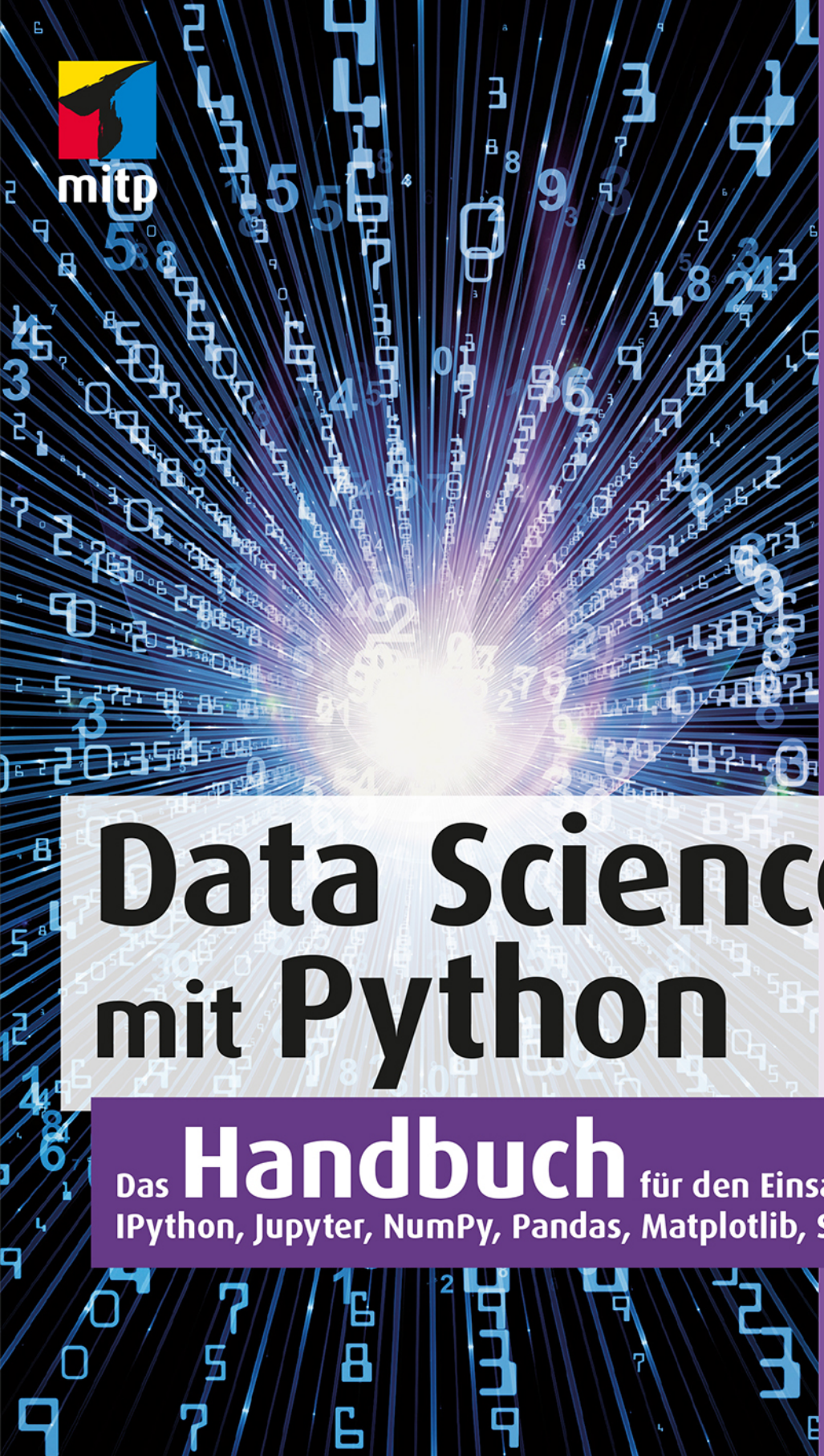

mitp

Jake VanderPlas

Data Science mit Python

Das Handbuch für den Einsatz von IPython, Jupyter, NumPy, Pandas, Matplotlib, Scikit-Learn

Jake VanderPlas

Data Science mit Python

Das Handbuch für den Einsatz von IPython, Jupyter, NumPy, Pandas, Matplotlib, Scikit-Learn

Übersetzung aus dem Englischen
von Knut Lorenzen

mitp

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 978-3-95845-696-9
1. Auflage 2018

www.mitp.de
E-Mail: mitp-verlag@sigloch.de
Telefon: +49 7953 / 7189 - 079
Telefax: +49 7953 / 7189 - 082

Authorized German translation of the English edition of
Python Data Science Handbook – Essential Tools for Working with Data
ISBN 978-1491912058 © 2017 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.

