**Data Analytics Notes**

Class List

Kris R, Art A, Tim H, Tim K, Rob H, Rob B, Abhishek (“Abbey”), Priya P, Ruth D, Lacy S, Khagan K., Ben K, Bob N

* \* \* \* \* \* \* \* \* \* \* \*

**Module 2 Notes**

## Loops

We're going to calculate the total daily volume in 2018 using **loops**.

**IMPORTANT**

**Loops** tell a computer to repeat lines of code over and over (and over, and over) again. for loops tell the computer to repeat the lines of code a specific number of times. You can think of a for loop like telling the computer to "run this code for as many loops as I tell you to." There are a few different kinds of loops, but for loops are the workhorse of loops. It's entirely possible that you'll never need to use any other kind of loop.

To find the total daily volume, we'll loop through every row in our stock data worksheet and check if the ticker for that row is DQ. If it is, we'll add its daily volume to our total volume.

In VBA, the syntax for a for loop has a beginning, middle, and end. The beginning is one line that tells VBA that we're opening a for loop; the middle is the block of code to be repeated; and the end is one line that closes the block of code.

The opening line of a for loop uses the keyword For and an **iterator**. Iterators are named variables that change values over the course of the for loop, usually increasing by 1, thus holding the number of times the loop has repeated. For example, if we wanted a for loop that would loop exactly 3 times, the opening line would look like this:

For i = 1 to 3

Inside the code block, we can use the iterator like any other variable. So, if we want to display 3 message boxes in a row, showing the number of times the loop has repeated, our code block would look like this:

For i = 1 To 3

MsgBox (i)

Now we need to close the for loop with our last line. To tell VBA that the for loop has ended, we use the keyword Next and the iterator we used: i.

**NOTE**

We need to specify the iterator because there can be more than one for loop going on. These are **nested** for loops, which we'll get to later.

So our full for loop looks like this:

For i = 1 To 3

MsgBox (i)

Next i

Make a new macro and name it DQAnalysis, then run the for loop to make sure it works.

Because the iterator is treated like any other variable inside of the code block, we can use it to interact with our data. To see this, change the for loop to display each of the stock data column names in a MsgBox. There are 8 columns, so change the iterator to go from 1 to 8, and then change the MsgBox to display Cells(1, i).

For i = 1 To 8

MsgBox (Cells(1, i))

Next i

**Patterns**

In programming, **design patterns** (or just **patterns)** are templates to solve similar problems. Patterns aren't necessarily code but rather reusable structures to help us write our code.

The pattern we used follows this general structure:

1. Initialize a variable to hold a sum.
2. Set the variable to zero.
3. Start a for loop.
4. Use a conditional to increase the sum variable by a value.
5. End the loop.

**IMPORTANT**

Design patterns are larger than a single programming language. They offer a way to organize a process so that it can be put into code.

Programmers use **pseudocode** to break down algorithms and processes into a design pattern without being tied to a specific language. Pseudocode can range from writing a list of tasks in natural language to formatting code with indentation and adding simple keywords. The main purpose of pseudocode is to think through the logic of the code before actually writing it.

In pseudocode, the pattern looks like this:

totalValue = 0

For i = Start To finish

If Condition Then

totalValue = totalValue + currentValue

End If

Next i

You should always be on the lookout for new patterns to use. Every time you solve a single problem, consider whether you can use that same pattern to solve another problem.

The opposite of a good design pattern is an anti-pattern. **Anti-patterns** are common responses to problems, but they may be ineffective, too specialized, or generally counterproductive. Quick and dirty workarounds, called **kludges**, often use anti-patterns.

**NOTE**

Anti-patterns and kludges are part of a broader idea of code smells, where the code works and solves the problem it's supposed to, but something about the code indicates that there is probably a more elegant and productive solution to the problem.

We've already avoided one anti-pattern: an unnamed magic number for the row count. Let's take it even further and, instead of giving our magic number a name, we'll have VBA figure out the value for us. This way, if the data changes, our code will still work.

**Research Practice**

A ton of search results will come up, but most of them will fall into four categories:

Official documentation

Stack Overflow

Quora

Expert blog posts

**Official Documentation**

For VBA questions, the official documentation is published by Microsoft. In general, official documentation is a reference guide published by the creator of the language or software. You might think this is the logical place to check first, but that's not always the case. Because official documentation is meant to be an exhaustive reference, it's often extremely dense and difficult to understand, especially when you're first learning a programming language. In the worst cases, the documentation isn't even up-to-date and therefore contains incorrect information. It's often better to get explanations from people who are actually using the code.

**Stack Overflow**

[Stack Overflow (Links to an external site.)](https://stackoverflow.com/) is almost always the best resource for getting practical solutions. Programmers will half-joke that their job is just googling problems and copying and pasting Stack Overflow code. Stack Overflow works like this: someone asks a question about a problem they're running into and experts will provide answers, almost always with sample code. The expertise level is top-notch; sometimes the person who invented the program will be the one giving the answer! Answers are voted on, so you can get a sense of which answers are more authoritative than others.

**Quora**

[Quora (Links to an external site.)](https://www.quora.com/) has a similar format to Stack Overflow, but the questions and answers tend to be more theoretical. The expertise level is similarly high, but the emphasis is more on understanding concepts rather than just getting some code that works. If you're struggling to do something, go to Stack Overflow; if you're struggling to understand something, go to Quora.

**Expert Blog Posts**

On the internet, anyone can claim to be an expert, so blog posts vary wildly in quality—especially because there is no mechanism for people to give feedback on the quality of a blog's explanations. However, expert blog posts tend to blend sample code and theory, so a good post will help a new concept click.

Narrow Your Search

There's no reason to feel awkward about using search engines to figure out problems. Welcome to being a programmer! A big part of any programming job is googling solutions and spending a lot of time on Stack Overflow. Even experienced programmers do this on a daily basis.

Our search "VBA get number of rows with data" gives a number of results, and many look promising. There's a Stack Overflow link, a link to the official documentation, and a handful of blogs and forums. Click on a few links and skim the information to get a feel for how different sites answer questions like this. Here are two curated, potential solutions. (Don't worry about finding the exact links to these. And remember that you might find even better solutions!)

**Logical operators**

**Logical operators,** also called Boolean operators, link more than one condition together, which allows for more complicated conditional arguments. The logical operators in VBA are And, Or, and Not. That is:

* condition1 And condition2 will only evaluate as true if **both** condition1 and condition2 are true.
* condition1 Or condition2 will evaluate as true if **either** condition1 or condition2 are True.
* Not condition will give the **opposite** value of whatever condition is.

In addition, there is a "not equal to" **comparison operator** that checks whether two values are not equal to each other. In VBA, the "not equal to" operator is two angle brackets: **<>**

The condition to check if the current row's ticker is DQ is

Cells(i, 1).Value = "DQ"

And the condition to check if the previous row's ticker is not DQ is

Cells(i - 1, 1).Value <> "DQ"

Since we want both conditions to be true, we'll join them together with the And operator in our if statement. This new code will be inserted in the for loop directly after the if block that computes totalVolume.

If Cells(i, 1).Value = "DQ" And Cells(i - 1, 1).Value <> "DQ" Then

'set starting price

End If

Before the for loop, create a variable for starting price. Since the prices have decimal values, we'll use the Double data type.

Dim startingPrice As Double

Price data is in the sixth column, so the value in Cells(i, 6) has the starting price.

startingPrice = Cells(i, 6).Value

Put together, the code should look like this:

If Cells(i, 1).Value = "DQ" And Cells(i - 1, 1).Value <> "DQ" Then

startingPrice = Cells(i, 6).Value

End If

To find the ending price, follow the same code pattern.

**REWIND**

That's right: we're reusing a design pattern! We'll apply the pattern used to find the starting price to our process for finding the ending price by following these steps:

1. Initialize a variable to store ending price as a double data type.
2. Check if the current row is the last row of DQ's data:
   * Check that the current row's ticker is DQ.
   * Check that the next row's ticker is not DQ.
3. If so, set the ending price to the current row's closing price.

Give it a try. Your code should look like this:

Dim startingPrice As Double

Dim endingPrice As Double

If Cells(i, 1).Value = "DQ" And Cells(i + 1, 1).Value <> "DQ" Then

endingPrice = Cells(i, 6).Value

End If

When we're done with the analysis for a ticker, we'll need to output the results, which means we'll be activating the output worksheet. When we start on a new ticker, we'll need to reactivate the data worksheet, so that code goes inside the innermost loop.

Using comments to show where we're going to put our code is a good idea. So far it would look something like the following. (Hold off on making any changes to your code just yet!)

Sub AllStocksAnalysis()

'Find number of rows (before both loops)

For i = 0 to 11

ticker = tickers(i)

For j = 2 to RowCount

'Activate data worksheet

Next j

'Output results

Next i

End Sub

**NOTE**

We can't initialize a variable more than once because VBA will assume that we're trying to create a new variable and accidentally gave it the name of an existing variable. Therefore, we don't want to put our Dim statements inside loops.

Before we start putting code into our new loop structure, we should formulate a plan. We'll use this plan to keep our code blocks organized, using comments as our structure.

We can reuse a lot of the code we've already written in the DQAnalysis subroutine, but we'll need to rearrange it to fit our new ticker loop. Remember, we want to perform the same kind of analysis we did for DQ, but for every stock in our ticker list. We also don't want to waste time rewriting code that we've already written.

Let's write our plan.

**Map Out a Plan**

Since this code might get a little complicated, we should start by writing a basic outline of the program flow. Then we'll use comments to organize it all before we write the actual code.

Our new macro should do the following:

1. Format the output sheet on the "All Stocks Analysis" worksheet.
2. Initialize an array of all tickers.
3. Prepare for the analysis of tickers.
   * Initialize variables for the starting price and ending price.
   * Activate the data worksheet.
   * Find the number of rows to loop over.
4. Loop through the tickers.
5. Loop through rows in the data.
   * Find the total volume for the current ticker.
   * Find the starting price for the current ticker.
   * Find the ending price for the current ticker.
6. Output the data for the current ticker.

**Write the Macro**

Let's put the plan into action. The following video provides an overview of this process from start to finish so you know what your code should look like.

