analysis of the words used:</strong></p>
12    <p>Day1</p>
13    
14    <p>Day2</p>
15    
16    <p>Day3</p>
17    
18    <p>Day4</p>
19    
20    <p>Day5</p>
21    
22    <p>Day6</p>
23    
24    <p></p>
25  </body>
26 </html>

```

Es soll hier nicht im Detail auf die Erstellung, Syntax und Formatierung von HTML-Code eingegangen werden, aber eine HTML-Webseite ist generell in zwei Bereiche unterteilt: 1) den `<head></head>`-Bereich, der z. B. Metadaten wie den Titel der Webseite beinhaltet, und 2) den `<body></body>`-Bereich, in dem der Inhalt der Webseite eingetragen wird. Die Tags `<p></p>` stehen für die jeweiligen Textabsätze und beinhalten die Kurzbeschreibungen für die eingebetteten Bilder (``).

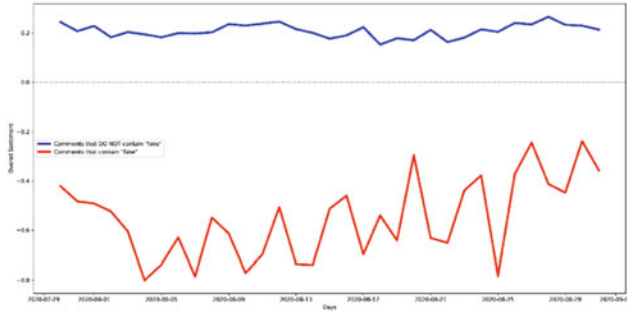
Im letzten Schritt muss nun noch die erstellte *index.html*-Datei und die durch das Python-Analyseskript erzeugten und gespeicherten Bilder in das S3-Bucket hochgeladen werden. Hierzu wird das S3-Bucket aufgerufen und der Button *Hochladen* ausgewählt. Die Dateien können dann per Drag-and-Drop in das Upload-Feld gezogen werden. Danach müssen die Checkboxes *Ich akzeptiere, dass vorhandene Objekte mit demselben Namen überschrieben werden* und – weiter unten – *Jeder (öffentlicher Zugriff – Lesen)* und die Warnmitteilung aktiviert werden. Anschließend werden die Dateien mit einem Klick auf *Hochladen* hochgeladen. Danach kann im Browser die Webseite über die vorher gespeicherte Adresse (Endpunkt) aufgerufen werden und ist weltweit verfügbar (Abb. 13.6).



In a first step, we extracted YouTube comments from videos that included the term "covid".

Next, we filtered the comments into two sets: (1) comments that included the word "fake" and (2) comments that did not have the word "fake".

Analysis of the sentiments:



Analysis of the words used:

Day1



Abb. 13.6 Erfolgreiche Visualisierung auf dem S3-Bucket

13.4 Cloud Web Services: Zusammenfassung

Dieses Kapitel hat demonstriert, wie es kostenlos und mit einfachen Mitteln möglich ist, einen in Python geschriebenen Web Crawler auf einer EC2-Instanz zu implementieren. Über einen Cron-Daemon wurde der Web Crawler täglich automatisiert gestartet, suchte nach dem vordefinierten Begriff *corona* auf YouTube, extrahierte die Video-Links und dazugehörige Kommentare und speicherte sie in einer CSV-Datei. Anschließend wurde die CSV-Datei von der EC2-Instanz heruntergeladen und analysiert. Hierbei wurden Sentiment- und Word-Cloud-Vergleiche von YouTube-Kommentaren, die das Wort *fake* enthielten, mit YouTube-Kommentaren, die das Wort *fake* nicht enthielten, durchgeführt. Die Ergebnisse wurden visualisiert und als PNG-Dateien gespeichert. Abschließend wurden die Bilder in eine HTML-Datei eingebettet und in ein S3-Bucket hochgeladen, wo sie öffentlich verfügbar gemacht wurden.

Teil V

Mitarbeiteranalyse



Anwendungsbeispiel: Mitarbeiterabwanderung 14

Kevin Bauer

Verfahren des maschinellen Lernens (ML) können genaue Vorhersagen über unbekannte Zustände liefern. Diese Vorhersagen wiederum informieren spezifische Entscheidungen, welche unter Unsicherheit getroffen werden müssen. Akkurate Vorhersagen sind oft der Schlüssel zu optimalen Entscheidungen. Nehmen wir zum Beispiel Anstellungsentscheidungen: Die Einstellung des besten Kandidaten für eine bestimmte Stelle unter einer Reihe von Bewerbern erfordert eine möglichst präzise Vorhersage über die zu erwartende Leistung der Kandidaten nach der Einstellung. In ähnlicher Weise erfordern Beförderungsentscheidungen eine möglichst akkurate Einschätzung darüber, wer ein effektiver Vorgesetzter auf der zu besetzenden Führungsposition sein wird. Durch äußerst kostengünstige und präzise Vorhersagen können ML-Applikationen dazu beitragen, diese Art von Entscheidungen zu verbessern und insgesamt die Effizienz zu erhöhen.

Aus einer technischen Perspektive besitzen ML-Algorithmen zwei unterschiedliche Reize. Zum einen erlauben diese Methoden komplexe Strukturen in großen Datensätzen (mehr oder weniger) automatisch zu entdecken, ohne dass explizit angegeben werden muss wonach die Maschine suchen soll. Zum anderen ermöglichen ML-Algorithmen die flexible Approximation funktionaler Zusammenhänge, d. h. Modelle zu erzeugen/schätzen, welche sich häufig gut verallgemeinern lassen. Dadurch sollen letztendlich neue Beobachtungen, die nicht dazu genutzt wurden um das Modell zu schätzen, möglichst präzise vorhergesagt werden.

Im Folgenden gehen wir ein Beispiel für ein potentielles ML-Projekt durch, das sich auf die Vorhersage der Arbeitskräfteabwanderung konzentriert. Unerwartete Arbeitskräfteabwanderung kann für Unternehmen problematisch sein, insbesondere dann wenn die Ausbildung neuer Arbeitnehmer mit erheblichen Kosten und anfänglichen Produktivitätsverlusten verbunden ist.

K. Bauer (✉)
Frankfurt am Main, Deutschland
E-mail: bauer@safe-frankfurt.de

14.1 Mitarbeiterabwanderung: Vorbereitung und Definition des Problems

Wir werden die folgenden für Python bereitgestellten Hauptbibliotheken verwenden. Diese Bibliotheken ermöglichen es uns Daten zu speichern, zu manipulieren und zu analysieren. Die Bibliotheken enthalten verschiedene wichtige Methoden, die es uns erlauben verborgene Muster zu erkennen und (hoffentlich) neue Erkenntnisse zu generieren. Die erkannten Muster sollen uns letztlich dazu in die Lage versetzen, bessere Entscheidungen treffen zu können. Wir werden die nachfolgend aufgelisteten Bibliotheken verwenden, welche bereits in diesem Buch an verschiedenen Stellen vorgestellt:

- Pandas
- Numpy
- Matplotlib
- Seaborn
- Scikit-Learn

Im Folgenden werfen wir einen kurzen Blick auf jede Einzelne.

Wir beginnen mit Pandas. Pandas ist unsere Bibliothek, welche uns Methoden und Datenstrukturen zur Verfügung stellt, damit wir die Rohdaten zunächst speichern und für spätere Analysen aufbereiten können. Pandas ist eine Python-Bibliothek, die Datenstrukturen und eine Vielzahl einfacher und zugleich mächtiger Methoden für anfängliche Analysen bereitstellt (z. B. die Berechnung simpler zusammenfassender Statistiken). Dazu werden wir zunächst den Datensatz, an dem wir arbeiten, in ein von Pandas bereitgestelltes DataFrame-Objekt laden – eine relationale Datenstruktur. Neben vielen anderen Möglichkeiten enthält Pandas integrierte Methoden zur Gruppierung, Filterung, Zusammenführung, Isolierung und Analyse von Daten. Vor diesem Hintergrund ist Pandas eine äußerst mächtige Bibliothek für verschiedenste Data Science Projekte. Das Codebeispiel 14.1 zeigt den Import der Bibliothek.

Listing 14.1 Import pandas as pd

```
1 import pandas as pd
```

Unser zweites Arbeitspferd wird die Bibliothek *Numerical Python* oder kurz NumPy sein. Wie der Name bereits andeutet, stellt diese Bibliothek wichtige Methoden zur Verfügung, die für verschiedene mathematische Operationen notwendig sind. NumPy ermöglicht uns die Verwendung spezifischer Array- und Matrix-Datentypen, die für Statistik, Ökonometrie und Datenanalyse unerlässlich sind. NumPy bietet äußerst effiziente Methoden zur Manipulation und Extraktion von Informationen aus diesen Strukturen, die in vielen ML-Projekten sehr nützlich sind. Das Codebeispiel 14.2 zeigt den Import der Bibliothek.

Listing 14.2 Import numpy as np

```
2 import numpy as np
```

Als nächstes benötigen wir Möglichkeiten zur visuellen Darstellung unserer Analyseergebnisse. Die visuelle Darstellung ist ein wichtiger Teil in jedem ML-Projekt, da sie es uns ermöglicht (i) Muster zu erkennen und (ii) anderen Personen diese Muster visuell zu illustrieren. Zur visuellen Darstellung werden wir Matplotlib verwenden – eine Bibliothek die speziell auf das Erzeugen von Graphen ausgelegt ist. Das Design der API ist der des Software-Pakets MATLAB sehr ähnlich. Das Codebeispiel 14.3 zeigt wie wir die Bibliothek importieren.

Listing 14.3 Import matplotlib

```
3 import matplotlib.pyplot as plt
```

Zusätzlich zu Matplotlib verwenden wir Seaborn – eine Sammlung von Methoden die im Allgemeinen als Ergänzung zu Matplotlib angesehen werden sollte, nicht als Substitut. Diese Bibliothek bietet uns eine größere Auswahl an Graphen, die uns bei der Veranschaulichung und Vermittlung von Informationen helfen. Das Codebeispiel 14.4 zeigt den Import der Bibliothek.

Listing 14.4 Import seaborn

```
4 import seaborn as sns
```

Schließlich benötigen wir eine Bibliothek, die uns verschiedene ML-Algorithmen zur Verfügung stellt. Hier nutzen wir die Bibliothek Scikit-Learn, welche eine ganze Reihe verschiedener ML-Algorithmen enthält, bspw. logistische Regressionen, Entscheidungsbäume, Support-Vektor-Maschinen und einfache Neuronale Netze. Scikit-Learn ist nicht nur sehr effizient bei der Implementierung der Algorithmen, sondern auch sehr einfach zu nutzen. Eine der ansprechendsten Eigenschaften ist die Tatsache, dass die APIs der Scikit-Learn-Methoden einer gemeinsamen, intuitiven Struktur folgen. Hinweis: Anstatt die gesamte Bibliothek zu importieren, die sehr umfangreich ist, importieren wir die spezifischen Methoden, die wir tatsächlich verwenden, beginnend mit den Folgenden. Wir werden später darauf eingehen, was sie tun, sobald wir sie ins Spiel bringen. Weitere Methoden importieren wir an späterer Stelle. Das Codebeispiel 14.5 zeigt den Import von Scikit-Learn in unser Skript. Im weiteren Verlauf dieses Kapitels wird Scikit-Learn als Sklearn referenziert.

Listing 14.5 Import Sklearn

```
5 from sklearn.utils.random import sample_without_replacement
6 from sklearn.utils import resample
7 from sklearn import preprocessing
8 from sklearn.preprocessing import LabelBinarizer
9 from sklearn.preprocessing import OneHotEncoder
10 from sklearn.model_selection import train_test_split
11 from sklearn.metrics import accuracy_score, confusion_matrix, recall_score,
    precision_score
```

Sehr gebräuchliche Lernalgorithmen sind Lineare und Polynomiale Regression, Logistische Regression, k-nächste Nachbarn, Support-Vektor-Maschinen, Entscheidungsbäume, Zufallswälder, Ensemble-Verfahren und Neuronale Netze. All diese verschiedenen Techniken haben ihre eigenen Vorzüge und Gefahren. Für unser Beispiel-ML-Projekt, das wir im Folgenden durchgehen, bedienen wir uns dem (i) k-nächste Nachbarn (instanzbasiertes Lernen) und (ii) Random Forests (instanzbasiert, Lernen nach der Ensemble-Methode).

Beide Algorithmen sind überwachte Lernmethoden (supervised learning). „Lernen“ bezieht sich auf die, mehr oder weniger, automatische Identifikation und Approximation funktionaler Zusammenhänge in großen Datensätzen ohne explizite Kodierung durch einen Menschen. „Überwacht“ spiegelt die Vorstellung wider, dass Labels (auch als abhängige Variablen oder Zielvariablen bezeichnet) als Lehrer dienen, die dem Algorithmus mitteilen, wann er durch die Verarbeitung der Features (unabhängiger Variablen) eine richtige oder falsche Ausgabe produziert hat. Mathematisch gesehen erzeugt jeder ML-Algorithmus des überwachten Lernens eine hoch flexible Funktionsapproximation, welche die Labels neuer Datenpunkte, die nicht in den Trainingsprozess involviert wurden, möglichst präzise vorherzusagen kann – akkurate Out-of-sample Vorhersagen. In diesem Sinne sind überwachte Algorithmen Optimierungsprobleme, die nach einer Hypothese suchen, welche die Muster und Beziehungen zwischen Variablen in den Trainingsdaten am besten erklärt. Die Zielfunktion dieser Optimierung wird häufig als Verlustfunktion (Loss-function) bezeichnet und quantifiziert wie falsch bzw. korrekt eine Hypothese ist. Die Nebenbedingungen (Hyperparameter-Optimierung, Regularisierung) sollen sicherstellen, dass sich die approximierte Funktion gut verallgemeinert.

Wir werden nun anhand eines Beispieldatensatzes zeigen, wie ML-Algorithmen entwickelt werden können, die eine Entscheidung unter Unsicherheit informieren, unterstützen und möglichst verbessern. Wir werden nacheinander die folgenden Schritte durchgehen:

- Auf-/Vorbereitung der Daten
- Sampling und Erzeugung von Trainings- und Testdaten
- Explorative Analysen auf Trainingsdaten
- Auswahl und Training eines Modells
- Testen und Optimieren des Modells

Unser Ziel besteht darin, die zukünftige Abwanderung einzelner Mitarbeiter innerhalb eines fiktiven Unternehmens so genau wie möglich vorherzusagen. Es sollte stets beachtet werden, dass die Vorhersage einer bestimmten Variable nie ein Selbstzweck ist. Stattdessen sollte ein zuvor definiertes Ziel damit verfolgt werden. ML-Algorithmen können daher den größten Mehrwert liefern, wenn sie eine Vorhersage erzeugen, die einen Entscheidungsprozess unterstützen. In unserem Fall könnte es beispielsweise sein, dass die Personalabteilung versucht, den Abgang von wichtigem Humankapital frühzeitig zu identifizieren und zu verhindern. Der von uns erstellte ML-Algorithmus soll zu diesem Ziel beitragen, indem er der Personalabteilung eine Vorhersage über die Wahrscheinlichkeit des Ausscheidens von

einzelnen Mitarbeitern erzeugt. Diese Vorhersagen können genutzt werden, um über die Einleitung bestimmter Gegenmaßnahmen zu entscheiden, welche die Abwanderung verhindern könnten. Wie dieses Beispiel illustriert, ist der ML-Algorithmus selbst nicht die Lösung des Problems, sondern kann gezielt als Werkzeug genutzt werden, dass einen Teil (nämlich eine Vorhersage) zur Lösung des Problems beisteuern kann. Wenn also über den Einsatz von ML-Applikationen in Unternehmen nachgedacht wird, sollte stets damit begonnen werden Lösungen für relevante Probleme zu identifizieren welche möglichst akkurate Vorhersagen benötigen.

Die meisten Algorithmen des maschinellen Lernens benötigen eine große Menge Daten, um richtig zu funktionieren. Einer der Hauptgründe dafür ist, dass kleine Datensätze tendenziell mehr Rauschen, z. B. in Form statistischer Ausreißer, erfassen. Statistisches Rauschen kann potenziell zu Problemen der Überanpassung (Overfitting) führen. Im Allgemeinen bedeutet Overfitting, dass identifizierte Muster nicht über den Trainingsdatensatz hinaus verallgemeinert werden können, weil der Algorithmus sich zu sehr auf die Erklärung von Mustern konzentriert hat, die durch Zufall nur in dem Trainingsdatensatz existieren. Häufig ist ein erheblicher Aufwand erforderlich, um Overfitting zu verhindern. Dazu werden Regularisierungstechniken verwendet, welche die Fähigkeit des Algorithmus zur Aufnahme von Rauschen einschränken. Selbst für sehr einfache Probleme benötigt man normalerweise Tausende von Beispielen. Für komplexe Probleme wie bspw. die Verarbeitung eines Bildes oder natürlicher Sprache werde oftmals Millionen von Beispielen benötigt. Hinweis: Es ist manchmal möglich, Teile bestehender Modelle wiederzuverwenden, z. B. kann ein Neuronales Netz, das für eine andere Klassifizierungsaufgabe trainiert wurde, als erste Instanziierung eines anderen Neuronalen Netzes verwendet werden. Dieser Prozess wird manchmal auch als Transfer-Lernen bezeichnet. Dabei können allerdings andere Probleme auftreten. Beispielsweise können neuronale Netze bei sequentiellen Lernaufgaben unter katastrophalem Vergessen leiden. In unserem Beispiel-Projekt werden wir auf diese Problematik allerdings nicht weiter eingehen.

14.2 Mitarbeiterabwanderung: Auf-/Vorbereitung der Daten

Für unser Beispiel-Projekt, dass sich, wie bereits erklärt, mit einem zentralen HR-Problem beschäftigt, verwenden wir einen öffentlich zugänglichen Datensatz von Kaggle.com¹. Dieser Datensatz enthält Informationen zu HR-Statistiken eines fiktiven, nicht-weiter spezifizierten Unternehmens. Um den Datensatz zu verwenden, muss dieser heruntergeladen und lokal abgespeichert werden (Hinweis: Dazu muss man sich bei Kaggle registrieren). Die Daten umfassen 1470 verschiedene Beobachtungen, – Zeilen – die jeweils 35 verschiedene Variablen – Spalten – enthalten.

¹ <https://www.kaggle.com/pavansubhasht/ibm-hr-analytics-attrition-dataset/data>), zuletzt abgerufen am 12. Februar 2022.

Wir beginnen mit dem Import der Daten in ein von Pandas bereitgestelltes DataFrame-Objekt. Unsere Daten liegen als CSV-Format vor, daher verwenden wir die `read_csv`-Methode von Pandas. Wenn die Datei im Excel-Format vorliegt, müssen wir die `read_excel`-Methode verwenden. Die API zum Laden einer CSV-Datei in ein Pandas DataFrame-Objekt ist sehr intuitiv und wir im Codebeispiel 14.6 aufgezeigt.

Listing 14.6 Import der Daten

```

11 data_hr = pd.read_csv('/Users/kev/Desktop/Data/hr_data.csv',      # Dateipfad
12                      sep=',',                                     # Trennzeichen dass in der Datei
                        verwendet wird
13                      header=0,                                    # Angabe der ersten Zeile die
                        eingelesen und Spaltennamen verwendet werden soll
14                      encoding='utf8',                            # zu verwendende Kodierung beim Lesen
                        /Schreiben des DataFrames
15                      na_values='n/a')                            # Interpretation fehlender Werte als
                        Datentyp NaN
  
```

Unsere Daten sind nun als ein DataFrame-Objekt eingelesen. Allgemein ist ein DataFrame eine zwei-dimensionale, relationale Datenstruktur. Jede einzelne Observation die sich in unserem Datensatz befindet, wird einer eigenen Zeile zugeordnet. Dementsprechend repräsentiert die y-Dimension die Observationen. Auf der x-Dimension hingegen sind die einzelnen Werte der Variablen abgebildet, die zu einer Observation gehören. In unserem Beispiel-Datensatz repräsentiert jede Zeile, eine bestimmte Person, die durch verschiedene Ausprägungen von Variablen, z. B. Age, Attrition, Department, ..., beschrieben ist. Wie zuvor erklärt, sind wir daran interessiert die Abwanderung (=Attrition) von Mitarbeitern vorherzusagen. Dementsprechend ist unsere Zielvariable die Variable Attrition. Alle anderen Variablen sind potentielle Features die dazu genutzt werden Attrition zu erklären. Ein erzeugtes ML-Modell ist letztlich dazu in der Lage auf Basis der gegebenen Features die zugehörige Abwanderung möglichst akkurat vorherzusagen.

Nun da wir unsere Datei geladen haben, können wir uns die Daten etwas genauer anschauen. Dadurch können wir uns anfänglich mit der Struktur der Daten vertraut machen. Pandas bietet uns verschiedene Methoden, welche uns schnell einen Überblick über unsere Datenstruktur verschaffen können. Die Methode `DataFrame.head(n=N)` erlaubt es uns die ersten N Zeilen des DataFrame anzusehen. Die Funktion ist äußerst nützlich, um schnell zu testen, ob die eingelesenen Werte den richtigen Datentypen besitzen. Hier schauen wir uns die ersten 3 Einträge an. Es kann jede beliebige ganze Zahl eingegeben werden. Wenn keine Zahl definiert ist, werden automatisch die ersten 5 Zeilen angezeigt. Die Umsetzung und die Ausgabe zeigt das Codebeispiel.

Listing 14.7 DataFrame ansicht

```

16 # Auswahl der ersten 3 Eintraege (Spalten) im Datensatz
17 # Anstelle von n=3 kann jede andere beliebige ganze Zahl eingegeben werden
18 data_hr.head(n=3)
  
```

Listing 14.8 Konsolenausgabe

1		Age	Attrition	BusinessTravel	DailyRate		Department	DistanceFromHome	Education	...
2	0	41	Yes	Travel_Rarely	1102		Sales	1	2	...
3	1	49	No	Travel_Frequently	279		Research & Development	8	1	...
4	2	37	Yes	Travel_Rarely	1373		Research & Development	2	2	...
5										
6		EducationField	EmployeeCount	...	YearsSinceLastPromotion		YearsWithCurrManager			
7	0	Life Sciences	1	...	0		5			
8	1	Life Sciences	1	...	1		7			
9	2	Other	1	...	0		0			
10										
11		[3 rows x 35 columns]								

Um die Datentypen der einzelnen Variablen zu betrachten, können wir die Funktion `DataFrame.dtypes` verwenden. Diese Funktion zeigt uns die Namen und zugehörigen Datentypen aller in dem `DataFrame` vorhandenen Variablen an. Die Datentypen werden später noch von großer Bedeutung sein, da die meisten ML-Algorithmen nur bestimmte Datentypen als Eingabewerte akzeptieren. Die Umsetzung zeigt das Codebeispiel 14.9 und die Konsolenausgabe 14.10.

Listing 14.9 DataFrame dtypes

```
19 # Ausgabe der im Datensatz enthaltenen Variablen und des zugehoerigen Datentyps
20 data_hr.dtypes
```

Listing 14.10 Konsolenausgabe

1	Age	int64
2	Attrition	object
3	BusinessTravel	object
4	DailyRate	int64
5	Department	object
6
7	WorkLifeBalance	int64
8	YearsAtCompany	int64
9	YearsInCurrentRole	int64
10	YearsSinceLastPromotion	int64
11	YearsWithCurrManager	int64
12	dtype: object	

Variablen in unserem Datensatz sind entweder ganzzahlige Werte (`int64`) oder Strings, welche als Objekt-Datentyp (`object`) codiert sind. Der Datentyp „object“ ist ein flexibles Konstrukt, welches jedes Python-Objekt enthalten kann.

Eine Liste der einzelnen Variablennamen (Spaltennamen) kann mithilfe der Funktion `DataFrame.columns` erzeugt werden. Diese Liste gibt uns einen schnellen Überblick über die vorhandenen Spalten Variablen und wie diese angesprochen werden können. Wie bei jeder einfachen Python Liste, können wir auch hier die Elemente direkt ansprechen. Hierzu zeigen die Beispiele 14.11 und 14.12 die entsprechenden Eingaben und Konsolenausgaben.

Listing 14.11 DataFrame columns

```
21 # Erzeugen einer Liste, welche alle in dem DataFrame enthaltenen Variablennamen
    beinhaltet
22 list(data_hr.columns)
```

Listing 14.12 Konsolenausgabe: Datentypen im DataFrame

```

1 'Age',
2 'Attrition',
3 'BusinessTravel',
4 ...
5 'YearsInCurrentRole',
6 'YearsSinceLastPromotion',
7 'YearsWithCurrManager'

```

DataFrame-Objekte bieten eine Vielzahl an Möglichkeiten bestimmte Zeilen, Spalten, und Werte der eingelesenen Daten einzeln anzusprechen. Zum einen ist es möglich, einzelne Spalten über deren entsprechende Variablennamen anzusprechen. Dies erfolgt ähnlich zu der Ansprache von Elementen in Listen über eine Slicing-Operation, die von uns erfordert, dass wir den Namen der Spalte(n) die wir uns anschauen möchten kennen. Die Variablennamen werden als Liste übergeben. Analog zu anderen Python-Objekten können einzelne Teile des DataFrame, die durch Slicing ausgewählt sind, einer eigenen Variable übergeben und somit immer wieder angesprochen werden. Dabei erzeugt Python eine Kopie des ausgewählten Teils des DataFrame, sodass sich Änderungen nicht auf das Original auswirken. Ein Beispiel findet sich im folgenden Codebeispiel [14.13](#).

Listing 14.13 Teil des DataFrame

```

24 # Auswahl eines DataFrames der nur eine Spalte mit Variablennamen Attrition besitzt.
25 data_hr[['Attrition']]
26
27 # Auswahl eines DataFrames der zwei Spalten mit den Variablennamen Attrition und
    Education besitzt.
28 data_hr[['Age', 'Education']]
29
30 # Zuordnen des ausgewählten Teils des DataFrames zu einer expliziten Variable
31 # Operationen auf data_hr_slice wirken sich nicht auf die Werte in data_hr aus
32 data_hr_slice = data_hr[['Age', 'Education']]

```

Zu beachten: Wenn eine einzelne Variable als String und nicht als Liste übergeben wird, so ist das erzeugte Objekt kein DataFrame mehr, sondern ein Series-Objekt was eine Art Spaltenvektor ist. Ein Beispiel findet sich im folgenden Codebeispiel [14.14](#).

Listing 14.14 Teil des DataFrame

```

25 # Auswahl einer Series/ eines Spaltenvektors
26 # Beachte: Kein DataFrame Objekt!
27 data_hr['Age']

```

Pandas bietet weitere Slicing-Methoden die uns das Auswählen und Ansprechen bestimmter Bereiche des DataFrame ermöglichen. Häufig werden Teile des DataFrame durch den DataFrame.iloc[row, column] Slicing-Operator angewählt. Der DataFrame.iloc Operator erlaubt die Ansprache von Teilbereichen ausschließlich durch die Übergabe von ganzen Zahlen, Boolean-Listen, oder Objekten die ausschließlich ganze Zahlen beinhalten. Dieser Operator erlaubt es allerdings nicht, Spalten durch deren zugehörigen Variablennamen anzusprechen.

Beachte: Für den Fall dass ein angeforderter Indexierer außerhalb der Grenzen des DataFrame liegt, wird ein IndexError ausgelöst. Dies ist nicht der Fall wenn ein Slice-Indexierer

verwendet wird, da diese Operationen eine Indexierung außerhalb der Grenzen erlauben. Das Codebeispiel 14.15 zeigt verschiedene Anwendungsbeispiele.

Listing 14.15 Slice-Indexierer

```

39 # Auswahl der Zeile 0 des DataFrames
40 data_hr.iloc[0]
41
42 # Auswahl der Spalte 0 des DataFrames
43 # Beachte: Der angezeigte Bereich ein Series Objekt kein DataFrame
44 data_hr.iloc[:, 0]
45
46 # Auswahl der Zeilen 0, 3 und 7 des DataFrames
47 data_hr.iloc[[0,3,7]]
48
49 # Auswahl aller Zeilen von 0 bis 5 (=6-1) DataFrames
50 data_hr.iloc[0:6]
51
52 # Auswahl aller Zeilen und der einzelnen Spalten 1, 2 und 5
53 data_hr.iloc[:, [1,2,5]]
54
55 # Auswahl aller Zeilen von 0 bis 1 mit den Spalten 0 bis 40
56 # Beachte: Es existieren nur 35 Spalten. Dennoch wird kein IndexError ausgelöst,
57 # sondern nur die maximal mögliche Anzahl angezeigt
58 data_hr.iloc[0:2, 0:40]
```

Eine weitere wichtige Zugriffsmöglichkeit ist das Boolesche-Indexieren über einen sogenannten Boolean-Vektor. Dieser Vektor besteht ausschließlich aus True und False Werten, die letztlich angeben, welche der Observationen in einem DataFrame ausgewählt werden sollen (True wird ausgewählt, False nicht). Die Werte des Boolean-Vektors werden durch die Prüfung von einer einzelnen oder mehrerer verbundener Bedingungen zugeordnet. Ist eine Bedingung erfüllt, so wird der Wert True zugeordnet – andernfalls der False Wert.

Im Folgenden Codebeispiel Codebeispiel 14.16 sind zwei Beispiele bezogen auf unseren Datensatz aufgeführt. Bei dem ersten Beispiel wählen wir alle Observationen aus, für die der Wert der Variable Age 40 oder größer ist. In dem zweiten Beispiel kombinieren wir zwei Bedingungen und wählen gleichzeitig nur bestimmte Spalten aus: Wir wählen für alle Observationen, für die der Wert der Variable Age 40 oder größer UND der Wert der Variable DistanceFromHome kleiner 7 ist, die Spalten Age, Attrition und Education aus.

Listing 14.16 Slice-Indexierer mit Bedingung

```

52 # Beispiel 1
53 # Auswahl aller Beobachtungen/ Zeilen fuer die der Wert der Spalte Alter
54 # groesser-gleich 40 ist
55 data_hr[
56     (data_hr['Age'] >= 40) # Der Boolean-Vektor testet die Spalte Age und ordnet jeder Zeile den Wert True zu
57     ] # fuer welche die Bedingung in der entsprechenden Spalte erfuehlt ist
58
59 # Beispiel 2
60 # Auswahl aller Beobachtungen/ Zeilen fuer die die Werte der Spalten Age
```

```

61 # und DistanceFromHome resueektive groesser-gleich 40 und kleiner 7 sind.
62 # Dabei werden nur die Spalten Age, Attrition und Education ausgegeben.
63 data_hr[
64     (data_hr['Age'] >= 40) &                                # Wir verbinden die konjunktiven Bedingungen ueber den &
65     (data_hr['DistanceFromHome'] < 7)                       # Nur wenn beide Bedingungen GLEICHZEITIG erfuehlt sind,
66     ][['Age', 'Attrition', 'Education']]                   # wird der Wert True zugeordnet
67                                                         # Wir kombinieren den Boolean-Vektor den wir fuer die Auswahl
68                                                         # der Zeilen verwenden, mit einer Liste zur Auswahl von
69                                                         # Spalten / Variablen

```

Die Konsolenausgabe 14.17 zeigt das Ergebnis für das Beispiel 2.

Listing 14.17 Konsolenausgabe

```

1      Age  Attrition  Education
2      0      41      Yes         2
3      6      59      No         3
4     18      53      No         4
5      ...      ...      ...      ...
6    1455      40      No         4
7    1457      40      No         4
8    1468      49      No         3
9
10 [251 rows x 3 columns]

```

Pandas erlaubt es allerdings nicht nur auf Daten innerhalb eines DataFrame zuzugreifen, sondern auch neue Daten hinzuzufügen, bestehende Daten zu verändern und zu löschen. Die Pandas API bietet äußerst intuitive und umgängliche Möglichkeiten dies zu tun. Da wir im Folgenden öfters auf diese Möglichkeiten zurückgreifen werden, wird an dieser Stelle kurz auf die Zuweisung neuer Variablen eingegangen. Das Codebeispiel 14.18 zeigt, wie wir verschiedene neue Variablen zu unserem DataFrame hinzufügen und wie wir Variablen wieder löschen können. Diese Art der Bearbeitung des ursprünglichen Datensatzes wird häufig auch als Feature-Engineering bezeichnet.

Listing 14.18 DataFrame weitere Datenmanipulationen

```

60 # Fuegt dem DataFrame neue Variable mit dem Namen 'new_variable' hinzu.
61 # In jeder Spalte nimmt diese Vairable den Wert 5 an.
62 data_hr['new_variable'] = 5
63
64 # Fuegt dem DataFrame neue Variable mit dem Namen
65 # 'new_variable' hinzu. Der Wert dieser Variable
66 # in einer Spalte entspricht dem Produkt der
67 # Werte der Variablen Age und Education aus
68 # der entsprechenden Spalte.
69 data_hr['Age_Education'] = data_hr['Age']*data_hr['Education']
70
71 # Fuegt dem DataFrame neue Variable mit dem Namen 'older_than_30' hinzu.
72 # Die Variable nimmt den Wert 1 an, wenn der Wert Age der Observation
73 # groesser als 30 ist, sonst 0.
74 data_hr['older_than_30'] = np.where(data_hr['Age'] > 30,           # Bedingung die auf True / Falase getestet wird
75                                   1,                             # Zugeordneter Wert wenn die Bedingung True ist  0)
76                                   # Zugeordneter Wert wenn die Bedingung False ist
77
78
79 # Fuegt dem DataFrame neue Variable mit dem Namen 'age_bins' hinzu.
80 # Der Variable Variable nimmt den Wert 1 an, wenn Age groesser als 0 und kleiner-gleich als 10 ist;
81 # den Wert 2 wenn Age groesser als 10 und kleiner-gleich 20 ist, ...
82 data_hr['age_bins'] = pd.cut(data_hr['Age'],
83                             bins=[0, 10, 20, 30, 40, np.inf],
84                             labels=[1, 2, 3, 4, 5])

```



```
83 # Loeschen von Variablen
84 # Wir koennen nicht relevante Variablen / Spalten ueber die df.drop Methode loeschen.
85 # Dazu muessen wir dem Befehl die entsprechenden Variable(n) uebergeben.
86 # Der der Variablen 'new_variable', 'Age_Education' und 'older_than_30'
87 data_hr.drop(
88     ['new_variable', 'Age_Education', 'older_than_30'], # uebergebene Variablen
89     axis = 1, # Axis=1 definiert, dass wir Spalten loschen moechten.
90     inplace = True # inplace=True stellt sicher, dass die Aenderungen in dem eigentlichen
                    # DataFrame
91 ) # Objekt uebernommen werden, da Pandas ansonsten eine temporaere Kopie anlegt
92 # dessen Aenderungen sich nicht auf das Original auswirken.
```

Nun da wir einen ersten Einblick in die Struktur und den Zugriff auf DataFrame-Objekte, erhalten haben, fangen wir damit an unseren Datensatz für die Anwendung von ML-Algorithmen vorzubereiten. Dabei wenden wir nicht nur die bisherigen Techniken an, sondern lernen weitere Eigenschaften der Pandas API kennen. Die Vorbereitung der Daten für ML Analysen nimmt häufig sehr viel Zeit in Anspruch und muss äußerst gewissenhaft durchgeführt werden. Die Auf- und Vorbereitung von Daten umfasst das Bereinigen des Datensatzes, die Definition wie mit fehlenden Werten umgegangen werden soll, die Umwandlung von Datentypen, die Erzeugung neuer Variablen aus vorhandenen Variablen, die Standardisierung von Daten und Reskalierung von Daten und vielem mehr. Glücklicherweise ist unser Datensatz bereits in einem sehr guten Zustand, sodass wir uns viele Schritte sparen können und schneller zu der Anwendung von ML-Algorithmen übergehen können.

Zunächst befassen wir uns mit der Identifikation fehlender Werte in unserem Datensatz. Fehlende Spaltenwerte einzelner Observationen, können sich als problematisch erweisen, wenn viele solcher einzelner Lücken im Datensatz existieren. Nutzen wir eine Spalte als Inputfeature für ein ML-Modell, so können wir nur von Observationen Gebrauch machen, die deren entsprechende Spaltenwerte nicht fehlen. Um zu prüfen, ob unser Datensatz fehlende Werte – in Pandas als NaN (Not a Number) Werte bezeichnet – beinhaltet, können wir die `df.isnull()` oder die `df.isna()` Methode anwenden. Beide Methoden geben uns ein DataFrame-Objekt, welches nur aus True und False Werten besteht. Ist ein Wert True, so ist das entsprechende Element in dem DataFrame auf dem die Methode angewendet wird leer. Das Codebeispiel 14.19 zeigt das Vorgehen.

Listing 14.19 DataFrame auf null nan Werte überprüfen

```
74 # Testen auf fehlende Werte
75 data_hr.isnull()
76 data_hr.isna()
```

Das Ergebnis des Codebeispiels 14.19 zeigt die nachfolgende Konsolenausgabe 14.20.

Listing 14.20 Konsolenausgabe

```
1      Age  Attrition  BusinessTravel  ...  YearsSinceLastPromotion  YearsWithCurrManager
2  0      False      False           False  ...                False                False
3  1      False      False           False  ...                False                False
4  2      False      False           False  ...                False                False
5  ...      ...      ...           ...  ...                ...                ...
6  1464  False      False           False  ...                False                False
7  1465  False      False           False  ...                False                False
8  1466  False      False           False  ...                False                False
9
10 [1470 rows x 35 columns]
```

Wie wir sehen können, existieren in unserem Datensatz keine fehlenden Werte. Dies reduziert die benötigte Vorarbeit enorm, da wir nicht definieren und einarbeiten müssen, wie wir mit diesen Fällen umgehen. Für den Fall das Spaltenwerte fehlen, können grundsätzlich verschiedene Techniken verwendet werden:

- Füllen der werte durch Mittelwerte/Medianwerte
- Löschen der Spalte in der sich der/die fehlenden Werte befinden
- Löschen der Zeilen in der sich der/die fehlenden Werte befinden
- Fehlende Werte als eigene, informationshaltige Werte verwenden (z. B. Wert -1 zuordnen)

Welche dieser Techniken bzw. Kombination von Techniken verwendet werden sollte, hängt stark von dem Datensatz und der unterliegenden Informationsstruktur ab. Es ist wichtig zu beachten, dass immer versucht werden sollte den Informationsverlust zu minimieren. Das heißt, dass wenn immer es möglich ist und Sinn macht das Löschen von Zeilen Spalten zu umgehen, sollte dies auch getan werden. Das Codebeispiel 14.21 zeigt die Anwendung einiger Möglichkeiten der Datenbereinigung, bezogen auf ein künstlich erzeugtes DataFrame.

Listing 14.21 Optionen zur Handhabung fehlender Werte

```
77
78 # Identifikation fehlender Werte
79 df_example.isna()
80
81 # Option 1
82 df_example.dropna(axis = 0)
83
84 # Option 2
85 df_example.dropna(axis = 1)
86
87 # Option 3
88 df_example.fillna('No Information')
```

Als nächstes befassen wir uns mit der Umwandlung von Datentypen. Die meisten ML Methoden, aber nicht alle (z.B. Entscheidungsbäume, Random Forests), können ausschließlich numerische Variablen verarbeiten. Daher müssen wir alle nicht-numerischen Variablen in eine numerische Form umwandeln, bevor wir die ML-Algorithmen anwenden können. Es gibt mehrere Möglichkeiten dies zu tun.

Hier verwenden wir den OneHotEncoder den Numpy uns zur Verfügung stellt. Diese Methode kann kategoriale Variablen unabhängig von der Anzahl ihrer möglichen Ausprägungen in numerische Datentypen umwandeln. Bei dem One-Hot-Encoding wird für jeden Wert den eine Variable annehmen kann eine neue binäre Variable hinzugefügt. Diese sogenannten Dummy-Variablen repräsentieren jeweils eine der verschiedenen Ausprägungen, welche die ursprüngliche Variable annehmen kann. Eine Dummy-Variablen nimmt den Wert 1 an, wenn der Wert der ursprünglichen Variable gleich der Ausprägung ist, welche diese

Dummy-Variable repräsentiert. Ansonsten wird der Dummy-Variable der Wert 0 zugeordnet. Nehmen wir zum Beispiel an, dass die fiktive Variable 'Frucht' die Ausprägungen 'Apfel' oder 'Birne' annehmen kann. Wird die Variable Frucht durch One-Hot-Encoding umgewandelt, so entstehen zwei neue Dummy-Variablen 'DummyApfel' und 'DummyBirne'. In einer Zeile in der die Variable Frucht den Wert 'Apfel' besitzt, ist der Wert von 'DummyApfel' gleich 1 und der Wert von 'DummyBirne' gleich 0. Diese Art der Codierung ist vor allem dann sinnvoll, wenn die Werte der ursprünglichen, zu kodierenden Variable keine ordinale Beziehung aufweisen, d. h., die Unterschiede zwischen den Ausprägungen nicht metrisch sinnvoll interpretiert werden können.

Wir sind in der Lage, diese Art der Codierung durchzuführen, indem wir die Klasse `LabelBinarizer` von `Sklearn` instanziiieren und verwenden. Bei dieser Art der Transformation werden die Ausprägungen Kategorien der Variable zunächst alphabetisch aufsteigend sortiert und dann in die zuvor beschriebenen Dummy-Variablen transformiert. Um zu sehen, welche und wie viele Kategorien codiert werden, können wir zunächst die `Pandas value_counts()`-Methode verwenden. Das Codebeispiel 14.22 zeigt die Verwendung.

Listing 14.22 Verwendung von `value_counts`

```

153 # Die methode gibt uns die verschiedenen Werte welche die Variable
154 # annimmt aus. Zusaetzlich erhalten wir Informationen zu der Anzahl
155 # der Observationen welche die Werte annehmen.
156 data_hr['BusinessTravel'].value_counts()
```

Das Ergebnis zeigt die Konsolenausgaben 14.23 der Anwendung des Codebeispiels 14.22.

Listing 14.23 Konsolenausgabe

```

1 Travel_Rarely      1043
2 Travel_Frequently   277
3 Non-Travel         150
4 Name: BusinessTravel, dtype: int64
```

In dem Beispiel können wir sehen, dass die Variable 'BusinessTravel' drei verschiedene Werte annehmen kann. Zusätzlich zu den möglichen Ausprägungen zeigt uns die `value_counts()`-Methode die Anzahl der Observationen in unserem Datensatz an, welche die entsprechende Ausprägung annehmen. Anhand dieser Ausgabe wissen wir nun, dass eine One-Hot-Codierung der Variable `BusinessTravel` drei unterschiedliche Dummy-Variablen erzeugt. Dazu instanziiieren wir als nächstes die One-Hot-Encoding Klasse. Diese stellt uns die Methoden zur Verfügung, welche es uns erlaubt die entsprechende Umwandlung nicht-numerischer Variablen durchzuführen. Zunächst wird die Umwandlung exemplarisch anhand der Variable 'BusinessTravel' aufgezeigt. Anschließend definieren wir uns eine Funktion, welche die Umwandlung automatisiert für alle nicht-numerischen Variablen durchführt.

Die Vorgehensweise erfolgt in drei Schritten: (i) Wir verwenden die `fit_transform()` Methode um für nicht-numerische Variablen Dummy-Variablen zu erzeugen. Die Dummy-Variablen werden dabei anfänglich als Matrix-Objekt ausgegeben. (ii) Das Matrix-Objekt wandeln wir anschließend in ein `DataFrame`-Objekte um. (iii) Abschließend fügen wir

das DataFrame an unser ursprüngliches DataFrame an und löschen die originalen, nicht-numerischen Variablen.

Das Codebeispiel 14.24 zeigt die Instanziierung der One-Hot-Encoding Klasse und die Anwendung der `fit_transform()` Methode im ersten Schritt.

Listing 14.24 Verwendung des one-hot-encoders

```

155 # Instanziierung der Klasse
156 one_hot_encoder = OneHotEncoder()
157
158 # Schritt 1:
159 # Anwendung der fit_transform() Methode
160 # Es wird eine duennbesetzte Matrix erzeugt, die nur aus 0en und 1en besteht
161 temp_object = one_hot_encoder.fit_transform(data_hr[['BusinessTravel']])

```

Die erzeugte Matrix wurde dem `temp_object` zugeordnet. Neben der Matrix erzeugt der Encoder eine sortierte Liste, welche die Namen der Variablenwerte enthält, die in binäre Variablen umgewandelt wurden. Diese verwenden wir, um den neuen Dummy-Variablen entsprechende Namen zuzuordnen. Die Liste ist als Objekt `'categories'` aufrufbar. Über eine Schleife können wir auf die Elemente der Liste zugreifen und neue, verständliche Variablennamen für die Dummies erzeugen. Diese neuen Namen speichern wir in einer Liste. Das Codebeispiel 14.25 zeigt wie wir dies tun.

Listing 14.25 Schleife für Dummy Variablen

```

169 # Gibt eine Liste mit den Werten der Variable aus, welche in Dummy-Variablen
    umgewandelt wurden
170 # Wie zuvor angesprochen, wurden die Werte zunaechst alphabetisch geordnet.
171 one_hot_encoder.categories_
172
173 # Erzeugung der Namen fuer die umgewandelten Variablen
174 new_names = [] # Eine leere Liste in die wir die
    neuen Namen einfuegen
175 for name in one_hot_encoder.categories_: # Iteration ueber die Elemente der
    Kategorien
176     temp_name = 'BusinessTravel' + '_' + name # Neuer Name
177     new_names.append(temp_name) # Hinzufuegen zur Liste

```

Im nächsten Schritt nutzen wir die erzeugte Matrix `temp_object` und die Liste der Variablennamen `new_names`, um ein DataFrame-Objekt zu erzeugen, dass aus die Dummy-Variablen beinhaltet. Diese Umwandlung der Matrix in das DataFrame-Objekt ist im Codebeispiel 14.26 dargestellt. Das resultierende DataFrame ist in der Konsolenausgabe 14.27 dargestellt.

Listing 14.26 Matrix to DataFrame

```

177 # Schritt 2:
178 # Erzeugung eines DataFrame Objekts,
179 # dass die umgewandelten Variablen beinhaltet
180 temp_df = pd.DataFrame(temp_object.toarray(),           # Die Matrix wird umgewandelt
                        und erzeugt ein DataFrame
181                        columns = new_names              # Dem DataFrame werden die
                                                         neuen Namen zugeordnet
182                        )
183
184 # Ausgabe des neu erzeugten DataFrames
185 temp_df

```

Listing 14.27 Konsolenausgabe

	BusinessTravel_Non-Travel	BusinessTravel_Travel_Frequently	BusinessTravel_Travel_Rarely
0	0.0	0.0	1.0
1	0.0	1.0	0.0
2	0.0	0.0	1.0
...
1467	0.0	0.0	1.0
1468	0.0	1.0	0.0
1469	0.0	0.0	1.0

Das temporäre DataFrame können wir nun in einem letzten Schritt an unser ursprüngliches DataFrame-Objekt anhängen (konkatenerieren) und die als Dummy-Variablen Codierte kategoriale Variable aus dem DataFrame löschen. Das Codebeispiel 14.28 zeigt diesen letzten Schritt.

Listing 14.28 Matrix to DataFrame

```

177 # Schritt 3:
178 # Konkatenation des DataFrame Objekts mit dem
179 # ursprünglichen DataFrame ueber die DataFrame.merge() Methode
180 data_hr = data_hr.merge(temp_df,                       # Das anzufuegende DataFrame
                        left_index=True,                 # Die beiden DataFrames sollen
                                                         auf Basis ihrer Indizes
181                        right_index=True)               # konkateniert werden, damit die
                                                         entsprechenden Observationen
                                                         # zusammengefuegt werden.
182
183
184
185 # Loeschen der nun umgewandelten und als Dummies eingefuegten Variable
186 data_hr = data_hr.drop('BusinessTravel', axis = 1)
187
188
189 # Alternativ koennen wir die DataFrames auch ueber die concat methode konkatenieren
190 data_hr = pd.concat([data_hr, temp_df], axis=1)

```

Um den Prozess der Umwandlung der Daten zu automatisieren, und die zuvor aufgezeigten Schritte nicht einzeln für jede identifizierte nicht-numerische Variable durchführen zu müssen, definieren wir eine Funktion. Diese Funktion akzeptiert ein DataFrame-Objekt als Eingabe, transformiert alle enthaltenen nicht-numerischen Variablen durch One-Hot-

Encoding.² Im Zuge der Umwandlung entfernt die Funktion alle nicht-numerischen Variablen die keine Variation aufweisen und somit keine verwertbaren Informationen beinhalten. Dadurch wird unser Datensatz übersichtlicher. Gleichzeitig vereinfachen wir die Anwendung von Modellen, bei denen wir explizit entscheiden müssen, welche Features das Modell nutzen soll (z. B. logistische Regression), da wir die Anzahl der möglichen Features auf die informationshaltigen Variablen reduzieren. Als Ausgabe liefert die Funktion ein DataFrame-Objekt, welches nur numerische Variablen enthält. Das Codebeispiel 14.29 zeigt die Funktion.

Listing 14.29 Implementierung einer Methode zum automatisierten One-Hot-Encoding

```

216 # Alternativ koennen wir die DataFrames auch ueber die concat methode konkatenieren
217 data_hr = pd.concat([data_hr, temp_df], axis=1)
218
219 # Funktion zur automatisierten Erkennung und Umwandlung kategorischer Variablen
220 def cat_encoder(data):
221     '''A function to automatically transform categorical variables into one hot
222         encoded variables
223
224         Deletes non-varying variables.
225
226         ''',
227     # Loeschen konstanter Variablen
228     initial = data.columns
229     for var in initial:
230         if (len(data[var].value_counts()) == 1):
231             data = data.drop(var, axis = 1)
232
233     vars_ = data.columns
234     num_vars = data._get_numeric_data().columns
235     non_num = list(set(vars_) - set(num_vars))
236
237     dfs = []
238
239     for var in non_num:
240         encoder = OneHotEncoder()
241         temp_object = encoder.fit_transform(data[[var]])
242         name = str(var) + '_' + encoder.categories_[0]
243         temp_df = pd.DataFrame(temp_object.toarray(), columns = name)
244         dfs.append(temp_df)
245         data = data.drop(var, axis = 1)
246     dfs.append(data)
247
248     outcome = pd.concat(dfs, axis = 1)

```

² In diesem Beispiel haben wir einen entsprechenden Encoder selber von Grund auf erzeugt. Es gibt natürlich auch fertige Pakete (z. B., `category_econders`) das solche Methoden fertig anbietet. Das programmieren eigener Methoden soll hier und an weiteren Stellen dazu dienen, die zugrundeliegenden Thematiken besser zu durchdringen.

