Course Overview

Hi, my name's Terry Toy. Welcome to Getting Started with Data Analysis using Python. I've been programming for over 15 years, and Python is a great language to learn. So if you're an Excel power user, or perhaps new to data analysis, I welcome you to this course. In this course, we'll learn not only about Python, but specifically how to apply it to data analysis. We'll learn how to collect, clean, persist, analyze, and how to create stunning data visualizations. The core of this course is analyzing data, and we'll learn about data structures such as lists, dictionaries, tuples, and sets, and we'll apply these structures to market analysis, because there's so much information about the stock market. And regarding to persisting data, we're going to learn how to work with CSV files and SQLite, a relational database. We'll learn how to insert and collect data, and store it all within a database, then how to retrieve that data, whether from a CSV file or a SQLite database, and to build our data structures, our lists, our dictionaries, our tuples, and then use that to analyze our data, and then build stunning visualizations. So I'm excited; I hope you'll join me in this course. Let's learn about Python and data analysis.

Introduction to Python Data Analysis

Course Introduction

Hello, and welcome to Getting Started with Data Analysis Using Python. My name's Terry Toy, and I'm excited to introduce this course, which bridges software development and data analysis. To learn more about data analysis, we'll be using stock market data, since there's just a ubiquitous amount information regarding stocks. So first you might ask who this course is for? So if you're a developer new to data analysis, this is a great course for you, or if you're new to Python. Similarly, if you are an Excel expert and you're new to programming, this course is great for you because we'll assume that you know nothing about programming and we'll start with the basics. As a means to learn more about data analysis, we'll be studying stock market data since there's such a ubiquitous amount of stock market data available. In addition, we'll be building tools which may be useful to individual investors. Now we'll be using Yahoo's API calls to gather stock market data, and we'll demonstrate that in the next module. Now as a course overview, we'll start with What is Data Analysis? And we'll start specifically by learning how to collect data. In our case, we'll be using Yahoo API calls to gather stock market data, and we'll start by cleaning our data. That's the first thing we need to do is filter out data that's not applicable and learn how to filter or to focus on the core value data that we want. And then we can learn how to persist that data, whether that's to text files or whether that's to a database, we can then store it, and later on retrieve that data from our data store. We'll learn how to analyze data using data structures such as dictionaries, and lists, and tuples, and sets, and that's going to be the core focus of this course, learning those data structures and how to intermix them. And then finally, we'll learn how to visualize data. We'll be using different libraries available from Python. With data visualization, we can spot major trends, and understand greater insights into our data. We're going to be using Python, because Python is one of the excellent tools for data analysis. Not only is it a full-fledged software development platform, but it's great for data analysis, or mini libraries that we can use to help us analyze and collect that data, and better understand our data, and data analysis, or data science, is one of the fastest fields growing in software development. Now one of the cores of analyzing data is understanding basic data structures. We have, for example, lists. Now lists is one of the most flexible items in Python, and we can use it to append or pop, or we can use it to index and sort our data. A list is denoted in Python by these square brackets. Another data structure we'll be using in this course are dictionaries. Dictionaries are key value pairs where each key must be unique, a common use of dictionaries or counting the occurrence of unique words in a document. Dictionaries in Python are denoted with these curly brackets. Similarly, we have tuples. Tuples are a sequence of immutable Python objects, and they are denoted by these parentheses. We can use tuples to store records because records have a finite number of elements, such as first name, last name, birthdate, and address. In this case, we can reference them by index, such as the 0, first, second, and third element, accordingly. Finally, we have a set. A set is denoted by the keyword set, and also parentheses. A set are unordered collection of unique and immutable objects, and a common use in Python is for membership testing. Finally, we'll learn about data visualization. And data visualization allows us to present our data in stunning ways, using stunning graphics so that we can gather insight into the trends and the value of our data. And we'll learn, we'll be using pygal, and you can learn more at pygal. org. So let's go ahead and get started with a demo. We're going to start with the basics. First, we're going to learn how to install Python. We'll be using Python 2. 7. Next, we'll learn how to use the command line or the terminal, and we'll be using basic command line commands such as cd. Cd allows us to change the directory to move up and down different directories. We'll learn mkdir to create a folder. We'll be using the text editor, IDLE, which comes with Python, or you can use whatever text editor you feel most comfortable with. And then finally, we'll learn how to run or execute our Python scripts from the command line. So let's go ahead and get started. We'll be, we've looked at a course overview of who this course is for and why, and now we're looking at how to install and set up our Python, or download our Python, and then how to execute those scripts using the command line. So let's go ahead and get started, and look at a quick demo and learn about these features of Python and how to get started.

Installing and Executing Python Scripts

Now that we understand our course objectives, let's focus on a demo. And in this demo, we want to accomplish a couple of tasks. First, we want to learn how to install Python and set up our development environment. And secondly, we want to learn about the command line, and how to use IDLE, which is our integrated development environment for this course. Visit python. org to download and install Python software. Now with the Mac, the OS, and Linux, Python is already installed, so I'm going to download it and install it here on a Windows machine. Go ahead and set the defaults, and we'll download this into a folder called Python 2. 7 because we'll be downloading the Python 2. 7 version. Now for Windows, I'm going to explicitly set the path so that the environmental variables are set accordingly, and we can run Python scripts from the command line. Once we've downloaded and installed Python, we should continue with all the defaults, and then we'll be ready to go and write some code. Before we can write some code, we need to learn the basics of command lines, and we start by going to the Start icon in Windows, and we type in cmd, and this launches a new command window. And within here, we can type in cd c:, and that's going to bring us to our C Drive. That's where we downloaded and installed our Python 2. 7. In this case, I'm already on my C Drive, so nothing changes. In this case, I'm going to type in cd\, and that's going to bring us to our root directory of C. And now I can type in cd python27. I'm going to navigate to this directory that we just created where we downloaded and installed our software, and here I can do a dir, d-i-r, and see all of the files within that directory. Now I'm going to create a new subdirectory, mkdir, m-k-d-i-r, space, I'm going to call it mycode. I'm going to put all of our code within this folder, or within this directory called mycode. So next let's launch IDLE, and we can go to the same Start icon window, and type in i-d-l-e, and that's going to launch our IDE development environment, and this brings up a prompt where we can test our installation of Python to make sure it's working. Let's try some simple commands here, such as 2 + 3, and we can see the results returned back to us. And similarly, let's do a print statement, print, and let's do print hello and see the value come back to us when we hit Return. So our Python is set up; it's running well. And the next idea is to create a file. So let's create a new program. We're going to create a script here and actually call it hello from a py file, or actually that's what the command is going to do. It's going to print that value out. And we're going to save this in the directory we just created, mycode, and I'm going to name this hello. py. Make sure you put the. py because we need to save the extension, and essentially we're telling the compiler, or the operating system, that we have a Python file. So once we save it, here's how we execute it from the command window here. We can cd into our folder mycode, and if we do a dir here we can see that our file is created, our hello. py file, and to execute it, we simply type python, and then we type in hello. py, the name of the file that we created, and now when we hit Return, we actually run that script, and the results, in this case the print statement, is returned right there in our command window. So that's a brief introduction to IDLE, to the command line, and how we can create our first scripts, how we can write some code, and execute it within our command window. As a brief summary, recall you can always go to the Windows Start button here, and then you can type cmd to launch a command window, or i-d-l-e to launch the IDLE interface, or integrated development environment. In the next clip, we'll give a brief summary, then move onto our next module and learn how to collect stock market data.

Summary

We are learning about data analysis using Python, and as a summary of our first module, we started by learning who this course is for. So whether you're a programmer and you're new Python, or whether you are an expert in Excel and you want to learn programming, this is a great bridge course between software development and data analysis. We downloaded and installed Python, and in this case, we're working with Python version 2. 7. Now once we install and get Python running, we learned about the command line or the terminal, and how to execute some basic commands within the command line, such as cd, mkdir, and then how to execute our scripts. We're going to studying stock market data since there's just an abundance of available stock market data, and it's easy to collect and gather and apply those or build those into our data structures, which, we'll be using dictionaries, and lists, and tuples, and sets. We'll also learn about data visualization using pygal. org, where you can learn more about the different data visualizations available within, and how we can use those, and tap those, and build those through Python. So let's go ahead and get started. In the next module, we're going to be focusing on how to gather stock market data and how to filter and clean that data. We'll be gathering our data using Yahoo APIs, finance APIs, and then collect that data, and learning how to basically read and persist that data within to text files.

Collecting Stock Market Data

Introduction to Collecting Data

We are continuing our course with data analysis and Python, and we come to this module where we will be collecting stock market data. Now whether we're collecting and cleaning data, we're going to apply these concepts to whatever our application may be. In our case, we're going to start off by learning about the Yahoo API. This will allow us to collect stock market data from Yahoo in real time. Now unfortunately, halfway through this course, the Yahoo API has been temporarily disabled, and I'm not sure if it's coming back, so we're going learn a more generic approach. We'll use Google Finance and actually download CSV files and then import it into our code. Now it's not as versatile as a Yahoo API, but nonetheless, as data analysts or data scientists, we had to be aware that whenever we're using other APIs, these APIs may be deprecated or changed, and we need to update our code accordingly. So in this course, you'll see examples of the Yahoo API, as well as Google Finance. Then we'll learn about cleaning data, and cleaning data is filtering out the data that is waste or that is inaccurate and focusing on the core data. And then we'll be learning about Python, writing some code, and learning to persist that data into files, whether that's text files or CSV files. Now when you're collecting data, there are many concepts that we can apply to. We can gather our data from APIs, in our case. In other cases, we may have web scrapers that go out and gather the HTML and parse it and gather the data that we need. We may be getting the data internally or from a business partner in a CSV file format, and we can bring that in, that data accordingly. Or we may be getting it from a database or a data warehouse using SQL commands to gather that data. Or we may be getting it from sensors, which involve analog converted to digital, and then we can process it in the Python world once we have it in digital. So the idea with cleaning data is we want to take the data, and anything that's not coming across appropriately or that will throw our data off, we want to cleanse that and remove that. In our case, we'll be looking at the Yahoo Finance and using a GET command. Now this the Yahoo Finance example. This is a URL, and if you type this URL in, we can actually extract or gather in real-time data stock market data based on Yahoo. Now Yahoo has limitations on its use, so it can't be for commercial purposes, but notice in the URL there's different values. We have, for example, the code, or the stock ticker value here, and that stock ticker value so we can change it to whatever stock we're looking at, and we also have a period. In this case, we're looking at a year's worth of data. So let's go ahead and type that URL in, and we'll do a demo so that we can understand the data that we're getting. Then we can write some code that can navigate this URL, and we can filter out the header information because there's some header information there we don't want to need, and we want to focus on the data. And then we can split each line as we're importing the data. And then finally, at the end, we'll manage that data by extracting it to writing and reading from CSV files or text files. So let's go ahead and get started with our demo.

Demo: IEX Trading Platform Data

Welcome back to Introduction to Data Analysis Using Python, and we are in our lesson where we're collecting data, and we're focusing on our demo. And the first thing we want to do is understand our data, so we'll go ahead and look at the data that we've downloaded from the IEX trading platform. And then we'll write some code. And the first thing we'll do is we'll parse the data, just get more familiar with the data, select the key fields that we want to study and gather, and then parse the file name, the stock symbol within the file, and then in the next demo we'll persist the data. So these are the couple of the URLs we looked at in the last module, or in the last section, and we want to focus on the IEX trading platform, so let's go ahead and type in that URL here. And what we can do is we will download two stocks. We'll focus first on Apple, and we'll download one year's worth of data, and then we'll do again with Ford. I'll be changing the symbol here from aapl, which is Apple's symbol, to f, which is the Ford symbol, and let's go ahead and download. So we've just essentially downloaded two years' worth of two stocks, and I encourage you to do more stocks just to explore the information. Now let's go ahead and open up the file, and what we can see is we get a lot of data here. We get the date, the open, the high price, the low price, the close price, the volume. And we get much more, so you can explore the fields that you want to focus on. And we're going to focus on a few, and these are zero-based, so we're going to focus, as we get into the Python code, we're going to focus on the first field, which is date, and we'll get the open field, we'll get the close, and we'll get the volume. We're just going to grab a handful of these, but you can focus on the fields that you feel are most applicable. Next, let's look at some baseline code for reading data from files, and first we want to import the CSV file, and then we're going to open the file that, and this is what it's named, what we downloaded, the IEX\_stock with the symbol, the chart, the 1y. csv. And we're going to be opening it up in read mode. We're going to store all of our data within a reader variable. We can also have headers because we do have headers within this column. We can print those headers. And then we're going to iterate through the reader, and we're really going to only focus on four fields, the date, the open, the close, and the volume, and those are in the 0, the 1, the 4, and the 5 position relative. Finally, we increment our counter, we print the counter at the very end, and then we close the file. So let's go ahead and run this code. Now it's important to note that the file that we just downloaded needs to be a JSON or in the same folder as this code, and so let's go ahead and type python iex. py. That's the file we've named it. And when we run it, we can actually see that we get a year's data worth for Apple stock, and we can see all the headers on top, and then we can see a year's worth of data, and it goes all the way from 2017 to 2018, and it goes from all the stock prices. So we can actually look at the data a little bit closer to verify that what we're getting. But this is pretty basic, and let's try to do something a little more interesting. Let's create a list of stocks, and in this case, we have two. We have Apple and Ford and aapl and f for Ford. And then we can iterate through all of these two stocks. And I encourage you to even do more than two. And what we could do is now we can create a for loop, and let's go ahead and indent this entire block of code. And when we indent this entire block of code, what we're going to do now is go through and go through all of our stocks. In our case, we've downloaded stocks for two, for Apple and for Ford. So we have this file name, and the file name is very consistent, and when we download it, so we can now just really create a string with the appropriate file name and change stk to the appropriate variable. So in the first case, it'll be Apple. In the second case, it'll be Ford. So this will go through all of our files. If we have 20 stocks or 30 stocks, it doesn't really matter. And now we can also print the stock name. Now notice in the whole file there never appeared Apple or Ford, and it just had the dates, and it had the open and the close, the appropriate data. So now I can actually store the stock symbol. In this case, we install, initially when we ran the code it had Apple, and now it has Ford in the bottom, and that allows us just to do more analysis and to store everything into one file. And we're calling that data file myData, or we will do that in the next demo as we output it. So in this case, we haven't persisted the data yet. We're only taking the data, opening up the CSV files and processing the data to the screen, and what we're going to do next is persist the data in the next demo.

