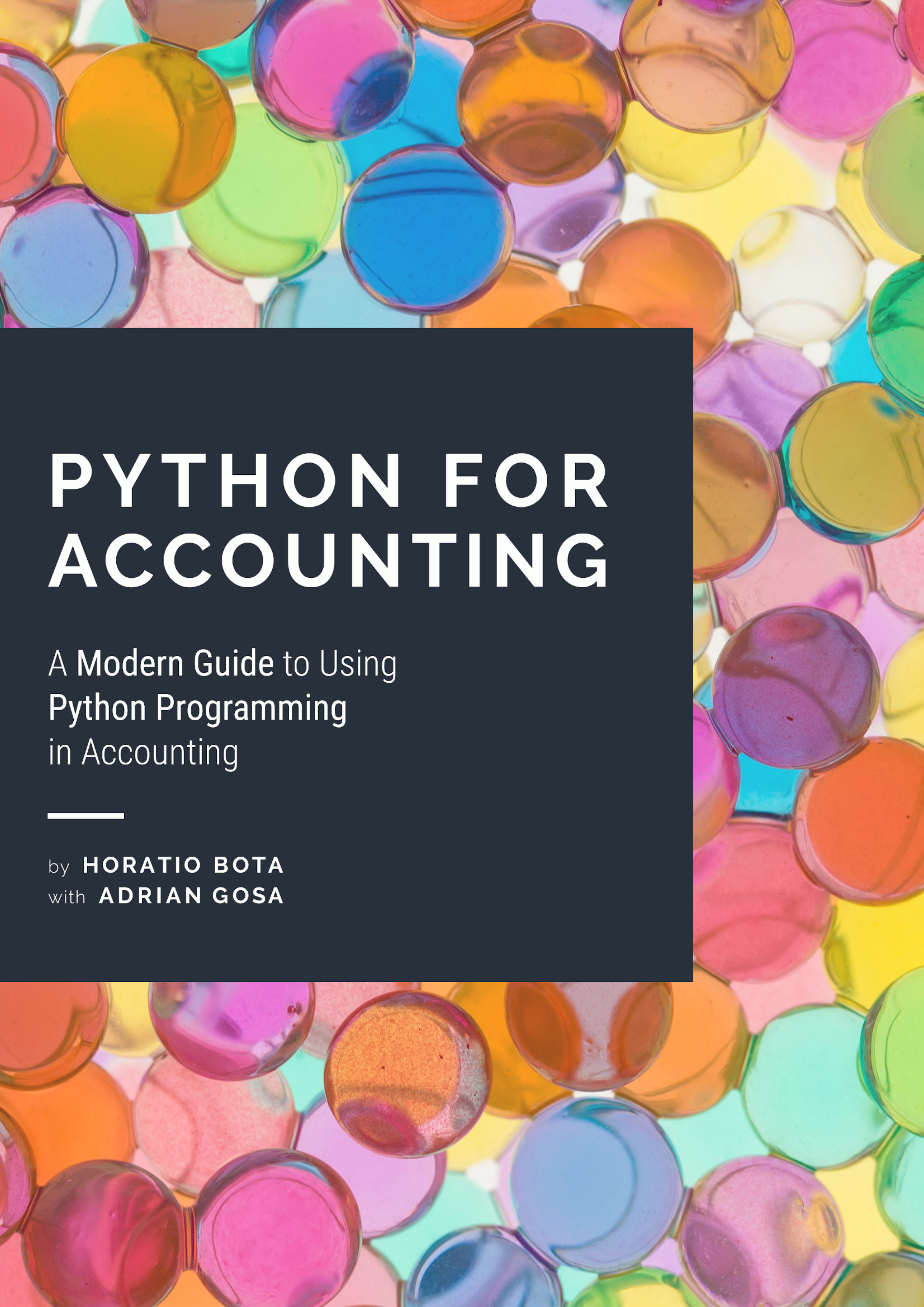


PYTHON FOR ACCOUNTING

A **Modern Guide** to Using **Python Programming** in Accounting

by **HORATIO BOTA**
with **ADRIAN GOSA**

Python for Accounting

A Modern Guide to Using
Python Programming in Accounting

by Horatio Bota
with Adrian Gosa

Library of Congress Control Number: 2021901161
ISBN: 978-973-0-33892-8
Version identifier: 115431b

About the author

Horatio Bota is a freelance Data Scientist with over seven years of experience in data analytics and data science. He has previously worked at Microsoft Research, J.P. Morgan Chase, and several startups in the U.K. He holds a B.Sc. and a Ph.D. in Computing Science from the University of Glasgow.

