O'REILLY®

Deutsche Ausgabe

Python für Excel

Eine moderne Umgebung für Automatisierung und Datenanalyse

Felix Zumstein

Übersetzung von Frank Langenau

Python für Excel

Eine moderne Umgebung für Automatisierung und Datenanalyse

Felix Zumstein

Deutsche Übersetzung von Frank Langenau

O'REILLY®

Felix Zumstein

Lektorat: Alexandra Follenius
Übersetzung: Frank Langenau
Korrektorat: Sibylle Feldmann, *www.richtiger-text.de*
Satz: III-Satz, *www.drei-satz.de*
Herstellung: Stefanie Weidner
Umschlaggestaltung: Karen Montgomery, Michael Oréal, *www.oreal.de*

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 978-3-96009-197-4
PDF 978-3-96010-716-3
ePub 978-3-96010-717-0
mobi 978-3-96010-718-7

1. Auflage 2022
1., korrigierter Nachddruck 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.

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

Inhalt

Vorwort

Microsoft unterhält ein Feedback-Forum für Excel auf UserVoice[1], in dem jeder die Möglichkeit hat, neue Ideen einzureichen, über die andere abstimmen können. Die meistgenannte Funktionsanforderung, »Python als Excel-Skriptsprache«, hat ungefähr doppelt so viele Stimmen wie die zweithäufigste. Obwohl seit der Aufnahme der Idee im Jahr 2015 nicht wirklich etwas passiert ist, witterten Excel-Benutzer Ende 2020 Morgenluft, als Guido van Rossum, der Schöpfer von Python, twitterte (https://oreil.ly/N1_7N), dass sein »Ruhestand langweilig« sei und er zu Microsoft wechseln werde. Ob sein Wechsel einen Einfluss auf das Zusammenspiel von Excel und Python hat, weiß ich nicht. Aber ich weiß, was diese Kombination so überzeugend macht und wie Sie Excel und Python gemeinsam nutzen können – heute. Und das ist, kurz gesagt, das, worum es in diesem Buch geht.

Die treibende Kraft hinter der Geschichte von *Python für Excel* ist die Tatsache, dass wir in einer Welt voller Daten leben. Heutzutage sind riesige Datensätze für jeden und über alles verfügbar. Oftmals sind diese Datensätze so groß, dass sie nicht mehr in ein Tabellenblatt passen. Vor einigen Jahren hätte man dies vielleicht noch als *Big Data* bezeichnet, aber heutzutage ist ein Datensatz mit ein paar Millionen Zeilen wirklich nichts Besonderes mehr. Excel hat sich weiterentwickelt, um mit diesem Trend fertigzuwerden: Eingeführt wurde *Power Query*, um solche Datensätze zu laden und zu bereinigen, die nicht mehr in ein Tabellenblatt passen, und das Add-in *Power Pivot*, um Datenanalysen auf diesen Datensätzen durchzuführen und die Ergebnisse zu präsentieren. Power Query basiert auf der Formelsprache *Power Query M* (kurz M), während Power Pivot Formeln mithilfe von *Data Analysis Expressions* (DAX) definiert. Und wenn Sie einige Dinge in Ihrer Excel-Datei automatisieren möchten, verwenden Sie VBA (*Visual Basic for Applications*), die in Excel integrierte Automatisierungssprache. Für etwas ziemlich Einfaches haben Sie es letztlich mit VBA, M und DAX zu tun. Ein Problem dabei ist, dass Ihnen alle diese Sprachen nur in der Microsoft-Welt weiterhelfen – vor allem auf Excel und Power BI trifft dies zu. (Power BI werde ich kurz in Kapitel 1 vorstellen.)

1 *https://web.archive.org/web/20201209042635/https://excel.uservoice.com/forums/304921-excel-for-windows-desktop-application/filters/top*

Auf der anderen Seite ist Python eine Allzweckprogrammiersprache, die bei Analysten und Data Scientists zu einer der beliebtesten Sprache avanciert ist. Wenn Sie Python mit Excel verwenden, steht Ihnen eine Programmiersprache zur Verfügung, die für alle Aspekte der Geschichte geeignet ist, sei es, um Excel zu automatisieren, auf Datensätze zuzugreifen und sie vorzubereiten oder um Aufgaben der Datenanalyse und Visualisierung wahrzunehmen. Vor allem aber können Sie Ihre Python-Kenntnisse außerhalb von Excel wiederverwenden: Wenn Sie Ihre Rechenleistung erhöhen müssen, können Sie Ihr quantitatives Modell, Ihre Simulation oder Ihre Anwendung für maschinelles Lernen einfach in die Cloud verlagern, wo praktisch unbegrenzte Rechenressourcen auf Sie warten.

Warum ich dieses Buch geschrieben habe

Durch meine Arbeit an *xlwings*, dem Excel-Automatisierungspaket, das Sie in Teil IV dieses Buchs kennenlernen werden, stehe ich in engem Kontakt mit vielen Anwendern, die Python für Excel einsetzen – sei es über den *Issue Tracker* (*https://oreil.ly/ZJQkB*) auf GitHub, aufgrund einer Frage auf *StackOverflow* (*https://stackoverflow.com/*) oder bei einer Veranstaltung wie einem Treffen oder einer Konferenz.

Regelmäßig werde ich gebeten, Quellen für die ersten Schritte mit Python zu empfehlen. Obwohl es sicherlich keinen Mangel an Python-Einführungen gibt, sind sie oft entweder zu allgemein (ohne irgendwas über Datenanalyse) oder zu spezifisch (vollständig wissenschaftlich). Excel-Benutzer befinden sich jedoch eher in der Mitte: Sie arbeiten sicherlich mit Daten, aber eine vollständige wissenschaftliche Einführung mag ihnen zu technisch erscheinen. Außerdem haben sie oftmals spezielle Anforderungen und Fragen, die in den vorhandenen Materialien nicht beantwortet werden. Einige dieser Fragen lauten:

- Welches Python-Excel-Paket brauche ich für welche Aufgabe?
- Wie verschiebe ich meine Power-Query-Datenbankverbindung zu Python?
- Was ist in Python äquivalent zu den Features AutoFilter und Pivottabelle von Excel?

Ich habe dieses Buch geschrieben, damit Sie ohne vorherige Python-Kenntnisse in der Lage sind, Ihre Excel-orientierten Aufgaben zu automatisieren und die Python-Tools für Datenanalyse und wissenschaftliche Berechnungen ohne Umwege in Excel zu nutzen.

Für wen dieses Buch gedacht ist

Wenn Sie als fortgeschrittener Excel-Benutzer die Grenzen von Excel mit einer modernen Programmiersprache überwinden möchten, ist dieses Buch genau das richtige für Sie. Normalerweise bedeutet dies, dass Sie jeden Monat Stunden damit zubringen, große Datenmengen herunterzuladen, zu bereinigen, zu kopieren und in

unternehmenskritische Tabellen einzufügen. Es gibt zweifellos verschiedene Möglichkeiten, die Grenzen von Excel zu überwinden, doch dieses Buch konzentriert sich darauf, wie Sie für diese Aufgaben Python verwenden.