```

248
249     return outcome
250
251 # Anwendung der Funktion
252 data_hr_transformed = cat_encoder(data_hr)

```

Durch die Anwendung dieser Funktion haben wir nun unseren Datensatz 'data_hr' so transformiert, dass nur noch numerische Variablen vorhanden sind. Damit wir nicht den Überblick über unsere Vorarbeit verlieren, wird dem transformierten Datensatz der Name 'data_hr_transformed' zugeordnet. Es sei angemerkt, dass im Rahmen dieses Prozesses auch unsere originale Zielvariable Attrition, welche die Werte Yes und No annimmt, in zwei Dummy-Variablen umgewandelt wurde: Attrition_Yes und Attrition_No. Im Folgenden werden wir die Dummy-Variable Attrition_Yes als Label nutzen.

Nachdem wir nun die nicht-numerischen Variablen transformiert haben, wenden wir uns als nächstes den numerischen, nicht-binären Variablen in unserem Datensatz zu. Numerische Variablen, welche nicht binär codiert sind, besitzen meistens unterschiedliche Skalen und Einheiten. Beispielsweise nimmt die Variable 'Age' Werte im Bereich von 18 bis 60 an und wird in Jahren gemessen. Die Variable 'DistanceFromHome' hingegen nimmt Werte im Bereich 1 bis 29 an und besitzt die Einheit Kilometer.

Einige ML-Algorithmen, z. B. Support Vector Machines oder Neuronale Netze, reagieren sensitiv auf unterschiedliche Skalen und weisen geringere Leistung auf, wenn Variablen stark unterschiedliche Werteausprägungen annehmen. Dies liegt kurz gesagt daran, dass Inputvariablen, die eine deutlich größere Varianz aufweisen als die anderer Inputvariablen, die Zielfunktion dominieren. Dadurch wird verhindert, dass wertvolle Informationen von den Variablen mit geringeren Varianzen extrahiert und Muster gelernt werden. In anderen Worten: Bei sich stark unterscheidenden Skalen kann es vorkommen, dass bestimmte Variablen allein wegen ihrer Skala übermäßige Auswirkungen auf das Modell haben.

Um einer solchen Verzerrung entgegen zu wirken, reskalieren wir die numerischen Inputvariablen. Die am häufigsten benutzten Methoden sind das Standardisieren und das Normalisieren der Variablen. Standardisieren bedeutet, dass eine Variable so skaliert wird, dass sie den Durchschnitt Null und die Varianz Eins besitzt. Dadurch werden Variablen mathematisch gesehen einheitslos und können direkt miteinander verglichen werden, da Veränderungen in Relation zu ihrer Standardabweichung interpretiert werden. Um den standardisierten Wert \hat{x} zu erhalten, subtrahieren wir von jedem Wert X den Durchschnitt der Variable μ . Anschließend wird die Differenz durch die Standardabweichung σ dividiert.

$$\hat{x} = \frac{X - \mu}{\sigma}$$

Von einer Normalisierung sprechen wir, wenn die Werte einer numerischen Variable so transformiert werden, dass sich die reskalierten Ausprägungen alle in einem Intervall zwischen zwei Zahlen, typischerweise zwischen 0 und 1 oder -1 und 1 , befinden. Um eine Skalierung zwischen 0 und 1 zu erreichen, subtrahieren wir von jedem Wert X einer

Variable den minimalen Wert den diese Variable in dem Datensatz annehmen kann X_{min} , und dividieren diesen Wert durch die Differenz zwischen dem maximalen Wert und den minimalen Wert den diese Variable in dem Datensatz annehmen kann ($X_{max} - X_{min}$). Zusammengefasst führen wir die folgende Operation durch:

$$\hat{x} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Für den Fall dass die Werte einer Variable nach der Reskalierung in einem Intervall zwischen zwei anderen Werten (y_{max} , y_{min}) liegen soll, erfolgt dies über die folgende Operation:

$$\tilde{x} = \hat{x} \cdot (y_{max} - y_{min}) + y_{min}$$

Sklearn stellt uns Klassen für diese beiden Transformationen zur Verfügung: `StandardScaler` und `MinMaxScaler`. Beide Verfahren können nur auf numerische Daten angewendet werden. Andere Datentypen müssen ausgeschlossen werden, ansonsten tritt ein Fehler auf. Die Umwandlung numerischer Variablen erfordert wenige Schritte. Das Codebeispiel [14.30](#) zeigt exemplarisch, wie die Umwandlung für die Variable `Age` erfolgen kann.

Listing 14.30 Reskalierung von Variablen

```

256 # Instanziierung der Klasse zur Normalisierung
257 normal_scaler = preprocessing.MinMaxScaler(feature_range = (min_, max_))
258 # Normalisierung der numerischen Variable Age
259 data_hr_transformed[['Age']] = normal_scaler.fit_transform(data_hr_transformed[['Age'
    ]])
260
261 # Instanziierung der Klasse zur Standardisierung
262 standard_scaler = preprocessing.StandardScaler()
263 # Standardisierung der numerischen Variable Age
264 data_hr_transformed[['Age']] = standard_scaler.fit_transform(data_hr_transformed[['
    Age']])

```

Um die Reskalierung flexibel und automatisiert durchführen zu können, erzeugen wir uns erneut eine Funktion. Diese Funktion akzeptiert ein `DataFrame` sowie die Angabe welche Art der Reskalierung durchgeführt werden soll als Input und gibt das entsprechend skalierte `DataFrame` als Output aus. Standardmäßig werden numerische Variablen im Intervall 0 und 1 normalisiert. Um zu einer Standardisierung zu wechseln, muss Inputparameter `normalization` auf `False` gesetzt werden. Die Funktion erkennt automatisch, welche Variablen umgewandelt werden sollen. Hinweis: Wie bei der Transformation der nicht-numerischen Variablen entfernt die Funktion numerische Variablen die keine Variation in unserem Datensatz aufweisen. Der Aufbau und die Anwendung der Funktion ist im Codebeispiel [14.31](#) aufgeführt.

Listing 14.31 Implementierung einer Funktion zur Reskalierung von Variablen

```

256 # Funktion zur Reskalierung numerischer Variablen durch
257 # Normalisierung oder Standardisierung
258 def rescaling(data, normalization = True, min_ = 0, max_ = 1):
259     '''A function to rescale numerical variables of a data set.
260
261     The default is normalization in the interval 0 and 1.
262     Change min_ and max_ to alter the interval.
263     To switch to standardization, set normalization = False'''
264
265
266 # Loeschen konstanter Variablen
267 initial = data.columns
268 for var in initial:
269     if (len(data[var].value_counts()) == 1):
270         data = data.drop(var, axis = 1)
271
272
273 # Identifikation der numerischen Variablen
274 num = data._get_numeric_data().columns
275
276 # Normalisierung (Default)
277 if normalization == True:
278
279     # Instanziierung der Klasse zur Normalisierung zwischen
280     # den Werten min_ und max_ die der Funktion uebergeben werden
281
282     normal_scaler = preprocessing.MinMaxScaler(feature_range = (min_, max_))
283
284     # Normalisierung der numerischen Variablen
285     data[num] = normal_scaler.fit_transform(data[num])
286
287 else:
288
289     # Instanziierung der Klasse zur Standardisierung
290
291     standard_scaler = preprocessing.StandardScaler()
292
293     # Standardisierung der numerischen Variablen
294     data[num] = standard_scaler.fit_transform(data[num])
295
296
297 return data
298
299 # Anwendung der Funktion mit Standard-Parametern
300 # Demnach normalisieren wir die Daten in den Bereich zwischen 0 und 1
301 data_hr_rescaled = rescaling(data_hr_transformed)

```

Mit der Reskalierung numerischer Variablen in unserem Datensatz haben wir die Vor- und Aufbereitung unseres Datensatzes abgeschlossen. Wie wir sehen können, müssen viele verschiedene Datentransformationen durchgeführt werden, bevor wir mit den eigentlichen Analysen und dem Erzeugen von Modellen beginnen können – sogar wenn der Datensatz wie in unserem Beispiel bereits in einer sehr hohen Qualität vorliegt. Es sollte daher nicht überraschen, dass diese Schritte bei ML-Projekten in der Regel den größten Teil der Zeit in Anspruch nehmen. An dieser Stelle sei angemerkt, dass die Schritte unbedingt in der richtigen Reihenfolge ausgeführt werden müssen und diese Reihenfolge über verschiedene Projekte hin variieren kann. Um die Transformationssequenzen im Allgemeinen zu erleichtern und zu automatisieren, stellt Sklearn die sogenannte Pipeline-Klasse zur Verfügung (`from sklearn.pipeline import Pipeline`). Da die Anwendung etwas fortgeschrittener ist, wird sie hier nicht behandelt. Stattdessen werden interessierte Leser auf weiterführende Literatur und Dokumentationen verwiesen.³

14.2.1 Mitarbeiterabwanderung: Sampling und Erzeugung von Trainings- und Testdaten

Als nächstes befassen wir uns mit Sampling-Methoden. Sampling-Methoden erlauben es die Anzahl der Observationen in einem Datensatz zu manipulieren. Beispielsweise ist es manchmal nicht praktikabel, mit dem gesamten Datensatz zu arbeiten, u. a. wenn er zu groß ist, um auf einmal verarbeitet zu werden. In diesem Fall kann man durch Sampling-Methoden eine Zufallsstichprobe aus den vorhandenen Daten ziehen und diese anstelle des gesamten Datensatzes dazu nutzen Analysen zu betreiben oder ML-Modelle zu erzeugen. Dies ist einfach erklärt möglich, da eine zufällig gezogene Stichprobe im Durchschnitt die gleichen statistischen Eigenschaften wie der gesamte Datensatz aufweist. Das Nutzen von Zufallsstichproben ist ein sehr häufig verwendetes Verfahren (siehe z. B. Stochastik Gradient Descent). Wie der Name suggeriert, wird eine definierte Menge an Observationen des gesamten Datensatzes zufällig und mit gleicher Wahrscheinlichkeit ausgewählt. Die Zufallsauswahl ist dabei entscheidend, da ansonsten Sample-Selektion-Probleme auftreten können, d. h., durch die Auswahl einer bestimmten Gruppe von Observationen identifizieren wir Zusammenhänge, die sich nicht verallgemeinern lassen.

Auch im Kontext des Samplings stellt uns Sklearn eine Methode zur Verfügung, die das Auswählen einer Zufallsstichprobe für uns übernimmt: Die „`sample_without_replacement`“ Methode. Die Methode geht dabei wie folgt vor. Zunächst wird ein numpy array erzeugt, das eine definierte Anzahl an ganzzahligen Werten beinhaltet, die zufällig aus einem definierten Intervall gezogen werden. Jeder Wert in dem Intervall wird mit der gleichen Wahrscheinlichkeit dem numpy array zugefügt. Dieses numpy array nutzen wir anschließend als Index-Liste, um die entsprechenden Observationen aus dem originalen Datensatz auszuwählen.

³ Siehe zum Beispiel: <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>, zuletzt abgerufen am 12. Februar 2022.

Das folgende Codebeispiel [14.32](#) zeigt diese zwei Schritte. Das neu erzeugte DataFrame mit dem Namen `random_sample` könnte dazu benutzt werden, um Analysen zu betreiben und ML-Modelle zu erzeugen.

Listing 14.32 Verwendung der Funktion `sample_without_replacement`

```

304 # Zufaelliche Auswahl 500 unterschiedlicher Observationen
305 random_index = sample_without_replacement(n_population=len(data_hr),      # Obere
      Grenze der ausgewaehlten Werte
306                                     n_samples=500,                      # Anzahl
      der ausgewaehlten Werte
307                                     random_state=0)                      #
      Kontrolle fuer moegliche Replikation
308
309 # Nachdem wir den Index erzeugt haben waehlen wir die entsprechenden Observationen
310 # ueber die df.iloc Operation aus.
311 random_sample = data_hr.iloc[data_hr_rescaled]
```

An dieser Stelle sei angemerkt, dass unser Datensatz mit 1470 Observationen es nicht erforderlich macht Modelle Anhand von Zufallsstickproben zu erzeugen. Die Anzahl der Observationen ist klein genug, damit wir mit den gesamten Daten arbeiten können.

Eine andere wichtige Funktion die Sampling übernimmt, ist das ausbalancieren der Daten, sodass die Anzahl der Observationen unterschiedlicher, vorherzusagender Zielvariablen Labels ausgeglichen ist. Wenn die Unterschiede in der Anzahl der Observationen die bestimmte Labels annehmen groß ist, kann es vorkommen, dass ML-Algorithmen die Mehrheitsklasse auf Kosten der Minderheitsklasse(n) deutlich präziser vorhersagen können. Dies ist vor allem dann ein Problem (aber nicht nur dann), wenn der unbalancierte Datensatz nicht die wahre Verteilung der Klassen in der Realität widerspiegelt, oder wenn wir daran interessiert sind, die Minderheitsklasse so präzise wie möglich vorherzusagen. Dieses Problem wird als „Sample-Imbalance-Problem“ bezeichnet. Die meisten ML-Algorithmen funktionieren am besten, wenn die Anzahl der Observationen in jeder Klasse etwa gleich groß ist. Dies liegt daran, dass die meisten Algorithmen darauf ausgelegt sind, die Accuracy zu maximieren und Fehler zu minimieren, was technisch gesehen „einfacher“ zu erreichen ist, wenn der Algorithmus den Fokus der Optimierung auf die Mehrheitsklasse legt.

Um zu sehen ob und wie stark die vorherzusagenden bzgl. der Ausprägungen des Labels (=Klassen) in einem Datensatz unbalanciert sind, können wir die zuvor bereits angewendete Pandas Methode `value_counts()` verwenden (siehe Codebeispiel [14.23](#)). Werden keine weiteren Parameter übergeben, wird die absolute Anzahl der einzelnen Ausprägungen der übergebenen Variable ausgegeben. Übergeben wir zusätzlich den Parameter `'normalize=True'`, erhalten wir die relativen Anteile. Wir wenden diese Methode auf unseren aufbereiteten Datensatz in Bezug auf unser Label `Attrition_Yes` an. Das Codebeispiel [14.33](#) zeigt die Anwendung und die Konsolenausgabe [14.34](#) die resultierende Ausgabe der `value_counts()` Methode.

Listing 14.33 Identifikation von Imbalances

```

304 # Anzeige der Wertaussprägungen der Variable unseres Interesses
305 data_hr_rescaled['Attrition_Yes'].value_counts()

```

Listing 14.34 Konsolenausgabe

```

1  0.0 1233
2  1.0 237
3  Name: Attrition_Yes, dtype: int64

```

Wie wir sehen können, ist die Anzahl der Klasse die wir eigentlich vorhersagen möchten (Attrition_Yes = 1), deutlich in der Unterzahl. Dementsprechend ist es sinnvoll die Klassen auszubalancieren.

Zwei in der Praxis häufig verwendete Möglichkeiten dieses Problem abzuschwächen sind: (i) Zufällig ausgewählte Observationen der Mehrheitsklasse aussortieren bis die Anzahl der Observationen der Mehrheits- und Minderheitsklasse in etwa ausbalanciert sind (Under-sampling), und (ii) dem Datensatz vorhandene, zufällig ausgewählte Datenpunkte der Minderheitsklasse duplizieren, bis die Anzahl der Observationen der Mehrheits- und Minderheitsklasse etwa ausbalanciert sind (Oversampling).

Beide Varianten haben ihre Tücken. Undersampling führt dazu, dass der Datensatz verkleinert wird und möglicherweise äußerst informationsreiche Observationen im Trainingsprozess nicht beachtet werden. Undersampling fügt dem Datensatz die gleichen Observationen mehrfach hinzu, sodass Algorithmen diese einzelnen Datenpunkte möglicherweise eher „auswendig“ lernen, anstelle statistische Muster zu erkennen. Die Nachteile der beiden Methoden müssen immer im Einzelfall abgewogen werden, sodass es auf den vorhandenen Datensatz und den auszuwählenden Algorithmus ankommt, welche Variante besser geeignet ist. Hinweis: Insbesondere bei Oversampling ist es kritisch die Daten in Trainings- und Testdaten aufzuteilen BEVOR wir die Klassen ausbalancieren. Tun wir dies nicht, kann das dazu führen, dass genau dieselben Beobachtungen sowohl in dem Trainings- als auch in dem Testdatensatz vorhanden sind, wodurch die Performance und Generalisierung des Modells überschätzt werden können.

Beide Varianten können durch die Sklearn-Methode `resample` implementiert werden. Diese Methode akzeptiert einen Datensatz der auf eine bestimmte Anzahl von Observationen reduziert oder erhöht werden soll als Input und liefert einen größeren oder kleineren Datensatz als Ausgabe. Dazu geben wir zusätzlich an, wie viele Observationen (`n_samples`) der neue Datensatz enthalten soll und ob Observationen aus dem ursprünglich übergebenen Datensatz mehrfach vorkommen dürfen (`replace`), was notwendigerweise der Fall sein muss, wenn upsampling betrieben wird. Das folgende Codebeispiel [14.35](#) zeigt die Anwendung der `resampling`-Methode für Under- und Oversampling.

Listing 14.35 Verwendung der Funktion Undersampling

```

320 # Upsampling der Minderheitsklasse
321 up_sampled = resample(data_hr_rescaled[data_hr_rescaled['Attrition_Yes'] == 1],
                        # Datensatz der resampled werden soll

```

```

322         n_samples=len(data_hr_rescaled[data_hr_rescaled['
           Attrition_Yes'] == 0]),      # Neue Anzahl der enthaltenen
           Datenpunkte
323         replace=True)

           # Neue Datenpunkte koennen mehrfach gezogen werden
324 # Konkatenation der beiden Klassen
325 balanced_upsample = pd.concat([data_hr_rescaled[data_hr_rescaled[' Attrition_Yes' ] ==
           0], up_sampled])
326
327 # Downsampling der Mehrheitsklasse
328 down_sampled = resample(data_hr_rescaled[data_hr_rescaled[' Attrition_Yes' ] == 0],
           # Datensatz der resampled werden soll
329         n_samples=len(data_hr_rescaled[data_hr_rescaled['
           Attrition_Yes' ] == 1]),      # Neue Anzahl der
           enthaltenen Datenpunkte
330         replace=False)

           # Neue Datenpunkte koennen NICHT mehrfach gezogen werden
331 # Konkatenation der beiden Klassen
332 balanced_downsample = pd.concat([data_hr_rescaled[data_hr_rescaled[' Attrition_Yes' ]
           == 1], down_sampled])

```

Um unsere Arbeit zu reduzieren und die Automatisierung zu erhöhen, definieren wir erneut eine Funktion, die wiederholt angewendet werden kann und es flexibel erlaubt sowohl Under- als auch Oversampling zu betreiben. Die Funktion nimmt zunächst 2 Parameter entgegen: (i) die Variable anhand der Resampling betrieben werden soll und (ii) das DataFrame-Objekt das wir balancieren möchten. Dann prüft die Funktion ob die Klassen ausbalanciert sind oder nicht und gibt uns dies aus. Wenn die Klassen nicht balanciert sind werden wir aufgefordert anzugeben, ob Under- oder Oversampling betrieben werden soll. Die Ausgabe der Funktion ist ein ausbalancierter Datensatz in Form eines DataFrame-Objekts. Der Aufbau und die Anwendung der Funktion ist im Codebeispiel [14.36](#) dargestellt. Die Ausgabe der Funktion für Undersampling findet sich in der Konsolenausgabe [14.37](#).

Listing 14.36 Implementierung einer Methode zum Ausbalancieren von Daten

```

320 # Definition einer Funktion die es erlaubt einen uebergebenen Datensatz
321 # durch Over- oder Undersampling zu Balancieren.
322 def balancing(target, df):
323     ''' A simple function to balance a data set according to target input'''
324
325     if df[target].value_counts()[0] == df[target].value_counts()[1]:
326         print('Die Klassen sind ausbalanciert: ', '\n', df[target].value_counts())
327
328         return df
329
330     elif df[target].value_counts()[0] > df[target].value_counts()[1]:
331         maj_class = df[target].value_counts().keys()[0]

```

```

332     maj = df[target].value_counts()[0]
333     majority = df[df[target] == df[target].value_counts().keys()[0]]
334
335     min_class = df[target].value_counts().keys()[1]
336     min_ = df[target].value_counts()[1]
337     minority = df[df[target] == df[target].value_counts().keys()[1]]
338
339     elif df[target].value_counts()[0] < df[target].value_counts()[1]:
340         maj_class = df[target].value_counts().keys()[1]
341         maj = df[target].value_counts()[1]
342         majority = df[df[target] == df[target].value_counts().keys()[1]]
343
344         min_class = df[target].value_counts().keys()[0]
345         min_ = df[target].value_counts()[0]
346         minority = df[df[target] == df[target].value_counts().keys()[0]]
347
348     print('Die Klassen sind nicht ausbalanciert: \n'
349         '\nDie Mehrheitsklasse ist ', target, '=', maj_class, ' mit ', maj, '
350         'Observationen.'
351         '\nDie Minderheitsklasse ist ', target, '=', min_class, ' mit ', min_, '
352         'Observationen.')
```

351

```

352     sampling = input('\nSoll Undersampling (u) oder Oversampling (o) betrieben werden
353     ? ')
354
355     while sampling not in ['u', 'U', 'o', 'O']:
356         sampling = input('Ungueltige Eingabe.' '\nSoll Undersampling (u) oder
357         Oversampling (o) betrieben werden? ')
358
359     if sampling in ['U', 'u']:
360
361         down_sampled = resample(majority,
362                                 n_samples=min_,
363                                 replace=False)
364
365         output = pd.concat([minority, down_sampled]).sample(frac=1)
366
367         print(' \nDer Datensatz wurde durch Undersampling ausbalanciert.')
```

366

```

367     elif sampling in ['O', 'o']:
368
369         up_sampled = resample(minority,
370                                 n_samples=maj,
371                                 replace=True)
372
373         output = pd.concat([majority, up_sampled]).sample(frac=1)
374
375         print(' \nDer Datensatz wurde durch Oversampling ausbalanciert.')
```

376

```

377     return output
378
379 # Anwendung der definierten Funktion
380 temporary = balancing('Attrition_Yes', data_hr_rescaled)

```

Listing 14.37 Konsolenausgabe

```

1 Die Klassen sind nicht ausbalanciert:
2
3 Die Mehrheitsklasse ist Attrition_Yes = 0.0 mit 1233 Observationen.
4 Die Minderheitsklasse ist Attrition_Yes = 1.0 mit 237 Observationen.
5
6
7 Soll Undersampling (u) oder Oversampling (o) betrieben werden? u
8
9 Der Datensatz wurde durch Undersampling ausbalanciert.

```

Der letzte Schritt bevor wir mit den Analysen und dem Erzeugen von Modellen beginnen können, ist das Aufteilen unserer Daten in zwei verschiedene Partitionen: dem Trainings- und dem Testdatensatz. Wie die Namen der beiden Partitionen suggerieren wird der Trainingsdatensatz dazu verwendet Modelle zu trainieren, wohingegen der Testdatensatz dazu dient, die erzeugten Modelle auf ihre Generalisierbarkeit zu testen.

Um die Daten in Trainings- und Testdatensatz aufzuteilen, kann man verschiedene Möglichkeiten nutzen. Sklearn bietet die Methode `train_test_split()` zur automatischen und zufälligen Partitionierung des Ausgangsdatensatzes. Eine Faustregel bezüglich des Verhältnisses zwischen Training- und Testdaten ist ca. 4 zu 1, also ca. 80 % der Observationen werden als Trainingsdaten und die restlichen 20 % als Testdaten genutzt, um zu testen inwiefern das Modell auf Daten performed, die es bis dato nicht gesehen hat. In anderen Worten: ob das Modell sich gut verallgemeinern lässt. Die `train_test_split()` Methode nimmt den gesamten Datensatz aufgeteilt in Label und Features entgegen und liefert uns jeweils einen Trainings- und Testdaten, die getrennt in Features und Labels ausgegeben werden. Zu beachten: Das Label, welches uns interessiert, ist `Attrition_Yes`. Der Datensatz enthält noch die Variable `Attrition_No`, die natürlicherweise eine perfekte negative Korrelation zu `Attrition_Yes` aufweist. Diese Variable darf nicht Teil der Features sein und muss daher, wie im folgenden Beispiel aufgezeigt, entfernt werden. Eine beispielhafte Aufteilung unseres vorbereiteten Datensatzes in Trainings- und Testdaten findet sich im Codebeispiel 14.38. Für das Training unserer ML-Modelle an späterer Stelle werden wir die `train_test_split()` Methode analog verwenden.

Listing 14.38 Verwendung der Funktion `train_test_split`

```

388 # Zufällige Aufteilung des Datensatzes in Trainings- und Testdatensatz
389 X_train, X_test, Y_train, Y_test = train_test_split(
390     data_hr_rescaled.drop(['Attrition_Yes', 'Attrition_No'], axis=1), #
391     data_hr_rescaled['Attrition_Yes'],                               # Binaeres
392     test_size=0.2,                                                  # Groesse des
393     random_state=0)

```

14.2.2 Explorative Analysen auf Trainingsdaten

Jetzt da wir verstehen wie unsere zunächst aufbereiteten Daten in Trainings- und Testdaten aufgeteilt werden können, können wir damit beginnen die Trainingsdaten (zunächst explorativ) auf Muster zu untersuchen. Die Partition der Daten, bevor mit den Analysen begonnen wird, ist wichtig, da wir die Testdaten ausschließlich dazu nutzen wollen, ganz am Ende zu testen, inwiefern sich unsere Erkenntnisse verallgemeinern lassen. Vor diesem Hintergrund gehört es auch dazu, dass die Testdaten nicht bereits auf Muster zu untersuchen – nicht einmal explorativ. Zusammenhänge die wir glauben zu erkennen, könnten unsere Wahrnehmung und Eindrücke verzerren und weitere Analysen und letztlich das zu schätzende Modell (bewusst oder unbewusst) beeinflussen. Wenn wir das Modell dann auf den Testdaten auf Generalisierbarkeit testen, würden wir mit hoher Wahrscheinlichkeit eine zu hohe Performance messen und somit Overfitting betreiben.

An dieser Stelle macht es Sinn, noch einmal genauer auf Overfitting Probleme einzugehen. Supervised Learning Algorithmen sind drauf ausgelegt, in Daten vorhandene funktionale Zusammenhänge zu approximieren, welche sich gut generalisieren lassen. Overfitting bezieht sich auf das Problem, dass das Modell die vorhandenen Daten auf denen es trainiert wurde, äußerst präzise erklären kann, allerdings bei neuen Daten, die nicht in den Trainingsprozess einbezogen wurden, nur eine geringe Performance aufweist. Eine geringe Performance auf Testdaten interpretieren wir als ein deutliches Zeichen dafür, dass sich unsere Erkenntnisse nicht verallgemeinern lassen, obwohl dies das eigentliche Ziel von ML-Algorithmen ist (eine hohe Out-of-sample Performance).

Overfitting Probleme treten bei sehr vielen ML-Algorithmen auf und hängen mit dem größten Vorteil zusammen, den ML-Algorithmen bieten: das approximieren flexibler Funktionen die sich komplexen, hochdimensionalen Datenstrukturen anpassen. Jeder dieser möglichen Funktionen stellt letztendlich eine Hypothese über die realen Zusammenhängen dar. Umso komplexer die Funktionen die wir schätzen sind, desto mehr Hypothesen existieren und desto besser können vorliegende Datensätze durch das Modell erklärt werden. Allerdings würde die Funktion dann auch lernen in den Trainingsdaten auftretendes, statistisches Rauschen und Ausreißer zu approximieren, welche nur durch zufällige Korrelationen ein Muster in dem vorhandenen Datensatz erkennen lassen und nicht verallgemeinert werden können.

Eine Möglichkeit Overfitting Probleme einzudämmen wird als Regularisierung bezeichnet. Regularisierung ist letztlich nichts anderes als das Limitieren der Komplexität der zu schätzenden Funktion. Wenn wir einen ausreichenden Grad der Regularisierung auswählen, können wir Vorteile der komplexen und flexiblen Funktionen ausnutzen, ohne dabei den Fehler zu begehen die Funktion übermäßig auf die vorliegenden Daten anzupassen. Leider gibt es keine allgemeingültigen Verfahren oder Regeln die wir anwenden können um den optimalen Grad der Regularisierung zu bestimmen. Dieser muss durch empirisches Testen identifiziert werden. Wenn wir unsere ML-Modelle trainieren, werden wir Regularisierung durch die Auswahl von Hyperparametern betreiben.

Zunächst beginnen wir allerdings mit einer kurzen explorativen Analyse, die dabei helfen kann die Spezifikation von Modellen zu informieren, bspw. indem wir auf Basis dieser Analysen die Auswahl der Inputfeatures eingrenzen. Dabei sei darauf hingewiesen, dass die im Folgenden trainierten ML-Modelle auch ohne die explorativen Analysen erzeugt werden können. Explorative Analysen sind besonders dann sinnvoll, wenn die Auswahl der Features für ein Modell manuell ausgewählt werden bzw. eine Begrenzung der Features gegeben dem Modell sinnvoll ist (z. B. bei logistischen Regressionen). Dennoch soll an diesem Punkt kurz auf explorative Analysen eingegangen werden, da diese häufig einen wichtigen Teil der Arbeit in einem Data Science Projekt ausmachen.

Ein guter Startpunkt für explorative Analysen ist die Betrachtung von einfachen Korrelationen zwischen unserem Label und unseren Features (in ökonomischen Begriffen: unserer abhängigen und unseren unabhängigen Variablen). Dazu konkatenieren wir den Vektor `Y_train`, der die Labels der Observationen beinhaltet, mit der Feature Matrix `X_train` und nutzen Pandas' Methode `DataFrame.concat()`. Diese erzeugt eine einfache Korrelations-Matrix. Hinweis: Wir nutzen die „spearman“ Spezifikation, d. h. es werden standardisierte Korrelationskoeffizienten zwischen -1 und 1 ausgegeben. Anschließend schauen wir uns die (aufsteigend sortierten) Korrelationskoeffizienten für unser Label an. Das Codebeispiel [14.39](#) zeigt das Vorgehen.

Listing 14.39 Konkatenieren der Features und Labels und Korrelationsberechnung

```

408 # Konkatenieren der Features und Labels des Trainingsdatensatzes
409 train_set_w_labels = pd.concat([X_train, Y_train], axis=1)
410
411 # Erzeugung einer Korrelationsmatrix
412 pearson_matrix = train_set_w_labels.corr(method = 'spearman')
413
414 # Korrelationen zwischen der abhaengigen und den unabhaengigen Variablen
415 pearson_matrix['Attrition_Yes'].sort_values(ascending=False)
```

Das Ergebnis der Korrelationen zwischen unserem Label und unseren Features zeigt die Konsolenausgabe [14.40](#).

Listing 14.40 Konsolenausgabe

```

1  Attrition_Yes                1.000000
2  OverTime_Yes                0.288263
3  MaritalStatus_Single       0.252404
4  ...
5  JobRole_Research Scientist -0.001251
6  ...
7  OverTime_No                -0.288263
8  TotalWorkingYears          -0.290993
9  MonthlyIncome              -0.294903
10
11 Name: Attrition_Yes, dtype: float64
```

Die Ausgabe liefert uns erste Informationen über lineare Zusammenhänge in den Daten. Korrelationskoeffizienten liegen stets in dem Intervall zwischen 1 und -1. Umso größer der absolute Wert eines Koeffizienten für zwei Variablen, desto stärker ausgeprägt ist deren linearer Zusammenhang. Ein Wert nahe 0 besagt, dass kein/ nur ein sehr schwacher linearer Zusammenhang besteht. Die Ausgabe zeigt uns bspw., dass ein (relativ) starker, negativer Zusam-

menhang zwischen der Abwanderung und dem monatlichen Einkommen besteht. Dementsprechend ist das durchschnittliche monatliche Einkommen von Personen, die nicht abgewandert (Attrition_Yes = 0) sind, höher, als das durchschnittliche monatliche Einkommen abgewanderter Personen (Attrition_Yes = 1). Umgekehrt deutet sich an, dass Überstundenleistende Mitarbeiter tendenziell eher abwandern, als diejenigen welche dies nicht tun. Es ist wichtig zu beachten, dass Korrelationen (i) keine kausalen Zusammenhänge abbilden (z. B. können wir nicht darauf schließen, dass Überstunden Abwanderung auslösen) und (ii) nur lineare Zusammenhänge abbilden können (z. B. können wir anhand der Ausgabe nicht darauf schließen, dass „JobRole_Research Scientist“ unabhängig von der Abwanderung ist, da selbst quadratische Zusammenhänge nicht erkannt werden). Vor allem Verfahren wie Support-Vector-Maschinen oder Neuronale Netze identifizieren höher-dimensionale Zusammenhänge, die einfache lineare Korrelationsmaße nicht erfassen. Demnach sollten Features die eine geringe Korrelation mit dem Label aufweisen nicht zwangsläufig ignoriert werden, da wichtige nicht-lineare Zusammenhänge existieren könnten.

Eine weitere Möglichkeit sich einen ersten Eindruck über strukturelle Zusammenhänge in den Daten zu verschaffen, besteht darin, grafische Illustrationen zu erzeugen. Hierzu können die von Matplotlib und Seaborn zur Verfügung gestellten Methoden verwendet werden. Wir beginnen damit, Scatter- und Pairplots der numerischen Variablen zu erzeugen, die laut der Korrelationen einen relevanten Zusammenhang mit dem Label/der abhängigen Variable erahnen lassen. Wir erzeugen die Verteilungen und 2-D Zusammenhänge für beide Klassen unseres Labels (Attrition_Yes = 1: Orange, Attrition_Yes = 0: Blau). Das Codebeispiel [14.41](#) zeigt die Umsetzung und die Abb. [14.1](#) das entsprechende Ergebnis.

Listing 14.41 Pairplot der ausgewählten Variablen

```

418 # uebersicht zu Verteilungen und Moeglichen Zusammenhaengen durch einen Pairplot
419 sns.pairplot(train_set_w_labels[['Attrition_Yes',
420                                'MonthlyIncome',
421                                'JobLevel',
422                                'YearsAtCompany',
423                                'JobSatisfaction',
424                                'DistanceFromHome',
425                                'JobInvolvement',
426                                'WorkLifeBalance',
427                                'DailyRate']],
428             hue = 'Attrition_Yes',                               # 3. Dimension des
429             markers=['D', 'o'],                                   # Unterschiedliche
430             kind='reg')                                           # Anzeigen linearer
                                                                    Zusammenhaenge

```

Die grafischen Illustrationen geben uns einige weitere Aufschlüsse über mögliche Zusammenhänge. Beispielsweise deutet sich an, dass Individuen die abgewandert sind (im Vergleich zu denen die nicht abgewandert sind), einen längeren Weg zur Arbeit haben. Ebenfalls



Abb. 14.1 Der Pairplot des Trainsets

suggestieren die Darstellungen erneut, dass das monatliche Einkommen eines nicht abgewanderten Arbeiters höher als der eines Arbeiters der abgewandert ist. Gleichzeitig scheint es so, dass die Anstellungsdauer positiv mit dem monatlichen Einkommen zusammenhängt. Es könnte daher sein, dass die Abwanderung, und somit eine kürzere Anstellungsdauer, die Höhe des monatlichen Einkommens begrenzt, sodass eine negative Korrelation vorliegt.

Neben dem Pairplot gibt es weitere Möglichkeiten zur grafischen Illustration von Zusammenhängen und Verteilungen die sich in den vorliegenden Daten entdecken lassen. Häufig verwendete Methoden sind (i) relation plots, (ii) joint plots und (iii) categorical plot. Das folgende Codebeispiel 14.42 zeigt wie diese Methoden angewendet werden. Die resultierenden Plots finden sich in den nachfolgenden Abb. 14.2a bis 14.2e.

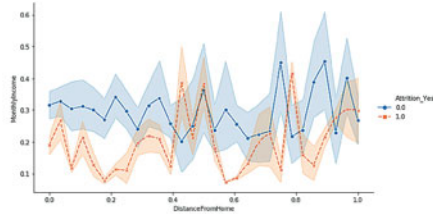
Listing 14.42 Weitere Abbildungen

