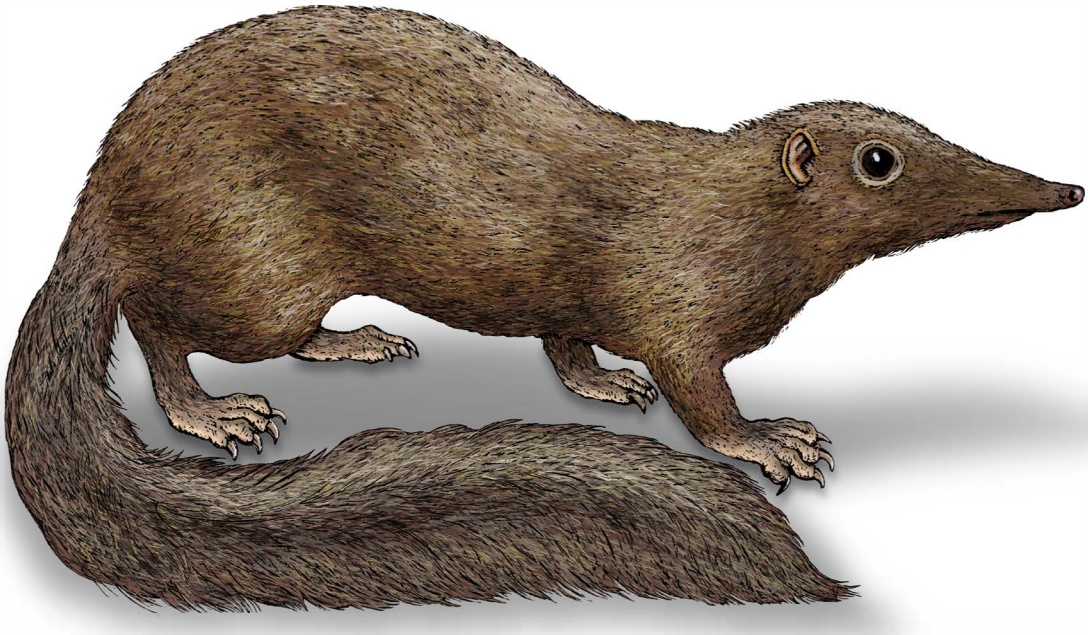
O'REILLY®

3. Auflage

Datenanalyse mit Python

Auswertung von Daten mit pandas, NumPy und Jupyter



powered by jupyter

Wes McKinney

Übersetzung von Kathrin Lichtenberg und Thomas Demmig

3. AUFLAGE

Datenanalyse mit Python

Auswertung von Daten mit pandas, NumPy und Jupyter

Wes McKinney

Übersetzung von Kathrin Lichtenberg und Thomas Demmig

O'REILLY®

Wes McKinney

Lektorat: Alexandra Follenius
Übersetzung: Kathrin Lichtenberg, Thomas Demmig
Copy-Editing: Sibylle Feldmann, *www.richtiger-text.de*
Satz: III-satz, *www.drei-satz.de*
Herstellung: Stefanie Weidner
Umschlaggestaltung: Karen Montgomery, Michael Oréal, *www.oreal.de*

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

ISBN:
Print 978-3-96009-211-7
PDF 978-3-96010-752-1
ePub 978-3-96010-753-8
mobi 978-3-96010-754-5

3. Auflage 2023
1., korrigierter Nachdruck 2024

Wieblinger Weg 17
69123 Heidelberg

Authorized German translation of the English edition of *Python for Data Analysis, 3rd Edition, ISBN 9781098104030 © 2022 Wesley McKinney*. This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint »O'REILLY«.
O'REILLY ist ein Markenzeichen und eine eingetragene Marke von O'Reilly Media, Inc. und wird mit Einwilligung des Eigentümers verwendet.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: *kommentar@oreilly.de*.

Inhalt

Vorwort

Die erste (englischsprachige) Auflage dieses Buchs wurde 2012 veröffentlicht, als die Open-Source-Bibliotheken zur Datenanalyse mit Python (insbesondere pandas) ganz neu waren und sich rasant weiterentwickelten. Als es an der Zeit war, 2016/2017 die zweite Auflage zu schreiben, musste ich das Buch nicht nur an Python 3.6 anpassen (in der ersten Auflage kam noch Python 2.7 zum Einsatz), sondern auch den neuen Funktionalitäten Rechnung tragen, die sich in den fünf Jahren dazwischen entwickelt haben. Jetzt ist es 2022, und es gab weniger Änderungen an Python (wir sind aktuell bei Erscheinen dieses Buchs bei Python 3.11), aber pandas hat sich stets weiterentwickelt.

In dieser dritten Auflage ist es mein Ziel, die Inhalte an die aktuellen Versionen von Python, NumPy, pandas und anderen Projekten anzupassen, dabei aber in Bezug auf neuere Python-Projekte aus den letzten paar Jahren eher zurückhaltend vorzugehen. Da dieses Buch für viele Vorlesungen an Universitäten und für Experten in ihrem beruflichen Alltag zu einer wichtigen Quelle geworden ist, möchte ich Themen vermeiden, die eventuell in ein oder zwei Jahren schon wieder unwichtig geworden sind. So sollte sich das Buch auch noch 2023 oder 2024 gut nutzen lassen.

Ein neues Feature der dritten Auflage ist die (englischsprachige) Open-Access-Onlineversion auf meiner Website unter *https://wesmckinney.com/book*, die als Ressource und praktischer Rückgriff für Besitzer der Papier- oder Digitalversion dieses Buchs dient. Ich plane, den Inhalt dort möglichst aktuell zu halten – wenn Sie also die gedruckte Version dieses Buchs besitzen und über etwas stolpern, das nicht richtig funktioniert, sollten Sie dort nachschauen, ob sich etwas geändert hat.

Konventionen in diesem Buch

Folgende typografische Konventionen gelten in diesem Buch:

Kursiv
: Kennzeichnet neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateierweiterungen.

`Nichtproportionalschrift`
: Kennzeichnet Programmlistings sowie Programmelemente in Absätzen, wie etwa Variablen- oder Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter.

`Nichtproportionalschrift fett`
: Stellt Befehle oder anderen Text dar, der wortwörtlich vom Benutzer eingetippt werden sollte.

`Nichtproportionalschrift kursiv`
: Zeigt Text, der durch Werte ersetzt werden soll, die der Benutzer vorgibt oder die sich aus dem Kontext ergeben.

Dieses Symbol kennzeichnet einen Tipp oder Vorschlag.