Es ist hilfreich, wenn Sie ein grundlegendes Verständnis von Programmierung mitbringen, denn wenn Sie schon einmal eine Funktion oder eine `for`-Schleife geschrieben haben (egal in welcher Programmiersprache), verfügen Sie schon über eine Vorstellung davon, was eine ganze Zahl oder ein String ist. Von diesem Buch können Sie möglicherweise auch profitieren, wenn Sie es gewohnt sind, komplexe Zellformeln zu schreiben, oder Erfahrung darin besitzen, aufgezeichnete VBA-Makros zu optimieren. Nicht erwartet werden Python-spezifische Kenntnisse, da Sie zu allen Tools, die wir hier verwenden, und zu Python selbst Einführungen bekommen.

Ein erfahrener VBA-Entwickler findet in diesem Buch regelmäßig Vergleiche zwischen Python und VBA, mit denen es möglich ist, die üblichen Probleme zu umschiffen und sofort loszulegen.

Dieses Buch kann auch hilfreich sein, wenn Sie als Python-Entwickler die verschiedenen Möglichkeiten kennenlernen müssen, die Python hat, um mit der Excel-Anwendung und den Excel-Dateien umgehen zu können, damit Sie je nach den Anforderungen Ihrer Geschäftskunden das richtige Paket auswählen können.

Wie dieses Buch aufgebaut ist

In diesem Buch zeige ich Ihnen alle Aspekte der Python-für-Excel-Geschichte, gegliedert in vier Teile:

Teil I: Einführung in Python
: Dieser Teil beleuchtet zunächst die Gründe dafür, dass Python ein so angenehmer Begleiter für Excel ist, und stellt dann die Tools vor, die wir in diesem Buch verwenden: die Anaconda-Python-Distribution, Visual Studio Code und Jupyter Notebooks. Außerdem erfahren Sie in diesem Teil genug über Python, um den Rest des Buchs zu meistern.

Teil II: Einführung in pandas
: Bei pandas handelt es sich um die Standardbibliothek von Python für die Datenanalyse. In diesem Teil lernen Sie, wie sich Excel-Arbeitsmappen durch eine Kombination aus Jupyter Notebooks und pandas ersetzen lassen. Normalerweise ist pandas-Code sowohl einfacher zu pflegen als auch effizienter als eine Excel-Arbeitsmappe. Und Sie können mit Datensätzen arbeiten, die für ein Excel-Tabellenblatt viel zu groß sind. Im Unterschied zu Excel können Sie mit pandas Ihren Code überall dort ausführen, wo Sie wollen, auch in der Cloud.

Teil III: Excel-Dateien ohne Excel lesen und schreiben

In diesem Teil geht es um die Bearbeitung von Excel-Dateien mit einem der folgenden Python-Pakete: pandas, OpenPyXL, XlsxWriter, pyxlsb, xlrd und xlwt. Diese Pakete sind in der Lage, Excel-Arbeitsmappen direkt vom Datenträger zu lesen und darauf zu schreiben, und ersetzen somit die Excel-Anwendung: Da Sie keine Installation von Excel benötigen, funktionieren sie auf jeder Plattform, die Python unterstützt, einschließlich Windows, macOS und Linux. So wendet man ein Reader-Paket typischerweise an, um Daten aus Excel-Dateien einzulesen, die man jeden Morgen von einem externen Unternehmen oder System erhält, und diese Daten in einer Datenbank zu speichern. Ein Writer-Paket liefert zum Beispiel die Funktionalität hinter der berühmten Schaltfläche *Nach Excel exportieren*, die man in fast jeder Anwendung findet.

Teil IV: Die Excel-Anwendung mit xlwings programmieren

In diesem Teil erfahren Sie, wie Sie Python mit dem Paket xlwings einsetzen können, um die Excel-Anwendung zu automatisieren, anstatt Excel-Dateien vom Datenträger zu lesen und darauf zu schreiben. Demzufolge ist für diesen Teil eine lokale Installation von Excel erforderlich. Sie lernen, wie Sie Excel-Arbeitsmappen öffnen und sie vor Ihren Augen bearbeiten. Neben dem Lesen und Schreiben von Dateien via Excel erstellen Sie interaktive Excel-Tools. Mit diesen ist es dann möglich, eine Schaltfläche anzuklicken, um Python etwas ausführen zu lassen, was Sie vielleicht schon mit VBA-Makros bewerkstelligt haben – beispielsweise eine umfangreiche Berechnung. Außerdem lernen Sie, wie Sie benutzerdefinierte Funktionen (*User-Defined Functions*, UDFs) in Python statt in VBA schreiben.

Es ist wichtig, den grundlegenden Unterschied zwischen Lesen und Schreiben von Excel-*Dateien* (Teil III) und dem Programmieren der Excel-*Anwendung* (Teil IV) zu verstehen, wie es Abbildung E-1 veranschaulicht.

Da Teil III keine Installation von Excel erfordert, funktioniert alles auf allen Plattformen, die Python unterstützen, namentlich Windows, macOS und Linux. Demgegenüber setzt Teil IV Plattformen voraus, die Microsoft Excel unterstützen, d. h. Windows und macOS, da sich der Code auf eine lokale Installation von Microsoft Excel stützt.

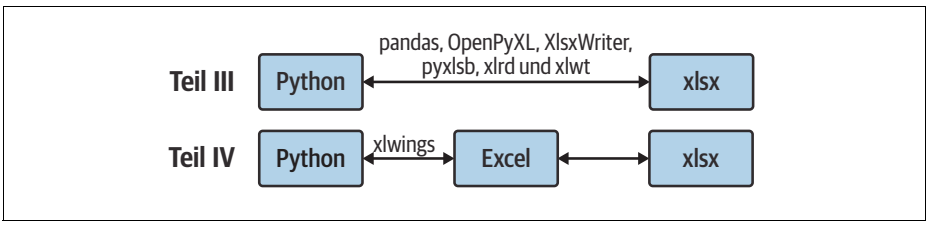

Abbildung E-1: Lesen und Schreiben von Excel-Dateien (Teil III) vs. Programmieren der Excel-Anwendung (Teil IV)

Python- und Excel-Versionen

Die Beispiele in diesem Buch sind mit der Anaconda-Python-Distribution 2020.11 (die Python 3.8 verwendet) und 2021.11 (die Python 3.9 verwendet) getestet worden. Wenn Sie eine neuere Version von Python verwenden möchten, beispielsweise Python 3.10, folgen Sie bitte den Anweisungen auf der Homepage des Buchs. Gelegentlich habe ich einen Kommentar hinterlassen, wenn sich etwas zwischen Python 3.8 und Python 3.9 geändert hat.

