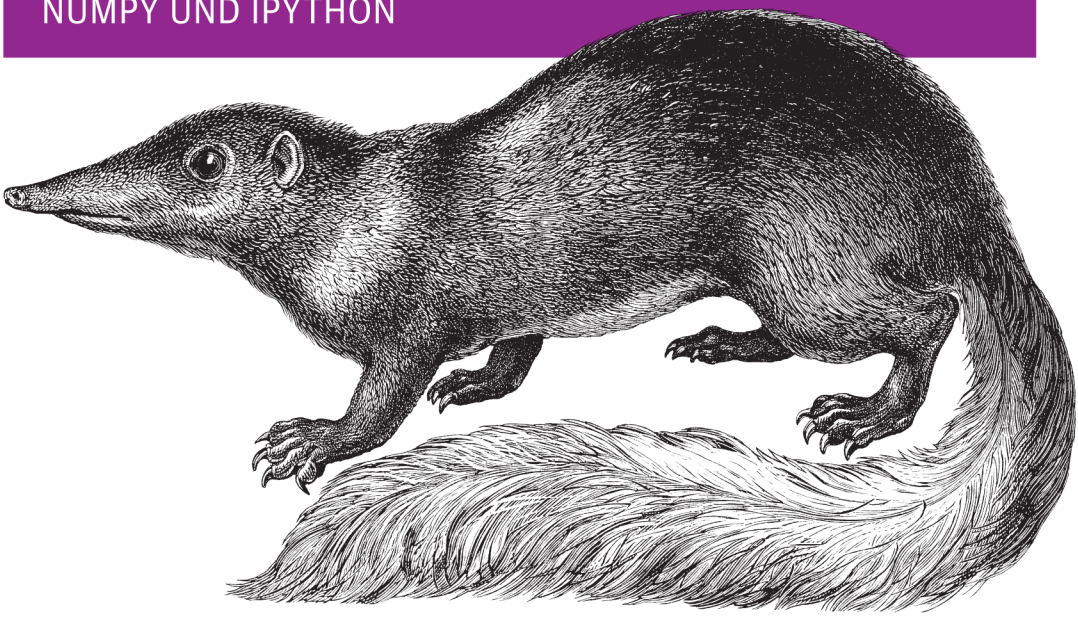
O'REILLY®

2. Auflage

# Datenanalyse mit Python

AUSWERTUNG VON DATEN MIT PANDAS, NUMPY UND IPYTHON



powered by

jupyter

Wes McKinney

Übersetzung von Christian Tismer, Kristian Rother und Kathrin Lichtenberg

2. AUFLAGE

# Datenanalyse mit Python

*Auswertung von Daten mit Pandas, NumPy und IPython*

**Wes McKinney**

*Deutsche Übersetzung von Kristian Rother, Christian Tismer & Kathrin Lichtenberg*

O'REILLY®

Wes McKinney

Lektorat: Alexandra Follenius
Übersetzung: Kristian Rother, Christian Tismer und Kathrin Lichtenberg
Korrektorat: Sibylle Feldmann, www.richtiger-text.de
Satz: III-satz, www.drei-satz.de
Herstellung: Stefanie Weidner
Umschlaggestaltung: Karen Montgomery, Michael Oréal, www.oreal.de
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

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 978-3-96009-080-9
PDF 978-3-96010-213-7
ePub 978-3-96010-214-4
mobi 978-3-96010-215-1

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.

2. Auflage

Wieblinger Weg 17
69123 Heidelberg



5 4 3 2 1 0

# Inhalt

# Vorwort

## Neu in der 2. Auflage

Die 1. (englischsprachige) Auflage dieses Buchs wurde 2012 veröffentlicht, als die Open-Source-Bibliotheken zur Datenanalyse mit Python (wie etwa pandas) ganz neu waren und sich rasant weiterentwickelten. In dieser aktualisierten und erweiterten 2. Auflage habe ich die Kapitel überarbeitet, um sowohl den inkompatiblen Änderungen und überholten Teilen als auch den neuen Funktionalitäten Rechnung zu tragen, die sich in den letzten sechs Jahren gezeigt haben. Ich habe außerdem neue Inhalte hinzugefügt, in denen ich Tools vorstelle, die es 2012 noch nicht gab oder die damals noch nicht ausgereift genug waren. Und schließlich habe ich versucht, zu vermeiden, über neue oder topaktuelle Open-Source-Projekte zu schreiben, die bisher noch keine Chance hatten, zu reifen. Ich möchte nämlich, dass diese Auflage für die Leser im Jahr 2020 oder 2021 noch fast genauso relevant ist wie 2019.