Hinter diesem Symbol verbirgt sich eine allgemeine Bemerkung.

Dieses Element symbolisiert einen Warnhinweis.

Benutzung von Codebeispielen

Sie finden die Daten und dazugehöriges Material für jedes Kapitel im GitHub-Repository dieses Buchs unter *http://github.com/wesm/pydata-book*, auch gespiegelt nach *https://gitee.com/wesmckinn/pydata-book*, falls Sie keinen Zugriff auf GitHub haben.

Das Buch soll Ihnen bei Ihrer Arbeit helfen. Ganz allgemein gilt: Wenn in diesem Buch Beispielcode angeboten wird, können Sie ihn in Ihren Programmen und Dokumentationen verwenden. Sie müssen sich dafür nicht unsere Erlaubnis einholen, es sei denn, Sie reproduzieren einen großen Teil des Codes. Schreiben Sie zum

Beispiel ein Programm, das mehrere Teile des Codes aus diesem Buch benutzt, brauchen Sie keine Erlaubnis. Verkaufen oder vertreiben Sie Beispiele aus O'Reilly-Büchern, brauchen Sie eine Erlaubnis. Beantworten Sie eine Frage, indem Sie dieses Buch und Beispielcode daraus zitieren, brauchen Sie keine Erlaubnis. Binden Sie einen großen Anteil des Beispielcodes aus diesem Buch in die Dokumentation Ihres Produkts ein, brauchen Sie eine Erlaubnis.

Wir freuen uns über eine Erwähnung, verlangen sie aber nicht. Eine Erwähnung enthält üblicherweise Titel, Autor, Verlag und ISBN, zum Beispiel: »*Datenanalyse mit Python* von Wes McKinney, O'Reilly 2023, ISBN 978-3-96009-211-7.«

Falls Sie befürchten, zu viele Codebeispiele zu verwenden oder die oben genannten Befugnisse zu überschreiten, kontaktieren Sie uns unter *kommentar@oreilly.de*.

Danksagungen

Dieses Werk ist das Produkt aus vielen Jahren der Zusammenarbeit und Hilfe sowie fruchtbarer Diskussionen mit und von Menschen auf der ganzen Welt. Ich möchte einigen von ihnen danken.

In Memoriam: John D. Hunter (1968–2012)

Unser lieber Freund und Kollege John D. Hunter verstarb am 28. August 2012 an Darmkrebs. Erst kurz zuvor hatte ich das Manuskript für die erste Auflage dieses Buchs fertiggestellt.

Man kann Johns Einfluss und Vermächtnis in der wissenschaftlichen Python-Gemeinde nicht hoch genug einschätzen. Er entwickelte nicht nur matplotlib Anfang der 2000er-Jahre (in einer Zeit, als Python nicht annähernd so beliebt war), sondern war auch an der Herausbildung der Kultur einer wichtigen Generation von Open-Source-Entwicklern beteiligt, die zu den Säulen des Python-Ökosystems gehören, das wir heute oft als so selbstverständlich hinnehmen.

Ich hatte das Glück, John zu Beginn meiner Open-Source-Karriere im Januar 2010 kennenzulernen, gerade als pandas 0.1 herausgekommen war. Seine Inspiration und seine Unterstützung halfen mir selbst in den düstersten Zeiten, meine Vision von pandas und Python als erstklassige Datenanalysesprache voranzutreiben.

John stand Fernando Pérez und Brian Granger sehr nahe, die IPython, Jupyter und vielen anderen Initiativen in der Python-Gemeinde den Weg bereiteten. Wir vier hatten gehofft, gemeinsam an einem Buch zu arbeiten, doch am Ende war ich derjenige mit der meisten freien Zeit. Ich bin mir sicher, er wäre stolz auf das gewesen, was wir einzeln und als Gemeinschaft im Laufe der letzten fünf Jahre erreicht haben.

Danksagungen für die dritte Auflage (2022)

Vor mehr als zehn Jahren habe ich mit dem Schreiben der ersten Auflage dieses Buchs begonnen, und vor mehr als 15 Jahren begann meine Reise als Python-Programmierer. Seitdem hat sich viel geändert! Python hat sich von einer relativen Nischensprache für die Datenanalyse zur beliebtesten und am weitesten verbreiteten Sprache entwickelt, die die Mehrzahl (wenn nicht sogar die Mehrheit!) der Arbeiten in den Bereichen Data Science, maschinelles Lernen und künstliche Intelligenz unterstützt.

Ich habe seit 2013 nicht mehr aktiv zum Open-Source-Projekt pandas beigetragen, aber seine weltweite Gemeinschaft ist weiter gewachsen und kann als Modell einer Community-getriebenen Open-Source-Softwareentwicklung dienen. Viele »Next Generation«-Python-Projekte, die mit Tabellendaten arbeiten, modellieren ihre Benutzeroberflächen direkt nach pandas, was zeigt, dass das Projekt einen beständigen Einfluss auf die Entwicklung des Python-Ökosystems der Data Science besitzt.

Ich hoffe, dieses Buch kann weiterhin als wertvolle Quelle für Studierende und viele andere Personen dienen, die daran interessiert sind, etwas zum Arbeiten mit Daten in Python zu lernen.

Besonders dankbar bin ich O'Reilly, dass ich eine »Open Access«-Version dieses Buchs auf meiner Website unter *https://wesmckinney.com/book* bereitstellen kann, sodass hoffentlich noch mehr Menschen erreicht werden können und ihnen dabei geholfen wird, besser in die Welt der Datenanalyse einzusteigen. J. J. Allaire war dabei unverzichtbar, er half mir, das Buch von Docbook XML nach *Quarto* (*https://quarto.org*) zu portieren – einem neuen und wunderbaren Publishing-System (Druck und Web) für Wissenschaft und Technik.

Vielen Dank auch an meine Fachkorrektoren Paul Barry, Jean-Christophe Leyder, Abdullah Karasan und William Jamir, deren umfassendes Feedback die Lesbarkeit, Klarheit und Verständlichkeit dieses Buchs deutlich verbessert hat.

Danksagungen für die zweite Auflage (2017)

Es sind fast auf den Tag genau fünf Jahre vergangen, seit ich im Juli 2012 das Manuskript für die erste Auflage dieses Buchs beendet habe. Eine Menge hat sich geändert. Die Python-Gemeinde ist unglaublich gewachsen, und das sie umgebende Ökosystem der Open-Source-Software gedeiht.

