

O'REILLY®

ÜBERSETZUNG DER
2. Auflage

Handbuch Data Science mit Python

Grundlegende Tools
für die Arbeit mit Daten



Jake VanderPlas

Übersetzung von Knut Lorenzen
und Jørgen W. Lang

Copyright und Urheberrechte:

Die durch die dpunkt.verlag GmbH vertriebenen digitalen Inhalte sind urheberrechtlich geschützt. Der Nutzer verpflichtet sich, die Urheberrechte anzuerkennen und einzuhalten. Es werden keine Urheber-, Nutzungs- und sonstigen Schutzrechte an den Inhalten auf den Nutzer übertragen. Der Nutzer ist nur berechtigt, den abgerufenen Inhalt zu eigenen Zwecken zu nutzen. Er ist nicht berechtigt, den Inhalt im Internet, in Intranets, in Extranets oder sonst wie Dritten zur Verwertung zur Verfügung zu stellen. Eine öffentliche Wiedergabe oder sonstige Weiterveröffentlichung und eine gewerbliche Vervielfältigung der Inhalte wird ausdrücklich ausgeschlossen. Der Nutzer darf Urheberrechtsvermerke, Markenzeichen und andere Rechtsvorbehalte im abgerufenen Inhalt nicht entfernen.

Handbuch Data Science mit Python

Grundlegende Tools für die Arbeit mit Daten

Jake VanderPlas

*Deutsche Übersetzung von
Knut Lorenzen und Jørgen W. Lang*

O'REILLY®

Jake VanderPlas

Lektorat: Alexandra Follenius

Übersetzung: Knut Lorenzen, Jörgen W. Lang

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

Druck und Bindung: mediaprint solutions 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 978-3-96009-225-4

PDF 978-3-96010-812-2

ePub 978-3-96010-813-9

mobi 978-3-96010-814-6

1. Auflage 2024

Translation Copyright für die deutschsprachige Ausgabe © 2024 dpunkt.verlag GmbH

Wieblinger Weg 17

69123 Heidelberg

Authorized German translation of the English edition of *Python Data Science Handbook, 2nd Edition*, ISBN 9781098121228 © 2023 Jake VanderPlas. 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.

Hinweis:

Dieses Buch wurde mit mineralölfreien Farben auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie. Hergestellt in Deutschland.



Schreiben Sie uns:

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

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen.

Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Alle Angaben und Programme in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autor noch Übersetzer noch Verlag können jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.

Inhalt

Einleitung	13
-------------------------	-----------

Teil I: Mehr als normales Python: Jupyter

1 Der Einstieg in IPython und Jupyter	21
Die IPython-Shell starten	21
Das Jupyter Notebook starten	22
Hilfe und Dokumentation in IPython	22
Tastaturkürzel in der IPython-Shell	27
2 Erweiterte interaktive Features	31
Magische Befehle in IPython	31
Verlauf der Ein- und Ausgabe	33
IPython und Shell-Befehle	36
3 Debugging und Profiling	41
Fehler und Debugging	41
Profiling und Timing von Code	45
Weitere IPython-Ressourcen	50

Teil II: Einführung in NumPy

4 Die Datentypen in Python	55
Python-Integer sind mehr als nur ganzzahlige Werte	56
Python-Listen sind mehr als nur einfache Listen	57
Arrays feststehenden Typs in Python	59
Arrays anhand von Listen erzeugen	59
Neue Arrays erzeugen	60
NumPys Standarddatentypen	61
5 Grundlagen von NumPy-Arrays	63
Attribute von NumPy-Arrays	63
Indizierung von Arrays: Zugriff auf einzelne Elemente	64
Slicing: Teilmengen eines Arrays auswählen	65
Arrays umformen	68
Arrays verketteten und aufteilen	69

6	Berechnungen mit NumPy-Arrays: universelle Funktionen	71
	Langsame Schleifen	71
	Kurz vorgestellt: UFuncs	72
	NumPys UFuncs im Detail	73
	UFunc-Features für Fortgeschrittene	77
	UFuncs: mehr erfahren	79
7	Aggregationen: Minimum, Maximum und alles dazwischen	81
	Summieren der Werte eines Arrays	81
	Minimum und Maximum	82
	Beispiel: Durchschnittliche Größe der US-Präsidenten	84
8	Berechnungen mit Arrays: Broadcasting	87
	Kurz vorgestellt: Broadcasting	87
	Für das Broadcasting geltende Regeln	89
	Broadcasting in der Praxis	91
9	Vergleiche, Maskierungen und boolesche Logik	95
	Beispiel: Regentage zählen	95
	Vergleichsoperatoren als UFuncs	96
	Boolesche Arrays verwenden	98
	Boolesche Arrays als Maskierungen	100
	Verwendung der Schlüsselwörter »and« bzw. »or« und der Operatoren & bzw.	101
10	Fancy Indexing	103
	Fancy Indexing im Detail	103
	Kombinierte Indizierung	105
	Beispiel: Auswahl zufälliger Punkte	105
	Werte per Fancy Indexing modifizieren	107
	Beispiel: Daten gruppieren	108
11	Arrays sortieren	111
	Schnelle Sortierung in NumPy: np.sort und np.argsort	112
	Nach Zeilen und Spalten sortieren	112
	Teilsortierungen: Partitionierung	113
	Beispiel: k nächste Nachbarn	113
12	Strukturierte Daten: NumPys strukturierte Arrays	117
	Strukturierte Arrays erzeugen	118
	Erweiterte zusammengesetzte Typen	119
	Record-Arrays: strukturierte Arrays mit Pfiff	120
	Weiter mit Pandas	120

Teil III: Datenbearbeitung mit Pandas

13 Kurz vorgestellt: Pandas-Objekte	125
Das Pandas-Series-Objekt	125
Das Pandas-DataFrame-Objekt	128
Das Pandas-Index-Objekt	131
14 Daten indizieren und auswählen	133
Series-Daten auswählen	133
DataFrame-Daten auswählen	136
15 Mit Pandas-Daten arbeiten	141
UFuncs: Indexerhaltung	141
UFuncs: Indexanpassung	142
UFuncs: Operationen mit DataFrame und Series	144
16 Handhabung fehlender Daten	147
Kompromisse beim Umgang mit fehlenden Daten	147
Fehlende Daten in Pandas	148
Pandas nullfähige Datentypen	151
Mit Nullwerten arbeiten	152
17 Hierarchische Indizierung	157
Mehrfach indizierte Series	157
Methoden zum Erzeugen eines MultiIndex	161
Indizierung und Slicing eines MultiIndex	163
Multi-Indizes umordnen	166
18 Datenmengen kombinieren: concat und append	171
Verkettung von NumPy-Arrays	172
Einfache Verkettungen mit pd.concat	172
19 Datenmengen kombinieren: merge und join	177
Relationale Algebra	177
Join-Kategorien	178
Angabe der zu verknüpfenden Spalten	180
Mengenarithmetik bei Joins	183
Konflikte bei Spaltennamen: das Schlüsselwort suffixes	184
Beispiel: Daten von US-Bundesstaaten	185
20 Aggregation und Gruppierung	191
Planetendaten	191
Einfache Aggregationen in Pandas	192
GroupBy: Aufteilen, Anwenden und Kombinieren	194