Demo: Writing Data to Files

Welcome back to Getting Started with Data Analysis Using Python, and in the last demo we downloaded from the IEX trading website. We downloaded stock information for two stocks, and it contained a year's worth of data. Now we're going to take that data and write it to a CSV file using Python. And so the basic template for code for writing data to files is this: We're going to import the CSV. Then we're going to open our file, and it's going to be a CSV file. And in this case, we're going to focus on write, and we'll name our file, abbreviate it as f so we can refer to it below. We're going to write our core code here, and then we're going to have a write method. That's going to be actual data. Instead of writing it to the screen, we'll be writing it to the file. And then we close our file. Let's look at the code we wrote in the last demo, and what we made is an option for us to enter a couple of stocks. We're going to go ahead and take that data from those few stocks, we're going to open them as files, and then we're going to read through them and print them out to the screen. And now instead, we're going to do that exact same thing, but this time we're going to write them to a file. So let's cut and paste our opening line, and this is the line that opens our myData. csv file in write mode, and we're going to reference this file as f in our code below. So we can indent all of this other code. This is the code that goes through the CSV files which we downloaded. It has all the stock data. And the output of this is going to be myData. csv. And so what we need to do then is instead of printing to the screen, we want to do an f. write, and we're going to write all of the information before. And as before, we're storing four bits of information, and you can choose the information that you feel is relevant, but we'll be focusing on the date, the open, the close, and the volume. So this will work, and it'll basically store and output everything to a file, but there's a couple things we need to do. Number one is a simple way. There are other ways with a CSV file, but we only have a few lines here. A simple way of actually creating a CSV file is just to concatenate everything with commas in between because what we want is a comma-delineated file, so we're going to have the stock, then we're going have a comma, and then the next data field, then a comma, next data field all the way until we have the date, the open, the close, and the volume and also the stock price. It starts with the stock price. And then finally, at the very end we want to print done just to know that we've gone through everything. We've opened up all the CSV files, we read all the information, we got a year's worth of data for however many stocks we're processing, and we've outputted everything into a myData. csv file. And the final thing I need to do at the end is to add a new line here because otherwise everything is just going to be on one line, and we want to have one line per stock entry per day. When we execute the code, we get 252 for each stock, and we have 2 stocks, and then we get the done message to show that we're done. So let's go ahead and open the myData file which was created in that same directory, and we see all of our output. First we have all of the Apple stock, and we have this for the last year. We have the key elements, our data elements that we, the open, the close, and the volume. And then we have all of the Ford. Now that we've collected and compiled the data, we can do all kinds of analysis. For example, we can look at which days had the highest trading volume, or we can compare the performance of one stock against the other. So to recap, in our first demo, we used Python to clean data and then to collect it and open up the CSV files, and then in this demo, we took that data, after we've compiled it, and wrote to a new CSV file called myData. csv. And you can open these CSV files with Excel or Microsoft or any type of spreadsheet application, or you can even open them up with Notepad.

Summary

In this module, we learned about collecting data. We first learned and focused on collecting data from a couple of data sources. First, we used the Yahoo APIs, and then we learned how to read and write data into text files or in CSV files. Next, we learned about cleaning data. In our case, that was relatively simple. We just wanted to filter out the header information and focus on the data from the Yahoo APIs. And then we learned about persisting data into text files. We learned about a snippet of code and how we can open up a text file, or a CSV file, or a log file, and then also opening it for writing, or reading, or pending, or appropriate. In the next section, we're going to learn about data structures, which is the heart of this course. We're going to learn about lists, dictionaries, tuples, and sets, and we're going to learn how we can intermix these and build powerful tools of analyzing our data. And then in the section after that, or the module after that, we'll be looking at persisting data again, but this time, instead of to text files, we'll be persisting data into a database.

Understanding Python Data Structures

Introdution

Hello, and welcome back to Getting Started with Data Analysis Using Python. We're starting our next module, Introduction to Python Data Structures. Now data structures are a powerful element or a key element in any data analysis. My name's Terry Toy, and let's get started. In this module, we're going to focus on data structures, four of them in particular. We're going to start with lists, and lists are annotated with these square brackets. Then we're going to go dictionaries. Dictionaries use these curly brackets. Tuples use these parentheses. And finally, we'll look at sets. Set uses the keyword set followed by parentheses. These are the four data structures. Each one has different uses, and different strengths and weaknesses. With each data structure, we'll learn first the core or key concepts of these data structures, we'll give a basic demo, which we'll walk through some of the core methods and different ways we can use code, and then we'll apply these data structures to our stock analysis application, which we're building throughout this course. So that's how we'll get started, and it will be the structure for each data structure. We'll start with lists. Lists are the most versatile and widely used data structure in Python. And the core idea is that lists are a collection of items. And lists can store all kinds of data, complex data, and can be nested in many ways. We can test for membership with many of these data structures, and with lists and many others we'll use the keywords in and not in. There are many methods for lists, some of them are append or length, insert, pop, and sort. For our stock market analysis of lists, we'll actually go through a year's worth of data and find those days in which the stock price changed by more than 2%. Next, we'll look at dictionaries. Dictionaries, the key core concept with dictionaries is key value pairs. A dictionary is a, consists of a key, and a key could be any type of structure. We'll start with very simple keys, such as a stock price symbol, and the values could be anything. In our case, we'll start with something simple with the name. Now further, going further along, we'll develop more complex ideas. Now some of the basic methods with dictionaries, they include signing and updating, deleting, iterating through the keys and the values, and also finding the length. Constant look up time is a key concept with dictionaries because once we create a dictionary, we can access any element using a constant time. And for our stock market analysis, we'll join data. We'll create a lookup table in a dictionary and be able to access any of those elements quickly with the structure of a dictionary. Next, we'll turn to tuples. Tuples can also be pronounced tuples, either way, but I'll be using tuples in this course. The core idea is that a tuple is an immutable data object. It has a set number of fields, and, for example, if it has three or four, it always will have that number of items, and you cannot add or subtract from a tuple. Some of the key methods with a tuple include length, max, min, and splice. We can also use tuples to model records, and that's essentially what we'll be doing for our stock market analysis. We'll model a record or the fields in a database. The same idea, it will represent, for example, first name or last name and birthdate, if we were looking at students, but for our stock market, we'll look at the symbol, perhaps the date, and the closing price. Finally, we'll look at sets, and we'll look at the key methods from sets, and these include adding or discarding or updating. We can also have union and intersection with sets. But the key concept with sets is unique membership. And for our stock market analysis, we'll find a set of unique stock symbols as we iterate our watch lists, our open and close positions. So in this module, we'll learn about data structures. Specifically, we'll learn about lists, dictionaries, tuples, and sets, and we'll apply them to our stock market analysis application.

Lists

One of the most versatile data structures in Python is the list, and the core idea with a list, it's a collection of items, and you can store very complex items within a Python list. And within these lists, you can easily test for membership in or not in, you can test for the length, you can append or add an item quickly, you can insert it into a specific position, or you could pop or remove an item from any position within the list. And you can also sort lists very easily. In this clip, we'll learn first the basics about the Python lists, and then we'll apply it to our stock application. And we'll filter out the data, looking at how much the data has changed each day over the last year, and focusing on those stocks that have changed more than 2%. As we begin our introduction into Python data structures, we'll start with the basics, and we'll start looking at the Python data structure known as a list because a list is certainly the most versatile and commonly used data structure. A list is essentially a collection of items, and in Python, a list can store almost any type of element. So let's go ahead and create IDLE, or let's go ahead and start IDLE, which is a Python command line, and do some basics with lists. Now some of the common methods we'll look at are append, insert, but we'll start with initializing, and we'll start with a variable called myList, and we'll initialize it to square brackets or an empty list, and if we print the values out, we can see that we have, the square brackets are no items within the list. We can hard code items, for example, we're going to create myList, and create a few stocks, and we'll put Ford and Google into this. And we can print myList, and we have the appropriate two values, Ford and Google. Now we can also append items. Now if we want to programmatically, instead of hard coding items, we can append an item. And we can append it, by appending it we will automatically put it at the end of the list. And in this case, I'll add Apple, AAPL, and append it to the end of the list, and let's go ahead and print myList, and we can see now the three values with AAPL at the end. Now if we want to insert a value, and insert it in a specific place, we can use 0 base index, and 0 base index meaning it's going to insert it just before the place we specify. In this case, we're going to put it just before the first element, or after the 0 element, so it's going to go between Ford and Google, and we're going to insert Qualcomm, or QCOM, that stock value. And now we can print myList, and we can see Qualcomm was inserted right after Ford, and we now have four items in our list. Another common method is sort, and x, in this case, x = sorted, and myList in parentheses is going to actually create a brand-new list, and it's just going to be sorted. And by default it will sort it alphabetically. So if I can print this new list, x, we can now see it's alphabetical with Apple, Ford, Google, and Qualcomm sorted alphabetically. Now we can also pop items, and that's the idea of removing it. And it uses the same idea as insert. We can pop a specific item. In this case, I'm going to pop, if you don't put any value in, it's going to be the most recent or the last value. In this case, I'm going to put the first, again it's 0 base, and it tells you that we have popped, or removed, Ford from the list. And now we are down to three items in our list called x. Remember, we created a copy of it when we did the sorted, when we did it the sorted method. So another idea with lists is the test of membership idea, and this uses two keywords. It's either going to be in. In means it contains or the item is within the list, and we can use not in, which means the item is not in the list. So here we're going to test for membership, and we're going to see if the stock symbol GM is in my stocks. And if so, it's going to print True. Now in this case, GM is not in the list, so nothing is displayed back. Let's test the other thing, the other case. Now if GM is not in my stocks, in this case it will print False, and we'll get a positive identification that it is not in the list, and when we run this case we get the value False. Let's go ahead and get an affirmative case in which something is in membership, and let's check for Ford. If Ford is in myStocks, then we're going to print true, and we go ahead and execute this statement, and we get the value true because Ford is within the four-stock list. So that's a basic introduction into membership and not membership using the in and not in keywords. Let's go ahead and see these exact same commands in a Python script rather than in the Python command line. And if we go ahead and execute these scripts, we see the exact same values, we can see a list, we can see the append method, the pop method, the sorted, creating a new list, and the test for membership within that list. And we get the values accordingly. Next, let's start with a demo, and let's apply the same methods that we've done and just learned about, and apply them to a real case scenario involving our stock application. And what we're going to do is we're going to iterate through the Yahoo API and look for those stocks that we're investigating, and see only those times in which the stock price had changed by more than 2%. So remember, let's take a quick refresher on this Yahoo API, and remember the values coming back. We have the date, we have the close, the high, the low, and the open. And we're really focused on the close and the open. Now the close is going to be in the first position, and the open will be in the fourth position. And we're going to extract those values from the API, do a simple calculation, and test whether it's more than 2% or not, and only display those values. So the first thing I want to do is start with the code we looked at at the last module, and then we'll remove all elements in regards to writing to a file, because I don't need to write to a file. So I'm taking these three lines of code, I'm removing them. That's when we were going through and looking at writing the data to the file, and I'm going to dedent, or move this one clip back, and now we're back to our base code where we go through and look at the Yahoo Finance API, and we're going to look at our stocks, and we're going to get every value for the last year because that's what the API is returning. And to simplify it, I'm going to start by looking at just one stock, Apple. I'm going to create a list, eventually I'm going to populate or append to this list, all of the values, all the times when it has changed more than 2%, and I'm going to append that and store those in this list called 2%. But before I want to append it to the list, I just want to see if I can print it to the screen to test the algorithm to see if it's working accordingly. So here within our data, this is within our clean data again, and we're going to create three, or two variables, or three variables, eventually. One is going to be cls, which I'll call that as a close, and remember close was in the first position of the splitLine, and the open, which I'll call opn, was in the fourth position, and so that's going to be splitLine 4. Again, this is the values coming back from the Yahoo API, and we're extracting the close and open position. So I'll create a variable called change, c-h-g, and the change will represent the close minus the open divided by the original, or the open position, and that will give us the change. And in this case, we're only going to be looking for the change if the change in the positive direction changed by more than 0. 02, or 2%. And in this case, we're going to print out the values because we just want to test to see if the algorithm is working. Now eventually, we're going to take the values and insert them in this 2% list that we've created, which represents all stock prices that have changed by more than 2%. In this case, we're going to print out the screen, the stock symbol, and we're going to concatenate it with change. Now eventually I'll update this so that we include the date because right now that's not in there, but this is just the first pass to see if we can get the algorithm working. Now I can't store, all of the values coming back from the Yahoo API are strings, and I can't do calculations with strings, so I need to convert them to floats. So I'll convert the close and open variables to floats, and then get that value, a change will be a float, but I need to convert that back to a string when I'm printing it out to the screen. So let's go ahead and run this code and see if we can just show, instead of a year's worth of data for Apple stock, if I can see something much smaller, just the times that Apple has changed by more than 2%. And let's go ahead and execute this code, and we get back the following values, and we can see that it changed to more than 2% approximately 4 times, so instead of a whole year's worth of data, we're just getting a few concentrated times in which it meets the terms of our algorithm, which is a change by more than 2%. And so we can add another item, another stock in here very easily. We can add Ford in, and do the same type of comparison. And let's go ahead and run the same code again, and we can see that Ford changed substantially more times, maybe 8, 12 times, it increased by more than 2%. Now eventually when I clean up this code I'll add in the dates so we'll get a better feel for not only the change values, but the dates. Instead of just printing out the screen the stock symbol and the percent change, now I'm going to append the values to the list, and then eventually I'm going to iterate through this list that we have and print the values out at the final end. And so I'm going to create this list, we've already created and initialized this list called 2%, and I'm going to pin the values called the stock symbol, and concatenate it with a space, as well as concatenate it with the string value of the change. And so we can see how much everything changed, and instead of count showing all of the counts for the entire year, I'm going to go ahead and indent this value, and just show the counts of which the stock changed by 2% or more, or more than 2%. And on the very end, I'm going to print the count, and then we'll go ahead and iterate this list that we've created. And as we iterate it through the entire stock API, one year's worth of data, and now append it, the appropriate values to the 2% list, we're just now going to iterate through this new list called 2%, and print out the values. Let's go ahead and run this code, and we can see the number of times that both Apple and Ford changed by 2%. I can also change this by adding another stock, and we can simply just put in our top list, and we're going add Microsoft to this list, and then once we do that, we can run this one more time just to see that it works. We're storing all these values again in a list called 2%. And then finally, at the very end, iterating through, and just displaying those values. We run this code, and we get the same thing, and we can see that twice Microsoft changed. So I'm going to amend this code one final time and add in the dates, and then we can run the code again. To add in the date, it's very simple. When we're appending the values to our 2% list, we merely need to splitLine and take the 0 position because the 0 position has the date. And remember, the date was stored in as year, year, year, year, month, month, day, day. And so it's those values that will be printed out or appended to the list, concatenated with the stock symbol, and then concatenated with the change, our screen variation of the change. And let's go ahead and execute this code, and we can see, as we run it, we can now see the dates that these stocks are changing more than 2%, then we see the stock itself, and then we see the percentage changed. So let's focus just on the Microsoft values. We can see here on the fifth month, on May 16th, it changed by more than 2%, and also in December we can see another date in which it changed by 2%. So we're gaining some tools by using these lists. One final step is to sort the 2% list we just created. So instead of seeing the changes first with the Apple, Ford, and then Microsoft stock, we'll see the changes sorted based on the date. And when we run the code, we get the following. And here we can see all of the changes happening at different times, starting with the most distant, and then ending on the bottom of the page with the most recent. So lists give us powerful tools to collect data, and then later to sort them. In this lesson, we learned about lists. We learned about all the basic methods such as sorting, inserting, testing for membership, and popping items off the list. The key thing about lists is lists are general containers, and they can store very complex objects. In the next section, we're going to learn about dictionaries, then we're going to learn about tuples and sets, and then finally, we'll learn how to incorporate all these data structures together.