Diese neue Auflage des Buchs hätte es ohne die unablässigen Bemühungen der pandas-Entwickler nicht gegeben, die das Projekt und seine Gemeinschaft zu einem der Eckpfeiler des Python-Data-Science-Ökosystems gemacht haben. Zu ihnen gehören unter anderem Tom Augspurger, Joris Van den Bossche, Chris Bartak, Phillip Cloud, gfyoung, Andy Hayden, Masaaki Horikoshi, Stephan Hoyer, Adam

Klein, Wouter Overmeire, Jeff Reback, Chang She, Skipper Seabold, Jeff Tratner und y-p.

Für ihre Hilfe und Geduld beim Schreiben dieser zweiten Auflage möchte ich den O'Reilly-Mitarbeitern danken: Marie Beaugureau, Ben Lorica und Colleen Toporek. Ihr technisches Expertenwissen brachten Tom Augspurger, Paul Barry, Hugh Brown, Jonathan Coe und Andreas Müller ein. Danke schön.

Die erste Auflage dieses Buchs wurde in viele Sprachen übersetzt, darunter Chinesisch, Französisch, Deutsch, Japanisch, Koreanisch und Russisch. Das Übersetzen des Inhalts, der dadurch einem viel breiteren Publikum zugänglich wird, ist eine gigantische und oft undankbare Aufgabe. Ich danke den Übersetzern, dass sie Menschen auf der ganzen Welt helfen, das Programmieren und die Benutzung von Datenanalysewerkzeugen zu erlernen.

Ich hatte außerdem das Glück, dass mich Cloudera und Two Sigma Investments in den letzten Jahren bei meinen Open-Source-Entwicklungsarbeiten unterstützt haben. Oft sind Open-Source-Projekte trotz einer nicht unbeträchtlichen Benutzerbasis äußerst armselig mit Ressourcen ausgestattet. Deshalb wird es immer wichtiger – und ist auch das einzig Richtige –, dass Unternehmen die Entwicklung von wichtigen Open-Source-Projekten unterstützen.

Danksagungen für die erste Auflage (2012)

Dieses Buch hätte ich ohne die Unterstützung vieler Menschen niemals schreiben können.

Unter den O'Reilly-Mitarbeitern bin ich meinen Lektorinnen Meghan Blanchette und Julie Steele unheimlich dankbar, die mich durch den Prozess begleitet haben. Mike Loukides arbeitete mit mir während der Entwurfsphase zusammen und half mir, das Buch real werden zu lassen.

Viele Menschen haben mich als technische Gutachter unterstützt. Besonders danken möchte ich Martin Blais und Hugh Brown für ihre Hilfe bei den Beispielen für dieses Buch, bei der Übersichtlichkeit und beim Aufbau. James Long, Drew Conway, Fernando Pérez, Brian Granger, Thomas Kluyver, Adam Klein, Josh Klein, Chang She und Stéfan van der Walt haben jeweils ein oder mehrere Kapitel begutachtet, umfangreich kritisiert und von vielen verschiedenen Gesichtspunkten aus beleuchtet.

Diverse großartige Ideen für Beispiele und Datensätze kamen von Freunden und Kollegen in der Datencommunity, darunter Mike Dewar, Jeff Hammerbacher, James Johndrow, Kristian Lum, Adam Klein, Hilary Mason, Chang She und Ashley Williams.

Ich stehe natürlich in der Schuld zahlreicher Pioniere in der wissenschaftlichen Open-Source-Python-Community. Sie haben mir geholfen, das Fundament meiner Entwicklungsarbeit zu legen, und mich beim Schreiben dieses Buchs ermutigt: das

IPython-Kernteam (Fernando Pérez, Brian Granger, Min Ragan-Kelly, Thomas Kluyver und andere), John Hunter, Skipper Seabold, Travis Oliphant, Peter Wang, Eric Jones, Robert Kern, Josef Perktold, Francesc Alted, Chris Fonnesbeck und viele weitere, die hier nicht erwähnt werden können. Verschiedene Menschen gaben mir darüber hinaus ihre Unterstützung sowie Ideen und Ermutigung: Drew Conway, Sean Taylor, Giuseppe Paleologo, Jared Lander, David Epstein, John Krowas, Joshua Bloom, Den Pilsworth, John Myles-White und viele andere, die ich vergessen habe.

Ich möchte außerdem einer Reihe von Menschen aus meinen Lehrjahren danken. Zuallererst danke ich meinen früheren Kollegen bei AQR, die mich über die Jahre bei meiner Arbeit an pandas angefeuert haben: Alex Reyfman, Michael Wong, Tim Sargen, Oktay Kurbanov, Matthew Tschantz, Roni Israelov, Michael Katz, Aris Levine, Chris Uga, Prasad Ramanan, Ted Square und Hoon Kim. Und schließlich danke ich meinen akademischen Lehrmeistern Haynes Miller (MIT) und Mike West (Duke University).

Eine Menge Hilfe bekam ich im Jahr 2014 von Phillip Cloud und Joris Van den Bossche beim Aktualisieren der Codebeispiele in diesem Buch und beim Beheben einiger anderer Ungenauigkeiten, die Änderungen in pandas geschuldet waren.

Auf persönlicher Ebene danke ich Casey, die meinen tagtäglichen Schreibprozess unterstützte und meine Höhen und Tiefen tolerierte, als ich trotz eines überfüllten Terminplans den endgültigen Entwurf zusammenschrieb. Meine Eltern schließlich lehrten mich, immer meinen Träumen zu folgen und mich nie mit weniger zufriedenzugeben.

KAPITEL 1

Einleitung

1.1 Worum geht es in diesem Buch?

Dieses Buch befasst sich mit dem Manipulieren, Verarbeiten, Sortieren und Komprimieren von Daten in Python. Mein Ziel ist es, einen Wegweiser zu den Teilen der Programmiersprache Python und ihrem datenorientierten Bibliothekssystem zu bieten, die Ihnen helfen, zu einem effektiven Datenanalytiker zu werden. Auch wenn das Wort »Datenanalyse« im Titel dieses Buchs auftaucht, liegt der Fokus eher auf der Python-Programmierung, seinen Bibliotheken und Tools als auf einer Methodologie zur Datenanalyse. Es ist die Python-Programmierung, die sie für die Datenanalyse brauchen.