21	Pivot-Tabellen	203
	Gründe für Pivot-Tabellen	203
	Pivot-Tabellen von Hand erstellen	204
	Die Syntax von Pivot-Tabellen	204
	Beispiel: Geburtenraten	207
22	Vektorisierte String-Operationen	213
	Kurz vorgestellt: String-Operationen in Pandas	213
	Liste der Pandas-String-Methoden	214
	Beispiel: Rezeptdatenbank	218
23	Zeitreihen verwenden	223
	Kalenderdaten und Zeiten in Python	223
	Zeitreihen in Pandas: Indizierung durch Zeitangaben	227
	Datenstrukturen für Zeitreihen in Pandas	227
	Gleichförmige Sequenzen: pd.date_range	228
	Häufigkeiten und Abstände	229
	Resampling, zeitliches Verschieben und geglättete Statistik	231
	Beispiel: Visualisierung von Fahrradzahlungen in Seattle	236
24	Leistungsstarkes Pandas: eval und query	243
	Der Zweck von query und eval: zusammengesetzte Ausdrücke	243
	Effiziente Operationen mit pandas.eval	244
	DataFrame.eval für spaltenweise Operationen	246
	Die DataFrame.query-Methode	248
	Performance: Wann eval und query verwendet werden sollten	248
	Weitere Ressourcen	249

Teil IV: Visualisierung mit Matplotlib

25	Allgemeine Tipps zu Matplotlib	253
	Matplotlib importieren	253
	Stil einstellen	253
	show oder kein show? – Anzeige von Diagrammen	253
26	Einfache Liniendiagramme	261
	Anpassen des Diagramms: Linienfarben und -stile	264
	Anpassen des Diagramms: Begrenzungen	266
	Diagramme beschriften	268
	Stolpersteine in Matplotlib	270

27 Einfache Streudiagramme	271
Streudiagramme mit plt.plot erstellen	271
Streudiagramme mit plt.scatter erstellen	273
plot kontra scatter: eine Anmerkung zur Effizienz	276
Visualisierung von Messunsicherheiten	276
28 Dichtediagramme und Konturdiagramme	281
Visualisierung einer dreidimensionalen Funktion	281
Histogramme, Binnings und Dichte	285
Zweidimensionale Histogramme und Binnings	287
29 Anpassen der Legende	291
Legendenelemente festlegen	293
Legenden mit Punktgrößen	295
Mehrere Legenden	296
30 Anpassen von Farbskalen	299
Farbskala anpassen	300
Beispiel: Handgeschriebene Ziffern	304
31 Untergeordnete Diagramme	307
plt.axes: untergeordnete Diagramme von Hand erstellen	307
plt.subplot: untergeordnete Diagramme in einem Raster anordnen	309
plt.subplots: das gesamte Raster gleichzeitig ändern	310
plt.GridSpec: kompliziertere Anordnungen	312
32 Text und Beschriftungen	315
Beispiel: Auswirkungen von Feiertagen auf die Geburtenzahlen in den USA	315
Transformationen und Textposition	317
Pfeile und Beschriftungen	319
33 Achsenmarkierungen anpassen	323
Vorrangige und nachrangige Achsenmarkierungen	323
Markierungen oder Beschriftungen verbergen	325
Anzahl der Achsenmarkierungen verringern oder erhöhen	326
Formatierung der Achsenmarkierungen	328
Zusammenfassung der Formatter- und Locator-Klassen	330
34 Matplotlib anpassen: Konfigurationen und Stylesheets	333
Diagramme von Hand anpassen	333
Voreinstellungen ändern: rcParams	335

35	Dreidimensionale Diagramme in Matplotlib	341
	Dreidimensionale Punkte und Linien	342
	Dreidimensionale Konturdiagramme	343
	Drahtgitter- und Oberflächendiagramme	344
	Triangulation von Oberflächen	346
	Beispiel: Visualisierung eines Möbiusbands	348
36	Visualisierung mit Seaborn	351
	Seaborn-Diagramme	352
	Kategoriale Diagramme	356
	Beispiel: Ergebnisse eines Marathonlaufs	359
	Weiterführende Ressourcen	365
	Weitere Grafikbibliotheken für Python	366

Teil V: Machine Learning

37	Was ist Machine Learning?	369
	Kategorien des Machine Learning	369
	Qualitative Beispiele für Machine-Learning-Anwendungen	370
	Zusammenfassung	379
38	Kurz vorgestellt: Scikit-Learn	381
	Datenrepräsentierung in Scikit-Learn	381
	Die Estimator-API	384
	Anwendung: Handgeschriebene Ziffern untersuchen	392
	Zusammenfassung	397
39	Hyperparameter und Modellvalidierung	399
	Überlegungen zum Thema Modellvalidierung	399
	Auswahl des besten Modells	403
	Lernkurven	410
	Validierung in der Praxis: Rastersuche	414
	Zusammenfassung	415
40	Feature Engineering	417
	Kategoriale Features	417
	Texte als Features	418
	Bilder als Features	420
	Abgeleitete Features	420
	Vervollständigung fehlender Daten	422
	Feature-Pipelines	423

41 Ausführlich: Naive Bayes-Klassifikation	425
Bayes-Klassifikation	425
Gaußsche naive Bayes-Klassifikation	426
Multinomiale naive Bayes-Klassifikation	429
Einsatzgebiete für naive Bayes-Klassifikation	432
42 Ausführlich: Lineare Regression	435
Einfache lineare Regression	435
Regression der Basisfunktion	437
Regularisierung	441
Beispiel: Vorhersage des Fahrradverkehrs	445
43 Ausführlich: Support Vector Machines	451
Gründe für Support Vector Machines	451
Support Vector Machines: Maximierung des Randbereichs	453
Beispiel: Gesichtserkennung	461
Zusammenfassung	465
44 Ausführlich: Entscheidungsbäume und Random Forests	467
Gründe für Random Forests: Entscheidungsbäume	467
Estimator-Ensembles: Random Forests	471
Random-Forest-Regression	473
Beispiel: Random Forest zur Klassifikation handgeschriebener Ziffern	475
Zusammenfassung	477
45 Ausführlich: Hauptkomponentenanalyse	479
Hauptkomponentenanalyse: ein Überblick	479
Hauptkomponentenanalyse als Rauschfilter	487
Beispiel: Eigengesichter	489
Zusammenfassung	492
46 Ausführlich: Manifold Learning	493
Manifold Learning: »HELLO«	494
Multidimensionale Skalierung	495
Nichtlineare Mannigfaltigkeiten: lokal lineare Einbettung	500
Überlegungen zum Thema Manifold-Methoden	502
Beispiel: Isomap und Gesichter	503
Beispiel: Visualisierung der Strukturen in Zifferndaten	507
47 Ausführlich: k-Means-Clustering	511
Kurz vorgestellt: der k-Means-Algorithmus	511
Expectation-Maximization	513
Beispiele	518

48 Ausführlich: Gaußsche Mixture-Modelle	525
Gründe für GMM: Schwächen von k-Means	525
EM-Verallgemeinerung: gaußsche Mixture-Modelle	528
Wahl des Kovarianztyps	532
GMM als Dichteschätzung	532
Beispiel: GMM zum Erzeugen neuer Daten verwenden	536
49 Ausführlich: Kerndichteschätzung	539
Gründe für Kerndichteschätzung: Histogramme	539
Kerndichteschätzung in der Praxis	543
Auswahl der Bandbreite durch Kreuzvalidierung	544
Beispiel: Nicht ganz so naive Bayes-Klassifikation	545
50 Anwendung: Eine Gesichtserkennungspipeline	551
HOG-Features	552
HOG in Aktion: eine einfache Gesichtserkennung	553
Vorbehalte und Verbesserungen	557
Weitere Machine-Learning-Ressourcen	559
Index	561

Was ist Data Science?