**Notes from Nick**

**Notes 2.1**

* The activities in this class will complement Lessons **2.0.1 Make Your Way With VBA** through **2.2.4 Get DQ’s Yearly Return for 2018**. The students will benefit from these activities if they‘ve progressed through these lessons, which cover the following concepts, techniques, and tasks:
* Enable Macros
* Create a basic macro
* Understand data types
* Demonstrate the use of variables and arrays in VBA
* Understand and create conditional If-Then and If-ElseIf-Then statements
* Understand and control programmatic flow using for loops
* Use logical and comparison operators in conditional statements

[10:55](https://osuvirtdatapt-wnh9099.slack.com/archives/C02K60VU5M3/p1637423747017800)

**Notes 2.2**

* The activities in this class will complement Lessons **2.3.1 Create a New Worksheet and Subroutine** through **2.5.1 Make a Run Button**. The students will benefit from these activities if they‘ve progressed through these lessons, which cover the following concepts, techniques, and tasks:
* Understand and use arrays and indices
* Use for loops to access data in nested arrays
* Debug code
* Format cells based on a condition
* Create interactive features like buttons

Module 2 Challenge

References – Roerick98 – VBA-Challenge\_Reference\_2

FBullman-VBA-Challenge-Reference\_3

Slack – good – ensure to tag Nick in message!

SSH Key – Use Finder . Copy that and save to clipboard. “Add new Key” – copy / paste keys. Use Public key.

Module 3 – Welcome to PyPoll

In this module you will be assisting a Colorado Board of Elections employee Tom in an election audit of the tabulated results of the U.S. Congressional precinct in Colorado. You are tasked with reporting the total number of votes cast, the total number of votes for each candidate, the percentage of votes for each candidate, and the winner of the election based on the popular vote. This is normally done in Excel but Tom’s manager wants to see if there is a way to automate the process using Python. If the audit is done successfully with Python, the code you and Tom write will be used to audit not only other congressional districts but also senatorial districts and local elections. There are three primary voting methods that you and Tom will take into account: Main-in ballots, punch cards and direct recording electronic or DRE counting. Mail-in ballots are usually counted at the central office. Punch cards are collected and then fed into a machine that tabulates vote totals and sends the results to the central office. Finally memory cards from DRE counting machines are sent to the central office and read by a computer. Altogether the votes cast by these three methods will determine the final election results. After the votes are counted, your job is to generate a vote count report to certify this U.S. Congressional race.

**You** will be using Python to write algorithms that will assist the confirmation and analysis of election results. Seth, Tom's manager, has informed you that you'll need to set up Python.

Notes for the Class slides

**Notes 3.1**

* The activities in this class will complement Lessons **3.0.1 Power of Python** through **3.2.11 Printing Formats**. The students will benefit from these activities if they‘ve progressed through these lessons, which cover the following concepts, techniques, and tasks:
* Create a Python file
* Execute Python files
* Assign variables
* Understand Python data types
* Perform calculations using Python
* Create and retrieve data from lists
* Create and retrieve data from tuples
* Create and retrieve data from dictionaries
* Use the print() function to print data to the screen

Tip from Nick – keep code modular

Key to accessing Lists – Indexing

Python index starts at zero not 1

Use a lot of print statements to verify code

Last two class exercises review tomorrow – Hobby book and one prior.

Review last 7 min of class from Mon (2:26)

Nick-dictionaries are fairly straightforward to setup but harder to pull data

[9:22](https://osuvirtdatapt-wnh9099.slack.com/archives/C02K60VU5M3/p1638022963021300)

**Notes 3.2**

**My notes from 3.2 Modules – 3.3-3.6**

**Dependencies** are modules and packages, or a programming script that someone else has written, that allows you to increase the functional programming of your code, or speed and efficiency.

Python is an open-source language, which means that it is easy for others to write dependencies that can be used with Python. You can think of the relationship between dependencies, modules, and packages like Russian nesting dolls.

**Dependencies** are the largest "doll," like the Python datetime module. Inside the datetime module are functions, classes, or variables, which are the second-largest doll. The methods used for functions and classes are in the third-largest doll.

**Packages** are folders that contain a set of Python modules. The folders in the packages may contain various subpackages, or other folders. To import packages, we use the import statement, as we did with the datetime module.

**Modules** are a separate software component. They are usually Python files with a .py extension. The name of the module will be the name of the file. A Python module can have functions, classes, or variables defined and implemented.

Modules can be used in a variety of applications and functions with other programs. They may contain hundreds or thousands of lines of code, so it would be foolish to write or repurpose the code every time you need to use it. This type of programming can lead to many syntax or logical errors in the program that would require an enormous amount of time to correct.

Modules are easy to use and maintain, and they provide reusability with a simple statement like import datetime. To use a specific function, class, or variable from a module, you use a statement like from import.

If your script requires the use of programs, modules, and packages, one of the first steps is to import dependencies for your Python script.

Let's put this concept to good use and walk through how to read a CSV file

## The CSV Module

In Python there's a built-in module called csv, which allows users to easily pull in data from external CSV files and perform operations on them.

The csv module is imported by using the import statement followed by the module name, csv.

The csv module has many functions that allow us to read and write tabular data in CSV format. With the csv module, we can read data from a file that was generated by Excel and write data to a file in a format that can be read by Excel.

To see all the functions available in the csv module, follow these steps:

1. Launch the Python interpreter.
2. Type import csv to import the module.
3. Press Enter.
4. Type dir(csv). The "dir" is short for "directory".

The Python interpreter should look like this:

>>> import csv

>>> dir(csv)

Press Enter. The output will look like this:

['Dialect', 'DictReader', 'DictWriter', 'Error', 'OrderedDict', 'QUOTE\_ALL', 'QUOTE\_MINIMAL', 'QUOTE\_NONE', 'QUOTE\_NONNUMERIC', 'Sniffer', 'StringIO', '\_Dialect', '\_\_all\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', '\_\_version\_\_', 'excel', 'excel\_tab', 'field\_size\_limit', 'get\_dialect', 'list\_dialects', 're', 'reader', 'register\_dialect', 'unix\_dialect', 'unregister\_dialect', 'writer']

If you look closely at the output, you'll see a function called reader. We'll use this function to read the CSV file that contains the election data.

Using the dir() function, we can pass:

1. A Python module, like the csv module. The dir() function will return all the functions available in the csv module.
2. A variable, like a dictionary {'key':'value'}, for example the counties\_dict dictionary. The dir() function will return all the functions available on that variable.
3. >>> dir({'Arapahoe': 422829, 'Denver': 463353, 'Jefferson': 432438})

['\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']

1. A data type, like str. The dir() function will return all the attributes and methods that can be used with the str data type.

>>> dir(str) '\_\_add\_\_', '\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_getnewargs\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_mod\_\_', '\_\_mul\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_rmod\_\_', '\_\_rmul\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find', 'format', 'format\_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',

## Open a File

You can access a file in a folder on your computer if you know the direct file path. If you do not know the direct file path, but know the folder and filename, you can access the file indirectly.

After providing the file path in our Python script, we will be able to open and read the file. When the program reads the file, it creates a **file object** in the computer's memory, which provides a way for the program to work with that file. In our script, we can use a variable to reference the file object.

The general format for opening a file is, file\_variable = open(filename, mode).

Let's break down what each component is doing in the general format.

* file\_variable is the name of the variable that will reference the file object.
* filename is a string specifying the name of the file.
* mode is a string specifying the mode for reading or writing the file object. The possible modes are:
  + "r": Open a file to be read.
  + "w": Open a file to write to it. This will overwrite an existing file and create a file if one does not already exist.
  + "x": Open a file for exclusive creation. If the file does not exist, it will not create one.
  + "a": Open a file to append data to an existing file. If a file does not exist, it creates one, if a file has been created the data will be added to the file.
  + "+": Open a file for reading and writing.

Now that we know how to open a file, we need to open our election\_results.csv file and read the data in the file.

## Read Data from a File

Let's go over how to read data from a file using both a direct path and an indirect path.

#### Direct Path to the File

First, open the CSV file using the direct path method. The direct path to our election\_results.csv file will be Resources/election\_results.csv.

Using VS Code, type the following in the PyPoll.py file to assign a variable to our file path.

# Assign a variable for the file to load and the path.

file\_to\_load = 'Resources/election\_results.csv'

When we type Resources/election\_results.csv, we are telling the computer to get the election\_results.csv file that is located in the "Resources" folder.

Next, we will open the file, file\_to\_load, with the open() function, using the "r" mode to read the file. Then, we'll print the filename object. After reading the file,  close the file with the close() function. In between the opening and closing of the file is where we will read the data and perform our analysis.

Below our file assignment variable, file\_to\_load, add the following code:

# Open the election results and read the file.

election\_data = open(file\_to\_load, 'r')

# To do: perform analysis.

# Close the file.

election\_data.close()

**IMPORTANT**

Closing a file disconnects the program from the file. It's important that you close the file after you read a file and write data to a file.

When you read data from a file and it is not closed at the end of the operation, you can lose some of the data. When you write data to a file, the data is not stored in the file at first. It is written to a "buffer" in the computer memory and may be overwritten later if the file is not closed. Once you close the file, the data is stored in the file.

Python has a way to read and write to a file without needing to use the open() and close() functions every time. We simply replace the open() function with the with statement.

The with statement opens the file and ensures proper acquisition or release of any data without having to close the file, to ensure that the data isn't lost or corrupted.

The format for the with statement is the following:

with open(filename) as file\_variable:

The file\_variable is used to reference the file object throughout the script.

Let's modify this code, using the with statement instead of the open() and close() functions. We'll print the file variable, election\_data, to the screen.

Edit your code below the file assignment variable, file\_to\_load, to look like this:

# Open the election results and read the file

with open(file\_to\_load) as election\_data:

# To do: perform analysis.

print(election\_data)

The with statement ends with a colon, which means we need to indent on the next line, as we did with if-else statements and for loops.

Save the PyPoll.py file and run the file in the VS Code terminal. The output in VS Code will look something like this:

<\_io.TextIOWrapper name='Resources/election\_results.csv' mode='r' encoding='UTF-8'>

In this output, the \_io.TextIOWrapper is a Python class that will allow us to read or write data to and from the file when we used the appropriate methods and attributes. The name represents the path of the file object, and the computer tells us that the file is open in "read" mode with UTF-8 encoding.

#### Indirect Path to the File

Sometimes we won't know the direct path to the file on our computer, only that it's in a specific folder. Usually, you will know the direct path, but in a real-world setting, you may be given the indirect path to the file by a fellow coworker or your manager.

To access and open a file for which the direct path is unknown, we use the os module.

The os module allows us to interact with our operating system. We can see all the different attributes and methods that the os module uses by importing the module and typing print(dir(os)) in the Python interpreter.

>>> import os

>>> dir(os)

The list is quite extensive, as you can see.

Python provides a submodule os.path that allows us to access files on different operating systems, like macOS and Windows.

The os.path submodule contains several useful functions to make it easier to join a path, as shown by typing dir(os.path) in the Python interpreter.