Dieses Buch wurde erstmals 2012 veröffentlicht. Später begannen die Leute damit, *Data Science* als Oberbegriff für alles, von einfachen, deskriptiven Statistiken bis hin zu ausgefeilteren statistischen Analysen und maschinellem Lernen, zu verwenden. Das Python-Open-Source-Ökosystem für Datenanalyse (oder Data Science) hat sich seitdem auch deutlich vergrößert. Es gibt jetzt viele weitere Bücher, deren Fokus spezifisch auf diesen fortgeschritteneren Methodiken liegt. Ich hoffe, dass dieses Buch als adäquate Vorbereitung dient, nach der Sie sich den domänenspezifischeren Quellen zuwenden können.

Manche mögen einen Großteil dieses Buchs eher dem Bereich der »Datenverarbeitung« statt der »Datenanalyse« zuordnen. Wir verwenden dafür im Englischen auch gern die Begriffe *Wrangling* oder *Munging*.

Welche Arten von Daten?

Was meine ich, wenn ich von »Daten« spreche? Der hauptsächliche Fokus liegt auf *strukturierten Daten*, einem bewusst vage gehaltenen Begriff, der viele verschiedene verbreitete Formen von Daten umfasst, wie etwa:

- Tabellarische oder in Spreadsheets angeordnete Daten, in denen jede Spalte einen anderen Typ aufweisen könnte (Strings, numerische, kalendarische Daten

oder andere). Dies schließt die meisten Datenarten ein, die üblicherweise in relationalen Datenbanken oder in tabulator- oder kommaseparierten Textdateien gespeichert werden.

- Mehrdimensionale Arrays (Matrizen).
- Mehrere Tabellen mit Daten, untereinander verbunden durch Schlüsselspalten (entspricht den in SQL geläufigen Primär- und Fremdschlüsseln).
- Zeitreihen mit festen oder variablen Intervallen.

Diese Liste ist keinesfalls vollständig. Auch wenn es nicht immer offensichtlich ist, kann ein großer Prozentsatz an Datensätzen in eine strukturierte Form umgewandelt werden, die sich besser für die Analyse und Modellierung eignet. Falls das nicht möglich ist, kann man möglicherweise Features aus den Datensätzen extrahieren und sie in eine strukturierte Form bringen. Beispielsweise könnte man eine Sammlung von Zeitungsartikeln zu einer Worthäufigkeitstabelle verarbeiten, mit der sich eine Stimmungsanalyse durchführen ließe.

Den meisten Benutzern von Tabellenverarbeitungsprogrammen wie Microsoft Excel, dem vielleicht am weitesten verbreiteten Datenanalysetool, sind diese Arten von Daten nicht fremd.

1.2 Warum Python für die Datenanalyse?

Für viele Menschen ist die Programmiersprache Python ausgesprochen reizvoll. Seit ihrem ersten Erscheinen im Jahr 1991 ist Python neben Perl, Ruby und anderen zu einer der beliebtesten interpretierten Programmiersprachen geworden. Die Beliebtheit von Python und Ruby hat besonders seit 2005 stark zugenommen, weil sich darin Webseiten bauen lassen – nicht zuletzt dank ihrer zahllosen Webframeworks wie Rails (Ruby) und Django (Python). Solche Sprachen werden oft als *Skriptsprachen* bezeichnet, weil sich mit ihnen schnell kleine Programme oder *Skripte* schreiben lassen, um Aufgaben zu automatisieren. Ich persönlich mag den Begriff »Skriptsprachen« nicht, da er den Beigeschmack hinterlässt, dass damit keine ernsthafte Software herzustellen ist. Unter den interpretierten Sprachen hat sich um Python herum aus verschiedenen historischen und kulturellen Gründen eine große und aktive wissenschaftliche und Datenanalysecommunity entwickelt. In den letzten 20 Jahren ist aus der Sprache Python, die man »auf eigene Gefahr« einsetzt, eine der wichtigsten Sprachen für die Datenwissenschaft, das maschinelle Lernen bzw. Machine Learning und die allgemeine Softwareentwicklung im akademischen und industriellen Bereich geworden.

Im Bereich der Datenanalyse und des interaktiven Computings sowie der Datenvisualisierung wird Python zwangsläufig mit anderen weitverbreiteten Programmiersprachen und Tools, sowohl Open Source als auch kommerzieller Art, wie R, MATLAB, SAS, Stata und anderen verglichen. In den letzten Jahren haben Pythons

verbesserte Open-Source-Bibliotheken (wie etwa pandas und scikit-learn) es zu einer beliebten Alternative für Datenanalyseaufgaben werden lassen. In Kombination mit seiner Stärke als Mehrzweckprogrammiersprache ist Python eine ausgezeichnete Wahl für datenzentrierte Anwendungen.

Python als Kleister

Zum Erfolg von Python in der wissenschaftlichen Datenverarbeitung hat auch beigetragen, wie leicht sich C-, C++- und Fortran-Code integrieren lassen. Die meisten modernen Rechenumgebungen teilen einen ähnlichen Grundstock von ererbten Fortran- und C-Bibliotheken, die für lineare Algebra, Optimierung, Integration, schnelle Fourier-Transformation und weitere Algorithmen genutzt werden können. Das Gleiche ist der Fall in vielen Unternehmen und staatlichen Labors. Auch sie nutzen Python, um die Altsoftware der letzten Jahrzehnte miteinander zu verknüpfen.

Diverse Programme bestehen aus kleinen Codeteilen, die häufig ablaufen, während große Mengen an »Kleister-Code« nicht oft benutzt werden. In vielen Fällen fällt dessen Rechenzeit kaum ins Gewicht; man sollte sich daher besser um die Optimierung der programmtechnischen Flaschenhälse kümmern, wie etwa um das Umsetzen des Codes in eine maschinennahe Sprache wie C.

Das »Zwei-Sprachen-Problem« lösen

In vielen Unternehmen ist es üblich, zum Forschen, Experimentieren und Testen neuer Ideen eine speziellere Programmiersprache wie etwa SAS oder R zu benutzen und diese Konzepte dann später auf ein größeres Produktionssystem zu übertragen, das in Java, C# oder C++ geschrieben ist. Zunehmend stellt sich nun heraus, dass sich Python nicht nur für das Forschen und das Prototyping eignet, sondern auch zum Herstellen der Produktionssysteme. Wieso sollte man zwei Entwicklungsumgebungen vorhalten, wenn eine ausreicht? Ich glaube, dass immer mehr Unternehmen diesen Weg gehen werden, da es oft beträchtliche organisatorische Vorteile mit sich bringt, wenn sowohl die Forscher als auch die Softwareentwickler die gleichen Programmierwerkzeuge verwenden.