Zu den wichtigsten Aktualisierungen in der 2. Auflage gehören:

- Der gesamte Code einschließlich des Python-Tutorials wurde an Python 3.6 angepasst (die 1. englischsprachige Auflage benutzte Python 2.7, die 1. deutsche Auflage war für Python 3.4 bearbeitet worden).
- Aktualisierte Python-Installationsanweisungen für die Anaconda-Python-Distribution und andere notwendige Python-Pakete.
- Updates auf die neuesten Versionen der pandas-Bibliothek aus dem Jahr 2017.
- Ein neues Kapitel über einige fortgeschrittene pandas-Tools mit weiteren Anwendungstipps.
- Eine kurze Einführung in die Benutzung von statsmodels und scikit-learn.

Außerdem habe ich einen großen Teil des Inhalts der 1. Auflage neu organisiert, um das Buch für Anfänger leichter zugänglich zu machen.

## 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 dazugehörendes Material für jedes Kapitel im GitHub-Repository dieses Buchs unter *http://github.com/wesm/pydata-book*.

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 eine CD-ROM mit Beispielen 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 2019, ISBN 978-3-96009-080-9.«

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 1. 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 kritischen 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 Anfang meiner Open-Source-Karriere im Januar 2010 kennenzulernen, gerade als pandas 0.1 herausgekommen war. Seine Inspiration und 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 2. Auflage

Es sind fast auf den Tag genau fünf Jahre vergangen, seit ich im Juli 2012 das Manuskript für die 1. 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 2. 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 1. 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 1. Auflage

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 außerdems 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, 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.

### 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 Samm-

lung 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 »Skriptsprache« 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 zehn 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 hat Pythons verbesserte Unterstützung für 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.

## 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 schnelles, einfaches und ausdrucksstarkes 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 hohe Leistungsfähigkeit bei Arrays in NumPy mit der flexiblen Datenmanipulation von Spreadsheets und relationalen Datenbanken (wie etwa SQL). Es bietet eine ausgefeilte 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.

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 die am weitesten verbreitete und daher ausgesprochen 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 16 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, Ihre Produktivität sowohl im interaktiven Arbeiten als auch in der Softwareentwicklung zu maximieren. Es unterstützt einen *Execute-Explore*-Workflow anstelle des typischen *Edit-Compile-Run*-Workflows, den man in vielen anderen Programmiersprachen pflegt. Außerdem bietet es einen einfachen Zugriff auf die Shell und das Dateisystem Ihres Betriebssystems. 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.

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. Auch andere Programmiersprachen haben Kernel für Jupyter implementiert, sodass Sie neben Python noch weitere Sprachen in Jupyter verwenden können.

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

In den Begleitmaterialien zu diesem Buch (*http://github.com/wesm/pydata-book*) finden Sie Jupyter-Notebooks mit allen Codebeispielen aus den einzelnen Kapiteln.

## SciPy

SciPy (*http://scipy.org*) ist eine Sammlung von Paketen, die sich mit einer Reihe von klassischen Problemfeldern beim wissenschaftlichen Rechnen befassen. Hier ist eine Auswahl der Pakete:

`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 2010 ist scikit-learn (*http://scikit-learn.org*) zum wichtigsten Machine-Learning-Toolkit für Python-Programmierer geworden. Mehr als 1.500 Menschen aus der ganzen Welt haben in den sieben Jahren 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 oder 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 benutzt.

## 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 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 kostenlose Anaconda-Distribution. Als ich das Buch geschrieben habe, gab es Anaconda sowohl mit Python 2.7 als auch mit 3.6. Das kann sich aber irgendwann ändern. Dieses Buch nutzt Python 3.6, verwenden Sie daher ebenfalls diese oder auch eine höhere Version.

### Windows

Um auf Windows loslegen zu können, laden Sie den Anaconda-Installer (*http://anaconda.com/downloads*) herunter. Ich empfehle Ihnen, den Installationsanweisungen für Windows zu folgen, die Sie auf der Anaconda-Downloadseite finden.