Dictionaries

Another common data structure is dictionaries, and dictionaries allow us to essentially store key value pairs. And key value pairs, in our case, are going to be the symbols and the stock names. We can assign or update values very simply with dictionaries. We can also test for membership, whether it is in or not in a dictionary. We can delete items in a dictionary, and another core method is being able to iterate, and we can display both the key and the values. We can look at the length of the dictionary. And the key value, or the key idea with dictionaries is it's a constant look up time. So once you load a dictionary, you could easily find any element within that dictionary at a constant time, it's relatively quick. It is very efficient for joining data, and that's what we'll be doing in this lesson is joining data from two different data sources. So we're going to start off with the basics and go over some of those basic methods we just talked about, how to assign, and initialize, and update different items within a dictionary. So let's go ahead and open up IDLE, and once we open up IDLE and we're at the command line, the Python command line, let's initialize and create a basic dictionary. So here myDict =, and we use these curly brackets to indicate that we're initializing a dictionary. And when we print it out, initially we don't have any items. We can go ahead and create some items, or assign some items, and here I'm going to create a symbol using our stock metaphor. We're going to have a symbol called ABC, and that's going to represent a company called ABC. So here we have key value pairs, in this case they're both strings. But again, dictionaries can very complex, and as we get further along, we'll be nesting data structures together. For example, a dictionary of tuples with values, might be tuples, or sets or lists or a combination thereof. So here I'm assigning some additional values. I have three elements in this dictionary, and they are basically representing three stock symbols, ABC, XYZ, and ZZZ, and here we have the corresponding companies for those three values. Now if I print the dictionary, we're going to see all three values here. We can see them in key value pairs, first we see the key, and then the corresponding value, and we see that for all three elements within the dictionary. Now the next thing we can do also is we can delete and item, and we simply use d-e-l, and then myDict, and then we put in the key. So we are deleting this one entry within the dictionary, and now if we were to print the dictionary, instead of having three items, we will only have two items. So in addition to adding and deleting items, we can also update items. Really, just by updating items, it's just reassigning the key. So in this case, we're going to take the key, XYZ, and we're going to reassign it to a new value. Now if I added a different value, if I added a brand-new key, it is essentially adding that item to it, and in this case, I'm updating it because that key already exists within the dictionary. So I'm going to call this Updated Company XYZ, and now we can print out the items, and we can see the two values here for our two elements within the dictionary. Now other ideas with dictionaries is we can test for membership, and we use the in and not in. And so in this case, I can just use an if statement. If XYZ in the dictionary, then I can print a value of True. And in this case, it's going to test for membership. Again, we can use in or not in, and if we go ahead and execute it, in this case we can see that it is true because XYZ, that key is in the dictionary, and that's what we're testing for is the keys. Next, I'm going to test for membership in one line. So I'll use the print statement, and in one line I'm going to test if this value, ABC, this key, is not in the dictionary, and I'm going to explicitly use the method. keys because later on we're going to use the method. values as we iterate through and get different, look at different parts of the dictionaries, the key and the value. And here we can see False because ABC is not in the dictionary. That's what it's asking, and in this case it is, and that's why it returns false. So similarly, we can check for values. I'm going to test for if Company V is in myDict. values, and in this case, the value stores all the company names. And there is no Company V, so we can, again, put the print statement in here, and in one line, not only test the keys, but also test the values, or membership within the keys and the values, and we get the appropriate response. Next, we're going to look at iterating through dictionaries, and when we iterate through dictionaries we can print out both the keys and the values. And so let's look at for x in myDict. keys. We're going to print x, so in this case we're going to go through all the keys, and we can print out the two companies, XYZ and ABC, and those are the methods. And similarly, we can iterate and look at the values for x in myDict. values, and here we're going to iterate and print x, and we are going to get all of the values. And here we get the company names, and those are the values. Similarly, we can iterate and look at both by doing something like the following, for x in myDict, and we're going to print x, and x is going to default to the keys. And if you want to get the values, you just look at the lookup value of the dictionary, and that's myDict, and in square brackets, you put in the key, and then you get the company name. So that's the idea with dictionaries. They are key value pairs, and we can easily iterate through them and look at both the keys and the values, and we can test for membership inside and out. So now we're going to turn out attention to a demo and try to do something a little more complex, something a little more real. And we're going to continue with our stock metaphor, and before we do that, we're going to have to get some data, some new data. Previously we looked at Yahoo Finance, but now I'm going to go to NASDAQ. I'm going to go to this website here, and we're going to download all of the stocks here. We can do different exchanges, NASDAQ, New York Stock Exchange, or American Stock Exchange. In this case, we're going to get the New York Stock Exchange, download all the data. It's in a file called companylist and it's. csv. And here if we open up the file we can see, that's a lot of information. It has the symbols on the far left and the company names, and it has the closing price, the last sale, the market cap. And you can see in the header all the different types of information. We're really just going to focus on the first two elements here. As we learn about tuples, we'll get more elements, but we're going to focus on the first two elements, the symbol and the name, and those are the first two elements in the data, those are the first two elements within the file. Now let's look at some code. In this snippet, we're going to open explicitly a CSV file, so we're going to import csv, we're going to open up the file that we just downloaded, and we're going to store that in a variable called dataFile. We're going to open it up in read mode only, and then we're going to the reader within, the method with the CSV files, and then we can just iterate through this entire file, and we can print out all of the data. And in this case, we have quite a bit of data within the file. Let's go ahead and run this script, python csv-reader. py, and let's go ahead and execute it. We can see we have a lot of data. So just to get a better understanding of the data, let's go ahead and add a few more lines of code. Let's add a count and let's break once we reach 10. And let's focus on 2 elements, the first 2 columns, data 0 and data 1, and here we can see the top 10 lines, and we get first the symbol, and then we get the stock name. So what we're going to do now is to take this snippet and load it within a dictionary. Let's initialize our dictionary. We're going to call it myDict, and I'm going to initialize this to 0, and now I'm going to do the actual assignment, or the creation of the elements of adding elements to the dictionary as we iterate through the CSV file. And so I'm going to do myDict, and in the square brackets here I'm designating the key, and then I'm designating the value. Next, I added three lines of code. First, I'm going to get the length of the dictionary, and then iterate both through and print out the symbol and the values, and we run that, we get all of this list of companies. And the next step we can do is to try to integrate this within, and use it as a lookup table with a master list of stocks that we're interested in. So the first thing I'm going to do is I'm going to take all of this code that returns a dictionary, and I'm going to turn it into a function, and it's going to return a dictionary, so I'm going to take this block of code and indent it. And the output of this code is going to be the dictionary that we've created. And I'm later going to use that, and call this as a lookup table to get the names of some symbols that I'm interested in. So I'm going to create a list of stocks, and these are stocks that I'm interested in. And to make things very simple, I'm just going to list a few stocks here that I know are within the New York Stock Exchange. So I'm going to list IBM, Ford, and Disney, and these three stocks I'm going to look up. Here I have the symbols, and I'm going to use the dictionary that I've created that has over 3, 000 elements, and I'm going to look up what are the names of these 3 symbols. And in order to do that, I need to create an instance here of this function of this getStockDict. So I'll call that myDict, and I'll call this function that we just created getStockDict, and it's going to return this dictionary of 3, 000 stock elements again with key values pairs. And once I have that, I can use that as a lookup. And so now as I'm iterating my list of stocks, here I have three stocks, if x is in my dictionary, so if that key, that symbol is within this dictionary, then I'm going to print out this symbol value. Or in this case, I'm just going to print out the value. When we run this, we get the three companies, not the symbols here, we get the actual names of the companies because we've joined this list here, and we've joined it with this dictionary that we've created with this CSV file we downloaded. Now I'm going to add Microsoft, and I know that Microsoft is not within the New York Stock Exchange. It's on the NASDAQ. So when I run this, I don't get any difference. I just get the three companies, and there is nothing found for Microsoft, so I'm going to put an else in here. Else, if it's not found, it's not in the dictionary, then I'm going to print not found. And in this case, as I iterate through, I'm going to get the appropriate, I'm going to get the three stock names that I do know, and then not found for Microsoft, which is not within the New York Stock Exchange. In this clip, we learned about dictionaries, and we looked at all the basic methods that you can do with dictionaries, adding an item, updating it, removing an item, testing for membership, and also iterating through both the keys and the values. And we learned one of the core concepts of a dictionary is that a dictionary has constant look up time. And we used that in our stock application. We downloaded a large file, loaded a dictionary, and then we were able to quickly look up the company names based on the symbols that we were interested in. In the next section, we'll be looking at tuples and sets, and then we'll be learning how to nest all of these data structure together.