Es wird in diesem Buch davon ausgegangen, dass Sie eine moderne Version von Excel verwenden, d. h. mindestens Excel 2007 unter Windows und Excel 2016 unter macOS. Die lokal installierte Version von Excel, die zum Microsoft 365-Abonnement gehört, funktioniert ebenfalls perfekt – ich empfehle sie sogar, da sie über die neuesten Funktionen verfügt, die Sie in anderen Versionen von Excel nicht finden. Zudem ist es die Version, die ich für dieses Buch verwendet habe. Wenn Sie also mit einer anderen Version von Excel arbeiten, kann es sein, dass Sie kleine Unterschiede im Namen oder in der Position eines Menüelements feststellen.

In diesem Buch verwendete Konventionen

In diesem Buch gelten die folgenden typografischen Konventionen:

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

`Schreibmaschinenschrift`
: Wird in Programmlistings verwendet und im Fließtext für Programmelemente wie zum Beispiel Variablen- oder Funktionsnamen, Datenbanken, Datentypen, Umgebungsvariablen, Anweisungen und Schlüsselwörter.

`Schreibmaschinenschrift fett`
: Kennzeichnet Befehle oder andere Texte, die vom Benutzer buchstäblich eingegeben werden sollen.

`Schreibmaschinenschrift kursiv`
: Zeigt Text, der ersetzt werden soll durch Werte, die der Benutzer bereitstellt, oder Werte, die sich aus dem Kontext ergeben. Außerdem werden die in Tabellen kursiv gedruckten Transformationen derzeit von PyTorch nicht unterstützt.

Dieses Element kennzeichnet einen Tipp oder Vorschlag.

Dieses Element kennzeichnet einen allgemeinen Hinweis.

Dieses Element kennzeichnet eine Warnung oder einen Achtungshinweis.

Codebeispiele

Ich unterhalte eine Webseite (*https://xlwings.org/book*) mit ergänzenden Informationen, die Ihnen bei der Arbeit mit diesem Buch helfen sollen. Diese Seite sollten Sie sich unbedingt anschauen, vor allem dann, wenn Sie auf ein Problem stoßen.

Ergänzendes Material (Codebeispiele, Übungen usw.) stehen Ihnen unter *https://github.com/fzumstein/python-for-excel* zum Download zur Verfügung. Um dieses Begleitmaterial herunterzuladen, klicken Sie auf die grüne Schaltfläche *Code* und wählen dann *Download ZIP*. Nachdem Sie die ZIP-Datei heruntergeladen haben, klicken Sie unter Windows mit der rechten Maustaste auf diese Datei und wählen *Alle extrahieren*, um die im Archiv enthaltenen Dateien in einen Ordner zu entpacken. Unter macOS doppelklicken Sie hierzu einfach auf die Datei. Wenn Sie sich mit Git auskennen, können Sie das Archiv auch per Git auf Ihre lokale Festplatte klonen. Den Zielordner können Sie frei wählen, doch ich verweise im Buch gelegentlich auf folgenden Ordner:

```
C:\Users\username\python-for-excel
```

Wenn Sie die ZIP-Datei unter Windows einfach herunterladen und entpacken, erhalten Sie eine ähnliche Ordnerstruktur wie die folgende (wobei die sich wiederholenden Ordnernamen zu beachten sind):

```
C:\...\Downloads\python-for-excel-1st-edition\python-for-excel-1st-edition
```

Damit Sie Beispiele im Buch leichter verfolgen können, kopieren Sie diesen Ordner am besten in einen Ordner, den Sie unter *C:\Users\<Benutzername>\python-for-excel* anlegen. Das Gleiche gilt für macOS: Kopieren Sie die Dateien nach */Users/<Benutzername>/python-for-excel*.

Wenn Sie eine technische Frage oder ein Problem in Bezug auf die Codebeispiele haben, senden Sie bitte eine E-Mail an *kommentar@oreilly.de*.

Dieses Buch soll Ihnen bei Ihrer täglichen Arbeit helfen. Falls Beispielcode zum Buch angeboten wird, dürfen Sie ihn im Allgemeinen in Ihren Programmen und für

Dokumentationen verwenden. Sie müssen uns nicht um Erlaubnis bitten, es sei denn, Sie kopieren einen erheblichen Teil des Codes. Wenn Sie zum Beispiel ein Programm schreiben, das einige Codeblöcke aus diesem Buch verwendet, benötigen Sie keine Erlaubnis. Sollten Sie aber Beispiele aus O'Reilly-Büchern verkaufen oder verbreiten, ist eine Erlaubnis erforderlich. Wenn Sie eine Frage beantworten und dabei dieses Buch oder Beispielcode aus diesem Buch zitieren, brauchen Sie wiederum keine Erlaubnis. Aber wenn Sie erhebliche Teile des Beispielcodes aus diesem Buch in die Dokumentation Ihres Produkts einfließen lassen, ist eine Erlaubnis einzuholen.

Wir schätzen eine Quellenangabe, verlangen sie aber nicht. Eine Quellenangabe umfasst in der Regel Titel, Autor, Verlag und ISBN, zum Beispiel: »*Python für Excel*« von Felix Zumstein (O'Reilly). Copyright 2022 dpunkt.verlag, ISBN 978-3-96009-197-4.«

Sollten Sie der Meinung sind, dass Sie die Codebeispiele in einer Weise verwenden, die über die oben erteilte Erlaubnis hinausgeht, kontaktieren Sie uns bitte unter *kommentar@oreilly.de*.

Danksagung

Als Erstautor bin ich unglaublich dankbar für die Hilfe, die ich auf dem Weg bis zu diesem Buch von vielen Menschen bekommen habe – sie haben mir diese Reise sehr erleichtert!

Für ihre großartige Arbeit möchte mich bedanken bei meiner Lektorin Melissa Potter von O'Reilly – sie hat mich motiviert, im Zeitplan gehalten und mir geholfen, dieses Buch in eine lesbare Form zu bringen. Danke auch an Michelle Smith, die mit mir an dem ursprünglichen Buchvorschlag gearbeitet hat, und Daniel Elfanbaum, der nicht müde wurde, meine technischen Fragen zu beantworten.

Ein großes Dankeschön geht an alle meine Kollegen, Freunde und Kunden, die viele Stunden damit verbracht haben, meine ersten rudimentären Entwürfe zu lesen. Ihr Feedback war entscheidend, um das Buch verständlicher zu machen, und ein paar Fallstudien sind inspiriert worden von einigen realen Excel-Problemen, die sie mit mir geteilt haben. Mein Dank geht an Adam Rodriguez, Mano Beeslar, Simon Schiegg, Rui Da Costa, Jürg Nager und Christophe de Montrichard.