['\_\_all\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', '\_get\_sep', '\_joinrealpath', '\_varprog', '\_varprogb', 'abspath', 'altsep', 'basename', 'commonpath', 'commonprefix', 'curdir', 'defpath', 'devnull', 'dirname', 'exists', 'expanduser', 'expandvars', 'extsep', 'genericpath', 'getatime', 'getctime', 'getmtime', 'getsize', 'isabs', 'isdir', 'isfile', 'islink', 'ismount', 'join', 'lexists', 'normcase', 'normpath', 'os', 'pardir', 'pathsep', 'realpath', 'relpath', 'samefile', 'sameopenfile', 'samestat', 'sep', 'split', 'splitdrive', 'splitext', 'stat', 'supports\_unicode\_filenames', 'sys']

In this output, we can see there is a function called join. The join() function joins our file path components together when they are provided as separate strings; then, it returns a direct path with the appropriate operating system separator, forward slash for macOS or backward slash for Windows.

**REWIND**

Different operating systems use different path separators to separate files and folders:

* macOS uses the forward-slash: /
* Windows uses the backslash: \

To declare a variable for the file to load, connect the os.path submodule with the join() function, like this: os.path.join(). This is called chaining.

**Chaining** is a programmatic style that is used for making multiple method calls on the same object. This is a common practice that makes code look clean and concise.

Inside the parentheses of the join() function, we will add the folder and file to join together. In this case, we'll add the Resources folder and election\_results.csv separated by a comma, like this:

os.path.join("Resources", "election\_results.csv")

Then, we use a filename variable to reference the path to election\_data.csv, like this:

file\_to\_load = os.path.join("Resources", "election\_results.csv")

Let's put all of this to practical use! In the VS Code PyPoll.py file, complete the following steps:

1. Import the csv and os modules.
2. Add the filename variable that references the path to election\_results.csv.
3. Open the election\_results.csv using the with statement as the filename object, election\_data.
4. Print the filename object.

Your PyPoll.py file should look like this:

import csv

import os

# Assign a variable for the file to load and the path.

file\_to\_load = os.path.join("Resources", "election\_results.csv")

# Open the election results and read the file.

with open(file\_to\_load) as election\_data:

# Print the file object.

print(election\_data)

When we run this file in the VS Code terminal, the output is similar to when we used the direct path for the file variable:

<open file 'Resources/election\_results.csv', mode 'r' at 0x10479c780>

**IMPORTANT**

You'll notice that we made comments before the code to explain what we were doing. This is a good practice in coding. Not only will this help others reading your code, but it will also help you refresh your memory if you have to revisit your code a few months down the line.

**NOTE**

For more information, see the [documentation on file and directory access (Links to an external site.)](https://docs.python.org/3.7/library/os.path.html).

**Class 3.2**

**Class view 3.2 Start here 1:14:29**

The activities in this class will complement Lessons **3.3.1: Import and Inspect the Data** through **3.6.3: Write the Winning Candidate's Results to a Text File**. The students will benefit from these activities if they‘ve progressed through these lessons, which cover the following concepts, techniques, and tasks:

* Open and read a CSV file
* Write data to a text file
* Create variables
* Perform calculations
* Use conditional statements
* Use repetition statements

Nick – tip – make sure you are careful with parentheses and conditionals

Working with CSV files

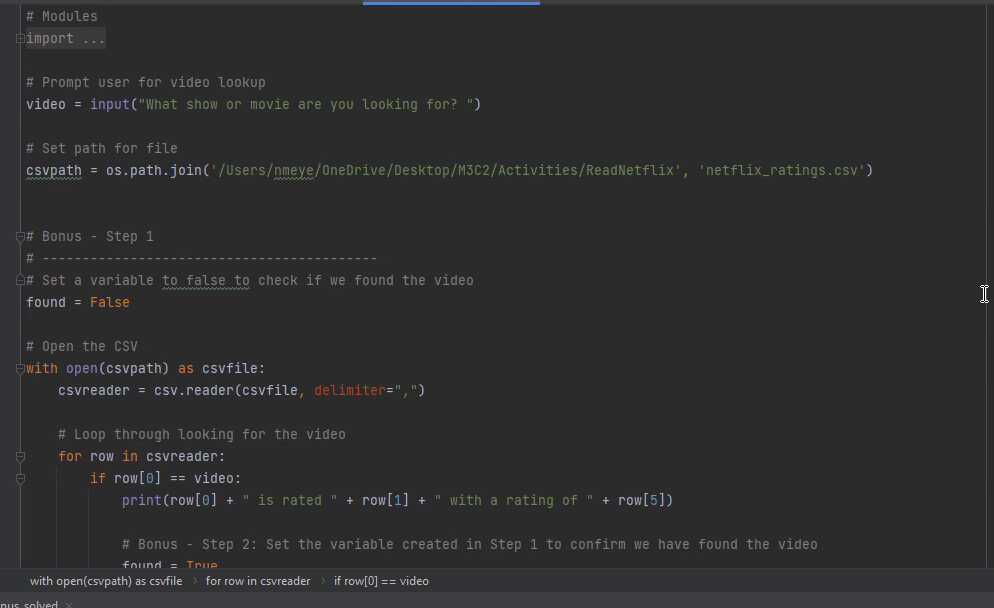
Steps – 1. Find 2. Open 3. Review

In Python – typically use OS import to open

“Next” function to eliminate header

Csv\_header= next(csvreader)

Last module in class 3.2



Challenge – Module 3

Step 1 – Comparing files – Pypoll.py to PyPoll\_Challenge\_starter\_code.py

44: total\_votes + = 1 or total\_votes = total\_votes = 1

88: new line in “Challenge”: f”County Votes:\n”)

117: votes = candidate\_votes[candidate\_name] or votes = candidate\_votes.get(candidate\_name)

Step 2 – Run both files and copy results

C:\Users\RFNichol\AppData\Local\Programs\Python\Python37\python.exe C:/Users/RFNichol/Election\_Analysis/PyPoll.py

Election Results

-------------------------

Total Votes: 369,711

-------------------------

Charles Casper Stockham: 23.0% (85,213)

Diana DeGette: 73.8% (272,892)

Raymon Anthony Doane: 3.1% (11,606)

-------------------------

Winner: Diana DeGette

Winning Vote Count: 272,892

Winning Percentage: 73.8%

-------------------------

Process finished with exit code 0

# End of PyPoll.py

C:\Users\RFNichol\AppData\Local\Programs\Python\Python37\python.exe C:/Users/RFNichol/Election\_Analysis/PyPoll.py

Election Results

-------------------------

Total Votes: 369,711

-------------------------

Charles Casper Stockham: 23.0% (85,213)

Diana DeGette: 73.8% (272,892)

Raymon Anthony Doane: 3.1% (11,606)

-------------------------

Winner: Diana DeGette

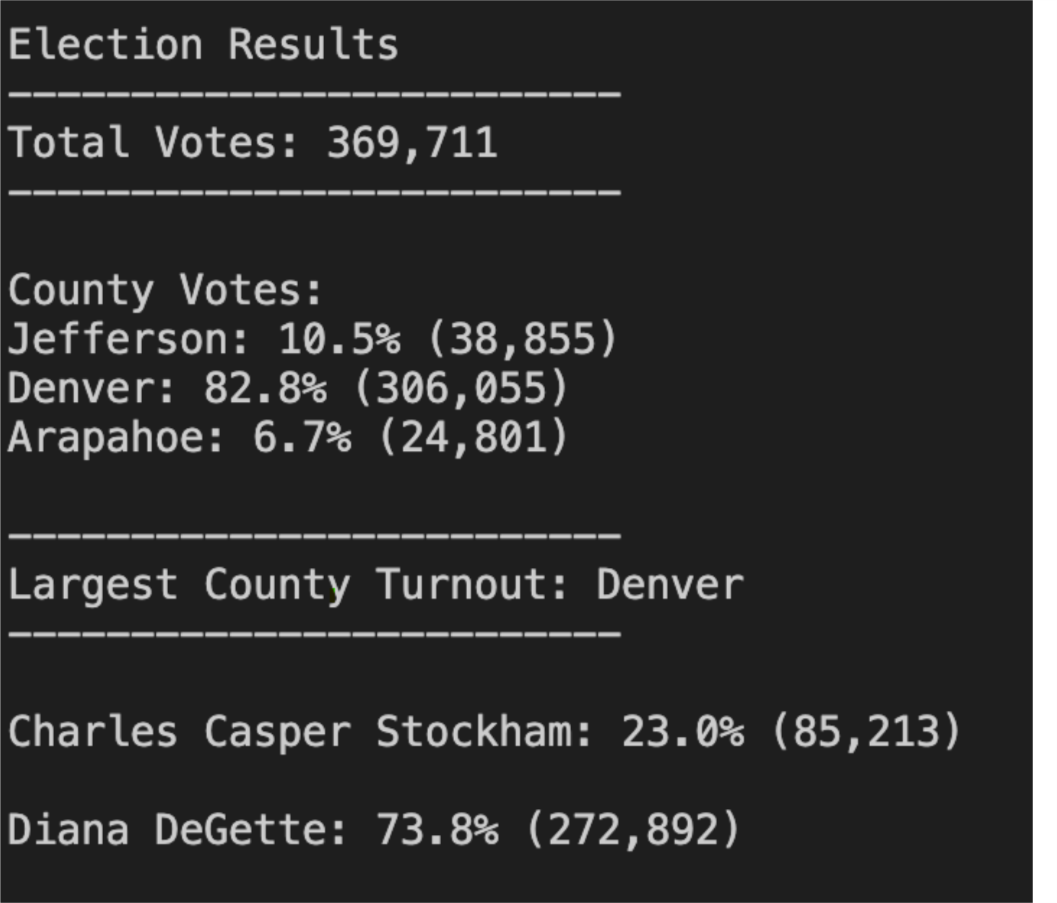
Winning Vote Count: 272,892

Winning Percentage: 73.8%

-------------------------

Process finished with exit code 0

End of PyPoll\_Challenge\_starter\_code.py



Step 3 – Follow steps in challenge

4b – adding values to counties\_dict

candidate\_options = [] # This is a list

candidate\_votes = {} # This is dictionary

candidate\_name # candidate\_name = row[2] extracts candidate name from each row

counties\_list = [] # List of counties

counties\_votes = {} #This is dictionary, key should be county\_name, value should votes cast

counties\_name # counties\_name = row[1] extracts county name from each row

votes = candidate\_votes.get(candidate\_name) #variable to track candidate’s votes

c\_votes = counties\_votes.get(counties\_name) #variable to track counties’ votes

vote\_percentage

c\_vote\_percentage

winning\_candidate # variable to track winning candidate

winning\_count # variable to track winning candidate’s votes

winning\_percentage #variable to track winning candidate’s percentage of votes

winning\_county # variable to track winning county

winning\_county\_vote #variable to track winning county vote

winning-county\_percentage # variable to track winning county vote percentage

Question – how do you / what is process to see if a term has been used before

Next step – clean up code after 103 print(counties\_results)