In den vergangenen zehn Jahren gab es ein paar neue Anläufe, das »Zwei-Sprachen-Problem« zu lösen, etwa in Form der Programmiersprache Julia. Um aus Python so viel wie möglich herauszuholen, muss man in vielen Fällen in einer Low-Level-Programmiersprache wie C oder C++ programmieren und dann Python-Bindings zu diesem Code erstellen. Daher hat die »Just-in-Time«- (JIT-)Compiler-Technologie von Bibliotheken wie Numba eine Möglichkeit geschaffen, für viele Rechenalgorithmen eine ausgezeichnete Performance zu erreichen, ohne die Programmierumgebung von Python verlassen zu müssen.

Warum nicht Python?

Obwohl Python eine ausgezeichnete Umgebung zum Erstellen vieler Arten analytischer Anwendungen und universeller Systeme ist, gibt es eine Reihe von Einsatzgebieten, für die es sich weniger eignet.

Python ist eine interpretierte Programmiersprache. Das heißt, im Allgemeinen läuft der meiste Python-Code deutlich langsamer als Code, der in einer kompilierten Sprache wie Java oder C++ geschrieben wurde. Da *Programmierzeit* oft wertvoller ist als *Rechenzeit*, gehen viele gern diesen Kompromiss ein. In einer Anwendung mit sehr niedrigen Latenzzeiten oder hohen Anforderungen an die Ressourcen (wie etwa in einem stark beanspruchten Handelssystem) dürfte die Zeit, die für das Programmieren in einer maschinennahen Sprache wie C++ aufgewandt wird, um eine maximal mögliche Produktivität zu erzielen, gut investiert sein.

Python ist nicht die ideale Wahl für hochparallele Multithread-Anwendungen, speziell Anwendungen mit vielen CPU-abhängigen Threads. Der Grund dafür ist das, was man als *Global Interpreter Lock* (GIL) bezeichnet, ein Mechanismus, der den Interpreter daran hindert, mehr als eine Python-Anweisung gleichzeitig auszuführen. Die technischen Gründe für die Existenz des GIL zu erklären, würde den Rahmen dieses Buchs sprengen. Zwar stimmt es, dass in vielen Anwendungen zur Verarbeitung von Big Data ein Cluster aus Computern nötig ist, um einen Datensatz in einer vernünftigen Zeit verarbeiten zu können, aber dennoch gibt es Situationen, in denen ein einzelner Prozess mit mehreren Threads wünschenswert ist.

Das soll jetzt nicht heißen, dass Python nicht in der Lage dazu wäre, echt parallelen multithreaded Code auszuführen. Python-C-Erweiterungen, die natives Multithreading (in C oder C++) einsetzen, können Code parallel ausführen, ohne durch das GIL beeinträchtigt zu werden, solange sie nicht regelmäßig mit Python-Objekten interagieren müssen.

1.3 Grundlegende Python-Bibliotheken

Für diejenigen, die weniger vertraut sind mit dem Python-Ökosystem und den Bibliotheken, die in diesem Buch verwendet werden, möchte ich hier einen kurzen Überblick über einige von ihnen bieten.

NumPy

NumPy (*http://numpy.org*), kurz für *Numerical Python*, ist schon lange einer der Eckpfeiler des wissenschaftlichen Programmierens in Python. Es bietet die Datenstrukturen, Algorithmen und den Bibliothekskleister, der für die meisten wissenschaftlichen Anwendungen nötig ist, die in Python numerische Daten verarbeiten. NumPy enthält unter anderem folgende Dinge:

- Ein schnelles und effizientes mehrdimensionales Array-Objekt namens *ndarray*.
- Funktionen zum Durchführen von elementweisen Berechnungen mit Arrays oder mathematischen Operationen zwischen Arrays.
- Werkzeuge zum Lesen und Schreiben von Array-basierten Datenmengen auf Datenträger.
- Lineare Algebra-Operationen, Fourier-Transformationen und Zufallszahlengeneratoren.
- Eine ausgereifte C-API, die Python-Erweiterungen und nativen C- oder C++-Code aktiviert, die auf die Datenstrukturen und Rechenfähigkeiten von NumPy zugreifen können.

NumPy erweitert Python nicht nur um die Fähigkeit, Arrays zu verarbeiten. Ein Hauptzweck in der Datenanalyse besteht in seinem Einsatz als Container für die Daten, die zwischen Algorithmen und Bibliotheken hin- und hergereicht werden. Für numerische Daten sind NumPy-Arrays effizienter in der Speicherung und Manipulation von Daten als andere in Python integrierte Datenstrukturen. Außerdem können Bibliotheken, die in einer maschinennäheren Sprache wie etwa C oder Fortran geschrieben sind, direkt auf den Daten in einem NumPy-Array operieren, ohne sie zuvor in eine andere Speicherform kopieren zu müssen. Das bedeutet, viele numerische Berechnungswerkzeuge für Python erkennen NumPy-Arrays entweder als primäre Datenstruktur an oder arbeiten zumindest nahtlos mit NumPy zusammen.

pandas

pandas (*http://pandas.pydata.org*) bietet umfangreiche Datenstrukturen und Funktionen für ein intuitives und flexibles Arbeiten mit strukturierten oder tabellarischen Daten. Seit seinem Auftauchen im Jahr 2010 hat es dazu beigetragen, aus Python eine starke und produktive Datenanalyseumgebung zu machen. Die pandas-Objekte, die in diesem Buch hauptsächlich benutzt werden, sind `DataFrame`, eine tabellenförmige, spaltenorientierte Datenstruktur mit Titeln für Zeilen und Spalten, und `Series`, ein eindimensionales Array-Objekt, ebenfalls mit Titel.

pandas kombiniert die Arrays aus NumPy mit der flexiblen Datenmanipulation von Spreadsheets und relationalen Datenbanken (wie etwa SQL). Es bietet eine praktische Indizierung, wodurch man die Daten einfach umgestalten, zurechtschneiden und zusammenfassen sowie Teilmengen von Daten auswählen kann. Da die Datenmanipulation, -vorbereitung und -bereinigung eine so wichtige Fähigkeit der Datenanalyse ist, befindet sich pandas ganz besonders im Fokus dieses Buchs.