Lektorat: Sabine Schulz
Sprachkorrektorat: Sibylle Feldmann
Coverbild: © agsandrew / fotolia.com
Satz: III-satz, Husby, www.drei-satz.de

Inhaltsverzeichnis

Einleitung

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 dieser Begriff 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 übereifriger 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 industriellen und akademischen Anwendungsbereichen 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 0.1).

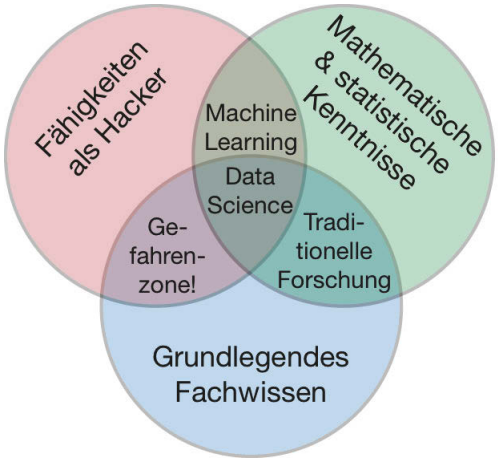

Abb. 0.1: Das Venn-Diagramm zur Data Science von Drew Conway

Zwar sind einige der Bezeichnungen für die Schnittmengen nicht ganz ernst gemeint, 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 Aufgaben

eines Statistikers, der (immer größer werdende) Datenmengen modellieren und zusammenfassen kann, die Arbeit des Informatikers, der Algorithmen für die effiziente Speicherung, Verarbeitung und Visualisierung dieser Daten entwerfen kann, 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.

Das habe ich im Sinn, wenn ich Sie dazu auffordere, 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, Mausklicks 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.

An wen richtet sich das 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, Entwickler 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 der 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 IPython, NumPy, Pandas, Matplotlib, Scikit-Learn und ähnliche Tools –, um Daten effektiv zu speichern, zu handhaben und Einblick in diese Daten zu gewinnen.

Warum Python?

Python hat sich in den letzten 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 gekennzeichneter 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 später im Buch noch Erwähnung finden.

Falls Sie auf der Suche nach einer Einführung in die Programmiersprache Python sind, empfehle ich das dieses Buch ergänzende Projekt *A Whirlwind Tour of the Python Language* (`https://github.com/jakevdp/WhirlwindTourOfPython`). Hierbei 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.

Python 2 kontra Python 3

In diesem Buch wird die Syntax von Python 3 verwendet, die Spracherweiterungen enthält, die mit Python 2 inkompatibel sind. Zwar wurde Python 3 schon 2008 veröffentlicht, allerdings verbreitete sich diese Version insbesondere in den Communitys von Wissenschaft und Webentwicklung nur langsam. Das lag vor allem daran, dass die Anpassung vieler wichtiger Pakete von Drittherstellern an die neue Sprachversion Zeit benötigte. Seit Anfang 2014 gibt es jedoch stabile Versionen der für die Data Science wichtigsten Tools, die sowohl mit Python 2 als auch mit Python 3 kompatibel sind, daher wird in diesem Buch die neuere Syntax von Python 3 genutzt. Allerdings funktionieren die meisten Codeabschnitte dieses Buchs ohne Änderungen auch in Python 2. Wenn Py2-inkompatible Syntax verwendet wird, weise ich ausdrücklich darauf hin.

Inhaltsübersicht

Alle Kapitel in diesem Buch konzentrieren sich auf ein bestimmtes Paket oder Tool, das für die mit Python betriebene Data Science von grundlegender Bedeutung ist.