About the content reviewer

Adrian Gosa is a Senior Accountant at *Nike Europe*, with over seven years of business and finance experience. He has previously worked at PricewaterhouseCoopers and Deloitte, covering a wide variety of industries. Adrian holds an M.A. in Business Economics and an M.Sc. in Quantitative Finance from the University of Glasgow.

About the technical reviewer

Alexandra Vtyurina is a Research Scientist at *Kira Systems* in Toronto, where she uses Python to analyze user behavior. She holds a B.Sc. and M.Sc. in Computer Science from the Southern Federal University, and is about to get her Ph.D. in Computer Science from the University of Waterloo.

Contact

Did you like the book? Did you find it helpful? We'd love to add your name to our list of testimonials on the website! Please email us at **contact@pythonforaccounting.com**.

If you'd like to report any mistakes, typos, or offer suggestions on how we can improve this book, please email us at **errata@pythonforaccounting.com**.

Get updates via Twitter

If you'd like to be notified of updates to the book, follow **@pfabook** on Twitter.

Contents

Introduction 1

If you work in accounting, you've probably heard the rumors: artificial intelligence and automation are set to *"reshape the accounting function"* over the next few years. *When exactly* the robots take over[1] is still uncertain, but *how* they'll do that is already clear: someone will program them. More likely than not, that someone is you.

1: The deadline keeps getting postponed since the 1950s.

You're best suited to automate the repetitive tasks in your job because you already know how: you have lots of domain knowledge and are used to working with computers and data.[2] You even have programming experience: most accounting work involves creating custom computer programs through point-and-click operations instead of code — these programs are called spreadsheets. You probably know what *"functions"* or other programming concepts are, and more importantly, you know how to *think* about data manipulation in powerful ways.

2: And because you're reading this book, which means you want to use programming in your work.

Unfortunately, the tools used in accounting today limit how expressive you can be when working with data. How many times have you found yourself digging through Excel's menus for a command that almost does what you want it to, but not entirely? How painless is it to work with large files in Excel? How simple is it to automate the boring parts of your workflow?

Unlike Excel, Python is a tool designed for expressing any data manipulation you can think of. On top of that, Python is fast, handles spreadsheets that Excel can't even open, and working with it always leaves a trace (i.e., code) that you can check and run whenever you need to. Even better, because you're already familiar with programming ideas and have the right mental models for working with data, all you need to start using Python is a guide on how to add it to your accounting toolkit.

This book is your guide. It starts with Python programming basics and goes all the way to making interactive data visualization using Python. On the way, it introduces many of the tools that have become a foundation for data science — all the while keeping its focus on accounting tasks and data (i.e., on using Python with spreadsheets, not instead of them).

The rest of this introduction explains how Python enables powerful data analysis practices that can strip away much of the boring parts of your work. Hopefully, by the end of this chapter, it will be clear why reading the rest of this book is worth your time.

What this book is about

This book is about computer programming more than it is about accounting practices: it teaches you how to use the Python programming language to work with tables. You'll use general ledgers, cash flow statements, and other accounting datasets throughout the book, but the goal is to teach you how to work with those datasets using Python code rather than introduce new accounting ideas on the way.

There's more than one kind of code: websites, smartphone apps, or the operating system on your computer are all built with code, but each is made with different programming languages and tools. Even when handling tables with Python, you can quickly glue spreadsheets together in a few lines of code, or you can code up an entire data analysis project. Both approaches use code, but larger projects employ more of Python's gears and tools.