Stellen Sie nun sicher, dass alles korrekt konfiguriert ist. Um ein Kommandozeilenfenster zu öffnen, klicken Sie auf das Startmenü und gehen in die Eingabeaufforderung (auch bekannt als *cmd.exe*). Versuchen Sie, den Python-Interpreter zu starten, indem Sie **`python`** eintippen. Sie sollten eine Nachricht sehen, die Ihre soeben installierte Version von Anaconda angibt:

```
C:\Users\wesm>python
Python 3.5.2 |Anaconda 4.1.1 (64-bit)| (default, Jul  5 2016, 11:41:13)
[MSC v.1900 64 bit (AMD64)] on win32
>>>
```

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

### Apple (OS X, macOS)

Laden Sie den OS X-Anaconda-Installer herunter, der *Anaconda3-4.1.0-MacOSX-x86_64.pkg* oder ähnlich heißen sollte. Doppelklicken Sie zum Starten des Installers auf die *.pkg*-Datei. Wenn der Installer läuft, hängt er den ausführbaren Anaconda-Pfad automatisch an Ihre *.bash_profile*-Datei an. Diese befindet sich unter */Users/$USER/.bash_profile*.

Um zu prüfen, ob alles funktioniert, starten Sie IPython in einem Kommandozeilenfenster (Sie erhalten eine Kommandozeile im Programm Terminal):

```
$ ipython
```

Zum Verlassen der Shell drücken Sie Ctrl-D oder tippen **`exit()`** ein und drücken die 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 Anaconda verläuft das Einrichten ähnlich wie bei macOS. Der Installer ist ein Shell-Skript, das im Terminal ausgeführt werden muss. Je nachdem, ob Sie ein 32-Bit- oder 64-Bit-System haben, müssen Sie entweder den x86-(32-Bit-) oder den x86_64-(64-Bit-)Installer installieren. Sie haben dann eine Datei, die *Anaconda3-4.1.0-Linux-x86_64.sh* oder ähnlich heißt. Um sie zu installieren, führen Sie dieses Skript mit der bash-Shell aus:

```
$ bash Anaconda3-4.1.0-Linux-x86_64.sh
```

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

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

Der Anaconda-Installer fragt Sie möglicherweise, ob er Ihrer `$PATH`-Variablen das *bin/*-Verzeichnis voranstellen soll. Falls Sie nach der Installation irgendwelche Probleme haben, können Sie das auch selbst erledigen. Ändern Sie dazu Ihre *.bashrc* (oder *.zshrc*, falls Sie die zsh-Shell benutzen) folgendermaßen:

```
export PATH=/home/$USER/anaconda/bin:$PATH
```

Anschließend können Sie entweder einen neuen Terminalprozess starten oder Ihre *.bashrc* erneut ausführen mit `source ~/.bashrc`.

## Python-Pakete installieren oder aktualisieren

Irgendwann werden Sie beim Lesen dieses Buchs den Wunsch verspüren, Python-Pakete zu installieren, die nicht Teil der Anaconda-Distribution sind. Im Allgemeinen können Sie das mit dem folgenden Befehl erledigen:

```
conda install package_name
```

Sollte das nicht funktionieren, hilft Ihnen das pip-Paketverwaltungsprogramm:

```
pip install package_name
```

Sie können Pakete mit dem Befehl `conda update` aktualisieren:

```
conda update package_name
```

pip unterstützt außerdem Upgrades mit dem Flag `--upgrade`:

```
pip install --upgrade package_name
```

Im Laufe des Buchs werden Sie mehrfach Gelegenheit haben, diese Befehle auszuprobieren.

Sie können sowohl conda als auch pip benutzen, um Pakete zu installieren, Sie sollten jedoch nicht versuchen, conda-Pakete mit pip zu aktualisieren, da dies zu Umgebungsproblemen führen könnte. Bei der Benutzung von Anaconda oder Miniconda ist es am besten, wenn Sie die Aktualisierung zuerst mit conda versuchen.

## Python 2 und Python 3

Die erste Version der Python 3.x-Interpreter-Serie wurde Ende 2008 veröffentlicht. Sie brachte eine Reihe von Änderungen mit sich, die dazu führten, das älterer Python 2.x-Code inkompatibel wurde. Da seit der allerersten Veröffentlichung von Python im Jahr 1991 bereits 17 Jahre vergangen waren, wurde der »Schnitt« durch die Freigabe von Python 3 angesichts der Lehren aus dieser Zeit als positiver Schritt betrachtet.