In diesem Buch geht es darum, Data Science mithilfe von Python zu betreiben, daher stellt sich unmittelbar die Frage: Was ist *Data Science* überhaupt? Das genau zu definieren, erweist sich als überraschend schwierig, insbesondere in Anbetracht der Tatsache, wie geläufig dieser Begriff inzwischen geworden ist. Von lautstarken Kritikern wird er mitunter als eine überflüssige Bezeichnung abgetan (denn letzten Endes kommt keine Wissenschaft ohne Daten aus) oder für ein leeres Schlagwort gehalten, das lediglich dazu dient, Lebensläufe aufzupolieren, um die Aufmerksamkeit über-eifriger Personalverantwortlicher zu erlangen.

Meiner Ansicht nach übersehen diese Kritiker dabei einen wichtigen Punkt. Trotz des mit dem Begriff einhergehenden Hypes ist Data Science wohl die beste Beschreibung für fachübergreifende Fähigkeiten, die in vielen Anwendungsbereichen in Wirtschaft und Wissenschaft immer wichtiger werden. Entscheidend ist hier die *Interdisziplinarität*: Ich halte Drew Conways Venn-Diagramm, das er im September 2010 erstmals in seinem Blog veröffentlichte, für die beste Definition von Data Science (siehe Abbildung 1).

Zwar sind einige der Bezeichnungen für die Schnittmengen etwas überspitzt formuliert, aber dennoch erfasst dieses Diagramm das Wesentliche dessen, was gemeint ist, wenn man von »Data Science« spricht: Es handelt sich um ein grundlegend interdisziplinäres Thema. Data Science umfasst drei verschiedene und sich überschneidende Bereiche: die *Statistik*, um (immer größer werdende) Datenmengen modellieren und zusammenfassen zu können, die *Informatik*, um Algorithmen für die effiziente Speicherung, Verarbeitung und Visualisierung dieser Daten entwerfen zu können, und das erforderliche *Fachwissen* (das wir uns als das »klassisch« Erlernte eines Fachgebiets vorstellen können), um sowohl die angemessenen Fragen zu stellen als auch die Antworten im richtigen Kontext zu bewerten.

In diesem Sinne möchte ich Sie ermutigen, Data Science nicht als ein neu zu erlernendes Fachwissensgebiet zu begreifen, sondern als neue Fähigkeiten, die Sie im Rahmen Ihres vorhandenen Fachwissens anwenden können. Ob Sie über Wahlergebnisse berichten, Aktienrenditen vorhersagen, Mausclicks auf Onlinewerbung

optimieren, Mikroorganismen auf Mikroskopbildern identifizieren, nach neuen Arten astronomischer Objekte suchen oder mit irgendwelchen anderen Daten arbeiten: Ziel dieses Buchs ist es, Ihnen die Fähigkeit zu vermitteln, neuartige Fragen über das von Ihnen gewählte Fachgebiet zu stellen und diese zu beantworten.

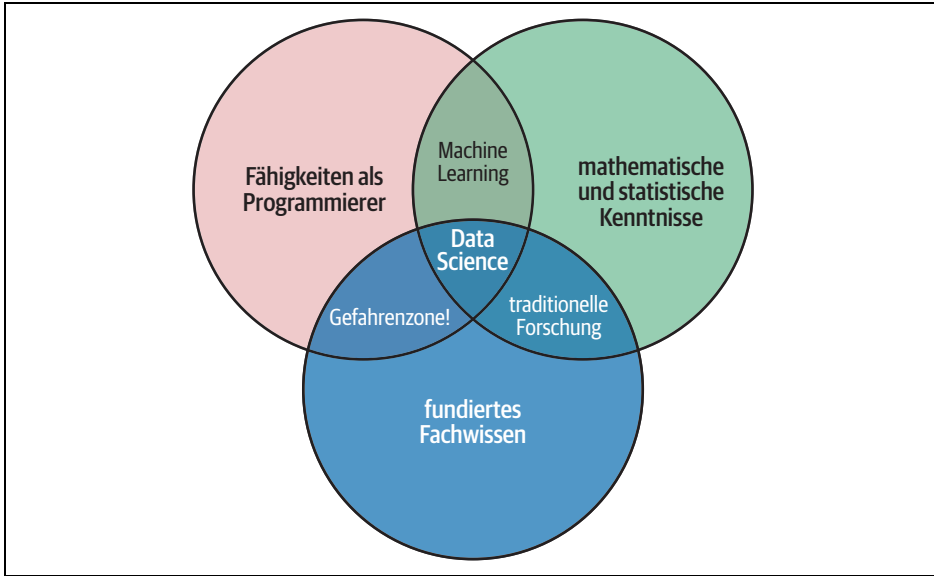


Abbildung 1: Das Venn-Diagramm zur Data Science von Drew Conway (Quelle: Drew Conway (<https://oreil.ly/PkOOw>), Abdruck mit freundlicher Genehmigung)

An wen richtet sich dieses Buch?

Sowohl in meinen Vorlesungen an der Universität Washington als auch auf verschiedenen technisch orientierten Konferenzen und Treffen wird mir am häufigsten diese Frage gestellt: »Wie kann man Python am besten erlernen?« Bei den Fragenden handelt es sich im Allgemeinen um technisch interessierte Studenten, Entwicklerinnen oder Forscher, die oftmals schon über umfangreiche Erfahrung mit dem Schreiben von Code und der Verwendung von rechnergestützten und numerischen Tools verfügen. Die meisten dieser Personen möchten Python erlernen, um die Programmiersprache als Tool für datenintensive und rechnergestützte wissenschaftliche Aufgaben zu nutzen. Für diese Zielgruppe ist eine Vielzahl von Lernvideos, Blogbeiträgen und Tutorials online verfügbar. Allerdings frustriert mich bereits seit geraumer Zeit, dass es auf obige Frage keine wirklich eindeutige und gute Antwort gibt – und das war der Anlass für dieses Buch.

Das Buch ist nicht als Einführung in Python oder die Programmierung im Allgemeinen gedacht. Ich setze voraus, dass die Leserinnen und Leser mit der Programmiersprache Python vertraut ist. Dazu gehören das Definieren von Funktionen, die Zuweisung von Variablen, das Aufrufen der Methoden von Objekten, die Steuerung des Programmablaufs und weitere grundlegende Aufgaben. Das Buch soll vielmehr

Python-Usern dabei helfen, die zum Betreiben von Data Science verfügbaren Pakete zu nutzen – Bibliotheken wie die im folgenden Abschnitt genannten und ähnliche Tools –, um Daten effektiv zu speichern, zu handhaben und Einblick in diese Daten zu gewinnen.