As it happens, accounting tasks are a mix of quick data wrangling (e.g., moving rows between spreadsheets, cleaning text entries, reconciling values across files) and extensive data analysis projects (e.g., margin or costing analysis, inventory forecasts). You can easily use Python for both, but you need to use more of Python's machinery for larger projects. As we progress with the book, I'll introduce different parts of that machinery and go through several data wrangling projects using accounting datasets before attempting a Python-based sales analysis project. This way, you'll see how to use Python for the boring parts in your work first, and how to set up an entire data analysis project in Python towards the end of the book.

Whom this book is for

If you use a computer at work in any way, knowing how to write even a few lines of Python code will make you more productive (and you might even find coding fun). This book is for anyone who works with accounting data in Excel and wants to use Python to improve their workflow:

- Accounting, finance, or economics students
- Accountants working with anything from payroll to costing
- Business controllers
- Auditors
- Financial planners

If you are in any of these groups, you've likely experienced the frustration of using Excel with larger spreadsheets or of trying to get an Excel chart to look the way you want it to. Fortunately, Python can help with both of these — and more.

You don't need any Python or general coding knowledge to get started. However, you'll get the most out of this book if you know what a pivot table is and how to use at least some of Excel's functions (e.g., `SUM` or `LEN`). If you've been using Excel in your work for a while, you're good to go.

Some of you may have used `R` or Python[3] before. In that case, the first part of the book will serve as a refresher for general programming concepts and guide you through setting up the tools you need to run Python data analysis code. If this is you, feel free to jump ahead and read the chapters you find most useful.

3: Both `R` and Python are programming languages used widely for data analysis and statistics.

Why read this book

Much has been written about coding as an essential skill, with everyone from Steve Jobs to Barack Obama encouraging people to learn how to code. While knowing how to code can be useful for anyone working with digital content, it's a crucial skill for accounting professionals because accounting is a type of programming that uses the wrong tools.

What's wrong with Excel for accounting

Python is essential in tech and science, but in accounting, Excel is king because most accountants are familiar with Excel, and it works well for certain tasks.[4] However, there are genuine drawbacks to using Excel as your primary data analysis tool.

4: Data entry is one of those tasks.

The first drawback is that the work you do in Excel is not easily reproducible. With Excel, you don't have a record of all the steps you took in your analysis, so you can't rerun those steps if something goes wrong[5] or if you get a new dataset that needs the same kind of analysis. Repeating the same actions over and over, every time the data changes or Excel crashes is not only time consuming and annoying, but exceedingly error-prone.

5: And Excel does have a fondness for crashing without saving your work, at the worst of times.

A related drawback is that data in spreadsheets just *is*: you don't always know how it was generated, where it was sourced from, or if anyone accidentally modified critical values in the email back-and-forth. Most analyses involve many cleaning or transformation steps, which can't be easily communicated over email chains, so often you can't check the assumptions that went into generating a certain dataset. Because operations in Excel don't leave a trace, errors introduced in a spreadsheet get discovered too late, after reports have been published and decisions already taken.[6]

6: "Growth in the Time of Debt" is a well-known academic paper written by two Harvard professors that made the case for austerity measures back in 2010. It is also one of the most famous cases of how consequential Excel data analysis errors can be.

Lastly, Excel has hard limits on the size of data you can work with. An Excel spreadsheet can have at most 1 048 576 rows and 16 384 columns.[7] Even though these limits seem reasonable, you don't need a massive business to gather data that go beyond these limits nowadays. Even if your data is within limits, how easy is it to open or work with a large dataset in Excel?[8]

7: There are more hard limits, not just on the number of rows or column. You can read more about them in the Excel specifications and limits article on the Microsoft Office support website.

8: You can find a recent example of how consequential Excel's limits are at: "Excel: Why using Microsoft's tool caused Covid-19 results to be lost".

All of the drawbacks mentioned above can be overcome by handling data using modern tools and practices (i.e., code, documentation, tests) that are open and explicit at every step (so you can easily find errors and fix them), scalable (that don't freeze your computer at over a million rows), and documented (so that anyone in your team can reproduce your analysis whenever they need to). Which is where Python comes in.

What Python is

Python is a programming language. A programming language is a set of rules[9] for writing text such that your computer can understand what that text means. When you install Python on your computer, you install a program (just like you install Excel) that knows how to turn text (i.e., Python code) into instructions for your computer processor to run.