2012 benutzte ein Großteil der Wissenschafts- und Datenanalysegemeinschaft immer noch Python 2.x, weil viele Pakete noch nicht vollständig kompatibel zu Python 3 waren. Daher setzte die 1. englischsprachige Auflage dieses Buchs ebenfalls auf Python 2.7, für die 1. deutsche Auflage war es auf Python 3.4 überarbeitet worden. Benutzer können nun jedoch frei zwischen Python 2.x und 3.x wählen und genießen im Allgemeinen in beiden Varianten volle Bibliotheksunterstützung.

Allerdings wird Python 2.x ab dem Jahr 2020 nicht mehr weiterentwickelt werden (und das schließt auch wichtige Sicherheitspatches ein). Daher ist es keine gute Idee, neue Projekte in Python 2.7 zu starten. Aus diesem Grund benutzen wir in diesem Buch Python 3.6, eine weitverbreitete, gut unterstützte stabile Version. Wir sind dazu übergegangen, Python 2.x als »Legacy Python« zu bezeichnen, Python 3.x dagegen nennen wir einfach »Python«. Ich rate Ihnen, das ganz einfach genauso zu machen.

Dieses Buch nutzt Python 3.6 als Grundlage. Sie haben vielleicht eine neuere Version, doch die Codebeispiele sollten aufwärtskompatibel sein. In Python 2.7 dagegen könnten manche Codebeispiele anders oder gar nicht funktionieren.

### Integrierte Entwicklungsumgebungen (Integrated Development Environments – IDEs) und Texteditoren

Wenn ich nach meiner Standardentwicklungsumgebung gefragt werde, antworte ich fast immer: »IPython sowie ein Texteditor«. Üblicherweise schreibe ich ein Programm, teste es schrittweise und debugge dann jedes Teil in IPython- oder Jupyter-Notebooks. Dazu ist es hilfreich, wenn man interaktiv mit den Daten herumspielen und visuell bestätigen kann, dass die Manipulationen am Datensatz wunschgemäß funktionieren. Bibliotheken wie pandas und NumPy sind extra für eine einfache Benutzung in der Shell vorgesehen.

Manche Nutzer ziehen jedoch eine reich ausgestattete IDE den vergleichsweise primitiven Texteditoren wie Emacs oder Vim vor. Hier sind einige, die Sie einmal ausprobieren könnten:

- PyDev (kostenlos), eine IDE, die auf der Eclipse-Plattform aufbaut
- PyCharm von JetBrains (für kommerzielle Benutzer abobasiert, für Open-Source-Entwickler kostenlos)
- Python Tools für Visual Studio (für Windows-Anwender)
- Spyder (kostenlos), eine in Anaconda enthaltene IDE
- Komodo IDE (kommerziell)

Aufgrund von Pythons Beliebtheit bieten die meisten Texteditoren, wie Atom und Sublime Text 2, ausgezeichnete Python-Unterstützung.

## 1.5 Community und Konferenzen

Abgesehen von einer Internetsuche sind die Mailinglisten für wissenschaftliches Python im Allgemeinen sehr hilfreich, wenn man Fragen hat. Schauen Sie sich zum Beispiel folgende an:

- pydata: eine Google-Groups-Liste für Fragen zu Python für die Datenanalyse und pandas
- pystatsmodels: für Fragen zu statsmodels oder pandas
- Mailingliste für scikit-learn (*scikit-learn@python.org*) und Machine Learning in Python ganz allgemein
- numpy-discussion: für NumPy-bezogene Fragen
- scipy-user: für allgemeine Fragen zu SciPy oder wissenschaftliches Python

Ich habe bewusst keine URLs für diese Listen angegeben, da sich diese ändern könnten. Sie finden sie aber leicht über eine Internetsuche.

Jedes Jahr finden auf der ganzen Welt Konferenzen für Python-Programmierer statt. Wenn Sie andere Python-Programmierer treffen wollen, die Ihre Interessen teilen, sollten Sie nach Möglichkeit eine dieser Konferenzen besuchen. Viele der