```

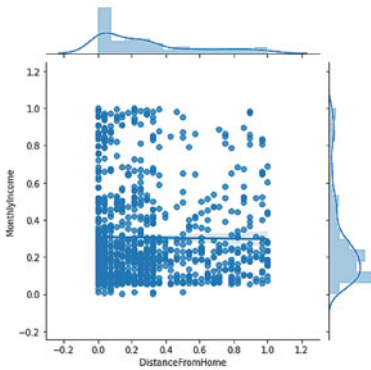
45 # Beziehungen zwischen Mittelwerten
46 sns.relplot(x='DistanceFromHome',
47             y='MonthlyIncome',
48             hue='Attrition_Yes', style='Attrition_Yes', markers=True,
49             kind='line',
50             height=5, aspect=1.7,
51             data=train_set_w_labels)
52
53 # Scatterplot, Verteilungen und lineare Zusammenhaenge
54 sns.jointplot('DistanceFromHome',
55              'MonthlyIncome',
56              data=train_set_w_labels[train_set_w_labels['Attrition_Yes']==0],
57              kind='reg')
58
59 sns.jointplot('DistanceFromHome',
60              'MonthlyIncome',
61              data=train_set_w_labels[train_set_w_labels['Attrition_Yes']==1],
62              kind='reg')
63
64 # Illustrationen fuer kategorische Variablen durch Verteilungen
65 sns.catplot(x='JobSatisfaction',
66            y='MonthlyIncome',
67            hue='Attrition_Yes',
68            data=train_set_w_labels,
69            kind='bar')
70
71 sns.catplot(x='JobSatisfaction',
72            y='MonthlyIncome',
73            hue='Attrition_Yes',
74            data=train_set_w_labels,
75            kind='violin')
76
77 sns.catplot(x='JobSatisfaction',
78            y='MonthlyIncome',
79            hue='Attrition_Yes',
80            data=train_set_w_labels,
81            kind='box')

```

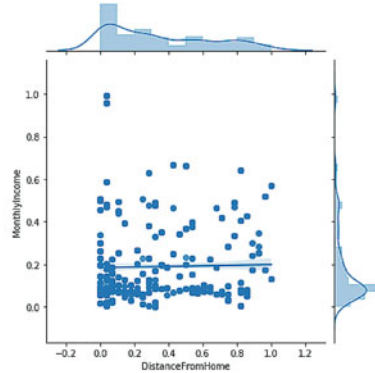
An dieser Stelle werden wir die explorativen Analysen beenden und uns auf das Training von ML-Algorithmen konzentrieren. Abschließend sei noch angemerkt, dass einfache explorative Analysen auch deswegen wichtig sind, um sich zunehmend mit Daten vertraut zu machen und deren Struktur besser zu verstehen. Dies ist für die erfolgreiche Durchführung von Data-Science-Projekten unabdinglich.



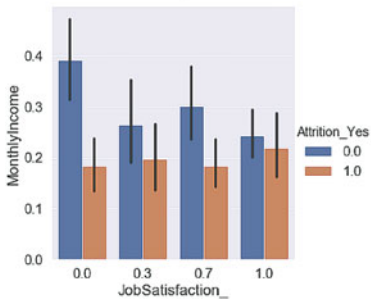
(a) Der Relplot auf Basis der Daten.



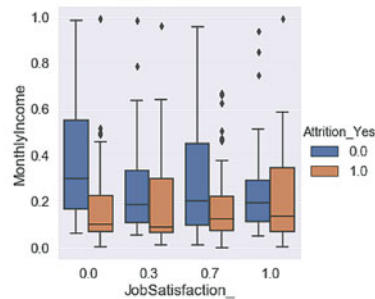
(b) Rel plot: Label = 0



(c) Rel plot: Label = 1



(d) Catplot der Variante 'bar'



(e) Catplot der Variante 'box'

Abb. 14.2 (a) Der Relplot auf Basis der Daten. (b) Rel plot: Label = 0. (c) Rel plot: Label = 1. (d) Catplot der Variante ‚bar‘. (e) Catplot der Variante ‚box‘

14.2.3 Auswahl und Training von ML Modellen

Nach unserer Vorarbeit sind wir bereit, ML Modelle auszuwählen, zu trainieren und auf deren Generalisierbarkeit zu prüfen. Dank der vorherigen Schritte wird der Arbeitsaufwand für das Trainieren und Testen der Modelle sich in Grenzen halten. Wie zuvor bereits beschrieben, bezeichnet das Training von Modellen das schätzen derer Parameter, sodass das Modell die

im Trainingsdatensatz vorhandenen Muster bestmöglich approximieren kann. Das Testen ist die Anwendung und Berechnung der Güte des geschätzten Modells auf dem Testdatensatz.

Wir beginnen mit der Erzeugung eines recht simplen ML Modells: K-Nearest Neighbour (KNN). KNN ist ein Beispiel für instanzbasierte Lernalgorithmen. Das bedeutet, dass der Algorithmus neue Observationen klassifiziert, indem die neuen Observationen direkt mit allen in dem Trainingsdatensatz vorhandenen Observation auf ihre Ähnlichkeit verglichen werden. Die Klassifikation einer neuen Observation erfolgt auf Basis einer Mehrheitsentscheidung der k ähnlichsten Observationen im Trainingsdatensatz. Einer neuen Observation wird immer diejenige Klasse zugeordnet, welcher die Mehrheit der k ähnlichsten Observationen im Trainingsdatensatz angehört.

Bezogen auf unsere Fragestellung bedeutet dies, dass die Abwanderung eines neuen Mitarbeiters dadurch bestimmt wird, dass der Algorithmus die k ähnlichste(n) Person(en) aus dem Trainingsdatensatz identifiziert, deren Klasse identifiziert (Attrition_Yes=0 oder Attrition_Yes=1) und zählt, wie häufig die einzelnen Klassen vorkommen. Die am häufigsten unter den k ähnlichsten Observationen vorkommende Klasse wird der neuen Observation zugeordnet. Die Ähnlichkeit ist die metrische Distanz zwischen den Observationen in Bezug auf deren Attribute bzw. unabhängigen Variablen. Je „weiter entfernt“ zwei Observationen sind, umso weniger ähnlich sind sie sich; je „näher“ sich zwei Observationen sind, umso ähnlicher sind sie sich. Häufig wird die Euklidische Distanz als Metrik für die Ähnlichkeit zwischen den Observationen verwendet. Ein Beispiel: Nehmen wir an, dass drei unterschiedliche Datenpunkte (Observationen) durch die Vektoren $a=(0,0,0)$, $b=(1,2,3)$ und $c=(1,1,1)$ beschrieben sind, wobei a und b Datenpunkte aus einem Trainingsdatensatz sind und c eine neue Observation ist. Jeder Zahlenwert repräsentiert die Ausprägung einer Variable bspw. (x , y , z). Die euklidische Distanz zwischen den Datenpunkten $a=(0,0,0)$ und $b=(1,2,3)$ beträgt $\sqrt{(0-1)^2 + (0-2)^2 + (0-3)^2} \approx 3742$. Die euklidische Distanz zwischen den Datenpunkten $a=(0,0,0)$ und $c=(1,1,1)$ beträgt $\sqrt{(0-1)^2 + (0-1)^2 + (0-1)^2} \approx 1732$. Somit wären sich die Observationen a und c ähnlicher als die a und b und wir würden c die gleiche Klasse zuordnen der a zugeordnet ist.

Hinweis: Vor der Anwendung des KNN Algorithmus undersampeln wir unsere Trainingsdaten, damit die Anzahl der Klassen ausbalanciert ist. Würden wir Oversampling auf dem Trainingsdatensatz betreiben, würde der Trainingsdatensatz identische Observationen mehrfach beinhalten. Dies würde dazu führen, dass Datenpunkte die nach dem Oversampling in unserem Trainingsdatensatz mehrfach vorkommen, einen stärkeren Einfluss auf die Klassifizierung neuer Observationen haben. Als Ergebnis würde unser Modell verzerrte Ergebnisse liefern.

Wir beginnen damit, die benötigten Klassen zur Erzeugung von KNN Modellen zu importieren. Anschließend teilen wir unsere Daten in Trainings- und Testdaten zu separieren (75 % vs. 25 %) und wenden unsere zuvor definierte balancing Funktion auf den Trainingsdaten an, um Undersampling zu betreiben. Anschließend können wir damit beginnen, das Modell zu trainieren. Dafür instanziiieren wir das KNN Modell mit dem zufällig ausgewählten Wert $k = 10$. Der Wert k , also die Anzahl der zu prüfenden ähnlichsten Observationen, ist der

Hyperparameter des Modells, den wir im Folgenden noch optimieren werden. Das instanzierte Modell wird über `.fit()` auf den übergebenen Trainingsdaten trainiert und wir erhalten unser Modell dem wir den Namen `knnmodel` geben. Die Implementierung findet sich im Codebeispiel 14.43 zeigt die einzelnen Schritte.

Listing 14.43 Partition der Daten in Trainings- und Testdaten und Training des Modells

```

478 # Importieren der Algorithmus Klasse und der Methoden zu Berechnung von Performance-
      Kennzahlen
479 from sklearn.neighbors import KNeighborsClassifier
480
481 # Aufteilen der Daten in Trainings- und Testdaten
482 X_train, X_test, Y_train, Y_test = train_test_split(
483     data_hr_rescaled.drop(['Attrition_Yes', 'Attrition_No'], axis=1), # Features
484     data_hr_rescaled['Attrition_Yes'],                               # Binaeres
      Label unseres Interesses
485     test_size=0.25,                                                # Groesse des
      Testdatensatzes
486     random_state=0)
487
488 # Undersampling der Trainingsdaten
489 balanced = balancing('Attrition_Yes', pd.concat([X_train, Y_train], axis=1))
490
491 # Separierung von Labels und Features
492 X_train = balanced.drop(['Attrition_Yes'], axis=1)
493 Y_train = balanced['Attrition_Yes']
494
495 # Instanziierung und Training des Algorithmus
496 # Wir beginnen mit dem zufaellig ausgewaehlten Wert k=10.
497 knnmodel = KNeighborsClassifier(n_neighbors=10).fit(X_train, Y_train)

```

Durch die `model.predict()` Methode können wir das trainierte Modell jetzt dazu nutzen, um auf Basis einer übergebenen Feature-Matrix Vorhersagen über die zu diesen Features gehörenden Labels, also Klassen, zu erzeugen. Wie bereits erklärt, bewerten wir die Güte eines Modells anhand dessen Performance auf dem Testdatensatz, da diese Datenpunkte nicht in den Trainingsprozess miteinbezogen wurden. Dies geschieht in 2 Schritten: (i) Wir nutzen die Testdaten-Features `X_test`, um auf Basis des Modells mit der `model.predict()` Methode Vorhersagen `Y_pred` über die uns bekannten Labels `Y_test` zu erzeugen. (ii) die vorhergesagten Labels `Y_pred` vergleichen wir mit den wirklichen Labels `Y_test`. Der Vergleich erlaubt es uns zu identifizieren, wie häufig das Modell die verschiedenen Klassen der verschiedenen Observationen der Testdaten richtig oder falsch vorhergesagt hat. Auf Basis dieser Zahlen lassen sich unterschiedliche Performance-Kennzahlen herleiten, die uns einen Überblick über die Generalisierbarkeit des Modells verschaffen. Wir fokussieren uns auf 3 verschiedene Kennzahlen:

- Precision = $\frac{TP}{TP+FP}$
- Recall = $\frac{TP}{TP+FN}$
- Accuracy = $\frac{TP+FP}{TP+FP+TN+FN}$

Dabei bezeichnet (i) TP die Anzahl der Observationen die korrekt als Attrition_Yes=1 identifiziert werden, (ii) FP die Anzahl der Observationen die fälschlich als Attrition_Yes=1 identifiziert werden, (iii) TN die Anzahl der Observationen die korrekt als Attrition_Yes=0 identifiziert werden und (iv) FN die Anzahl der Observationen die korrekt als Attrition_Yes=0 identifiziert werden. Dementsprechend zeigt uns die Precision-Kennzahl die Genauigkeit des Modells bezogen auf den Anteil der korrekt als Attrition_Yes=1 identifizierten Observationen unter allen als Attrition_Yes=1 identifizierten Observationen. Die Recall-Kennzahl hingegen gibt uns Aufschluss über die Sensitivität des Modells bezogen auf den Anteil der richtig erkannten Attrition_Yes=1 Observationen unter allen Observationen deren Label wirklich Attrition_Yes=1 ist. Die Accuracy-Kennzahl, wie der englische Name andeutet, gibt den Anteil der Observationen an, für die das Modell das korrekte Label vorhergesagt hat. Jede dieser Kennzahlen hat individuell ihre eigenen Tücken. Insgesamt sollten daher stets alle Werte betrachtet werden. Der einzelne Beitrag zur Bewertung der Güte des Modells hängt letztlich von der Fragestellung ab. Insbesondere ist zu beachten, dass die Precision- und Recall-Metriken häufig gegenläufig sind, da die Identifikation möglichst vieler der wirklichen Attrition_Yes=1 Observationen mit einer erhöhten Fehlklassifikation zusammenhängt (Erhöhung der FP Klassifikationen).

Zur Berechnung der verschiedenen Kennzahlen stellt uns Sklearn vordefinierte Methoden zur Verfügung, welche wir importieren und anschließend nutzen können. Da wir wiederholt verschiedene Modelle entwickeln und auf deren Güte testen, definieren wir uns eine Funktion, die ein Modell, ein Feature-DataFrame und den zugehörigen Label-Vektor als Eingabeparameter akzeptieren und als Ausgabe verschiedene Performance-Metriken liefert: Eine Konfusionsmatrix (zur Anzeige von TP, FP, TN, FN), Accuracy, Precision und Recall. Anschließend nutzen wir die definierte Funktion, um zu sehen wie gut unser Modell ist. Dazu übergeben wir unser Modell und die Testdaten (getrennt als Feature-DataFrame und Label Vektor). Die Implementierung und Anwendung Funktion findet sich im Codebeispiel 14.44. Die erzeugte Konfusionsmatrix ist in Abb. 14.3a zu finden, die Ausgabe in Konsolenausgabe 14.45.

Listing 14.44 Implementierung einer Methode zur Überprüfung der Modell Performance

```

478 # Definition einer Funktion die uns die Performance eines Modells ausgibt
479 def performance_tester(model, features, real_labels):
480     '''A function to compute performance measures of a model on a test set'''
481
482
483
484     # Benötigte Libraries

```

```

485     from sklearn.metrics import accuracy_score, confusion_matrix, recall_score,
        precision_score
486
487     # Predictions
488     pred_labels = model.predict(features)
489
490     # Metriken: Confusion Matrix, Accuracy, Precision, Recall
491     cm = confusion_matrix(real_labels, pred_labels)
492     acc = accuracy_score(real_labels, pred_labels)
493     prec = precision_score(real_labels, pred_labels)
494     rec = recall_score(real_labels, pred_labels)
495
496
497     # Erzeugen eines DataFrames zur Visualisierung der Confusion Matrix
498     df_cm = pd.DataFrame(cm, range(2), range(2))
499     # Visualisierung CM
500     fig = sns.heatmap(df_cm,
501                       fmt='d',
502                       annot=True,
503                       cmap="YlGnBu")
504
505     sns.set(font_scale=1.4)
506     plt.title('Confusion matrix', fontsize = 17)    # Titel + Schriftgroesse
507     plt.xlabel('Predicted', fontsize = 15)          # X-Achse +
        Schriftgroesse
508     plt.ylabel('True', fontsize = 15)              # Y-Achse +
        Schriftgroesse
509     fig_size = plt.rcParams['figure.figsize']
510     fig_size[0] = 5
511     fig_size[1] = 5
512     plt.rcParams["figure.figsize"] = fig_size
513     plt.show()
514
515     # Ausgabe von Accuracy, Precision, Recall
516     print('Model performance:',
517          '\nAccuracy: ', round(acc,3), '\n' + 'Precision: ', round(prec,3), '\n' + '
        Recall: ', round(rec,3))
518
519     # Testen der Performance
520     print('Training data performance:\n')
521     performance_tester(knnmodel, X_train, Y_train)
522     print(2*'\n', '\nTest data performance:\n')
523     performance_tester(knnmodel, X_test, Y_test)

```

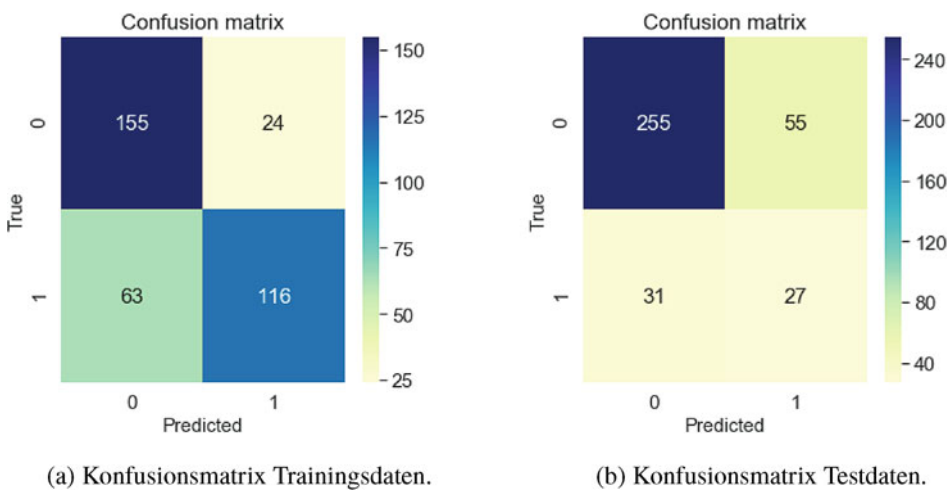


Abb. 14.13 (a) Konfusionsmatrix Trainingsdaten. (b) Konfusionsmatrix Testdaten

Listing 14.45 Konsolenausgabe:

```
1 Training data performance:
2
3 Model performance:
4 Accuracy: 0.757
5 Precision: 0.829
6 Recall: 0.648
7
8
9 Test data performance:
10
11 Model performance:
12 Accuracy: 0.766
13 Precision: 0.329
14 Recall: 0.466
```

Die Ergebnisse geben uns Aufschluss darüber, wie gut unser Modell wirklich ist. Betrachten wir zunächst die Ergebnisse bezüglich der Testdaten. Auf den ersten Blick deutet sich an, dass unser einfaches und noch nicht optimiertes Modell eine recht gute Performance erreicht. Insgesamt klassifiziert das Modell 76,6 %, also mehr als drei-viertel der Observationen im Testdatensatz korrekt. Ein Blick auf die Konfusionsmatrix zeigt sehr schnell, dass das Modell deutlich besser darin ist Observationen der Klasse Attrition_Yes=0 korrekt zu identifizieren. Dazu kommt, (i) dass von allen Vorhersagen dass eine Observation der Klasse Attrition_Yes=1 angehört nur 32,9 % korrekt sind und (ii) von allen in den Testdaten enthaltenen Observationen die das Label Attrition_Yes=1 besitzen weniger als die Hälfte erkannt werden. Vergleichen wir die Performance-Metriken auf den Testdaten mit denen, die wir in Bezug auf die Trainingsdaten erhalten, wird klar, dass wir Overfitting betreiben, da das Modell auf den Trainingsdaten deutlich höhere Precision und Recall Werte (und nur einen marginal kleineren Accuracy Wert) aufweist.

In Anbetracht der Zielstellung dass wir die Abwanderung von Mitarbeitern, also `Attrition_Yes=1` möglichst akkurat vorhersagen möchten, sollten wir versuchen, dass Modell zu verbessern. Ein Blick auf die Kennzahlen bzgl. der Trainingsdaten verrät, dass unser Modell eine deutlich höhere Güte für die Daten besitzt, auf denen es trainiert wurde. Diese Beobachtung deutet darauf hin, dass das Modell unter Overfitting leidet – es kann die in den Trainingsdaten enthaltenen Muster gut erklären, allerdings lassen diese sich nicht verallgemeinern.

Als nächstes werden wir versuchen den Hyperparameter k zu optimieren. Die Optimierung erfolgt dabei stets durch empirisches Testen. In anderen Worten: die funktionale Form unseres Modells wird auf Basis der Daten bestimmt und nicht durch von uns getätigte Annahmen.

Für unser KNN Modell bedeutet dies, dass wir ausprobieren für welche Anzahl k der ähnlichsten Observationen das Modell die beste Performance in Bezug auf unsere Performance-Kennzahlen aufweist. Damit wir im Zuge des austesten verschiedener k Werte kein Overfitting auf den Testdaten betreiben, d. h. k so zu optimieren, dass unser Modell den besten Performance-Wert auf den Testdaten aufweist und die Testdaten damit implizit in den Trainingsprozess miteinzubeziehen, wenden wir das sogenannte Kreuzvalidierungsverfahren (Engl.: *cross validation*) an. Im Rahmen der Kreuzvalidierung werden die Trainingsdaten zufällig, in N (hier $N = 10$) gleich große Teile aufgeteilt. Davon werden $N-1$ (hier $N-1 = 9$) Teile zum Trainieren des Modells verwendet und der übrig gebliebene Teil (Validierungsdatensatz) dient dazu die Performance des trainierten Modells zu testen. Für jeden k -Wert den wir untersuchen, wenden wir dieses Verfahren mehrfach an (hier 15 mal). Dadurch erhalten wir für jeden Wert des Hyperparameters k eine Verteilung von (hier 15) Performance-Werten. Letztlich wählen wir den k -Wert aus, für den die durchschnittliche Performance-Metrik maximal ist. Sobald wir das optimale k identifiziert haben, trainieren wir das Modell anschließend auf Basis aller Trainingsdaten und testen die Generalisierbarkeit erneut anhand der Testdaten, die durch dieses Verfahren nach wie vor zu keinem Zeitpunkt in den Trainingsprozess eingebunden wurden.

An dieser Stelle sei angemerkt, dass die Sklearn Bibliothek verschiedene Methoden zur automatisierten Kreuzvalidierung im Zuge der Hyperparameter Optimierung zur Verfügung stellt. Zum einen kann die *GridSearchCV* Methode verwendet werden, die eine umfassende Suche über spezifizierte Parameterwerte durchführt. Wird eine Vielzahl verschiedener Parameterkombinationen überprüft, bietet sich die *RandomizedSearchCV* Methode an, die eine randomisierte Suche für Hyperparameter durchführt. Beide Methoden verfügen über äußerst eine intuitive API und können einfach verwendet werden. Im Folgenden werden diese beiden Methoden allerdings nicht verwendet. Stattdessen erzeugen wir eine eigene Methode, welche zu dem gleichen Ergebnis führt. Dieses Vorgehen hat den Zweck, dass Schritt für Schritt im Code illustriert und verstanden werden soll, wie die Kreuzvalidierung durchgeführt wird.

Wir testen die Werte für k im Bereich 1 bis 15. Zwar ist das Intervall frei wählbar, allerdings sollte hier beachtet werden, dass wir Overfitting dringend vermeiden wollen. Für unser KNN Modell bedeutet das, dass wir keinen zu hohen k Wert wählen, da eine neue

Observation ansonsten basierend auf einer großen Zahl der im Trainingsdatensatz enthaltenen Datenpunkte klassifiziert wird. Dadurch würde die Wahrscheinlichkeit steigen, dass die Klassifizierung auf Basis von spezifischem statistischen Rauschen erfolgt; also Variationen die nur zufällig in unserem Trainingsdatensatz auftreten und nicht generalisierbar sind.

Das Codebeispiel 14.46 zeigt, wie wir durch ein iteratives Verfahren die Performance-Kennzahlen für verschiedene k Werte berechnen und in die entsprechenden Listen accuracies, precisions und recalls einfügen. Auf Basis der Listen identifizieren wir die entsprechend der Kennzahlen optimalen Werte für k . Mithilfe von Matplotlib generieren wir uns eine grafische Illustration welche in Abhängigkeit des Wertes k die verschiedenen Performance-Kennzahlen anzeigen. Die erzeugte Grafik findet sich in Abb. 14.4.

Listing 14.46 Implementierung der Hyperparameter Optimierung des KNN Modells

```

623 # Hyperparameter Optimierung
624
625 # Liste zur Speicherung der Performance Metriken fuer bestimmte k Werte
626 accuracies = []
627 precisions = []
628 recalls = []
629
630 # aeussere Schleife fuer das Testen der Hyperparameter
631 # Wir testen: k = 1, ..., 15
632 for k in range(1,16):
633
634     # Liste fuer die einzelnen Metriken der Cross-Validierung
635     temp_accs = []
636     temp_precs = []
637     temp_recs = []
638
639     # Cross-Validierung
640     for val in range(0,15):
641         # Zufaelliche Aufteilung der Trainingsdaten
642         X_train_, X_val, Y_train_, Y_val = train_test_split(
643             X_train,
644             Y_train,
645             test_size=0.1,    # 20 % werden als Validierungsdatensatz verwendet
646             random_state=val) # Zufallszahl zur Replikation
647
648         # Modell schaeetzen
649         knnmodel = KNeighborsClassifier(n_neighbors=k).fit(X_train_, Y_train_)
650
651         # Vorhersagen
652         Y_val_pred = knnmodel.predict(X_val)
653
654         # Abhaengig vom Problem koennen unterschiedliche Metriken besser sein
655         acc = accuracy_score(Y_val, Y_val_pred)

```

```
656         prec = precision_score(Y_val, Y_val_pred)
657         rec = recall_score(Y_val, Y_val_pred)
658
659         # Performance Metrik fuer die einzelnen Runde der Validierung
660         temp_accs.append(acc)
661         temp_precs.append(prec)
662         temp_recs.append(rec)
663
664
665         # Berechnung der durchschnittlichen Performance fuer einen bestimmten k Wert
666         # Durchschnitt wird der initialen Liste hinzugefuegt
667         accuracies.append(np.mean(temp_accs))
668         precisions.append(np.mean(temp_precs))
669         recalls.append(np.mean(temp_recs))
670
671     # Performance-Metriken verschiedener k Werte
672     plt.plot(range(1, 16), accuracies)
673     plt.plot(range(1, 16), precisions)
674     plt.plot(range(1, 16), recalls)
675     plt.xlim(1,15)
676     plt.xticks(range(1, 16))
677
678     # Grafik Spezifikationen
679     plt.xlabel('Number of Neighbors k')
680     plt.ylabel('Performance')
681     plt.legend(['Accuracy', 'Precision', 'Recall'])
682     plt.title('Comparison of performance measures')
683     plt.tick_params(direction='out', length=6, width=3, colors='b',
684                    grid_color='b', grid_alpha=0.5)
685     fig_size = plt.rcParams["figure.figsize"]
686     fig_size[0] = 12
687     fig_size[1] = 5
688     plt.rcParams["figure.figsize"] = fig_size
689
690     plt.show()
691
692     # Ausgabe der optimalen k-Werte bezogen auf unterschiedliche Performance Kennzahlen
693     opt_k_acc = np.argmax(accuracies) + 1
694     opt_k_prec = np.argmax(precisions) + 1
695     opt_k_rec = np.argmax(recalls) + 1
696     print('Optimales k Accuracy =', opt_k_acc,
697           '\nOptimales k Precision =', opt_k_prec,
698           '\nOptimales k Recall =', opt_k_rec)
```



Abb. 14.4 Optimaler Hyperparameter des KNN Modells

Listing 14.47 Konsolenausgabe:

```

1 Optimales k Accuracy = 13
2 Optimales k Precision = 12
3 Optimales k Recall = 13

```

Die Illustration zeigt uns, dass für den Wert $k=13$ sowohl die Accuracy- als auch die Recall-Metrik maximiert ist. Das Modell ist bzgl. der Precision-Metrik optimal für den Wert $k=12$. Zu beachten: Die exakten optimalen Werte sollten analytisch über die Listen identifiziert werden. In dem Codebeispiel erfolgt die Identifikation über die `np.argmax()` Methode welche uns die Listenposition der maximalen Werte ausgibt (da Listenpositionen bei 0 anfangen zu zählen, addieren wir die 1).

In unserem Beispielproblem sind wir insbesondere daran interessiert so viele der abwanderungswilligen Mitarbeiter zu erkennen wie möglich, um dann gegebenenfalls Gegenmaßnahmen einleiten zu können, welche die Abwanderung aufhalten sollen. Wir nehmen an dieser Stelle an, dass die inkorrekte Klassifizierung eines Mitarbeiters als abwanderungswillig und darauf basierend eine nicht-notwendige Einleitung von Gegenmaßnahmen keine hohen Kosten erzeugt (wohingegen eine vermeidbare Abwanderung mit enormen Kosten verbunden ist). Daher gewichten wir die Recall-Metrik stärker als die Precision-Metrik und setzen $k=13$ als optimalen Hyperparameter. Würden die Kosten (der Nutzen) einer falschen (richtigen) Klassifizierung der Personen ein anderes Kosten-Nutzen-Verhältnis in Bezug auf die vorliegende Problemstellung aufweisen, könnte eine stärkere Gewichtung der Precision-Metrik sinnvoll sein (z.B. wenn die unnötige Einleitung von Gegenmaßnahmen mit sehr hohen Kosten verbunden ist).

Nun da wir den für unser Problem optimalen Hyperparameter $k=13$ identifiziert haben, trainieren wir das Modell für $k=13$ auf Basis aller Trainingsdaten erneut und testen die

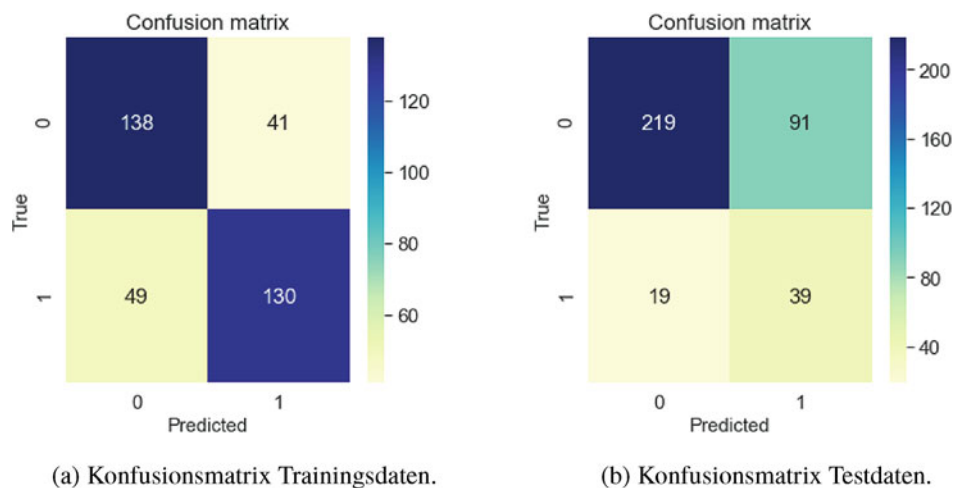


Abb. 14.5 (a) Konfusionsmatrix Trainingsdaten. (b) Konfusionsmatrix Testdaten

Performance auf unseren Testdaten. Die Implementierung und die entsprechenden Ausgaben finden sich in dem nachfolgenden Codebeispiel 14.48 und den Ausgaben 14.49 (Abb. 14.5).

Listing 14.48 Implementierung des optimierten KNN Modells

```

623 # Instanziierung und Training des Algorithmus mit Accuracy optimalem k-Wert
624 knnmodel = KNeighborsClassifier(n_neighbors=opt_k_acc).fit(X_train, Y_train)
625
626 # Testen der Performance
627 print('Training data performance:\n')
628 performance_tester(knnmodel, X_train, Y_train)
629 print(2*\n', '\nTest data performance:\n')
630 performance_tester(knnmodel, X_test, Y_test)

```

Listing 14.49 Konsolenausgabe:

```

1 Training data performance:
2
3 Model performance:
4 Accuracy: 0.749
5 Precision: 0.76
6 Recall: 0.726
7
8
9
10 Test data performance:
11
12 Model performance:
13 Accuracy: 0.701
14 Precision: 0.3
15 Recall: 0.672

```

Wie wir sehen, konnten wir die für uns am meisten relevante Recall-Metrik beachtlich steigern; können entfernen von zuvor 46,6 % auf 67,2 %. Nach der Optimierung ist unser Modell also deutlich besser darin abwanderungswillige Mitarbeiter zu identifizieren. Auf der anderen Seite sehen wir, dass diese Steigerung auf Kosten verringerter Accuracy und Precision

möglich war. Insgesamt konnten wir den Anteil der identifizierten abwanderungswilligen Personen um fast 45 % erhöhen (von zuvor 27 auf jetzt 39 Personen). Abhängig von den Kosten die eine zuvor nicht erkannte Abwanderung eines Mitarbeiters erzeugt, kann diese Veränderung enorme Einsparungen bedeuten.

Das geschätzte Modell könnte nun in Arbeitsprozesse eingebaut werden, um menschliche Entscheidungen zu unterstützen. Beispielsweise könnten HR-Manager das Modell dazu gebrauchen vor einem Mitarbeitergespräch eine Einschätzung zu erhalten, wie wahrscheinlich es ist, dass die Person das Unternehmen verlassen wird. Diese maschinell erzeugte Vorhersage, zusammen mit anderen Faktoren wie bspw. eine Einschätzung über die Produktivität des Mitarbeiters, kann dann als Grundlage zur Entscheidung über die Einleitung entsprechender Gegenmaßnahmen genutzt werden (z. B. eine Lohnerhöhung, eine Beförderung, nichts tun). Dazu kann der HR-Manager einfach die Features des Mitarbeiters, auf denen das Modell trainiert wurde, herausuchen und als Inputfaktoren an das trainierte Modell übergeben.

Damit das Modell nicht zunächst immer wieder von Neuem geschätzt werden muss, was insbesondere bei großen Datenmengen viel Zeit in Anspruch nehmen kann, können wir das trainierte Modell abspeichern und immer wieder laden, sobald es benötigt wird. Eine Bibliothek die uns hierzu Methoden zur Verfügung stellt ist Joblib. Aus der Joblib Bibliothek importieren wir die *dump* und *load* Methoden, die uns respektive das Speichern und Laden unserer Modelle ermöglicht. Die Anwendung der beiden Methoden ist äußerst intuitiv: Wir speichern unsere Modelle, indem wir das Modell-Objekt und den Speicherpfad an die *dump*-Methode übergeben; Wir laden unsere gespeicherten Modelle, indem wir der *load*-Methode den Speicherpfad übergeben, an dem wir unsere Modelle zuvor abgespeichert haben. Das Codebeispiel 14.50 zeigt, wie genau die *load* und *dump* Methoden genutzt werden können, um das zuvor trainierte KNN Modell abzuspeichern.

Listing 14.50 Abpeichern und Laden eines trainierten Modells

```

623 # Abspeichern des trainierten Modells
624 from joblib import dump, load
625 dump(knnmodel, 'PATH/knn.joblib')
626
627 # Laden des trainierten Modells
628 clf = load('PATH/knn.joblib')
```

Mit dem KNN Algorithmus, einem recht einfachen Modell, in Kombination mit einer Optimierung des Hyperparameters *k*, konnten wir ohne größeren Aufwand eine passable Vorhersage Performance erreichen. An dieser Stelle sei angemerkt, dass wir versuchen könnten den KNN Algorithmus weiter zu verbessern. Die API der *KNeighborsClassifier()* Klasse erlaubt es uns beispielsweise die Metrik zur Berechnung der Distanzen zwischen Datenpunkten zu verändern. Standardmäßig wird die euklidische Distanz verwendet (auch als L2 Norm bezeichnet). Eine Anpassung der Metrik (z. B. hin zur Verwendung der absoluten Distanz zwischen Datenpunkten – L1 Norm) hat einen Effekt auf die Widerstandsfähigkeit des Modells gegenüber Ausreißern in einem Datensatz und somit auf die Generalisierbarkeit.

Zum Abschluss dieses Kapitels testen wir nun, ob wir durch die Auswahl eines anderen Algorithmus die Performance weiter steigern können. Wir verwenden den sogenannten *Random Forest Classifier*.

Wie der Name bereits andeutet, setzt sich ein *Random Forest* aus mehreren verschiedenen Entscheidungsbäumen zusammen. Basierend auf dem Trainingsdatensatz lernen einzelne Entscheidungsbäume logische Regeln welche neuen Observationen ein bestimmtes Label zuordnen. Diese Regeln kann man sich als Sequenz aufeinander folgender Fragen vorstellen. Bezogen auf unseren Datensatz könnte ein einzelner Baum bspw. die folgende Fragesequenz identifizieren. 1. Ist die Person älter als 30 Jahre? – Ja; 2. Ist die Person Weiblich? – Nein; 3. Ist der Arbeitsweg der Person länger als 5 KM? – Ja. Ergebnis: Gegeben der Antworten auf die Fragesequenz wird die Person höchstwahrscheinlich abwandern. Während des Trainingsprozesses identifiziert der Algorithmus (mehr oder weniger) automatisch die informativsten Fragen, um Observationen schnellstmöglich zu klassifizieren. Die Mehrheitsklasse die zu einer bestimmten Kombination von Antworten auf eine erlernte Fragesequenz gehört ist letztlich das zugeordnete Label.

Um eine neue Observation zu klassifizieren, wird anhand der erlernten Sequenz von Fragen ein Frage-Antwort-Spiel gespielt. Die neue Observation wird dann basierend auf der Kombination von Antworten klassifiziert. Diese Art logische Regeln abzuleiten und auf neue Observationen anzuwenden, macht einzelne Entscheidungsbäume zu sehr leistungsfähigen Algorithmen, die in der Lage sind, komplexe Datensätze auf Muster zu untersuchen. Random Forests bestehen aus einer Vielzahl unterschiedlicher, zufälliger Entscheidungsbäume. Jeder einzelne Baum nimmt eine Klassifikation vor. Einer neuen Beobachtung wird die Klasse zugeordnet, welche von der Mehrheit der Bäume vorhergesagt wird. In anderen Worten: In dem Wissen dass einzelne Bäume (zufällig) falsch liegen können, verlassen wir uns auf die *Weisheit der Menge*, sodass sich nicht-systematische Fehler einzelner Bäume gegenseitig aufheben.

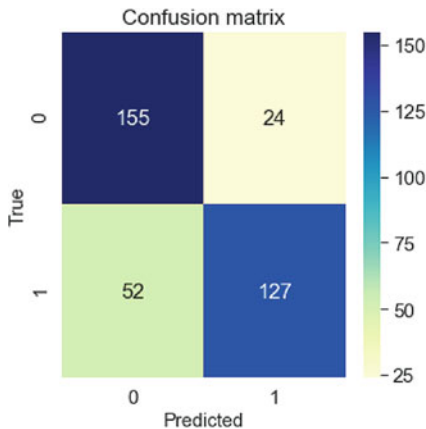
ML Verfahren, wie der Random Forest Algorithmus, bei denen die Klassifikation durch eine Gruppe von einzelnen Modellen erfolgt, werden als *Ensemble Learning Verfahren* bezeichnet. Die Intuition hinter diesen Modellen ist wie bereits angedeutet sehr simpel. Anstelle dass wir ein einziges komplexes Modell fragen, dass an sich mit höherer Wahrscheinlichkeit Overfitting Probleme mit sich bringt, aggregieren wir die Vorhersagen einer Vielzahl von einfachen Modellen und nutzen deren gemeinsames Wissen. Es zeigt sich, dass diese Ensemble Verfahren äußerst hohe Performance aufweisen. Random Forests, obwohl sie nur aus einfachen Entscheidungsbäumen bestehen, gehören zu den leistungsfähigsten ML-Algorithmen, die uns heutzutage zur Verfügung stehen. Einer der wohl größten Vorteile die Random Forests (Entscheidungsbäume) bieten ist, dass sie sehr wenig Datenaufbereitung voraussetzen. Beispielsweise erfordern diese Modelle keine Skalierung der Features anhand der wir das Modell trainieren möchten (trotzdem können skalierte Daten verarbeitet werden!).

Ein weiterer Vorteil besteht darin, dass uns Sklearn, trotz der hohen Leistungsfähigkeit des Modells, eine sehr intuitive Klasse zur Verfügung stellt, mit der wir Random Forests bauen können: `RandomForestClassifier`. Diese Klasse laden und verwenden wir nun im Folgenden.

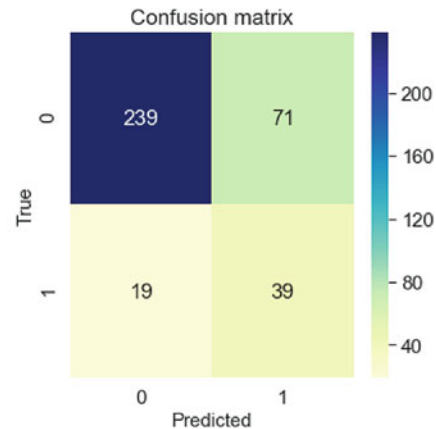
Für das Training des Random Forests nutzen wir unseren bereinigten und reskalierten Datensatz. Das Vorgehen zur Entwicklung des Modells ist analog zu den Schritten die wir für das KNN Modell durchgeführt haben: (i) Import benötigter Methoden, (ii) Partition der Daten in Trainings- und Testdaten, (iii) balancieren der Trainingsdaten bzgl. des Labels, (iv) anfängliches Trainieren und Testen eines naiven Modells, (v) Optimierung der Hyperparameter, (vi) erneutes Trainieren und Testen eines optimierten Modells.

Analog zu dem KNN Modell wenden wir Undersampling an um den Trainingsdatensatz zu balancieren. Undersampling eignet sich für Random Forests tendenziell besser, da diese Art der Algorithmen zu Overfitting neigen kann und Oversampling das Auftreten dieser Problematik durch Vervielfältigung einzelner Datenpunkte noch verschlimmern kann.

Zunächst trainieren wir einen Random Forest der aus 10 ($=n_estimators$) einzelnen Entscheidungsbäumen besteht, die jeweils eine Tiefe – die Länge der Fragensequenz – von 3 ($=max_depth$) besitzen. Die Werte dieser Hyperparameter sind zufällig ausgewählt. Das geschätzte Modell testen wir durch die Anwendung unserer zuvor definierten `performance_tester` Funktion. Das Codebeispiel 14.51 zeigt die Implementierung dieser ersten fünf Schritte. Abb. 14.6b und Konsolenausgabe 14.52 zeigen die erzeugten Ausgaben.



(a) Konfusionsmatrix Trainingsdaten.



(b) Konfusionsmatrix Testdaten.

Abb. 14.6 (a) Konfusionsmatrix Trainingsdaten. (b) Konfusionsmatrix Testdaten

Listing 14.51 Implementierung eines naiven Random Forest Modells

```

623 # Import des RandomForest Classifiers von Scikit-Learn
624 from sklearn.ensemble import RandomForestClassifier
625
626 # Aufteilung der Daten in Trainings- und Testdaten
627 X_train, X_test, Y_train, Y_test = train_test_split(
628     data_hr_rescaled.drop(['Attrition_Yes', 'Attrition_No'], axis=1), #
        Trainingsdaten ausser binaere Labels
629     data_hr_rescaled['Attrition_Yes'],                                # Binaeres
        Label unseres Interesses
630     test_size=0.25,                                                # Groesse des
        Testdatensatzes
631     random_state=0)
632
633 # Undersampling
634 balanced = balancing('Attrition_Yes', pd.concat([X_train, Y_train], axis=1))
635
636 # Separierung von Labels und Features
637 X_train = balanced.drop(['Attrition_Yes'], axis=1)
638 Y_train = balanced['Attrition_Yes']
639
640 # Training eines Modells mit zufaelligen Hyperparametern
641 random_forest_model = RandomForestClassifier(
642     max_depth=3,                                                    # Maximale Tiefe einzelner Baeume
643     n_estimators=10,                                                # Anzahl einzelner
        Klassifikationsbaeume
644 ).fit(X_train, Y_train)
645
646 # Testen der Performance
647 print('Training data performance:\n')
648 performance_tester(random_forest_model, X_train, Y_train)
649 print(2*'\n', '\nTest data performance:\n')
650 performance_tester(random_forest_model, X_test, Y_test)

```

Listing 14.52 Konsolenausgabe:

```

1  Training data performance:
2
3  Performance des Modells:
4  Accuracy: 0.788
5  Precision: 0.841
6  Recall: 0.709
7
8
9
10 Test data performance:
11
12 Performance des Modells:
13 Accuracy: 0.755
14 Precision: 0.355
15 Recall: 0.672

```

Wie wir sehen können, konnten wir auf dem Testdatensatz mit einer zufälligen Auswahl der Hyperparameter auf Anhieb eine Accuracy von 75,5 %, einen Recall von 67,2 % und eine Precision von 35,5 % erreichen. In anderen Worten: ohne eine Optimierung angewendet

zu haben erreicht unser Random Forest eine höhere Performance bzgl. der Accuracy und des Recalls, bei gleichem Recall-Wert. Ein Blick auf die Performance-Metriken für die Trainingsdaten suggeriert erneut, dass das trainierte Modell deutlich besser darin ist Personen des Trainingsdatensatzes korrekt zu klassifizieren. Es zeigt sich daher, dass wir auch hier zu einem gewissen Grad Overfitting betreiben.

Wie an vorheriger Stelle werden wir auch bei dem Random Forest versuchen die Performance zu erhöhen, indem wir die Hyperparameter kalibrieren. Der einzige Unterschied besteht darin, dass wir nun 2 Hyperparameter (die Anzahl der Bäume und die Tiefe der einzelnen Bäume) durch empirisches Experimentieren anhand der Kreuzvalidierungsmethode optimieren. Wie in dem Codebeispiel 14.53 gezeigt, nutzen wir eine doppelte Schleife, um die für unser Problem optimalen Hyperparameter zu identifizieren. Wir testen alle möglichen Kombinationen der Hyperparameter `n_estimators = 1,2,3,...,7` und `max_depth=10,20,30,...,100`. Die Konsolenausgabe 14.54 zeigt das Ergebnis.

Listing 14.53 Optimierung der Hyperparameter des Random Forests

```

623 # Hyperparameter Optimierung
624 # Variation der maximal moeglichen Tiefe der einzelnen Baeume
625 # Array von 3 Zahlen die im gleichen Abstand zwischen 2 und 7 liegen:
626 # 2,3,4,5,6,7
627 max_depth = np.linspace(2, 7, 6)
628
629 # Anzahl der Baeume im Wald
630 # Array von 0 bis N-1, jeweils um 1 erhoeht und mit 10 multipliziert:
631 # 10,20,30,40,50,60,70, 90, 100
632 n_trees = (np.arange(10)+1)*10
633
634 # Accuracies bestimmter Kombinationen
635 accuracies = []
636 precisions = []
637 recalls = []
638
639 # Schleife ueber die Tiefe der Baeume
640 for d in max_depth:
641     # Schleife ueber die Anzahl der Baeume
642     for n in n_trees:
643
644         # Accuracies einzelner Validierungen
645         temp_accs = []
646         temp_precs = []
647         temp_recs = []
648
649         for val in range(15):
650
651             # Zufaelliche Aufteilung der Trainingsdaten
652             X_train_, X_val, Y_train_, Y_val = train_test_split(
653                 X_train,
```

```

654         Y_train,
655         test_size=0.2,    # 20 % werden als Validierungsdatensatz verwendet
656         random_state=val) # Zufallszahl zur Replikation
657
658     # Modell schaeetzen
659     random_forest_model = RandomForestClassifier(
660         max_depth=d,
661         n_estimators=n).fit(X_train_, Y_train_)
662
663     # Vorhersagen
664     Y_val_pred = random_forest_model.predict(X_val)
665
666     # Performance Metrik, hier: Accuracy
667     # Abhaengig vom Problem koennen unterschiedliche Metriken besser sein
668     acc = accuracy_score(Y_val, Y_val_pred)
669     prec = precision_score(Y_val, Y_val_pred)
670     rec = recall_score(Y_val, Y_val_pred)
671
672     # Performance Metrik fuer die einzelnen Runde der Validierung
673     temp_accs.append(acc)
674     temp_precs.append(prec)
675     temp_recs.append(rec)
676
677     # Zufuegen des Durchschnittswertes und der n,d Kombination zu der Liste
678     accuracies.append([np.mean(temp_accs), d, n])
679     precisions.append([np.mean(temp_precs), d, n])
680     recalls.append([np.mean(temp_recs), d, n])
681
682     # Identifikation der optimalen Werte
683     temp_acc = []
684     temp_rec = []
685     temp_prec = []
686
687     for i in range(len(accuracies)):
688         temp_acc.append(accuracies[i][0])
689         temp_rec.append(recalls[i][0])
690         temp_prec.append(precisions[i][0])
691     opt_acc = np.argmax(temp_acc)
692     opt_rec = np.argmax(temp_rec)
693     opt_prec = np.argmax(temp_prec)
694
695     d_opt_acc = accuracies[opt_acc][1]
696     n_opt_acc = accuracies[opt_acc][2]
697
698     d_opt_rec = accuracies[opt_rec][1]
699     n_opt_rec = accuracies[opt_rec][2]
700
701     d_opt_prec = accuracies[opt_prec][1]
702     n_opt_prec = accuracies[opt_prec][2]

```