9: Much like the English language is a set of rules of what different words mean and how you can put one after the other to form sentences.

It's worth highlighting that the word *"Python"* refers to three distinct concepts:

- The ***Python programming language*** is the set of rules that defines how Python code can be written and interpreted by humans or computers (i.e., what symbols or keywords you can use when writing code, such as `for` or `if`, what they mean, etc.);
- The ***Python interpreter*** (or Python implementation) is the computer program that can read Python code[10] and transform it into instructions that your computer's processor can understand and execute. Interpreters are often developed by the same community that decides what the Python language rules are. There are many Python interpreters available (just like there are many applications you can use to work with spreadsheets, not just Excel); the one we'll use is called `IPython` (short for *interactive Python*).[11] When you install Python on your computer, what you install is a Python interpreter and the Python standard library.
- The ***Python standard library*** is a collection of Python packages[12] that come with the Python interpreter. You can think of packages as extensions to the main application, which is the Python interpreter

10: In fact, Python code is just text that follows certain formatting rules. Python code files are plain text files.

11: IPython enables the style of interactive computing we'll be using throughout the book.

12: More details on what exactly these packages are in the following chapters.

(just like browser extensions or Excel add-ins). Most of the standard library packages are developed and maintained by the same community that develops the Python language. Python's standard library is commonly referenced as one of its main strengths; it provides packages that are useful for various tasks, such as creating graphical user interfaces, working with zip files, and many others.

Python isn't just any other programming language: it is *the* most popular programming language in the world right now.[13] This popularity means that there's a lot of community support around the language: lots of Q&A content on the web related to Python, many tutorials, and plenty of books on how to use Python in different domains. One of these domains, where it has taken over, is data analysis: Python is now the most popular programming language for data science, machine learning, or scientific computing, making it an excellent tool for accounting.

13: In 2020, according to the Popularity of Programming Language Index, which ranks programming languages based on how often people search for tutorials on Google.

Python is designed for code readability first: no unnecessary punctuation, no curly brackets, and English words instead of operators wherever possible (e.g., `and` instead of `&&`). Because of its emphasis on readability, it's easy to learn, easy to write, and easy on your fingers as well (for example, Python code is typically $3-5\times$ shorter than Java[14] code). Even if you haven't had any exposure to Python, you can already understand it, though you might not be comfortable expressing yourself with it yet:

14: Java is a programming language developed by the Oracle Corporation.

```python
def multiply_by_two(number):
    return number * 2

multiply_by_two(10)
```

```
20
```

A more familiar example might be the side-by-side comparison between two expressions of the same function, in `VBA`[15] and Python, shown in listing 1.1 on the following page. You can see there how Python is designed to be readable and concise, in contrast to `VBA`, which has many confusing keywords and operators.

15: Visual Basic for Applications or `VBA` is the programming language integrated in Microsoft Office applications. As of 2020, it is *"the most dreaded"* programming language according to a Stack Overflow developer survey.

Python is free to use and runs on almost any computer you can think of (including your smartphone). The source code is entirely open, and the language was designed from the ground up around principles of openness (unlike, for example, Java, which is managed by Oracle). Python's creators describe it as *"powerful... and fast; plays well with others; runs everywhere; is friendly and easy to learn; is Open"*, which covers most of the reasons behind its popularity.

```vba
Function Extract_Number_from_Text(Phrase
    As String) As Double
Dim Length_of_String As Integer
Dim Current_Pos As Integer
Dim Temp As String
Length_of_String = Len(Phrase)
Temp = ""
For Current_Pos = 1 To Length_of_String
If (Mid(Phrase, Current_Pos, 1) = "-")
    Then
  Temp = Temp & Mid(Phrase, Current_Pos,
    1)
End If
If (Mid(Phrase, Current_Pos, 1) = ".")
    Then
 Temp = Temp & Mid(Phrase, Current_Pos,
    1)
End If
If (IsNumeric(Mid(Phrase, Current_Pos, 1)
    )) = True Then
    Temp = Temp & Mid(Phrase, Current_Pos
    , 1)
 End If
Next Current_Pos
If Len(Temp) = 0 Then
    Extract_Number_from_Text = 0
Else
    Extract_Number_from_Text = CDbl(Temp)
End If
End Function
```

```python
def extract_number_from_text(phrase):
    return float(
        "".join([
            c for c in phrase
                if c.isdigit() or
                   c in ["-", "."]
            ]) or 0
        )
```

Listing 1.1: A function that extracts a decimal number from some text given as input. The left version is written in VBA, the right version is in Python. Both examples are incomplete but illustrate the differences in readability between VBA and Python. VBA example is taken from www.automateexcel.com.