Einige Hintergrundinformationen: Ich begann Anfang 2008 mit der Entwicklung von pandas, als ich bei AQR Capital Management arbeitete, einem Finanzdienstleister. Damals hatte ich ganz klare Anforderungen, die kein Tool aus meiner Werkzeugkiste komplett allein erfüllte:

- Datenstrukturen mit gekennzeichneten Achsen, die eine automatische oder explizite Datenausrichtung unterstützen – dies verhindert oft auftretende Fehler aufgrund falsch ausgerichteter Daten und das Arbeiten mit unterschiedlich indizierten Daten, die aus unterschiedlichen Quellen stammen.
- Integrierte Zeitreihenfunktionalität.
- Die gleichen Datenstrukturen behandeln sowohl Zeitreihendaten als auch Nichtzeitreihendaten.
- Arithmetische Operationen und Reduktionen, die die Metadaten erhalten.
- Flexible Behandlung fehlender Daten.
- Merge und andere relationale Operationen, die in beliebten Datenbanken (z.B. SQL-basierten) zu finden sind.

Ich wollte alle diese Dinge gern an einem Ort vereint sehen, vorzugsweise in einer Sprache, die sich gut für die allgemeine Softwareentwicklung eignet. Python schien dafür ein guter Kandidat zu sein, allerdings waren damals noch keine Datenstrukturen und Werkzeuge enthalten, die diese Funktionalität geboten hätten. Da pandas anfangs vor allem zum Lösen von Finanz- und Geschäftsanalyseproblemen dienen sollte, zeichnet es sich vor allem durch seine Zeitreihenfunktionen sowie Werkzeuge für das Arbeiten mit zeitindizierten Daten aus Geschäftsprozessen aus.

Einen Großteil der Jahre 2011 und 2012 verbrachte ich damit, zusammen mit meinen ehemaligen AQR-Kollegen Adam Klein und Chang She pandas in dieser Hinsicht zu erweitern. 2013 zog ich mich aus der täglichen Projektarbeit zurück, und pandas ist seitdem zu einem komplett von der Community verantworteten und gewarteten Projekt geworden – mit mehr als 2.000 Beitragenden auf der ganzen Welt.

Anwender der Sprache R für statistische Berechnungen werden den Begriff *DataFrame* kennen, da das Objekt nach dem vergleichbaren R-Objekt `data.frame` benannt wurde. Anders als bei Python sind DataFrames in die Programmiersprache R und seine Standardbibliothek integriert. Dadurch sind viele Features, die man in pandas findet, entweder Teil der R-Kernimplementierung oder werden durch Zusatzpakete zur Verfügung gestellt.

Der Name pandas selbst ist von *Panel Data* (Paneldaten) abgeleitet, einem Ökonomiebegriff für mehrdimensionale strukturierte Datenmengen, sowie von *Python Data Analysis* selbst.

matplotlib

matplotlib (*http://matplotlib.org*) ist die beliebteste Python-Bibliothek zum Zeichnen und für andere zweidimensionale Datenvisualisierungen. Ursprünglich von John D. Hunter geschaffen, wird sie nun von einem großen Entwicklerteam betreut. Sie dient dem Herstellen von Zeichnungen, die sich für eine Veröffentlichung eignen. Es gibt zwar auch andere Visualisierungsbibliotheken für Python-

Programmierer, doch matplotlib ist immer noch sehr verbreitet und ziemlich gut in das restliche Ökosystem integriert. Ich denke, sie ist eine gute Wahl als Standardvisualisierungswerkzeug.

IPython und Jupyter

Das *IPython-Projekt* (*http://ipython.org*) startete 2001 als Nebenprojekt von Fernando Pérez, der einen besseren interaktiven Python-Interpreter herstellen wollte. In den folgenden 20 Jahren wurde es zu einem der wichtigsten Tools im modernen Python-Werkzeugkasten. IPython bietet zwar selbst keine Rechen- oder Datenanalysewerkzeuge, erlaubt Ihnen aufgrund seiner Struktur aber, interaktiv zu arbeiten sowie in der Softwareentwicklung eingesetzt zu werden. Es unterstützt einen *Execute-Explore*-Workflow anstelle des typischen *Edit-Compile-Run*-Workflows, den man in vielen anderen Programmiersprachen pflegt. Außerdem bietet es einen integrierten Zugriff auf die Shell und das Dateisystem Ihres Betriebssystems, was in vielen Fällen den manuellen Wechsel zwischen Terminalfenster und Python-Session überflüssig macht. Da die Programmierung zur Datenanalyse zu einem Großteil auf Erkundung (Exploration), Trial-and-Error und Iterationen beruht, kann IPython Ihnen helfen, den Job schneller zu erledigen.

2014 kündigten Fernando und das IPython-Team das *Jupyter-Projekt* (*http://jupyter.org*) an, eine breitere Initiative zum Entwickeln interaktiver, sprachunabhängiger Rechenwerkzeuge. Das IPython-Web-Notebook wurde zum Jupyter-Notebook, das jetzt mehr als 40 Programmiersprachen unterstützt. Das IPython-System kann nun als *Kernel* (ein Programmiersprachenmodus) für die Benutzung von Python mit Jupyter verwendet werden.

IPython selbst ist eine Komponente des viel breiteren Jupyter-Open-Source-Projekts, das eine produktive Umgebung für das interaktive und untersuchende Arbeiten bietet. Sein ältester und einfachster »Modus« ist eine erweiterte Python-Shell, die das Schreiben und Testen von Python-Code sowie die Fehlersuche beschleunigen soll. Die IPython-Shell und die Jupyter-Notebooks eignen sich besonders für die Datenuntersuchung und -visualisierung. Zudem können Sie das IPython-System über Jupyter-Notebooks nutzen.

Mit dem Jupyter-Notebook können Sie darüber hinaus Inhalte in Markdown und HTML erstellen. Sie haben also eine Möglichkeit, mit Auszeichnungen versehende Dokumente mit Code und Text anzulegen.

Ich persönlich benutze IPython und Jupyter fast immer, wenn ich mit Python arbeite, etwa zum Ausführen, Debuggen und Testen von Code.

In den *Begleitmaterialien zu diesem Buch auf GitHub* (*http://github.com/wesm/pydata-book*) finden Sie Jupyter-Notebooks mit allen Codebeispielen aus den einzelnen Kapiteln. Haben Sie keinen Zugriff auf GitHub, können Sie den *Mirror bei Gitee* (*https://gitee.com/wesmckinn/pydata-book*) ausprobieren.

SciPy

SciPy (*http://scipy.org*) ist eine Sammlung von Paketen, die sich mit einer Reihe von grundlegenden Problemfeldern beim wissenschaftlichen Rechnen befassen. Hier ist eine Auswahl der Tools aus den diversen Modulen:

`scipy.integrate`
: Numerische Integrationsroutinen und Lösung von Differenzialgleichungen.