Question 6e

Struggling with 6f – pencil it out

No results – why

Check logic on each parameter

Counties\_list – should mimic candidate\_options 3 appearances

If candidate\_name not in candidate\_options

If counties\_name not in counties\_list # confirmed

Candidate\_options.append(candidate\_name)

Counties\_list.append(counties\_name) #confirmed

Counties\_votes – should mimic candidate\_votes

Candidate\_votes = {}

Counties\_votes = {}

Candidate\_votes[candidate\_name] = 0

Counties\_votes[counties\_name] = 0

Candidate\_votes[candidate\_name] +=1

Counties\_votes[counties\_name] += 1

For candidate\_name in candidate\_votes

For counties\_name in counties\_votes

Votes = candidate\_votes.get(candidate\_name)

C\_votes = counties\_votes.get(counties\_name)

Winning\_county should mimic Winning\_candidate

Winning\_candidate = “”

Winning\_county = “”

Winning\_candidate = candidate\_name

Winning\_county = counties\_name

Winning\_candidate\_summary = ()

Winning\_county\_summary = ()

f”Winner: {winning\_candidate}\n”

f”Largest County Turnout: {winning\_county}\n”

txt.file.write(winning\_candidate\_summary)

txt.file.write(winning\_county\_summary)

winning\_county\_vote mimics winning\_count

if (votes > winning\_count) and (vote\_percentage > winning\_percentage)

if(c\_votes > winning\_county\_votes) and c\_vote\_percentage > winning\_county\_percentage)

winning\_count = votes

winning\_county\_vote = c\_votes

f"Winning Vote Count: {winning\_count:,}\n"

f"Largest County Turnout: {winning\_county}\n"

winning\_county\_percentage mimics winning\_percentage

winning\_percentage = 0

winning\_county\_percentage = 0

v\_vote\_percentage mimics vote\_percentage

if (votes > winning\_count) and (vote\_percentage > winning\_percentage):

if (c\_votes > winning\_county\_vote) and (c\_vote\_\_percentage > winning\_county\_percentage):

winning\_percentage = vote\_percentage

winning\_county\_percentage = c\_vote\_percentage

f"Winning Percentage: {winning\_percentage:.1f}%\n"

PyPoll Reference “sierrah154” PyPoll\_Challenge

Removed this line

Removed: from pip.\_internal.models import candidate

**Module 4 PyCitySchools with Pandas**

**What You Will Learn**

By the end of this module, you will be able to:

* Open Jupyter Notebook files from local directories using a development environment.
* Read an external CSV file into a DataFrame.
* Format a DataFrame column.
* Determine data types of row values in a DataFrame.
* Retrieve data from specific columns of a DataFrame.
* Merge, filter, slice, and sort a DataFrame.
* Apply the groupby() function to a DataFrame.
* Use multiple methods to perform a function on a DataFrame.
* Perform mathematical calculations on columns of a DataFrame or Series.

**Planning Your Schedule**

Here's a quick look at the lessons and assignments you'll cover in this module. You can use the time estimates to help pace your learning and plan your schedule.

* Introduction to Module 4 (30 minutes)
* Anaconda Installation and Jupyter Notebook (1 hour)
* Creating and Activating A Development Environment (30 minutes)
* Working with Jupyter Notebook and Pandas (1 hour 30 minutes)
* Convert CSV Files to a Pandas DataFrame (1 hour)
* Exploring the Data (1 hour 30 minutes)
* Verify the Clean Student Data (15 minutes)
* Generate the School District Summary (1 hour 30 minutes)
* Generate the School Summary (1 hour 30 minutes)
* High and Low Performing Schools (1 hour)
* Average Math and Reading Scores by Grade (1 hour)
* Group Scores by School Spending per Student(1 hour)
* Group Scores by School Size (1 hour)
* Group Scores by School Type (30 minutes)
* Application (5 hours)

For class tonight: 4.1

* The activities in this class will complement Lessons 4.0.1: Using Pandas and Jupyter Notebook through 4.8.1: Set the Index to the School Name. The students will benefit from these activities if they‘ve progressed through these lessons, which cover the following concepts, techniques, and tasks:
* Create a conda environment
* Create a Jupyter Notebook file
* Understand the difference between a data Series and a DataFrame
* Load and read a CSV file using Pandas
* Use Pandas functions and methods on DataFrames
* Merge DataFrames
* Format values in DataFrame columns

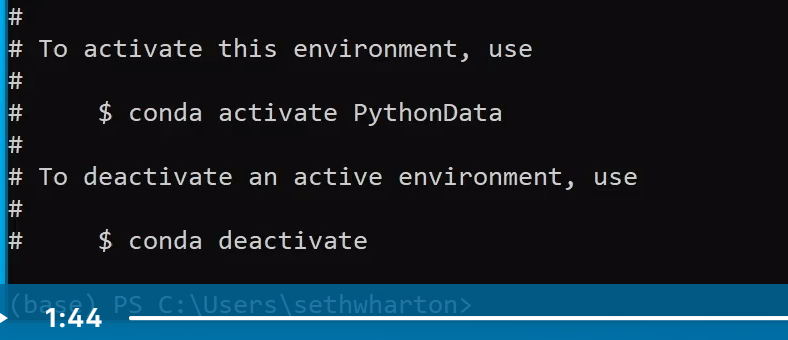
[2:41](https://osuvirtdatapt-wnh9099.slack.com/archives/C02K60VU5M3/p1638819704026700)

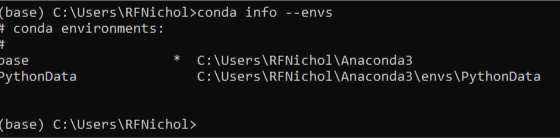
For Wednesday: 4.2

* The activities in this class will complement Lessons 4.8.1: Set the Index to the School Name through 4.13.2: Create a DataFrame for the Scores by School Type. The students will benefit from these activities if they‘ve progressed through these lessons, which cover the following concepts, techniques, and tasks:
* Group data into similar records and perform calculations on the grouped data
* Filter a DataFrame based on a condition using logical operators
* Combine series to create a DataFrame
* Segment and sort values into discrete bins
* Format values in DataFrame columns

**Python 3.7**

**Setting Up Virtual Environment**

****

****

**Start up at 4.7.3**

**Class 4.2**



[**Xavier Jackson**](https://app.slack.com/team/U02LW76BVL3)  [7:12 PM](https://osuvirtdatapt-wnh9099.slack.com/archives/C02KYL1LWSC/p1639008779104600)

Portland CrimeInstructions

* Read in the csv using Pandas and print out the DataFrame that is returned.
* Get a count of rows within the DataFrame in order to determine if there are any null values.
* Drop the rows which contain null values.
* Search through the "Offense Type" column and "replace" any similar values with one consistent value.

[**Xavier Jackson**](https://app.slack.com/team/U02LW76BVL3)  [7:49 PM](https://osuvirtdatapt-wnh9099.slack.com/archives/C02KYL1LWSC/p1639010970105300)

Good MoviesInstructions

* Use Pandas to load and display the CSV provided in Resources.
* List all the columns in the data set.
* We're only interested in IMDb data, so create a new table that takes the Film and all the columns relating to IMDB.
* Filter out only the good movies—i.e., any film with an IMDb score greater than or equal to 7 and remove the norm ratings.
* Find less popular movies that you may not have heard about - i.e., anything with under 20K votes

**Challenge 4**

**Reference - https://github.com/xguilxr/PyCitySchools\_Challenge**

**Headers**

1 # Add the Pandas dependency.

2 # Files to load.

3 # Read the school data file and store it in a Pandas DataFrame. [school\_data-df = …]

4 # Read the student data file and store it in a Pandas DataFrame. [student\_data-df = …]

5 # Determine if there are any missing values in the school data.

6 # Determine if there are any missing values in the student data.

7 # Determine if there are any missing values in the school data.

8 # Determine if there are any missing values in the student data.

9 # Determine if there are any missing values in the student data.

10 # Determine if there are not any missing values in the school data.

11 # Determine if there are not any missing values in the student data.

12 # Determine data types for the school DataFrame.

13 # Determine data types for the student DataFrame.

14 # Add each prefix and suffix to remove to a list.

15 # Iterate through the words in the "prefixes\_suffixes" list and replace them with an empty space, "".

16 # Display the student\_data Dataframe

17 # Combine the data into a single dataset, i.e.merge dataframes. [school\_data\_complete\_df = pd.merge(…)]

18 # Get the total number of students.

19 # Get the total number of students, alternative method.

20 # Calculate the total number of schools.

21 # Calculate the total number of schools, alternative method.

22 # Calculate the total budget.

23 # Calculate the average reading score. [average\_reading\_score]

24 # Calculate the average math score. [average\_math\_score]

25 # Is the student passing math?

26 # Get all the students who are passing math in a new DataFrame.

27 # Get all the students that are passing reading in a new DataFrame.

28 # Calculate the number of students passing math.

29 # Calculate the number of students passing reading.

30 # Calculate the percent that passed math.

31 # Print the passing\_math\_percentage.

32 # Calculate the percent that passed reading.

33 print(passing\_reading\_percentage)

34 # Calculate the students who passed both math and reading.

35 # Calculate the number of students who passed both math and reading.

36 # Calculate the overall passing percentage.

37 # Adding a list of values with keys to create a new DataFrame. [district\_summary\_df]

38 # Determine the School Type

39 # Series of calculations on “per\_...” variable.” [Groupby is used]

40 # Format "Total Budget" to have the comma for a thousands separator, a decimal separator, and a "$".

41 # Create the DataFrame [per\_school\_summary\_df]

42 # Format the Total School Budget and the Per Student Budget columns.

43 # Format the "Total Students" to have the comma for a thousands separator.

44 # Format the columns. [in district\_summary\_df]

45 # Reorder the columns in the order you want them to appear.

46 # Determine the school type.

47 # Add the per\_school\_types into a DataFrame for testing.

48 # Calculate the total student count. [per\_school\_counts]

49 # Calculate the total student count, alternate method [concern that using same variable with different equation]

50 # Calculate the total student count. [concern that using same variable with different equation]

51 # Calculate the total school budget.

52 # Calculate the per capita spending.

53 # Calculate the math scores.