```

703
704 print('\nAccuracy:', '\nOptimal depth of trees: ', d_opt_acc, '\nOptimal number of
    trees: ', n_opt_acc)
705 print('\nRecall:', '\nOptimal depth of trees: ', d_opt_rec, '\nOptimal number of
    trees: ', n_opt_rec)
706 print('\nPrecision:', '\nOptimal depth of trees: ', d_opt_prec, '\nOptimal number of
    trees: ', n_opt_prec)

```

Listing 14.54 Konsolenausgabe:

```

1 Accuracy:
2 Optimal depth of trees: 6.0
3 Optimal number of trees: 100
4
5 Recall:
6 Optimal depth of trees: 6.0
7 Optimal number of trees: 70
8
9 Precision:
10 Optimal depth of trees: 6.0
11 Optimal number of trees: 100

```

Es stellt sich heraus, dass `n_estimators=70` und `max_depth=6` die optimalen Hyperparameter bezüglich der Recall-Metrik sind. Dementsprechend sollte unser Wald aus 30 einzelnen Entscheidungsbäumen bestehen, die jeweils eine Tiefe von 7 besitzen. Diese Werte nutzen wir, um auf Basis aller Trainingsdaten ein neues Random Forest Modell zu schätzen, welches anschließend auf den Testdaten hinsichtlich dessen Generalisierbarkeit überprüft wird. Dazu nutzen wir erneut unsere zuvor spezifizierte `performance_tester` Funktion. Die erzeugten Ergebnisse finden sich in Abb. 14.7a und Konsolenausgabe 14.56.

Listing 14.55 Implementierung des optimierten Modells

```

623 # Training mit optimierten Hyperparametern
624 random_forest_model = RandomForestClassifier(
625     max_depth=d_opt_rec,
626     n_estimators=n_opt_rec,
627 ).fit(X_train, Y_train)
628
629 # Testen der Performance
630 print('Training data performance:\n')
631 performance_tester(random_forest_model, X_train, Y_train)
632 print(2*'\n', '\nTest data performance:\n')
633 performance_tester(random_forest_model, X_test, Y_test)

```

Listing 14.56 Konsolenausgabe:

```

1 Training data performance:
2
3 Model performance:
4 Accuracy: 0.95
5 Precision: 0.976
6 Recall: 0.922
7
8
9
10 Test data performance:
11
12 Model performance:
13 Accuracy: 0.774
14 Precision: 0.381
15 Recall: 0.69

```

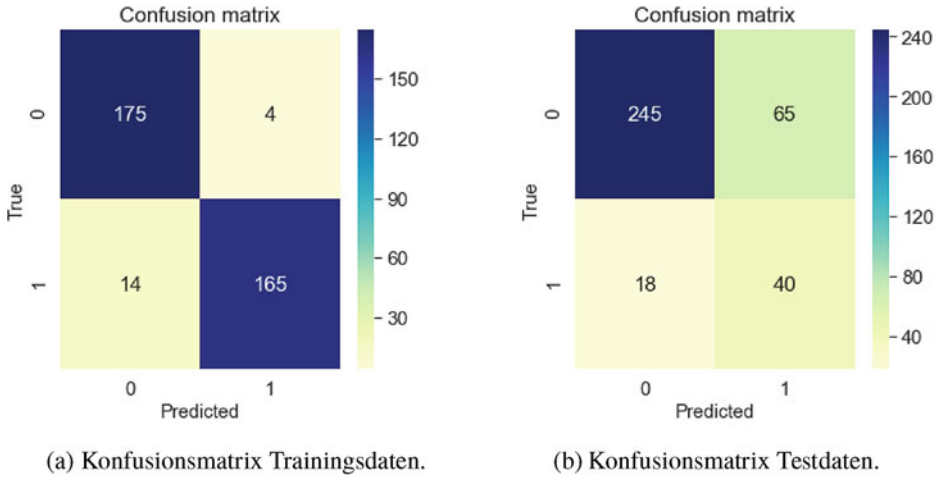


Abb. 14.7 (a) Konfusionsmatrix Trainingsdaten. (b) Konfusionsmatrix Trainingsdaten

Die ausgegebenen Ergebnisse zeigen, dass wir die Leistung unseres Modells in Bezug auf alle drei Performance-Metriken weiter steigern konnten, indem wir die empirisch optimierten Hyperparameter Werte verwenden. Müssten wir uns entscheiden eines unserer geschätzten Modelle in Betrieb zu nehmen, würden wir uns für dieses optimierte Random Forest Modell entscheiden.

Abschließend soll noch auf eine äußerst nützliche Eigenschaft von Random Forests hingewiesen werden, welche sich auf eine häufig angeführte Problematik bei der Anwendung von ML-Algorithmen bezieht.

Wie im Laufe dieses Kapitels erklärt, wird die Mehrheit heutiger ML-Algorithmen dazu verwendet, Vorhersagen über die Ausprägung von Labels (abhängige Variable) zu erzeugen. Dazu werden Kombinationen von Features (unabhängigen Variablen) verwendet. Es ist wichtig zu verstehen, dass es bei der Erzeugung der Vorhersagen nicht darum geht den isolierten Einfluss einer bestimmten unabhängigen Variable auf die abhängige Variable zu identifizieren (inference), sondern auf Basis der Gesamtheit der unabhängigen Variablen die abhängige Variable möglichst akkurat vorherzusagen (prediction). Vor diesem Hintergrund wird bei vielen ML-Algorithmen wenig Zeit darauf verwendet zu untersuchen, wie einzelne unabhängige Variablen die Vorhersage beeinflussen. Häufig ist dies auch ein schweres Unterfangen, welches nur approximativ umgesetzt werden kann (siehe z. B. Shapley-Values oder LIME bei Neuronalen Netzen). Dies ist der Grund warum einige ML-Algorithmen wiederholt unter der Kritik stehen Black-Boxen zu sein, deren innere Funktionsweise intransparent ist und nicht ausreichend nachvollzogen werden kann.

Vor dem Hintergrund dass es bei vielen Entscheidungen von hoher Bedeutung ist zu verstehen, und erklären zu können, warum ein ML Modell eine bestimmte Vorhersage erzeugt hat, benötigen wir Werkzeuge die uns einen Einblick die die interne Funkti-

onsweise der Algorithmen ermöglichen. Beispielsweise sollte man in der Lage sein zu erklären, warum bestimmte Bewerber in einem Einstellungsprozess durch Algorithmen abgelehnt wurden. Bei Random Forests stellt das automatisch erzeugte Objekt `model.feature_importances_` ein solches Werkzeug dar.

Feature Importances geben an, wie wichtig ein Attribut bei der Klassifizierung einer Observation ist. Technisch betrachtet ist der Importance-Wert eines einzelnen Merkmals der durchschnittliche Informationsgewinn (über alle Bäume im Wald hinweg) wenn das Merkmal als Kriterium zur Aufteilung eines gegebenen Datensatzes in Teilmengen genutzt wird. Ein relativ hoher Importance-Wert gibt an, dass ein bestimmtes Merkmal eine größere Rolle bei der Klassifizierung spielt als Merkmale deren Importance-Wert klein ist.

Die Importance-Werte eines jeden Merkmals wird bei der Schätzung eines Random Forest Modells stets automatisch mitberechnet und kann durch `model.feature_importances_` schnell und einfach aufgerufen werden. Es sei angemerkt, dass die Importance-Werte verzerrt sein können, wenn unterschiedliche Merkmale große Unterschiede in der Anzahl möglicher Ausprägungen aufweisen. Dies liegt kurz gesagt daran, dass Informationsgewinne von Merkmalen, die eine relativ hohe Anzahl möglicher Ausprägungen besitzen, tendenziell überschätzt werden. An dieser Stelle werden wir davon absehen auf diese Problematik weiter einzugehen.

Das Codebeispiel 14.57 illustriert die Vorgehensweise zur Ausgabe und grafischen Illustration der Importance-Werte. Zunächst weisen wir der Variable `feature_importances` die mit der Berechnung unseres optimierten Modells erzeugten Importance-Werte zu. Wir nutzen dafür ein Series Objekt – einen Spaltenvektor – dessen Index den Variablennamen entspricht. Anschließend plotten wir die 10 Variablen, welche (laut der Importance-Werte) den größten Einfluss bei der Klassifikation besitzen. Die erzeugte Ausgabe ist in Abb. 14.8 zu finden.



Abb. 14.8 Feature importances

Listing 14.57 Implementierung der Ausgabe der Feature Importances

```
623 # Zuweisung der Importance-Werte als Spaltenvektor
624 feature_importances = pd.Series(random_forest_model.feature_importances_, index =
    X_train.columns)
625 # Grafische Illustration der 10 einflussreichsten Merkmale
626 feature_importances.nlargest(10).plot(kind='barh')
627 plt.show()
```

Es stellt sich heraus, dass bei unserem optimierten Modell die Variablen *YearsAtCompany* und *MonthlyIncome* relativ gesehen den größten Einfluss bei der Klassifizierung von abwanderungswilligen Mitarbeitern haben. Diese Information hilft uns nicht nur dabei besser zu verstehen wie bestimmte Vorhersagen zustande gekommen sind. Sie sondern könnte auch als Grundlage dienen, die interne Organisationsstruktur zu modifizieren, sodass die Abwanderung bestimmter Mitarbeiter präventiv verringert wird.



Anwendungsbeispiel: Get Your Things Done – Modernes Zeitmanagement

15

Benjamin M. Abdel-Karim

Für die Berufsgruppen im Dienstleistungssektor, wie beispielsweise Unternehmensberater, Juristen oder Programmierer, ist es von zentraler Bedeutung, ihre Leistungen minutengenau zu erfassen und transparent für sich und ihre Partner zu dokumentieren. Allerdings kann die genau Erfassung der genutzten Zeit für spezifische Aufgaben aufwendig sein. Zudem reichen simple Lösungen wie Notizen oder Stempelkartensysteme für eine effiziente und kundenorientierte Lösung nicht aus, um tiefere Einblicke in die genutzte Zeit zu erhalten und Synergieeffekte zu identifizieren. Das Ziel dieses Buchkapitels ist es, mithilfe von Wearables und modernen Analysemethoden einige Inspiration zu liefern, wie sich die moderne Zeiterfassung in Unternehmen realisieren lässt. In einem ersten Schritt wird das entsprechende Wearable vorgestellt, um daran die Datenanalyse entlang des Data-Science-Prozesses in einem zweiten Schritt zu vollziehen.

15.1 Zeitmanagement: Datensatz und Fragestellung

Mithilfe moderner Geräte (Wearables) lassen sich Daten im Alltag erfassen. Smart Watches oder Fitnessarmbänder sind heute beliebte Begleiter. Wearables sind Geräte, die zu einem bestimmten Zweck konzipiert wurden und sich dadurch auszeichnen, dass sie nah am Körper bzw. in der Umgebung des Nutzers verwendet werden. Für das Themenfeld der Zeiterfassung gibt es einige Anbieter, die den Versuch wagen, eine intuitive Erfassung der Zeit für beliebige Aktivitäten zu ermöglichen.

B. M. Abdel-Karim (✉)
Frankfurt am Main, Deutschland
E-mail: BenjaminM.Abel-Karim@gmx.de

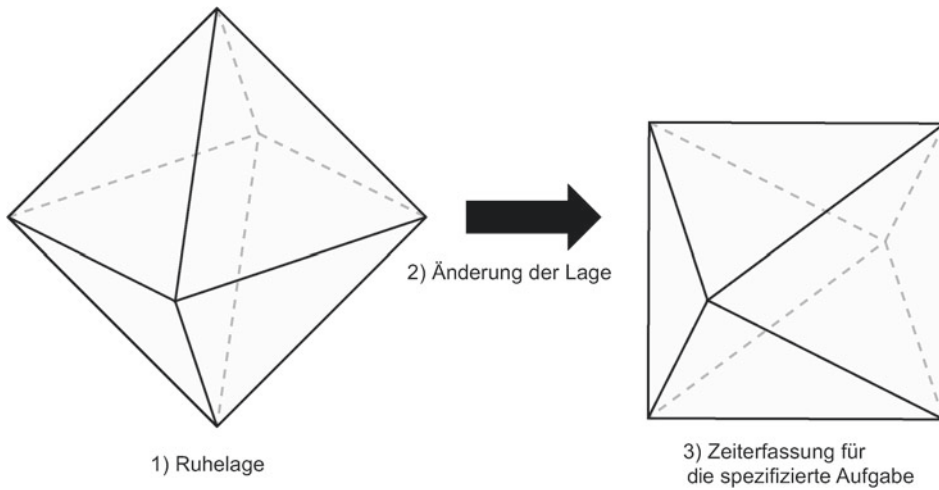


Abb. 15.1 Grundfunktionalität des Time Tracker Wearable

Für dieses Buch wird exemplarisch das Time Tracker Wearable der Firma Timeular¹ verwendet. Hierbei handelt es sich um ein kleines Gerät mit Lagesensor in der Form eines Oktaeders. Die Seiten des Oktaeders können beliebig beschriftet werden, sodass jede Seite eine spezifische Aufgabe zugewiesen bekommt (z. B. E-Mails schreiben). Sobald der Nutzer das Gerät aus seiner Station nimmt und so auf den Schreibtisch legt und die Fläche mit einer gewünschten Aufgabe oben liegt, beginnt die Zeiterfassung der jeweilige Aufgaben. Durch die Form des Wearable ergeben sich acht Flächen für jeweils acht Aufgaben. Diese Aufgaben können mithilfe der bereitgestellten Software des Herstellers beliebig konfiguriert werden. Die Abb. 15.1 zeigt die Grundfunktionalität des Time Tracker Wearable.

Die Abb. 15.1 zeigt das Time Tracker Wearable zunächst in seiner Ruhelage (1 in Abb. 15.1). Sofern der Nutzer die Lage des Time Tracker verändert (2 in Abb. 15.1), registriert der Lagesensor im Inneren diese Änderung und löst die Zeiterfassung für die entsprechende Aufgabe (3 in Abb. 15.1) aus. Durch die vorherige Konfiguration mithilfe der Software² wird jeder Seite des Oktaeders eine spezifische Aufgabe zugeordnet. Bei Veränderung der Lage ist die Software in der Lage, durch die Lageinformationen zu ermitteln, welche Aufgabe ausgeführt wird. Solche Objekte werden auch als Tangible User Interface (TUI) bezeichnet, also Objekte, die durch ihre physikalischen Erscheinungen und die Möglichkeit der Manipulation der physikalischen Eigenschaft, eine digitale Übersetzung ermöglichen. Die Integration von Design und Funktionalität soll eine schnelle und präzise Zeiterfassung ermöglichen.

Im Anwendungsfall des Buchs hat also jeder Mitarbeiter die Möglichkeit, acht typische Aufgaben zu definieren und mithilfe des Wearable zu erfassen. Für dieses Fallbeispiel

¹ <https://timeular.com>, zuletzt abgerufen am 1. Dezember 2020.

² <https://timeular.com/product/pro-subscription/>, zuletzt abgerufen am 1. Dezember 2020.

kommt damit ein für dieses Buch erstellter Datensatz zum Einsatz. Die Daten wurden hierbei mithilfe des Time Tracker Wearable³ erfasst und über die Software des Herstellers als .CSV-Datei exportiert. Im Zentrum dieses Kapitels steht die Datenanalyse der Zeiterfassung. Zum besseren Verständnis des Anwendungsbeispiels wird für dieses Buch angenommen, dass die Zeiterfassung von einer Mitarbeiterin aus einer klassischen Beratungsgesellschaft erstellt worden ist. Im Folgenden wird diese Mitarbeiterin Marlene genannt. Marlene arbeitet für die Beratungsgesellschaft als Data Scientist. Dabei entwickelt sie als Spezialistin mithilfe von modernen Methoden von Data Science Lösungen für die Herausforderungen der Klienten der Beratungsgesellschaft. Ein zentraler Bestandteil ihrer Stellenbeschreibung ist die Modellierung und Implementierung von Prototypen. Dabei arbeitet sie mit interdisziplinären Teams, Klienten, Kunden, Mitarbeitern verschiedener Fachabteilungen und Dienstleistern zusammen. Außerdem umfasst ihre Stelle Führungs- und Administrationsaufgaben. Sicherlich ist die Systematisierung von Aufgaben in Kategorien bei einem vielseitigen Beruf kein triviales Unterfangen, sodass sich zahlreiche Literatur mit dieser Thematik befasst. Für einen ersten Systematisierungsversuch hat sich Marlene an den Herstellerempfehlungen des Time Tracker Wearable orientiert⁴, sodass die folgende Aufzählung als Spezifikationsliste zur Konfiguration der Seitenflächen des Oktaeders dienen kann:

- Administrative Aufgaben (Administrative)
- Beratung (Consulting)
- Führungsaufgaben (Feedback)
- Implementierung (Development)
- Design und Konzeption (Design)
- Meetings
- Weiterbildung (Learning)
- Mittagspause (Lunch)

Administrative Aufgaben umfassen alle internen Verwaltungsaufgaben, wie beispielsweise das Verfassen von E-Mails an ihre Mitarbeiter oder Vorgesetzten. Die Beratungstätigkeit umfasst Gespräche mit Kunden, Präsentationen und alle kundenbezogenen Gespräche im Kontext des jeweiligen Klientenprojekts. Implementierungen beziehen sich auf kundenbezogene Aufgaben hinsichtlich der Entwicklung von Prototypen. Damit einher gehen auch Tätigkeiten in Bezug auf Design und Konzeption. Die obligatorischen Meetings sind interne Aktivitäten. Im Zuge ihres Arbeitsvertrags hat Marlene Anspruch auf Weiterbildung, die sie sich selbst aussuchen kann, sodass sie Zeit für ihre persönliche Weiterentwicklung aufwendet. Vor diesem Hintergrund ergibt sich ein vielseitiges Berufsbild mit unterschiedlichen

³ Das hier verwendete Gerät dient als Erfassungshilfe. Allerdings gibt es auf dem Zeiterfassungsmarkt zahlreiche Anbieter mit unterschiedlichen Alternativen, sodass der interessierte Leser aus einer Fülle von Alternativen die passende Lösung für sich wählen kann.

⁴ <https://support.timeular.com/en/articles/2519222-what-should-i-track>, zuletzt abgerufen am 1. Dezember 2020.

Aufgaben. Dieser Umstand führt jedoch zu einem herausfordernden Grad an Komplexität, um die aufgewendete Zeit für diese einzelnen Aufgaben akkurat zu erfassen. Für die Unternehmensberatung ist eine genaue Zeiterfassung essenziell, um die Kosten entsprechend der Kostenstellen zuweisen zu können und damit also auch die Einteilung in interne und externe Kosten. Die Klienten haben dabei ein besonderes Interesse, dass die externen Kosten präzise erfasst werden, weil sie unmittelbar durch die Abrechnung dafür bezahlen müssen. Damit haben also diejenigen Beratungsfirmen im Wettbewerb um die Klienten einen Vorteil, die imstande sind, die Mitarbeiterkosten transparent in Rechnung zu stellen. Zudem hat die präzise Erfassung der Arbeitszeit entsprechend der Aufgaben für die Mitarbeiter einen Vorteil, indem sie Projekte effizient managen können. Im Rahmen dieses Anwendungsbeispiels hat Marlene sich dazu entschlossen, ihre Aufgaben ein Jahr lang mithilfe des Time Tracker zu protokollieren und stellt die Daten für dieses Buch zur Verfügung. Ausgehend von dem Anwendungsbeispiel ergeben sich mögliche Fragen für die Analyse des Datensatzes:

- Welcher Anteil entfällt auf die einzelnen Aufgaben?
- Wie lange wird ein Aufgabenbereich durchschnittlich bearbeitet?
- Wie lassen sich die Stunden automatisiert erfassen und abrechnen?
- Gibt es Muster in der Bearbeitung der Aufgaben, aus denen sich der Zeitbedarf für bestimmte Aufgaben vorhersagen lässt?

Auf Basis dieser Fragestellungen wird der Datensatz im Folgenden genauer vorgestellt und das Pre-processing für eine erste Analyse durchgeführt.

15.2 Zeitmanagement: Pre-processing

Für die weiteren Prozessschritte wird das `pandas`-Modul verwendet. Die Daten werden aus der `.CSV`-Datei in ein `DataFrame` geladen und über den Befehl `df.columns.tolist()` werden die Titel der einzelnen Spalten auf der Konsole ausgegeben. Zusätzlich wird mit der `shape`-Anweisung die Dimension des `DataFrame` auf der Konsole ausgegeben, um die Anzahl der Spalten und Zeilen anzuzeigen. Das Vorgehen zeigt das Quellcodebeispiel [15.1](#).

Listing 15.1 Datenimport

```
1 # Code Get Your Things Done
2 # @author: Benjamin M. Abdel-Karim
3 # @since: 2020-06-28
4 # @version: 1.0
5 import pandas as pd
6 import seaborn as sns
7 import matplotlib.pyplot as plt
8
9 df = pd.read_csv('report.csv', sep=',')
10 print(df.columns.tolist())
```

Die Ausführung des Quellcodebeispiels 15.1 zeigt die Konsolenausgabe 15.2.

Listing 15.2 Konsolenausgabe: `df.columns.tolist()` und `df.shape()`

```
1 [ 'TimeEntryID', 'StartDate', 'StartTime', 'StartTimeOffset', 'EndDate',  
   'EndTime', 'EndTimeOffset', 'Duration', 'ActivityID', 'Activity', '  
   SpaceId', 'Space', 'Username', 'Note', 'Mentions', 'Tags' ]  
2 (4380, 16)
```

Die Konsolenausgabe 15.2 zeigt, dass der zur Verfügung gestellte Datensatz aus 16 Spalten mit 4380 Zeilen besteht. Die einzelnen Spalten des Datensatzes lassen sich wie folgt zusammenfassen:

- 'TimeEntryID' = Eine ID, die der Hersteller setzt
- 'StartDate' = Das Startdatum der erfassten Aktivität
- 'StartTime' = Die Startzeit der erfassten Aktivität
- 'StartTimeOffset' = Korrekturfaktor für die Erfassung der Zeitzone
- 'EndDate' = Das Enddatum der erfassten Aktivität
- 'EndTime' = Die Endzeit der erfassten Aktivität
- 'EndTimeOffset' = Korrekturfaktor für die Erfassung der Zeitzone
- 'Duration' = Dauer der erfassten Aktivität
- 'ActivityID' = Eine ID, die der Hersteller verwendet
- 'Activity' = Die Bezeichnung der Aktivität
- 'SpaceId' = Eine ID für den sogenannten Space
- 'Space' = Bezeichnung des Space
- 'Username' = Der Benutzername
- 'Note' = Hinterlegte Notizen vom Nutzer in der Software für die jeweilige Aktivität
- 'Mentions' = Eine Variable, die der Hersteller verwendet, um weitere Features für die Software zu realisieren
- 'Tags' = Vom Nutzer vergebene Tags für einzelne Aktivitäten

Durch das Setzen eines Breakpoint bei der letzten Anweisung und die Ausführung über den Variableninspektor wird im Date-Frame-Objekt ersichtlich, dass die Spalten 'Note', 'Mentions' und 'Tags' mit NaN belegt sind. Durch den Import der Daten wurden die leeren Einträge durch die NaN-Markierung gekennzeichnet. Vor diesem Hintergrund sind diese Spalten für das weitere Vorgehen zu vernachlässigen und können aus dem DataFrame entfernt werden. Ebenso sind für die oben angeführten Fragestellung die vom Hersteller gesetzten ID irrelevant. Außerdem bietet der Hersteller die Möglichkeit an, ein Oktaeder für mehrere Bereiche (sogenannte Spaces) zu konfigurieren. Damit hat der Nutzer beispielsweise die Möglichkeit, den Oktaeder für das berufliche Umfeld und für ein Hobby zu konfigurieren. Die Erfassung der unterschiedlichen Konfiguration erfolgt über die Spalten 'SpaceId' und 'Space'. Für den Anwendungsfall in diesem Buch kommt ausschließlich ein Space zum Einsatz. Damit können die Spalten 'TimeEntryID', 'ActivityID',

'SpaceId' ebenfalls aus dem Datensatz entfernt werden. Das Entfernen von irrelevanten Spalten hat den Vorteil der Minimierung des Speicherbedarfs und damit die Reduktion von Rechenkapazitäten. Jede Spalte im Date Frame erzeugt Verwaltungsaufwand, auch dann, wenn Spalten nicht unmittelbar verwendet werden. Daher kann es Sinn ergeben, insbesondere bei großen Datensätzen frühzeitig Spalten zu entfernen. Die Entfernung von irrelevanten Spalten zeigt das Quellcodebeispiel 15.3.

Listing 15.3 Spalten entfernen aus einem DataFrame

```
13 # -----  
14 # ----- Pre-processing -----  
15 # -----  
16 # Alle unnoetigen Zeilen entfernen.  
17 df = df.drop(['TimeEntryID', 'ActivityID', 'SpaceId', 'Space',  
18             'Username', 'Note', 'Mentions', 'Tags'], axis=1)
```

Im Quellcodebeispiel 15.3 werden mithilfe der `drop(...)`-Anweisung die entsprechenden Spalten aus dem DataFrame entfernt. `drop(...)` erwartet hierbei die Bezeichnungen der Spaltennamen als Liste und entfernt diese aus dem DataFrame. Das Ergebnis wird im Quellcodebeispiel in die Variable `df` geschrieben, sodass das alte Date Frame, mit dem bereinigte Daten Frame überschrieben wird.

Nachdem die Spalten entfernt worden sind, scheint es sinnvoll, das neue DataFrame zu betrachten. Dies geschieht mithilfe der Date-Frame-Funktion `info()`. Durch den Aufruf von `print(df.info())` wird auf der Konsole eine aktualisierte Beschreibung des DataFrame erzeugt und angezeigt. Das Ergebnis zeigt die Konsolenausgabe 15.4.

Listing 15.4 Konsolenausgabe: `print(df.info())`

```
1 <class 'pandas.core.frame.DataFrame'>  
2 RangeIndex: 4035 entries, 0 to 4034  
3 Data columns (total 8 columns):  
4 #   Column           Non-Null Count  Dtype  
5 ---  ---  
6 0   StartDate        4035 non-null   object  
7 1   StartTime        4035 non-null   object  
8 2   StartTimeOffset  4035 non-null   int64  
9 3   EndDate          4035 non-null   object  
10 4   EndTime          4035 non-null   object  
11 5   EndTimeOffset    4035 non-null   int64  
12 6   Duration         4035 non-null   object  
13 7   Activity         4035 non-null   object  
14 dtypes: int64(2), object(6)  
15 memory usage: 252.3+ KB
```

Die Konsolenausgabe 15.4 zeigt, dass nach der Bereinigung der Daten acht Spalten (0–7) mit den zentralen Informationen übrig bleiben. Zwei von den acht Spalten sind als Integer (int64) gespeichert. Das sind die beiden Spalten, in dem der Zeitverschiebungsfaktor mit einer Zahl

(hier 200 für 2h) codiert ist. Die anderen Spalten werden als `object`-Datentyp angezeigt. Das `DataFrame` zeigt Strings oder gemischte Datentypen als `object`-Datentypen an⁵. Der tiefere Blick über den Variableninspektor zeigt, dass es sich um Strings handelt. Hierbei werden sowohl die Zeitdaten als String gespeichert als auch die Daten über die Aktivität. Ausgehend von dieser Erkenntnis scheint es sinnvoll, die Daten des Strings in einen anderen Datentyp zu typcasten, um numerische Operationen, entsprechend der Fragestellungen, durchführen zu können. Für die Umwandlung der String-codierten Zeitinformationen eignet sich die Verwendung eines speziellen Datenobjekts.

Das `datetime`-Modul stellt Funktionen zur Manipulation von Datumsangaben und Uhrzeiten zur Verfügung. Im Zentrum dieses Moduls steht die Extraktion von Zeitinformationen und deren Manipulation im Fokus. Eine ausführliche Beschreibung für dieses Modul liefert die Modul Webseite⁶. Der Vorteil in der Verwendung des `datetime`-Moduls besteht darin, dass es mit dem `Date-Frame`-Objekt als Datenstruktur kompatibel ist. Vor diesem Hintergrund bietet das `Pandas`-Modul zahlreiche Funktionen an, um mit `Datetime`-Objekten zu interagieren. Für dieses Anwendungsbeispiel besteht das nächste Zwischenziel darin, die String-codierten Zeitangaben in `Datetime`-Objekte umzuwandeln, sodass diese entsprechend analysiert werden können. Das Quellcodebeispiel 15.5 zeigt die spaltenweise Umwandlung von Strings im `Date Frame` zu `Datetime`-Objekten.

Listing 15.5 Von Strings zu `Datetime`-Objekten

```

23  # Umwandlung von Zeilenwerten in datetime format.
24  df[ 'StartDate' ] = pd.to_datetime(df[ 'StartDate' ])
25  df[ 'StartDate' ] = pd.to_datetime(df[ 'StartDate' ], format='%d/%m/%Y')
26
27  df[ 'StartTime' ] = pd.to_datetime(df[ 'StartTime' ])
28  df[ 'StartTime' ] = pd.to_datetime(df[ 'StartTime' ], format='%H:%M%S').dt.
    time
29
30  df[ 'EndDate' ] = pd.to_datetime(df[ 'EndDate' ])
31  df[ 'EndDate' ] = pd.to_datetime(df[ 'EndDate' ], format='%d/%m/%Y')
32
33  df[ 'EndTime' ] = pd.to_datetime(df[ 'EndTime' ])
34  df[ 'EndTime' ] = pd.to_datetime(df[ 'EndTime' ], format='%H:%M%S').dt.time
35
36  df[ 'Duration' ] = pd.to_timedelta(df[ 'Duration' ])

```

⁵ Die Webseite https://pbpython.com/pandas_dtypes.html, gibt eine Übersicht der relevanten `DataFrame`-Datentypen. Zuletzt abgerufen am 1. Dezember 2020.

⁶ Modul-Datetime-Webseite <https://docs.python.org/3/library/datetime.html>, zuletzt abgerufen am 1. Dezember 2020.

Im Quellcodebeispiel 15.5 werden die Spalten mit Zeitangaben sequenziell umgewandelt. Zunächst wird mithilfe `pd.to_datetime()` und dem entsprechenden Spaltennamen, wie beispielsweise `df['StartDate']` jeder Eintrag dieser Spalten zu einem `DateTime`-Objekt transformiert. Jedes dieser Objekte besteht aus einer Zeitangabe in einem fest definierten Format. Diese Formatierung besteht in der Angabe von Jahr, Monat, Tag und der Uhrzeit in Form von Stunden, Minuten und Sekunden. Für die Spalte `'StartDate'` reicht die Zeitstruktur in Form von Tag, Monat und Jahr aus. Durch die angehängte Spezifikation der Formatierung durch die Formatierungsanweisung `format= '%d%m%Y')` wird die Anpassung auf das Datumformat vorgenommen. Analog dazu wird die Spalte `'StartTime'` auf die ausreichenden Informationen Stunden, Minuten und Sekunden reduziert. Deshalb wird eine zusätzliche Anweisung ausgeführt, in dem nur die notwendigen Informationen berücksichtigt werden. Dies gelingt mit dem zusätzlichen Formatierungsauftrag `format= '%H:%M:%S').dt.time,` der direkt für alle Einträge der Spalte ausgeführt wird. Die Operationen werden analog für die Angabe des Endes einer Aufgabe in Form der Spaltenumwandlung für `EndDate` und `'EndTime'` ausgeführt.

15.3 Zeitmanagement: Explore the Data

Im vorherigen Abschn. 15.2 wurden die Vorbereitungen getroffen, um die Daten als Zeitinformationen verarbeiten zu können. In diesem Abschnitt geht es nun darum, ein Gefühl für die Daten zu erlangen und damit die Daten besser zu verstehen. Als erster Schritt bietet sich hierzu die Ausgabe einiger statistischer Kennzahlen an. Zunächst einmal wäre es sinnvoll zu wissen, wie viele unterschiedliche Aktivitäten im Datensatz gespeichert sind. Die `Date-Frame-Funktion unique()` beantwortet die Frage, nach den einzigartigen Ausprägungen in einer Spalte. Hierzu kann die entsprechende Spalte auf dem `Date Frame` mittels Übergabe der Spaltenbezeichnung `df['Activity']` aufgerufen werden. Anschließend wird das Ergebnis durch `.tolist()` in eine Liste umgewandelt und in der Variable `LActivity` gespeichert sowie auf der Konsole ausgegeben. Das Quellcodebeispiel 15.6 zeigt die Feststellungen der einzigartigen Ausprägungen in der Spalte `'Activity'` des `DataFrame`.

Listing 15.6 Nutzung von `unique()`

```

38 # -----
39 # ----- Explore the data -----
40 # -----
41 # Ausgabe der unterschiedlichen Aktivitaeten
42 LActivity = df['Activity'].unique().tolist()
43 print(LActivity)

```

Das Ergebnis der Listenerstellung aus dem Quellcodebeispiel 15.6 zeigt die Konsolenausgabe 15.7.

Listing 15.7 Konsolenausgabe: LActivity aus unique()

```
1 ['Design', 'Lunch', 'Feedback', 'Development', 'Meetings', 'Learning', 'Consulting', 'Administrative']
```

Das Ergebnis weist acht Aktivitäten aus, die mit der Beschreibung aus der Einleitung des Anwendungsbeispiels (Abschn. 15.1) übereinstimmen. Im nächsten Schritt bietet sich die Erstellung einer ersten deskriptiven Statistik an. Hierbei bietet sich die Betrachtung der Spalte 'Duration' an, um die Verteilung der Arbeitszeiten zu bestimmen. Das Quellcodebeispiel 15.8 zeigt die Umsetzung.

Listing 15.8 Nutzung von describe()

```
45 # Deskriptive Statistik
46 print(df['Duration'].describe())
```

Das Ergebnis des Quellcodebeispiel 15.8 mit der Anweisung zeigt die Konsolenausgabe 15.9.

Listing 15.9 Konsolenausgabe: Describe

```
1 count                4035
2 mean      0 days 00:47:18.444609
3 std       0 days 00:23:07.570501
4 min              0 days 00:02:00
5 25%       0 days 00:29:05.500000
6 50%              0 days 00:48:01
7 75%       0 days 01:04:15.500000
8 max              0 days 01:55:17
9 Name: Duration, dtype: object
```

Die Konsolenausgabe 15.9 zeigt die Verteilung der Aktivitäten. Entsprechend dieser aggregierten Information bietet es sich nun an, die Aktivitäten und ihre zeitlichen Ausprägungen mit statistischen Kennzahlen grafisch darzustellen. Es eignet sich die Verwendung des Boxplot an. Ein Boxplot stellt die Verteilungskennzahlen, wie den Median, das untere 25-Prozent-Quartil, das obere 75-Prozent-Quartil sowie das Minimum und das Maximum grafisch dar.

Die Erstellung eines Boxplot gelingt mit dem seaborn-Modul. Allerdings müssen die Zeitangaben für das seaborn-Modul leicht angepasst werden, indem aus den Datenobjekten Integer-Werte erstellt werden. Dies gelingt mithilfe der Funktion `dt.total_seconds().astype(int)`, die auf der entsprechenden Spalte des Date Frame mit dem Namen 'Duration' aufgerufen werden kann. Hierbei werden die Sekunden auf dem Datetime-Objekt extrahiert und als Integer-Datentyp in eine neue Spalte 'Duration_seconds' gespeichert. Entsprechend der besseren Lesbarkeit werden die Sekunden durch die Anweisung `df['Duration_seconds'] / 60.0` in Minuten umgewandelt und in einer weiteren Spalte 'Duration_minutes' abgelegt. Die Spalte 'Duration_minutes' wird dann an

eine Boxplot-Abbildung übergeben. Die Abbildung selbst wird hinsichtlich ihres Designs für das Buch angepasst. Das Quellcodebeispiel 15.10 zeigt die Datentransformation von Datetime zu Integer-Werten und die anschließende Übergabe an den Boxplot zur grafischen Darstellung.

Listing 15.10 Boxplot für die Aktivitäten entsprechend ihres zeitlichen Anspruchs

```

48 # Datentransformation von datetime zu Int Werten
49 df['Duration_seconds'] = pd.to_timedelta(df['Duration']).dt.
    total_seconds().astype(int)
50 df['Duration_minutes'] = df['Duration_seconds'] / 60.0
51
52 # Boxplot fuer die Aufgaben und ihren Zeitanspruch
53 fig, ax = plt.subplots()
54 ax.spines['right'].set_visible(False)
55 ax.spines['top'].set_visible(False)
56 ax.yaxis.set_ticks_position('left')
57 ax.xaxis.set_ticks_position('bottom')
58 ax = sns.boxplot(x="Activity", y="Duration_minutes", data=df,
59                 palette='Greys', order=sorted(LActivity))
60 plt.xticks(rotation=45, fontsize=6)
61 plt.xlabel('Aufgaben')
62 plt.ylabel('Zeit in Minuten')
63 plt.tight_layout()
64 plt.savefig('2_Activity_Boxplot.pdf')
65 plt.show()

```

Der Boxplot selbst soll nur mit zwei sichtbaren Achsen versehen werden und in einer grauen Farbpalette gehalten sein. Die Beschriftungen der X-Achse werden um 90 Grad gedreht und die Anordnung der Boxplots für jede Aktivität erfolgt entsprechend einer alphabetischen Sortierung auf der Grundlage der Aktivitäten, die bereits in der Liste `LActivity` identifiziert worden sind. Der Aufruf `sorted` sortiert anschließend die unsortierte Liste in alphabetischer Reihenfolge. Der optionale Parameter `order` erwartet diese sortierte Liste und ordnet die Boxplots für die Aktivitäten entsprechend an. Das Ergebnis des Quellcodebeispiels 15.10 zeigt die Abb. 15.2 in Form des Boxplot.

Aus Sicht der Unternehmensberatung und aus der Perspektive von Marlene kann es sinnvoll sein, zu wissen, welche Aktivitäten wieviel der Arbeitszeit in Anspruch nehmen. Zu diesem Zweck erscheint es sinnvoll, herauszufinden, welche Aufgaben welchen Minutenanspruch benötigt haben. Das Quellcodebeispiel 15.11 zeigt die Zählung der Minuten für die Aktivitäten in der Spalte 'Activity'.

Listing 15.11 Finde Aufgabe im Date Frame mit loc

```

67 # Bestimme den Anteil der Aktivitaeten
68 # Fuer eine Kategorie.
69 dfAdministrative = df.loc[df['Activity'] == 'Administrative']
70 iAdministrativeInMin = dfAdministrative['Duration_minutes'].sum()

```

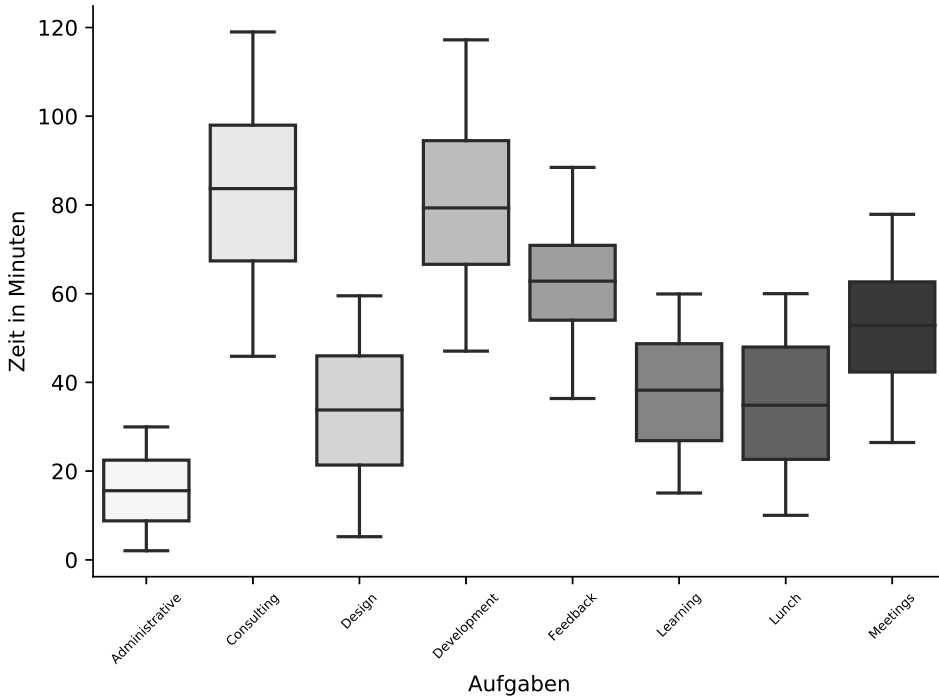


Abb. 15.2 Zeitliche Verteilung der Aktivitäten in Form von Boxplots

Die DataFrame-Funktion `loc` ermöglicht es unter anderem, alle Zeilen mit einer bestimmten Bedingung zu identifizieren, sodass das Ergebnis ein neues DataFrame ist. Die Funktion `loc()` des DataFrame im Quellcodebeispiel 15.11 ermöglicht den Zugriff auf eine Gruppe von Zeilen und Spalten über die Ausprägungen der Aktivitäten in der Spalte 'Activity'. Die Bedingung hierbei ist, alle Zeilen zu finden, bei denen in der Spaltenzeile 'Activity' der Eintrag 'Administrative' zu finden ist. Das Ergebnis im Quellcodebeispiel 15.11 ist ein neues DataFrame mit dem Namen `dfAdministrative`. Auf diesem DataFrame wird nun die Funktion `sum()` aufgerufen, um alle Minuten aus der Spalte 'Duration_minutes' im neuen DataFrame `dfAdministrative` zu summieren. Dieses Vorgehen könnte nun für alle weiteren Aktivitäten implementiert werden. Allerdings würden hierbei zusätzliche DataFrames und damit unweigerlich zusätzliche Quellcodes entstehen, was die Lesbarkeit und die Wartbarkeit des Quellcode minimieren würde.

Eine effizientere Lösung ist die Nutzung der Date-Frame-Funktion `groupby()`. Die Funktion ist imstande, die Zeilen eines DataFrame zu gruppieren. Im Zuge dieses Anwendungsbeispiels lassen sich die Minuten für jede Aktivität entsprechend gruppieren, was zu einem neuen DataFrame führt. Das Quellcodebeispiel 15.12 zeigt die Umsetzung.

Listing 15.12 Verwendung von Groupby für die Aktivitäten

```

72 # Kompakte Variante fuer alle Aktivitaeten
73 dfActivityMin = df.groupby('Activity')['Duration_minutes'].sum()

```

Im Gegensatz zum ersten Ansatz in Form des Quellcodebeispiels 15.11 zeigt sich, dass die Umsetzung in 15.12 nur eine einzige Zeile lang ist. Der Vorteil bei der Anwendung besteht darin, dass durch den Aufruf `df.groupby('Activity')['Duration_minutes'].sum()` zunächst alle Einträge nach ihren Aktivitäten gruppiert werden. Anschließend werden die Einträge in der Spalte 'Duration_minutes' durch die `sum()`-Anweisung summiert. Das Ergebnis `dfActivityMin` ist eine Datentypstruktur vom Typ Series, ein eindimensionales ndarray mit Achsenbeschriftungen, das alle Aktivitäten und deren Minutenanzahl umfasst. Auf Basis der aggregierten Minuteninformationen lässt sich nun ermitteln, wie viel Prozent auf jede einzelne Aktivität abgefallen sind. Das Vorgehen zeigt das Quellcodebeispiel 15.13.

Listing 15.13 Anteil der Aktivitäten an der genutzten Arbeitszeit