`scipy.linalg`
: Routinen aus der linearen Algebra und Matrixzerlegungen, die den Rahmen von `numpy.linalg` übersteigen.

`scipy.optimize`
: Funktionsoptimierer (Minimierer) und Algorithmen zur Wurzelbestimmung.

`scipy.signal`
: Werkzeuge zur Signalverarbeitung.

`scipy.sparse`
: Schwach besetzte Matrizen und Löser von schwach besetzten linearen Gleichungssystemen.

`scipy.special`
: Wrapper um SPECFUN, eine FORTRAN-Bibliothek, die viele verschiedene mathematische Funktionen enthält, wie etwa die `gamma`-Funktion.

`scipy.stats`
: Gewöhnliche stetige und diskrete Wahrscheinlichkeitsverteilungen (Dichtefunktionen, Stichproben, stetige Verteilungsfunktionen), verschiedene statistische Tests und weitere deskriptive Statistik.

Gemeinsam bilden NumPy und SciPy eine relativ vollständige und ausgereifte Rechengrundlage für viele traditionelle wissenschaftliche Rechenanwendungen.

scikit-learn

Seit dem Beginn des Projekts im Jahr 2007 ist *scikit-learn* (*http://scikit-learn.org*) zum wichtigsten Machine-Learning-Toolkit für Python-Programmierer geworden. Mehr als 2.000 Menschen aus der ganzen Welt haben bereits daran mitgearbeitet. Es enthält Untermodule für Modelle wie:

- Klassifizierung: SVM, nächste Nachbarn, Random Forest, logistische Regression usw.
- Regression: LASSO, Ridge Regression usw.
- Clusteranalyse: *k*-means, Spectral Clustering usw.
- Dimensionsreduktion: PCA, Feature Selection, Matrixfaktorisierung usw.
- Modellauswahl: Rastersuche, Kreuzvalidierung, Metriken
- Vorverarbeitung: Feature-Extraktion, Normalisierung

Gemeinsam mit pandas, statsmodels und IPython hat scikit-learn entscheidend dazu beigetragen, aus Python eine produktive Data-Science-Programmiersprache zu machen. Ich kann zwar keine umfassende Anleitung für scikit-learn in dieses Buch aufnehmen, stelle aber kurz einige seiner Modelle vor und beschreibe, wie man diese zusammen mit anderen Werkzeugen aus diesem Buch einsetzt.

statsmodels

statsmodels (*http://statsmodels.org*) ist ein statistisches Analysepaket, das aus der Arbeit des Stanford-Statistikprofessors Jonathan Taylor hervorgegangen ist, der eine Reihe von Regressionsanalysemodellen in der Programmiersprache R implementiert hat. Skipper Seabold und Josef Perktold begründeten 2010 offiziell das neue statsmodels-Projekt, das mittlerweile zu einem Projekt mit einer beträchtlichen Nutzer- und Mitarbeiterbasis angewachsen ist. Nathaniel Smith entwickelte das Patsy-Projekt, das ein durch das Formelsystem von R inspiriertes Framework zur Formel- oder Modellspezifikation für statsmodels bietet.

Im Gegensatz zu scikit-learn enthält statsmodels Algorithmen für die klassische (vor allem frequentistische) Statistik und Ökonometrie. Dazu gehören Untermodule wie:

- Regressionsmodelle: lineare Regression, generalisierte lineare Modelle, robuste lineare Modelle, lineare Mixed-Effects-Modelle usw.
- Varianzanalyse (*Analysis of Variance*, ANOVA)
- Zeitreihenanalyse: AR, ARMA, ARIMA, VAR und andere Modelle
- nichtparametrische Methoden: Kerndichteschätzung, Kernel-Regression
- Visualisierung der Ergebnisse statistischer Modelle

statsmodels konzentriert sich vor allem auf die Inferenzstatistik (schließende Statistik), für die es Unsicherheitsschätzungen und *p*-Werte für Parameter liefert. scikit-learn ist dagegen eher vorhersageorientiert.

Analog zu scikit-learn werde ich eine kurze Einführung in statsmodels liefern und beschreiben, wie man es mit NumPy und pandas einsetzt.

Andere Pakete

Es gibt mittlerweile viele weitere Pakete, die sich in einem Buch über Data Science behandeln ließen. Dazu gehören einige der neueren Projekte wie TensorFlow oder PyTorch, die in den Bereichen maschinelles Lernen und künstliche Intelligenz große Verbreitung gefunden haben. Da es andere Bücher gibt, die sich mehr auf diese Projekte konzentrieren, empfehle ich, dieses Buch für die Grundlagen beim Umgang mit Daten in Python zu nutzen, um sich dann fortgeschrittenen Quellen zuzuwenden, die schon mehr Wissen erwarten.

1.4 Installation und Einrichtung

Da jeder Python für unterschiedliche Anwendungen nutzt, gibt es keine eindeutige Regel dafür, wie man es einrichtet und welche Zusatzpakete nötig sind. Viele Leserinnen und Leser haben vermutlich nicht die komplette Python-Entwicklungsumgebung, die man zur Arbeit mit diesem Buch benötigt. Deshalb werde ich hier ausführlich erläutern, wie man es auf den verschiedenen Betriebssystemen installiert. Ich empfehle die Miniconda-Distribution, bei der es sich um eine Minimalinstallation des Paketmanagers Conda handelt, und dazu *conda-forge* (*https://conda-forge.org*), eine durch die Community betreute Softwaredistribution, die auf Conda basiert. Dieses Buch nutzt Python 3.10, aber wenn es eine neuere Version gibt, können Sie die gern installieren. Sollte diese Anleitung mit der Zeit veraltet sein, können Sie auf meiner *Website zum Buch* (*https://wesmckinney.com/book*) vorbeischauen, wo ich versuchen werde, die Installationsanweisungen immer aktuell zu halten.

Miniconda auf Windows

Um auf Windows loslegen zu können, laden Sie den Miniconda-Installer für die neueste verfügbare Python-Version (aktuell 3.9) von (*https://conda.io*) herunter. Ich empfehle Ihnen, den Installationsanweisungen für Windows zu folgen, die Sie auf der Conda-Website finden und die sich geändert haben können, seit diese Zeilen geschrieben wurden. Die meisten Anwenderinnen und Anwender werden die 64-Bit-Version nutzen wollen, aber wenn diese nicht auf Ihrem Windows-Rechner läuft, können Sie stattdessen die 32-Bit-Version installieren.