Tuples

Now that we've learned about lists and dictionaries, we turn our attention to tuples. And the core idea with tuples is tuples are immutable, that is, they store a fixed number of elements. So because they are immutable, common uses are storing fields in a database, so we store, essentially, a collection of records. And we're going to do that, illustrate that, both in the basics and our demo. Now when you want to access a specific data element, you access it by position starting with 0. We'll demonstrate this, as well. And some common functions with tuples is length, max, min, and splice. So we can show a sub-range of positions within the tuple. We can also test for membership. We can look for whether it's in and not in. So let's go ahead and get started with some basics. We'll look, we'll pull up the command line for Python, and go ahead and test some basic commands with tuples. Here I have 3 tuples, 1, 2, and 3, and they store each 3 elements, first with the stock symbol, the company name, and then the closing price. So I can print my tuple, and I'll get the corresponding record with the three elements. I can also print just a specific element within the tuple. In this case, I'm going to start with a 0 base, and I get the stock symbol for the second tuple. I can also print a range, in this case, splicing out the data. I can go from 0 to 1 element, and that's only going to give me one data element, and as we'll see, ZZZ here, which is the better sleep company, and then similarly, I can print out a range here. So I'm going to look at myTup, the third one, and go from 0 to the second position, and now it's going to print out both the symbol and the company name. Other functions available for tuples include the length. So I can print the length of my tuple. In this case, I have three elements, accordingly. And I can also find the min or the max, so I'm going to create a new tuple. I'm going to call it myNum, and I'm going to assign some basic numerical values here. And once we have a tuple with some numerical values, or string values, we can search for, or we can check for the min or the max. In this case, I'm going to print the minimum of myNum, and I get the corresponding value. One other feature about tuples is we can test for membership. So I'm going to check if 7 is in myNum, and in this case it's false because 7 is not a member, or it's not one of the data elements within the tuple. Finally, we can iterate, or we can create nested elements with our data structure, so I'm going to use both lists and dictionaries in our demo. Let's do a quick example with lists. I'm going to create a brand-new list, and I'm going to use the square brackets here, and then I'm going to append three elements to my list. I'm going to append first the myTup1, and then myTup2, and then finally, myTup3. Now once we have a list and we append the elements, we essentially have a list of tuples. So this really common to have a list of database structures, or database records, or in this case tuples can represent database records. And once we have that, we can now print it out, we can see the three tuples within myList, and we can also iterate my list for x in myList, and we can go ahead and print each tuple out. And we'll do that by printing out x and we get the corresponding values. The 3 records are all of the tuples, and if we want to see just a specific data element within here, we can do that as well by using the 0 position. In this case, I'll iterate for x in myList, and I'll see only the final element, the second element. And we get accordingly just the closing prices for those three stocks. So now we're starting to nest data elements and getting closer. In our demo, we'll continue this idea. We'll use tuples, and we'll nest them inside lists and inside dictionaries. For our demo, we'll return to our Yahoo Finance application, and remember, this data is the API for Yahoo, and we get these elements back where we get the date, the close, the high, the low, the open, and the volume. And we're going to take some of this key data we want and store it into a tuple, and then we're going to have first a list of tuples, and then we're going to build a dictionary similarly off it. So let's go ahead and start first with our list. And here's the code we looked at previously, and this goes out and looks at the Yahoo API and grabs the data back, it looks for only the data we want. I've copied the data elements here because it's nice to have that handy for reference, and we're going to store these values within a tuple. So let's create a list. I'm going to call it stockList, and now I'm going to append it. As I loop through, as I'm going through the Yahoo API, I'm going to create a tuple, and then append that tuple to my stock list, so I'll have a list of tuples. Now one of the values I'm going to store within the tuple, well, let's choose different elements here, let's go ahead and we'll use a splitLine because these are going to be the data elements. And I'm going to choose, first I'm going to cut and paste and copy these out, and then I'm going to choose the 0, the first and the fifth. So the 0 position is going to be the date, and the first one is going to be the close, and the last one's going to be the volume. So here we can see the 0, the 1, and the 5. Now I'm going to take my stockList, and I'm going to append my tuple to the list. So as we iterate through the Yahoo API, we build a list of tuples, and the tuples consist of three data elements, first the date, the close, and the volume for the Apple stock. So now I can print the length of this list, and we get 250, or approximately 250 records, which is the closing price for Apple for the entire year. Let's go ahead and open up our command line, run the code, and we can see 252 days' worth of data within the stockList. So we have here a list of tuples. And we can do something similar. Let's get a little bit more complex and use, first, let's iterate through and see all of the elements within the stockList, and then we're going to actually build a dictionary. So we can actually go through, and not just see that we have 252, but actually see the elements, and we can see each of the 3 elements within the tuple. We have basically a list of tuples. Each tuple has three elements. And let's go ahead now and focus on a dictionary, and try to build something a little bit more complex. Remember, a dictionary is key value pairs, and in this case, our key is going to be a tuple. So I'm going to change this to a dictionary, and instead of appending to a dictionary, we assign values, so I'm going to do the stockDict, and then within that I'm going to create each entry with a key, and that key is going to be a tuple. So let's go ahead and rename this myTupleKey. And the tuple is going to consist of the stock. In this case, as we iterate through the first time, it's going to be Apple. We're going to add additional stocks. And the tuple is really going to be two elements. It's going to be the stock symbol and the date. The date is represented in the 0 position of the splitLine. So that's going to be the key of the tuple, and the value is going to be the closing price. So here the splitLine, the closing price, is the first position within our Yahoo API. Once we build the dictionary, we can get the closing price for Apple stock for any day. So in this case, we're looking at 20170412, or April 12th. When we run the script, we get the following, and we can see the closing price. Next let's add additional stocks. So we built a dictionary for Apple and we can easily add additional stocks, in this case we'll add another stock for Microsoft. And as we iterate through we're now going to hit the Yahoo API twice, load dictionary, which would be larger, and not only include the last year's worth for Apple stock, but also include Microsoft stock. And so when we print out, we can choose a couple different days for Apple, and we'll also choose a couple different days for Microsoft. So in this case, we're looking at 2017, April 12th, and let's go ahead and look at the beginning of the year for Microsoft and for Apple. Let's look at January 3rd, the beginning of the year. So we can see now, we can compare prices over several months, or any two days within the year's worth of data that we have, and instantly pull those closing values. Let's go ahead and run this script and see the results. We have the closing price of Microsoft and Apple stock over those two dates. In this lesson, we learned about tuples. We learned all the basic methods. Tuples are data elements, and they start in the 0 position. And the key concept is tuples are immutable, so they're ideal for storing database records or properties of an object. In this chapter, we looked at creating lists of tuples, and we also created a dictionary where the key was a tuple. Next, we'll learn about sets.

Sets

In this module, we started by learning about lists, then dictionaries, and tuples, and we finally learn about sets. Now set is initialized using the set word, then we can add and discard items. These are some of the key methods. We can update items. We can also test for union and intersection. We can test for membership using the in and not in keywords. And the key concept with sets is unique membership. When we iterate through a set, we're always going to have only one copy of any one element. There will never be any duplicates. Let's open our Python command line and test some basic commands in regards to set. Now the first line, s1 = set(), this is the initialization of a set. So s1, we're going to add an element. We're going to add the element 1, and then we're going to update 1, 2, and 3. So in this case, remember, sets can only have a unique membership. Notice 1 is initially added in the second line, and then also added in the third line, updated, so when we print s1 and we see all the elements, it's just 1, 2, and 3. Any time we have a duplicate, the set is, that element of the set is not added. So I can add 4, and now print s1 again, and I have 4 elements, 1, 2, 3, and 4. They are all unique, as always. Now set 2, let's initialize another set, and we're going to, in this case, initialize it with some values. And I'm going to put in some values here, 4, 5, 6. Now we have 3 elements in set 2, and we have 4 elements in set 1, and let's go ahead and print s2 so we can verify that we have the elements as we expect. We can use union and intersection to join these, and we'll do that using s1. union(s2), and then we can see the resulting elements. It's all 1, 2, 3, 4, 5, and 6. Now notice 4 is duplicated in both set 1 and set 2, and it's not included in the union because it can only, there can be no duplicates. Similarly, we can look at the intersection, and there's only 1 element that represents the intersection between set 1 and set 2. We can also test for membership with sets, and we do that simply with the in and not in command or keywords. In this case, I'm going to test 7 in set 1, and that is false. And similarly, I can test for 3, is that in set 1, and that case is true. So sets are good for testing for unique membership. So let's go ahead and take a look at a demo. In our case, we're going to have a few stock collections, and we're going to loop through those, or iterate through those, and find them to be unique. We have a watchlist, and that consists of some stock symbols that we're interested in. Then we have openPos, and that represents open positions. Now I've commented out on the far right all of the elements of the tuple. The first element of the tuple is the symbol, then we have the buy price, and then we have the number of shares. Next, we have closedPos, and that is another tuple. The first element of the tuple is a symbol, followed by the buy price, the sell price, and the number of shares. So let's create a set called myStocks, and myStocks will be a unique set of symbols that we're interested in from all of our positions including the watchlist, the open positions, and the closed positions. So in order to get this set of stocks, I'll iterate through each of these lists. So for x in watchlist, I'm going to now add myStocks. Now similarly, I can iterate through my open and closed positions. For x in open positions, in this case, it's a tuple that I'm iterating through, and I'm only interested in the symbols, so I'm going to choose, or add to the set, x in the 0 position, because that represents the symbol. Similarly, for the closed positions, I'm going to do the same thing. I'm going to have a tuple with 4 elements, but the only element that I'm interested in is the first one because that's where the symbol is, so myStocks. add, and then x, and I'm looking at the 0 position. So when I iterate through all three of these lists, a set is automatically excluding any duplicates. And now if I print myStocks and execute this code, we can see a unique set of stocks based on after we've iterated through all of these lists, and we get the unique set. Now in this case, I have an error, and I can see that closedPos is not defined. So I look at line 4, and I can see that I've forgotten my equal sign, and that's causing the error. So if I correct this, just by adding the equal sign, I can rerun the script, and then I should have a unique set of stocks in the set called myStocks. And when I run it, I do get Intel, Ford, Apple, Qualcomm, Microsoft, and Disney, and notice that all the duplicates have been removed. In this lesson, we learned about sets, and the only one key concept with sets is sets involve unique membership, and duplicates are automatically removed.

Summary

In this module, we learned about data structures. We started by learning about the list. The list is the most versatile data structure with Python, and a list is annotated by these square brackets. A list can store complex data. In our case, we used it to analyze a year's worth of data, and to show only those times in which a stock price changed more than 2%. The next data structure we learned about are dictionaries, and dictionaries are annotated by these curly brackets. And the key idea with dictionaries is they are a constant look up time. Dictionaries are key value pairs, and the value could be any type of data structure. We can have nested data structures, and similarly, the key can be a data structure, as well. We started with something simple in which the key was the stock symbol, and the value was the name of the company. The next data structure we looked at was tuples, and tuples are annotated with these parentheses. And tuples are excellent for storing records. Any time we have a set number of fields, a tuple is excellent for storing that data. To access a specific element within a tuple, we used the position, and we start with 0. Next, we learned about sets. Sets are annotated by the keyword set and parentheses. And the unique idea about sets is we have unique membership. Any time we have duplicates, they are immediately, or automatically removed from a set. And in our case, we use sets to iterate through a watch list, in open and closed positions, and to define a unique set of stocks within those elements, and to remove all duplicates. Sets can also be used to find a union and intersection between data sets. Now that we've learned about data structures, the next step in learning about data analysis is to learn how to persist data. We'll learn how to take all of the information we've gathered from Yahoo APIs and insert them into a database. We'll also learn how to retrieve this data from the database, and then build these data structures, not only from web APIs, but from our persisted data.

Persisting Data in Databases and Files

Introduction

Welcome back to our introduction to data analysis using Python. In this module, we're going to focus on Persisting Data. Persisting data is the ability to store data, and then to recall it and use it over again. Common methods are with a database, but we're also going to look at files, and in a particular case, of pickle files. So in this module, we're going to focus mostly on the database, and we'll be using sqlite3. SQLite is a file-based database system, and it's very powerful, and we use all the basic SQL commands to access it. We'll also learn about files, and specifically pickle files, as well as how to organize our data in data objects. So the first thing we're going to learn about, or some of the core topics we're going to learn about in this module include, first we're going to learn about sqlite3. That's the database engine we'll be using, it's readily available with Python, and it's very powerful. We'll learn about the DB Browser for SQLite. This is a graphical user interface to the SQLite. It allows us to quickly create tables, update data, view data, and many other powerful things, all in a graphical user interface. We'll also learn about CRUD, that's Create, Read, Update, and Delete. And we'll use Python code to manage the data. We'll create basic code in Python to execute SQL commands. Pickle files are another way to persist data. In this case, we're persisting complex or serialized data structures, so the dictionaries, lists, tuples that we learned about previously, we can actually store that as a pickle file, rather than extracting the data or retrieving the data and rebuilding those structures. We'll learn about data objects, which is another way to organize our data. We'll also look at the SQLite command-line, which is another tool instead of the graphical user interface to access the data stored within our SQLite database. Finally, to apply context to these topics, we'll build out our stock market application. So let's go ahead and get started. We're going to start with SQLite. This is our database engine, and one of the fundamental ways of persisting data.