54 # Calculate the average math scores. [per\_school\_averages]

55 # Calculate the average test scores. [per\_school\_math; per\_school\_reading]

56 # print(per\_school\_math)

57 # print(per\_school\_reading)

58 # Calculate the passing scores by creating a filtered DataFrame.

59 print(per\_school\_passing\_reading)

60 # Calculate the number of students who passed both math and reading.

61 # Calculate the overall passing percentage.

62 # Reorder the columns in the order you want them to appear. # Assign district summary df the new column order.

63 # Sort and show top five schools.

64 # Sort and show bottom five schools.

65 # Create a grade level DataFrames.

66 # (ninth\_graders).head

67 # # Group each grade level DataFrame by the school name for the average math score.

68 # print(ninth\_grade\_math\_scores)

69 # print(eleventh\_grade\_math\_scores)

70 # Group each grade level DataFrame by the school name for the average reading score.

71 # print(ninth\_grade\_reading\_scores)

72 print(twelfth\_grade\_reading\_scores)

73 # Combine each grade level Series for average math scores by school into a single DataFrame.

74 # Combine each grade level Series for average reading scores by school into a single DataFrame.

75 # Format each grade column.

76 # Make sure the columns are in the correct order.

# Remove the index name. # Display the DataFrame.

77 # Format each grade column. # Make sure columns are in correct order. # Remove the index name and # Display the dataframe.

78 # Get the descriptive statistics for the per\_school\_capita.

79 # Cut the per\_school\_capita into the spending ranges.

80 # Cut the per\_school\_capita into the spending ranges, alternative one

81 # Cut the per\_school\_capita into the spending ranges, alternative two.

82 # Cut the per\_school\_capita into the spending ranges, alternative three.

83 # Establish the spending bins and group names.

84 # Categorize spending based on the bins.

85 # Calculate averages for the desired columns.

86 # Assemble into DataFrame.

87 # Formatting

88 # Establish the bins.

89 # Categorize spending based on the bins.

90 # Calculate averages for the desired columns.

91 # Assemble into DataFrame.

92 # Formatting.

93 # Calculate averages for the desired columns.

94 # Assemble into DataFrame.

95 # Formatting

**Have to reinsert**

Inserted calculations –

“Calculate the overall passing percentage”

Per\_overall\_passing\_percentage = per\_passing\_math\_reading / per\_school\_counts \* 100

Overall\_passing\_percentage = overall\_passing\_math\_reading\_count / student\_count \* 100

Possible functions to use

Replace() – cleaning\_students\_names

For **word** in [variable]…str.replace(**word,\*\*\*) # \*\*\* is whatever to replace**

Put the cleaned students' names in another list. Possible need for student scores that were replaced

student\_names = student\_data\_df["student\_name"].tolist()

student\_names

If the name is greater than or equal to 3, add the name to the list.

for name in student\_names:

if len(name.split()) >= 3:

students\_fixed.append(name)

# Get the length of the students' names that are greater than or equal to 3.

len(students\_fixed)

# Define a function that calculates the percentage of students that passed both

# math and reading and returns the passing percentage when the function is called.

def passing\_math\_percent(pass\_math\_count, student\_count):

return pass\_math\_count / float(student\_count) \* 100

# Call the function.

passing\_math\_percent(passing\_math\_count, total\_student\_count)

# A list of my grades.

my\_grades = ['B', 'C', 'B' , 'D']

# Convert the my\_grades to a Series

my\_grades = pd.Series(my\_grades)

my\_grades

0 B

1 C

2 B

3 D

dtype: object

# Using the format() function.

my\_grades = [92.34, 84.56, 86.78, 98.32]

for grade in my\_grades:

print("{:.0f}".format(grade))

92

85

87

98

# Formatting

per\_school\_summary\_df["Average Math Score"] = per\_school\_summary\_df["Average Math Score"].map("{:.1f}".format)

per\_school\_summary\_df["Average Reading Score"] = per\_school\_summary\_df["Average Reading Score"].map("{:.1f}".format)

per\_school\_summary\_df["% Passing Math"] = per\_school\_summary\_df["% Passing Math"].map("{:.0f}".format)

per\_school\_summary\_df["% Passing Reading"] = per\_school\_summary\_df["% Passing Reading"].map("{:.0f}".format)

per\_school\_summary\_df["% Overall Passing"] = per\_school\_summary\_df["% Overall Passing"].map("{:.0f}".format)

per\_school\_summary\_df

Reading Math



Taking out 9th graders 83.89608 83.35094

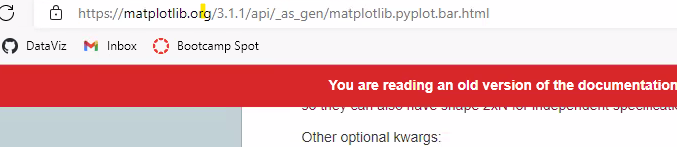
**Module 5**

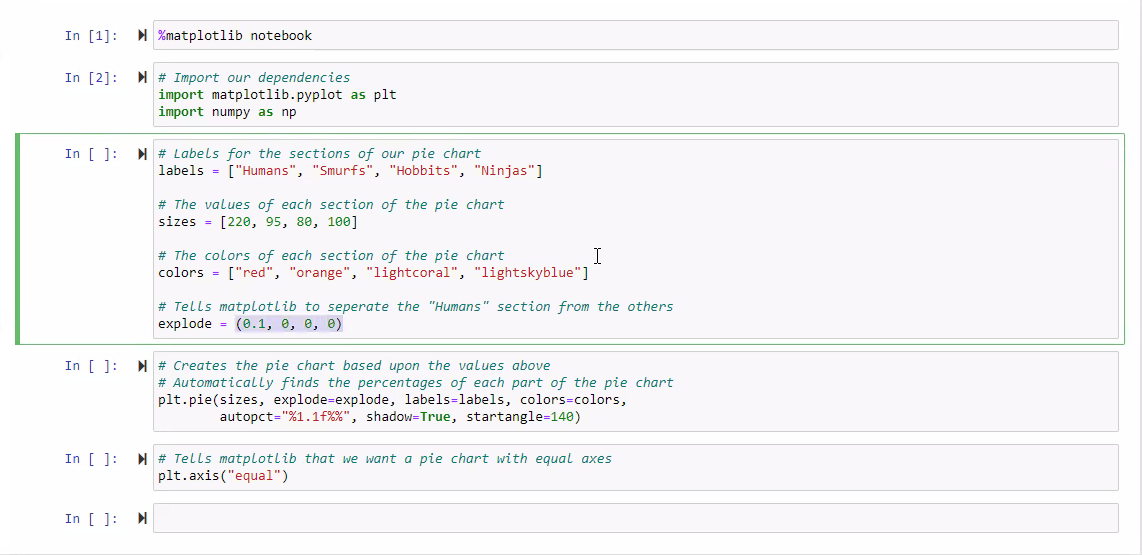
## Annotate Charts Using the Object-Oriented Interface

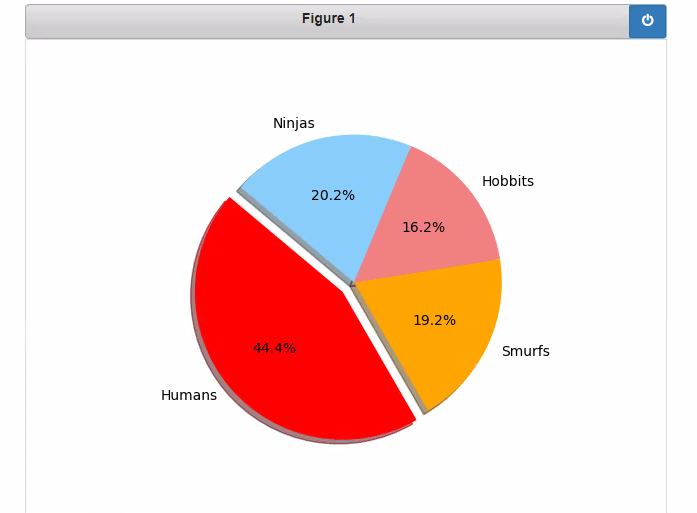
Annotating graphs using the object-oriented interface is similar to using the MATLAB approach, with a slightly different syntax.

Here are a few methods you can use to annotate your chart using the object-oriented interface method:

|  |  |
| --- | --- |
| **Matplotlib Object-Oriented Functions** | **Feature** |
| fig, ax = plt.subplots(figsize=(w, h)) | Change the size of the figure in pixels. Add this in the subplots() function. |
| ax.plot(x, y, label='line') | Add a label that will be added to the legend. |
| ax.set\_ylim(min, max) | Sets the min and max range of the y-axis. |
| ax.set\_xlim(min, max) | Sets the min and max range of the x-axis. |
| ax.set\_xlabel('x label') | Add a label to the x-axis. |
| ax.set\_ylabel('y label') | Add a label to the y-axis. |
| ax.set\_title("Title") | Add a title. |
| ax.legend() | Add a legend. |
| ax.grid() | Add a grid to the chart. |
| \*\* plt.savefig("add a path and figure extension") | Saves the figure with the given extension. Added at the end of your script. |
|  |  |

****

****

****

**Notes 5.1**

* The activities in this class will complement Lessons **5.0.1: Visualizing Ride-Sharing Data** through **5.1.9: Chart Extras**. The students will benefit from these activities if they‘ve progressed through these lessons, which cover the following concepts, techniques, and tasks:
* Create line, bar, pie, and scatter charts
* Add a title and x- and y-axes labels
* Add error bars and minor ticks
* A note on potential errors caused by the Matplotlib library:
* The %matplotlib notebook magic command for the displaying the notebook on the backend is used in a number of activities. It not only makes a plot interactive, but it also allows it to be updated after the initial plot. If students encounter weirdness during the activities, check to see that they use this line before importing the plotting libraries.
* On Windows the %matplotlib notebook command may have to be called twice to ensure that plots can be updated after the intiatl plot is made.
* %matplotlib notebook
* %matplotlib notebook
* Ideally, students should update their Matplotlib to version 2.2 or later. A bug in earlier versions resizes plots after an image is exported with the savefig() method.
* Encourage students to develop the habit of exploring the Matplotlib documentation. A large part of the process of developing plots with the library is reading [examples](http://matplotlib.org/examples/index.html), so it is important for them to become accustomed to this workflow.