```

75 # Pie chart
76 # @source: https://medium.com/@kvnampara/a-better-visualisation-of-pie-charts-by-matplotlib-935b7667d77f
77 LLabels = dfActivityMin.index
78 LSizes = dfActivityMin.tolist()
79
80 # colors
81 LColors = ['#DCDCDC', '#D3D3D3', '#C0C0C0', '#A9A9A9',
82           '#808080', '#696969', '#BCBCBC', '#E0E0E0']
83
84 # Pie Plot
85 fig, ax = plt.subplots()
86 ax.pie(LSizes, colors=LColors, labels=LLabels, autopct='%1.1f%%',
87       startangle=90)
88
89 # Lege weissen Ring ueber den Pie Plot. Fuer modernes Layout.
90 centre_circle = plt.Circle((0, 0), 0.70, fc='white')
91 fig.gca().add_artist(centre_circle)
92
93 # Sicherstellen des gleichen Seitenverhaeltnisses zwischen
94 # Pie Plot und der weissen Abdeckung.
95 ax.axis('equal')
96 plt.tight_layout()
97 plt.savefig('3_Activity_Plot.pdf')
98 plt.show()

```

Zum Zweck der Zeitnutzung wird im Quellcodebeispiel 15.13 ein sogenannter Pieplot (Tortendiagramm) mit modifiziertem Design erstellt. Hierzu werden zunächst aus der Series `dfActivityMin` (Unterform des DataFrame) die Bezeichnungen der Aktivitäten

LLabels und die Werteinformationen in Form der Minuten LSizes ausgelesen. Zudem werden für die geeignete Darstellung in diesem Buch sechs Grautöne in Form von hexadezimalen Farbcodes festgelegt. Die Werte werden dann einem Pieplot-Objekt übergeben. Das Objekt selbst wird dem Abbildungsachsen-Objekt mit `ax.pie(...)` angehängt. Anschließend soll ein Teil des Pieplot durch eine weiße Fläche abgedeckt werden, sodass die Grafik einer modernen Form, wie beispielsweise dem Donutplot (Ringdiagramm), ähnelt. Diese Ringdarstellung verzichtet, im Gegensatz zum Tortendiagramm, auf die Füllung des Innenraums, sodass eine freie Fläche für weitere Informationen genutzt werden kann. In den hier angeführten Grafikmodulen ist dieses Ringdiagramm nicht standardmäßig enthalten, sodass es auf Basis des Tortendiagramms selbst erstellt werden muss. Hierzu wird eine weiße Fläche mit `plt.Circle((0, 0), 0.70,c='white')` definiert. Diese definierte Fläche wird der initialisierten Abbildung durch `fig = plt.gcf()` und `fig.gca().add_artist(centre_circle)` hinzugefügt. Abschließend werden die Achsen des Pieplot und die weiße Fläche mit `ax.axis('equal')` zentriert, sodass sie übereinander liegen und die Abbildung in einem Breitformat `plt.tight_layout()` als .PDF `plt.savefig()` gespeichert werden kann.

Das Ergebnis des Quellcodebeispiels 15.13 zeigt die Abb. 15.3.

Die Abb. 15.3 zeigt, dass die Beratung und die Entwicklung von Prototypen der Klienten die größten Aktivitäten ausmacht. Die anderen Anteile setzen sich aus den weiteren Aufgaben, wie Meetings, Führungsverantwortung, Weiterbildung, Administrativem und Konzeptionellem zusammen.

Im Rahmen der Kundenabrechnungen ist es wichtig zu identifizieren, welche Zeitaufwendungen abrechenbar sind. Hierzu bietet sich zunächst die Einteilung der Aktivitäten in nicht abrechenbare und abrechenbare Stunden an. Mithilfe einer Bedingungsprüfung über die Einträge des DataFrame kann solch eine Klassifizierung erfolgen. Das Quellcodebeispiel 15.14 zeigt das Vorgehen.

Listing 15.14 Werteüberprüfung in DataFrame zur Klassifikation von Werten mit loc

```

100 # Find abrechenbare Stunden mit Hilfe binärer codierung
101 df.loc[df['Activity'] == 'Administrative', 'type'] = 0
102 df.loc[df['Activity'] == 'Consulting', 'type'] = 1
103 df.loc[df['Activity'] == 'Development', 'type'] = 1
104 df.loc[df['Activity'] == 'Design', 'type'] = 1
105 df.loc[df['Activity'] == 'Feedback', 'type'] = 0
106 df.loc[df['Activity'] == 'Learning', 'type'] = 0
107 df.loc[df['Activity'] == 'Lunch', 'type'] = 0
108 df.loc[df['Activity'] == 'Meetings', 'type'] = 0
109
110 print(df['type'].value_counts())

```

Die Funktion `loc()` des DataFrame im Quellcodebeispiel 15.14 wird genutzt, um einen Vergleich der Spalteneinträge in der Spalte 'Activity' auf einen beliebigen Wert zu ermögli-

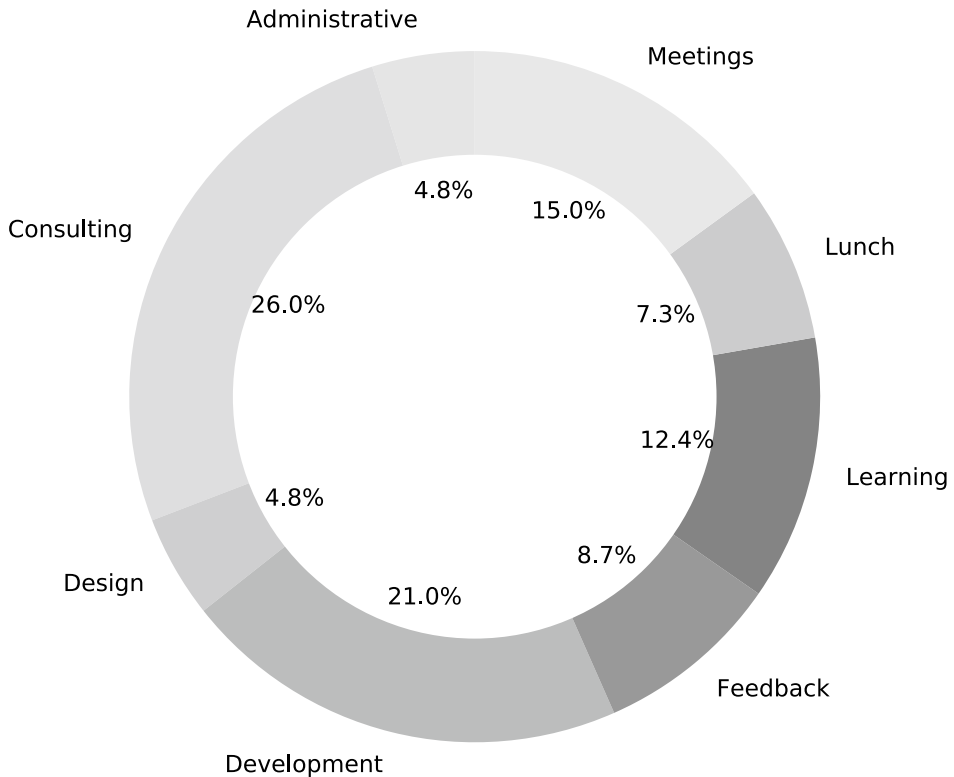


Abb. 15.3 Zeitliche Verteilung der Aktivitäten in Form eines Ringdiagramms

chen. Sofern in einer betreffenden Zeile ein solcher beliebiger Wert vorhanden ist, wird mit dem angehängten Kommentar eine neue DataFrame-Spalte 'type' deklariert, die einen beliebigen Wert übergeben bekommt, um ihn in der neuen Spalte in der entsprechenden Zeile zu speichern. Damit lässt sich die formulierte Anweisung wie folgt verallgemeinern:

```
df.loc[df['Spalte']== 'Vergleichswertin Zeile der Spalte',
'Neue Spalteim DataFrame']='Neuer Wertin Zeile der neuen Spalte'
```

Im Quellcodebeispiel 15.14 werden also die Aktivitäten entsprechend ihres Typs mit 0 für „nicht abrechenbar“ und mit 1 für „abrechenbar klassifiziert“. Mithilfe der Funktion `.value_counts()` lässt sich nun die Anzahl der abrechenbaren Stunden bestimmen und auf der Konsole ausgeben.

Für die Erstellung einer Rechnung wäre es natürlich sinnvoll, die angefallenen Stunden mit einem Preis zu versehen und den Endpreis für die abrechenbaren Stunden auszuweisen. Dies gelingt mit einer einfachen Multiplikation der Minutenanzahl mit dem Stundensatz

Tab. 15.1 Abrechenbare Stunden in Euro für das Arbeitsjahr

Aktivität	Jahrespreis auf Stundenbasis:
Consulting	856.658,88
Design	158.223,33
Development	690.603,06

dividiert durch 60 (für den Stundensatz auf Minutenbasis). Das Quellcodebeispiel 15.15 zeigt die Vorgehensweise.

Listing 15.15 Berechnung der abrechenbaren Stunden

```
112 # Berechnung der abrechenbaren Stunden
113 df['Price'] = df['Duration_minutes'] * (1000/60)
114
115 # Groupby operation mit Bedingung
116 dfPrice = df[df['type'] == 1].groupby('Activity')['Price'].sum()
117 dfPrice.to_latex('GesamtsummeAbrechenbar.tex')
118
119 # Summiere einen Preis
120 dPrice = df['Price'][df['type'] == 1].sum()
```

Im ersten Schritt des Quellcodes wird direkt auf der DataFrame-Spalte 'Duration_minutes' die Berechnung des Preises durchgeführt. Für dieses Anwendungsbeispiel wird einfach ein Preis von 1000.000 pro Stunde als Beraterhonorar festgesetzt. Dieser Preis wird im Codebeispiel noch durch 60 dividiert, um den Minutenpreis zu bestimmen. Dieser Minutenpreis wird mit der Anzahl an Minuten in jeder Zeile multipliziert und in einer neuen Spalte 'Price' gespeichert. Zur Bestimmung der Gesamtsumme für jede relevante Aktivität wird die .groupby()-Funktion entsprechend durch eine vorgelagerte Anweisung erweitert. Hierbei wird auf dem DataFrame eine zusätzliche Bedienung eingefügt. Durch df[df['type'] == 1] werden nur die Zeilen berücksichtigt, die mit einer 1 in der 'type' Zeile versehen sind. Das Ergebnis ist wieder eine Series. Die neue Series wird für dieses Buch als L^AT_EX-Tabelle mithilfe der Funktion to_latex exportiert. Die Gesamtsumme aller relevanten Positionen wird über die Funktion sum() auf dem DataFrame ermittelt, bei dem eine vorangestellte Bedingung df['type'] == 1 die Erfassung der relevanten Daten sicherstellt. Das Ergebnis zeigt Tab. 15.1.

Die Gesamtsumme, wie in Variable dPrice ermittelt, beläuft sich auf etwa 1.705.485,28EUR für die angefallene abrechenbare Arbeitsleistung von Marlene für ihre geleistete Arbeit im vergangenen Jahr.

15.4 Zeitmanagement: Model the data

Ausgehend von den Vorarbeiten im vorherigen Abschnitt besteht nun ein gutes Verständnis über die Daten. Vor diesem Hintergrund stellt sich nun die Frage, ob sich eine übergeordnete Struktur in den Daten findet, mit der sich ein Modell entsprechend trainieren lässt, um dadurch die Arbeitszeit für bestimmte Aktivitäten vorhersagen zu können, entsprechend der Eingangsfragen des Anwendungsbeispiels aus Abschn. 15.1. Für diese Fragestellung eignet sich zunächst die Betrachtung der Daten. Auf Basis des DataFrame `df` lässt sich erkennen, dass die Zielgröße für ein späteres Modell der Zeitaufwand sein kann. Vor dem Hintergrund, dass die Sekunden und Minuten einen hohen Grad an Präzision erfordern, kann es eine Überlegung sein, die Zielvariable in Stunden zu transformieren. Diese Maßnahme normiert die Zeitangabe in gewisser Weise für das Modell, sodass die Prognosequalität erhöht werden kann, weil das Modell weniger Ausprägungen erlernen muss. Damit ergibt sich die Erstellung einer neuen Spalte im Data Frame '`Duration_h`', die das Ergebnis der Spalteneinträge aus '`Duration_minutes`' dividiert durch 60 in Stunden speichert. Als Feature bietet sich zunächst die Betrachtung der Aktivitäten '`Activity`' an. Allerdings sind die Aktivitäten als Strings gespeichert und repräsentieren damit kategorische Daten, die sich schlecht im Sinn eines KI-Systems zur Prognose verarbeiten lassen. Für die Transformation von kategorischen Daten hin zu numerischen Daten sind zahlreiche Verfahren denkbar. Allerdings hat sich im Lauf der Zeit im Bereich der KI-Forschung ein bestimmtes Verfahren durchgesetzt, das im angelsächsischen Raum als One Hot Encoding bezeichnet wird. Das One-Hot-Encoding-Verfahren transformiert jede Ausprägung der kategorischen Daten zu einem Vektor aus Nullen und Einsen, sodass eine Matrix aus den kategorischen Variablen und ihrem Vorhandensein entsteht. Die Formalisierungen 15.1 zeigen die aktuelle Struktur der betreffenden Variablen und das Ergebnis des One Hot Encoding in 15.2 für dieses Anwendungsbeispiel.

$$\begin{bmatrix} \textit{Duration_h} & \textit{Activity} \\ 15.9 & \textit{Administrative} \\ 106.4 & \textit{Consulting} \\ 59.37 & \textit{Design} \\ \dots & \dots \end{bmatrix} \quad (15.1)$$

Die Formalisierung 15.1 zeigt den Datensatz in seiner aktuellen Struktur. Hierbei sind die Spalten '`Duration_h`' und '`Activity`' in ihren Dimensionen identisch, da jede Zeile einen zusammenhängenden Satz an Informationen repräsentiert.

$$\begin{bmatrix} \textit{Duration_h} & \textit{Administrative} & \textit{Consulting} & \textit{Design} & \dots \\ 15.9 & 1 & 0 & 0 & \dots \\ 106.4 & 0 & 1 & 0 & \dots \\ 59.37 & 0 & 0 & 1 & \dots \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \quad (15.2)$$

Durch das One Hot Encoding in der Formalisierung 15.2 werden die kategorischen Daten in Form der Spalte 'Activity' zu einer Matrix transformiert. Die Spalten repräsentieren jeweils ein einziges Merkmal. Die Spalte 'Duration_h' bleibt dabei unangetastet. Jede Zeile in der 'Duration_h'-Spalte wird durch Einsen mit der entsprechenden Merkmalsausprägung verknüpft. Damit referenziert jede Eins aus den Kategorienausprägungen an die Zeilenposition auf die entsprechenden Minutenwerte. Für die Formalisierungshilfe bedeutet dies, dass die Eins in der ersten Zeile in Spalte 'Administrative' auf den Wert 15,9 aus der 'Duration_h'-Spalte verweist. Der Wert 106,4 wird über die Eins der Ausprägung der Spalte 'Consulting' zugeordnet.

Der Vorteil des One Hot Endcoding besteht darin, dass durch die Datentransformation Zustände definiert werden. Damit können KI-Systeme die Datenkonstellationen für diese Zustände erlernen, indem sie versuchen, entsprechende Funktionen abzuleiten. Zur Veranschaulichung kann beispielsweise die folgende Datenkonstellation 15.3 zu einem Ergebnis zwischen 0,5 und 0,6h führen.

$$\begin{bmatrix} \textit{Administrative} & \textit{Consulting} & \textit{Design} & \dots \\ 1 & 0 & 0 & \dots \end{bmatrix} \quad (15.3)$$

Ein anderer Vorteil des One-Hot-Endcoding-Verfahrens ist seine Ressourcensparsamkeit in den gängigen Programmiersprachen. Allerdings ist der Einsatz des One-Hot-Endcoding-Verfahrens mit Bedacht einzusetzen. Beliebig viele kategorische Variablen mittels One Hot Endcoding zu transformieren, ist zwar möglich, allerdings muss über die Sinnhaftigkeit nachgedacht werden. Eine beliebig große Anzahl von möglichen Zuständen kann zu Problemen führen. Zum einen gilt, dass der Rechenaufwand mit der Ausdehnung der One-Hot-Matrix für das KI-Model ansteigt, insbesondere bei tiefen künstlichen neuronalen Netzwerken. Zum anderen ist der Informationsgehalt der einzelnen Ausprägung zu berücksichtigen. Daher gilt, dass die Auswahl der passenden Variablen reflektiert werden sollte.

Für den Einsatz des One Hot Endcoding in Python bietet sich zur Implementierung die Funktion `pd.get_dummies()` an. Alternativ kann beispielsweise das Modul `sklearn` genutzt werden. Dieses Modul stellt im Rahmen von `sklearn.preprocessing` den `LabelBinarizer` bereit, der die Funktion `OneHotEncoder()` anbietet. Allerdings empfiehlt es sich, zunächst die Daten entsprechend der oben genannten Argumentation von Minuten in Stunden zu transformieren. Das Quellcodebeispiel 15.16 zeigt zunächst die Datentransformation.

Listing 15.16 Datentransformation von Minuten zu Stunden

```

122 # -----
123 # ----- Model -----
124 # -----
125 # Datenvorbereitung fuer das Modell mit One Hot Endcoding
126 df['Duration_h'] = df['Duration_minutes']/60
127
128 # Unterschiede in der Skalierung
129 fig, ax = plt.subplots(2, figsize=(7,7))
130 sns.distplot(df['Duration_minutes'], color='darkgray',
131              hist_kws=dict(edgecolor='k', linewidth=0.5), ax=ax[0])
132 sns.distplot(df['Duration_h'], color='k',
133              hist_kws=dict(edgecolor='w', linewidth=0.5), ax=ax[1])
134 # Entfernung der Raender
135 ax[0].spines['right'].set_visible(False)
136 ax[0].spines['top'].set_visible(False)
137 ax[0].yaxis.set_ticks_position('left')
138 ax[0].xaxis.set_ticks_position('bottom')
139 ax[1].spines['right'].set_visible(False)
140 ax[1].spines['top'].set_visible(False)
141 ax[1].yaxis.set_ticks_position('left')
142 ax[1].xaxis.set_ticks_position('bottom')
143 plt.savefig('4_Distribution.pdf')
144 plt.show()

```

Das Quellcodebeispiel [15.16](#) zeigt die Datentransformation von Minuten zu Stunden und die grafische Gegenüberstellung der Daten mithilfe der Histogramme. Die beiden Histogramme werden mithilfe des `ax` den jeweiligen Achsen der Abbildung zu gewiesen. Anschließend werden, wie bisher gezeigt, für beide Unterabbildungen die obere und rechte Kante entfernt. Das Ergebnis zeigt die Abb. [15.4](#).

Die Abb. [15.4](#) zeigt, dass sich die Verteilung der Daten nach der Transformation, wie zu erwarten war, nicht verändert hat. Allerdings verrät der genauere Blick auf die Skalierung der X-Achse, dass das Datenspektrum deutlich gestaucht worden ist. Der Wertebereich hat sich von [0 auf 120] zu [0 auf 2] geändert. Damit muss sich das KI-System im späteren Verlauf auf weniger Lösungsmöglichkeiten konzentrieren. Nach dieser genaueren Betrachtung kann nun das eigentliche One Hot Endcoding stattfinden. Für diesen Anwendungsfall bietet auf Basis des aktuellen Quellcodes die Verwendung der Date-Frame-Funktion `pd.get_dummies()` an, weil damit unmittelbar auf dem DataFrame operiert werden kann. Das Quellcodebeispiel [15.17](#) zeigt das Vorgehen.

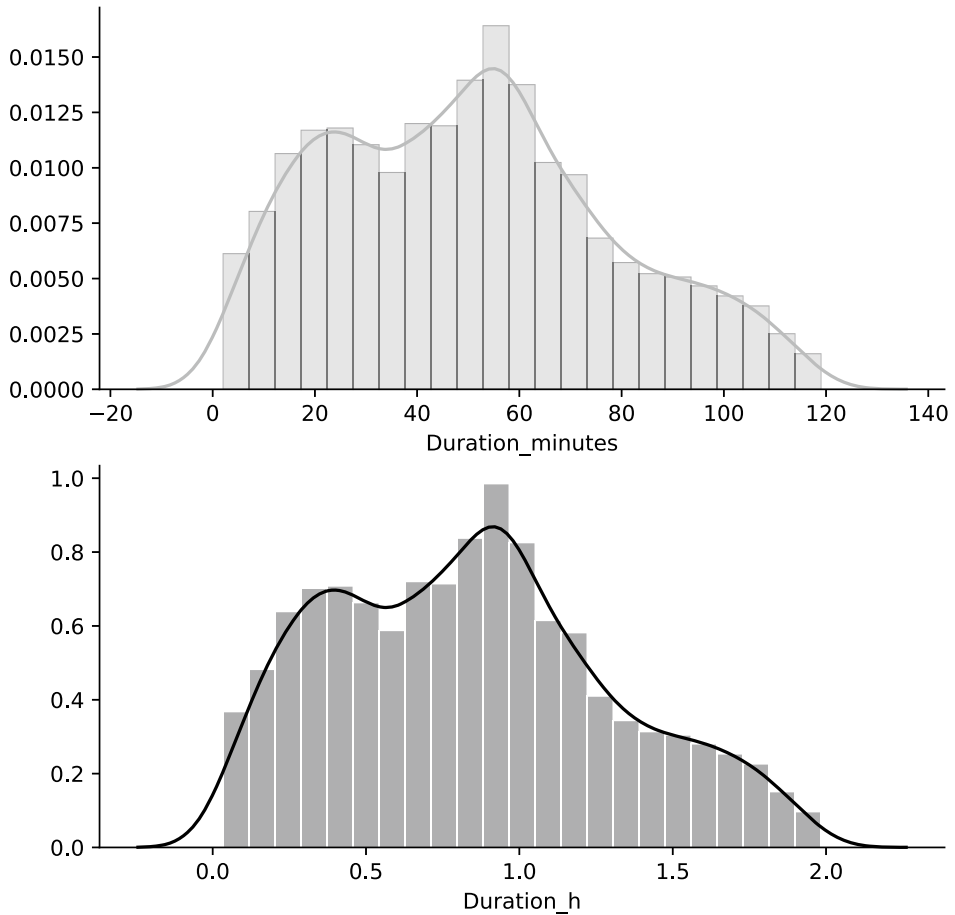


Abb. 15.4 Gegenüberstellung der Daten aus der Datentransformation von Minuten zu Stunden

Listing 15.17 Umsetzung des One Hot Encoding

```

146 # One Hot Encoding
147 dfDummies = pd.get_dummies(df['Activity'])
148 X = dfDummies.to_numpy()
149 y = df['Duration_h'].to_numpy() # Try later: 'Duration_minutes'
150 y = y.astype('int')
```

Im Quellcodebeispiel 15.17 wird mithilfe der Funktion `pd.get_dummies()` die betreffende DataFrame-Spalte 'Activity', analog zu den Formalisierungen 15.1 und 15.2, zu einer Matrix transformiert. Diese Matrix wird in `dfDummies` als Data Frame von der Funktion `pd.get_dummies()` zurückgegeben und gespeichert. Im Anschluss wird ein KI-System auf dem Modul `sklearn` verwendet, sodass sich an dieser Stelle die Transformation der

Zielvariable und der Features (in Form der Daten im One Hot Encoding) anbietet. Die Daten müssen dabei als numpy array vorliegen, sodass die Variable `dfDummies` durch die DataFrame-Funktion `to_numpy()` in ein numpy array `X` transformiert wird. Zudem wird die Zielvariable `df[Duration_h]` ebenfalls durch `to_numpy()` zu einer Numpy-array-Variable. Zusätzlich wird das Ergebnis `y` nochmal explizit als numpy array aus Integer-Werten mit `astype('int')` deklariert. An dieser Stelle wird bei der Variablenbenennung von der bisher getroffenen Konvention abgewichen. Die Variablen `X` und `y` werden in Literatur und Praxis im Kontext von `sklearn` und im Rahmen der KI-Anwendung in der Regel so benannt. Damit der Leser sich an diesen Beispielen orientieren kann, wird daher die verbreitete Benennungskonvention in diesem speziellen Fall verwendet, um den Wiedererkennungswert zu erhöhen.

Der nächste Schritt ist die Aufteilung der Daten in Train und Test Set. Das Quellcodebeispiel 15.18 zeigt das Vorgehen.

Listing 15.18 Umsetzung des One Hot Encoding

```

152 # Notwendige Pakete. Aus didaktischen Gruenden hier.
153 from sklearn.model_selection import train_test_split
154 from sklearn.linear_model import Perceptron
155 from sklearn import metrics
156 # Datenaufteilung in Train und Test.
157 X_train, X_test, y_train, y_test = \
158     train_test_split(X, y, test_size=0.33)

```

Im Quellcodebeispiel 15.18 werden zunächst die notwendigen Module aus `sklearn` importiert. Der erste Import ermöglicht die Aufteilung der Daten in Train und Test Set. Die anderen beide Importe dienen der Realisierung des Modells sowie dessen Auswertung. Aus didaktischen Gründen werden die Importe für die Datenvorbereitung im Rahmen des Modelltrainings, des Modelltestings sowie für das Modell an sich erst jetzt eingeführt. Für eine professionelle Umsetzung sollten die Importe zu Beginn des Skripts, analog zu den bisherigen Importen, eingefügt werden. Mithilfe der Funktion `train_test_split` lassen sich die Daten in einen Test- und Trainingsdatensatz aufteilen. Hierbei wird einen Faktor von 0,33 (33 %) für die Größe des Testdatensatzes festgelegt. Die Funktion liefert auf Basis des Feature Set `X` und der Zielvariable `y` die Datensätze zum Training in Form von `X_train` und `y_train` sowie die Datensätze zum Testen des Modells in Form von `X_test` und `y_test` zurück.

15.4.1 Theoretische Modellgrundlagen

Im Rahmen dieses Anwendungsbeispiels soll ein einfaches künstliches neuronales Netzwerk eingesetzt werden. Im Allgemeinen lassen sich diese Netzwerke als Informations-,verarbeitungssysteme begreifen (vgl. Hoffmann, 2015, S. 129). Dabei finden sich in der Forschung

zahlreiche Formen, sodass eine übergreifende Definition schwierig ist und die bisherige Literatur keine allgemeingültige Definition liefert (vgl. Rey & Wende, 2011, S. 15). Vor diesem Hintergrund ist der Begriff künstliches neuronales Netzwerk als Sammelbegriff zu verstehen (Poddig & Sidorovitch, 2001). In den 1970er-Jahren entstand eine anfängliche Euphorie zur Erschaffung von Prognoseverfahren auf Basis künstlicher neuronaler Strukturen, die bis dahin nur in der Natur zu finden waren. Allerdings trat zum Ende der 1990er-Jahre die Erkenntnis ein, dass die künstlichen neuronalen Netzwerke nicht signifikant besser seien als die bisherigen klassischen Verfahren. Allerdings erleben künstliche neuronale Netzwerke mit dem Aufkommen hochleistungsfähiger Computerprozessoren eine Renaissance. Heute befassen sich zahlreiche Anwendungen und Forschungsgebiete mit dem Einsatz dieser Systeme.

Kernbestandteil jedes künstlichen neuronalen Netzwerks sind die Neuronen⁷. Diese Neuronen sind über gerichtete Verbindungen miteinander verbunden und können bei Bedarf Informationen weiterleiten. In diesem Kontext lassen sich diese künstlichen neuronalen Netzwerke zunächst als mathematische Konstruktionen begreifen. Ein Neuron ist demnach eine grundlegende Verarbeitungseinheit innerhalb des Netzwerks (vgl. Rey & Wende, 2011, S. 19).

Die Funktionsweise der einzelnen Neuronen besteht im Empfang von Informationen x_j , die im Inneren des Neurons akkumuliert werden. Die empfangenen Informationen können von anderen Neuronen stammen oder auf externem Wege dem Netzwerk hinzugefügt worden sein, beispielsweise als Feature-Datensatz.

Die Verbindungsgewichte $w_{i,j}$ geben die Stärke an, mit denen die Inputs in die Neuronen einfließen. Der Input wird innerhalb der Neuronen durch eine Outputfunktion⁸ weiterverarbeitet und in Form eines Ausgangssignals y_i ausgegeben. Die Abb. 15.5 visualisiert die Informationsverarbeitung eines Neurons.

Aus der Abb. 15.5 lassen sich drei zentrale Phasen der Informationsverarbeitung innerhalb eines Neurons feststellen⁹. Zu Beginn steht die Berechnung des Nettoeingangssignals net_i aus den Eingangsinformationen x_j , die entsprechend mit den Verbindungsgewichten w_{ij} gewichtet werden. Zumeist geschieht die Berechnung analog zur Formel 15.4 über die gewichtete Linearkombination der m eingehenden Informationen (vgl. Poddig & Sidorovitch, 2001, S. 402 ff.).

⁷ Neben diesem biologischen Begriff des Neurons werden in der Literatur weitere Definitionen verwendet. Dazu zählen *Unit*, *Proceeding element* bzw. *Processing unit* oder *Knoten* (node).

⁸ In einigen Netzwerktopologien kommt ein definierter Schwellenwert zum Einsatz, der durch eine Aktivierungsfunktion realisiert wird.

⁹ An dieser Stelle ist anzumerken, dass in der Literatur keine konsistente Benennungskonvention für die Schritte der Informationsverarbeitung dargestellt werden. Außerdem kann die Abfolge der hier angeführten Phasen variieren (vgl. Poddig & Sidorovitch, 2001, S. 372).

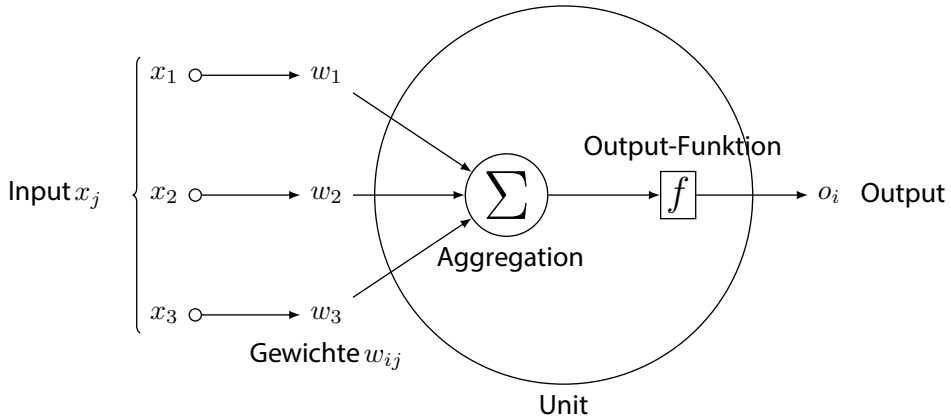


Abb. 15.5 Der Informationsverarbeitungsprozess eines Neurons

$$net_i = \sum_{j=1}^m x_i * w_{i,j} \quad (15.4)$$

- net_i Nettoeingangssignal einer Unit
- x_i Eingangsinformation
- w_{ij} Verbindungsgewicht

Das entsprechende Aktivitätsniveau a_i ergibt sich aus einer Aktivierungsfunktion g . Aus dem Nettoeingangssignal aus der Formel 15.4 wird, gegebenenfalls unter Betrachtung einer zeitlichen Verzögerung, das aktuelle Aktivierungsniveau a_i der Unit U_i entsprechend der Gl. 15.5 bestimmt.

$$a_i = g(net_i, a_i(t - 1)) \quad (15.5)$$

- a_i Aktivitätsniveau
- g Aktivierungsfunktion
- $(t - 1)$ Zeitliche Verzögerung

$$a_i = net_i \quad (15.6)$$

Zur Bestimmung des Outputs o_i einer Unit U_i erfolgt die Einsetzung des Aktivitätsniveaus in eine entsprechende Outputfunktion f .

$$o_i = f(a_i) \quad (15.7)$$

- o_i Der Output der betrachteten Unit U_i
- $f(a_i)$ Die Outputfunktion

Dementsprechend ergibt sich die Outputfunktion aus dem Aktivitätsniveau, das wiederum das Ergebnis der Verarbeitung des Nettoeingangssignals ist. Die Formel 15.8 fasst alle Schritte zusammen.

$$o_i = f \left(g \left(\sum_{j=1}^m x_i * w_{i,j} \right) \right) \quad (15.8)$$

Die hier dargestellten Funktionen (Formeln 15.4 bis 15.8) dienen zur Illustration der allgemeinen Prozessverarbeitungsschritte eines einzelnen Neurons. Sie sind trotz oder gerade wegen ihrer Einfachheit in vielen praktischen Anwendungsfeldern¹⁰ weit verbreitet (vgl. Hoffmann, 2015, S. 160). Kritisch anzumerken ist, dass bei der Modellierung von künstlichen neuronalen Netzwerken die Wahl der Funktionen erheblichen Einfluss auf die Modellergebnisse haben kann (vgl. Crone, 2010, S. 167 f.; vgl. Gupta & Rao, 1994, S. 7).

Das hier dargestellte neuronale Netzwerk, bestehend aus einem Neuron, kann somit als Perzeptron bezeichnet werden. Das Perzeptron stellt den evolutionären Anfang der künstlichen neuronalen Netzwerke dar und fußt auf der Arbeit von Frank Rosenblatt. Dieses Netzwerk wurde das erste Mal 1958 vorgestellt (Rosenblatt, 1958).

15.4.2 Implementierung des Modells

Die Implementierung eines Perzeptrons als einfachstem Vertreter der künstlichen neuronalen Netzwerke kann mithilfe von Perceptron, das Bestandteil der Bibliothek `sklearn.linear_model` aus dem Modul `sklearn` ist, realisiert werden. Das Quellcodebeispiel 15.19 zeigt die Implementierung des künstlichen neuronalen Netzwerks.

Listing 15.19 Implementierung eines Perceptron-Netzwerks

```

160 # -----
161 # Create model
162 # -----
163 # Einfaches Perceptron
164 model = Perceptron()
165
166 # -----
167 # Create train

```

¹⁰ Die Entwicklung verschiedenster künstlich neuronaler Netzwerke für spezifische Aufgabestellungen hat eine Vielzahl von Aggregierungs-, Aktivierungs-, und Outputfunktionen hervorgebracht (vgl. Crone, 2010, S. 170).

```

168 # -----
169 model.fit(X_train, y_train)
170
171 # -----
172 # Evaluate model
173 # -----
174 y_pred = model.predict(X_test)
175
176 # Accuracy:
177 dAccuracy = metrics.accuracy_score(y_test, y_test)
178 print("Accuracy:", metrics.accuracy_score(y_test, y_pred))

```

Im ersten Schritt des Quellcodebeispiels 15.19 wird der Variable `model` ein Standard-Perzeptron zugewiesen; damit besteht das Modell aus einem einzigen Neuron. Wie vorher angeführt (Abschn. 15.4.1), gibt es zahlreiche Konfigurationsmöglichkeiten für die Wertebelegungen der Neuronen im Sinn der Funktionen. In diesem Kontext wird auch von Hyperparameter gesprochen. Das Finden einer optimalen Wertebelegung für künstliche neuronale Netzwerke ist eine Teildisziplin der KI. Vor diesem Hintergrund verwendet dieses Buch die Standardeinstellungen des Moduls¹¹. Das Modell wird anschließend mit der Funktion `model.fit(...)` auf Basis des geteilten Datensatzes auf den Trainingsdaten trainiert. Anschließend erfolgt mit der Funktion `model.predict(...)` die Erzeugung der Prognoseergebnisse des künstlichen neuronalen Netzwerks. Durch die Übergabe der Prognoseergebnisse an die Funktion `metrics.accuracy_score()` aus dem Modul `metrics` können die Prognoseergebnisse `y_test` mit den wahren Werten des Datensatzes `y_test` verglichen werden.

15.5 Zeitmanagement: Interpretation der Ergebnisse

Die Ergebnisse des Perzeptrons sind erstaunlich. Mit einem einzelnen Neuron lassen sich Prognosegenauigkeiten, in Abhängigkeit der zufällig gezogenen Datensatzkonstellation, zwischen 0,74 und 0,88 (74 % bzw. 88 %) erzielen. Dies ergibt sich aus der Ausführung der letzten Zeile des Quellcodebeispiels 15.19. Allerdings muss an dieser Stelle darauf hingewiesen werden, dass das Komplexitätsproblem relativ gering ist. Dies ergibt sich auf der Anzahl der Features. Zudem zeigt der Test mit der Verwendung der Minutenanzahl als Zielgröße in der Variable `Duration_minutes`, dass das Netzwerk nur in einem engen Wertebereich funktioniert. Für den eigenständigen Test kann der Leser die Zeile

¹¹ Eine detaillierte Beschreibung der Standardeinstellung findet sich auf der Webseite des Moduls https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html, zuletzt abgerufen 1. Dezember 2020.

`y = df['Duration_h'].to_numpy()` durch `y = df['Duration_h'].to_numpy()` durch `y = df['Duration_minutes'].to_numpy()` ersetzen. Wird dieser Test durchgeführt, beträgt die Prognosegüte weniger als 2,0 %. Dieser Sachverhalt zeigt, dass der erfolgreiche Einsatz von künstlichen neuronalen Netzwerken abhängig von der zugrunde liegenden Datenstruktur und der damit verbunden Komplexität ist. Sofern diese Herausforderungen jedoch gemeistert werden, können künstliche neuronale Netzwerk ihr Potenzial freisetzen.

Teil VI

Finanzanalyse



Marcel Zeuch

Ogleich der bewusst allgemein gehaltenen Kapitelüberschrift soll dieses vorliegende Kapitel dazu dienen, einen Einblick in die Portfolioanalyse von Wertpapieren bereitzustellen. Analog können allerdings auch andere Assets, wie Immobilienportfolios oder Lagerbestände, anhand der nachfolgenden Überlegungen analysiert und ausgewertet werden. Ein kurzer Einblick in die Wichtigkeit und Funktion einer Portfolioanalyse soll dabei den Einstieg in dieses Kapitel liefern.

Sowohl vor allem historische als auch aktuelle Krisen zeigen, wie volatil und instabil der Finanzmarkt sein kann. Für Anleger und Investoren bieten diese Kursbewegungen Chancen und bringen gleichzeitig viele Risiken. Eine stetige Portfolioanalyse, auch in Zeiten positiver Kursbewegungen, ist demnach essenziell, um eine anhaltend positive Renditeentwicklung zu gewährleisten und den Überblick über das eigene Depot zu behalten. Diesen Prozess zu automatisieren und eine fundamentale grafische Auswertung bereitzustellen, ist Ziel dieses Kapitels. Eine Konkretisierung dieser Fragestellung und Betrachtung der notwendigen Daten zur Umsetzung dieses Vorhabens ist Bestandteil von Abschn. 16.1.

Es gibt verschiedene Möglichkeiten, Wertpapiere zu identifizieren. Insbesondere in Deutschland gebräuchliche Kennnummern sind die zwölfstellige, sogenannte International Securities Identification Number (ISIN) und die Wertpapierkennnummer (WKN). In Abschn. 16.3 und 16.4 wird erörtert, wie anhand der ISIN alle notwendigen Daten zur weiteren Wertpapieranalyse gesammelt werden können. Dabei bedient sich dieses Buch der kostenfrei zugänglichen Programmierschnittstelle, auch API (Application Programming Interface) genannt, des Datenanbieters Alpha Vantage.¹

Der Hauptteil des hier behandelten Anwendungsbeispiels schließlich findet sich in Abschn. 16.5. Nach ersten Codebeispielen in den zuvor genannten Abschnitten wird hier

¹ <https://www.alphavantage.co/>, zuletzt abgerufen am 12. Februar 2022.

das vollständige Skript zur Portfolioanalyse erarbeitet. Während es in Python verschiedene Lösungen zur Implementierung von Benutzeroberflächen, sogenannten Graphical User Interfaces (GUI), gibt, wird hier die Implementierung über die Bibliothek `dash` vorgestellt. Die Bibliothek `dash` grenzt sich dadurch ab, dass bei der Skriptausführung ein Pfad generiert wird, mittels dem die Benutzeroberfläche in einem beliebigen, unterstützten Browser aufgerufen und verwendet werden kann. Darüber hinaus bietet sie den Vorteil einer vergleichsweise sehr simplen Handhabbarkeit. Gleichzeitig kombiniert sie effizient interaktive Benutzeroberflächen sowie zugehörige grafische Auswertungen der zugrunde liegenden Datensätze.

Der fünfte Abschnitt dient der abschließenden Betrachtung der implementierten Lösung und liefert Ideen für weiterführende Analysen, die den Rahmen dieses Buchs übersteigen würden.

16.1 Portfolioanalyse: Datensatz und Fragestellung

Ausgangslage des hier vorliegenden Anwendungsbeispiels ist ein fiktives Musteraktiende-
pot, das über die letzten fünf Jahre hinweg betrachtet werden soll. Im Detail besteht dieses aus 32 zufällig gewählten Aktien aus dem Aktienindex MSCI World. Von Anfang 2015 bis zum zweiten Quartal 2020 wurden die Aktien des Portfolios insgesamt 202 Mal gehandelt, wobei sowohl Handelstag als auch Handelsrichtung und Anzahl der gehandelten Aktien zufällig generiert worden.

Bei diesem vorliegenden Depot soll eine Benutzeroberfläche implementiert werden, die die Kursentwicklung des Depots sowie einzelner Titel über verschiedene Betrachtungszeiträume zur Anzeige zulässt. Zusätzlich sollen die Depotwerte nach ihrem Land und der jeweiligen Industrie kategorisiert werden. Eine solche Übersicht kann aus Gründen der Diversifikation sehr interessant und aufschlussreich sein. Das beschriebene Analysetool wird dabei, wie in Abb. 16.1 dargestellt, in diesem Kapitel umgesetzt.

Um die beschriebenen Funktionalitäten zu implementieren bedarf es nur wenigen Eingabeparametern. Für alle Aktien aus dem vorliegenden Depot werden die folgenden Informationen benötigt:

- Zugehörige Identifikationsnummer (ISIN)
- Handelstag
- Handelsrichtung
- Gehandeltes Nominal, d. h. die Anzahl an Aktien
- Handelspreis

Die ersten drei Werte des Musterportfolios werden in Tab. 16.1 dargestellt. Dabei wird der Kauf einer Aktie mit einer 1 repräsentiert, ein Verkauf entsprechend mit einer 0. Eine gesamte Darstellung des Musterportfolios wird online bereitgestellt.

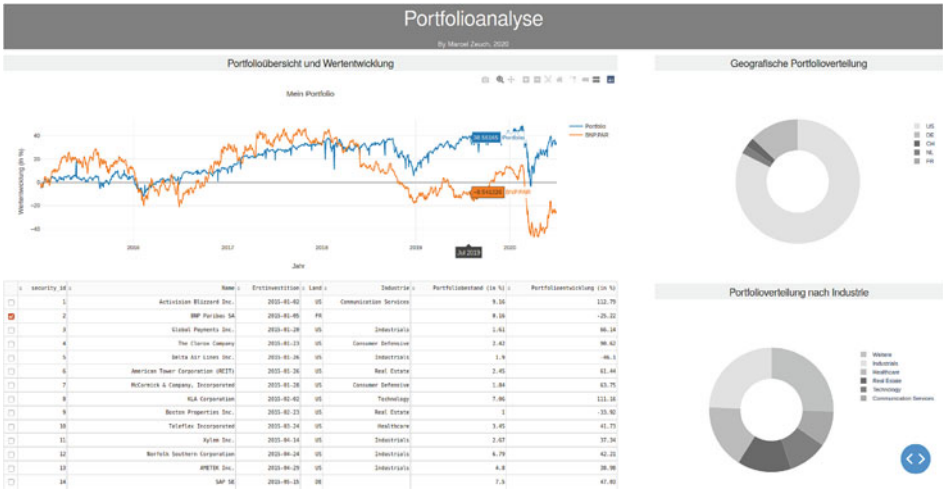


Abb. 16.1 Implementierung des vorgestellten Portfolioanalysetools

Tab. 16.1 Auszug der Eingabeparameter des Musterportfolios

ISIN	Handelstag	Handelsrichtung	Nominal	Handelspreis
US00507V1098	2015-01-02	1	50	20.13
FR0000131104	2015-01-05	1	21	47.30
US37940X1028	2015-01-20	1	11	87.90

16.2 Bereitstellung einer Datenbank

Insbesondere aus Sicht der Performance des hier vorgestellten Portfolioanalysetools ist es sinnvoll, eine Datenbank zur Umsetzung zu bemühen; diese Vorgehensweise bietet sich jedoch auch an, um bei Bedarf Funktionalitäten zu erweitern. Um die Idee dieser Art der Umsetzung zu begründen, soll an dieser Stelle auf den zu bearbeitenden Datensatz eingegangen werden. Neben den bereitzustellenden Datenpunkten, die beispielhaft in Tab. 16.1 wiedergegeben werden, sind insbesondere Daten zu den historischen Aktienkursen der im Portfolio gehaltenen Wertpapiere nötig, um die Entwicklung des Portfolios bestimmen zu können. Gleichmaßen werden in dem vorgestellten Tool statische Daten, wie Name und Industrie der jeweiligen Wertpapiere, verarbeitet. All diese Daten werden dabei durch den Datenanbieter Alpha Vantage bereitgestellt. Der zu implementierende Code greift diese Daten über Anfragen an die entsprechende Schnittstelle ab. Insbesondere zwei Charakteristika dieser API begründen den Vorteil einer Datenbank.

- Einerseits erlaubt die Schnittstelle von Alpha Vantage nur eine begrenzte Anzahl an Datenanfragen, sowohl pro Minute als auch pro Tag. Das sind derzeit fünf Anfragen pro Minute sowie insgesamt 500 Anfragen pro Tag. Werden pro Aktie zwei Anfragen gestellt, ergibt das im Fall des vorgestellten Musterportfolios mindestens 64 Anfragen, die entsprechend über 12 min in Anspruch nehmen. Durch eine vorige Speicherung der Daten in der Datenbank stehen diese instantan zur Verfügung und müssen nur innerhalb des Codes verarbeitet werden. Zudem kann das Ansprechen der Schnittstelle in einem separaten Skript implementiert und somit nur bei Bedarf genutzt werden.
- Andererseits kann es interessant sein, das vorgestellte Tool nicht nur auf Tagesdaten mit den jeweiligen Handelsschlusskursen umzusetzen, sondern mit granulareren Daten zu arbeiten. So stellt Alpha Vantage Marktdaten auch in kleineren Intervallen, wie 15-Minuten-Intervallen, zur Verfügung. Diese Daten können jedoch nicht weiter als etwa ein halbes Jahr in der Vergangenheit heruntergeladen werden. Eine Speicherung dieser Daten in einer eigenen Datenbank löst auch diesen Umstand.

Insbesondere aufgrund des ersten angeführten Arguments arbeitet das in diesem Buch vorgestellte Portfolioanalysetool mit einer Datenbank. Zur Demonstration wird auf eine gängige MySQL-Datenbank zurückgegriffen. Hierbei handelt es sich um eine klassische relationale Datenbank. Diese bietet die Möglichkeit, die unterschiedlichen Daten, die für die Portfolioanalyse verarbeitet werden, zum einen effizient zu strukturieren und zum anderen leicht untereinander zu verknüpfen. Zur Installation der Datenbank sei dabei auf Standardreferenzen oder Beiträge aus dem Internet verwiesen. Kurz soll jedoch die Architektur der aufzubauenden Datenbank diskutiert werden. Die Datenbank gliedert sich dabei in einzelne Schemata, die wiederum verschiedene Tabellen enthalten. Diese bilden die Grundlage der weiteren Vorgehensweise. Im Rahmen dieses Kapitels soll mit vier verschiedenen Tabellen gearbeitet werden. Folgende vier Tabellen sollen angelegt werden:

- `statics` – Speicherung aller relevanten statischen Informationen, wie z. B. ISIN, Handelswährung oder Industrie einer Aktie
- `portfolio_transactions` – Speicherung aller Transaktionen, die im Portfolio durchgeführt worden sind
- `marketdata_daily` – Speicherung aller relevanten Marktdaten zu den gehandelten Aktien
- `marketdata_fx_daily` – Speicherung von Marktdaten zu relevanten Wechselkursen

Für die hier vorgestellte Umsetzung sind das Herunterladen und die Speicherung von Wechselkursen wichtig, da Alpha Vantage teilweise Marktdaten in der entsprechenden Landeswährung bereitstellt. Um Volumina vergleichbar zu machen, ist eine Währungsumrechnung deshalb erforderlich. Im hier vorliegenden Fall werden lediglich die Wechselkurse für EURUSD benötigt. Bevor diese Tabellen jedoch angelegt werden können, muss zunächst das entsprechende Schema in der Datenbank konstruiert werden. In Anlehnung an den Titel

des vorliegenden Buchs erhält dieses den Namen `data_science`. Der zugehörige Befehl in MySQL ergibt sich zu:

Listing 16.1 Anlegen des Schemas `data_science` in MySQL