IPython und Jupyter (Kapitel 1)
: Diese Pakete bieten eine Umgebung für Berechnungen, die von vielen Data Scientists genutzt wird, die Python einsetzen.

NumPy (Kapitel 2)
: Diese Bibliothek stellt das `ndarray`-Objekt zur Verfügung, das ein effizientes Speichern und die Handhabung dicht gepackter Datenarrays in Python ermöglicht.

Pandas (Kapitel 3)
: Diese Bibliothek verfügt über das `DataFrame`-Objekt, das ein effizientes Speichern und die Handhabung gekennzeichneter bzw. spaltenorientierter Daten in Python gestattet.

Matplotlib (Kapitel 4)
: Diese Bibliothek ermöglicht flexible und vielfältige Visualisierungen von Daten in Python.

Scikit-Learn (Kapitel 5)
: Diese Bibliothek stellt eine effiziente Implementierung der wichtigsten und gebräuchlichsten Machine-Learning-Algorithmen zur Verfügung.

Natürlich umfasst die PyData-Welt viel mehr als diese fünf 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 fünf genannten Pakete 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.

Verwendung der Codebeispiele

Unter `https://github.com/jakevdp/PythonDataScienceHandbook` steht ergänzendes Material (Codebeispiele, Abbildungen usw.) zum Herunterladen zur Verfügung. Das 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 einer CD-ROM, die Codebeispiele aus dem Buch enthält, 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 *»Data Science mit Python« von Jake VanderPlas (mitp Verlag 2017), ISBN 978-3-95845-695-2*.

Installation der Software

Die Installation von Python und der 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 (`http://conda.pydata.org/miniconda.html`) 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://www.continuum.io/downloads`) 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 ipython-notebook
```

Wir werden im gesamten Buch noch weitere, spezialisiertere Tools verwenden, die zum wissenschaftlich orientierten Ökosystem in Python gehören. Für gewöhnlich ist zur Installation lediglich eine Eingabe wie `conda install paketname` erforderlich. Weitere Informationen über conda, beispielsweise über das Erstellen und Verwenden von conda-Umgebungen (die ich nur nachdrücklich empfehlen kann), finden Sie in der Onlinedokumentation (`http://conda.pydata.org/docs/`).

Konventionen dieses Buchs

In diesem Buch gelten die folgenden typografischen Konventionen:

Kursive Schrift
: Kennzeichnet neue Begriffe, Dateinamen und Dateinamenserweiterungen.

`Nicht proportionale Schrift`
: Wird für URLs, 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.

Die Webseite zum Buch

Der Verlag hält auf seiner Website weiteres Material zum Buch bereit. Unter `http://www.mitp.de/695` können Sie sich Beispielcode herunterladen.

Über den Autor

Jake VanderPlas ist seit Langem User und Entwickler der SciPy-Umgebung. Derzeit ist er als interdisziplinärer Forschungsdirektor an der Universität Washington tätig, führt eigene astronomische Forschungsarbeiten durch und berät dort ansässige Wissenschaftler, die auf vielen verschiedenen Fachgebieten arbeiten.

Kapitel 1

Mehr als normales Python: IPython

Für Python stehen viele verschiedene Entwicklungsumgebungen zur Verfügung, und häufig werde ich gefragt, welche ich für meine eigenen Arbeiten verwende. Einige Leute überrascht die Antwort: Meine bevorzugte Entwicklungsumgebung ist IPython (`http://ipython.org/`) in Kombination mit einem Texteditor (entweder Emacs oder Atom – das hängt von meiner Stimmung ab). IPython (Abkürzung für *Interactive Python*) wurde 2001 von Fernando Perez in Form eines erweiterten Python-Interpreters ins Leben gerufen und hat sich seither zu einem Projekt entwickelt, das es sich zum Ziel gesetzt 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 IPython als die interaktive Steuerkonsole dazu vorstellen.

IPython 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 gleich erörtern. IPython ist außerdem sehr eng mit dem Jupyter-Projekt verknüpft (`http://jupyter.org`), das ein browserbasiertes sogenanntes Notebook zur Verfügung stellt, das bei der Entwicklung, der Zusammenarbeit, dem Teilen und sogar der Veröffentlichung von Ergebnissen der Data Science gute Dienste leistet. Tatsächlich ist das IPython-Notebook eigentlich ein Sonderfall der umfangreicheren Jupyter-Notebook-Struktur, die Notebooks für Julia, R und andere Programmiersprachen umfasst. 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 IPython-Notebooks verfasst.

Bei IPython geht es darum, Python effizient für wissenschaftliche und datenintensive Berechnungen interaktiv einsetzen zu können. In diesem Kapitel werden wir zunächst einige der Features von IPython 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.

1.1 Shell oder Notebook?

Es gibt im Wesentlichen zwei verschiedene Methoden, IPython zu verwenden, die wir in diesem Kapitel betrachten: die IPython-Shell und das IPython-Notebook. Ein Großteil des Inhalts dieses Kapitels betrifft beide, und die Beispiele verwenden im Wechsel Shell und Notebook – je nachdem, was am praktischsten ist. In den Abschnitten, die lediglich für eines der beiden Verfahren von Bedeutung sind, werde ich ausdrücklich darauf hinweisen.

Doch zunächst einmal folgen einige Hinweise zum Starten der IPython-Shell und zum Öffnen eines Notebooks.

1.1.1 Die IPython-Shell starten

Wie die meisten Teile dieses Buchs sollte dieses Kapitel 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. (Wir erörtern das ausführlicher in Abschnitt 1.2, »Hilfe und Dokumentation in IPython«.)

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