**Notes 5.2**  
The activities in this class will complement Lessons **5.5.1: Get the Percentage of Fares for Each City Type** through **5.7.2: Pie Chart for the Percentage of Drivers for Each City Type**. The students will benefit from these activities if they‘ve progressed through these lessons, which cover the following concepts, techniques, and tasks:

* Use Pandas functions and methods on DataFrames
* Group data into similar records and perform calculations on the grouped data
* Filter a DataFrame based on a condition using logical operators
* Create pie and bubble charts
* Add and modify chart features

**Module 5.2.2 – Overview of the Project**

You come up with the following list of steps and deliverables:

* Import your data into a Pandas DataFrame.
* Merge your DataFrames.
* Create a bubble chart that showcases the average fare versus the total number of rides with bubble size based on the total number of drivers for each city type, including urban, suburban, and rural.
* Determine the mean, median, and mode for the following:
  + The total number of rides for each city type.
  + The average fares for each city type.
  + The total number of drivers for each city type.
* Create box-and-whisker plots that visualize each of the following to determine if there are any outliers:
  + The number of rides for each city type.
  + The fares for each city type.
  + The number of drivers for each city type.
* Create a pie chart that visualizes each of the following data for each city type:
  + The percent of total fares.
  + The percent of total rides.
  + The percent of total drivers.

Omar has approved the project scope. It's time to get to work!

Challenge 5

V. Isualize has given you and Omar a brand-new assignment. Using your Python skills and knowledge of Pandas, you’ll create a summary DataFrame of the ride-sharing data by city type. Then, using Pandas and Matplotlib, you’ll create a multiple-line graph that shows the total weekly fares for each city type. Finally, you’ll submit a written report that summarizes how the data differs by city type and how those differences can be used by decision-makers at PyBer.

# Reference: [wanghen21](https://github.com/wanghen21), “Pyber\_challenge.ipynb”

**Location of the PyBer\_Analysis folder**

**http://localhost:8888/tree/PyBer\_Analysis**

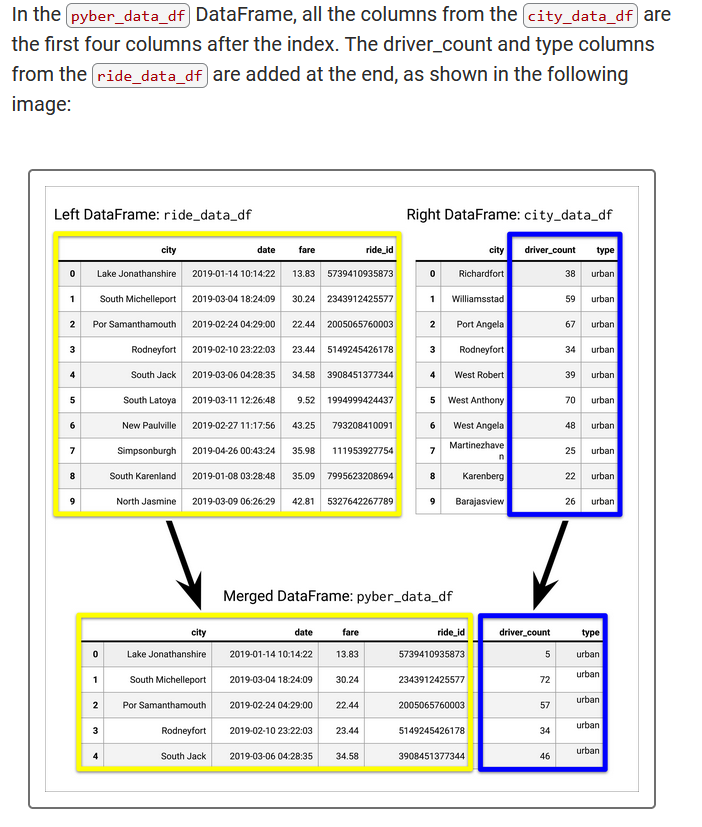
**Merging DataFrame**

# Combine the data into a single dataset

pyber\_data\_df = pd.merge(ride\_data\_df, city\_data\_df, how="left", on=["city", "city"])

# Display the DataFrame

pyber\_data\_df.head()

****

**Module 6**

Jack is the head of analysis for the user interface team. He works for PlanMyTrip, a leading travel technology company. Jack asked me to help him collect and present data for customers via the search page, which they will then filter based on their preferred travel criteria in order to find their ideal hotel anywhere. To perform this task. I used a Jupyter Notebook and the city PI module to get the cities for more than 500 random latitudes and longitudes then I performed requests on the open weather map API and retrieve the JSON weather data from these cities. I then added the weather data to the Panda’s dataframe. From there I used Matplotlib to create a series of scatter plots to show the relationship between the latitude and a variety of weather parameters for over 500 cities around the world. As part of the analysis I completed a series of statistical calculations on the data using linear regression on the weather parameters in the Northern and Southern hemispheres. This data helped my team predict the best time of the year for people to plan their vacation. Finally, I exported the data, cleaned it and used the weather data to choose the best cities for a vacation based on certain weather criteria and then mapped these cities using Jupyter G Maps and the Google Places API.

At the most fundamental level, Jack needs help answering a question: How might we provide real-time suggestions for our client's ideal hotels? Your first task was to define what you meant by "ideal." So, over the course of the conversation, you narrowed that to hotels that were (1) within a given range of latitude and longitude and that (2) provided the right kind of weather for the client.

**Basic Project Plan**

Here's an outline of your project plan:

* **Task:** Collect and analyze weather data across cities worldwide.
* **Purpose:** PlanMyTrip will use the data to recommend ideal hotels based on clients' weather preferences.
* **Method:** Create a Pandas DataFrame with 500 or more of the world's unique cities and their weather data in real time. This process will entail collecting, analyzing, and visualizing the data.

Your analysis of the data will be split into three main parts, or stages.

1. **Collect the Data**
   * Use the NumPy module to generate more than 1,500 random latitudes and longitudes.
   * Use the citipy module to list the nearest city to the latitudes and longitudes.
   * Use the OpenWeatherMap API to request the current weather data from each unique city in your list.
   * Parse the JSON data from the API request.
   * Collect the following data from the JSON file and add it to a DataFrame:
     + City, country, and date
     + Latitude and longitude
     + Maximum temperature
     + Humidity
     + Cloudiness
     + Wind speed
2. **Exploratory Analysis with Visualization**
   * Create scatter plots of the weather data for the following comparisons:
     + Latitude versus temperature
     + Latitude versus humidity
     + Latitude versus cloudiness
     + Latitude versus wind speed
   * Determine the correlations for the following weather data:
     + Latitude and temperature
     + Latitude and humidity
     + Latitude and cloudiness
     + Latitude and wind speed
   * Create a series of heatmaps using the Google Maps and Places API that showcases the following:
     + Latitude and temperature
     + Latitude and humidity
     + Latitude and cloudiness
     + Latitude and wind speed
3. **Visualize Travel Data**

Create a heatmap with pop-up markers that can display information on specific cities based on a customer's travel preferences. Complete these steps:

* + Filter the Pandas DataFrame based on user inputs for a minimum and maximum temperature.
  + Create a heatmap for the new DataFrame.
  + Find a hotel from the cities' coordinates using Google's Maps and Places API, and Search Nearby feature.
  + Store the name of the first hotel in the DataFrame.
  + Add pop-up markers to the heatmap that display information about the city, current maximum temperature, and a hotel in the city.

**Create a Jupyter Notebook**

**C:\Users\rfnichol\OneDrive - COOPER TIRE & RUBBER COMPANY\Personal\Data Analytics Boot Camp\Module\_6\_World\_Weather\_Analysis**

**C:/Users/rfnichol/OneDrive - COOPER TIRE & RUBBER COMPANY/Personal/Data Analytics Boot Camp/Module\_6\_World\_Weather\_Analysis**

**/Users/rfnichol/OneDrive - COOPER TIRE & RUBBER COMPANY/Personal/Data Analytics Boot Camp/Module\_6\_World\_Weather\_Analysis**