```
1 CREATE DATABASE data_science;
```

Dieses Kommando wird, wie auch die vier nachfolgenden, in der Datenbank selbst ausgeführt. Konkret werden die vorgestellten Tabellen nun wie folgt in MySQL angelegt:

Listing 16.2 Anlegen der Tabelle `statics` in MySQL

```
1 CREATE TABLE data_science.statics(  
2     id INT NOT NULL,  
3     ticker VARCHAR(45) NULL,  
4     name VARCHAR(256) NULL,  
5     isin VARCHAR(45) NOT NULL,  
6     industry VARCHAR(256) NOT NULL,  
7     PRIMARY KEY (id));
```

Listing 16.3 Anlegen der Tabelle `portfolio_transactions` in MySQL

```
1 CREATE TABLE data_science.portfolio_transactions(  
2     id INT NOT NULL,  
3     security_id INT NOT NULL,  
4     trade_date DATE NOT NULL,  
5     trade_direction INT NOT NULL,  
6     nominal INT NOT NULL,  
7     trade_price DOUBLE NOT NULL,  
8     PRIMARY KEY (id));
```

Listing 16.4 Anlegen der Tabelle `marketdata_daily` in MySQL

```
1 CREATE TABLE data_science.marketdata_daily(  
2     security_id INT NOT NULL,  
3     date DATE NOT NULL,  
4     open DOUBLE NOT NULL,  
5     high DOUBLE NOT NULL,  
6     low DOUBLE NOT NULL,  
7     close DOUBLE NOT NULL,  
8     volume BIGINT(255) NOT NULL);
```

Listing 16.5 Anlegen der Tabelle `marketdata_fx_daily` in MySQL

```
1 CREATE TABLE data_science.marketdata_fx_daily(  
2     date DATE NOT NULL,  
3     open DOUBLE NOT NULL,  
4     high DOUBLE NOT NULL,  
5     low DOUBLE NOT NULL,  
6     close DOUBLE NOT NULL);
```

Bevor im nächsten Abschnitt die Ansprache der Schnittstelle von Alpha Vantage und die Ansprache der Datenbank in Python diskutiert werden, soll an dieser Stelle noch auf die zuvor verwendeten Datentypen eingegangen werden. In diesem Anwendungsfall können fünf unterschiedliche Datentypen differenziert werden.

- **BIGINT** – Kurzform für Big Integer. Dieser Datentyp umfasst besonders große ganzzahlige Zahlen
- **DATE** – Dieser Datentyp speichert Datumsangaben
- **DOUBLE** – Der Datentyp Double speichert rationale Zahlen, d. h. Zahlen mit endlich vielen Nachkommastellen
- **INT** – Kurzform für Integer. Dieser Datentyp speichert entsprechend ganzzahlige Zahlen
- **VARCHAR** – Der Datentyp Varchar speichert Zeichenketten

Die Zahlen, die teilweise in Klammern hinter den einzelnen Datentypen aufgeführt werden, entsprechen der maximalen Länge an Buchstaben bzw. Ziffern, die gespeichert werden können. Der Zusatz **NULL** beschreibt, dass das entsprechende Datenfeld a priori keine Eingabe erwartet. Entsprechend beschreibt der Zusatz **NOT NULL**, dass ein Eintrag für das entsprechende Datenfeld erwartet wird. Die übrigen Bezeichnungen kennzeichnen die Spaltennamen der jeweiligen Tabellen.

Hierbei ist zu beachten, dass die Eingabeparameter aus Tab. 16.1 in der Datenbank in zwei unterschiedliche Tabellen, **statics** und **portfolio_transactions**, aufgeteilt werden. Dabei entspricht die **id** eines Wertpapiers gerade der **security_id** in der Tabelle **portfolio_transactions**. Um diese beiden Tabellen in der Datenbank mit den Daten des Musterportfolios zu befüllen, werden die Daten des Musterportfolios in zwei **.csv**-Dateien entsprechend der Struktur der jeweiligen Datenbanktabellen abgespeichert. Leere Felder werden dabei mit **\N** gefüllt. Die **.csv**-Dateien können dann mit dem entsprechenden Pfadnamen der Datei sowie dem jeweiligen Tabellennamen mittels dem nachstehenden Befehl in die Datenbank geladen werden.

Listing 16.6 Upload einer **.csv**-Datei in eine Tabelle der Datenbank

```
1 load data local infile 'PFADNAME DER DATEI' into table
2 data_science.TABELLENNAME fields terminated by ',' optionally
3 enclosed by '"' ignore 1 lines;
```

Die entsprechenden **.csv**-Dateien für das vorgestellte Musterportfolio werden online zur Verfügung gestellt.

16.3 Portfolioanalyse: Pre-processing

Ausgangslage dieses zweiten Abschnitts ist die aufgesetzte Datenbank, die bereits alle Informationen zum gehandelten Musterportfolio beinhaltet. Diese Daten müssen im weiteren Ver-

lauf ergänzt und schließlich verarbeitet und aufbereitet werden. Um die bisherigen Daten im historischen Verlauf bewerten und auswerten zu können, ist es notwendig, entsprechende Marktdaten zu den gehandelten Wertpapieren zu aggregieren. Diese sollen ebenfalls in der Datenbank gespeichert werden. Ziel dieses zweiten Abschnitts ist es entsprechend, die Ansprache von API und Datenbank in Python zu diskutieren, um die weitere Datenaggregation und -verarbeitung umsetzen zu können.

Entsprechend der Reihenfolge, wie die Daten später verarbeitet werden sollen, wird dieser Abschnitt durch eine Diskussion zum Zugriff auf die Schnittstelle für die Marktdaten eröffnet. Der in diesem Kapitel exemplarisch vorgestellte Marktdatenanbieter ist dabei Alpha Vantage. Alpha Vantage bietet einen kostenlosen Zugang zu vielen Aktien, insbesondere mit Fokus auf die USA sowie Foreign Exchange (FX) Daten. Gleichzeitig führt Alpha Vantage verschiedene statische Daten zu Aktien sowie viele unterschiedliche technische Indikatoren, die über die zugehörige Schnittstelle angefragt werden können. In jedem Fall ist es hierzu nötig, zunächst einen persönlichen Schlüssel zur API zu beantragen, der jeder Anfrage eindeutig einen Benutzer zuordnet. Dieser Key lässt sich unter Angabe einer E-Mail-Adresse unter <https://www.alphavantage.co/support/#api-key> anfordern. Eine ausführliche Dokumentation, wie die verschiedenen Daten bei Alpha Vantage über die API angefragt werden können, findet sich unter <https://www.alphavantage.co/documentation/>. Im Rahmen dieses Kapitels werden insbesondere Marktdaten für Aktien sowie Daten für FX, jeweils auf Tagesschlusskursen basierend, untersucht. Zur Ansprache der API wird die Bibliothek `requests` eingebunden. Diese ermöglicht es, auf einfache Weise Serveranfragen im Web zu stellen und die entsprechenden, in HTML zurückgegebenen Skripte zu analysieren. Informationen, die so normalerweise im Browser dargestellt werden, können auf diese Weise automatisiert eingelesen und verarbeitet werden. Im konkreten Fall soll diese Funktionalität für die erstgehandelte Aktie des vorgestellten Musterportfolios genauer erläutert werden. Hierbei handelt es sich um die Aktie von Activision Blizzard. Für diese sollen exemplarisch statische Daten sowie Marktdaten über die API von Alpha Vantage angefragt werden. Alpha Vantage nutzt dabei den Börsenticker einer Aktie, um diese den entsprechenden Marktdaten zuzuordnen. Dieser muss somit zuerst ermittelt werden. Alpha Vantage bietet hierfür eine Suchfunktion, die Funktion `SYMBOL_SEARCH`, die es ermöglicht, den Börsenticker einer Aktie über einen vorgegebenen Suchbegriff zu identifizieren. Die Suchfunktion lässt dabei beliebige Zeichenketten als Suchwörter zu. Aufgrund der gegebenen Datenlage bietet es sich an, nach der ISIN der Aktie zu suchen. Dieses Vorgehen hat den Vorteil, dass die ISIN einer Aktie eindeutig ist. Der offiziellen Dokumentation von Alpha Vantage folgend, wird eine entsprechende Suchanfrage im Browser mittels https://www.alphavantage.co/query?function=SYMBOL_SEARCH&keywords=US00507V1098&apikey=APIKEY gestellt. Die Ausgabe, die über den Browser zurückgegeben wird, kann genauso auch in Python automatisiert entgegengenommen und verarbeitet werden. Der nachstehende Code zeigt die Umsetzung.

Listing 16.7 Umsetzung der Anfrage des Tickers zu einer spezifizierten ISIN

```

1  # Importierung der notwendigen Bibliothek
2  import requests
3
4  # Festsetzung der URL zur Anbindung an die API von Alpha Vantage
5  api_url = 'https://www.alphavantage.co/query'
6
7  # Bereitstellung der Eingabeparameter fuer die Anfrage an die API
8  inp = {"function": 'SYMBOL_SEARCH',
9        "keywords": 'US00507V1098',
10       "apikey": 'APIKEY'}
11
12 # Suchanfrage an die Schnittstelle
13 response = requests.get(api_url, inp)
14 out = response.json()
15
16 # Ausgabe der empfangenen Rueckgabe
17 print(out)

```

Der in Listing 16.7 dargestellte Code liefert die nachfolgende Konsolenausgabe:

Listing 16.8 Konsolenausgabe: print(out) zur Funktion SYMBOL_SEARCH

```

1  {'bestMatches': [{'1. symbol': 'ATVI', '2. name': 'Activision Blizzard Inc.', '3.
    type': 'Equity', '4. region': 'United States', '5. marketOpen': '09:30', '6.
    marketClose': '16:00', '7. timezone': 'UTC-05', '8. currency': 'USD', '9.
    matchScore': '0.1250'}, {'1. symbol': 'ATVIX', '2. name': 'Athena Behavioral
    Tactical Fund Class I', '3. type': 'Mutual Fund', '4. region': 'United States',
    '5. marketOpen': '09:30', '6. marketClose': '16:00', '7. timezone': 'UTC-05', '
    8. currency': 'USD', '9. matchScore': '0.1176'}, {'1. symbol': 'ATV134.SAO', '2.
    name': 'Activision Blizzard Inc.', '3. type': 'Equity', '4. region': 'Brazil/
    Sao Paolo', '5. marketOpen': '10:00', '6. marketClose': '17:30', '7. timezone':
    'UTC-03', '8. currency': 'BRL', '9. matchScore': '0.0952'}]}

```

Während die Ausgabe in Listing 16.8 zunächst verwirrend erscheint, folgt sie doch einer stringenten Zusammensetzung. Die Funktion SYMBOL_SEARCH ordnet jedem Symbol neun verschiedene Merkmale zu, die entsprechend durchnummeriert werden. Diese werden jeweils als ein einziges Listenelement ausgegeben und sind als Dictionary formatiert. Ebenfalls erfolgt die gesamte Ausgabe als Dictionary. Im vorliegenden Fall ordnet Alpha Vantage der vorgegebenen ISIN insgesamt drei verschiedene Symbole zu. Das erste Symbol, mit der höchsten angezeigten Übereinstimmung, entspricht der gehandelten Aktie des Musterportfolios. Entsprechend soll auch in der späteren Implementierung immer nur das erste Listenelement verarbeitet werden. Um das in Abb. 16.1 dargestellte Tool zu realisieren, wird neben dem Symbol und dem Namen auch die zugehörige Währung gespeichert. Greift man über den jeweiligen Schlüssel auf ein Element des Dictionary zu, so lässt sich die Speicherung dieser Daten wie folgt umsetzen:

Listing 16.9 Umsetzung der Datenspeicherung aus dem Dictionary

```
1 # Speicherung des Symbols, des Namens und der zugehoerigen Waehrung einer Aktie
2 ticker = (out['bestMatches'][0])[1].symbol
3 name = (out['bestMatches'][0])[2].name
4 currency = (out['bestMatches'][0])[8].currency
```

```

5                                     password=PASSWORD)
6
7     cursor = connection.cursor()
8
9     # Datenspeicherung durch Update der Datenbank
10    sql_update = "update statics set ticker = '" + ticker + \
11                  "' , name = '" + name + "' , currency = '" + \
12                  currency + "' where id = 1"
13    cursor.execute(sql_update)
14    connection.commit()

```

16.4 Portfolioanalyse: Explore the Data

Ziel dieses dritten Abschnitts ist es, die Kenntnisse aus den vorigen Abschnitten zu ergänzen und fortzuführen. Nachdem die Ansprache der API und der MySQL-Datenbank diskutiert wurden, soll untersucht werden, wie alle weiteren benötigten Daten, sowohl aus der API als auch der Datenbank heraus, verarbeitet werden können. Damit wird der Grundstein für die eigentliche Implementierung des vorgestellten Tools in den folgenden Abschnitten gelegt. Hierzu soll zunächst ein genauerer Blick auf die angefragten Marktdaten geworfen werden. Listing 16.10 stellte bereits den entsprechenden Code zur Anfrage an die API bereit. Die zugehörige Konsolenausgabe liest sich wie folgt:

Listing 16.12 Konsolenausgabe: print(out) zur Funktion TIME_SERIES_DAILY

```

1  {'Meta Data': {'1. Information': 'Daily Prices (open, high, low, close) and Volumes',
2                '2. Symbol': 'ATVI', '3. Last Refreshed': '2020-10-09', '4. Output Size': 'Full
3                size', '5. Time Zone': 'US/Eastern'}, 'Time Series (Daily)': {'2020-06-30': {'
4                1. open': '75.6900', '2. high': '76.4300', '3. low': '75.4200', '4. close': '
5                75.9000', '5. volume': '6535740'}, '2020-06-29': {'1. open': '75.8200', '2. high
6                ': '76.5150', '3. low': '74.9300', '4. close': '75.5000', '5. volume': '5099362'
7                }, '2020-06-26': {'1. open': '76.5400', '2. high': '76.8400', '3. low': '74.6500
8                ', '4. close': '76.4000', '5. volume': '11572670'}, ...}}

```

Analog zur Funktion SYMBOL_SEARCH gibt Alpha Vantage die Marktdaten in einem geschachtelten Dictionary zurück. Neben allgemeinen Informationen zu Beginn der Ausgabe besteht die historische Zeitreihe jeweils immer aus den folgenden Informationen: Jedem Tag sind die Angaben open, high, low, close und volume zugeordnet. Das heißt, wie hoch war der Börsenöffnungskurs bzw. -schlusskurs, welcher war der höchste bzw. der niedrigste gehandelte Wert am entsprechenden Handelstag und welches Volumen wurde an diesem Tag an der entsprechenden Börse gehandelt.

Um effizient auf diese Daten zugreifen zu können ist es sinnvoll an dieser Stelle zusätzlich die Bibliothek pandas zu involvieren. Mithilfe der Funktion DataFrame.from_dict lassen sich die Daten aus dem Dictionary in einen DataFrame konvertieren. Hierzu greift man zunächst mit dem Schlüssel Time Series (Daily) auf diese zu.

Listing 16.13 Überführung der bereinigten Marktdaten in einen DataFrame

```

1 # Importierung zusätzlicher Bibliotheken
2 import pandas as pd
3
4 # Zugriff auf die bereinigten Marktdaten und Verarbeitung in einem DataFrame
5 marketdata = out['Time Series (Daily)']
6 df = pd.DataFrame.from_dict(marketdata, orient="index")
7
8 # Ausgabe des DataFrame
9 print(df)

```

Man erhält die nachfolgende Konsolenausgabe:

Listing 16.14 Konsolenausgabe: `print(df)` der Marktdaten als DataFrame

```

1          1. open  2. high  3. low  4. close  5. volume
2  2020-06-30  75.6900  76.4300  75.4200  75.9000   6535740
3  2020-06-29  75.8200  76.5150  74.9300  75.5000   5099362
4  2020-06-26  76.5400  76.8400  74.6500  76.4000  11572670
5  2020-06-25  76.6100  76.7700  74.4600  76.2100   6157310
6  2020-06-24  76.4600  77.1700  74.4300  75.5800   7744934
7  ...          ...          ...          ..          ...
8  1999-11-05  14.5600  15.5000  14.5000  15.3800   1308267
9  1999-11-04  14.6900  14.7500  14.2500  14.6200   207033
10 1999-11-03  14.3100  14.5000  14.1200  14.5000    61600
11 1999-11-02  14.2500  15.0000  14.1600  14.2500   128817
12 1999-11-01  14.1900  14.3800  13.9400  14.0600   173233
13
14 [5270 rows x 5 columns]

```

Während die statischen Daten jeweils einzeln über ein Update in die Datenbank geschrieben werden können, ist dies für die Marktdaten nicht möglich. Die statischen Informationen wie Ticker und Name einer Aktie ergänzen lediglich noch nicht befüllte Datenfelder. Die Marktdaten zu einer Aktie müssen hingegen für jeden Handelstag neu in der Datenbank angelegt werden. Gleichzeitig ist es bei der vorliegenden Anzahl an Daten effizienter, diese aggregiert in die Datenbank zu schreiben. Diese Idee kann durch die Erzeugung einer temporären .csv-Datei umgesetzt werden. Um der Struktur der Datenbanktabelle `marketdata_daily` gerecht zu werden, ist eine Ergänzung dieses Datensatzes um die entsprechende Security ID des jeweiligen Wertpapiers nötig. Der nachstehende Code zeigt die Umsetzung unter Verwendung der Funktionen `Series` und `concat`.

Listing 16.15 Erweiterung des DataFrame `df`

```

1 # Ergänzung des DataFrame um die jeweilige Security ID
2 sec = (pd.Series(1*len(df), index=df.index)).rename('security_id')
3 df = pd.concat([sec, df], axis=1)

```

Der Upload der Marktdaten kann daraufhin wie folgt implementiert werden:

Listing 16.16 Upload der Marktdaten in die Datenbank über eine .csv-Datei

```

1  # Importierung zusätzlicher Bibliotheken
2  import os
3
4  # Temporäre Speicherung der Marktdaten in einer .csv-Datei und Upload in die Datenbank
5  df.to_csv('tempfile.csv')
6
7  sql_load = """load data local infile 'PFADNAME DER DATEI'
8              into table data_science.marketdata_daily
9              fields terminated by ','
10             optionally enclosed by '"'
11             ignore 1 lines"""
12  cursor.execute(sql_load)
13  connection.commit()
14
15  os.remove('tempfile.csv')
```

Um später einzelne Wertpapiere vergleichen zu können, müssen diese die gleiche Währung besitzen. Das wird insbesondere wichtig, wenn es darum geht, das Portfolio nach dem Marktwert der einzelnen Aktien zu gewichten. Aus diesem Grund ist es einerseits von großer Bedeutung, die entsprechende Währung des jeweiligen Wertpapiers in der Datenbank zu speichern, andererseits wird der tagesgenaue Wechselkurs benötigt, um verschiedene Währungen ineinander umrechnen zu können. Dieser kann über Alpha Vantage mit der Funktion `FX_DAILY` angefragt werden. Benötigter Eingabeparameter ist dabei das entsprechende Währungspaar. Für das hier vorgestellte Musterportfolio ist das das Währungspaar EURUSD. Die Eingabeparameter für die Ansprache der API werden wie in Listing 16.17 definiert. Die Verarbeitung der Daten und der Upload in die Datenbank erfolgen analog zu den zuvor diskutierten Marktdaten.

Listing 16.17 Eingabeparameter zur Ansprache der Funktion `FX_DAILY`

```

1  inp = {"function": 'FX_DAILY',
2        "from_symbol": 'EUR',
3        "to_symbol": 'USD',
4        "outputsize": 'full',
5        "apikey": APIKEY}
```

Um die vollständige Funktionalität des hier vorgestellten Tools nutzen zu können, sollen zusätzlich noch automatisiert Daten zu dem jeweiligen Land sowie der jeweiligen Industrie einer Aktie gespeichert werden. Diese können über Alpha Vantage beispielsweise über die Funktion `OVERVIEW` angefragt werden. Die Eingabeparameter für die Ansprache der API werden wie nachfolgend definiert. Die Verarbeitung der Daten und der Upload in die Datenbank erfolgen dabei analog zu den übrigen angelegten, statischen Daten.

Listing 16.18 Eingabeparameter zur Ansprache der Funktion OVERVIEW

```
1 inp = {"function": 'OVERVIEW',  
2       "symbol": ticker,  
3       "apikey": APIKEY}
```

In Ergänzung zum Upload von Daten in die Datenbank ist es ebenfalls essenziell zu verstehen, wie in Python auf die notwendigen Daten aus der Datenbank zugegriffen werden kann, um diese weiter zu verarbeiten. Hierfür sollen zunächst wichtige Grundbefehle diskutiert werden, die dabei helfen, Daten in der Datenbank geeignet zu strukturieren. Sind die Daten einmal wie benötigt aggregiert, so lässt sich die Datenbanktabelle einfach in einen DataFrame auslesen. Folgende Grundbefehle werden in diesem Abschnitt genutzt. Befehle, die analog in der Bibliothek pandas als Funktionen bereitgestellt werden, sollen der Vollständigkeit halber dennoch mit in diese Übersicht aufgenommen werden.

- **group by *** – Gruppiert die gegebene Tabelle nach den Werten einer vorzugebenden Spalte. Alle Werte in der vorgegebenen Spalte werden dedupliziert, d. h. auf eindeutige Werte reduziert. Die übrigen Spalten enthalten im Allgemeinen den jeweils zugehörigen zuerst auftretenden Wert. Der Befehl **group by** bietet sich beispielsweise dazu an, einzelne Datengruppen zu summieren sowie maximale oder minimale Werte zu bestimmen.
- **if(*, *, *)** – Die **if**-Anweisung entspricht von der Funktionalität her der klassischen Formel aus den Office-Paketen. Im ersten Argument wird eine zu prüfende Bedingung aufgestellt. Tritt diese ein, wird das zweite Argument ausgeführt. Im anderen Fall das dritte Argument.
- **join * on *** – Der Befehl **join** fügt zwei Tabellen zusammen. Hierbei muss einerseits deklariert werden, welche Tabelle der vorgegebenen Tabelle hinzugefügt werden soll. Andererseits müssen beide Tabellen mindestens eine Spalte mit identischen Werten besitzen, über die die jeweiligen Daten zugeordnet werden können.
- **left(*, *)** – Der Funktion **left** werden zwei Argumente übergeben. Das zweite Argument ist eine Zahl. Es werden dann nur die entsprechend ausgewählte Anzahl an Ziffern und Buchstaben des ersten Arguments dargestellt.
- **left join * on *** – Der Befehl **left join** entspricht von der Funktionalität her dem Befehl **join**. Jedoch findet die Datenzuordnung bei letzterem eindeutig statt. Der Befehl **left join** hingegen erlaubt es, dass einem Datenfeld der vorgegebenen Tabelle mehrfach Daten aus der anzufügenden Tabelle zugeordnet werden können. Insbesondere bedeutet das, dass die vorgegebene Tabelle potenziell mehr Einträge durch die Zusammenführung erhält.
- **limit *** – Definiert die Anzahl an Zeilen, die ausgegeben werden.
- **max(*)** – Entspricht der Maximumfunktion. Der maximale Wert einer Datengruppe wird ausgegeben.
- **min(*)** – Entspricht der Minimumfunktion. Der minimale Wert einer Datengruppe wird ausgegeben.

- `order by *` – Ordnet die gegebene Tabelle nach den Werten einer vorzugebenden Spalte. Die Zusätze `asc` und `desc` legen fest, ob die Ordnung auf- oder absteigend erfolgen soll. Ohne entsprechenden Zusatz werden die Werte aufsteigend geordnet.
- `round(*,*)` – Entspricht der Rundungsfunktion. Die Zahl im ersten Argument der Funktion wird mit der Anzahl an Nachkommastellen der im zweiten Argument angegebenen Zahl gerundet.
- `sum(*)` – Entspricht der Summenfunktion. Die Summe einer Datengruppe wird ausgegeben.
- `where *` – Restrings die Werte einer Tabelle durch eine vorgegebene Einschränkung, beispielsweise eine vorgegebene (Un-)Gleichung. Mehrere Einschränkungen können durch den Befehl `and` konjugiert werden.

Insbesondere bei der Zusammenführung zweier oder mehrerer Tabellen ist es wichtig, diese separat und explizit zu benennen. Das ist notwendig, um eine doppelte oder mehrfache Datenbezeichnung zu vermeiden. Um auf eine Tabelle in der Datenbank zuzugreifen, genügt die Eingabe `select * from TABELLENNAME`. Das Symbol `*` veranlasst dabei die Ausgabe aller Spalten, die in der Tabelle vorhanden sind und kann nach Belieben auch durch eine Auflistung der relevanten Spaltennamen ersetzt werden. Das anschließende Auslesen einer Tabelle in Python und die Speicherung in einem DataFrame werden wie folgt implementiert:

Listing 16.19 Abfrage an die Datenbank und Speicherung in einem DataFrame

```

1 # Abfrage an die Datenbank und Speicherung im DataFrame 'df'
2 sql_select = "ABFRAGE AN DIE DATENBANK"
3 cursor.execute(sql_select)
4 df = pd.DataFrame(cursor.fetchall())

```

Die Funktion `fetchall` stellt dabei sicher, dass die gesamte aggregierte Tabelle in dem DataFrame gespeichert wird. Alternativ kann nur die erste Zeile der Datenbanktabelle in den DataFrame geladen werden, wenn die Funktion `fetchone` genutzt wird. Bei dem vorgestellten Code in Listing 16.19 werden darüber hinaus keine Spaltenbezeichnungen mit ausgelesen. So kann entweder über die Spaltennummerierung auf die Daten zugegriffen werden, alternativ lassen sich die Spalten nach Belieben benennen. In dem vorliegenden Kapitel wird dieser zweite Ansatz verfolgt, sodass nachvollziehbar ist, welche Daten weiterverarbeitet werden.

16.5 Portfolioanalyse: Model the Data

Nach den Vorbereitungen in den vorigen Abschnitten soll nun die eigentliche Modellierung des Tools zur Portfolioanalyse anhand des Musterportfolios bereitgestellt werden. Die Grundlagen zur Anbindung der benötigten API als auch an die Datenbank wurden dabei in den Abschn. 16.3 und 16.4 diskutiert.

Neben der eigentlichen Programmierung ist es wichtig, sich zunächst ein Konzept zur Umsetzung zu überlegen. Wie in Abb. 16.1 dargestellt, soll das zu implementierende Portfolioanalysetool die Kursentwicklung des gesamten Depots sowie einzelner Titel über verschiedene Betrachtungszeiträume zulassen. Zusätzlich sollen die gehaltenen Depotwerte nach dem Land und der jeweiligen Industrie kategorisiert werden. Für eine grundlegende Variante dieser Übersicht soll die Auswertung auf Tagesdaten basieren. Wie in der Finanzindustrie üblich soll hierzu der jeweilige Börsenschlusskurs als maßgebende Kurskennzahl verwendet werden. Ziel ist es entsprechend, die Wertentwicklung des Musterportfolios anhand der Börsenschlusskurse zu modellieren, wobei die Wertentwicklung des Portfolios als prozentuale Veränderung dargestellt werden soll. Anders ausgedrückt: Es soll untersucht werden, wie sich der Wert des Portfolios ausgehend von ursprünglich 100 % über den Lauf der Zeit entwickelt hat. Diese Betrachtung ist insofern sinnvoll, da sie die reine Wertentwicklung des Portfolios widerspiegelt und um Einzahlungen in und Auszahlungen aus dem Depot bereinigt ist. Gleichzeitig birgt diese Implementierung eine gewisse Komplexität, schließlich muss für jeden Handelstag, an dem es einen Zu- oder Verkauf im Portfolio gab, das Portfolio neu gewichtet werden, um die tatsächliche Wertentwicklung bestimmen zu können.

Der besseren Übersicht und einfacheren Handhabbarkeit wegen wird die vorgestellte Implementierung in zwei sich ergänzenden, jedoch separaten Skripten umgesetzt. Das erste Skript übernimmt dabei die Datenaggregation und die Speicherung der Daten in die Datenbank. Aufgrund der vorgegebenen Datenbeschränkungen bei Alpha Vantage kann der Durchlauf dieses Skripts mehrere Minuten in Anspruch nehmen. Das zweite Skript hingegen extrahiert die vorhandenen Daten aus der Datenbank, verarbeitet diese entsprechend weiter und setzt schließlich auch die Visualisierung der interaktiven Benutzeroberfläche mit dash um.

16.5.1 Datenaggregation und Speicherung in der Datenbank

Der Grundstein zur Implementierung, um alle notwendigen Daten zu aggregieren und anschließend in die Datenbank zu speichern, wurde bereits in den beiden vorigen Abschnitten gelegt. An dieser Stelle sollen jedoch einzelne Besonderheiten der Implementierung hervorgehoben und die generelle Struktur dieses ersten Skripts diskutiert werden. Der entsprechende Code ist online einseh- und abrufbar. Zusätzlich zu den bisher verwendeten Bibliotheken soll die Funktion `sleep` aus der Bibliothek `time` importiert werden. Hiermit lässt sich die Ausführung des Skripts an ausgewählten Positionen innerhalb des Codes für eine vorgegebene Zeitspanne pausieren. Diese Funktionalität soll dazu genutzt werden, die erlaubten fünf Anfragen pro Minute an die API von Alpha Vantage einzuhalten. Das Skript zur Datenaggregation und Speicherung in die Datenbank gliedert sich strukturell in drei Teile.

Nach der Bereitstellung aller notwendigen Bibliotheken und der Initialisierung der Schnittstelle zur MySQL-Datenbank wird diese in einem ersten Teil auf Wertpapiere über-

prüft, zu denen keine statischen Daten in der Datenbank vorhanden sind. Solche Wertpapiere erscheinen zum ersten Mal in der Datenbank und besitzen entsprechend auch noch keinen zugeordneten Ticker. Die Überprüfung erfolgt, indem die Länge des in Listing 16.20 generierten DataFrame `df_check` getestet wird.

Listing 16.20 Abfrage aller Security ID, zu denen noch keine Statics angelegt wurden

```

28 # Abfrage aller Security ID, zu denen noch keine Statics angelegt worden
29 sql_select = "select id from statics where ticker is null"
30 cursor.execute(sql_select)
31 df_check = pd.DataFrame(cursor.fetchall())

```

Ist der DataFrame `df_check` leer, kann das Anlegen fehlender Statics übersprungen werden. Andernfalls werden die entsprechenden fehlenden, statischen Daten analog zu den Codebeispielen in Listing 16.7, 16.9, 16.11 und 16.18 zusammengestellt und in die Datenbank gespeichert. Konkret zeigt Listing 16.21 die Umsetzung der Anfrage fehlender Ticker. Die Anfrage und Speicherung ist dabei als Schleife über die Anzahl der fehlenden Ticker implementiert.

Listing 16.21 Anfrage fehlender Ticker und Speicherung in der Datenbank

```

36 # Zusammenstellung aller ISIN der jeweiligen Security ID und Speicherung in
    DataFrame 'df_isin'
37 sql_select = "select id, isin from statics where ticker is null"
38 cursor.execute(sql_select)
39 df_isin = pd.DataFrame(cursor.fetchall())
40
41 # Anfrage des jeweiligen Ticker ueber die API
42 for i in range(0, len(df_isin)):
43
44     # Pausieren der Anfrage nach jeweils fuenf Security ID fuer 60 Sekunden
45     if (i != 0) and (i % 5 == 0):
46         sleep(60)
47
48     # Bereitstellung der Eingabeparameter fuer die Anfrage an die API
49     inp = {"function": 'SYMBOL_SEARCH',
50           "keywords": df_isin[1][i],
51           "apikey": APIKEY}
52
53     # Suchanfrage an die Schnittstelle und Datenspeicherung durch Update der
        Datenbank.
54     # Fehlerausgabe im Fall nicht vorhandener Daten
55     try:
56         response = requests.get(api_url, inp)
57         out = response.json()
58
59         sql_update = "update statics set ticker = '" + \
60                     (out['bestMatches'][0])[1. symbol'] + "', name = '" + \
61                     (out['bestMatches'][0])[2. name'] + "', currency = '" + \

```

```

62             (out['bestMatches'][0])[8. 'currency'] + "' where id = " +
                str(df_isin[0][i])
63         cursor.execute(sql_update)
64         connection.commit()
65     except:
66         print("Fuer folgende ISIN wird kein Ticker gefunden: " + str(df_isin[1][i]
            ))

```

Diese Struktur des Codes wiederholt sich im ersten Skript mehrfach. Die Zeilen 45 und 46 stellen dabei sicher, dass immer nur höchstens fünf Anfragen pro Minute an die API gestellt werden. Dieses Pausieren ist über eine einfache `if`-Anweisung gelöst, die genau dann eintritt, wenn die Laufvariable `i` bei Division durch fünf keinen Rest zurückgibt. Dies ist genau bei jeder fünften Iteration der Fall. Diese Operation wird durch das Prozentzeichen in Zeile 45 implementiert und ist auch unter dem Begriff `modulo` bekannt.

Entsprechend der Zusammenstellung der Datenbanktabelle in Zeile 37 enthält das Datenfeld `df_isin[1][i]` jeweils die zur Laufvariable `i` gehörende ISIN, die in Zeile 50 als Eingabeparameter verarbeitet wird. Zuletzt wird die Suchanfrage an die API und die Datenspeicherung in der Datenbank mit einer Fehlerausgabe implementiert, falls die entsprechenden Daten nicht durch Alpha Vantage bereitgestellt werden (können).

Analog zum vorgestellten Code wird die Funktion `OVERVIEW` genutzt, um die weiteren statischen Daten zu Land und Industrie eines Wertpapiers bei Alpha Vantage anzufragen und diese in der Datenbank zu ergänzen. Schließlich wird auch die Anfrage und Speicherung der gesamten Marktdatenhistorie in einer dritten Schleife implementiert. Die Umsetzung ist in Listing 16.22 dargestellt.

Listing 16.22 Anfrage der gesamten historischen Marktdaten und Upload in die Datenbank

```

104     # Festsetzen eines Zaehlers
105     count = 0
106
107     # Anfrage der gesamten historischen Marktdaten ueber die API fuer die neu
        angelegten Security ID
108     for i in range(0, len(df_ticker)):
109
110         # Pausieren der Anfrage nach jeweils fuenf Security ID fuer 60 Sekunden
111         if (i != 0) and (i % 5 == 0):
112             sleep(60)
113
114         # Bereitstellung der Eingabeparameter fuer die Anfrage an die API
115         inp = {"function": "TIME_SERIES_DAILY",
116              "symbol": df_ticker[1][i],
117              "outputsize": 'full',
118              "apikey": APIKEY}
119
120         # Suchanfrage an die Schnittstelle und Datenspeicherung im DataFrame 'dff'.
121         # Fehlerausgabe im Fall nicht vorhandener Daten
122         try:

```

```

123     response = requests.get(api_url, inp)
124     out = response.json()
125
126     marketdata = out['Time Series (Daily)']
127     df = pd.DataFrame.from_dict(marketdata, orient="index")
128     sec = (pd.Series([df_ticker[0][i]]*len(df), index=df.index)).rename('
        security_id')
129     df = pd.concat([sec, df], axis=1)
130
131     # Zusammenfuegen der verschiedenen Marktdaten in einen DataFrame
132     if i == count:
133         dff = df
134     else:
135         dff = pd.concat([dff, df])
136 except:
137     count = count + 1
138     print("Fuer folgende security ID werden keine Marktdaten gefunden: " +
        str(df_ticker[0][i]))
139
140 # Temporaere Speicherung der Marktdaten in einer .csv-Datei und Upload in die
141 Datenbank
142 dff.to_csv('tempfile.csv')
143
144 sql_load = """load data local infile 'tempfile.csv'
145             into table data_science.marketdata_daily
146             fields terminated by ','
147             optionally enclosed by '"'
148             ignore 1 lines"""
149 cursor.execute(sql_load)
150 connection.commit()
151
152 os.remove('tempfile.csv')

```

Während wie zuvor nur Marktdaten verarbeitet werden, die überhaupt durch Alpha Vantage bereitgestellt werden, gibt es in diesem Abschnitt des Codes eine Besonderheit zu beachten. Da der Code in Listing 16.22 auch für eine Vielzahl an verschiedenen Wertpapieren effizient ausgeführt werden soll, werden nicht bei jeder Iteration die angefragten Marktdaten direkt in die Datenbank gespeichert. Stattdessen werden sie für alle Wertpapiere zunächst in einem einzigen DataFrame aggregiert. Dieser wird anschließend in eine .csv-Datei konvertiert und diese dann in die Datenbank geladen. Die entsprechende Umsetzung ist durch die Zeilen 141 bis 151 gegeben. Für die eigentliche Aggregation ist es wichtig, den DataFrame mit einem ersten Datensatz zu initiieren und dann mit weiteren Datensätzen zu ergänzen. Diese Umsetzung findet sich in den Zeilen 132 bis 135. Die Überprüfung, ob die aktuelle Laufvariable *i* dem festgesetzten Zähler *count* entspricht, ist dabei notwendig, um den DataFrame zu initialisieren. Die abweichende Bedingung *i == 0* würde nicht genügen. Im Fall, dass keine Marktdaten für die erste Laufvariable bereitgestellt werden können, wäre der

DataFrame `dff` in Zeile 135 für die nachfolgende Laufvariable dann nicht definiert. Damit endet der erste Teil dieses Skripts. Allen neuen Wertpapieren wurden die entsprechenden statischen Daten zugeordnet sowie die gesamte Marktdatenhistorie wurde bis zum aktuellen Tag gespeichert.

Der zweite Teil des Skripts verarbeitet hingegen die Marktdaten bereits vorhandener Wertpapiere. Die Datenanfrage an die API unterscheidet sich dabei nicht von der zuvor diskutierten Vorgehensweise. Um Marktdaten jedoch weder doppelt in die Datenbank zu schreiben noch die Marktdaten neuer Wertpapiere direkt erneut anzufragen, wird die folgende Abfrage an die Datenbank genutzt. Diese bestimmt diejenigen Ticker, zu denen keine aktuellen Marktdateninformationen in der Datenbank gespeichert sind.

Listing 16.23 Abfrage an die Datenbank zur Bestimmung fehlender Marktdaten

```

159 # Abfrage an die Datenbank: Ausgabe aller Wertpapiere, fuer die keine aktuellen
    Marktdaten in der Datenbank
160 # vorhanden sind. Speicherung im DataFrame 'df_ticker'
161 sql_select = """select * from
162     (select m.security_id as 'security_id', s.ticker as 'ticker', max(m.date
        ) as 'last_date' from statics s
163     join marketdata_daily m on m.security_id = s.id
164     where s.ticker is not null
165     group by m.security_id) a
166     where a.last_date != curdate()-1"""
167 cursor.execute(sql_select)
168 df_ticker = pd.DataFrame(cursor.fetchall())

```

Konkret setzt sich die Abfrage in Listing 16.23 wie folgt zusammen: Zu Beginn wird die Tabelle `statics` selektiert und mit der Tabelle `marketdata_daily` zusammengeführt. Die Tabelle wird dabei auf Einträge restringiert, zu denen bereits ein Ticker gespeichert ist. Anschließend wird die gesamte Tabelle nach der Security ID gruppiert. Das heißt, alle unterschiedlichen Einträge zu jeweils einer Security ID werden in eine Zeile aggregiert. Dabei sollen nur die jeweilige Security ID selbst, der zugehörige Ticker sowie der jeweils maximale Tag innerhalb der Tabelle `marketdata_daily` an Datenfeldern zur Ansicht ausgegeben werden. Um doppelte Bezeichnungen und eine damit verbundene fehlerhafte Ausgabe zu vermeiden, werden den jeweiligen Datenfeldern die entsprechenden Tabellennamen in Form der Abkürzungen `m` und `s` mit angegeben. Diese, in den Zeilen 162 bis 165 konstruierte Tabelle, wird erneut mit der Abkürzung `a` bezeichnet. Gleichzeitig soll sie zuletzt auf alle Werte restringiert werden, bei denen der maximale Tag nicht dem gestrigen Tag entspricht. Die Funktion `curdate` gibt dabei immer den aktuellen Tag wieder. Die Bereitstellung dieser Tabelle im DataFrame `df_ticker` bietet somit Zugang zu allen Tickern, für die keine aktuellen Marktdaten gespeichert sind.

Mit der gewohnten Anfrage an die Schnittstelle von Alpha Vantage können die Marktdaten daraufhin wie in Listing 16.24 verarbeitet und in die Datenbank geschrieben werden. Die Betrachtung einer Partition der Marktdaten, gespeichert im DataFrame `dff` in Zeile 192, stellt sicher, dass nur Marktdaten in die Datenbank geschrieben werden, die dort noch nicht

gespeichert sind. Da der DataFrame `dff` eine deutlich reduzierte Anzahl an Daten enthält, können diese, wie implementiert, iterativ per `insert`-Befehl in die Datenbank gespeichert werden.

Listing 16.24 Verarbeitung der angefragten Marktdaten und Upload in die Datenbank

```

182     # Suchanfrage an die Schnittstelle und Datenspeicherung durch Einschreiben in die
        Datenbank.
183     # Fehlerausgabe im Fall nicht vorhandener Daten
184     try:
185         response = requests.get(api_url, inp)
186         out = response.json()
187
188         marketdata = out['Time Series (Daily)']
189         df = pd.DataFrame.from_dict(marketdata, orient="index")
190
191         # Restriktion der Marktdaten auf Daten, die zeitlich nach dem letzten in der
            Datenbank vorhandenen Datum liegen
192         dff = df[df.index > str(df_ticker[2][i])]
193
194         for j in range(0, len(dff)):
195
196             sql_insert = "insert into marketdata_daily (security_id, date, open, high
                , low, close, volume) values" + \
197                 "(" + str(df_ticker[0][i]) + ", '" + \
198                 dff.index[j] + "', '" + \
199                 dff['1. open'][j] + "', '" + \
200                 dff['2. high'][j] + "', '" + \
201                 dff['3. low'][j] + "', '" + \
202                 dff['4. close'][j] + "', '" + \
203                 dff['5. volume'][j] + "')"
204             cursor.execute(sql_insert)
205             connection.commit()
206     except:
207         print("Fuer folgende security ID werden keine Marktdaten gefunden: " + str(
            df_ticker[0][i]))

```

Der dritte Teil des ersten Skripts schließlich behandelt die Anfrage und Speicherung von FX-Marktdaten, konkreter den tagesgenauen Wechselkursen für EURUSD. Die entsprechende Anfrage wurde dabei bereits in Listing 16.17 diskutiert. Um analog zu den Marktdaten der Aktien eine Fallunterscheidung führen zu können, ob die Wechselkurse zum ersten Mal in die Datenbank geladen werden oder nur einzelne Werte aktualisiert werden müssen, dient die folgende Abfrage.

Listing 16.25 Abfrage und Speicherung der ersten Zeile der Tabelle `marketdata_fx_daily`