Warum Python?

Python hat sich in den vergangenen Jahrzehnten zu einem erstklassigen Tool für wissenschaftliche Berechnungen entwickelt, insbesondere auch für die Analyse und Visualisierung großer Datensätze. Die ersten Anhänger der Programmiersprache Python dürfte das ein wenig überraschen: Beim eigentlichen Design der Sprache wurde weder der Datenanalyse noch wissenschaftlichen Berechnungen besondere Beachtung geschenkt. Dass sich Python für die Data Science als so nützlich erweist, ist vor allem dem großen und aktiven Ökosystem der Programmpakete von Drittherstellern zu verdanken: Da gibt es *NumPy* für die Handhabung gleichartiger Array-basierter Daten, *Pandas* für die Verarbeitung verschiedenartiger und gelabelter Daten, *SciPy* für gängige wissenschaftliche Berechnungen, *Matplotlib* für druckreife Visualisierungen, *IPython* für die interaktive Ausführung und zum Teilen von Code, *Scikit-Learn* für Machine Learning sowie viele weitere Tools, die auf den folgenden Seiten vorgestellt werden.

Falls Sie auf der Suche nach einer Einführung in die Programmiersprache Python sind, empfehle ich das dieses Buch ergänzende Projekt *A Whirlwind Tour of Python* (<https://oreil.ly/jFtWj>). Bei diesem kurzen Bericht handelt es sich um eine Tour durch die wesentlichen Features der Sprache Python, die sich an Data Scientists richtet, die bereits mit anderen Programmiersprachen vertraut sind.

Inhaltsübersicht

Alle nummerierten Teile in diesem Buch konzentrieren sich auf ein bestimmtes Paket oder Tool, das für die mit Python betriebene Data Science von grundlegender Bedeutung ist. Sie sind in eigenständige Kapitel unterteilt, die jeweils ein bestimmtes Konzept behandeln.

- Teil I, »Mehr als normales Python: Jupyter«, stellt IPython und Jupyter vor. Diese Pakete bieten eine Umgebung für Berechnungen, die von vielen Data Scientists genutzt wird, die Python einsetzen.
- Teil II, »Einführung in NumPy«, konzentriert sich auf die NumPy-Bibliothek, die das `ndarray` bereitstellt, das ein effizientes Speichern und die Handhabung dicht gepackter Daten-Arrays in Python ermöglicht.
- Teil III, »Datenbearbeitung mit Pandas«, stellt die Pandas-Bibliothek vor. Sie verfügt über das `DataFrame`-Objekt, das ein effizientes Speichern und die Handhabung gelabelter bzw. spaltenorientierter Daten in Python gestattet.

- Teil IV, »Visualisierung mit Matplotlib«, konzentriert sich auf Matplotlib, eine Bibliothek, die flexible und vielfältige Visualisierungen von Daten in Python ermöglicht.
- Teil V, »Machine Learning«, zeigt die Bibliothek Scikit-Learn. Sie stellt eine effiziente Implementierung der wichtigsten und gebräuchlichsten Machine-Learning-Algorithmen zur Verfügung.

Natürlich umfasst die PyData-Welt viel mehr als diese sechs Pakete – und sie wächst mit jedem Tag weiter. Ich werde mich im Folgenden daher bemühen, Hinweise auf andere interessante Projekte, Bestrebungen und Pakete zu geben, die die Grenzen des mit Python Machbaren erweitern. Dessen ungeachtet sind die Pakete, auf die ich mich hier konzentriere, derzeit für viele der mit Python möglichen Aufgaben der Data Science von grundlegender Bedeutung, und ich erwarte, dass sie wichtig bleiben, auch wenn das sie umgebende Ökosystem weiterhin wächst.

Installation der Software

Die Installation von Python und den für wissenschaftliche Berechnungen erforderlichen Bibliotheken ist unkompliziert. In diesem Abschnitt finden Sie einige Überlegungen, denen Sie bei der Einrichtung Ihres Computers Beachtung schenken sollten.

Es gibt zwar verschiedene Möglichkeiten, Python zu installieren, allerdings empfehle ich zum Betreiben von Data Science die Anaconda-Distribution, die unter Windows, Linux und macOS auf ähnliche Weise funktioniert. Es gibt zwei Varianten der Anaconda-Distribution:

- Miniconda (<https://oreil.ly/dH7wJ>) besteht aus dem eigentlichen Python-Interpreter und einem Kommandozeilenprogramm namens *conda*, das als plattformübergreifender Paketmanager für Python-Pakete fungiert. Das Programm arbeitet in ähnlicher Weise wie die Tools *apt* oder *yum*, die Linux-Usern bekannt sein dürften.
- Anaconda (<https://oreil.ly/ndxjm>) enthält sowohl Python als auch *conda* und darüber hinaus eine Reihe vorinstallierter Pakete, die für wissenschaftliche Berechnungen konzipiert sind. Aufgrund der Größe dieser Pakete müssen Sie davon ausgehen, dass die Installation mehrere Gigabyte Speicherplatz auf der Festplatte belegt.

Alle in Anaconda enthaltenen Pakete können auch nachträglich der Miniconda-Installation hinzugefügt werden. Daher empfehle ich, mit Miniconda anzufangen.

Laden Sie zunächst das Miniconda-Paket herunter und installieren Sie es. Vergewissern Sie sich, dass Sie eine Version auswählen, die Python 3 enthält. Installieren Sie dann die in diesem Buch verwendeten Pakete:

```
[~]$ conda install numpy pandas scikit-learn matplotlib seaborn jupyter
```

Wir werden im gesamten Buch noch weitere, spezialisiertere Tools einsetzen, die zum wissenschaftlich orientierten Ökosystem in Python gehören. Für gewöhnlich ist

zur Installation lediglich eine Eingabe wie `conda install packagename` nötig. Sollten Ihnen einmal Pakete begegnen, die über den Standard-*conda*-Kanal nicht erhältlich sind, sollten Sie auf jeden Fall auf *conda-forge* (<https://oreil.ly/CCvwQ>) nachsehen, einem vielfältigen, von der Community gepflegten Repository mit *conda*-Paketen. Weitere Informationen über *conda*, beispielsweise über das Erstellen und Verwenden von *conda*-Umgebungen (die ich nur *nachdrücklich* empfehlen kann), finden Sie in der Onlinedokumentation (<https://oreil.ly/MkqPw>).

In diesem Buch verwendete Konventionen

In diesem Buch gelten die folgenden typografischen Konventionen:

Kursiv

Kennzeichnet neue Begriffe, URLs, Dateinamen und Dateierweiterungen.

Nicht proportionale Schrift

Wird für Programmlistings und im Fließtext verwendet, um Programmbestandteile wie Variablen- oder Funktionsbezeichnungen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter zu kennzeichnen.

Fette nicht proportionale Schrift

Kommandos oder sonstiger Text, der vom User buchstabengetreu eingegeben werden soll.