Why Python for accounting

For someone working in accounting right now, having to source data from various places (e.g., *SAP* or some other corporate database) and glue it together in Excel, learning Python should be extremely compelling. Just copy-pasting data from several large Excel files into a new spreadsheet is bound to generate a lot of frustration. And if you need to repeat this operation every few days, your only option is to do it all over again, manually. In Python, you can do this in a few lines of code, and once written, you can reuse the code repeatedly.

Using Python to glue data this way is a feature, not a side-effect. Python is designed for gluing things together (i.e., datasets, systems, software components), and this is one of the reasons it's suited for accounting. Through its libraries[16] Python already works with many of the tools used in accounting today (e.g., QuickBooks, Tableau, Excel, etc.) and can be used to glue them together.

16: I'll describe these tools and libraries more in the following section.

Besides gluing things together, Python is also great at data analysis. Python's ecosystem of tools has exploded over the past decade, both in the number of data analysis libraries and their adoption in different domains (including finance). These tools have been optimized for handling large data and can boost your accounting workflow because they:

- Work well with the software you already use (e.g., SAP, Excel);
- Allow you to handle data in explicit and easy-to-check ways;
- Work with large datasets that Excel can't even open;
- Enable you to automate repetitive manual tasks.

Switching to Python for handling data addresses many of Excel's drawbacks I mentioned earlier. Unlike point-and-click operations, data analysis code is always explicit, stays reproducible, and scales with your data. If this is not enough to get you excited about the rest of the book, consider that Microsoft has begun to embrace the world of open-source software and, in particular, the world of Python.[17] It's not unreasonable to imagine that in a few years, given its immense popularity, Python might replace `VBA` as the scripting language used in Excel.

17: Microsoft provides their very own introduction to Python course: docs.microsoft.com/en-us/learn/modules/intro-to-python. More to the point, the creator of Python started working for Microsoft in 2020.

In most analytics domains, accounting included, there's always some new tool to learn, which promises mythical productivity gains (e.g., Tableau, Looker, Alteryx, etc.) but ends up turning one kind of frustration into another. Python isn't one of these: it's a foundation on which you can build your own tools or glue those you're already using to make them work together.

A quick tour of Python's data tools

Whether in Excel or Python, handling data usually means reading, preparing, transforming, and visualizing tables (spreadsheets, `CSV` files, or any other kind of table). The easiest way to do that in Python is by using its data analysis libraries.

Python libraries are collections of code that allow you to run different operations on your data without writing all the code for those operations. They are almost always free to use and open-source (i.e., you can see the code they run, unlike, for example, Excel, which is closed-source).[18] Many are stored on github.com, a public repository of open-source libraries and projects (for various programming languages, not just Python).

18: They are usually developed by communities of programmers, engineers, scientists or enthusiasts who contribute their time and expertise because it benefits everyone (themselves and the wider community). You can contribute too!

To use libraries in your Python code, you need to install them on your computer (just like installing any other software) and import them into your Python code.

The main Python data analysis libraries you'll be using throughout this book are:

- **pandas** is *the* data analysis library for Python. It is designed for manipulating tabular data and the main tool you will use to work with Excel spreadsheets in Python. The `pandas` library aims to *"become the most powerful and flexible data analysis tool available in any (programming) language"*; given its widespread adoption, it has (arguably) already achieved this aim. `pandas` provides out-of-the-box code for reading, cleaning, and transforming data from various sources and is extremely fast even on large datasets (i.e., anything over 1 000 000 rows). A large part of this book is about using `pandas` together with Python (and other Python libraries) for accounting tasks.
- **NumPy** is a Python library for working with matrices and mathematical operations related to matrices.[19] There is a close link between `pandas` and `NumPy`: `pandas` uses the `NumPy` library extensively in its internal code. The `NumPy` library also fills in certain gaps in `pandas`'s functionality, which is why you'll often see them used together.
- **matplotlib** and **seaborn** are two of the most popular data visualization libraries used in Python-based data analysis. While `matplotlib` is very flexible and allows you to customize plots down to the smallest of details, `seaborn` is built on top of `matplotlib` to generate complex plots quickly, often in one line of code. Even though `matplotlib` and `seaborn` cover most plotting use-cases (and are far more versatile than Excel), many other Python data visualization libraries are available online for you to explore.