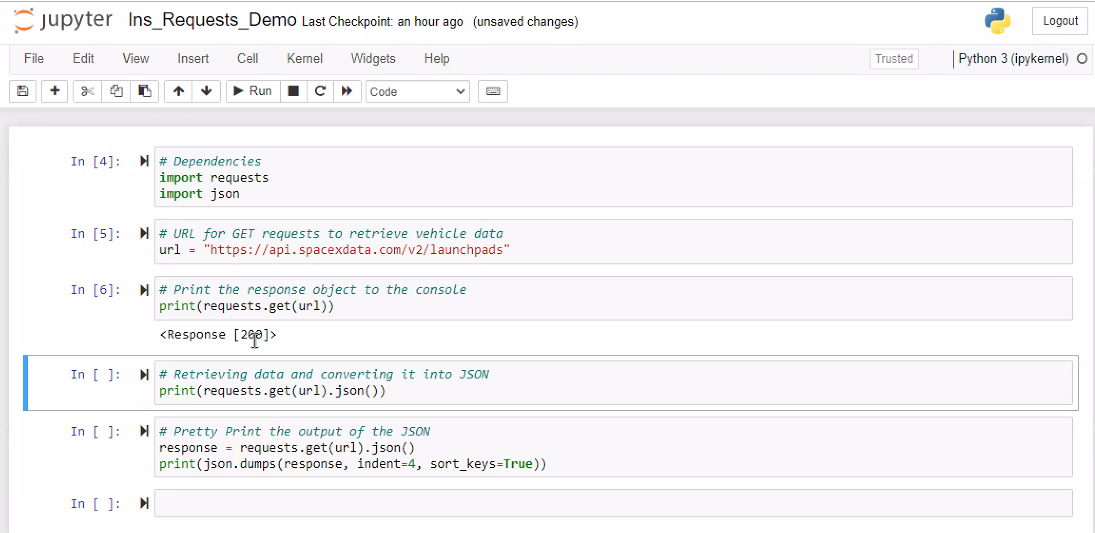
**Class 6.1**

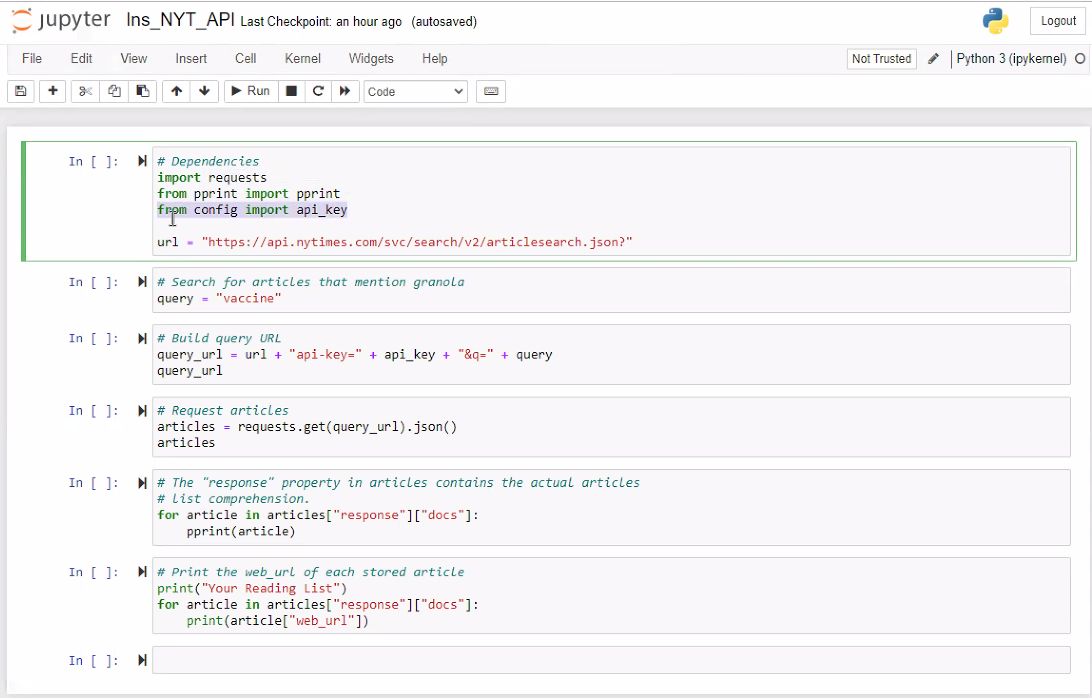
**Notes 6.1**

* The activities in this class will complement lessons **6.2.1 Understanding APIs** through **6.2.7 Create a DataFrame of City Weather Data**. The students will benefit from these activities if they've progressed through these lessons, which cover the following concepts, techniques, and tasks:
* Making API calls using Python's request library
* Traversing JSONs
* Using try-except blocks for error handling
* Making calls to different APIs
* Storing API data in Pandas DataFrames

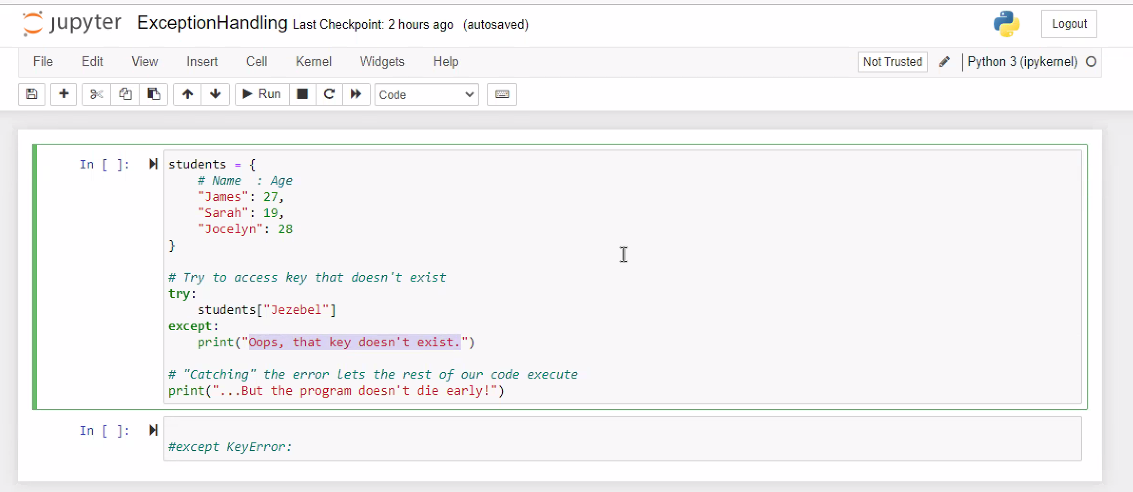
**Notes 6.2**

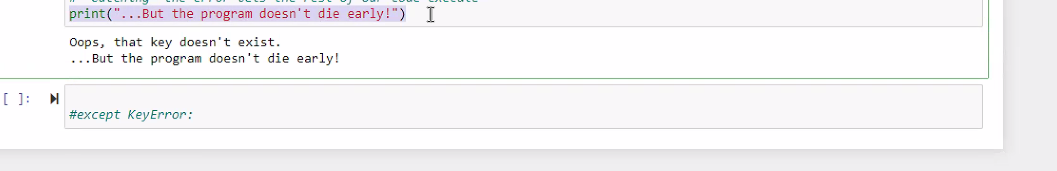
* The activities in this class will complement Lessons **6.5.1: Set Up Google Maps and Places API** through **6.5.4: Map Vacation Criteria**. The students will benefit from these activities if they‘ve progressed through these lessons, which cover the following concepts, techniques, and tasks:
* Making API calls to the Google Maps and Places API
* Using Pandas DataFrames to store data from API calls
* Plotting with Jupyter Gmaps
* Creating Heatmaps with Jupyter Gmaps