Hilfreiches Feedback habe ich auch von den Lesern der frühen Fassung erhalten, die auf der Onlinelernplattform von O'Reilly veröffentlicht wurde. Vielen Dank an Felipe Maion, Ray Doue, Kolyu Minevski, Scott Drummond, Volker Roth und David Ruggles!

Ich hatte das große Glück, dass hochqualifizierte technische Gutachter das Buch rezensiert haben, und ich weiß die harte Arbeit sehr zu schätzen, die sie unter großem Zeitdruck geleistet haben. Vielen Dank Jordan Goldmeier, George Mount, Andreas Clenow, Werner Brönnimann und Eric Moreira für all eure Hilfe!

Ein besonderer Dank geht an Björn Stiel, der nicht einfach technischer Rezensent war, sondern von dem ich auch viele der Dinge gelernt habe, über die ich in diesem Buch schreibe. Die schon mehrere Jahre dauernde Zusammenarbeit habe ich sehr genossen.

Nicht zuletzt möchte ich mich bei Eric Reynolds bedanken, der 2016 sein Excel-Python-Projekt in die xlwings-Codebasis integriert hat. Zudem hat er das gesamte Paket von Grund auf neu gestaltet, sodass meine schreckliche API aus den Anfangstagen der Vergangenheit angehört. Herzlichen Dank dafür.

TEIL I

Einführung in Python

KAPITEL 1

Warum Python für Excel?

Normalerweise beginnen Excel-Benutzer, ihre Tabellenkalkulationstools infrage zu stellen, wenn sie an eine Grenze stoßen. Klassischerweise passiert das, wenn z. B. Excel-Arbeitsmappen so viele Daten und Formeln enthalten, dass sie langsam werden oder im schlimmsten Fall sogar abstürzen. Es ist jedoch sinnvoll, dass Sie Ihr Setup hinterfragen, bevor die Dinge den Bach runtergehen: Wenn Sie mit unternehmenskritischen Arbeitsmappen arbeiten, bei denen Fehler zu finanziellen Ausfällen oder Rufschädigungen führen könnten, oder wenn Sie jeden Tag Stunden damit verbringen müssen, Excel-Arbeitsmappen manuell zu aktualisieren, sollten Sie lernen, wie Sie Ihre Prozesse mit einer Programmiersprache automatisieren können. Automatisierung verringert die Gefahr menschlicher Fehler, und Sie können Ihre Zeit mit produktiveren Aufgaben verbringen, als Daten in ein Excel-Tabellenblatt zu kopieren oder einzufügen.

In diesem Kapitel nenne ich einige Gründe, warum Python eine ausgezeichnete Wahl in Kombination mit Excel ist und welche Vorteile es gegenüber der in Excel integrierten Automatisierungssprache VBA bietet. Nachdem ich die Programmiersprache von Excel und deren Besonderheiten vorgestellt habe, gehe ich auf die speziellen Features ein, die Python im Vergleich zu VBA so viel stärker machen. Zunächst aber werfen wir einen Blick auf die Ursprünge unserer beiden Hauptakteure!

In Bezug auf die Computertechnik sind Excel und Python schon recht lange präsent: Excel wurde 1985 von Microsoft auf den Markt gebracht und war – was für viele überraschend sein mag – nur für den Apple Macintosh verfügbar. Erst 1987 bekam Microsoft Windows seine erste Version in Form von Excel 2.0. Allerdings war Microsoft auf dem Markt nicht der Vorreiter für Tabellenkalkulationen: VisiCorp brachte 1979 VisiCalc heraus, gefolgt von Lotus Software im Jahr 1983 mit Lotus 1-2-3. Und selbst Microsoft stieg nicht mit Excel ein: Drei Jahre zuvor brachte das Unternehmen Multiplan heraus, eine Tabellenkalkulation, die sich unter MS-DOS und einigen anderen Betriebssystemen ausführen ließ, aber nicht unter Windows.

Python wurde 1991 geboren, nur sechs Jahre nach Excel. Während aber Excel schon früh populär war, brauchte es bei Python etwas länger, bis es sich in bestimmten Bereichen wie der Webentwicklung oder der Systemadministration durchsetzte. Im Jahr 2005 mauserte sich Python zu einer ernsthaften Alternative für wissenschaftliche Berechnungen, als *NumPy* veröffentlicht wurde, ein Paket für Array-basierte Berechnungen und lineare Algebra. NumPy vereinigt zwei Vorgängerpakete und bündelt damit alle Entwicklungsanstrengungen rund um wissenschaftliches Rechnen in einem einzigen Projekt. Heute bildet es die Basis zahlreicher wissenschaftlicher Pakete, darunter *pandas*, das 2008 herauskam und weitgehend für die breite Akzeptanz von Python in Data Science und Finanzwesen verantwortlich ist, die nach 2010 einsetzte. Dank pandas ist Python neben R zu einer der am häufigsten verwendeten Sprachen für Aufgaben der Data Science, wie Datenanalyse, Statistik und maschinelles Lernen, geworden.

Die Tatsache, dass Python und Excel schon vor langer Zeit erfunden wurden, ist nicht die einzige Gemeinsamkeit: Excel und Python sind auch Programmiersprachen. Wahrscheinlich überrascht es Sie nicht, dies über Python zu hören, doch für Excel bedarf es vielleicht einer Erklärung, die jetzt umgehend folgt.

Excel ist eine Programmiersprache

Dieser Abschnitt beginnt mit einer Einführung in Excel als Programmiersprache. Damit können Sie besser verstehen, warum Probleme mit Tabellenkalkulationen regelmäßig in den Nachrichten auftauchen. Dann werfen wir einen Blick auf einige bewährte Verfahren, die sich in der Community der Softwareentwickler herausgebildet haben und die Sie vor vielen typischen Excel-Fehlern bewahren können. Wir schließen mit einer kurzen Einführung in Power Query und Power Pivot, zwei moderne Excel-Tools, die die Art von Funktionalität abdecken, für die wir stattdessen pandas verwenden werden.

Ist es nicht nur Ihre Einkaufsliste, die Sie mit Excel verwalten, verwenden Sie sicherlich auch Funktionen wie `=SUM(A1:A4)`, um einen Bereich von Zellen zu summieren. Wenn Sie einen Moment darüber nachdenken, wie das funktioniert, werden Sie feststellen, dass der Wert einer Zelle üblicherweise von einer oder mehreren anderen Zellen abhängt, die wiederum Funktionen verwenden können, die von einer oder mehreren anderen Zellen abhängen usw. Derartig verschachtelte Funktionsaufrufe unterscheiden sich nicht von der Funktionsweise anderer Programmiersprachen, nur dass Sie den Code in Zellen statt in Textdateien schreiben. Und sollte Sie das immer noch nicht überzeugt haben: Ende 2020 hat Microsoft angekündigt, *Lambda-Funktionen* einzuführen, womit Sie in der Excel-eigenen Formelsprache wiederverwendbare Funktionen schreiben können, und zwar ohne auf eine andere Sprache wie VBA zurückgreifen zu müssen. Laut dem Produktchef von Excel, Brian Jones, war dies das fehlende Teil, das Excel letztlich zu einer »echten« Programmiersprache macht.[1] Das bedeutet auch, dass Excel-Benutzer eigentlich Excel-Programmierer genannt werden sollten!

