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

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, Jørgen 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

Wieblinger Weg 17
69123 Heidelberg

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: *kommentar@oreilly.de*.

5 4 3 2 1 0

Inhalt

Teil III: Datenbearbeitung mit Pandas

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 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 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 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, 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.

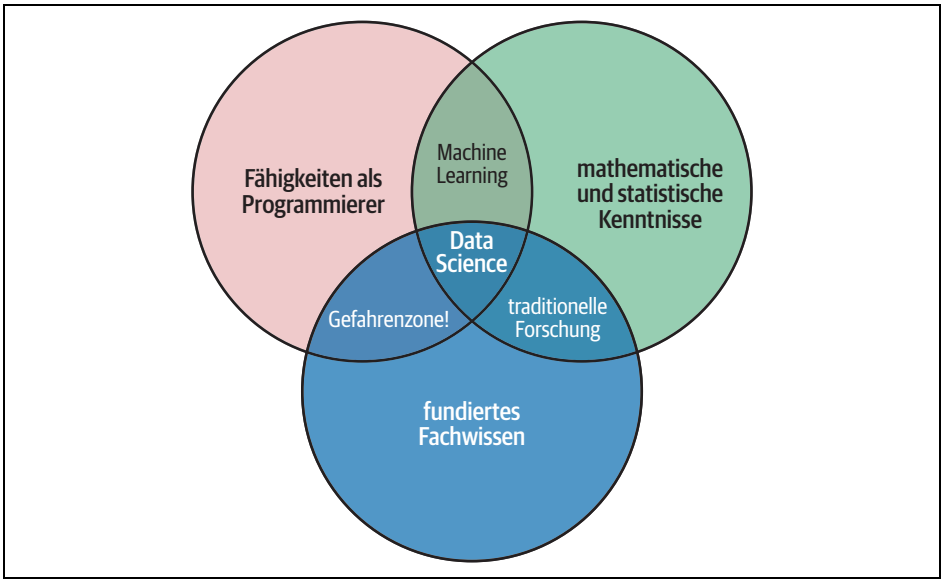

Abbildung 1: Das Venn-Diagramm zur Data Science von Drew Conway (Quelle: Drew Conway (https://oreil.ly/PkOOw), Abdruck mit freundlicher Genehmigung)

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 *kommentar@oreilly.de*.

TEIL I

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.

KAPITEL 1

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__         __getattribute__
```

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 `itertools` 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.