Kursive nicht proportionale Schrift

Text, der durch eigene Werte oder durch kontextabhängige Werte zu ersetzen ist.



Dieses Element steht für einen allgemeinen Hinweis.

Verwendung der Codebeispiele

Unter <http://github.com/jakevdp/PythonDataScienceHandbook> steht ergänzendes Material (Codebeispiele, Abbildungen usw.) zum Herunterladen zur Verfügung.

Dieses Buch soll Ihnen helfen, Ihre Arbeit zu erledigen. Den im Buch aufgeführten Code können Sie generell in Ihren eigenen Programmen und der Dokumentation verwenden. Sie brauchen uns nicht um Erlaubnis zu fragen, solange Sie nicht erhebliche Teile des Codes nutzen. Wenn Sie beispielsweise ein Programm schreiben, das einige der im Buch aufgeführten Codeschnipsel verwendet, benötigen Sie dafür keine Erlaubnis. Der Verkauf oder Vertrieb von Beispielen aus O'Reilly-Büchern bedarf hingegen einer Genehmigung. Das Beantworten von Fragen durch Verwendung von Zitaten oder Beispielcode aus diesem Buch muss nicht extra genehmigt werden. Die Verwendung erheblicher Teile des Beispielcodes in der Dokumentation Ihres eigenen Produkts erfordert jedoch eine Genehmigung.

Wir freuen uns über Quellennennungen, machen sie jedoch nicht zur Bedingung. Üblich ist die Nennung von Titel, Autor(en), Verlag, Erscheinungsjahr und ISBN, also beispielsweise: »*Handbuch Data Science mit Python, 2. Auflage* von Jake VanderPlas, O'Reilly 2024, ISBN 978-3-96009-225-4«.

Wenn Sie glauben, dass sich die Verwendung der Codebeispiele außerhalb der Fair-Use-Prinzipien oder der oben erwähnten Erlaubnis liegt, kontaktieren Sie uns bitte per E-Mail an komentar@oreilly.de.

Mehr als normales Python: Jupyter

Für Python stehen viele verschiedene Entwicklungsumgebungen zur Verfügung, und häufig werde ich gefragt, welche ich für meine eigenen Arbeiten verwende. Einige Menschen überrascht die Antwort: Meine bevorzugte Entwicklungsumgebung ist IPython (<http://ipython.org>) in Kombination mit einem Texteditor (entweder Emacs oder VSCode – das hängt von meiner Stimmung ab). Jupyter begann als die IPython-Shell, die 2001 von Fernando Perez in Form eines erweiterten Python-Interpreters ins Leben gerufen wurde und sich seither zu einem Projekt entwickelt hat, »Tools für den gesamten Lebenszyklus in der forschenden Informatik« – so Perez' eigene Worte – bereitzustellen. Wenn man Python als Motor einer Aufgabe von Data Science betrachtet, können Sie sich Jupyter als die interaktive Steuerkonsole dazu vorstellen

Jupyter ist nicht nur eine nützliche interaktive Schnittstelle zu Python, sondern stellt darüber hinaus eine Reihe praktischer syntaktischer Erweiterungen der Sprache bereit. Die nützlichsten dieser Erweiterungen werden wir hier erörtern. Die bekannteste, vom Jupyter-Projekt bereitgestellte Schnittstelle ist wohl das Jupyter Notebook, eine browserbasierte Umgebung, die für die Entwicklung, die Zusammenarbeit, das Teilen und sogar die Veröffentlichung von Ergebnissen der Data Science gute Dienste leistet. Um ein Beispiel für die Nützlichkeit dieses Notebook-Formats zu geben: Betrachten Sie einfach nur die Seite, die Sie gerade lesen. Das vollständige Manuskript dieses Buchs wurde in Form einer Reihe von Jupyter Notebooks verfasst.

In diesem Teil des Buchs werden wir zunächst einige der Features von Jupyter betrachten, die sich in der Praxis der Data Science als nützlich erweisen. Der Schwerpunkt liegt hierbei auf der bereitgestellten Syntax, die mehr zu bieten hat als die Standardfeatures von Python. Anschließend werden wir uns etwas eingehender mit einigen der sehr nützlichen *magischen Befehle* befassen, die gängige Aufgaben bei der Erstellung und Verwendung des Data-Science-Codes beschleunigen können. Zum Abschluss erörtern wir dann einige der Features des Notebooks, die dem Verständnis der Daten und dem Teilen der Ergebnisse dienen können.

Der Einstieg in IPython und Jupyter

Beim Schreiben von Code für Data Science verfolge ich normalerweise drei Arbeitsweisen: Ich benutze die IPython-Shell, um kurze Folgen von Befehlen auszuprobieren, das Jupyter Notebook für längere interaktive Analysen und um Inhalte mit anderen zu teilen sowie interaktive Entwicklungsumgebungen (IDEs) wie Emacs oder VSCode, um wiederverwendbare Python-Pakete zu erstellen. Dieses Kapitel konzentriert sich auf die ersten beiden Methoden: die IPython-Shell und das Jupyter Notebook. Der Einsatz einer IDE für die Softwareentwicklung ist ein wichtiges drittes Werkzeug im Repertoire der Data Scientists, auf das wir hier aber nicht direkt eingehen.

Die IPython-Shell starten

Wie die meisten Teile dieses Buchs sollte auch dieser nicht passiv gelesen werden. Ich empfehle Ihnen, während der Lektüre mit den vorgestellten Tools und der angegebenen Syntax herumzuexperimentieren. Die durch das Nachvollziehen der Beispiele erworbenen Fingerfertigkeiten werden sich als sehr viel nützlicher erweisen, als wenn Sie nur darüber lesen. Geben Sie auf der Kommandozeile `ipython` ein, um den Python-Interpreter zu starten. Sollten Sie eine Distribution wie Anaconda oder EPD (*Enthought Python Distribution*) installiert haben, können Sie möglicherweise alternativ einen systemspezifischen Programmstarter verwenden.

Nach dem Start des Interpreters sollte Ihnen eine Eingabeaufforderung wie die folgende angezeigt werden:

```
Python 3.9.2 (v3.9.2:1a79785e3e, Feb 19 2021, 09:06:10)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.21.0 -- An enhanced Interactive Python. Type '?' for help.
```

```
In [1]:
```

Nun können Sie fortfahren.

Das Jupyter Notebook starten

Das Jupyter Notebook ist eine browserbasierte grafische Schnittstelle für die IPython-Shell und besitzt eine große Vielfalt dynamischer Anzeigemöglichkeiten. Neben der Ausführung von Python-/IPython-Anweisungen gestattet das Notebook dem User das Einfügen von formatiertem Text, statischen und dynamischen Visualisierungen, mathematischen Formeln, JavaScript-Widgets und vielem mehr. Darüber hinaus können die Dokumente in einem Format gespeichert werden, das anderen Usern ermöglicht, sie auf ihren eigenen Systemen zu öffnen und den Code auszuführen.