1 Die Ankündigung der Lambda-Funktionen können Sie im Excel-Blog unter *https://oreil.ly/4-0y2* nachlesen.

Bei Excel-Programmierern gibt es jedoch eine Besonderheit: Die meisten von ihnen sind Geschäftsanwender oder Fachexperten ohne formale Informatikausbildung. Es sind Kaufleute, Buchhalter oder Ingenieure, um nur ein paar Beispiele zu nennen. Ihre Tools für die Tabellenkalkulation sind darauf ausgerichtet, ein Geschäftsproblem zu lösen, wobei sie häufiger mal bewährte Verfahren der Softwareentwicklung ignorieren. Infolgedessen mischen diese Tabellenkalkulationstools oft Eingaben, Berechnungen und Ausgaben auf denselben Tabellenblättern, sie führen Schritte aus, die nicht unbedingt nachvollziehbar sind, aber für eine ordnungsgemäße Funktion gebraucht werden, und kritische Änderungen erfolgen ohne jegliches Sicherheitsnetz. Mit anderen Worten: Den Tabellenkalkulationstools fehlt eine solide Anwendungsarchitektur, und oftmals sind sie weder gut dokumentiert noch ausreichend getestet. Manchmal haben diese Probleme verheerende Folgen: Wenn Sie vergessen, Ihre Arbeitsmappe neu zu berechnen, bevor Sie eine Aktientransaktion absetzen, kaufen oder verkaufen Sie möglicherweise die falsche Anzahl von Aktien und verlieren dadurch Geld. Und wenn es nicht nur Ihr eigenes Geld ist, mit dem Sie handeln, schaffen Sie es bis in die Nachrichten, wie wir gleich sehen werden.

Excel in den Nachrichten

Excel ist regelmäßig in den Nachrichten vertreten, und während ich dieses Buch geschrieben habe, kamen zwei neue Geschichten in die Schlagzeilen. In der ersten ging es um das HUGO Gene Nomenclature Committee, das einige menschliche Gene umbenannt hat, damit sie von Excel nicht mehr als Datumswerte interpretiert werden. Um beispielsweise zu verhindern, dass das Gen `MARCH1` in `1-Mar` umbenannt wird, wurde es in `MARCHF1` umbenannt.[2] In der zweiten Geschichte wurde Excel für die verspätete Meldung von 16.000 COVID-19-Testergebnissen in Großbritannien verantwortlich gemacht. Verursacht wurde das Problem dadurch, dass die Testergebnisse im älteren Excel-Dateiformat (*.xls*) festgehalten wurden, das mit maximal etwa 65.000 Zeilen umgehen kann. Größere Datensätze wurden jenseits dieser Grenze einfach abgeschnitten.[3] Während diese beiden Geschichten die anhaltende Bedeutung und Dominanz von Excel in der heutigen Welt zeigen, gibt es wahrscheinlich keinen »Excel-Vorfall«, der berühmter ist als der London Whale.

London Whale ist der Spitzname eines Händlers, dessen Handelsfehler JP Morgan im Jahr 2012 dazu zwangen, einen erschütternden Verlust von sechs Milliarden Dollar bekannt zu geben. Die Ursache für die Panne war ein Excel-basiertes Value-at-Risk-Modell, das das tatsächliche Risiko eines Geldverlusts in einem ihrer Portfolios erheblich unterschätzt hatte. Der *Report of JPMorgan Chase & Co. Manage-*

2 James Vincent, *Scientists rename human genes to stop Microsoft Excel from misreading them as dates*. The Verge, 6. August 2020, *https://oreil.ly/0qo-n*.

3 Leo Kelion, *Excel: Why using Microsoft's tool caused COVID-19 results to be lost*. BBC News, 5. Oktober 2020, *https://oreil.ly/vvB6o*.

ment Task Force Regarding 2012 CIO Losses[4] (2013) erwähnt, dass »das Modell mit einer Reihe von Excel-Tabellen arbeitet, die manuell durch Kopieren und Einfügen von Daten aus einer Tabelle in eine andere vervollständigt werden müssen«. Zusätzlich zu diesen operativen Problemen gab es einen logischen Fehler: In einer Berechnung wurde durch eine Summe statt durch einen Mittelwert dividiert.

Wenn Sie mehr von diesen Geschichten lesen möchten, sehen Sie sich die Webseite *Horror Stories* an (*https://oreil.ly/WLO-I*), die von der *European Spreadsheet Risks Interest Group* (EuSpRIG) unterhalten wird.

Um zu verhindern, dass Ihre Firma mit einer ähnlichen Story in den Nachrichten landet, sollten Sie sich den nächsten Abschnitt zu bewährten Verfahren ansehen, um Ihre Arbeit mit Excel erheblich sicherer zu machen.

Best Practices der Programmierung

Dieser Abschnitt macht Sie mit den wichtigsten Best Practices der Programmierung bekannt, darunter Trennung der Belange, DRY-Prinzip, Testen und Versionskontrolle. Wie Sie sich selbst überzeugen können, lassen sich die Best Practices einfacher befolgen, wenn Sie Python zusammen mit Excel verwenden.

Trennung der Belange

Zu den wichtigsten Entwurfsprinzipien in der Programmierung gehört die *Trennung der Belange*, die man auch als *Modularität* bezeichnet. Es bedeutet, dass zusammengehörende Funktionen in einem separaten Teil eines Programms untergebracht werden, damit dieser sich leicht ersetzen lässt, ohne die übrige Anwendung zu beeinträchtigen. Auf der obersten Ebene gliedert man eine Anwendung häufig in die folgenden Schichten:[5]

- Darstellungsschicht
- Geschäftsschicht
- Datenschicht

Um die Aufgaben dieser Schichten zu beschreiben, betrachten wir einen einfachen Konverter für Währungen, wie ihn Abbildung 1-1 zeigt. Die Excel-Datei *currency_converter.xlsx* finden Sie im *xl*-Ordner des Begleit-Repositorys.

Die Anwendung funktioniert wie folgt: Sie geben den Betrag und die Währung in die Zellen A4 bzw. B4 ein. Excel wandelt dies in US-Dollar um und schreibt das Ergebnis in Zelle D4. Viele Tabellenkalkulationen sind in dieser Weise konzipiert und werden von Unternehmen tagtäglich eingesetzt. Die Anwendung werde ich jetzt in ihre Schichten aufteilen:

4 Wikipedia verweist auf das Dokument in einer der Fußnoten (*https://oreil.ly/0uUj9*) in ihrem Artikel über den Fall.

5 Die Terminologie entstammt dem *Microsoft Application Architecture Guide, 2nd Edition*, der online verfügbar ist (*https://oreil.ly/8V-GS*), allerdings nicht mehr regelmäßig aktualisiert wird.