Sollten Sie gefragt werden, ob Sie die Installation nur für sich selbst oder für alle Benutzer auf Ihrem System durchführen wollen, wählen Sie die Option, die für Sie am passendsten ist. Installieren Sie Conda nur für sich, wird das für das Nachvollziehen der Beispiele aus diesem Buch ausreichen. Sie werden auch gefragt werden, ob Sie Miniconda zur Systemumgebungsvariablen `PATH` hinzufügen wollen. Wählen Sie das aus (was ich normalerweise tue), überschreibt diese Miniconda-Installation eventuell andere Python-Versionen, die Sie installiert haben. Entscheiden Sie sich dagegen, müssen Sie den Shortcut im Startmenü von Windows nutzen, um dieses Miniconda verwenden zu können. Der Name des Eintrags lautet beispielsweise »Anaconda3 (64-bit)«.

Ich gehe davon aus, dass Sie Miniconda nicht Ihrem System-`PATH` hinzugefügt haben. Um zu prüfen, ob alles korrekt konfiguriert wurde, öffnen Sie im Startmenü den Punkt *Anaconda Prompt (Miniconda3)* unter *Anaconda3 (64-bit)*. Versuchen Sie dann, den Python-Interpreter durch Eingabe von **`python`** zu starten. Sie sollten eine Ausgabe wie die folgende erhalten:

```
(base) C:\Users\Wes>python
Python 3.9 [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Sie beenden das Kommandozeilenfenster durch Eintippen des Befehls **`exit()`** und Drücken der Enter-Taste.

GNU/Linux

Unter Linux hängen die genauen Einzelheiten von Ihrer jeweiligen Linux-Fassung ab. Ich beschreibe das Vorgehen hier für Distributionen wie Debian, Ubuntu, CentOS und Fedora. Abgesehen von der Installation von Miniconda verläuft das Einrichten ähnlich wie bei macOS. Meist wird man die Datei für den normalen 64-Bit-Installer herunterladen wollen, der für die x86-Architektur gedacht ist (aber eventuell werden in Zukunft mehr Anwender auf aarch64 basierende Systeme nutzen). Der Installer ist ein Shell-Skript, das im Terminal ausgeführt werden muss. Sie haben dann eine Datei, die *Miniconda3-latest-Linux-x86_64.sh* oder ähnlich heißt. Um sie zu installieren, führen Sie dieses Skript mit der bash-Shell aus:

```
$ bash Miniconda3-latest-Linux-x86_64.sh
```

Manche Linux-Distributionen enthalten Versionen aller erforderlichen Python-Pakete (wenn auch manchmal in veralteten Versionen). Sie können mit dem eingebauten Paketmanager installiert werden, etwa mit apt. Ich demonstriere das hier mit Miniconda, weil es sich einfach auf andere Distributionen übertragen und auf die neuesten Versionen aktualisieren lässt.

Sie können auswählen, wo die Miniconda-Dateien abgelegt werden sollen. Ich empfehle Ihnen, die Dateien an der vorgegebenen Stelle in Ihrem Home-Verzeichnis zu installieren – zum Beispiel in */home/$USER/miniconda* (natürlich mit Ihrem Benutzernamen).

Der Installer fragt Sie möglicherweise, ob er Ihre Shell-Skripte anpassen soll, damit Miniconda automatisch aktiviert wird. Ich empfehle das (wählen Sie *yes*), da das sehr praktisch ist.

Starten Sie nach dem Abschluss der Installation einen neuen Terminalprozess und prüfen Sie, ob die neue Miniconda-Installation genutzt wird:

```
(base) $ python
Python 3.9 | (main) [GCC 10.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Um die Python-Shell zu verlassen, geben Sie **`exit()`** ein und drücken die Enter-Taste, oder Sie drücken Strg-D.

Miniconda auf macOS

Laden Sie den macOS-Miniconda-Installer herunter, der für auf Apple Silicon basierende macOS-Computer (ab 2020) *Miniconda3-4.1.0-latest-arm64.sh* oder ähnlich heißen sollte, während der Name für Intel-basierte Macs (bis 2020) *Mini-*

conda3-4.1.0-latest-x86_64.sh lautet. Öffnen Sie die Terminalanwendung in macOS und starten Sie den Installer (der sich vermutlich in Ihrem *Downloads*-Verzeichnis befindet) mit `bash`:

```
$ bash $HOME/Downloads/Miniconda3-latest-MacOSX-arm64.sh
```

Wenn der Installer läuft, konfiguriert er Miniconda automatisch in Ihrer Standard-Shell-Umgebung in Ihrem Standard-Shell-Profil. Dieses befindet sich wahrscheinlich unter */Users/$USER/.zshrc*. Ich empfehle, das zuzulassen – wenn Sie das nicht tun, müssen Sie in der Dokumentation zu Miniconda nachschauen, um weitermachen zu können.

Um zu prüfen, ob alles funktioniert, starten Sie Python in einem Kommandozeilenfenster (Sie erhalten eine Kommandozeile im Terminalprogramm):

```
$ python
Python 3.9 (main) [Clang 12.0.1 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Zum Verlassen der Shell drücken Sie Strg-D oder tippen **`exit()`** ein und drücken die Enter-Taste.

Python-Pakete installieren oder aktualisieren

Nachdem wir Miniconda auf unserem System eingerichtet haben, wollen wir nun die wichtigsten Pakete installieren, die in diesem Buch zum Einsatz kommen. Der erste Schritt ist, conda-forge als Standardpaketkanal zu konfigurieren, indem wir folgende Befehle in einer Shell ausführen:

```
(base) $ conda config --add channels conda-forge
(base) $ conda config --set channel_priority strict
```

Jetzt werden wir mit dem Befehl `conda create` eine neue conda-»Umgebung« erstellen, die Python 3.10 nutzt:

```
(base) $ conda create -y -n pydata-book python=3.10
```

Ist die Installation abgeschlossen, aktivieren Sie die Umgebung mit `conda activate`:

```
(base) $ conda activate pydata-book
(pydata-book) $
```

Sie müssen `conda activate` jedes Mal nutzen, um Ihre Umgebung zu aktivieren, wenn Sie ein neues Terminal öffnen. Informationen zur aktiven conda-Umgebung erhalten Sie jederzeit, wenn Sie im Terminal `conda info` eingeben.

Jetzt werden wir mit `conda install` die grundlegenden Pakete installieren, die in diesem Buch zum Einsatz kommen (und ihre Abhängigkeiten):

```
(pydata-book) $ conda install -y pandas jupyter matplotlib
```