DB Browser for Sqlite

In this module, we're learning how to persist data into a database. So we'll start by learning an introduction into SQL, which is Structured Query Language, and it's used to get information from a database. Some of the database operations we'll focus on is first, CRUD, which is creating, reading, updating, and deleting data from a database. We'll use SQL commands to do this. We'll also learn ORDER BY, which allows us to take our output records and order them in a specific manner. We'll learn about a WHERE clause, which allows us to filter data, and then LIKE, which is patterned recognition. With LIKE, we can return only those records that match a specific pattern. Let's start by visiting the following website, sqlitebrowser. org. At this website we'll find DB Browser for SQLite. Go ahead and download the appropriate version based on your operating system. They have versions for Windows, Linux, and Macintosh. Let's start with a demo of the DB Browser application we just downloaded. This application allows us a graphical user interface to work with SQLite. We'll learn how to use the graphical user interface, as well as SQL commands. So the first thing we want to do is create a new database. Now when we create a new database, we need to know where we're going to store our file, because SQLite is a file-based database. So in this case, I'm going to call it accordingly, sample. db, and we're going to save it, and note the place where we're saving it, or remember. And the first thing we can do is create a new table, and it allows us a default to that option, but we can go ahead and create it manually here with the CREATE TABLE button. And I'm going to create a table called users, and the users is going to store information about our users. And some of the fields are going to be email, last, and first, corresponding to their email address, the last name, and the first name. Now notice the different data types available. There is, for example, TEXT, INTEGER, BLOB, REEL, AND NUMERIC. In our case, we're going to be storing text information based on the email address, the last name, and the first name. Once we get to the stock market application, we'll be storing numeric data. But these are the different data types. Also, notice on the bottom, there's an ability to create this entire table using a script. Now we're going to learn about that more later. In this case, we're using a graphical interface to create a specific table. Now once we store, or once we've defined our table, we can write our changes. We're going to have to do that every time we're working with a graphical user interface. Next, let's go ahead and browse our data, and we don't have any data yet, so I'm going to go ahead and manually add some data. I've created three simple records, and once we've entered them in the user interface, make sure we write our changes. Now we can go back and browse our data, and we can see and confirm that all of the data is safe and persisted into the database. The next thing is we can actually use SQL commands, and we can use this Execute SQL pane here, and execute these commands, and verify our data. We can do it several ways, inserting, selecting, updating, and deleting. Let's go ahead and insert the data now, similarly as we did before, but instead of using the graphical interface, we're going to use a SQL command. And the insert command is insert into, the table name, which is users, and then we enter the values. And in this case, we have three values that we need to enter, and we need to enter them accordingly to the order of which the fields are located in the database. And in this case, we have the email address first, then the last name, and the first name. So in this case, I'll create a record, and we can see on the bottom that the record was successfully inserted into the database. Now if we go and browse the database, we can actually see this additional record has been entered. Notice the inserted name, jim kirk, is all lowercase. I'll be correcting that shortly using a SQL script and also through the graphical interface. But first, let's go the SQL pane and do some selects where we can pull back different records from the database. To retrieve records from the database, we use the select command, and select \*, that's a wildcard, from users, the table name, allows us to return all of the records in this table. In this case, I get all four records back. Now I can also order by any specific field. In this case, I'm going to order by last name, and now all the fields are returned, or all the records are returned sorted, in this case, by last name alphabetically. Now I used star as a wildcard figure, but it's better to explicitly point out the columns we want to return. In this case, I'm going to return two columns, email and last, and I'm going to add the third column, so now I'm going to explicitly point out the fields I want to return, email, last, and first, and I get all of the records ordered by last name, in this case. Now I can also add a where clause, and a where clause allows me to filter on specific records. In this case, I'm going to filter on last, greater than, a specific letter. So I'm only going to return those records in which it's greater than an F, in this case, and in this case I see that one name, Chan, has dropped off, and I've only gotten the remaining records that are all starting with something greater than F. So selects allow us to pull retrieved data back, we can order it in a specific way, and we can also filter using the where clause. Next, as we alluded to previously, let's update the lowercase letters on the last name for Kirk. In this case, I'm going to update users, that's the table name, and I'm going to set last=Kirk with a capital K here where the email=, accordingly, jim@mail. com. Now this where clause, whatever record applies to that, all of those records will be updated. In this case, it's only one record, and we can now see and verify, as we browse the data, that the name is now uppercase. Now similarly, I can use the graphical interface. In this case, I'm going to simply update it here in the graphical interface, and again, I'm going to have to write my changes back. Whenever I'm using the graphical interface, I need to write changes in order to persist them back to the database. And we can see that both names were updated accordingly. Let's go ahead and select, just to verify, and let's go ahead and select and pull back all of the records, and we can see that Jim Kirk, now everything is updated in his name. The LIKE operator allows us to filter on records based on pattern recognition, so in this case, I'm going to say where email like. And I'm going to use a couple wildcard figures. The percent sign means 0 or more. And in this case, it has to match this pattern where we have mail somewhere in the email address. And in this case, it's going to be all records, because all records match this pattern, and we get the return of all four records. Now if I want to use an underscore, that's for one character only, but we're going to focus mostly on the percent sign because that's more common. So if I want to filter just on a specific, I'm going to use sall with a percent sign, and in this case, the only returned record that matches this pattern is the one record, Sally, because it is like, or has that pattern with s-a-l-l, and then the percent sign means 1 or 0 or many records that follow that pattern. So we can use like to return only specific records from a field. Finally, we're going to look at delete. In this case, I'm going to delete from users the where clause, so if I don't use the where clause, I'll be deleting all of the records, in that case, and I don't want to do that. In this case, I just want to delete one record where the first name=Tom. I'm going to go ahead and execute this SQL command, and now if we look at the data, we only have three records because we deleted the one record accordingly. Finally, I'm going to execute a SQL command that creates a table, and to do that, I'm going to copy this existing table, and now I can paste this into my Execute SQL pane, and I'm going to change and update the records somewhat. In this case, I'm going to create a table. I'm going to call it posts, and we're going to create some fields. Users create posts in this example, and we have the first field called title, and that's going to be of TEXT. We're going to use dateposted, that's also going to be TEXT, and finally, we're going to use the topic. That's going to be TEXT, as well. Now if we want to add another column, we can call this article because this is going to be the article. It's going to go under a specific topic. We're going to have the dateposted and we're going to have the title, and that's also going to be TEXT, as well. Notice the article is in red, and one simple way of correcting this is to copy one of the existing fields, and we can do that, and then paste it in, and we'll see it's going to work a little bit better. What the problem is, is the single quote doesn't match up exactly to what the user interface is expecting, so the easiest way to correct that is just to copy an existing field, and then paste this is in over article, and we'll update it accordingly with the new title called article, which is the column name. Now if we execute this command, we will see it being successful, and let's go ahead and execute it now, and we can see the success on the bottom that the table was created. If we go to our database structures now, we have two tables. We have users, which was created previously, and we have posts. Now this is the new table, and it has new fields accordingly. And if we wanted to, we could insert data just as we did before, but that's one thing that we want to learn is that we can use a graphical interface to create a table, or we can execute scripts and likewise create a table through SQL commands. In the next lesson, we are going to learn specifically how to work with this data using Python code.

SQL Python Basics

In this module, we're learning how to persist data, and one way to persist data is with a database, so we're going to focus specifically on how to access a database with Python code. Now previously, we looked at how to access the database through the DB Browser for SQLite. Now we're going to learn how to use code. We'll learn how to read or collect records from a database, how to insert records into a database, how to update and delete records, and then finally, how to insert many records. Let's look at some code. We start by importing the sqlite3 library. Next, we need to find the path to the database. Now with Windows, we would use syntax similar to shown here. For Linux, I have it commented out. We would use something like this. Now once we can identify the location of our database, we set up a cursor, and then we execute our SQL statement. Now the most simple statement is here, SELECT email FROM users. Remember, we have three columns here, email, last, and first, representing the email address, the last name, and the first name. Those are the three columns in the database, but in our case, our SQL statement is only going to be returning one of those elements, that's the email address. Now once we get the data back from the SQL, we can iterate through all of the rows, and then print out the 0 element, or the only element in this case, which is the email address. Let's go ahead and pull up our command line and execute this code. So to execute the code we type in python and the name of the script, it's getRecords. py, and we can see it's returning the email address for the three records that we entered previously using the DB Browser. So next we can fine tune this statement. Let's explicitly pull out the three columns, last name, first name, and email, and let's execute this, as well as order it by the last name, and we run this code. Nothing really changes except that we need to pull out all of the data, and not only the 0 element, the first 1 and the second 1. And this is very similar to a tuple, so we can have the row and it's going to be coming back with the three items from the database, and we can now run this code, and we get not only the email address, but also the last name and the first name. The next step is to add some parameters because sometimes we want to select data and we want to filter on a specific number of records. So in this case, we're going to say WHERE last name equals and a question mark. Question mark is an example of a parameter, and in this case we're going to call the parameter x, and we'll define x = Kirk. That's the last name that we'll be looking for. And we need the comma here just to separate because Kirk itself is a string, which can be represented by four letters, K-i-r-k, so the comma here makes it very clear that we're passing this as one parameter. And so now when we run this script, python getRecords, instead of returning all of the records, we're only going to return this one record where the WHERE clause is appropriately, we're filtering out on a specific record. We can also have multiple question marks, so I'm going to add OR last=, and put another question mark here. So in this case, we're expecting two parameters to be passed, and in this case I'm going to put Kirk and Chan. Now let's go ahead and run this, and in this case, we will get two records back. And so the idea is that we can have multiple parameters and pass it. We're going to using a question mark, and we're essentially passing all of those values through a variable, in this case x. We can also look at scalar values such as count or max. In this case, I'm going to do SELECT count of email, and we're going to return one number in this case. And when we execute this statement, we can adjust this, as well, so that we're only going to be one value in the tuple coming back, and it's going to be the 0 position, row 0. When we execute this code, we simply type python, then the script name, and we get three back, and that's the number of records in the database. Next, let's look at an insert statement, and similarly, as before, we'll use parameters to pass the variables that we're going to be inserting. In this case, I'll be inserting john@mail. com, and a first name, John Doe. And we pass the parameters through the question marks as before, and the key difference is we have a commit, and then we're going to close the record. Commit is the key idea with an INSERT, UPDATE, or DELETE. Let's run the code, and while this code inserts the record, it does not return any results, so we don't know what happened. So we can return to our old script, python getRecords, and see if our new record was inserted, and here we can see john@mail. com is the last record, and it has been inserted into the database. So let's change this one more time, and let's add more one record. We'll put in Jane, so Jane Doe will be the user that we're inserting into the database. And we're going to add one more thing where we can actually print out the rows affected of our execute statement. And in this case, it's going to be c, which is our cursor, and row count, and that's a property. And when we execute this code again, now we're inserting Jane, and we can see one record was inserted. Let's build an update statement. For SQL, we do Update, the table name, which is users, and we set first, which is the column name, equal to a variable because we're going to pass that variable in. And we're going to have another variable where email =, and this is going to be the specific record, or in some cases it could be records, that we update. In this case I'm going to change his name to Johnny instead of being John for his first name. And when we go ahead and run this code, we type in the script name and python. Instead of insert, it's going to be update, and when we run the script the code is executed. In this case, we're not printing anything out, but I can add that in as before, and we can say print r, which is, excuse me, print c, that's the cursor, and it's going to be rowcount, which is the property. And in this case we can see how many records were affected by this update statement, and it is one record. The delete is very similar to the update, so I'll just briefly review it. Instead of being update, the SQL statement is going to be delete from, the table name, where email equals a specific value. Again, we'll put in the print so we can see how many rows were affected by this SQL statement. And we will run the script python getRecords, first just to verify how many records we do have in the database. In this case we have the five records, and we're going to delete one record. We're going to delete the John, and we can see the one record was deleted. And if we pull our records back, we can see now we only have four statements, and the john@mail. com has been deleted. So update and delete are very similar. We can print out the number or rows affected. The key thing is we need to commit those changes to the database. Without the commit we won't see the changes saved, reflected, or persisted into the database. Finally, let's look at how to insert many records, and we do this by building a list. In this case, I'm going to insert three records for Jill, Frank, and Desi, all with the same last name, AppleTree. And in this case, we have a list of tuples, and the parameters that we're going to pass to our insert statement is going to be that entire list. So when we run this code, python insertmany records, we get an error because I've forgotten one thing. When we want to execute many records, we have to do a couple things. First of all, just for cleanup, I'm going to add a comma here because, as before, we want to have a comma at the end of all the tuples, and in this case, I need to insert many. So I'm going to change the method from execute to executemany. And when we run this script, we get the number of records that were affected, it is three, and we can now go ahead and view our records using our script getRecords, and we can see all of the data in there. The three new records have been added. Let's consider some course notes. When we're dealing with databases, we often deal with reference integrity, or data integrity, such as primary keys and foreign keys. We're really not going to focus on that in this course because we want to use our data structures for those tools. And similarly, when you're dealing with databases, you often want to deal with different types of joins. Again, we're going to use our tools, and the real power of this is we can join data that's in a file with data that's in a database, and they don't all have to be in a database where we can use the database engine. And so that's one concept as we get further along and we start persisting some data into files. We're going to have the tools to join data from diverse data types. We'll take data from the database. We'll take data from the files. We'll clean them, we'll merge them, we'll join them, and that's the power of data analysis. Next, we'll apply these tools to our stock market analysis.