```

216 # Abfrage an die Datenbank: Ausgabe der ersten Zeile der Datenbanktabelle fuer FX-
    Marktdaten.
217 # Speicherung im DataFrame 'df_check'
218 sql_select = "select * from marketdata_fx_daily limit 1"
219 cursor.execute(sql_select)
220 df_check = pd.DataFrame(cursor.fetchall())

```

Ist die zurückgegebene Tabelle leer, das heißt, entspricht die Länge des DataFrame `df_check` dem Wert Null, so wird die gesamte Marktdatenhistorie gezogen und über einen `upload`-Befehl in die Datenbank gespeichert. Im anderen Fall werden lediglich die fehlenden Daten bis zum aktuellen Datum über einen `insert`-Befehl in der Datenbank gespeichert.

16.5.2 Verarbeitung der Daten und Implementierung der Benutzeroberfläche

Nach dem Durchlauf des ersten Skripts sind nun alle relevanten Informationen, die zur Umsetzung des vorgestellten Analysetools benötigt werden, in der Datenbank gespeichert. Von hier aus können die Daten ohne weiteren Zugriff auf die API von Alpha Vantage verarbeitet werden. Anschließend erfolgt die Implementierung der Benutzeroberfläche. Das zweite Skript ist dabei wie nachstehend strukturiert. Zunächst werden die Daten zur Darstellung des aktuellen Portfolios sowie zur Darstellung der geografischen Allokation und der Allokation nach Industrie aus der Datenbank aggregiert. Anschließend wird die historische Portfolioentwicklung in Python berechnet. In einem letzten Schritt werden diese Daten mit `dash` visualisiert und wie in Abb. 16.2 dargestellt.

Zusätzlich zu der bereits vorhandenen, von `dash` bereitgestellten Interaktivität soll die tabellarische Übersicht des aktuellen Portfolios ebenfalls dynamisch umgesetzt werden. Konkret soll die Auswahl einzelner Aktientitel im Portfolio ermöglicht werden, die in ihrer historischen Entwicklung im Portfolio gegeneinander oder mit der Gesamtperformance des Portfolios verglichen werden können, vergleichbar mit Abb. 16.2. Wie auch im vorigen Abschnitt soll nur eine Auswahl des implementierten Codes vorgestellt und diskutiert werden. Analoge oder sich wiederholende Abschnitte sind dem Leser zur eigenen Durchsicht zu überlassen. Der gesamte Code wird online zur Verfügung gestellt.

Der besseren Nachvollziehbarkeit halber sollen in diesem Abschnitt teilweise einzelne, zu diskutierende Codezeilen direkt im Fließtext in Klammern mit der jeweiligen Zeilennummerierung angegeben werden.

Folgende Bibliotheken werden zur Umsetzung des zweiten Skripts benötigt:

Listing 16.26 Initialisierung notwendiger Bibliotheken und Komponenten

```

6 # Importierung von dash und zugehoeriger Komponenten
7 import dash
8 import dash_core_components as dcc

```

```

9 from dash.dependencies import Input, Output
10 import dash_html_components as html
11 import dash_table
12
13 # Importierung weiterer Bibliotheken zur Portfolioanalyse
14 from datetime import datetime
15 import mysql.connector
16 import pandas as pd
17 import plotly.graph_objects as go

```

Zu Beginn des zweiten Skripts soll das aktuelle Portfolio abgebildet werden. Um dieses darzustellen, werden der Name der Aktie, die zugehörige ISIN sowie das früheste Datum, zu dem die Aktie gehandelt wurde, aggregiert (29). Weiter werden die ersten beiden Buchstaben der ISIN zur Identifikation der geographischen Allokation des Wertpapiers² sowie die gespeicherte zugehörige Industrie genutzt (30). Ergänzend werden schließlich noch der prozentuale Portfoliobestand sowie die prozentuale Portfolioentwicklung seit Erstinvestition bestimmt. Hierfür dienen zunächst eine leere Spalte als Platzhalter (30) sowie der zuletzt verfügbare Handelsschlusskurs der Aktie (31–32). Da sich diese Informationen aus verschiedenen Tabellen zusammensetzen, werden die Tabellen `portfolio_transactions` und `statics` mit dem Befehl `join` in eine gemeinsame Tabelle überführt (35). Weiterhin

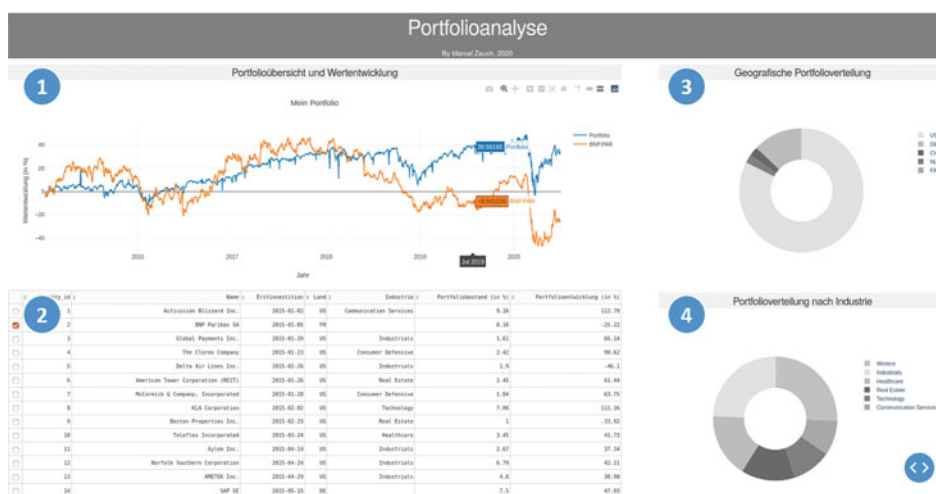


Abb. 16.2 Darstellung der zu implementierenden Benutzeroberfläche

² Zwar wurde in den vorigen Abschnitten die Bereitstellung des Landes eines Wertpapiers über die API von Alpha Vantage besprochen und diese Information in die Datenbank gespeichert. Für das vorgestellte Musterportfolio jedoch konnten nur amerikanischen Aktien das zugehörige Land über die Schnittstelle zugeordnet werden. Der hier implementierte Ansatz zeigt entsprechend einen zusätzlichen, alternativen Weg der Umsetzung auf.

sollen nur Wertpapiere betrachtet werden, für die überhaupt ein Ticker über die API zugeordnet werden konnte (36), da andernfalls keine weiteren (Markt-)Daten zur Auswertung zur Verfügung stehen. Gleichzeitig soll eine Portfolioübersicht erstellt werden, die genauso viele Zeilen aufweist, wie aktuell Wertpapiere im Portfolio gehalten werden. Da die Ausgangstabelle `portfolio_transactions` im Allgemeinen mehrere Zeilen pro Wertpapier enthält, wird diese nach der Security ID gruppiert (37). Für jede Security ID wird hierbei das aktuell gehaltene Nominal bestimmt (33). Schließlich werden nur Wertpapiere angezeigt, die im aktuellen Depot gehalten werden, deren aktuell gehaltene Stückzahl also positiv ist (38). Die zugehörige Datenbankabfrage ergibt sich entsprechend wie nachstehend:

Listing 16.27 Darstellung des aktuellen Portfolios in MySQL

```

27 # Abfrage an die Datenbank: Ausgabe des aktuellen Portfolios. Speicherung im
    DataFrame 'df_portfolio'
28 sql_select = """select * from
29     (select s.id, s.ticker, s.name, s.isin, min(p.trade_date) as '
        invested_since',
30     LEFT(isin,2) as 'country', s.industry, null,
31     round((select m.close from marketdata_daily m where m.security_id = p.
        security_id
32     order by m.date desc limit 1),2) as 'portfolio_value',
33     sum(if(p.trade_direction=1,1,-1)*p.nominal) as 'nominal_sum'
34     from portfolio_transactions p
35     join statics s on s.id = p.security_id
36     where s.ticker is not null
37     group by p.security_id) cp
38     where cp.nominal_sum > 0"""
39 cursor.execute(sql_select)
40 df_portfolio = pd.DataFrame(cursor.fetchall())
41
42 # Entsprechende Benennung der Spalten des DataFrame
43 df_portfolio.columns = ['security_id', 'Ticker', 'Name', 'ISIN', 'Erstinvestition', '
    Land', 'Industrie',
44     'Portfoliobestand (in %)', 'Portfolioentwicklung (in %)', '
        Summe Nominal']

```

Um den prozentualen Portfoliobestand jedes Wertpapiers bestimmen zu können, ist es notwendig, den gesamten Portfoliowert zu kennen. Diesen gibt die nachfolgende Query wieder. Hierbei wird das aktuelle Nominal jedes einzelnen Wertpapiers (48) mit dem jeweils letzten Börsenschlusskurs, bereinigt um den potenziellen Wechselkurs, multipliziert (49–51) und als `portfolio_value` deklariert. Schließlich werden die einzelnen Portfoliowerte zu einem Gesamtwert aufsummiert (47).

Listing 16.28 Wert des gesamten Portfolios zum letzten Handelstag

```

46 # Abfrage an die Datenbank: Aktueller Gesamtwert des Portfolios. Speicherung in der
    Variablen 'portfolio_value'
47 sql_select = """select sum(portfolio_value) from

```

```

48         (select (sum(if(p.trade_direction=1,1,-1)*p.nominal))*
49         (select round(if(s.currency='USD',m.close/mf.close,m.close),2) from
            marketdata_daily m
50         join marketdata_fx_daily mf on mf.date=m.date
51         where m.security_id = p.security_id order by m.date desc limit 1) as '
            portfolio_value'
52         from portfolio_transactions p
53         join statics s on s.id = p.security_id
54         where s.ticker is not null
55         group by p.security_id) pv"""
56 cursor.execute(sql_select)
57 portfolio_value = cursor.fetchone()[0]

```

In ähnlicher Weise, genauer als Kombination aus den beiden Queries zuvor, können auch die Portfolioallokation nach Land sowie die Portfolioallokation nach Industrie in MySQL zusammengestellt werden. Die entsprechenden Tabellen werden ausgelesen und später lediglich in geeigneter Form in einer Grafik wiedergegeben. Da beide Abfragen analog aufgebaut sind, soll hier nur auf die Portfolioallokation nach Land dargestellt werden. Die zweite Abfrage findet sich entsprechend im bereitgestellten Programmiercode zu diesem Buch.

Listing 16.29 Darstellung der Portfolioallokation nach Land

```

59 # Abfrage an die Datenbank: Sortierung des Portfolios nach Land. Speicherung im
    DataFrame 'df_country'
60 sql_select = """select cp.country, sum(cp.portfolio_value) as 'total_portfolio_value'
    from
61         (select s.name, s.isin, min(p.trade_date) as 'invested since', LEFT(
            isin,2) as 'country', s.industry,
62         (sum(if(p.trade_direction=1,1,-1)*p.nominal))*
63         (select round(if(s.currency='USD',m.close/mf.close,m.close),2) from
            marketdata_daily m
64         join marketdata_fx_daily mf on mf.date=m.date
65         where m.security_id = p.security_id order by m.date desc limit 1) as '
            portfolio_value',
66         sum(if(p.trade_direction=1,1,-1)*p.nominal) as 'nominal_sum'
67         from portfolio_transactions p
68         join statics s on s.id = p.security_id
69         where s.ticker is not null
70         group by p.security_id) cp
71         where cp.nominal_sum > 0
72         group by cp.country
73         order by total_portfolio_value desc
74         limit 5"""
75 cursor.execute(sql_select)
76 df_country = pd.DataFrame(cursor.fetchall())

```

Diese Abfrage gibt die fünf Länder wieder, die mit dem größten Volumen im Portfolio vertreten sind. Alle restlichen Länder werden in Python über die Differenz zum gesamten Portfoliowert aggregiert (81).

Listing 16.30 Zusammenfassen übriger Länder mit dem entsprechenden Volumen

```

78 # Gegebenenfalls Hinzufuegen fehlender Laender und dessen ausmachender Portfoliowert
79 if len(df_country) != len(df_portfolio.groupby('Land')):
80
81     csum = pd.DataFrame({0: ['Weitere'], 1: [round(portfolio_value - df_country[1].
82               sum(), 2)]}, index=[5])
83     df_country = pd.concat([df_country, csum])

```

Alternativ kann die Gruppierung der restlichen Länder auch über die MySQL-Datenbank zusammengestellt werden. Hierbei ist es wichtig zu beachten, dass die Gruppierung jedoch nur fehlerfrei erfolgen kann, wenn überhaupt mehr als fünf Länder im Portfolio vertreten sind. Dieser Abgleich wird im bereitgestellten Code durch die `if`-Anweisung in Zeile 79 sichergestellt.

Anschließend an diese Überlegungen soll nun die komplexere Konstruktion der Wertentwicklung des Portfolios diskutiert werden. Vor der eigentlichen Umsetzung in Python wird diese jedoch zunächst kurz theoretisch erörtert. Die Idee ist es, jeder Portfoliotransaktion den zugehörigen, aktuellen Portfoliowert zuzuordnen. Hierzu wird jeder Portfoliotransaktion die Anzahl der zu diesem Zeitpunkt gehaltenen Wertpapiere sowie deren errechneter Durchschnittspreis zugeordnet. Ersteres kann über die MySQL-Datenbank aufbereitet werden, indem die gehandelten Nominal, gewichtet nach Handelsrichtung, kumuliert werden. Anschließend wird für jede jemals gehandelte Aktie über ein Summenprodukt der Durchschnittspreis berechnet. Um verschiedene Facetten in diesem Anwendungsbeispiel abzudecken, sollen die Handelsrichtungen insbesondere dahingehend unterschieden werden, dass der durchschnittlich gehandelte Preis einer Aktie im Depot bei einem Verkauf dieser Aktie bestehen bleibt, allein die Anzahl der gehaltenen Aktien zu diesem Preis soll variieren. Nachdem jeder Aktie das kumulierte Nominal sowie der entsprechende Durchschnittspreis zugeordnet wurde, wird der Datensatz nach dem Datum gruppiert. Für jedes Datum wird das Summenprodukt über alle gehaltenen Aktien, deren Anzahl und Preis zu diesem Tag gebildet. Das Ergebnis ist eine Datentabelle, die für jeden Tag seit Auflegung des Portfolios den entsprechend gehaltenen Wert des Portfolios enthält.

Die zugehörige Umsetzung soll anhand der nachfolgenden Codebeispiele granularer besprochen werden. Zu Beginn soll eine Liste aller Portfoliotransaktionen mit den entsprechend kumulierten Nominalen in der Datenbank bereitgestellt werden. Diese wird zur weiteren Verarbeitung im DataFrame `df_return` gespeichert. Die Implementierung zeigt Listing 16.31. Von besonderem Interesse sind hierbei die Zeilen 113 und 114. Jedem Wertpapier wird hierdurch das bis zum jeweiligen Handelstag kumulierte Nominal zugeordnet.

Listing 16.31 Bereitstellung des kumulierten Nominals in einem DataFrame

```

110 # Abfrage an die Datenbank: Ausgabe aller Portfoliotransaktionen mit entsprechend
      kumuliertem Nominal.
111 # Speicherung im DataFrame 'df_return'
112 sql_select = """select pl.security_id, pl.trade_date, pl.trade_direction, pl.nominal,
      pl.trade_price,

```

```

113         (select sum(if(p0.trade_direction=1,1,-1)*p0.nominal) from
            portfolio_transactions p0
114         where p0.security_id=p1.security_id and p0.trade_date<=p1.trade_date) as
            'cumulative_nominal'
115         from portfolio_transactions p1""
116 cursor.execute(sql_select)
117 df_return = pd.DataFrame(cursor.fetchall())
118
119 # Entsprechende Benennung der Spalten des DataFrame
120 df_return.columns = ['security_id', 'trade_date', 'trade_direction', 'nominal', '
            trade_price', 'cumulative_nominal']
121
122 # Speicherung des kumulierten Nominals als Zahl, um Rechenoperationen hiermit
            ausfuehren zu koennen
123 df_return['cumulative_nominal'] = pd.to_numeric(df_return['cumulative_nominal'])

```

Als nächstes werden in einem zweiten DataFrame `df_sec` alle jemals gehandelten Wertpapiere in Form der Security ID gespeichert. Eine noch leere, zweite Spalte der ausgegebenen Tabelle wird zur Speicherung der berechneten Durchschnittspreise genutzt und entsprechend mit der Bezeichnung `cumulative_price` deklariert (132).

Listing 16.32 Ausgabe aller jemals gehandelten Wertpapiere

```

125 # Abfrage an die Datenbank: Ausgabe aller Security ID, die jemals gehandelt wurden.
            Speicherung im
126 # DataFrame 'df_sec'
127 sql_select = ""select security_id, null from portfolio_transactions group by
            security_id""
128 cursor.execute(sql_select)
129 df_sec = pd.DataFrame(cursor.fetchall())
130
131 # Entsprechende Benennung der Spalten des DataFrame
132 df_sec.columns = ['security_id', 'cumulative_price']

```

Bevor nun der kumulative Wertpapierpreis jedoch iterativ berechnet wird, werden die zwei folgenden leeren Listen definiert:

Listing 16.33 Definition der Listen `var` und `initial_date`

```

134 # Initialisierung der Liste 'var', in Anlehnung an das Wort 'variabel'. Jeder
            Listeneintrag wird spaeter
135 # Marktdateninformationen zu einer Security ID beinhalten
136 var = [[]]*(max(df_sec['security_id'])+1)
137
138 # Initialisierung der Liste 'initial_date'. Jeder Listeneintrag wird spaeter jeweils
            den Handelstag der Erstinvestition
139 # einer Security ID beinhalten
140 initial_date = [[]]*(max(df_sec['security_id'])+1)

```

Jeder Listeneintrag in `var` wird später einem eigenen DataFrame entsprechen. Dieser enthält die gesamte Information der Portfolioentwicklung jeweils einer Aktie seit der Erstinvestition. Die Liste `initial_date` hingegen wird das jeweilige Datum der Erstinvestition in eine Aktie enthalten. Die Länge der Listen orientiert sich dabei bewusst an dem betragsmäßig größten Wert aus `df_sec` und nicht an dessen Länge. So kann in jedem Fall sichergestellt werden, dass über die jeweilige Security ID auf die zu dieser Aktie gehörenden Daten zugegriffen werden kann.

Darauf aufbauend soll nun iterativ die Berechnung des durchschnittlichen Handelspreises bzw. damit die Berechnung der Portfolioentwicklung implementiert werden.

Listing 16.34 Initialisierung der Berechnung der historischen Portfolioentwicklung

```

142 # Berechnung der historischen Portfolioentwicklung. Durchfuehrung der Iteration nach
    Security ID
143 for i in range(0, len(df_sec)):
144
145     # Speicherung der jeweils zu iterierenden Security ID in der Variablen 'sec_id'
146     sec_id = df_sec.loc[i, 'security_id']
147
148     # Erstellung eines temporaeren DataFrame 'dff'. Dieser entspricht dem DataFrame '
    df_return' restringiert auf die
149     # jeweils zu iterierende Security ID 'sec_id'
150     dff = df_return[df_return['security_id'] == sec_id]
151
152     # Speicherung der Indexwerte des DataFrame 'dff' in der Variablen 'index'
153     index = list(dff.index.values)
154
155     # Speicherung des Handelstags der Erstinvestition der jeweils zu iterierenden
    Security ID 'sec_id'
156     initial_date[sec_id] = str(df_return.loc[min(index), 'trade_date'])

```

Wie zuvor beschrieben, sollen alle jemals gehandelten Wertpapiere, gespeichert im DataFrame `df_sec`, iterativ durchlaufen werden (143). Die entsprechende Security ID der jeweiligen Iteration wird in der Variable `sec_id` gespeichert (146). Der DataFrame `dff` beschreibt die Untermenge des DataFrame `df_return`, der nur die Daten dieser entsprechend aktuell zu iterierenden Security ID enthält (150). Alle Indexwerte dieser Datenmenge werden in einer separaten Liste namens `index` abgespeichert (153). In der Liste `initial_date` wird schließlich der aktuellen Security ID das Handelsdatum des niedrigsten Werts der Liste `index` zugeordnet (156). Dieses entspricht genau dem Handelstag, zu dem zum ersten Mal in das zugrundeliegende Wertpapier investiert wurde. Damit steht das Grundgerüst der ersten Schleife und aller grundlegenden Variablen, die zur Weiterverarbeitung und Umsetzung der Berechnung des durchschnittlichen Handelspreises benötigt werden. Eine zweite, innere Schleife durchläuft nun alle im DataFrame `dff` gespeicherten Handelstage zu dieser jeweiligen Security ID. Prinzipiell wird hierbei zwischen einem Kauf und einem Verkauf des Wertpapiers unterschieden, aus den bereits erklärten Gründen. Gleichzeitig dient eine erste `if`-Anweisung als eine Art Initialisierung. Hierbei ist keine Unterscheidung der Han-

delsrichtung notwendig, da es sich bei der ersten Transaktion eines Wertpapiers um einen Einkauf in das Depot handelt.

Listing 16.35 Initialisierung der Berechnung des kumulierten Portfoliowerts zu jedem Handelstag

```

160     for j in range(0, len(dff)):
161
162         # Erster Fall (Initialisierung): Handelstag der Erstinvestition
163         if index[j] == min(index):

```

Der kumulierte Handelspreis am Handelstag der Erstinvestition entspricht gerade dem Kaufpreis der Aktie. Entsprechend wird dieser als aktuell kumulierter Preis im DataFrame `def_sec` gespeichert.

Listing 16.36 Speicherung des kumulierten Handelspreises

```

168     df_sec.loc[i, 'cumulative_price'] = df_return.loc[index[j], 'trade_price']

```

Um später die Portfolioentwicklung eines Wertpapiers abbilden zu können, muss der Handelspreis einer Aktie in Relation zu den sich entwickelten Marktpreisen betrachtet und gegenübergestellt werden. Diese werden durch die nachfolgende Query innerhalb der Datenbank aggregiert und ausgelesen. Ausgangspunkt dieser Datenabfrage ist die Tabelle `marketdata_daily`. Diese wird mit den beiden weiteren Tabellen `marketdata_fx_daily` und `statics` verknüpft. Die `where`-Bedingung in den Zeilen 177 und 178 restringiert die Tabelle auf das entsprechend betrachtete Wertpapier und lässt lediglich alle Marktdaten ab Beginn der Erstinvestition in das Wertpapier zur Anzeige zu. Die Mitnahme der Tabelle `statics` in Zeile 176 ist notwendig, um die Währung, in der die Marktdaten des Wertpapiers gespeichert sind, zu erhalten. Im Fall der Währung USD soll der entsprechende Tageswechselkurs aus der Tabelle `marketdata_fx_daily` entnommen werden. Schließlich erfolgt die Speicherung der Tabelle im DataFrame `df_market`.

Listing 16.37 Bereitstellung der Marktdaten der zu iterierenden Security ID ab dem ersten Handelstag

```

173     sql_select = """select m.date, m.security_id, m.close, if(s.currency='EUR
174         ',1,mf.close),
175         null, null from marketdata_daily m
176         left join marketdata_fx_daily mf on mf.date = m.date
177         left join statics s on s.id = m.security_id
178         where m.security_id = """ + str(df_sec.loc[i, 'security_id'])
179         + \
180         " and m.date >= " + str(initial_date[sec_id]) + """
181         order by date desc"""
182     cursor.execute(sql_select)
183     df_market = pd.DataFrame(cursor.fetchall())
184
185     # Entsprechende Benennung der Spalten des DataFrame
186     df_market.columns = ['date', 'security_id', 'close', 'fx_close', '
187         portfolio_value', 'initial_value']

```

Die beiden letzten, noch leeren Spalten (174) dienen der Speicherung des tagesgenauen Portfoliowerts sowie des letzten Depotwerts der Aktie. Ersterer bestimmt sich als Produkt der zum jeweiligen Tag gehaltenen Anzahl der Aktien im Depot und des Tagesschlusskurses der Aktie, bereinigt um den Tagesschlusskurs der zugehörigen Währung.³ Der letzte Depotwert der Aktie berechnet sich hingegen im Allgemeinen als Produkt der zum jeweiligen Tag gehaltenen Anzahl der Aktien im Depot und des aktuell kumulierten Handelspreises. Da die Anzahl der gehaltenen Aktien im Depot per Definition bis zum nächsten Tag, an dem diese im Musterdepot gehandelt werden, unverändert bleibt, werden der tagesgenaue Portfolio- und der letzte Depotwert der Aktie direkt für alle zwischen den beiden Handelstagen liegenden Tage berechnet. Der Tag, an dem das Wertpapier das nächste Mal im Portfolio gehandelt wird, wird im Code durch den nachfolgenden, $j+1$ -ten, Eintrag in der Liste `index` repräsentiert. Die Implementierung der Berechnung des tagesgenauen Portfoliowerts ist in Listing 16.38 dargestellt.

Listing 16.38 Initialisierung und Berechnung des Portfoliowerts, vom ersten bis zum nächsten Handelstag

```

188     df_market.loc[(df_market['date'] >= df_return.loc[index[j], 'trade_date'
189                    ]) &
190                  (df_market['date'] < df_return.loc[index[j+1], 'trade_date'
191                    ]), 'portfolio_value'] = \
192     df_market['close'] / df_market['fx_close'] * \
193     df_return.loc[index[j], 'cumulative_nominal']

```

Analog erfolgt die Implementierung der Berechnung des letzten Depotwerts einer Aktie.

Listing 16.39 Initialisierung und Berechnung des letzten Depotwerts, vom ersten bis zum nächsten Handelstag

```

195     df_market.loc[(df_market['date'] >= df_return.loc[index[j], 'trade_date'
196                    ]) &
197                  (df_market['date'] < df_return.loc[index[j + 1], '
198                    trade_date'])), 'initial_value'] = \
199     df_sec.loc[i, 'cumulative_price'] / df_market['fx_close'] * \
200     df_return.loc[index[j], 'cumulative_nominal']

```

Mit Abschluss der Initialisierung der zweiten Schleife folgt nun die iterative Fallunterscheidung, ob eine Aktie zum entsprechenden Handelstag gekauft oder verkauft wurde. Dem Kauf einer Aktie kommt dabei im Sinn der Berechnung des kumulierten Handelspreises ein besonderes Interesse zu. Dieser bestimmt sich als Summenprodukt aus der Anzahl der zum Handelstag gekauften Aktien und deren Kaufpreis, dargestellt in Zeile 205, sowie der Anzahl der bereits im Depot gehaltenen Aktien und deren bisher kumulierten Handelspreis,

³ Im Allgemeinen ist darauf zu achten, dass der Handelskurs eines Wertpapiers und der entsprechende Marktpreis in der Datenbank nicht die gleiche Währung besitzen müssen. Das vorgestellte Musterportfolio ist zwar genau so konstruiert, bei abweichenden Währungen muss entsprechend die Implementierung jedoch vereinzelt adjustiert werden.

dargestellt in Zeile 206. Anschließend muss dieses Ergebnis noch durch die nun kumulierte Anzahl an Aktien im Depot geteilt werden, um den entsprechend neuen kumulierten Handelspreis zu erhalten.

Listing 16.40 Berechnung des kumulierten Handelspreises im Fall eines Zukaufs des Wertpapiers

```

200      # Zweiter Fall: Handelstage, an denen ein Kauf der zu iterierenden Security
      stattgefunden hat
201      elif df_return.loc[index[j], 'trade_direction'] == 1:
202
203          # Aktualisierung des kumulierten Handelspreises. Speicherung im DataFrame
            'df_sec'
204          df_sec.loc[i, 'cumulative_price'] = \
205              (df_return.loc[index[j], 'nominal'] * df_return.loc[index[j],
            'trade_price'] +
206              df_return.loc[index[j-1], 'cumulative_nominal'] * df_sec.loc[i,
            'cumulative_price']) / \
207              (df_return.loc[index[j], 'nominal'] + df_return.loc[index[j-1],
            'cumulative_nominal'])

```

Hiermit stellt der Code in Listing 16.40 das Äquivalent zum Codeabschnitt in Listing 16.36 dar. Nicht nur analog, sondern identisch zu Listing 16.38 und 16.39 erfolgen nach der Berechnung des kumulierten Handelspreises die Berechnungen des Portfoliowerts und des letzten Depotwerts. An dieser Stelle muss jedoch zusätzlich unterschieden werden, ob es sich bereits um den letzten Handelstag der zu iterierenden Security ID handelt. Diese Überprüfung erfolgt durch die `if`-Anweisung in Zeile 210. Die Implementierung in diesem Fall entspricht dabei weiterhin Listing 16.38 und 16.39, lediglich ohne die Restriktion auf einen nächsten Handelstag (189; 196).

Listing 16.41 Berechnung des Portfoliowerts und des letzten Depotwerts im Fall des letzten Handelstags des Wertpapiers

```

209      # Unterscheidung, ob es sich um den letzten Handelstag der zu
            iterierenden Security handelt
210      if index[j] == max(index):
211
212          # Berechnung des Portfoliowerts ab dem letzten Handelstag bis zum
            aktuellen Datum
213          df_market.loc[df_market['date'] >= df_return.loc[index[j], '
            trade_date'],
            'portfolio_value'] = \
214              df_market['fx_close'] / df_market['fx_close'] * df_return.loc[index[
            j], 'cumulative_nominal']
215
216          # Berechnung des originalen Portfoliowerts ab dem letzten Handelstag
            bis zum aktuellen Datum
217          df_market.loc[df_market['date'] >= df_return.loc[index[j], '
            trade_date'], 'initial_value'] = \
218              df_sec.loc[i, 'cumulative_price'] / df_market['fx_close'] * \
219

```

```
220 df_return.loc[index[j], 'cumulative_nominal']
```

Der dritte mögliche Fall, dass ein Wertpapier zu einem vorgegebenen Handelstag stattdessen verkauft wurde, ist einfach implementiert. Der kumulierte Handelspreis entspricht hierbei genau dem bisherigen Handelspreis, lediglich die Anzahl der gehaltenen Aktien im Depot verändert sich. Die Berechnungen des Portfoliowerts und des letzten Depotwerts entsprechen dann genau der in Listing 16.38, 16.39 und 16.41 vorgestellten Implementierung und können im online zur Verfügung gestellten Skript nachvollzogen werden.

Nachdem der aktuell zu iterierenden Security ID schließlich für jeden Tag die entsprechenden Handelspreise, Portfolio- und letzten Depotwerte zugeordnet wurden, wird nun noch der aktuelle Portfoliobestand (in %) sowie die Portfolioentwicklung seit Erstinvestition (in %) der jeweiligen Security ID im DataFrame `df_portfolio` gespeichert.

Listing 16.42 Speicherung des DataFrame `df_portfolio`

```
268 try:
269     # Bestimmung des aktuellen Portfoliobestands (in %) der zu iterierenden
        Security 'sec_id'. Speicherung im
270     # DataFrame 'df_portfolio'
271     df_portfolio.loc[df_portfolio['security_id'] == sec_id, 'Portfoliobestand (in
        %)' ] = \
272         round(df_market.loc[0, 'portfolio_value'] / portfolio_value * 100, 2)
273
274     # Bestimmung der Portfolioentwicklung seit Erstinvestition (in %) der zu
        iterierenden
275     # Security 'sec_id'. Speicherung im DataFrame 'df_portfolio'
276     df_portfolio.loc[df_portfolio['security_id'] == sec_id, 'Portfolioentwicklung
        (in %)' ] = \
277         round((df_portfolio.loc[df_portfolio['security_id'] == sec_id, '
        Portfolioentwicklung (in %)' ] /
278             df_sec.loc[i, 'cumulative_price']-1) * 100, 2)
279 except:
280     pass
```

Anschließend werden die berechneten Portfolio- und letzten Depotwerte noch in der Liste `var` sowie im DataFrame `df_dash` gespeichert. Die Daten jeder Security ID entsprechen dabei einem Listenelement in `var`. Der DataFrame `df_dash` hingegen wird mit jeder Iteration um die Portfolio- und letzten Depotwerte der jeweils nachfolgenden Security ID erweitert und enthält schlussendlich die berechneten Daten aller durchlaufenden Security ID. Damit schließt die in Zeile 143 geöffnete Schleife.

Listing 16.43 Initialisierung und Speicherung der berechneten Portfolio- und letzten Depotwerte

```
284 if i == 0:
285
286     var[sec_id] = df_market[['date', 'portfolio_value', 'initial_value']]
287     df_dash = df_market[['date', 'portfolio_value', 'initial_value']]
288
```

else:

```

# Speicherung der berechneten Portfoliowerte in der Liste 'var' und dem
  DataFrame 'df_dash'. Dieser wird
# iterativ erweitert
var[sec_id] = df_market[['date', 'portfolio_value', 'initial_value']]
df_dash = pd.concat([df_dash, df_market[['date', 'portfolio_value', '
  initial_value']]])

```

Während die Liste `var` die Portfolioentwicklung jeder einzelnen Aktie separat enthält, soll `df_dash` die Gesamtentwicklung des Portfolios widerspiegeln. Hierzu werden die einzelnen Einträge des DataFrame nach dem Handelsdatum gruppiert; zu jedem Tag werden die Portfolio- sowie die Depotwerte der einzelnen Aktien summiert (297). Zu jedem Tag enthält `df_dash` nun also einerseits den tagesaktuellen Markt- bzw. Portfoliowert und andererseits den Depotwert.

Listing 16.44 Datengruppierung im DataFrame `df_dash`

```

# Gruppierung des DataFrame 'df_dash' nach Datum. Die gespeicherten Portfoliowerte
  werden fuer jedes Datum summiert
df_dash = df_dash.groupby('date').sum()

```

Bevor die eigentliche App in Code gegossen werden kann, werden an dieser Stelle noch die weiteren grafischen Elemente der zu programmierenden Benutzeroberfläche final vorbereitet. Dazu werden in den Zeilen 310 bis 316 die Ringdiagramme zur Darstellung der Portfolioallokation nach Land und Industrie implementiert. Hierzu wird die Funktion `Figure` aus der Bibliothek `plotly.graph_objects` genutzt. Bei der geometrischen Grundfigur handelt es sich zunächst um ein Kreisdiagramm. Der Parameter `hole` sorgt durch die Festlegung eines Innenkreisradius für die Überführung in ein Ringdiagramm. Weiter werden die darzustellenden Werte und Beschriftungen, die zu verwendenden Farben und der Name der Grafik als Eingabeparameter definiert. Der vorgestellte Code in Zeile 312, respektive Zeile 316, bestimmt, welche Informationen beim Schweben („hovern“) des Mauszeigers über die Diagramme angezeigt werden sollen. Schließlich definiert der DataFrame `df_table_pf` als Teilmenge des zu Beginn eingeführten DataFrame `df_portfolio` (28–44) die Tabelle aller aktuell im Depot gehaltenen Wertpapiere, die in der zu implementierenden Benutzeroberfläche dargestellt werden sollen.

Listing 16.45 Vorbereitung der Daten zur grafischen Darstellung innerhalb der App

```

# Speicherung der Laender und deren zugehoeriger Portfoliowerte in den Variablen '
  clabels' und 'cvalues'
clabels = df_country[0]
cvalues = df_country[1]

# Speicherung der Industrien und deren zugehoeriger Portfoliowerte in den Variablen '
  ilabels' und 'ivalues'
ilabels = df_industry[0]

```

```

305 ivalues = df_industry[1]
306
307 # Festsetzung eines harmonisierten Farbschemas in Form der Liste 'colors'
308 colors = ['#E0E0E0', '#BCBCBC', '#696969', '#808080', '#A9A9A9', '#C0C0C0']
309
310 # Erstellung eines Ringdiagramms zur Darstellung der geografischen
    Portfolioverteilung
311 fig_country = go.Figure(data=[go.Pie(labels=clabels, values=cvalues, hole=.5,
    marker_colors=colors, name='Laender')])
312 fig_country.update_traces(hoverinfo='label+value+percent+name', textinfo='none')
313
314 # Erstellung eines Ringdiagramms zur Darstellung der Portfolioverteilung nach
    Industrie
315 fig_industry = go.Figure(data=[go.Pie(labels=ilabels, values=ivalues, hole=.5,
    marker_colors=colors, name='Industrie')])
316 fig_industry.update_traces(hoverinfo='label+value+percent+name', textinfo='none')
317
318 # Erstellung des DataFrame 'df_table_pf', der die aktuellen Portfoliopositionen in
    einer Tabelle abbildet
319 df_table_pf = df_portfolio[['security_id', 'Name', 'Erstinvestition', 'Land', '
    Industrie',
320                             'Portfoliobestand (in %)', 'Portfolioentwicklung (in %)']]

```

Schließlich kann die Implementierung der eigentlichen Benutzeroberfläche mit dash erfolgen. Github stellt hierzu ein erstes Repository an Apps zusammen,⁴ die als Vorlage zur eigenen Umsetzung dienen können. Das hier vorgestellte Tool zur Portfolioanalyse orientiert sich von der Struktur der Benutzeroberfläche her an der App Dash Wind Streaming.⁵ Zu Beginn der Umsetzung erfolgt die Speicherung externer Grafik- und Darstellungselemente (323). Die App wird daraufhin initialisiert (324–326) und ab Zeile 329 wird das Layout mit den entsprechenden Inhalten gesetzt. Hierzu soll erneut an das zu implementierende Layout erinnert werden. Dieses wird mit den unterschiedlichen Komponenten in Abb. 16.3 dargestellt.

Das erste Element bildet der Header der App; dieser wird als Überschrift in dem Format H1 festgesetzt. Kleinere Überschriften können mit aufsteigenden Zahlen umgesetzt werden. Durch die Funktion `textAlign` kann die Überschrift mittig gesetzt werden, die Schrift wird in diesem Fall mittels des Farbcodes `#FFFFFF` Weiß eingefärbt. Darüber hinaus kann mit dem Format `P` in HTML eine Unterüberschrift erzeugt werden, die analog der Überschrift parametrisiert werden kann. Die in Abb. 16.3 dargestellte graue Hinterlegung des Headers wird zu Ende der App in Zeile 428 umgesetzt.

⁴ <https://github.com/plotly/dash-sample-apps/tree/master/apps>, zuletzt abgerufen am 12. Februar 2022.

⁵ <https://github.com/plotly/dash-sample-apps/tree/master/apps/dash-wind-streaming>, zuletzt abgerufen am 12. Februar 2022. Der zugehörige Code findet sich unter <https://github.com/plotly/dash-sample-apps/blob/master/apps/dash-wind-streaming/app.py>, zuletzt abgerufen am 12. Februar 2022.

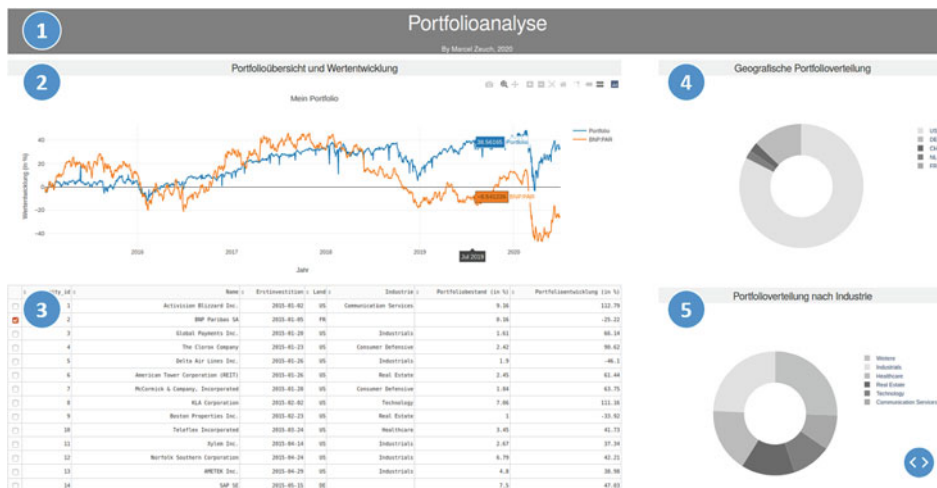


Abb. 16.3 Darstellung der unterschiedlichen HTML-Komponenten der zu implementierenden Benutzeroberfläche

Listing 16.46 App in dash - Header

```

322 # Initialisierung der Applikation in Dash. Speicherung externer Grafik- und
    Darstellungselemente in CSS
323 external_stylesheets = ['https://codepen.io/chriddyp/pen/bWLwgP.css']
324 app = dash.Dash(__name__, external_stylesheets=external_stylesheets)
325
326 server = app.server
327
328 # Festlegung des App-Layouts und Definition der Inhalte
329 app.layout = html.Div(
330     [
331         html.Div(
332             [
333                 html.Div(
334                     [
335                         html.H1("Portfolioanalyse", className="app__header__title",
336                             style={'textAlign': 'center', 'color': '#FFFFFF'}),
337                         html.P(
338                             "By Marcel Zeuch, 2020",
339                             className="app__header__title-grey",
340                             style={'textAlign': 'center', 'color': '#FFFFFF'}),
341                     ],
342                 ),
343                 className="app__header__desc",
344             ],
345         ),
346     ],
347     className="app__header",

```

347),

Das zweite Element bildet die Grafik, die die Portfolioentwicklung wiedergibt. Hier wird zunächst innerhalb des HTML-Elements die entsprechende Überschrift platziert (355–357). Als nächstes folgt das Einbinden des Grafikelements (358–360). Dieses kann direkt an dieser Stelle implementiert werden. In diesem vorgestellten Fall jedoch soll die Grafik später im Code über einen `app.callback` generiert werden. Hierdurch kann die Grafik interaktiv gestaltet werden, das heißt, die Grafik kann auf Eingaben des Nutzers der Benutzeroberfläche reagieren und entsprechend angepasst werden. Die festgelegte `id` innerhalb der Funktion `Graph` wird später der Identifikation des Grafikelements innerhalb des `app.callback` dienen.

Listing 16.47 App in dash - Grafik Portfolioentwicklung

```

352             html.Div(
353                 [
354                     html.Div(
355                         [html.H5("Portfoliouebersicht und Wertentwicklung
                                ", className="graph__title",
                                style={'textAlign': 'center', '
                                    backgroundColor': '#EFEFEF' })],
356                     ),
357                     dcc.Graph(
358                         id='portfolio'
359                     ),
360                     html.Div(id='graph-container')
361                 ],
362                 className="graph__container",
363             ),
364 
```

Das dritte Element bildet die Tabelle, welche das aktuelle Portfolio wiedergibt. Diese wurde bereits im DataFrame `df_table_pf` gespeichert (319–320). Um diese in die Benutzeroberfläche einzubinden, wird die Funktion `DataTable` aus der Bibliothek `dash_table` genutzt.⁶ Hier genutzte Eingabeparameter sind die zu verwendenden Spalten der Tabelle (369–371), die entsprechenden Daten (372) sowie Angaben zur Sortierung der Spalten und Selektierbarkeit der einzelnen Zeilen (373–375). Das Auswählen separater Zeilen wird es ermöglichen, später einzelne Wertpapiere gegeneinander und gegenüber dem Gesamtportfolio vergleichen zu können.

Listing 16.48 App in dash - Tabelle Portfolio

```

365             html.Div(
366                 [
367                     dash_table.DataTable(
368                         id='datatable-row-ids',
369                         columns=[

```

⁶ Siehe hierzu auch <https://dash.plotly.com/datatable>, zuletzt abgerufen am 12. Februar 2022.