```
IPython 4.0.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra
             details.
In [1]:
```

Nun können Sie fortfahren.

1.1.2 Das Jupyter-Notebook starten

Das Jupyter-Notebook ist eine browserbasierte grafische Schnittstelle für die Python-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 es anderen Usern ermöglicht, sie auf ihren eigenen Systemen zu öffnen und den Code auszuführen.

Das IPython-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 notebook
```

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 notebook
[NotebookApp] Serving notebooks from local directory: /Users/jakevdp/...
[NotebookApp] 0 active kernels
[NotebookApp] The IPython Notebook is running at: http://localhost:8888/
[NotebookApp] Use Control-C to stop this server and shut down all kernels...
```

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/`) eingeben.

1.2 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 meinem alltäglichen Arbeitsablauf haben.

Wenn ein technologisch interessierter Mensch darum gebeten wird, einem Freund, Familienmitglied oder Kollegen 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 genauso: Durchsuchbare Webressourcen wie Onlinedokumentationen, Mailinglisten und auf Stackoverflowbusiness.com gefundene Antworten enthalten jede Menge Informationen, auch (und 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 IPython-Tools für den schnellen Zugriff auf diese Informationen, nämlich das Zeichen `?` zum Durchsuchen der Dokumentation, die beiden Zeichen `??` zum Erkunden des Quellcodes und die Tab-Taste, die eine automatische Vervollständigung ermöglicht.

1.2.1 Mit ? auf die Dokumentation zugreifen

Die Programmiersprache Python und das für die Data Science geeignete Ökosystem schenken dem User 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(...)
    len(object) -> integer
    Return the number of items of a sequence or mapping.
```

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 geschieht sehr häufig. Daher verwendet IPython das Zeichen `?` als Abkürzung für den Zugriff auf die Dokumentation und weitere wichtige Informationen:

```
In [2]: len?
Type:        builtin_function_or_method
String form: <built-in function len>
Namespace:   Python builtin
Docstring:
len(object) -> integer
Return the number of items of a sequence or mapping.
```

Diese Schreibweise funktioniert praktisch mit allem, auch mit Objektmethoden:

```
In [3]: L = [1, 2, 3]
In [4]: L.insert?
Type:        builtin_function_or_method
String form: <built-in method insert of list object at 0x1024b8ea8>
Docstring:   L.insert(index, object) -- insert object before index
```

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:
```

```
list() -> new empty list
list(iterable) -> new list initialized from iterable's items
```

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