19: `NumPy` is a Python library for working with multi-dimensional arrays, not just matrices, efficiently. You can think of a table as a matrix of sorts, which is why `NumPy` can help when working with spreadsheets.

We'll cover how to install and use libraries in your Python code in the following chapters, but here is a quick example of how you can import the `pandas` Python library and read data from an Excel file using the `read_excel` function[20] (which is part of `pandas`):

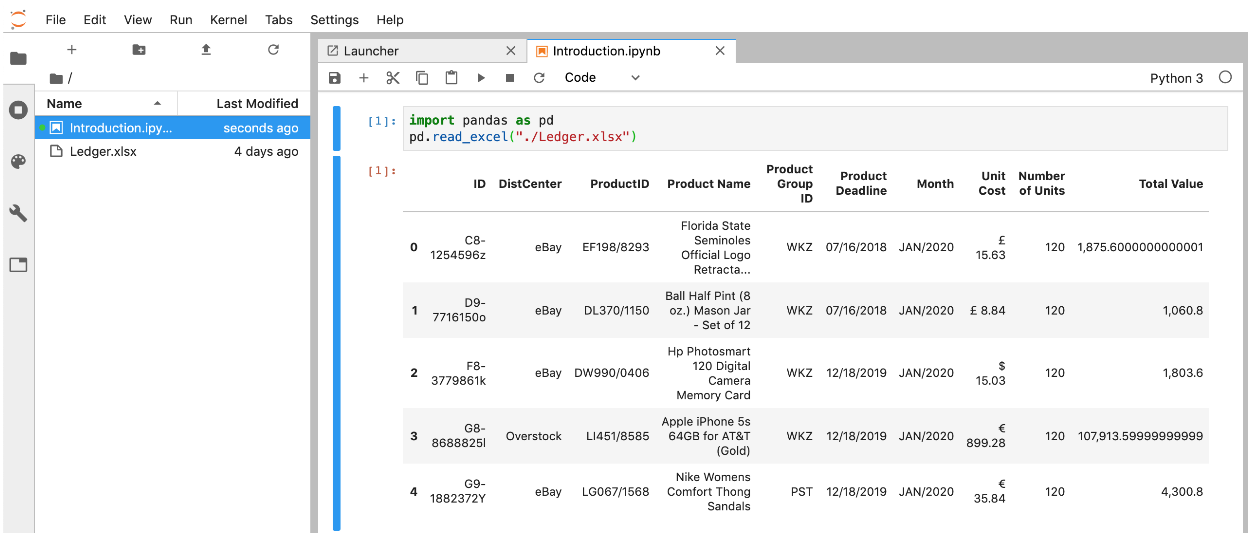
20: We'll also review functions in the next chapter.

```python
import pandas as pd

pd.read_excel('Q1Sales.xlsx')
```

Some of the libraries mentioned above depend on one another in fairly technical ways.[21] Getting more clarity on how they work together and which one does what exactly requires using them for a while. For now, to make some sense of these libraries, you can view them as black-boxes: collections of code that you feed data to (in the form of tables), and they give you different data back (in the form of a pivot table, a plot or just a single number). The next chapter will guide you through setting up these libraries on your computer (which is far easier than it might seem right now).