Stock Market Application

In this module, we're learning how to persist data. Previously, we learned about the DB Browser for SQLite, and then we learned some basic Python code of how to work with database data. Now we're going to apply those both, merge those two ideas together, and do some simple stock market analysis. And we'll do this more in the next module, but let's get started with the basics. Now remember this screen? Previously, we looked at the Yahoo API and we could pull information about a stock company. In this case, we're looking at Ford. We have the values, which would be the date, the close, the high, the low, the open, and the volume. We should be familiar with those, and write those in order because we're going to create a database with those exact fields. I've opened up the DB Browser for SQLite, and we're going to create a new database called stocks. db. And we're going to create a new table. The table is going to be called stockprices. Now eventually we're going to add all of the fields in that we just talked about, those six fields that we talked about and just labeled that we're going to be getting from the API. They'll include, as listed here, date, close, high, low, open, volume, and then at the very final, we're going to get the symbol. Now it's also worth noting the data types. For the first and the last, for the date and for the symbol, we'll use text, and for all other we'll use numeric because the data will be representing numbers. We also have this CREATE TABLE. If we wanted to run this as script and create it with Python code, we could using this script here. But we're going to go ahead and just focus on what we learned previously about the insert, and the reads, and the updates, and the deletes, and we're going to focus explicitly on the insert many because we're going to go through the API and gather all of the data, build a list, a list of tuples, and then use the code. Here are the two code templates we've used previously, and we're going to merge them. On the upper left we have the API where we go through and get the stock information from the Yahoo API, and on the lower right we have the information on how we open a database and we insert many records. Let's move this code into one file. Here we are with the Yahoo API code on top, and the database code commented out on the bottom. We're going to create a new variable called myList, and this going to store the list of tuples, and we'll later use this to insert the records into the database. We'll also need to add the stock symbol at the end of the tuple, so I'll add the Ford here, or F for Ford. To condense this code, I'll change the word splitLine to spl, just so that we have more space on the screen. We want to make our code more versatile, and not hard code the company Ford, so we'll create a list for s in, this will be our stocks, and I'll start off with Ford. Later on we'll add some more companies. And I'll go ahead and indent this so that we can iterate through multiple stocks, and we'll change it accordingly. We'll put the s here so that will load the database, and we'll update the API, the URL to the API, and put this into two lines so it's a little bit more readable here on the screen. Let's add another stock. We'll add Microsoft. So we'll iterate through this loop. First, we'll look at all of the stock values for Ford and then Microsoft. We can run this code by opening a command window and executing the script, and we can see we get approximately 500 records, so we have a year of data for each stock. Finally, let's add the database code. First, we'll change the name and point to our new database, which is called stocks. db that we just created. Next, we'll use the appropriate table name, which is stockprices, and we'll enter all the question marks for the parameters. These are all of the columns that are going to go into that stockprices. Now once we have that, we're going to insert, previously we did an example of employees, well now we're going to use myList. We're storing all of those records, approximately 500 records, and when we actually run this code, we will insert the records. And as a confirmation, we'll print out the row count. So let's run this code, and we get the appropriate number. The one difference here is the header that we saw previously, and now we're getting just the values of the stock. We can go to our DB Browser, look, and browse the data. Here we see all the data for Microsoft, and we can also do a select count, and we can confirm that we have the number of records that we expect. In this module, we're learning how to persist data. We took data from a Yahoo API, grabbed it, web scraped it, and put it into a database. Next, we'll be learning more tools. We'll look at the SQLite command line, and we'll look at pickle files.

Miscellaneous Tools and Pickle Files

In this module, we're learning how to persist data. We learned how to work with a database and with files, and now we're going to focus on some miscellaneous tools and techniques. We'll start by learning the SQLite command line. The SQLite command line is an excellent way to interface with the database where we can see schema information, table information, and data. Next, we'll learn about files, and in particular, pickle files. And pickle files are a way to serialize data and store dictionaries and lists and these complex data structures that we've learned about directly, and not to transfer them each time we take them out of a database or take them from a file. When working with files, we'll also learn about data objects, or data containers. For example, if we have a stock object, we can have properties, such as close, open, date, and symbol. So let's get started and learn about SQLite command line. The best place to get information and get started is at the sqlite. org website. And if you go to this website, you can download the appropriate software, and if we're going to download for Windows, we'll download the sqlite3. exe file. Within the bash or within Linux, you could just type in sqlite3, and it will open up the window to the command line. Now if we go ahead and run the sqlite3 command. exe in Windows, we can go ahead and start the application, and we can see we get a command line, and that's exactly what this is, it's a command-line interface to SQLite database. And here we can type in many things, the first one to start off with is. help, and we get all the general information and general commands. Some of the commands we'll focus on is. table,. headers, and. schema. But let's go ahead and open a database. Now we should navigate to where our database is, and we can do. open, and then we're going to open our stocks. db. And now we can hit. tables, and we can see we have one table called stockprices. We can do our. headers and turn those on, and then we can go ahead and pull out some data specifically. Now once we enter this, we should always end our SQL statement with a semicolon, so I'm going to, here, say, select \* from the table with a semicolon, and we get all of the data. Now if I want to limit just 5 records, I can say LIMIT 5, again ending with a semicolon, and we get here our 5 records. And we also see our tables, because it's on, we see all the header information, which is, or, excuse me, our header, so since it's on we see the date, close, high, low, open, volume, and symbol. Now let's look at. mode. Now by default, we are in the list mode, but I'll explicitly set that,. mode list, and when we run our SQL command with LIMIT 5, we get the 5 records just as before. But we can also change modes, so next let's change modes to. mode csv. When we execute this command, we get the same data, but the deliminator is now a comma. Let's change to. mode insert, and insert, limiting 5 as well, we get SQL statements, so we can build a table of SQL statements, and use these SQL statements or insert statements to insert into another database. This going to be helpful because later we're going to actually export the data to a file. So we can do a csv file or an insert file. Next, let's look at. schema, and when we execute this command, we get the scripts to create the tables. And here we can see all of the tables with the data elements and the data types. Another command is. once, and with the. once we can direct output, instead of to the screen, we can direct it to a file. So I'm going to put. once, and put output. txt, and then for the next command, for the next one command, I can put select \* from stockprices; and it's going to take all of that output, and instead of being directed to the screen, it's going to be directed to a file. I can pull up the file, and we'll see all of the data, instead of on the screen, we'll see it in the file. So this is an excellent way of taking any of the data that we're going to get out from a SQLite command, and instead of putting it to the screen, we can put it to file. Now notice that I'm in insert mode, so that's why we have a bunch of insert statements which we can load data into another database. Now if I were in. csv mode, then I could see all of the information in a csv file. Finally, with a SQLite command, you can enter any SQL command. We're going to go ahead and enter some scalars. We can use some where clauses, we can do some order by, and we can do a variety. And again, we can be in a variety of modes. We can be in list mode, or we can be in csv, or we can be in insert mode, get the appropriate data out. And if necessary, if we would like, we can export all of the data, or output all of the data, into a text file. Once you are done with the sqlite3 command line, you can type. exit, and that will close out the window. So that's a brief introduction into the SQLite command line where we have a very good tool to access a database and schema information. Next, we're going to turn to pickle files, and we'll also be looking at data objects, or data containers. Now, a pickle file is nothing more than our data structures that we learned about previously, and it's serialized, and we save it, essentially, in a serialized form. So instead of extracting it from a database or importing it from a csv and then building our dictionaries, or building our lists, or building our complex data structure, we can actually save it in that form, and store the objects, essentially, as a serialized file. So we looked at this code before, and in this case, we're going to go ahead and getStocks. It opens up our database, it pulls some information, and it builds or returns a list of tuples. And the tuples store the various positions, row 0, 1, 2, 3, all the way up to 6, which essentially are the date, close, high, low, open, and volume, accordingly, which is in our database. Now what we're going to do is when we pull this data out, we can actually see that we've gotten all the records, but now we're going to take this code, this little bit of code here, and create a pickle file. And it's really simple. So I'm going to create a my\_file, and I'm going to open it, and I'm going to create a name. This is the file that's going to store all of the pickle information. Again, the pickle is nothing more than the data that's serialized. It's going to be the list, or the list of tuples, in this case, and I'm going to use wb as the mode because we're going to be writing it back into binary. Now some of the earlier versions were in ASCII, but all the later ones are going to be in binary, so mode2 is going to be the example here. Now in order to take the data, the two things that we need to do is to dump it, and then we can also load it. So dumping it is actually saving it. So we're going to save the data structure called stockList, which is a list of tuples, and we're going to put it into my\_file, and we're going to be using the protocol version 2, in this case, this is the newer version, and we're generally always going to be using 2. So then I close the file, so it's just these 3 simple lines of code, and when I execute it I can verify that I have the appropriate 500 records, and what I've done, I've just created a pickle file. And the file is not only the data, but it's serialized, so it's the data in the form of the stock list, which is a list of tuples. Here I can see the file that was created, and we can see the pickle file, essentially, that it's right there. Now if we open it, it will be in binary so that we won't be able to read it, not through the text editor, but we can actually take this data and load it in. And to load it, we just use three lines of code. Similarly, we're going to use my\_file = open. We're going to open the file now. We saved it into stockPickle. dat. This is the format we've chosen. The. dat is just something, our own naming convention, and now we're going to be opening it in rb for read-only, and then we're going to take the pickle list, we're going to create it here. It's going to be the same serialized data we've had previously, which is a stock list, a list of tuples, and we're going to use the. load from the pickle, from the cPickle. Now notice we've imported the cPickle, so it's readily available. We can bring in this data. Now we can actually just write a for loop, iterate through, and we have the data. So a pickle file is nothing more than serialized data stored in an output binary file. In this case, we've called it stockPickle. dat. Now when we iterate through the stock list, remember it's a list of tuples. We can pull out the 0, the 1, and the 6 position, and this is going to correspond to the date, the close, and the symbol. And as we run this, we've extracted all of the data, which we've stored in binary format, and it's readily available with just a few lines of code. Our final topic in this lesson is to create a data container, or a data class. So I'm going to create a class, and I'm going to call it stock, and I'm going to use pass. This indicates it's just an empty container. Now instead of creating a list of tuples, I'm going to create a list of data objects. So I'm going to create a new instance of this class that I've created. It's going to be called Stock. Typically, we use the object as a class as an uppercase, so we're going to use stock. That's going to be the instance of the class, and it's going to be equal to stock in parentheses. And now I'm going to set my data elements, so I'm going to have date, that's going to be the first row, and I'm going to have close, high, low, open, volume, and symbol. So I'm going to go ahead and create all of these. Instead of knowing them by the position and building a tuple, I'm going to actually have named parameters, or named properties of this class. And this is similar to named tuples, but another way of doing it is with this empty container of stocks, and creating everything here. So as we finish up here, instead of appending the tuple, now we're going to append the data object. When we execute this code, we print the length of the stockList, and we can see that we get 500, or approximately 500 records just as before. Now in this case, instead of having a list of tuples, we're going to have a list of data objects. And to extract the data, we can iterate through the stockList, and we can print out the various data we want. And instead of using the positions, we're going to using x. close, x. symbol, and x. date. Those are the three data we can see, and if we execute the code, we get the appropriate close, the symbol, and the date. Next, let's take this list of data objects and save it, or persist it in a pickle file just as we did before. So we're going to create a my\_file. This is just a variable name, and I'm going to open, again, I'm going to open, this time I'm going to call it objectPickle. Again, you can call it anything you want. And I'm going to use the. dat extension. Again you can use any extension you want. It's going to be a binary file that we're going to create, and now we're going to cPickle, and we're going to dump it. And dump it means we're saving it. We're going to take this data structure that we've created, which is called stockList, which is a list of data objects, and we're going to save that into my\_file, and the my\_file is called objectPickle. dat, again, using 2, which is the binary format, and then we're going to close it. So when we go ahead and execute this code just as before, we're going to have a file, and we're going to have that file, it's going to be persisted, and it's going to be called objectPickle. dat. So let's go ahead and run this code. We get 502 records just as before, and we can open up Windows Explorer, and we can see this file has been saved. It's the object Pickle. dat. And just as before, we can now extract that data. In order to extra that data, we simply need to create an instance of the class because we're going to be extracting it and putting it into this. Just as before, we're going to create class, stock, and pass, indicating it's an open object. And as before, we have our two lines of code, which is my\_file that opens up the data object into read mode, in this case. Instead of dumping it, we're going to be loading it because we're going to be taking the data and bringing it in and putting it into myPickleList. And now we can iterate through for x in myPickleList. We're going to take the same instance of a stock. This time I'll call it stk, and we are going to set stock equal to x, and now we can go ahead and print stk. close,. volume, and all the properties accordingly, execute the code, and we can see that we have all the data. So we've dumped it into a file, and now we are loading it back from that file into this page and displaying all the data back to the user. So that's a brief introduction into data elements, as well as pickle files. So we've looked at SQLite command line, and we looked at persisting data into pickle files, and we've also looked at data containers or data classes, which is nothing more than data with properties such as close, high, low, volume, or if you had an employee object it could be last name, first name, date of hire, things like that. So it's another way of organizing data. These are some of the miscellaneous tools and techniques that are available to us, the SQLite command line, pickle files, as well as data containers.

Summary

In this module, we learned about persisting data. We spent the majority of our time learning about SQLite, which is a file-based database system, and comes included with Python. Sqlite3 is a powerful database engine where we can execute SQL commands. And we learned about the DB Browser, which is a graphical interface for working with SQLite, as well as the command-line interface. We learned how to manipulate these database SQL commands using Python code. Next, we looked at pickle files, and pickle files are serializing data, are dictionaries, are tuples, are other structures that we learned, and keeping them in that serialized format, and persisting them into a file. That way, when we extract the data, we don't have to recreate those data structures. We're actually storing them in that form. We also learned about data objects, which is a convenient way to organize data, and then we applied all of these concepts to our stock market application. In the next module, we'll focus more on our stock market application, and try to build additional and more powerful tools, and practice these concepts that we've learned.