Darstellungsschicht
: Dies ist die Schicht, die Sie sehen und mit der Sie interagieren, d. h. die Benutzeroberfläche: Die Werte der Zellen A4, B4 und D4 bilden zusammen mit ihren Beschriftungen die Darstellungsschicht des Währungskonverters.

Geschäftsschicht
: Diese Schicht realisiert die anwendungsspezifische Logik: Zelle D4 definiert, wie der Betrag in USD konvertiert wird. Die Formel `=A4 * VLOOKUP(B4, F4:G11, 2, FALSE)` lässt sich als Betrag mal Wechselkurs übersetzen.

Datenschicht
: Wie der Name vermuten lässt, greift diese Schicht auf die Daten zu: Der `VLOOKUP`-Teil von Zelle D4 übernimmt diese Aufgabe.

Die Datenschicht greift auf die Daten aus der Wechselkurstabelle zu, die in Zelle F3 beginnt und die als Datenbank dieser kleinen Anwendung dient. Sicherlich ist Ihnen aufgefallen, dass Zelle D4 in allen drei Schichten erscheint: Diese einfache Anwendung mischt Darstellungs-, Geschäfts- und Datenschicht in einer einzigen Zelle.

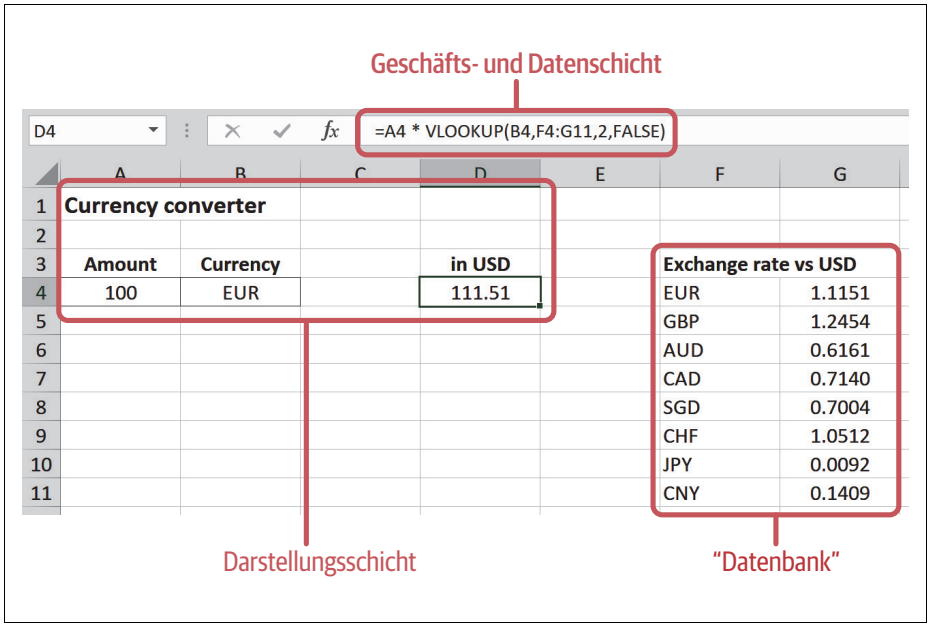

Abbildung 1-1: Die Excel-Datei currency_converter.xlsx

Für diesen einfachen Währungskonverter ist das nicht unbedingt ein Problem, aber oft wird aus einer kleinen Excel-Datei schon recht bald eine viel größere Anwendung. Wie lässt sich diese Situation verbessern? Die meisten professionellen Excel-Entwickler raten dazu, für jede Ebene – in der Excel-Terminologie üblicherweise *Eingaben*, *Berechnungen* und *Ausgaben* genannt – ein eigenes Tabellenblatt

zu verwenden. Oftmals kombiniert man das mit bestimmten Farbcodes für jede Schicht, z. B. einem blauen Hintergrund für alle Eingabezellen. In Kapitel 11 erstellen wir eine echte Anwendung, die auf diesen drei Schichten basiert: Excel wird die Darstellungsschicht sein, während die Geschäfts- und Datenschichten nach Python verlagert werden, wo es viel einfacher ist, den Code ordnungsgemäß zu strukturieren.

Nachdem Sie nun wissen, was sich hinter der Trennung von Belangen verbirgt, wollen wir herausfinden, worum es sich beim DRY-Prinzip handelt.

Das DRY-Prinzip

Der pragmatische Programmierer von Hunt und Thomas (Pearson Education) hat das DRY-Prinzip populär gemacht: *Wiederholen Sie sich nicht* (DRY – *Don't Repeat Yourself*). Doppelter Code bedeutet mehr Codezeilen und mehr Fehlerquellen, wodurch der Code schwieriger zu warten ist. Wenn sich Ihre Geschäftslogik in Ihren Zellformeln befindet, ist es praktisch unmöglich, das DRY-Prinzip anzuwenden, da es keinen Mechanismus gibt, durch den Sie den Code in einer anderen Arbeitsmappe wiederverwenden können. Leider bedeutet das, dass Sie ein neues Excel-Projekt üblicherweise damit beginnen, dass Sie die Arbeitsmappe aus dem vorherigen Projekt oder von einer Vorlage kopieren.

Wenn Sie in VBA schreiben, ist der am häufigsten anzutreffende wiederverwendbare Code eine Funktion. Eine Funktion bietet Ihnen Zugriff auf denselben Codeblock, zum Beispiel von mehreren Makros aus. Wenn Sie mehrere Funktionen haben, die Sie ständig verwenden, werden Sie sie vielleicht in mehreren Arbeitsmappen gemeinsam verwenden wollen. Das Standardinstrument für die gemeinsame Nutzung von VBA-Code in mehreren Arbeitsmappen sind Add-ins. Allerdings fehlt bei VBA-Add-ins eine zuverlässige Möglichkeit, sie zu verteilen und zu aktualisieren. Zwar hat Microsoft einen Excel-internen Add-in-Speicher eingeführt, um dieses Problem zu lösen, doch funktioniert er nur mit JavaScript-basierten Add-ins, kommt also für VBA-Programmierer nicht infrage. Demzufolge ist es immer noch üblich, mit VBA per Copy-and-paste zu arbeiten: Nehmen wir an, Sie bräuchten eine *kubische Spline-Funktion* in Excel. Mit einer kubischen Spline-Funktion lässt sich eine Kurve basierend auf wenigen Punkten in einem Koordinatensystem interpolieren. Häufig wird sie von Händlern festverzinslicher Wertpapiere verwendet, um eine Zinskurve für alle Laufzeiten auf der Grundlage einiger bekannter Kombinationen von Laufzeit und Zinssatz abzuleiten. Wenn Sie im Internet nach »kubische splines excel« suchen, dauert es nicht allzu lange, bis Sie auf eine Seite mit VBA-Code stoßen, die genau das erledigt, was Sie wollen. Problematisch dabei ist, dass diese Funktionen in den meisten Fällen von einer einzelnen Person mit wahrscheinlich guten Absichten, aber ohne formale Dokumentation oder Tests geschrieben wurden. Vielleicht funktionieren sie für die Mehrheit der Eingaben, aber was ist mit einigen ungewöhnlichen Randfällen? Wenn Sie mit

einem festverzinslichen Portfolio im Wert von mehreren Millionen handeln, brauchen Sie etwas, dem Sie vertrauen können. Zumindest werden Ihnen das Ihre internen Prüfer sagen, wenn sie feststellen, woher der Code stammt.