```
In [6]: def square(a):
  ....:     """a zum Quadrat zurückgeben."""
  ....:     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?
Type:        function
String form: <function square at 0x103713cb0>
Definition:  square(a)
Docstring:   a zum Quadrat zurückgeben.
```

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!

1.2.2 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 ansehen, das Sie interessiert. Mit einem doppelten Fragezeichen (??) stellt IPython eine Abkürzung für den Zugriff auf den Quellcode zur Verfügung:

```
In [8]: square??
Type:        function
String form: <function square at 0x103713cb0>
Definition:  square(a)
Source:
def square(a):
    "a zum Quadrat zurückgeben."
    return a ** 2
```

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

Sollten Sie damit weiter herumexperimentieren, werden Sie feststellen, dass ein angehängtes ?? manchmal überhaupt 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??
Type:        builtin_function_or_method
String form: <built-in function len>
Namespace:   Python builtin
Docstring:
len(object) -> integer
Return the number of items of a sequence or mapping.
```

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

1.2.3 Module mit der Tab-Vervollständigung erkunden

IPython besitzt eine weitere nützliche Schnittstelle: die Verwendung der Tab-Taste zur automatischen Vervollständigung und zum Erkunden des Inhalts von Objekten, Modulen und Namensräumen. In den folgenden Beispielen wird durch <TAB> angezeigt, dass die Tab-Taste 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 der Tab-Taste:

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

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

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

Wenn der Treffer eindeutig ist, wird die Zeile durch ein weiteres Drücken der Tab-Taste 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>
L.__add__           L.__gt__            L.__reduce__
L.__class__         L.__hash__          L.__reduce_ex__
```

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 (oft auch bezeichnet mit dem Spitznamen »dunder«-Methoden).

Tab-Vervollständigung beim Importieren

Auch beim Importieren von Objekten eines Pakets erweist sich die Tab-Vervollständigung als nützlich. Hier verwenden wir sie, um alle möglichen Importe des `itertools`-Pakets zu finden, deren Namen mit `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 prüfen, welche Importe für Ihr System verfügbar sind (das hängt davon ab, welche Skripte und Module von Drittherstellern für Ihre Python-Sitzung zugänglich sind):

```
In [10]: import <TAB>
Display all 399 possibilities? (y or n)
Crypto              dis                 py_compile
Cython              distutils           pyclbr
...                 ...                 ...

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

(Der Kürze halber sind hier wieder nicht alle 399 importierbaren Pakete und Module aufgeführt, die auf meinem System verfügbar sind.)

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 hält IPython mit dem Zeichen * eine Suche mit 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 Stringmethode, die an irgendeiner Stelle das Wort `find` enthält. Auf diese Weise können wir sie finden:

```
In [10]: 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.

1.3 Tastaturkürzel in der IPython-Shell

Auch wenn Sie nur wenig Zeit mit dem 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 Cmd-C und Cmd-V (bzw. Strg-C und Strg-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 dem User 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. Diese Tastaturkürzel stellt tatsächlich nicht IPython selbst zur Verfügung, sondern die von dem Programm genutzte GNU-Readline-Bibliothek. Aus diesem Grund unterscheiden sich einige Tastaturkürzel auf Ihrem Sys-

tem abhängig von der Konfiguration womöglich von den nachstehend aufgeführten. Einige der Tastaturkürzel funktionieren eventuell auch im browserbasierten Notebook, 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. Wir gruppieren die Befehle nach bestimmten Kategorien: Tastaturkürzel zum Navigieren, Tastaturkürzel bei der Texteingabe, Tastaturkürzel für den Befehlsverlauf und sonstige Tastaturkürzel.

1.3.1 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 (siehe Tabelle 1.1).

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

Tabelle 1.1: Tastaturkürzel zum Navigieren

1.3.2 Tastaturkürzel bei der Texteingabe

Jedem User 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 Betätigen 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.

Tabelle 1.2: Tastaturkürzel bei der Texteingabe

1.3.3 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 (siehe Tabelle 1.3).

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.

Tabelle 1.3: Tastaturkürzel für den Befehlsverlauf

Die Rückwärtssuche kann besonders praktisch sein. Wie Sie wissen, haben wir im vorigen 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):
    """a zum Quadrat zurückgeben."""
    return a ** 2
```