Analyzing Stock Market Data

Introduction

Hi, my name's Terry, and we are continuing our course, Getting Started with Data Analysis Using Python, and we've reached the part where we are going to do some stock market analysis. And we've learned about different data structures, we learned how to organize our data with data objects, we've learned how to use databases, and how to import and export data using CSV files. And now we're going to focus on trying to build something. We're going to look at the stock market, and the first thing we're going to do in our demo is to calculate the market value, so we're going to have a portfolio of stocks, and we're going to go through and have number of shares associated with those stock symbols. We're going to collect a CSV with a year's worth of data for those stocks, and then we're going to import it into our dictionary so we can do the calculations and calculate the value of our portfolio. And that's the first part, and the second part is we're going to look at our daily difference, and we're going to define our daily difference as the difference between the close and the open, and instead of sorting it chronologically, we're going to store this in a dictionary. We're going to actually sort it based on the difference value, so we'll start off with the differences, the dates in which it changed the least, and then end with the, which we saw, the greatest difference. So we'll be working on organizing our data, importing it from spreadsheets, and then eventually sorting our data. And our main focus is going to be with the data structures, which is the heart of this course. First of all, we have lists, and lists are collections of items, and they allow us to sort data, very flexible data structure within Python. And then we have dictionaries, and dictionaries are key value pairs, so they are unordered, and we're going to nest these ideas. So we're going to have, in our example, a dictionary in which the key is a tuple, and the value is a tuple. And we're going to also learn about tuples, of course. Tuples are excellent for storing records, and we can reference the position of a tuple based on its 0, 1, 2, 3, or whatever the position is, so it's 0-based. And that's a common way of referencing each item within a tuple. A set was the final data structure we covered in the course, and we're not going to use it in this example, but it's excellent for testing unique membership. We're going to be using Google Finance. Now initially, when we started this course, we were going to use Yahoo API, and that Yahoo API is temporarily not available, and that's really a major factor whenever we're using data analysis because if we're getting data from a Web API, or from a website, or if we're scrapping it, often that data changes, we don't have control over it, so we need to be flexible and be able to gather data from a wide variety of sources, and to adapt when things like that change. So we're going to be using Google Finance, or we could use any type of finance site from any brokerage where we can download a year's worth of data.

Stock Market Analysis Demo

Let's go ahead and get started. Let's look at our demo and build some basic tools for stock market analysis. In this lesson, we're going to build out our stock market application and look at practical ways in which we can use dictionaries, lists, tuples, and sets to analyze various data sets. And we'll be using Google Finance, so if you go to Google, you can look up any stock price, and if you go to the historical prices, you can actually download a CSV file, which has a year's worth of data for a particular stock. So we're going to go ahead and do that. We're looking at Google stock here, we're looking at historical prices, and then we can go on and download to a spreadsheet. If we download and open up that spreadsheet, we'll see a year's worth of data, and it includes the columns, the date, the open, the high, the low, the close, and the volume. And I'm going to go ahead and do this for a couple of stocks. I'll be using Google, or Alphabet, as the company, parent company's known as, and also Ford as another company. Let's start by highlighting our goals, or what we want to accomplish with this code. First, we want to calculate the value of the holdings over time, and we have a year's worth of data, so we'll be looking at the combination of a couple of stocks, we can do more, but we'll be calculating the value of those holdings over that one year. And second, we want to calculate and sort the daily difference of the holdings over time, and again, we're looking at a year's worth of data. We can go to Google Finance, or any other stock or brokerage firm that has this type of data and is widely available. So we want to start with this code by first importing the CSV and some date functions, because we have dates which represent the years, or the trading days of the year's worth of data. And next, we want to look at a dictionary, because this is where we're going to store the key value we want to get out. And the key is going to be a tuple, and the tuple is going to consist of the symbol. It's either going to be Google, goog, or Ford, and it's the date, the trading day. And the value of this dictionary is going to be a tuple, as well, and it's going to be the open and the close. Now first we're going to be looking at the close, primarily to calculate the values over time, but later on, for our second goal, we want to calculate the daily difference, so we're going to need both the open and the close. So let's do this first for just one company, and let's use the Alphabet, or Google, and it's goog. We've seen similar code before, but we know when we download the file from Google Finance, we know the file will be named the stock symbol,. csv, so I can create that name as a string, open up the file, insert it into a reader, and then use the next to skip the first line, which is the header. I've also saved as a comment the headers of the CSV file, date, open, high, low, close, and volume, and we're going to need the relative positions here. We're interested in the date, the open, and the close, and that's the 0, the first, and the fourth position. Next, we iterate through the reader, and we can build our dictionary. The dictionary, again, the key will be a tuple, and the tuple will consist of the stock symbol, and the trading date, and the value will be a tuple, as well, and it will be a float value of the open and the close. So we'll have the difference there between the open and the close. Finally, we can iterate through the dictionary just to test to see if we're capturing the data accordingly, and print out the length of the dictionary. So let's go ahead and run this code. I'm going to type in Python, and the name of the script, which is stocks. py, and when we execute this we see a year's worth of data. Here we have the tuple first, as the key, which is the stock symbol, and the date, the trading date, and then we have another tuple, which is the value, which is the open and the close prices. Finally, we get the length of the dictionary, which is 251 because there were 251 trading days in the last year. So let's take this bit of code and modify it some. First, I'll delete these comments here just to create a little bit more room, and what I want to do is to make this more universal, and not look specifically at downloading or importing just one file. Rather, we want to make this more universal so that we can import multiple files. So I'm going to go ahead and indent this space here, and I'm going to create a for loop, and we're going to iterate through multiple stocks. In this case, I'm going to use two stocks and I'm going to set this in a list, and we're going to have Google and then we're also going to have Ford noted by the symbols here. And once we do that, we can change our stock symbol to a variable, and I'll call it x, and then we can have both the google. csv file and the ford. csv file. Now we should've downloaded these files and placed them right next to our Python file so that it's readily available. We can go ahead and open this, and now when we run it, we're going to, instead of getting just one year's worth of data for one stock, we're going to get one year of data for two stocks, and we build in the dictionary and now display it out, we can see, first of all, we have 502 records total, so it's more than double, and we can see intermixed we have Google and Ford stocks for various dates because a dictionary is inherently not ordered. And we're going to have to order it in our second part of the problem. So we're going to use these data sets, use these lists, tuples, and dictionaries, and work with them in this example. So we can put our shares here to make this even more versatile, and we're going to put our shares here, Google, and I'm going to say that I have 10 shares, and then I'm also going to have another stock we'll be looking at, Ford, and I'm going to have 1, 000 shares here. So we're going to go ahead and iterate through this list, and I'm also going to need market dates, because we just saw that everything is out of order, and I want to sort them. In order to sort them, I'm going to create another structure, market dates, and I'm going to make this a list because a list is something that I can sort. So as I iterate through, instead of specifically typing in Google and Ford here, I'm going to actually go through the keys of the share. So if I want to add another stock, I simply go to myShares dictionary and add another stock and the number of shares. So now I'll iterate through both keys, in this case the keys are the stock symbol, Google and Ford, goog, and f for Ford. And also I want to build this stock, or excuse me, this marketDates, and that's going to be a list. So I'm going to take the dates, I'm going to do parse date0, that's just to verify it's a date, and if it's not in marketDates, I'm going to insert it here because I have two dates for every stock symbol. I only want to have one master list of just market dates, and this is going to allow me to sort it. And if I sort it, I have a chronological list of all the market dates, and then I can look up the dictionary values for Ford or for Google, accordingly, and get the high or low, and get the stock values. Let's look at the market dates a little bit closer. First, I'll comment out this other bit of code, and then we'll print out the market dates for x in marketDates, and we'll go ahead and execute this code. We'll print out the dates themselves, and we'll just verify we're getting the information we want. Now because all of the dates were sorted in chronological order, when we append it they are in chronological order, and when we display it, we can actually see all of these dates here in chronological order, starting with about a year ago, and moving up closer until we get to the current date, which would be on the top. Now that I have a chronological list of market dates, I can iterate through that list, and look up the specific values for my two stocks, Ford and Google, and I'm going to have the closing price, and I'm going to also have the number of shares. And if I calculate that, if I multiply those together and I sum them, I'll get my total. So I'm going to create a dictionary called total, and total is going to store my final values. Next, we iterate through the market dates for date in sorted marketDates, and I have nested loop for stock in shares. keys, and this will iterate through our two stocks. Now here's the key point. We are building a total dictionary, and the key value is going to be the date. And in order to load this value, we're iterating through a myDict, and this dictionary has two values for each market date. So here's the key concept here, if the date is in the total, that's the first time we're going through, we already have a date in the total, then I'm going to have to sum the existing value. So date, for total with a key value date is going to equal total with a key value date, the current value, plus the new values, which is going to be from myDict. It's going to be the stock and the date. It's going to be the first element because that's going to be the closing price, and also we're going to multiply that by the number of shares. So in the second case, in the else statement, we're, we do not need to sum in the total with the dictionary value of the date because it doesn't exist yet. And in this case, we can just backspace this out, and we have the total value with the key of the date, is going to equal the first value in myDict, and that's going to be the shares times the stocks times the closing price. So when we run through this, we're going to get a total value including both Google and Ford, and we can now sort, we can print that out accordingly. You can design your data structures in any way you like, and there is no inherently right or wrong way. These are just tools for you to manipulate your data as you need. In our case, we can iterate through the market dates in chronological order and print out both the date and the total. And the total, again, is going to have the value of both the Ford and Google combined sum together. Let's go ahead and execute this code, and we can see here the dates, and here's the sum value. And we can see all of the dates in chronological order. We've completed our first goal, and know the value of our holdings over time. And it's worth noting that we used an interim data structure, a list of market dates, in order to sort our information. We're going to use a different method here. We're going to use the operator, and instead of using the total, now we want to get the difference, and we're going to define the difference as the close minus the open. And we're just using that as an example. And this is going to be very similar to our previous code, but instead we're going to use the difference, and, instead of a total. And the difference is going to be a little bit more involved because we now have to take two values and subtract them out. So let's start with the else portion of the if clause, and we're going to have a difference dictionary here, and the key value is going to be the date. And what we need to get is the close value, and the close value is myDict, we take the stock and the date, this is the value, and we take the first position, and that represents the close. And now we can take the open position, and that's going to be the same thing except it's going to be in the 0 position, and so we're going to have the myDict with the key values, so the stock and the date and the 0 position. So this is the close minus the open, and then we can multiply it by the number of shares. Now in order to compute this for the if side of the portion, the if side of the clause, we just copy this, and we can paste it above, and the only difference here, now if it already exists within the dictionary, we have to take the existing value and add it in. So I'll first paste this in, and then we can add the extra value of what's currently in the difference dictionary. So here's the pasted in portion, and now we can take the diff, and we can add these two together. And then the only final thing I need to do is to change total to diff here within this first part of the clause. I'm going to clean up the code just a bit here, and add a few more lines so it can fit all within one screen here. Just so that we can see everything on one page, I've created this variable called closeLessOpen, which is the close minus the open, and I put it, this one line, and essentially broken it down into two lines. So now if we want to see and run this code, we can actually iterate through the market dates and print the date and the value stored within the difference dictionary. And let's go ahead and run this code, and we've called this stocks2. py. And I notice I'm getting an error in line 28 of my code, and I can see that I've misplaced one of the closing brackets, so that's easy to update. I can fix it both in the if and in the else part of the clause. It's the same error on both lines of code, so let's go ahead and correct that and run it again. And when we run it again, we're going to get the total differences for every date. What we see here is the difference dictionary. The key is the date, and it's sorted chronologically by date. And then we have the value, and the value is the difference for each of those days. On the very bottom the last entry, we see a difference of approximately $97. What we're going to do next is sort this based on the difference values. And in order to do that, we need to use the operator that we imported at the beginning, and we're going to sort, we're going to do sorted diff, that's going to be our variable, and it's going to be the sorted value of, we're going to take the dictionary items, so diff. items, and the key here is what we're going to sort it on, and in this case, we're going to use the operator. And we're going to sort it on the first position, which is the value, and if this were a data item, we could also sort it on the name of that data item, but in this case we're going to do it by position, and it's the first position. So now when we run through this, we're going to do the same idea for x in the sorted diff, which is the structure that we just created. We're going to iterate through and print the same values out, but instead of it being sorted chronologically by date now, it's going to be sorted chronologically, or sorted increasing values based on the difference of each day. So we can see the different days in which it changed the least, or a negative amount, all the way to when it changed the most, or a positive amount. Let's go ahead and run this script, and we get the following results. Here we can see the difference, and the maximum difference for all the last year was about $571, and now we've sorted, instead of chronologically, we've sorted by the difference value. In this lesson, we started by downloading some financial data from CSV files, and then importing them into our Python program. We learned to organize our data using lists, dictionaries, tuples, and sets, and once we organized it and did some calculations, we learned how to sort our data.

Summary

The central focus of this lesson was a demo, and we started by calculating the market value of a portfolio of stocks, and then we looked at calculating the daily difference, and we sorted it from the least to the most as far as a daily change within the stock market value. We then looked at data structures, and this is the heart of the course. We first learned about lists, and we illustrated those. Lists are an easy way to append and sort data. We used dictionaries. Dictionaries are key value pairs that allow for quick lookups. And tuples are excellent for storing records, and they're based on position, we could hit each element. And sets, although they were not part of this demo, we learned that they are excellent for unique membership. Finally, we looked at Google Finance, and we gathered all of our data from Google Finance. There's many ways to gather data, and as data scientists or analysts, we need to be flexible in gathering those data because the sources often change. In the next lesson, we're going to look at visualizing data and presenting stunning charts using pygal.