Wie der letzte Abschnitt dieses Kapitels zeigt, erleichtert Python die Verteilung des Codes mithilfe eines Paketmanagers. Bevor wir jedoch dazu kommen, wollen wir uns mit dem Testen befassen, einem der Eckpfeiler einer soliden Softwareentwicklung.

Testen

Wenn Sie einem Excel-Entwickler sagen, er möge Arbeitsmappen testen, wird er höchstwahrscheinlich einige Stichproben machen: auf eine Schaltfläche klicken und sehen, ob das Makro immer noch tut, was es tun soll, oder einige Eingaben ändern und prüfen, ob die Ausgabe vernünftig aussieht. Diese Strategie ist allerdings riskant: In Excel passieren leicht Fehler, die schwer aufzuspüren sind. Zum Beispiel können Sie eine Formel versehentlich durch einen fest codierten Wert überschreiben. Oder Sie vergessen, eine Formel in einer ausgeblendeten Spalte anzupassen.

Wenn Sie einem professionellen Softwareentwickler sagen, er möge seinen Code testen, schreibt er *Komponententests* (auch Modultests oder Unittests genannt). Wie aus dem Namen hervorgeht, handelt es sich um einen Mechanismus, um einzelne Komponenten eines Programms zu testen. Zum Beispiel stellen Komponententests sicher, dass eine einzelne Funktion eines Programms ordnungsgemäß funktioniert. Die meisten Programmiersprachen bieten eine Möglichkeit, Komponententests automatisch auszuführen. Dadurch lässt sich die Zuverlässigkeit Ihrer Codebasis erheblich erhöhen und einigermaßen sicherstellen, dass Sie beim Bearbeiten Ihres Codes nichts kaputt machen, was bisher funktioniert hat.

Wenn Sie sich das Tool zur Umrechnung von Währungen in Abbildung 1-1 ansehen, könnten Sie mit einem Test prüfen, ob die Formel in Zelle D4 bei den folgenden Eingaben das korrekte Ergebnis USD 105 zurückgibt: 100 EUR als Betrag und 1,05 als EUR-USD-Wechselkurs. Warum ist das hilfreich? Angenommen, Sie löschen versehentlich Zelle D4 mit der Umrechnungsformel und müssen sie neu schreiben: Anstatt den Betrag mit dem Wechselkurs zu multiplizieren, dividieren Sie durch den Wechselkurs – schließlich kann der Umgang mit Währungen verwirrend sein. Wenn Sie dann den obigen Test ausführen, erhalten Sie einen Testfehler, da 100 EUR/1,05 nicht mehr das erwartete Testergebnis 105 USD liefert. Auf diese Weise können Sie die fehlerhafte Formel erkennen und korrigieren, bevor Sie das Tabellenblatt an Ihre Benutzer übergeben.

Fast alle herkömmlichen Programmiersprachen bieten ein oder mehrere Test-Frameworks, um Komponententests ohne großen Aufwand schreiben zu können – aber nicht Excel. Glücklicherweise ist das Konzept der Komponententests einfach genug, und durch die Kopplung von Excel mit Python können Sie auf die leistungsfähigen Frameworks für Komponententests von Python zugreifen. Auch wenn eine ausführliche Darstellung von Komponententests den Rahmen dieses Buchs spren-

gen würde, lade ich Sie ein, einen kurzen Blick auf meinen Blogbeitrag (*https://oreil.ly/crwTm*) zu werfen, in dem ich das Thema anhand von praktischen Beispielen durchgehe.

Komponententests werden oft so eingerichtet, dass sie automatisch ausgeführt werden, wenn Sie Ihren Code an Ihr Versionskontrollsystem übergeben. Der nächste Abschnitt erläutert, was Versionskontrollsysteme sind und warum es schwierig ist, sie für Excel-Dateien zu verwenden.

Versionskontrolle

Ein weiteres Merkmal professioneller Programmierer ist, dass sie ein System zur *Versionskontrolle* oder *Quellcodekontrolle* verwenden. Ein *Versionskontrollsystem* (VCS) verfolgt Änderungen in Ihrem Quellcode im Laufe der Zeit, sodass Sie sehen können, wer was wann und warum geändert hat, und erlaubt Ihnen, zu jedem Zeitpunkt zu alten Versionen zurückzukehren. Das beliebteste Versionskontrollsystem ist heutzutage Git (*https://git-scm.com/*). Ursprünglich wurde es entwickelt, um den Linux-Quellcode zu verwalten, und hat seitdem die Programmierwelt erobert – selbst Microsoft hat Git im Jahr 2017 für die Verwaltung des Windows-Quellcodes übernommen. In der Excel-Welt kommt das mit Abstand beliebteste Versionskontrollsystem in Form eines Ordners daher, in dem Dateien wie diese archiviert werden:

```
currency_converter_v1.xlsx
currency_converter_v2_2020_04_21.xlsx
currency_converter_final_edits_Bob.xlsx
currency_converter_final_final.xlsx
```

Wenn sich der Excel-Entwickler – anders als in diesem Beispiel – an eine bestimmte Konvention im Dateinamen hält, ist daran an sich nichts auszusetzen. Doch wenn Sie den Versionsverlauf Ihrer Dateien lokal verwalten, entgehen Ihnen wichtige Aspekte der Versionskontrolle. Das betrifft unter anderem eine einfachere Zusammenarbeit, Peer-Reviews, Genehmigungsprozesse und Revisionsaufzeichnungen. Und wenn Sie Ihre Arbeitsmappen sicherer und stabiler machen möchten, werden Sie nicht auf solche Dinge verzichten wollen. Professionelle Programmierer verwenden in der Regel Git in Verbindung mit einer webbasierten Plattform wie GitHub, GitLab, Bitbucket oder Azure DevOps. Bei diesen Plattformen ist es möglich, mit sogenannten *Pull Requests* oder *Merge Requests* zu arbeiten. Entwickler können damit formell beantragen, dass ihre Änderungen in die Quellcodebasis übernommen werden. Ein Pull Request bietet die folgenden Informationen:

- Wer ist der Autor der Änderungen?
- Wann wurden die Änderungen vorgenommen?
- Welchen Zweck haben die in der Commit-Nachricht beschriebenen Änderungen?