```

370             {'name': i, 'id': i, 'selectable': True} for
371                 i in df_table_pf.columns
372         ],
373         data=df_table_pf.to_dict('records'),
374         sort_action='native',
375         sort_mode='multi',
376         row_selectable='multi',
377         html.Div(id='datatable-row-ids-container')
378     ],
379     className="graph__container",
380 ),

```

Die letzten beiden Elemente bilden die beiden Ringdiagramme, welche die Portfolioverteilung nach Land und Industrie grafisch abbilden. Beide Elemente bestehen jeweils aus einer Überschrift (390–394; 408–411) sowie der Funktion `Graph`, welche die zuvor definierten Figuren `fig_country` und `fig_industry` einbindet (399; 416). Anschließend werden die einzelnen, bis hierhin diskutierten HTML-Elemente nacheinander geschlossen; die grafische Benutzeroberfläche ist damit implementiert.

Listing 16.49 App in dash - Portfolio nach Land/Industrie

```

384         html.Div(
385             [
386                 html.Div(
387                     [
388                         html.Div(
389                             [
390                                 html.H5(
391                                     "Geografische Portfolioverteilung",
392                                     className="graph__title",
393                                     style={'textAlign': 'center', '
394                                         backgroundColor': '#EEEEEE'}
395                                 )
396                             ]
397                         ),
398                         dcc.Graph(
399                             id='country',
400                             figure=fig_country
401                         ),
402                     ],
403                     className="graph__container first",
404                 ),
405                 html.Div(
406                     [
407                         html.Div(
408                             [
409                                 html.H5(

```

```

409         "Portfolioverteilung nach Industrie",
410         className="graph__title",
411         style={'textAlign': 'center', '
412             backgroundColor': '#EEEEEE'}
413     )
414     ],
415     dcc.Graph(
416         id='industry',
417         figure=fig_industry
418     ),
419     ],
420     className="graph__container second",
421 ),
422 ],
423     className="one-third column overview__direction",
424 ),
425 ],
426     className="app__content",
427 ),
428 ],
429     className="app__container", style={'backgroundColor': colors[3]},

```

Um die Grafik der Portfolioentwicklung dynamisch zu gestalten, fehlt noch die Implementierung der Funktion `app.callback`, die im Folgenden diskutiert werden soll. Hierzu werden zu Beginn die notwendigen Eingabe- (436) und die entsprechenden Ausgabeparameter (435) festgelegt. Im vorliegenden Fall soll die Funktion die selektierten Zeilen der Tabelle des aktuellen Portfolios entgegennehmen und hierzu die Figur bzw. die Grafik ausgeben, die zur Identifikation `portfolio` korrespondiert.

Die konkrete Festlegung der Aktualisierung der Grafik wird schließlich über die selbst zu implementierende Funktion `update_graph` definiert. Sind keine Zeilen selektiert, wird dieser Eingabeparameter als leere Liste gesetzt (442–444). Andernfalls werden alle ausgewählten Security ID in der Liste `securities` gespeichert (446).

Listing 16.50 Initialisierung des `app.callback`

```

434 @app.callback(
435     Output('portfolio', 'figure'),
436     [Input('datatable-row-ids', 'selected_rows')]
437 )
438 def update_graph(selected_rows):
439
440     # Initialisierung der Variable 'selected_rows'. Diese beinhaltet die jeweils
441     # selektierten Zeilen
442     # der Portfoliouebersicht
443     if selected_rows is None:

```



```

444         selected_rows = []
445
446         securities = [df_table_pf.loc[i, 'security_id'] for i in selected_rows]

```

Nach dieser Initialisierung des `app.callback` gibt es im Wesentlichen zwei Fälle zu unterscheiden: Entweder sind keine Zeilen ausgewählt, in diesem Fall soll lediglich die prozentuale Entwicklung des Gesamtportfolios grafisch dargestellt werden. Oder es ist mindestens eine Zeile der Portfolioübersicht ausgewählt, dann sollen neben der Entwicklung des Gesamtportfolios auch die entsprechenden Entwicklungen der jeweils ausgewählten Aktien grafisch wiedergegeben werden.

Im ersten Fall erfolgt die Erstellung der Liniengrafik über die Eingabe der darzustellenden Daten und die Parametrisierung des zugehörigen Layouts. Als Werte der horizontalen Achse der Grafik dient das jeweilige Datum (455), auf der vertikalen Achse der Grafik dahingegen soll die prozentuale Entwicklung des Gesamtportfolios dargestellt werden (455). Die prozentuale Entwicklung errechnet sich dabei jeweils aus dem Quotienten des Portfoliowerts und des Depotwerts (455). Um die eingegebenen Werte als Liniengrafik wiederzugeben, dient die Festlegung des Typs als Linie in Zeile 456. Schließlich wird noch der Name der Grafik festgelegt. Innerhalb des Dictionary layout werden in diesem Fall der Titel der Grafik festgelegt (459), die Achsen beschriftet (460–461) sowie die Hintergrundfarbe der Grafik ausgewählt (462–463).

Listing 16.51 Darstellung des Gesamtportfolios innerhalb des `app.callback`

```

450     if selected_rows == []:
451
452         # Erstellung der Liniengrafik
453         figure = {
454             'data': [
455                 {'x': df_dash.index, 'y': (df_dash['portfolio_value'] / df_dash['
456                     initial_value'] - 1)*100,
457                 'type': 'line', 'name': 'Portfolio'}
458             ],
459             'layout': {
460                 'title': 'Mein Portfolio',
461                 'xaxis': {'title': 'Jahr'},
462                 'yaxis': {'title': 'Wertentwicklung (in %)'},
463                 'plot_bgcolor': '#FFFFFF',
464                 'paper_bgcolor': '#FFFFFF',
465             }
466         }

```

Die Implementierung des zweiten Falls gestaltet sich von der Idee der Umsetzung sehr ähnlich. Einige Besonderheiten sollen im Folgenden diskutiert werden.

Um die Wertentwicklung einzelner Wertpapiere gegeneinander und gegenüber dem Gesamtportfolio vergleichen zu können, ist es notwendig, die unterschiedlichen Datensätze zu normieren. Genauer: Die Wertentwicklung mehrerer Wertpapiere soll ab einem

gemeinsamen Datum erfolgen. Hierzu bietet sich das letzte Datum der verschiedenen Erstinvestitionen der Wertpapiere an. Dieses wird in Zeile 473 bestimmt und anschließend entsprechend formatiert (476). Andererseits ist es für einen Vergleich der Wertentwicklungen notwendig, dass alle Wertpapiere zum Startzeitpunkt bei einer Wertentwicklung von Null beginnen, das heißt, um ihre bisherige Wertentwicklung verschoben werden. Die entsprechende Umsetzung findet sich in den Zeilen 483 bis 484 sowie in den Zeilen 503 bis 506.

In dem hier vorgestellten Tool zur Portfolioanalyse soll bei jeder Auswahl von Wertpapieren auch immer das Gesamtportfolio mit angezeigt werden. Die Liste `data`, die die darzustellenden Daten der Grafik beinhaltet, wird aus diesem Grund zunächst mit den Daten des Gesamtportfolios initiiert (480–485).

Listing 16.52 Initialisierung der Darstellung einzelner, selektierter Wertpapiere innerhalb des `app.callback`

```

467     # Fall 2: Mindestens eine Zeile der Portfoliouebersicht ist ausgewaehlt
468     else:
469
470         # Bestimmung des letzten Erstinvestitionstags ueber alle ausgewaehnten
471         # Security ID hinweg. Erst ab
472         # diesem Handelstag werden die Wertentwicklungen der unterschiedlichen
473         # Security ID gegeneinander verglichen.
474         # Speicherung des Handelstags in der Variablen 'init'
475         init = max([initial_date[i] for i in securities])
476         init = datetime.strptime(init, '%Y-%m-%d').date()
477
478         # Berechnung der Wertentwicklung des gesamten Portfolios seit dem Handelstag
479         # 'init'. Speicherung im
480         # Listenelement 'data'
481         data = [{ 'x': df_dash[df_dash.index >= init].index,
482                   'y': ((df_dash.loc[df_dash.index >= init, 'portfolio_value'] /
483                         df_dash.loc[df_dash.index >= init, 'initial_value']) - 1) *
484                         100 -
485                   (((df_dash.loc[df_dash.index == init, 'portfolio_value'] /
486                     df_dash.loc[df_dash.index == init, 'initial_value']) - 1) *
487                     100)[0],
488                   'type': 'line', 'name': 'Portfolio' }]

```

Für jedes vom Benutzer ausgewählte Wertpapier werden anschließend die entsprechenden Daten über die Security ID aus der Liste `var` entnommen, aufbereitet und der Liste `data` hinzugefügt (500–507). Dieses Vorgehen ist über die Iteration aller ausgewählten Security ID umgesetzt (488). Schließlich wird die anzuzeigende Liniengrafik analog zum Codeausschnitt in Listing 16.51 erstellt. Die Ausgabe der Grafik an die implementierte Benutzeroberfläche erfolgt dann mit dem Befehl `return` (522).

Um wie in Abb. 16.3 dargestellt, eine passende Legende der einzelnen Daten bereitzustellen, bietet sich der Ticker des jeweiligen Wertpapiers an. Der Konstruktion der Tabelle `df_portfolio` geschuldet ist eine relativ umständliche Speicherung des jeweiligen Tickers nötig (491). Die implementierte Vorgehensweise erschließt sich bei Ansicht der nachfolgenden Konsolenausgabe in Listing 16.53.

Listing 16.53 Konsolenausgabe: `print(df_portfolio.loc[df_portfolio['security_id'] == i, 'Ticker'])`

```
1 1    BNP.PAR Name: Ticker, dtype: object
```

Wird diese Zeichenkette zunächst anhand der Leerzeichen aufgeteilt, so verbleiben bei Zugriff auf das fünfte Element (491) der Ticker und die Zeichenkette „Name“. Diese wiederum lassen sich durch die Trennung der gesamten Zeichenkette am Zeilenumbruch separieren, sodass Zeile 491 die Speicherung des Ticker in der Variable `name` umsetzt.

Das zweite Skript und damit der vorliegende Kapitelabschnitt schließlich endet mit der Aufforderung, das Skript kontinuierlich zu durchlaufen (525–526), um so die Interaktivität der Benutzeroberfläche gewährleisten zu können.

Listing 16.54 Fortsetzung der Darstellung einzelner, ausgewählter Wertpapiere innerhalb des `app.callback`

```
487     # Iteration nach selektierten Security ID
488     for i in securities:
489
490         # Speicherung des Ticker der jeweils zu iterierenden Security ID in der
491         # Variable 'name'
492         name = (str(df_portfolio.loc[df_portfolio['security_id'] == i, 'Ticker'])
493                 .split(' ')[4]).split('\n')[0]
494
495         # Ausgabe einer Listenlaenge in der Variablen 'count'. Diese ermoeoglicht
496         # den Zugriff auf den Wert
497         # der Wertentwicklung der zu iterierenden Security ID
498         count = len((str(((var[i].loc[var[i]['date'] == init, 'portfolio_value']
499                             / var[i].loc[
500                             var[i]['date'] == init, 'initial_value']) - 1) * 100).split(' ')))
501
502         # Berechnung der Wertentwicklung der zu iterierenden Security ID seit dem
503         # Handelstag 'init'.
504         # Speicherung in einem Listenelement, das der Liste 'data' hinzugefuegt
505         # wird
506         data = data + [{'x': var[i].loc[var[i]['date'] >= init, 'date'],
507                         'y': (((var[i].loc[var[i]['date'] >= init, '
508                                 portfolio_value'] /
509                                 var[i].loc[var[i]['date'] >= init, '
510                                 initial_value']) - 1) * 100)
511                         - pd.to_numeric(
512                         (str(((var[i].loc[var[i]['date'] == init, '
513                                 portfolio_value'] /
```

```

505         var[i].loc[var[i]['date'] == init, '
              initial_value'])
506         - 1) * 100).split(' ')[count-2]).split('\n')
              [0]),
507         'type': 'line', 'name': name]]
508
509     # Erstellung der Liniengrafik
510     figure = {
511         'data': data,
512         'layout': {
513             'title': 'Mein Portfolio',
514             'xaxis': {'title': 'Jahr'},
515             'yaxis': {'title': 'Wertentwicklung (in %)'},
516             'plot_bgcolor': '#FFFFFF',
517             'paper_bgcolor': '#FFFFFF',
518         },
519     }
520
521     # Ausgabe der erstellten Liniengrafik an die Applikation in Dash
522     return figure
523
524
525 if __name__ == '__main__':
526     app.run_server(debug=True)

```

16.6 Portfolioanalyse: Interpretation der Ergebnisse

Im vorliegenden Kapitel wurde ein Modell zur Portfolioanalyse und dessen Implementierung in Python diskutiert. Im Detail wurde erklärt, wie Schnittstellen im Internet, im konkreten Fall die Schnittstelle des Marktdatenanbieters Alpha Vantage, angesprochen werden und die Daten weiter verarbeitet werden können. Dazu wurde eine MySQL-Datenbank installiert, in der die Daten zum einen automatisiert eingelesen und zum anderen zur automatischen Weiterverarbeitung aufbereitet und ausgelesen wurden. Zwei mächtige Werkzeuge, die die Verarbeitung und grafische Aufbereitung in Python übernehmen, sind die Bibliotheken *pandas* und *dash*.

Während in diesem Buch eine grundlegende Variante eines Tools zur Portfolioanalyse besprochen wurde, so sind doch viele weitere Ergänzungen denkbar. Eine kurze Übersicht soll den Abschluss dieses Kapitels bilden.

- Neben den in diesem Kapitel genutzten Börsenschlusskursen stellt Alpha Vantage auch detailliertere Marktdaten bereit. Diese können bis auf 15 Minuten genau über die Schnittstelle angefragt und entsprechend granular weiterverarbeitet werden. Bei der Umsetzung eines solchen Detailgrads wäre es denkbar, das erste Skript, das die Datenanfrage über-

nimmt, automatisiert zu starten oder gegebenenfalls auch mehrmals über den Tag verteilt durchlaufen zu lassen.

- Ebenso lassen sich weitere Kennzahlen oder Handelsindikatoren über die Schnittstelle von Alpha Vantage anfragen und innerhalb des Programmiercodes zur Darstellung in der Benutzeroberfläche verarbeiten.
- Nicht nur Daten des Anbieters Alpha Vantage, sondern auch weitere Daten, die durch andere Marktdatenanbieter bereitgestellt werden, können ergänzend aufbereitet werden. Die vorgestellten Schritte und Codebeispiele in diesem Kapitel können hierzu als Blaupause für die Exploration weiterer Daten dienen.
- Aufbauend auf den drei erstgenannten Aspekten kann das in diesem Kapitel vorgestellte Tool nicht nur zur reinen Portfolioanalyse genutzt, sondern auch dahingehend ergänzt werden, Handelsentscheidungen zu unterstützen oder sogar Handelssignale für das eigene Portfolio zu generieren.
- Weiter könnten eigene Tradingstrategien auf den gespeicherten historischen Marktdaten sogar entwickelt oder validiert werden. Neben Handelssignalen für die bereits im Portfolio gehaltenen Wertpapiere könnten darüber hinaus Handelssignale für Wertpapiere auf einer zu definierenden Watchlist generiert werden.

Thematisches Lexikon

Data Science

Data Science

Data Science ist ein Sammelbegriff für die Vorverarbeitung, Analyse und Modellierung von Daten mithilfe moderner Methoden aus der Informatik mit dem Ziel, Erkenntnisse aus den Daten für verschiedene Zwecke zu gewinnen. Hierbei kommen in der Regel interdisziplinäre Teams zusammen, um das domänenspezifische und technologische Wissen entsprechend der Zielsetzung einzusetzen. Für mehr Information sei an dieser Stelle auf das Kap. 8 verwiesen.

Data Mining

Ein Teilbereich des Data Science ist das Data Mining. Zentraler Bestandteil des Data Mining ist die Gewinnung von Rohdaten für den Data-Science-Prozess. Hierbei können beispielsweise mittels sogenannter Crawler Programmcodes zum Auslesen von Informationen und Webseiten für die Informationsgewinnung abgefragt werden.

Daten

Verschiedene Arten von Daten spielen in der Informatik und insbesondere im Data-Science-Bereich eine besondere Rolle. Die Art der Datentypen definiert deren Menge und die möglichen Operationen. Zunächst lassen sich die folgenden grundlegenden Datentypen unterscheiden:

Nominale (kategorische) Daten beschreiben eine ungeordnete Datenmenge. Die definierte Relationen auf diese Daten ist die Gleichheitsrelation (=). Hierbei wären beispielsweise Namen von Dingen nominale Daten. Im Kontext der Arten von Daten

Ordinale Daten

wäre beispielsweise eine Spalte mit Strings, in einem DataFrame bestehend aus den Namen von Vornamen von Personen, eine Menge aus nominalen Daten.

sind geordnete Mengen, die auch als Tuple bezeichnet werden können. Auf dieser Datenmenge lassen sich drei Relationen definieren ($=$, $<$ und $>$). Ein Beispiel hierfür könnten Altersangaben in Form von Integer- oder Float-Werten von Personen sein. Diese lassen sich entsprechend ordnen.

Quantitative Daten

decken einen numerischen Bereich ab. Auf diesen Daten lassen sich alle bekannten arithmetischen Operationen durchführen. Damit sind folgende Operationen $=$, $<$ und $>$ möglich. Hierbei lassen sich noch diskrete und kontinuierliche quantitative Daten unterscheiden. Diskrete Daten sind im Wertebereich von ganzen Zahlen. Übertragen auf die Informatik sind das Integer. Kontinuierliche Daten weisen ein kontinuierliches Spektrum auf und lassen sich damit mit Nachkommazahlen, wie die Floater, beschreiben.

Data Pre-Processing**Data Cleaning**

beschreibt den Umgang mit fehlenden Werten (Missing Values) und ist kein triviales Unterfangen, weil der Umgang mit fehlenden Werten Einfluss auf die Analysen haben kann. Grundsätzlich sollten ersetzte Werte immer kenntlich gemacht werden, um dem Betrachter die Unterscheidung zwischen den realen und den gegebenenfalls synthetischen Daten zu ermöglichen. Beispielsweise können die fehlenden Werte wie folgt behandelt werden:

Globale Konstante

kann dazu dienen, Berechnungen, Modelle und Abbildungen nicht in ihrer Funktionsweise zu beeinträchtigen, indem für jeden fehlenden Wert eine globale Konstante (wie z. B. -1 , 0 oder 99999) eingesetzt wird. Allerdings sollte die Konstante eindeutig sein und nicht im Datensatz vorkommen.

Manuelle Korrektur

führt zum Austausch der fehlenden Werte durch einen Experten, indem dieser plausible Werte für die fehlenden Werte einfügt. Zwar kann dieses Verfahren sehr präzise sein, ist allerdings in Abhängigkeit des Datenumfangs äußerst aufwendig.

Mittelwerte	können dabei helfen, fehlende Werte zweckmäßig zu ersetzen. Allerdings gibt es hierbei verschiedene Ansätze die möglich sind, wie beispielsweise der gleitende Durchschnitt oder die Verwendung eines Mittelwerts auf Basis eines Teils der Daten.
Ignorieren	der Missing Values ist möglich und sicherlich die einfachste Lösung. Allerdings muss beachtet werden, dass dieses Vorgehen zum einem zum Informationsverlust führen kann. Zudem funktionieren unter Umständen Berechnungen, Modelle und Abbildungen mit fehlenden Werten nicht.
Interpolation	von Werten dient dazu, fehlende Werte zu ersetzen, indem der Missing Value aus der Verbindung zwischen vorherigem und folgendem Wert entsteht.
Pre-Processing	Die Vorverarbeitung der Daten (Pre-Processing) beschreibt eine erste Prüfung und Bereinigung der Daten. Hierbei werden fehlerhafte oder unvollständige Werte entfernt bzw. durch andere Platzhalter ersetzt. Dieser Schritt im Data-Science-Prozess nimmt in der Regel einen erheblichen Zeitbedarf in Anspruch und bedarf entsprechenden Domänenwissens, um die Daten geeignet aufzubereiten.
Missing Values	Die Rohdaten können fehlende Werte aufweisen. Dieser Umstand entsteht beispielsweise durch das Versagen von Sensoren zur Datenaufzeichnung oder durch einen temporären Verbindungsabbruch zwischen Sensor und Datenbank.
Not-a-Number	Die Kennzeichnung von Missing Values erfolgt in vielen Programmiersprachen durch Not-a-Number(NaN)-Einträge in den entsprechenden Datenstrukturen. Diese Einträge sind für die weiteren Analysen unbrauchbar und werden in der Regel entfernt oder durch andere Werte ersetzt. Ein Verfahren ist die Interpolation.

Machine-Learning-Modelle (eine kleine Auswahl)

Machine-Learning-Modelle Im Allgemeinen kann maschinelles Lernen als Sammelbegriff für Algorithmen betrachtet werden. Ziel ist es, Zusammenhänge in den Daten mit diesen Algorithmen abbilden zu können. Hierzu werden diese Algorithmen anhand eines Ausschnitts der Daten kalibriert (trainiert) und auf einem anderen Teil der Daten getestet. Zwei Verfahren im Rahmen des Trainings werden hierbei unterschieden, das überwachte und das unüberwachte Lernen. Beim überwachten Lernen sind die Trainingsdaten mit der Wahrheit gekennzeichnet. Das heißt, der Algorithmus kann seine Lösung mit der Wahrheit abglei-

	chen und gegebenenfalls seine Entscheidung anpassen. Bei unüberwachtem Lernen erhält der jeweilige Algorithmus die Möglichkeit, die Daten selbst in Gruppen einzuteilen, um übergeordnete Muster erkennbar zu machen.
Artificial Neural Network	ahmen die Funktionsweise der biologischen Vorlage nach. Künstliche neuronale Netze simulieren einen Verbund von Neuronen, die in der Regel in Schichten (Layer) angeordnet sind. Die einzelnen Neuronen empfangen und verarbeiten Informationen und geben das Ergebnis weiter.
Decision Tree	Entscheidungsbäume stellen Entscheidungsprozesse hierarchisch dar, indem sie entsprechende Variablen automatisch klassifizieren.
Logistische Regression	ist ein statistisches Verfahren, um Analysen mit binären abhängigen Variablen durchzuführen. Damit werden diese Verfahren eingesetzt, um die Beziehung zwischen einer oder einer Vielzahl von unabhängigen Variablen und einer abhängigen Variable zu beschreiben.
Naive Bayes	Dieser überwachte Klassifikator geht davon aus, dass die Werte der Merkmale angesichts der Klassenvariablen bedingt unabhängig sind. Das Modell basiert auf dem Bayes'schen Theorem, das die Wahrscheinlichkeit beschreibt, dass ein Ereignis eintritt, wenn zuvor Informationen über Bedingungen gegeben werden, die für das Ereignis relevant sein könnten.
Random Forests	kann als Klassifikationsverfahren verstanden werden, das aus mehreren Entscheidungsbäumen besteht, die während des Lernprozesses durch Zufallsterme wachsen. Damit ist der Random Forest ein Ensemble aus Entscheidungsbäumen, bei dem für die Klassifikation einer Klasse das Mehrheitsprinzip greift.
Support Vector Machine	ist ein Klassifikationsverfahren, bei dem Klassengrenzen in einem Raum festgelegt werden.

Model Training und Testing

Daten Aufteilung	ist ein notwendiger Schritt, um das Trainieren und Testen des Modells in geeigneter Form zu ermöglichen. Die Datenaufteilung (Data Split) zerlegt dabei die Daten in drei kleinere Datensätze, die in der Anzahl der Variablen identisch sind. Diese
-------------------------	--

Teildatensätze werden gemäß ihres Zwecks benannt (Trainings-, Validierungs- und Testdatensatz, „train, validation, test set“). Das prozentuale Verhältnis der Datenaufteilung kann je nach Verfügbarkeit der Daten variieren; beispielsweise können folgende Aufteilungsquoten zum Einsatz kommen:

- 60 % Training, 20 % Validierung, 20 % Test
- 70 % Training, 15 % Validierung, 15 % Test
- 80 % Training, 10 % Validierung, 10 % Test

Die Zerlegung des Ausgangsdatsatzes in Trainings-, Validierungs- und Testdatensatz kann auf mehrere Arten und Weisen erfolgen. Allerdings bietet sich die zufällige Einteilung der Datenzeilen in die drei Datensätze an, um systematische Muster in der Abfolge der Daten zu vermeiden.

Testdatensatz

ist eine Stichprobe der Daten, die verwendet werden, um das fertige kalibrierte Modell zu testen. Damit kommt dieser Teil des Datensatzes zum Einsatz, nachdem das Modell mithilfe der Trainings- und Validierungssätze trainiert und optimiert wurde. Die Verwendung des Testdatensatzes ist damit das State-of-the-Art-Vorgehen, um die Modelle zu testen und zu vergleichen.

Trainingsdatensatz

ist die Auswahl einer Stichprobe aus dem ursprünglichen Datensatz, der zum Training der Modelle verwendet wurde. Damit wird den Modellen während der Trainingsphase ausschließlich dieser Datenauszug vorgelegt.

Validierender Datensatz

dient zur Feinabstimmung der Modellhyperparameter. Hierbei werden diese Daten dem Modell außerhalb der Lernphase gezeigt. Die Resultate des Validierungssatzes werden genutzt, um die Hyperparameter des Modells zu aktualisieren.

Kreuzvalidierung:

Die Zerlegung der Daten für das Training, Testen und Validieren des Modells ist kein triviales Unterfangen. Im Sinn eines systematischen Vorgehens hat sich im Bereich von Data Science und Machine Learning die Kreuzvalidierung (Cross-validation) durchgesetzt. Das übergeordnete Ziel der Kreuzvalidierung ist die Generalisierung der Modellperformance, indem verschiedene Datensatzkonstellationen systematisch für das Training und Testen des Modells verwendet werden. Hierbei wird also jede Beobachtung berücksichtigt, die potenziell im Trainings-, Test- und Validierungsdatsatz vorkommt.

Literatur

- Aggarwal, C. C. (2015). *Data mining: The textbook*. Springer.
- Backhaus, K., Erichson, B., Plinke, W., & Weiber, R. (2018). *Multivariate Analysemethoden* (15. Aufl.). Gabler.
- Baier, L., Jöhren, F., & Seebacher, S. (2019). Challenges in the deployment and operation of machine learning in practise. *Proceedings of the 27th European Conference on Information Systems (ECIS)*, 8(14), 1–15.
- Basti, E., Kuzey, C., & Delen, D. (2015). Analyzing initial public offerings short-term performance using decision trees and SVMs. *Decision Support Systems*, 73(5), 15–27.
- Bichler, A., & Trommsdorff, V. (2009). Präferenzmodelle bei der Conjointanalyse. In *Conjointanalyse* (S. 59–71). Springer.
- Bittlingmayer, U. H., & Bauer, U. (2006). *Die Wissensgesellschaft: Mythos, Ideologie oder Realität?*. VS Verlag.
- Böhler, H., & Scigiliano, D. (2009). Traditionelle Conjointanalyse. In *Conjointanalyse* (S. 101–112). Springer.
- Boyd, D. M., & Ellison, N. B. (2007). Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1), 210–230.
- Brauer, J. (2009). *Grundkurs Smalltalk – Objektorientierung von Anfang an. Eine Einführung in die Programmierung* (Bd.3., erweiterte und überarbeitete Auflage mit 224 Abbildungen). Vieweg+Teubner.
- Burmester, A. B., Eggers, F., Clement, M., & Prostka, T. (2016). Accepting or fighting unlicensed usage: Can firms reduce unlicensed usage by optimizing their timing and pricing strategies? *International Journal of Research in Marketing*, 33(2), 343–356.
- Cao, L. (2017). Data science: A comprehensive overview. *ACM Computing Surveys*, 50(3), 1–42.
- Cnudde, S. D., & Martens, D. (2015). Loyal to your city? A data mining analysis of a public service loyalty program. *Decision Support Systems*, 73(5), 74–84.
- Cresci, S., Pietro, R. D., Petrocchi, M., Spognardi, A., & Tesconi, M. (2015). Fame for sale: Efficient detection of fake Twitter followers. *Decision Support Systems*, 80(12), 56–71.
- Crone, S. F. (2010). *Neuronale Netze zur Prognose und Disposition im Handel*. Springer.
- Donath, W. E., & Hoffman, A. J. (1973). Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5), 420–425.
- Donoho, D. (2017). 50 years of data science. *Journal of Computational and Graphical Statistics*, 26(4), 745–766.

- Eggers, F., Sattler, H., Teichert, T., & Völckner, F. (2018). Choice-based conjoint analysis. *Handbook of market research* (S. 1–39). Springer.
- Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. *Proceedings of the second international conference on knowledge discovery and data mining* (S. 226–231). AAAI Press.
- Evermann, J., Rehseb, J.-R., & Fettke, P. (2017). Predicting process behaviour using deep learning. *Decision Support Systems*, 100(8), 129–140.
- Fiedler, H., Kaltenborn, T., Lanwehr, R., & Melles, T. (2017). *Conjoint-analyse*. Rainer Hampp.
- Fisher, R. A. (1973). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2), 179–188.
- Fu, X., Chen, X., Shi, Y.-T., Bose, I., & Cai, S. (2017). User segmentation for retention management in online social games. *Decision Support Systems*, 101(9), 51–68.
- Gao, Y., Xu, A., Hu, P.J.-H., & Cheng, T.-H. (2017). Incorporating association rule networks in feature category-weighted naive bayes model to support weaning decision making. *Decision Support Systems*, 96(4), 27–38.
- Goodhue, D. L., Lewis, W., & Thompson, R. (2012). Does PLS have advantages for small sample size or non-normal data? *MIS Quarterly*, 36(3), 981–1001.
- Green, P. E., Krieger, A. M., & Agarwal, M. K. (1991). Adaptive conjoint analysis: Some caveats and suggestions. *Journal of Marketing Research*, 28(2), 215–222.
- Gupta, M. M., & Rao, D. H. (1994). *Neuro-control systems: Theory and applications*. IEEE Press.
- Hartmann, A., & Sattler, H. (2002). *Commercial use of conjoint analysis in Germany, Austria, and Switzerland*. Univ., Fachbereich Wirtschaftswiss., Inst. für Handel und Marketing.
- Heidenreich, S., & Kraemer, T. (2016). Innovations - Doomed to fail? Investigating strategies to overcome passive innovation resistance. *Journal of Product Innovation Management*, 33(3), 277–297.
- Heidenreich, S., Spieth, P., & Petschnig, M. (2017). Ready, steady, green: Examining the effectiveness of external policies to enhance the adoption of eco-friendly innovations. *Journal of Product Innovation Management*, 34(3), 343–359.
- Hinz, O., Schulze, C., & Takac, C. (2014). New product adoption in social networks: Why direction matters. *Journal of Business Research*, 67(1), 2836–2844.
- Hinz, O., Schlereth, C., & Zhou, W. (2015). Fostering the adoption of electric vehicles by providing complementary mobility services: A two-step approach using best-worst scaling and dual response. *Journal of Business Economics*, 85(8), 921–951.
- Hoffmann, D. (2015). *Langfristige Kapitalmarktprognosen – Eine empirische Untersuchung der Eignung von Historically Consistent Neural Networks zur langfristigen Kapitalmarktsimulation im Kontext der Alterssicherung* (Schriftreihe: Innovative Betriebswirtschaftliche Forschung und Praxis – Band 439). Verlag Dr. Kovac.
- Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2016). Extreme learning machine: Theory and applications. *Neurocomputing*, 70(12), 489–501.
- Hutson, M. (2017). Ai glossary: Artificial intelligence, in so many words. *Science*, 357(6346), 19.
- Hutto, C. J., & Gilbert, E. (2014). *Vader: A parsimonious rule-based model for sentiment analysis of social media text*. Ann Arbor: In Eighth international AAAI conference on weblogs and social media.
- Johnson, R. M., et al. (1987). Adaptive conjoint analysis. In *Sawtooth software conference proceedings* (S. 253–265). Sawtooth Software Ketchum.
- Jordan, M. I., & Mitchell, T. M. (2015). Machine learning - Trends, perspectives, and prospects. *Science*, 349(6245), 255–260.
- Kecher, C. (2011). *UML 2 – Das umfassende Handbuch* (Bd.4. aktualisierte und erw. Auflage). Galileo Press.

- Lau, R. Y. K., Liao, S. S. Y., Wong, K. F., & Chiu, D. K. W. (2012). Web 2.0 environmental scanning and adaptive decision support for business mergers and acquisitions. *MIS Quarterly*, 36(4), 1239–1268.
- Lee-Post, A., & Pakath, R. (2019). Numerical, secondary big data quality issues, quality threshold establishment, & guidelines for journal policy development. *Decision Support Systems*, 126(1), 113–135.
- Li, J., Tso, K. F., & Liu, F. (2017). Profit earning and monetary loss bidding in online entertainment shopping: The impacts of bidding patterns and characteristics. *Electronic Markets*, 25(10), 77–90.
- Lloyd, S. (1982). Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2), 129–137.
- Louviere, J. J., & Woodworth, G. (1983). Design and analysis of simulated consumer choice or allocation experiments: An approach based on aggregate data. *Journal of Marketing Research*, 20(4), 350–367.
- Lozano, M. G., Schreiber, J., & Brynielsson, J. (2017). Tracking geographical locations using a geo-aware topic model for analyzing social media data. *Decision Support Systems*, 99(1), 18–29.
- Luce, R. D., & Tukey, J. W. (1964). Simultaneous conjoint measurement: A new type of fundamental measurement. *Journal of Mathematical Psychology*, 1(1), 1–27.
- Martens, D., & Provost, F. (2014). Explaining data-driven document classifications. *MIS Quarterly*, 38(1), 73–99.
- Martens, D., Provost, F., Clark, J., & de Fortuny, E. J. (2016). Mining massive fine-grained behavior data to improve predictive analytics. *MIS Quarterly*, 40(4), 869–888.
- Milgram, S. (1967). *The small-world problem*. *Psychology Today*, 1(1), 61–67.
- Miller, K. M., Hofstetter, R., Krohmer, H., & Zhang, Z. J. (2011). How should consumers' willingness to pay be measured? An empirical comparison of state-of-the-art approaches. *Journal of Marketing Research*, 48(1), 172–184.
- Mjolsness, E., & DeCoste, D. (2001). Machine learning for science: State of the art and future prospects. *Science*, 293(5537), 2051–2055.
- Müller, O., Junglas, I., vom Brocke, J., & Debortoli, S. (2016). Utilizing big data analytics for information systems research: Challenges, promises and guidelines. *European Journal of Information Systems*, 25(5), 289–302.
- Murtagh, F., & Devlin, K. (2018). The development of data science: Implications for education, employment, research, and the data revolution for sustainable development. *Big Data and Cognitive Computing*, 2(2), 2–16.
- Page, A. L., & Rosenbaum, H. F. (1992). Developing an effective concept testing program for consumer durables. *Journal of Product Innovation Management: An International Publication of the Product Development & Management Association*, 9(4), 267–277.
- Pai, H.-T., Wu, F., & Hsueh, P.-Y.S. (2014). A relative patterns discovery for enhancing outlier detection in categorical data. *Decision Support Systems*, 67(11), 90–99.
- Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.
- Piri, S., Delen, D., & Liu, T. (2018). A synthetic informative minority over-sampling (SIMO) algorithm leveraging support vector machine to enhance learning from imbalanced datasets. *Decision Support Systems*, 106(2), 15–29.
- Poddig, T., & Sidorovitch, I. (2001). Künstliche neuronale Netze - Überblick, Einsatzmöglichkeiten und Anwendungsprobleme. In H. Hippner, U. Küsters, M. Meyer, & K. Wilde (Eds.), *Handbuch Data Mining im Marketing* (pp. 363–402). Vieweg.
- Poddig, T., Dichtl, H., & Petersmeier, K. (2008). *Statistik, Ökonometrie, Optimierung* (erweiterte, Bd.4). Uhlenbruch.

- Poddig, T., Varmaz, A., & Fieberg, C. (2015). *Computational finance - Eine Matlab. Uhlenbruch: Octave und Freemat basierte Einführung.*
- Pomberger, G., & Pree, W. (2004). *Software-Engineering Architektur-Design und Prozessorientierung* (3., völlig überarbeitete Auflage). Hanser.
- Raghavarao, D., Wiley, J. B., & Chitturi, P. (2010). *Choice-based conjoint analysis: Models and designs*. CRC Press.
- Reicha, Y., & Barai, S. (1999). Evaluating machine learning models for engineering problems. *Artificial Intelligence in Engineering*, 13(1), 257–272.
- Reinders, M. J., Frambach, R. T., & Schoormans, J. P. (2010). Using product bundling to facilitate the adoption process of radical innovations. *Journal of Product Innovation Management*, 27(7), 1127–1140.
- Rey, G. D., & Wende, K. F. (2011). *Neuronale Netze: Eine Einführung in die Grundlagen, Anwendungen und Datenauswertung* (2, vollständig überarbeitete und erweiterte Auflage). Huber.
- Rittgen, P. (2009). Self-organization of interorganizational process design. *Electronic Markets*, 19(10), 189–199.
- Roden, G. (2008). *Auf der Fährte von C - Einführung und Referenz*. Springer.
- Rogers, E. M. (1962). Diffusion of innovations (Free Press of Glencoe (1976) New product adoption and diffusion). *Journal of Consumer Research*, 2, 290–304.
- Rosenblatt, F. (1958). The perceptron. A probabilistic model for information storage and organization in the brain. *Psychological Reviews*, 65(6), 386–408.
- Russell, S. J., & Norvig, P. (2016). *Artificial intelligence: A modern approach*. Pearson.
- Sattler, H. (2005). Markenbewertung: State-of-the-art. In *Perspektiven der Kommunikationspolitik* (S. 33–58). Springer.
- Schlereth, C., & Skiera, B. (2012). Dise: Dynamic intelligent survey engine. In *Quantitative marketing and marketing management* (S. 225–243). Springer.
- See-To, E. W. K., & Yang, Y. (2017). Market sentiment dispersion and its effects on stock return and volatility. *Electronic Markets*, 27(4), 283–296.
- Shafique, U., & Kaiser, H. (2014). A comparative study of data mining process models (KDD, CRISP-DM and SEMMA). *International Journal of Innovation and Scientific Research*, 12(1), 217–222.
- Siering, M., Deokar, A. V., & Janze, C. (2018). Disentangling consumer recommendations: Explaining and predicting airline recommendations based on online reviews. *Decision Support Systems*, 107(3), 62–63.
- Singh, A., & Tucker, C. S. (2017). A machine learning approach to product review disambiguation based on function, form and behavior classification. *Decision Support Systems*, 97(5), 81–91.
- Skiera, B., & Gensler, S. (2002). Berechnung von Nutzenfunktionen und Marktsimulationen mit Hilfe der Conjoint-Analyse (Teil I). *WiSt-Wirtschaftswissenschaftliches Studium*, 31(4), 200–206.
- Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, 2, 2951–2959.
- Song, M., Parry, M. E., & Kawakami, T. (2009). Incorporating network externalities into the technology acceptance model. *Journal of Product Innovation Management*, 26(3), 291–307.
- Stehr, N. (2006). *Die Zerbrechlichkeit moderner Gesellschaften*. Velbrück Wissenschaft: Die Stagnation der Macht und die Chancen des Individuums.
- Teichert, T. (2001). *Nutzenschätzung in Conjoint-Analysen: Theoretische Fundierung und empirische Aussagekraft* (Bd. 282). Springer.
- Topuz, K., Zengul, F. D., Dag, A., Almekhi, A., & Yildirim, M. B. (2018). Predicting graft survival among kidney transplant recipients: A Bayesian decision support model. *Decision Support Systems*, 106(2), 97–109.
- Walczak, S., & Velanovich, V. (2018). Improving prognosis and reducing decision regret for pancreatic cancer treatment using artificial neural networks. *Decision Support Systems*, 106(2), 110–118.

- Wang, L., Gopal, R., Shankar, R., & Pancras, J. (2015). On the brink: Predicting business failure with mobile location-based checkins. *Decision Support Systems*, 76(8), 3–13.
- Wayne, G., & Pasternack, A. (2011). Canny minds and uncanny questions. *Science*, 333(6047), 1223–1224.
- Yang, M., Adomavicius, G., Burtch, G., & Ren, Y. (2018). Mind the gap: Accounting for measurement error and misclassification in variables generated via data mining. *Information Systems Research*, 29(1), 4–24.
- Zhao, M., Hoeffler, S., & Zauberaman, G. (2011). Mental simulation and product evaluation: The affective and cognitive dimensions of process versus outcome simulation. *Journal of Marketing Research*, 48(5), 827–839.

Stichwortverzeichnis

Symbols

L^AT_EX, 122

A

Accuracy, 206
Aktivierungsfunktion, 245
Aktivitätsniveau, 246
Amazon Web Services, 151
Artificial Neural Network, 298
Attribut, 95
Auswahlentscheidung, 101
AWS-Managementkonsole, 153

B

Barplot, 82
Bedeutungsgewicht, 107
Benennungskonvention, 15
Boxplot, 233
Breakpoint, 77
Built-in-Funktion, 12

C

CamelCase, 15
Choice Set, 96
Cloud Computing, 151
Cloud-basierter Webservice, 151
Conjoint-Analyse, 95
CSV, 74

D

Data Cleaning, 296
 Globale Konstante, 296
 Ignorieren, 297
 Interpolation, 297
 Manuelle Korrektur, 296
 Mittelwert, 297
Data Mining, 295
Data Science, 57, 295
 Data Analyst, 58
 Data Engineer, 58
 Data Scientist, 58
 Machine Learning Engineer, 58
Data-Science-Prozess, 60
 Data Preprocessing, 60
 Explore the Data, 60
 Interpretation, 60
 Model the Data, 60
DataFrame
 Ergebnisspalte, 125
 Relativer Anteil, 125
Daten, 295
 Nominale Daten, 295
 Ordinale Daten, 296
 Quantitative Daten, 296
Datenaufteilung, 298
Datencontainer, 26
Datetime-Objekt, 135
Debug-Funktion, 77
Debugger, 77
Decision Tree, 298
Depot, 253

Dictionary, 30
Diffusionsprozess, 93
Drag-and-Drop, 159
Dynamische Variablendeklaration, 43

E

Elastic Compute Cloud, 151
Entscheidungsbaum, 175
Entwicklungsumgebung, 9

F

FTP-Client, 158
Funktion, 49
 Built-in-Funktion, 49
 Funktionskopf, 51
 Funktionsrumpf, 51

H

Hashtag, 132
Hello World, 12

I

Indexierung, 27
Indexzugriff, 27
Informationsverarbeitungssystem, 244
Initialisieren, 28
Installationsdatei, 8
Integrated Development Environment, 9
International Securities Identification Number, 253
Internationaler Wettbewerb, 94

K

K-nearest Neighbour, 176
Kardinalität, 18
Käufergruppe, 98
Kommentar, 15
Konsolenausgabe, 13
Konsumentenpräferenz, 95
Kontrollstruktur, 35
 If-Else, 36
 Schleife, 41
 Iterieren über Listen, 42
 Schleifenkopf, 41
 Schleifenrumpf, 41

Try-Except, 45
While-Schleife, 44
Kreuzvalidierung, 299
Kruskal-Wallis-Test, 147

L

Laufzeit, 19
Linux-Instanz, 153
Liste, 25
 Index, 26
 Queue, 29
 Stack, 28
Logistische Regression, 126, 175, 298
 Odds-Ratio, 127

M

Machine-Learning-Modell, 297
Marketinganwendung, 95
Marketinginstrument, 93
Matplotlib, 81
Mention, 132
Merkmalsausprägung, 95
Missing Value, 297
Modul, 53
 Datetime, 231
 Standardmodul, 53
MongoDB, 134
Multivariate logistische Regressionsanalyse, 127

N

Naive Bayes, 298
Natural Language Processing, 151
Nettoeingangssignal, 245
Neuron, 245
NoSQL-Datenbank, 134
Not-a-Number, 297
Nutzerenerwartung, 95
Nutzerpräferenz, 94

O

One Hot Encoding, 184, 240
Optimierte Hyperparameter, 221
Ordinalität, 18
Outputfunktion, 247

P

Pairplot, [81](#)
Pandas, [75](#)
pandas
 DataFrame, [75](#)
Pipeline-Klasse, [192](#)
Portable Document Format, [85](#)
Portfolioanalyse, [253](#)
Pre-Processing, [297](#)
Precision, [206](#)
Primitiver Datentyp, [19](#)
 Boolean, [21](#)
 Float, [20](#)
 Integer, [20](#)
 String, [21](#)
Produkteigenschaft, [95](#)
Programmausführung, [19](#)
Project Interpreter, [11](#)
Protokoll, [158](#)
Prozessverarbeitungsschritt, [247](#)
Pseudozufallszahl, [53](#)
PyCharm, [10](#)

Q

Quellcodedokumentation, [13](#)
Query, [134](#)

R

Random Forests, [176](#), [298](#)
Recall, [206](#)
Referenzlevel, [105](#)
Regex, [136](#)
Regression
 Lineare Regression, [88](#)
Reveal-Livestream, [133](#)
Run, [12](#)

S

S3-Bucket, [151](#)
SARS-CoV2, [151](#)

Scatterplot, [81](#)
Schwellenwert, [245](#)
Seaborn, [81](#)
Set, [32](#)
SFTP - SSH File Transfer Protocol, [158](#)
Shapley-Value, [221](#)
Social-Media-Plattform, [131](#)
 Facebook, [131](#)
 Instagram, [131](#)
 Twitter, [131](#)
Software-Engineering, [15](#)
Steuerungsoperation, [19](#)
String-Konkatenation, [54](#)
Supervised Learning, [176](#)
Support Vector Machine, [298](#)
Support Vektor Maschine, [175](#)
Synthetische Daten, [74](#)

T

Terminal, [8](#)
Testdatensatz, [299](#)
Trainingsdatensatz, [299](#)
Transistor, [18](#)
Trending Topic, [132](#)
Twitter-Account, [133](#)
Twitter-API, [134](#)
Typcast, [84](#)

U

Unified Modeling Language, [35](#)

V

Validierender Datensatz, [299](#)
Variableninspektor, [77](#)
Verbindungsgewicht, [245](#)
Verbraucherakzeptanz, [94](#)

W

Wertebelegung, [19](#)