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

```
In [1]: def square(a):
    """a zum Quadrat zurückgeben."""
```

```
    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.

1.3.4 Sonstige Tastaturkürzel

Darüber hinaus gibt es einige weitere nützliche Tastaturkürzel, die sich keiner der bisherigen Kategorien zuordnen lassen (siehe Tabelle 1.4).

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

Tabelle 1.4: Sonstige Tastaturkürzel

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 benutzen. Haben Sie sich diese Fingerfertigkeiten einmal angeeignet, werden Sie sich sogar wünschen, dass diese Befehle auch an anderer Stelle zur Verfügung stünden.

1.4 Magische Befehle in IPython

Die beiden letzten Abschnitte zeigen, wie IPython es Ihnen ermöglicht, Python effektiv und interaktiv zu verwenden und zu erkunden. Nun kommen wir zu einigen Erweiterungen, die IPython der normalen Python-Syntax hinzufügt. Diese werden in IPython als »magische« Befehle oder Funktionen bezeichnet, und ihnen wird ein %-Zeichen vorangestellt. Die magischen Befehle sind dazu gedacht, verschiedene gängige Aufgaben, die bei einer Standarddatenanalyse immer wieder vorkommen, kurz und bündig zu erledigen. Von den magischen Befehlen/Funktionen (den sogenannten *Magics*) gibt es zwei Varianten: *Line-Magics*, denen ein einzelnes % vorangestellt wird und die jeweils eine einzelne Zeile verarbeiten, sowie *Cell-Magics*, die durch ein vorangestelltes %% gekennzeichnet sind und mehrzeilige Eingaben verarbeiten. Wir werden einige kurze Beispiele betrachten und befassen uns dann später in diesem Kapitel mit einer eingehenderen Erläuterung verschiedener nützlicher Magics.

1.4.1 Einfügen von Codeblöcken mit %paste und %cpaste

Beim Einsatz des IPython-Interpreters gibt es häufig das Problem, dass es beim Einfügen mehrzeiliger Codeblöcke zu unerwarteten Fehlern kommt, vor allem wenn der Text Einrü-

ckungen und vom Interpreter als Markierungen verwendete Zeichen enthält. Häufig ist das der Fall, wenn man auf einer Website Beispielcode entdeckt, den man im Interpreter einfügen möchte. Betrachten Sie die folgende einfache Funktion:

```
>>> def donothing(x):
...     return x
```

Der Code ist so formatiert, wie er im Python-Interpreter angezeigt werden soll. Wenn Sie ihn jedoch kopieren und direkt in IPython einfügen, erscheint eine Fehlermeldung:

```
In [2]: >>> def donothing(x):
   ...:     ...     return x
   ...:
  File "<ipython-input-20-5a66c8964687>", line 2
    ...     return x
                 ^
SyntaxError: invalid syntax
```

Beim direkten Einfügen gerät der Interpreter durch die zusätzlich vorhandenen Zeichen zur Eingabeaufforderung durcheinander. Aber keine Sorge – IPythons magische Funktion `%paste` ist dafür ausgelegt, genau diesen Typ mehrzeiliger mit Textauszeichnungen versehener Eingaben korrekt zu handhaben:

```
In [3]: %paste
>>> def donothing(x):
...     return x

## -- End pasted text --
```

Der `%paste`-Befehl fügt den Code ein und führt ihn aus, die Funktion kann nun also verwendet werden:

```
In [4]: donothing(10)
Out[4]: 10
```

Der Befehl `%cpaste` hat einen ganz ähnlichen Zweck und zeigt eine interaktive mehrzeilige Eingabeaufforderung an, in der Sie einen oder mehrere Codeschnipsel einfügen und der Reihe nach ausführen lassen können:

```
In [5]: %cpaste
Pasting code; enter '--' alone on the line to stop or use Ctrl-D.
:>>> def donothing(x):
:...     return x
:--
```

Diese magischen Befehle – und andere, auf die wir später noch zu sprechen kommen – stellen eine Funktionalität bereit, die mit einem herkömmlichen Python-Interpreter nur sehr schwer zu erzielen oder sogar unmöglich wäre.