- Welche Details der Änderungen werden in der mit `git diff` erzeugten Ansicht, die Änderungen für neuen Code grün und für gelöschten Code rot hervorhebt, als Unterschiede angezeigt?

Eine Kollegin oder ein Teamleiter kann somit die Änderungen überprüfen und Unregelmäßigkeiten feststellen. Oftmals ist ein zusätzliches Augenpaar in der Lage, den einen oder anderen Fehler auszumachen oder dem Programmierer anderweitig wertvolles Feedback zu geben. Angesichts dieser Vorteile stellt sich die Frage, warum Excel-Entwickler es vorziehen, das lokale Dateisystem und ihre eigene Namenskonvention zu verwenden, anstatt auf ein professionelles System wie Git zurückzugreifen.

- Viele Excel-Benutzer kennen Git einfach nicht, oder sie geben frühzeitig auf, denn Git hat eine relativ steile Lernkurve.
- Mit Git können mehrere Benutzer parallel an lokalen Kopien derselben Datei arbeiten. Nachdem alle ihre Arbeit committet haben, kann Git normalerweise alle Änderungen zusammenführen, ohne dass manuell eingegriffen werden muss. Bei Excel-Dateien funktioniert das nicht: Wenn sie parallel in separaten Kopien geändert werden, weiß Git nicht, wie es diese Änderungen wieder zu einer einzigen Datei zusammenführen soll.
- Selbst wenn es Ihnen gelingt, die vorgenannten Probleme in den Griff zu bekommen, bietet Git bei Excel-Dateien einfach nicht den gleichen Nutzen wie bei Textdateien. Git ist nicht in der Lage, Änderungen zwischen Excel-Dateien anzuzeigen, was einen zielführenden Peer-Review-Prozess verhindert.

Aufgrund all dieser Probleme hat mein Unternehmen xltrail (*https://xltrail.com/*) entwickelt, ein Git-basiertes Versionskontrollsystem, das mit Excel-Dateien umgehen kann. Es verbirgt die Komplexität von Git, sodass gewerbliche Anwender es bequem nutzen können, und ermöglicht auch die Verbindung zu externen Git-Systemen, falls Sie zum Beispiel Ihre Dateien bereits mit GitHub verfolgen. xltrail verfolgt verschiedene Komponenten einer Arbeitsmappe wie Zellformeln, benannte Bereiche, Power Queries und VBA-Code, sodass Sie auch von den klassischen Vorteilen der Versionskontrolle einschließlich der Peer-Reviews profitieren können.

Die Versionskontrolle mit Excel lässt sich auch dadurch vereinfachen, dass man die Geschäftslogik aus Excel herausnimmt und in Python-Dateien unterbringt, was wir in Kapitel 10 tun werden. Da sich Python-Dateien problemlos mit Git verfolgen lassen, haben Sie den wichtigsten Teil Ihres Tabellenkalkulationsprogramms unter Kontrolle.

Dieser Abschnitt ist zwar mit »Best Practices der Programmierung« überschrieben, stellt aber vor allem heraus, warum diese bei Excel schwerer zu befolgen sind als bei einer herkömmlichen Programmiersprache wie Python. Bevor wir uns Python zuwenden, möchte ich kurz Power Query und Power Pivot vorstellen, den Versuch von Microsoft, Excel zu modernisieren.

Modernes Excel

Die moderne Ära von Excel beginnt mit Excel 2007, als das Menüband und die neuen Dateiformate (z. B. *.xlsx* statt *.xls*) eingeführt wurden. Die Excel-Community meint aber mit »modernes Excel« die Tools, die mit Excel 2010 hinzugekommen sind: vor allem Power Query und Power Pivot. Mit diesen Tools können Sie eine Verbindung zu externen Datenquellen herstellen und Daten analysieren, die zu groß sind, um in ein Tabellenblatt zu passen. Da sich ihre Funktionen mit dem überschneiden, was wir mit pandas in Kapitel 5 machen werden, stelle ich sie im ersten Teil dieses Abschnitts kurz vor. Im zweiten Teil geht es um Power BI, eine eigenständige Business-Intelligence-Anwendung, die die Funktionen von Power Query und Power Pivot mit Visualisierungsfähigkeiten kombiniert – und das mit integrierter Unterstützung für Python!

Power Query und Power Pivot

Mit Excel 2010 hat Microsoft ein Add-in namens *Power Query* eingeführt. Power Query ermöglicht es, auf die unterschiedlichsten Datenquellen zuzugreifen, einschließlich Excel-Arbeitsmappen, CSV-Dateien und SQL-Datenbanken. Es bietet auch Verbindungen zu Plattformen wie Salesforce und kann sogar erweitert werden, um Verbindungen zu Systemen einzurichten, die standardmäßig nicht abgedeckt sind. Die Kernfunktionalität von Power Query ist auf Datensätze ausgerichtet, die zu groß für ein Tabellenblatt sind. Nach dem Laden der Daten können Sie in zusätzlichen Schritten die Daten bereinigen und bearbeiten, damit sie in einem brauchbaren Format in Excel ankommen. Zum Beispiel könnten Sie eine Spalte in zwei Spalten aufteilen, zwei Tabellen zusammenführen oder Ihre Daten filtern und gruppieren. Seit Excel 2016 ist Power Query kein Add-in mehr, sondern lässt sich direkt im Menüband auf der Registerkarte *Daten* über die Schaltfläche *Daten importieren* aufrufen. Unter macOS ist Power Query nur teilweise verfügbar – es wird jedoch aktiv weiterentwickelt, sodass es in einer zukünftigen Version von Excel vollständig unterstützt werden sollte.

Power Pivot geht Hand in Hand mit Power Query: Konzeptionell ist es der zweite Schritt, nachdem Sie Ihre Daten mit Power Query erfasst und bereinigt haben. Power Pivot hilft Ihnen, Ihre Daten direkt in Excel zu analysieren und ansprechend darzustellen. Stellen Sie es sich als eine herkömmliche Pivot-Tabelle vor, die wie Power Query mit großen Datensätzen umgehen kann. Mit Power Pivot können Sie formale Datenmodelle mit Beziehungen und Hierarchien definieren, und über die Formelsprache DAX können Sie berechnete Spalten hinzufügen. Power Pivot ist ebenfalls mit Excel 2010 eingeführt worden, bleibt aber ein Add-in und ist bislang nicht unter macOS verfügbar.

Wenn Sie mit Power Query und Power Pivot arbeiten und darauf aufbauend Dashboards erstellen möchten, ist Power BI einen Blick wert – sehen Sie sich an, warum!