Building Simple Data Visualizations

Introduction

Welcome back to our course, Introduction to Data Analysis Using Python, and in this module, we're going to focus on Data Visualization. And data visualization is taking all our data that we've analyzed and presenting it in stunning charts, or visualizations, or graphs that allow users to quickly grasp the key concepts and the data information. And we're going to be using pygal, and if you go to pygal, one of the first steps that we need to do is make sure we pip install pygal. It's very simple if you haven't already done it. Once you install it, we can start using the various features. Let's go to pygal. org. This is the website that documents many of the features that are available in the pygal library, and as you can see, we can create many different types of stunning graphs that are available, and it's very easy to output these files into svg files, which can be presented or embedded within our websites. These are some of the file types, or the chart types. We have line, bar, histogram, XY, pie, many more, as you can see through here. And as we go through, we're going to focus primary on the line charts, or the more basic charts, the bar charts. But let's take a quick look at some of the features that are available through pygal. For example, in this basic line chart, we can create this visualization, and mouse over, and see different data points. Here's a stacked version of the line chart. Here's a histogram. Here we can see various data elements, and present that to the user in stunning ways. Pie charts are another basic element where we can present percentages of different factors and show it as a whole in 100%, in the entire pie. We have box charts. So there are many types of charts that are available, and go ahead and browse through and look at these different charts that are available. And you can use them appropriately, how you want to capture the data, what you want to show in your analysis, and how you want to present it to the user to capture the most attention. Let's go ahead and focus back on our line graph. This the most basic one. All of them have a code sample as is illustrated here. It shows you exactly how to use the library, and how to import it into your Python, or what the exact steps are of the code, and present it out to the user. Next, let's turn our attention to styles, because with the pygal library, we're easily able to take the exact same data and present it in many different ways. Let's take a look at two samples. This is the Clean data. Here's an example of a way of formatting, or styling the data, and presenting it out to the user. Now let's take this exact same data, and use something substantially different. Let's look at the Dark green, and the Dark green appears this way. So you can go ahead and browse the different styles because it's very easy to change the styles and get the exact look and feel that you want to present to the user. Finally, we're going to output our charts and graphs as SVG files. Now the pygal library allows us many ways to output the data. We can output it as an image, such as a PNG file, or we can embed it into a webpage if we're using a Python web platform. In this module, we'll output the data as an SVG file, which we created in the same directory of our source code. And we can click on it and open it up in the browser and see the results. Let's get started with some demos. First, we'll look at some basic line and bar graphs, and we'll do some chart basics, just gathering data and presenting it, and working with some imaginary data. And then second, we'll actually try to grab our stock market analysis data that we've looked at previously, and we'll graph that and chart that to the user. So let's go ahead and get started and learn about data visualization with pygal, and data analysis using Python.

Demo: Chart Basics

In this module, we're learning about data visualization. So in this demo, let's get started with the chart basics. We're going to do some basic plotting using the pygal library, and build some basic visualizations, including line and bar charts. So if you go to pygal. org, this is the basics, again, and this is focused on bar graphs, or line graphs, and there's some sample code on the bottom. And we're going to start with some sample code similar to that. Let's look at some sample code. We start by importing the pygal library, as well as some styles where we can show the contrast between the Clean and the Dark green style. Next, we have some data elements. These are stored in a dictionary. We can use either a dictionary or a list. And notice the keys of this dictionary equate to the labels. The keys to data1 are 1, 2, 3, 4, 5, and similarly, the labels are the same. This line here determines the chart type. In this case, it's a bar graph, and the style, accordingly, as Dark green. We can add the data to the chart and render the file with these two lines of code. So let's go ahead and execute the Python script, and then we can view our SVG file here within the browser. And when we highlight over the different columns, we get the data accordingly. So let's add a little more complexity to this example. Let's create another data set. Here I've cut and pasted in data set 2 with slightly different values, and let's change the style from Dark green to Clean, just to show some contrast. The next step is to add this new data set to the graph, so let's cut and paste this, and when we cut and paste it we'll add the new line, data2, and so now we'll have both data1 and data2 appended to the graph. We can render the file just as before as an SVG file, and when we go ahead and run the script and view the graph, we can now see a graph with two data sets in the Clean style. Just as before, if we mouse over the columns, we see the corresponding data element. Now let's add another graph to this code. Instead of just focusing on a bar graph, let's add a line chart. So what I've done is I've cut and pasted and created two bar graphs, and I'm going to change bar\_chart everywhere to line\_chart. So let me go ahead and update the code here, and we can see everywhere before where it had bar\_chart, it now has line\_chart. This line designates the chart type. It's now going to be a line graph, and secondly, this line designates the output file, which would be called line\_chart. svg. After we execute our Python script, we can open our SVG file, and we see, instead of having a bar chart, we have a line chart. So in this lesson, we learned the basics about plotting using the pygal library. In the next lesson we're going to take the same principles and apply them to our stock market data.

Demo: Stock Market Charts

Now that we know the basics of the pygal library and the data visualization tools available, we're going to focus on stock market charts. We're going to go ahead and look at the data that we've studied throughout this course, stock market data, and apply those same principles, and use our pygal library to create some charts for stocks. And in this case, we'll be looking at two stocks. We're looking at Google and Amazon, and if you highlight over any of the dots, you see the date, the stock, the stock price, accordingly. We're going to go ahead and build this, and this is our final product in this demo. And we're going to start off with the basics of charts that we looked at before. First, we import the pygal library, and we're importing the styles. We're going to use the Clean style here. We set the title. We set the dates. And there's really just two key data structures that we need here. We need the dates, and that's going to be the labels on the bottom axis, and then we needed the data set, and that's going to be the list of all the closing prices. So we need a list of dates, that'll be a string, and a list of closing prices, and that's going to be a float value. Then we'll render everything into a stock-chart. svg file. Now remember, we need to get our data as we did before. We're going to go to Google here, and I'm going to hit Download. And once you download the prices, we'll have a year's worth of data. We're going to do this both for Alphabet, which is Google, as well as Amazon. So the next thing we need to do is to take the code we've looked at previously, and we're going to import that data. So we start off, I'm just going to do one stock first, I'm going to use Google, stockSymbols = goog. And then for symbol in stockSymbols I'm going to start with just one in the list. We create the CSV file, we open the CSV file, we read the data in, and we also skip the header line and we use reader. next to skip the header line. Now remember, the key bits of information we need is the dates, and that's going to be a list, and the data set. The dates is going to be a list of all the dates. In this case, we're looking a year back worth of data, and the data set is going to be the closing prices, which will be float values in a list as well. And so in order to get that data, we need to iterate through our CSV file for x in reader, and then we're going to do the dates. append. We're creating our two lists here. One of it's going to be the 0 position. The 0 position is the date. And then finally, we're going to convert to float the fourth position, and the fourth position is the close. And when we append that, and we iterate through the entire list of stocks for that one symbol, in this case, Google, we have the two lists, one with the dates and one with the closing price. We can now run this script. We'll do python stock. py, and then we go ahead and we can see, let's go ahead and refresh this chart, and we see for one stock we have the Google prices. Now notice the way the CSV file is downloaded, it's from the most recent to the most far away, a year away. So later on we're going to inverse this and get the plot, so instead of it going downward in price, it's actually going upward in price because the last year Google stock has actually been increasing. Now that we have a chart for Google, let's add Amazon. So I'm going to create a stockSymbols, and we're going to put Google, and we're going to add Amazon here. And then we're taking our stock prices, this is where we're going to store our data set of all the closing prices because now we have two, we're going to show two values in this dictionary. The key is going to be the symbol, and the value is going to be a list of closing prices. And so as we iterate through, we've gotten one, first time we went through, we got one, all the data for one stock, Google, and now we're going to iterate through and go through first, to get Google, and then get Amazon, and we're going to store that within this dictionary we just created, stockPrices, with the key of the symbol, and the value's going to be the data set, which is a list. And now on the bottom, when we're adding a data set, we can go, we can iterate through that same list for x in stockSymbols. We're going to have a line chart, and add in the Google price. And I'm going to change this. Instead of using x I'm going to use symbol, and so now we're going to go for symbol. In symbols we're going to get our two stock symbols, which is Amazon and Google, and we're going to add, first time through we're going to add all the values for Google, and the second time through we're going to add all the values for Amazon. And when we run this, we can see that now we have two stocks in here, we have one for Google and Amazon, and if we highlight any of the data points we have the corresponding value for that stock, the date and the closing price. So we need to clean this up. A couple things that we want to do. First of all, we have a lot of data points here. It's very detailed. If we want to keep that, we can, but if we want to just show maybe every tenth date, we can get a little bit of a cleaner graph or not as busy. And we also want to inverse. We talked about this shows the most recent prices and a year ago, but typically when we're looking at stock prices, we want to see the most recent prices on the far right and so we can see that the price is increasing. So I've changed that just with a simple list reverse. And once we reverse the list, instead of it going this way, implying the price is going down, it's actually the price is going up because the prices have been going up over the last year. So we do that, and we get the data accordingly. Finally, what we want to do is change the count. Instead of having a point for each date, we're going to have, in this case, we're going to iterate our counts and only show maybe every tenth date. That allows us just to have a little less detail if we want to focus, and this is user preference however you want to do it. And in this case, what I'm going to do is do a count, mod, 10, so I'm only going to show every tenth element, and I'm going to save those to the data set. And now when we run it, instead of having as much detail as we had before, we have only every tenth trading day, or every other week. And when we refresh this stock we can see this graph is just, it's not as busy. So however you want to present it, and whatever you want to do, whether you want to show all the data points, or just one-tenth, that's fine. So in this lesson, we started by focusing in on the charts, and we learned about pygal, and we learned the key, two data sets that we need are we need to have the dates and we need to have a list of closing prices, and we do that here. And in this case, we want more than one stock for Google and for Amazon, so we're going to have a loop for symbol in stockSymbols. And then in order to gather that data, we need to go through our CSV files. First, we download the CSV files for Google, for Amazon, we iterate through both of those, we build our CSV file, we open it, we set that data in the reader, we skip the first line, and then we're building our dates and out data sets, our two lists that we need. We're iterating through for each stock that we have. We can easily add more stocks here, as many as we want. But in this case, we're looking at Google and we're looking at Amazon, and as it iterates through, it builds a dictionary of stock prices, and that's a dictionary where the key is the symbol, and the value is a list of closing prices. So we've gotten a big introduction into data structures first, and now we've looked at pygal. And we first applied pygal to some basic charts, and now we're applying pygal data visualization libraries to our stock market prices.

Summary

In this course, Getting Started with Data Analysis using Python, we learned how to collect data, how to process data, and in this module we focused specifically on how to present our data using data visualization tools. We used, specifically, pygal, at pygal. org. Pygal. org has a vast amount of documentation and sample code available. Some of the key methods that we looked at are the chart types. We looked at line and bar charts. There's also pie charts, and many other available if you look at the documentation. We also looked at styles. We focused specifically on the Clean style and the Dark green style, but there are many, many other styles that you could look at. We looked at how to label our graphs. There's a title, method, and there's also an x\_labels method, and you can use this just to provide some additional information about the axes. And we also looked at rendering our data. We can render it to a file, to PNG, and we primarily used SVG files. You can also embed these charts into web pages. Now the core of this course is data structures. And we looked briefly at lists, dictionaries, tuples, and sets. And again, we had to use this data to set some of the properties in the pygal library. In this module, we gave two demos. First, we looked at the chart basics. We took some just generic data and presented it using pygal library, and then we looked at our data that we've looked at throughout this course, stock market analysis data, and we presented that data, as well, into data visualizations and charts. In the next module, we'll give a summary of this entire course and highlight some steps available for further study.

Course Summary and Next Steps

Course Review and Next Steps

We have reached the end of our course, Getting Started with Data Analysis using Python, and we come to our next module, Course Summary and Next Steps. In this module, we'll look and highlight some of the topics that we reviewed and learned in this course, then we'll consider some next steps that you can learn more about data analysis. This course was based on Python. Python is a great tool for general software development and for data analysis specifically. In this course, we covered the following topics. First, we learned how to gather and collect data. We started using the Yahoo Finance API, and quickly learned that this service may no longer be available, so we need to be flexible, and we used Google Finance and downloaded the date in to CSV files, and then imported them into our analysis tools. We also learned about databases. We also learned about cleaning data, so we need to take data that may not be accurate, or may not be applicable to our algorithms, and filter them out. We learned about data structures, and we learned specifically about lists, dictionaries, tuples, and sets. Then we turned our attention to databases, and learned basic SQL to insert data into the database, as well as retrieve data. And we used command-line tools with the database. Finally we learned about data visualization in the pygal library. The core of this course was on data structures, and first we learned about lists. Lists are a simple way to add or append an item to a collection, and it can sort it. Dictionaries are constant lookup time for each element, and they're good for joining data with other data sets. And then we learned about tuples, and tuples are great for storing records. We can match each item and position in the tuple to a field or column in the data set of a database. And then we learned about sets, and sets are unique membership. Data structures is the core of this course, and we used them, and we nested them together, for example, we can have a list of dictionaries, or a dictionary where the key is a tuple. Once we learned how to collect, organize, and analyze our data, we turned our attention to data visualization, and we used the pygal Python library. Once we installed it, we were able to build some basic graphs including line and bar charts. To learn more about data analysis, you might consider the following steps. First is NumPy. NumPy is a great statistical analysis Python library. You can use it, for example, to find the variance or standard deviation of data. Pandas is another Python tool available. It has two major structures, the series and the data frame, and it's good for analyzing a set of data or a matrix of data. Finally, there's Jupyter Notebook, previously known as IPython Notebook. It's a web interface where we can mix rich visualizations, diagrams, as well as analyze our data and incorporate our data and analysis together. Thank you for taking this course. I hope you've enjoyed Getting Started with Data Analysis Using Python.