21: And non-technical ways: `pandas`, `matplotlib`, and `NumPy` (as well as others) are all part of the *SciPy ecosystem*, a collection of open-source software for scientific computing in Python. The term *SciPy* is also used to refer to the community of people who use and develop this software, a conference dedicated to scientific computing in Python, and an actual Python library called `SciPy`. Don't worry if you're confused; I'll go through the parts you actually need to know.



```
[1]: import pandas as pd
     pd.read_excel("./Ledger.xlsx")
```

[1]:

	ID	DistCenter	ProductID	Product Name	Product Group ID	Product Deadline	Month	Unit Cost	Number of Units	Total Value
0	C8-1254596z	eBay	EF198/8293	Florida State Seminoles Official Logo Retracta...	WKZ	07/16/2018	JAN/2020	£ 15.63	120	1,875.6000000000001
1	D9-7716150o	eBay	DL370/1150	Ball Half Pint (8 oz.) Mason Jar - Set of 12	WKZ	07/16/2018	JAN/2020	£ 8.84	120	1,060.8
2	F8-3779861k	eBay	DW990/0406	Hp Photosmart 120 Digital Camera Memory Card	WKZ	12/18/2019	JAN/2020	$ 15.03	120	1,803.6
3	G8-8688825l	Overstock	LI451/8585	Apple iPhone 5s 64GB for AT&T (Gold)	WKZ	12/18/2019	JAN/2020	€ 899.28	120	107,913.59999999999
4	G9-1882372Y	eBay	LG067/1568	Nike Womens Comfort Thong Sandals	PST	12/18/2019	JAN/2020	€ 35.84	120	4,300.8

Figure 1.2: A screenshot of *JupyterLab*, the program you'll be using to write Python code. This example shows how to use `pandas` to read data from an Excel file, and what a table looks like in JupyterLab.

Two other tools — which are not Python libraries — will be essential in your Python adventure:

- **`JupyterLab`** is the software you'll use to write code. What Excel is to spreadsheets, JupyterLab is to data analysis files, which are called Jupyter notebooks (i.e., JupyterLab lets you create and edit Jupyter notebooks). Notebooks are designed to make coding by trial-and-error easy: you can run different pieces of Python code in a notebook and see what they do without having to design an entire program around them. Notebooks also allow you to mix different media types in the same document: code, tables, plots, images, and plain text. As you'll soon see, they're powerful tools for data analysis (and for learning). Just like the Python libraries I mentioned earlier, JupyterLab is open-source and free to use. The following chapter guides you through installing JupyterLab on your computer, but for now, you can see a screenshot of JupyterLab in figure 1.2.
- **`Anaconda`**[22] is an installer that bundles all the open-source Python libraries and tools I mentioned so far into a single package that you can use to get up-and-running with Python data analysis code fast. What the Microsoft Office suite is to Excel, Anaconda is to the Python libraries for data analysis you'll be using throughout this book (except it's free to use and open-source).

22: Or the *Anaconda Distribution*.

These are just a few of Python's libraries and tools for data analysis but are perhaps the most relevant for accounting. There are many others online, and you'll soon discover them on your own.

How to use this book

This book has a lot of code in it, shown directly in the main text. Try to read code examples, but also type them out by yourself in JupyterLab. You learn much faster when you type code by yourself, and you also get used to the mechanics of writing and running code (which will serve you well when you finish the book and have to navigate without a guide).

All code examples in the book look like this:

```
In [1]:  print('hello')                                  1
         print('python for accounting!')                 2
```

```
Out [1]: hello
         python for accounting!
```

The gray box at the top shows one or more Python *code statements* — the code here just prints some text, but all code examples will be similar in style to this one. When more than one line of code is listed in a code box, you'll see line numbers on the right of the box to make it easier to reference a particular line in the main text (e.g., line `1` prints a hello message).

The gray box code is what you need to run to produce the result shown in the blue box beneath it, which is the *code output*. Sometimes code doesn't produce any output (e.g., when you create a new variable), so there's no output box for that code. How exactly you run code and what the `In [1]:` and `Out [1]:` labels mean is covered in detail in the next chapter.