Das Jupyter Notebook wird zwar in einem Fenster Ihres Webbrowsers angezeigt und bearbeitet, allerdings ist eine Verbindung zu einem laufenden Python-Prozess erforderlich, um Code auszuführen. Geben Sie in Ihrer System-Shell folgenden Befehl ein, um diesen Prozess (der als »Kernel« bezeichnet wird) zu starten:

```
$ jupyter lab
```

Dieser Befehl startet einen lokalen Webserver, auf den Ihr Browser zugreifen kann. Er gibt sofort einige Meldungen aus, die zeigen, was vor sich geht. Dieses Log sieht in etwa folgendermaßen aus:

```
$ jupyter lab
[ServerApp] Serving notebooks from local directory: /Users/jakevdp/ \
PythonDataScienceHandbook
[ServerApp] Jupyter Server 1.4.1 is running at:
[ServerApp] http://localhost:8888/lab?token=dd852649
[ServerApp] Use Control-C to stop this server and shut down all kernels
(twice to skip confirmation).
```

Nach der Eingabe des Befehls sollte sich automatisch Ihr Standardbrowser öffnen und die genannte lokale URL anzeigen. Die genaue Adresse ist von Ihrem System abhängig. Öffnet sich Ihr Browser nicht automatisch, können Sie von Hand ein Browserfenster öffnen und die Adresse (in diesem Beispiel *http://localhost:8888/lab/*) eingeben.

Hilfe und Dokumentation in IPython

Auch wenn Sie die anderen Abschnitte dieses Kapitels überspringen, sollten Sie doch wenigstens diesen lesen: Ich habe festgestellt, dass die hier erläuterten IPython-Tools den größten Einfluss auf meinen alltäglichen Arbeitsablauf haben.

Wenn ein technologisch interessierter Mensch darum gebeten wird, einem Freund, einem Familienmitglied oder einer Kollegin bei einem Computerproblem zu helfen, geht es meistens gar nicht darum, die Lösung zu kennen, sondern zu wissen, wie man schnell eine noch unbekannte Lösung findet. Mit Data Science verhält es sich ähnlich: Durchsuchbare Webressourcen wie Onlinedokumentationen, Mailinglisten und auf Stackoverflowbusiness.com gefundene Antworten enthalten jede Menge Informationen, auch (oder gerade?) wenn es sich um ein Thema handelt, nach dem

Sie selbst schon einmal gesucht haben. Für einen leistungsfähigen Praktiker der Data Science geht es weniger darum, das in jeder erdenklichen Situation einzusetzende Tool oder den geeigneten Befehl auswendig zu lernen, sondern vielmehr darum, zu wissen, wie man die benötigten Informationen schnell und einfach findet – sei es nun mithilfe einer Suchmaschine oder auf anderem Weg.

Zwischen dem User und der erforderlichen Dokumentation sowie den Suchvorgängen, die ein effektives Arbeiten ermöglichen, klafft eine Lücke. Diese zu schließen, ist eine der nützlichsten Funktionen von IPython/Jupyter. Zwar spielen Suchvorgänge im Web bei der Beantwortung komplizierter Fragen nach wie vor eine Rolle, allerdings stellt IPython bereits eine bemerkenswerte Menge an Informationen bereit. Hier einige Beispiele für Fragen, bei deren Beantwortung IPython nach einigen wenigen Tastendrücken hilfreich sein kann:

- Wie rufe ich eine bestimmte Funktion auf? Welche Argumente und Optionen besitzt sie?
- Wie sieht der Quellcode eines bestimmten Python-Objekts aus?
- Was ist in einem importierten Paket enthalten?
- Welche Attribute oder Methoden besitzt ein Objekt?

Wir erörtern nun die Tools von IPython und Jupyter für den schnellen Zugriff auf diese Informationen, nämlich das Zeichen `?` zum Durchsuchen der Dokumentation, die beiden Zeichen `??` zum Erkunden des Quellcodes und die Tabulatortaste (Tab-Taste), die eine automatische Vervollständigung ermöglicht.

Mit `?` auf die Dokumentation zugreifen

Die Programmiersprache Python und das für die Data Science geeignete Ökosystem schenken den Nutzerinnen und Nutzern große Beachtung. Dazu gehört insbesondere der Zugang zur Dokumentation. Alle Python-Objekte enthalten einen Verweis auf einen String, den sogenannten *Docstring*, der wiederum in den meisten Fällen eine kompakte Übersicht über das Objekt und dessen Verwendung enthält. Python verfügt über eine integrierte `help`-Funktion, die auf diese Informationen zugreift und sie ausgibt. Um beispielsweise die Dokumentation der integrierten Funktion `len` anzuzeigen, können Sie Folgendes eingeben:

```
In [1]: help(len)
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

Je nachdem, welchen Interpreter Sie verwenden, wird der Text auf der Konsole oder in einem eigenen Fenster ausgegeben.

Die Suche nach der Hilfe für ein Objekt ist äußerst nützlich und sehr häufig notwendig. Daher verwendet IPython das Zeichen `?` als Abkürzung für den Zugriff auf die Dokumentation und weitere wichtige Informationen:

```
In [2]: len?
Signature: len(obj, /)
Docstring: Return the number of items in a container.
Type:      builtin_function_or_method
```

Diese Schreibweise funktioniert praktisch mit allem, auch mit Objektmethoden:

```
In [3]: L = [1, 2, 3]
In [4]: L.insert?
Signature: L.insert(index, object, /)
Docstring: Insert object before index.
Type:      builtin_function_or_method
```

Und sogar mit Objekten selbst – dann wird die Dokumentation des Objekttyps angezeigt:

```
In [5]: L?
Type:      list
String form: [1, 2, 3]
Length:    3
Docstring:
Built-in mutable sequence.
```

If no argument is given, the constructor creates a new empty list.
The argument must be an iterable if specified.

Wichtig zu wissen ist, dass das ebenfalls mit Funktionen und anderen von Ihnen selbst erzeugten Objekten funktioniert:

```
In [6]: def square(a):
....:     """Return the square of a."""
....:     return a ** 2
....:
```

Beachten Sie hier, dass wir zum Erstellen des Docstrings unserer Funktion einfach eine literale Zeichenkette in die erste Zeile eingegeben haben. Da Docstrings für gewöhnlich mehrzeilig sind, haben wir gemäß Konvention Pythons Schreibweise für mehrzeilige Strings mit dreifachem Anführungszeichen verwendet.

Nun verwenden wir das Zeichen `?`, um diesen Docstring anzuzeigen:

```
In [7]: square?
Signature: square(a)
Docstring: Return the square of a.
File:    <ipython-input-6>
Type:    function
```

Dieser schnelle Zugriff auf die Dokumentation via Docstring ist einer der Gründe dafür, dass Sie sich angewöhnen sollten, den von Ihnen geschriebenen Code immer zu dokumentieren!

Mit `??` auf den Quellcode zugreifen

Da die Programmiersprache Python sehr leicht verständlich ist, können Sie für gewöhnlich tiefere Einblicke gewinnen, wenn Sie sich den Quellcode eines Objekts an-

sehen, das Sie interessiert. Mit einem doppelten Fragezeichen (??) stellen IPython und Jupyter eine Abkürzung für den Zugriff auf den Quellcode zur Verfügung:

```
In [8]: square??  
Signature: square(a)  
Source:  
def square(a):  
    """Return the square of a."""  
    return a ** 2  
File:      <ipython-input-6>  
Type:     function
```

Bei so einfachen Funktionen wie dieser können Sie mit dem doppelten Fragezeichen einen schnellen Blick darauf werfen, was unter der Haube vor sich geht.

Wenn Sie damit weiter herumexperimentieren, werden Sie feststellen, dass ein angehängtes ?? manchmal gar keinen Quellcode anzeigt. Das liegt im Allgemeinen daran, dass das fragliche Objekt nicht in Python implementiert ist, sondern in C oder einer anderen kompilierten Erweiterungssprache. In diesem Fall liefert ?? dieselbe Ausgabe wie ?. Das kommt insbesondere bei vielen in Python fest integrierten Objekten und Typen vor, wie beispielsweise bei der vorhin erwähnten Funktion len:

```
In [9]: len??  
Signature: len(obj, /)  
Docstring: Return the number of items in a container.  
Type:     builtin_function_or_method
```

Der Einsatz von ? und/oder ?? bietet eine schnelle und leistungsfähige Möglichkeit für das Auffinden von Informationen darüber, was in einer Python-Funktion oder einem Python-Modul eigentlich geschieht.

Module mit der Tab-Vervollständigung erkunden

Eine weitere nützliche Schnittstelle ist die Verwendung der Tabulatortaste zur automatischen Vervollständigung und zum Erkunden des Inhalts von Objekten, Modulen und Namensräumen. In den folgenden Beispielen wird durch <TAB> angezeigt, dass die Tabulatortaste gedrückt werden muss.

Tab-Vervollständigung des Inhalts von Objekten

Jedes Python-Objekt besitzt verschiedene Attribute und Methoden, die ihm zugeordnet sind. Neben dem bereits erläuterten help verfügt Python über eine integrierte dir-Funktion, die eine Liste dieser Attribute und Methoden ausgibt. Allerdings ist es in der Praxis viel einfacher, die Tab-Vervollständigung zu verwenden. Um eine Liste aller verfügbaren Attribute anzuzeigen, geben Sie einfach den Namen des Objekts ein, gefolgt von einem Punkt (.) und einem Druck auf die Tab-Taste:

```
In [10]: L.<TAB>  
append() count    insert  reverse  
clear    extend    pop     sort  
copy     index    remove
```

Um die Anzahl der Treffer in der Liste zu verringern, geben Sie einfach den ersten oder mehrere Buchstaben des Namens ein. Nach dem Drücken der Tab-Taste werden dann nur noch die übereinstimmenden Attribute und Methoden angezeigt:

```
In [10]: L.c<TAB>
         clear() count()
         copy()
```

```
In [10]: L.co<TAB>
         copy() count()
```

Wenn der Treffer eindeutig ist, wird die Zeile durch ein weiteres Drücken der Tabulatortaste vervollständigt. Die folgende Eingabe wird beispielsweise sofort zu `L.count` vervollständigt:

```
In [10]: L.cou<TAB>
```

Python erzwingt zwar keine strenge Unterscheidung zwischen öffentlichen/externen und privaten/internen Attributen, allerdings gibt es die Konvention, Letztere durch einen vorangestellten Unterstrich zu kennzeichnen. Der Einfachheit halber werden die privaten und besonderen Methoden in der Liste standardmäßig weggelassen. Es ist jedoch möglich, sie durch ausdrückliche Eingabe des Unterstrichs anzuzeigen:

```
In [10]: L._<TAB>
         _add_          _delattr_      _eq_
         _class_        _delitem_     _format_()
         _class_getitem_() _dir_()   _ge_
         _contains_     _doc_     _getattr_>
```

Wir zeigen hier nur kurz die ersten paar Zeilen der Ausgabe. Bei den meisten handelt es sich um Pythons spezielle Methoden, deren Namen mit einem doppelten Unterstrich beginnen (engl. *double underscore*, daher oft auch »dunder«-Methoden genannt).

Tab-Vervollständigung beim Importieren

Die Tab-Vervollständigung kann außerdem beim Importieren von Objekten aus Paketen helfen. Hier benutzen wir sie, um alle möglichen Importe aus dem Paket `iter tools` zu finden, die mit der Zeichenkette `co` beginnen:

```
In [10]: from itertools import co<TAB>
         combinations() compress()
         combinations_with_replacement() count()
```

Auf ähnliche Weise können Sie die Tab-Vervollständigung einsetzen, um zu sehen, welche Importe auf Ihrem System verfügbar sind (das kann sich ändern, je nachdem, welche Skripte oder Module von Drittherstellern für Ihre Python-Sitzung sichtbar sind):

```
In [10]: import <TAB>
         abc          anyio
         activate_this appdirs
         aifc         appnope
         antigravity  argon2 >
```

```
In [10]: import h<TAB>
         hashlib html
         heapq http
         hmac
```

Mehr als die Tab-Vervollständigung: Suche mit Wildcards

Die Tab-Vervollständigung ist nützlich, wenn Ihnen die ersten paar Buchstaben des Namens eines Objekts oder Attributs bekannt sind, das Sie suchen, hilft aber nicht weiter, wenn Sie nach übereinstimmenden Zeichen in der Mitte oder am Ende einer Bezeichnung suchen. Für diesen Anwendungsfall halten IPython und Jupyter mit dem Zeichen `*` eine Suche mithilfe von Wildcards bereit.

Wir können das Zeichen beispielsweise verwenden, um alle Objekte im Namensraum anzuzeigen, deren Namen auf `Warning` enden:

```
In [10]: *Warning?
BytesWarning          RuntimeWarning
DeprecationWarning   SyntaxWarning
FutureWarning        UnicodeWarning
ImportWarning         UserWarning
PendingDeprecationWarning Warning
ResourceWarning
```

Beachten Sie, dass das Zeichen `*` mit allen Strings übereinstimmt – auch mit einer leeren Zeichenkette.

Nehmen wir nun an, wir suchten nach einer String-Methode, die an irgendeiner Stelle in ihrem Namen das Wort `find` enthält. Auf diese Weise können wir sie finden:

```
In [11]: str.*find*?
str.find
str.rfind
```

Ich finde diese Art der flexiblen Suche mit Wildcards äußerst nützlich, um einen bestimmten Befehl zu finden, wenn ich ein neues Paket erkunde oder mich mit einem bereits bekannten erneut vertraut mache.

Tastaturkürzel in der IPython-Shell

Auch wenn Sie nur wenig Zeit mit einem Computer verbringen, werden Sie vermutlich bereits festgestellt haben, dass sich das eine oder andere Tastaturkürzel für Ihren Arbeitsablauf als nützlich erweist. Am bekanntesten sind vielleicht `Strg-C` und `Strg-V` (bzw. `Cmd-C` und `Cmd-V`) zum Kopieren und Einfügen in vielen ganz verschiedenen Programmen und Systemen. Erfahrene User gehen oft sogar noch einen Schritt weiter: Gängige Texteditoren wie Emacs, Vim und andere stellen einen immensen Umfang an verschiedenen Funktionen durch komplizierte Kombinationen von Tastendrücken zur Verfügung.

Ganz so weit geht die IPython-Shell nicht, dennoch bietet sie einige Tastaturkürzel zum schnellen Navigieren beim Eingeben von Befehlen. Einige der Tastaturkürzel

funktionieren eventuell auch in browserbasierten Notebooks, allerdings geht es in diesem Abschnitt vornehmlich um die Tastaturkürzel in der IPython-Shell.

Sobald Sie sich daran gewöhnt haben, sind sie äußerst nützlich, um schnell bestimmte Befehle auszuführen, ohne die Hände von der Tastatur nehmen zu müssen. Sollten Sie Emacs-User sein oder Erfahrung mit Linux-Shells haben, wird Ihnen das Folgende bekannt vorkommen. Ich gruppiere die Befehle nach bestimmten Kategorien: Tastaturkürzel zum *Navigieren*, Tastaturkürzel bei der *Texteingabe*, Tastaturkürzel für den *Befehlsverlauf* und *sonstige* Tastaturkürzel.

Tastaturkürzel zum Navigieren

Dass die nach links und rechts weisenden Pfeiltasten dazu dienen, den Cursor in der Zeile rückwärts bzw. vorwärts zu bewegen, ist ziemlich offensichtlich, es gibt jedoch weitere Möglichkeiten, die es nicht erforderlich machen, Ihre Hände aus der gewohnten Schreibposition zu bewegen.

Tastaturkürzel	Beschreibung
Strg-A	Bewegt den Cursor an den Zeilenanfang.
Strg-E	Bewegt den Cursor an das Zeilenende.
Strg-B (oder Pfeiltaste nach links)	Bewegt den Cursor ein Zeichen rückwärts.
Strg-F (oder Pfeiltaste nach rechts)	Bewegt den Cursor ein Zeichen vorwärts.

Tastaturkürzel bei der Texteingabe

Jeder Nutzerin und jedem Nutzer ist die Verwendung der Rückschritttaste zum Löschen des zuvor eingegebenen Zeichens bekannt, allerdings sind oftmals einige Fingerverrenkungen erforderlich, um sie zu erreichen, und außerdem löscht sie beim Drücken jeweils nur ein einzelnes Zeichen. In IPython gibt es einige Tastaturkürzel zum Löschen bestimmter Teile der eingegebenen Textzeile. Sofort nützlich sind die Befehle zum Löschen der ganzen Textzeile. Sie sind Ihnen in Fleisch und Blut übergegangen, wenn Sie feststellen, dass Sie die Tastenkombinationen Strg-B und Strg-D verwenden, anstatt die Rückschritttaste zu benutzen, um das zuvor eingegebene Zeichen zu löschen!

Tastaturkürzel	Beschreibung
Rückschritttaste	Zeichen links vom Cursor löschen.
Strg-D	Zeichen rechts vom Cursor löschen.
Strg-K	Text von der Cursorposition bis zum Zeilenende ausschneiden.
Strg-U	Text vom Zeilenanfang bis zur Cursorposition ausschneiden.
Strg-Y	Zuvor ausgeschnittenen Text einfügen.
Strg-T	Die beiden zuletzt eingegebenen Zeichen vertauschen.

Tastaturkürzel für den Befehlsverlauf

Unter den hier aufgeführten von IPython bereitgestellten Tastaturkürzeln dürften diejenigen zur Navigation im Befehlsverlauf die größten Auswirkungen haben. Der Befehlsverlauf umfasst nicht nur die aktuelle IPython-Sitzung, Ihr gesamter Befehlsverlauf ist in einer SQLite-Datenbank im selben Verzeichnis gespeichert, in dem sich auch Ihr IPython-Profil befindet.

Die einfachste Zugriffsmöglichkeit auf Ihren Befehlsverlauf ist das Betätigen der Pfeiltasten nach oben und unten, mit denen Sie ihn schrittweise durchblättern können, es stehen aber noch andere Möglichkeiten zur Verfügung.

Tastaturkürzel	Beschreibung
Strg-P (oder Pfeiltaste nach oben)	Vorhergehenden Befehl im Verlauf auswählen.
Strg-N (oder Pfeiltaste nach unten)	Nachfolgenden Befehl im Verlauf auswählen.
Strg-R	Rückwärtssuche im Befehlsverlauf.

Die Rückwärtssuche kann besonders praktisch sein. Wie Sie wissen, haben wir im vorherigen Abschnitt eine Funktion namens `square` definiert. Durchsuchen Sie nun in einer neuen IPython-Shell den Befehlsverlauf nach dieser Definition. Wenn Sie im IPython-Terminal die Tastenkombination Strg-R drücken, wird Ihnen die folgende Eingabeaufforderung angezeigt:

```
In [1]:  
(reverse-i-search)`':
```

Beginnen Sie nun damit, Zeichen einzugeben, zeigt IPython den zuletzt eingegebenen Befehl an (sofern vorhanden), der mit den eingegebenen Zeichen übereinstimmt:

```
In [1]:  
(reverse-i-search)`squ': square??
```

Sie können jederzeit weitere Zeichen eingeben, um die Suche zu verfeinern, oder drücken Sie erneut Strg-R, um nach einem weiter zurückliegenden Befehl zu suchen, der zur Suchanfrage passt. Wenn Sie die Eingaben im letzten Abschnitt nachvollzogen haben, wird nach zweimaligem Betätigen von Strg-R Folgendes angezeigt:

```
In [1]:  
(reverse-i-search)`squ': def square(a):  
    """Return the square of a"""  
    return a ** 2
```

Sobald Sie den gesuchten Befehl gefunden haben, beenden Sie die Suche mit der Enter-Taste. Jetzt können Sie den gefundenen Befehl ausführen und die Sitzung fortsetzen:

```
In [1]: def square(a):  
    """Return the square of a"""  
    return a ** 2
```

```
In [2]: square(2)  
Out[2]: 4
```

Sie können auch die Tastenkombinationen Strg-P/Strg-N oder die Pfeiltasten nach oben und unten verwenden, um den Befehlsverlauf zu durchsuchen, allerdings werden dann bei der Suche lediglich die Zeichen am Anfang der Eingabezeile berücksichtigt. Wenn Sie also **def** eingeben und dann Strg-P drücken, wird der zuletzt eingegebene Befehl im Verlauf angezeigt (falls vorhanden), der mit den Zeichen **def** beginnt.

Sonstige Tastaturkürzel

Darüber hinaus gibt es einige weitere nützliche Tastaturkürzel, die sich keiner der bisherigen Kategorien zuordnen lassen.

Tastaturkürzel	Beschreibung
Strg-L	Terminalanzeige löschen.
Strg-C	Aktuellen Python-Befehl abbrechen.
Strg-D	Python-Sitzung beenden.

Insbesondere der Befehl Strg-C kann sich als nützlich erweisen, wenn Sie versehentlich einen sehr zeitaufwendigen Job gestartet haben.

Einige der hier vorgestellten Befehle mögen auf den ersten Blick vielleicht uninteressant erscheinen, Sie werden sie aber mit etwas Übung wie im Schlaf verwenden. Haben Sie sich diese Fingerfertigkeiten einmal angeeignet, werden Sie sich sogar wünschen, dass diese Befehle auch an anderer Stelle zur Verfügung stünden.