For the data analysis code you'll be writing, code output will often be a table. Output tables look like this:

```
In [2]:  import pandas as pd                              1
                                                          2
         pd.read_excel('Q1Sales.xlsx')                    3
```

```
Out [2]:       InvoiceNo         Channel        Product Name  ... Unit Price Quantity   Total
         0          1532      Shoppe.com  Cannon Water Bom...  ...      20.11       14  281.54
         1          1533         Walcart  LEGO Ninja Turtl...  ...       6.70        1    6.70
         2          1534        Bullseye                 NaN  ...      11.67        5   58.35
         3          1535        Bullseye  Transformers Age...  ...      13.46        6   80.76
         4          1535        Bullseye  Transformers Age...  ...      13.46        6   80.76
         ...         ...             ...                 ...  ...        ...      ...     ...
         14049     15581        Bullseye  AC Adapter/Power...  ...      28.72        8  229.76
         14050     15582        Bullseye  Cisco Systems Gi...  ...      33.39        1   33.39
         14051     15583  Understock.com  Philips AJ3116M/...  ...       4.18        1    4.18
         14052     15584        iBay.com                 NaN  ...       4.78       25  119.50
         14053     15585  Understock.com  Sirius Satellite...  ...      33.16        2   66.32

         [14054 rows x 12 columns]
```

The table above has 14 054 rows and 12 columns (you'll have to believe me), but you only see 10 of its rows and 6 of its columns (and an indicator at the bottom left that tells you how many rows and columns there are). I show tables in the book in this truncated form, with `...` instead of actual rows or columns, to make them fit in the main text, and because tables in JupyterLab also get truncated like this — at first, you might find this annoying. You'll soon discover you can work with data even without seeing it all.

You'll be working with several accounting datasets, and for some of the chapters ahead, you'll have to download extra files. Each chapter has an associated web page (i.e., that's where the link in the bottom right margin below takes you) where you can download additional resources; more on getting set up in the next chapter.

Learning to code is like learning a new language: first, you learn the alphabet (i.e., what the different Python keywords are), then you learn common words and phrases (i.e., how to use Python with `pandas` and other Python libraries), and then you try to express yourself fluently (i.e., code an entire data analysis project). Just like learning a new language, you need practice to become good at coding: go through each example in the book, type it out by yourself, and you'll be fluent in Python before you know it.

Roadmap

There are four parts to this book:

- **Part one: Python ABCs** introduces the building blocks of the Python programming language (i.e., how to define variables or functions, how to use lists and loops, and a few others). These building blocks are the foundation on which the data analysis tools used later in the book are built — and are also the glue that makes them all work together.
- **Part two: Working with tables** is where the rubber meets the road in using Python for accounting. This part of the book introduces the main features of `pandas`, the Python library you'll be using to work with tables — whether Excel spreadsheets, `CSV` files or any other kind of tabular data.
- **Part three: Visualizing data** shows you how to turn data into plots using some of Python's data visualization libraries (`matplotlib`, `seaborn`, and `hvplot`).
- **Part four: Sales analysis project** guides you through an end-to-end data analysis project that uses Python, Jupyter notebooks, `pandas`, and a few other Python libraries. The project looks at a wholesale supplier's sales data and tries to identify which products

and sales channels are most profitable. This type of analysis is common in management accounting; even if your work doesn't involve analyzing sales or inventory, you'll find many ideas in this part that can help you structure your own data analysis projects. This last part brings together many of the tools you will have learned about and covers the areas of a typical analysis project that aren't related to code: setting and documenting specific questions for the project, organizing data and code files, finding answers, and sharing results.

The first three parts of the book have several chapters that introduce new programming ideas. These chapters are short and focused, and all build on top of each other; ideally you should try to read them one after the other. There are also five project chapters that don't introduce new programming ideas but guide you through applying what you've been learning in an accounting setting. These chapters will walk you through (in order): filtering and splitting a large Excel file into multiple sheets, reading and cleaning a general ledger exported from QuickBooks, mining product reviews, filling missing values in a sales dataset, and making a waterfall plot from a cash flow statement. The last part of the book is a sales analysis project in its entirety.

Throughout the book, you'll find several sections and even a few chapters whose titles start with the word *"Overthinking"*. You can ignore these overthinking sections and chapters if you want to because they cover ideas that aren't strictly necessary in your learning journey. Still, you might find them interesting if you get bitten by the Python bug.

Good luck, and have fun!

Thank you for reading the first chapter!

If you want to learn more about using Python in accounting, head to www.pythonforaccounting.com and get your full copy of the book now.