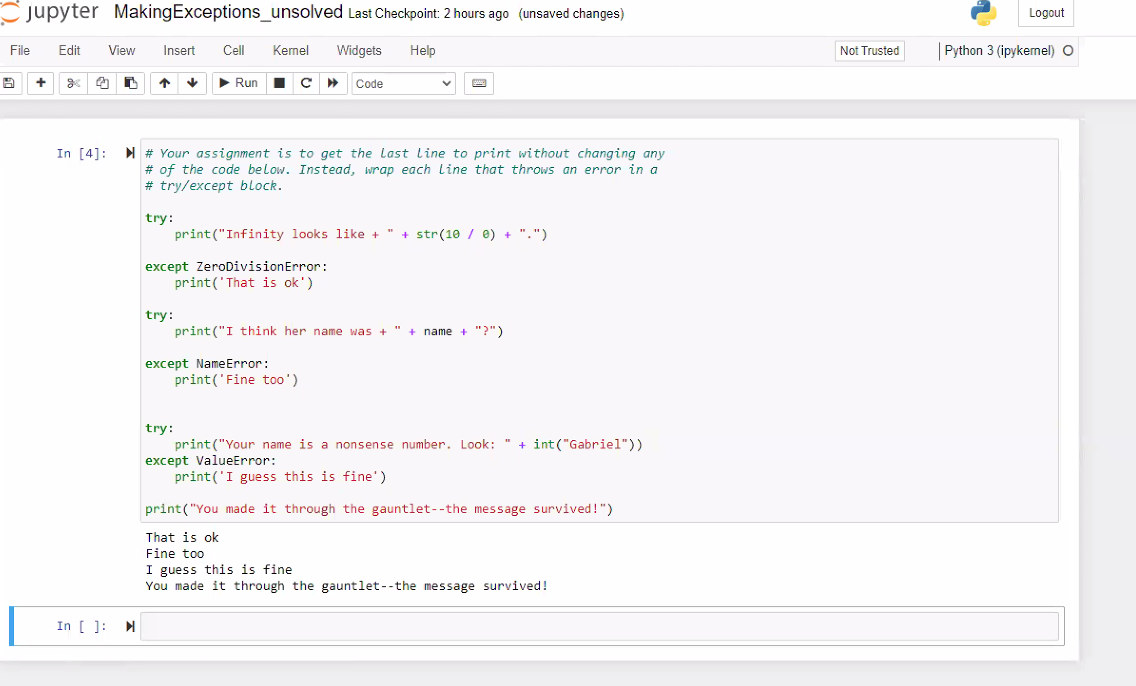
****

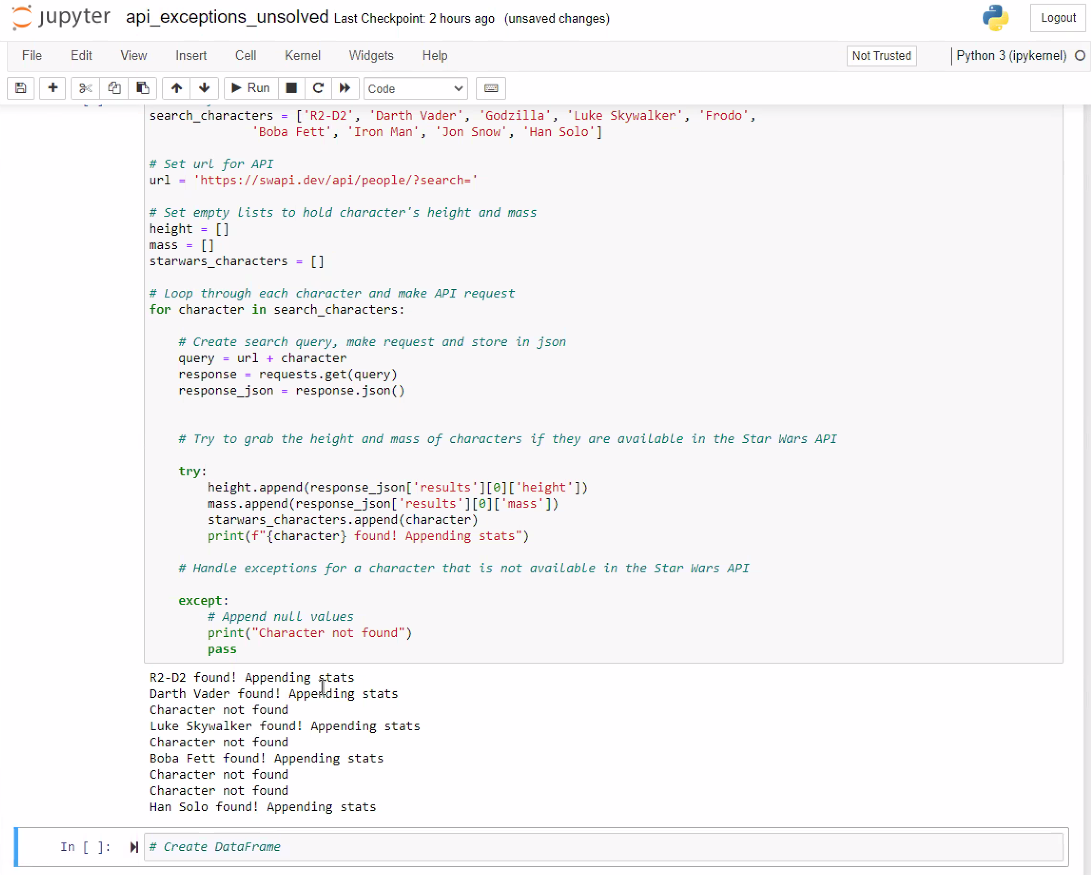
****

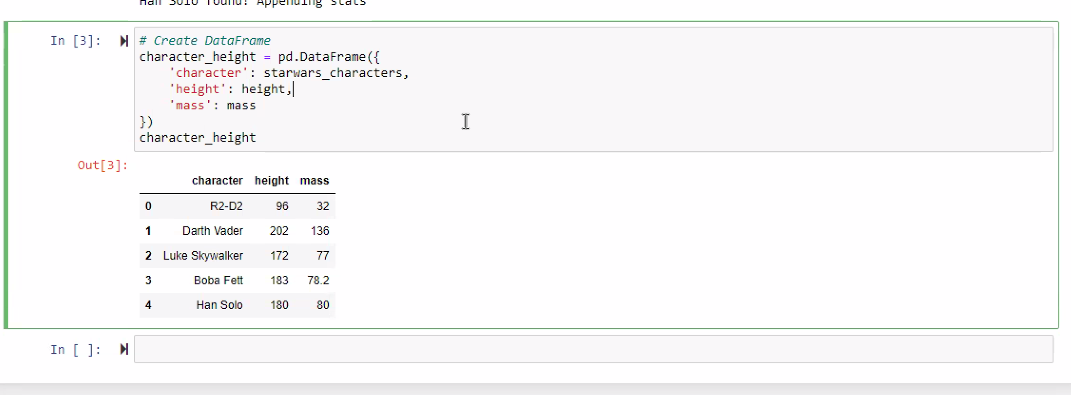
****

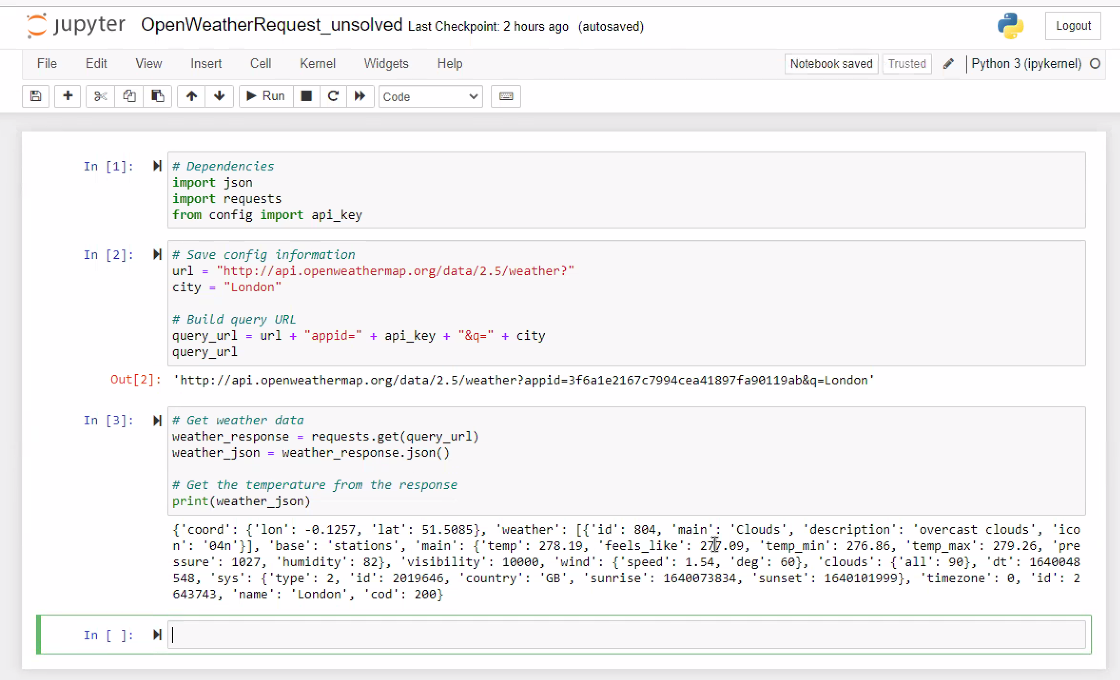
****

****

****

****

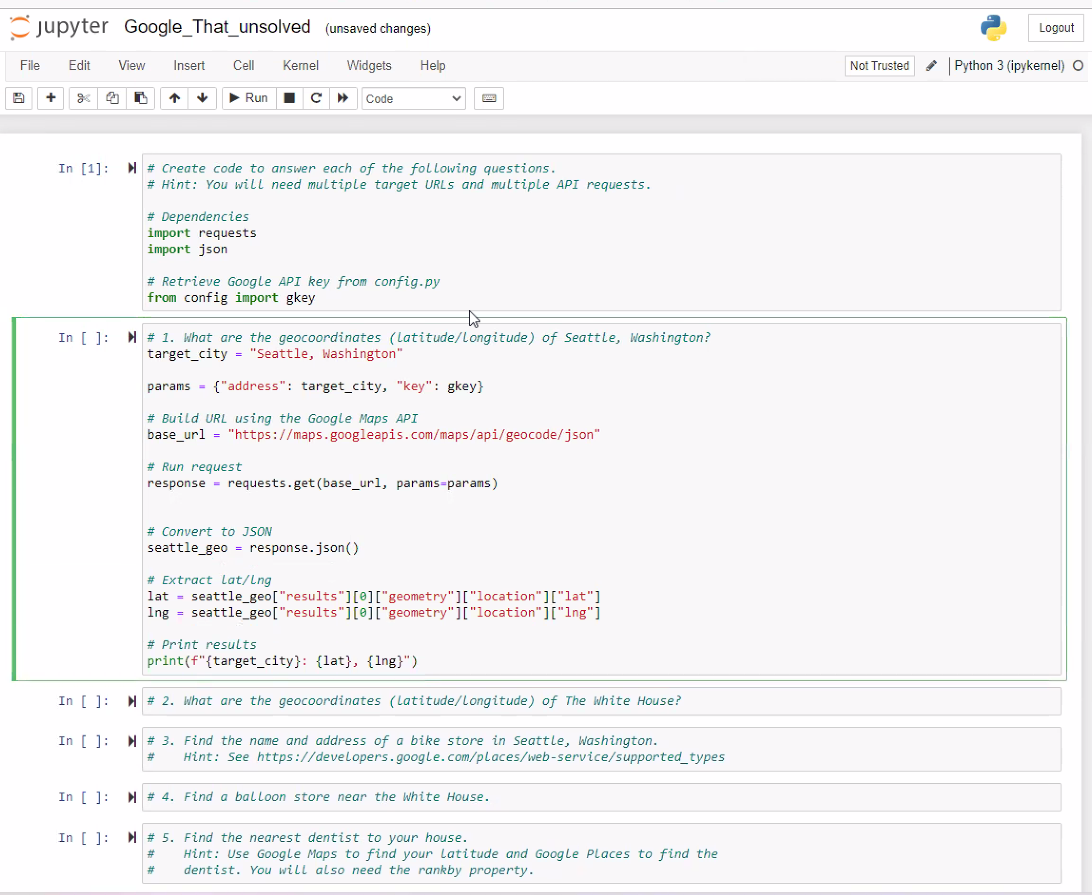
****

****

**Google API Key AIzaSyBb6rNLkvqV04Ompq3eK43qyQH9TYF0rFs**

**IDE – Integrated Development Environment, e.g. PyCharm**

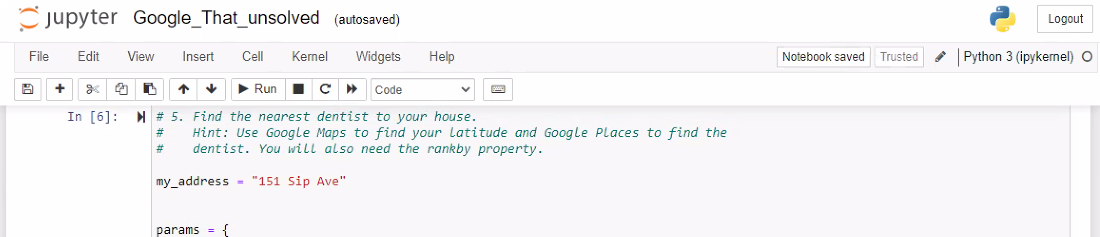
**Class 6.2**

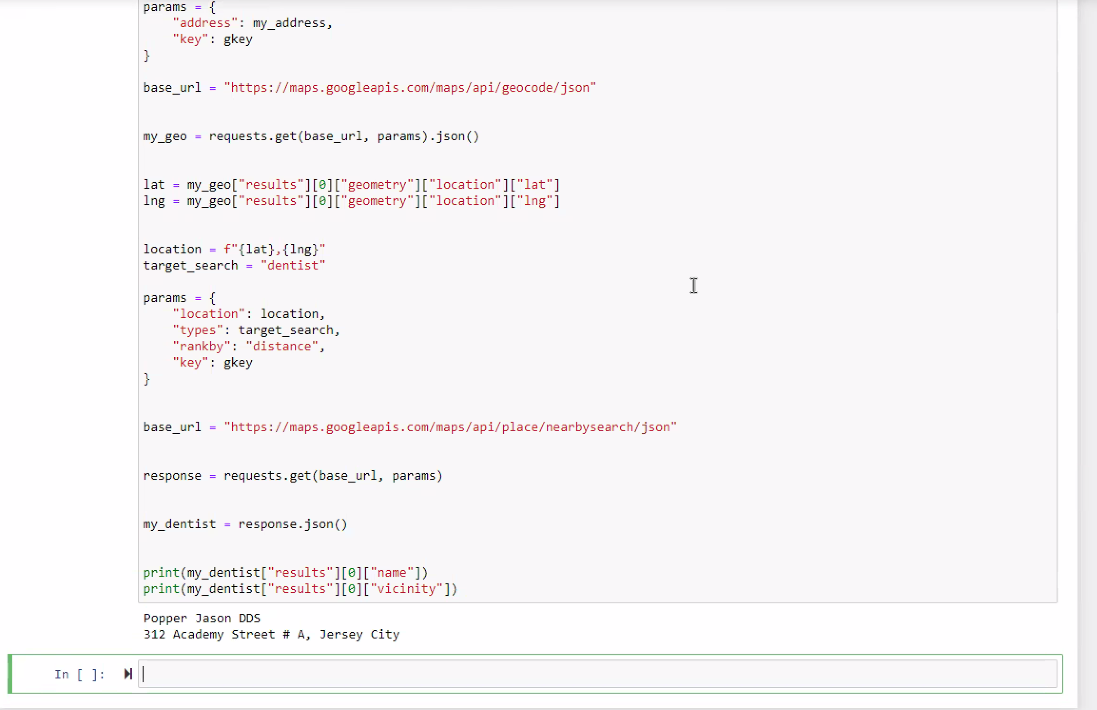
****

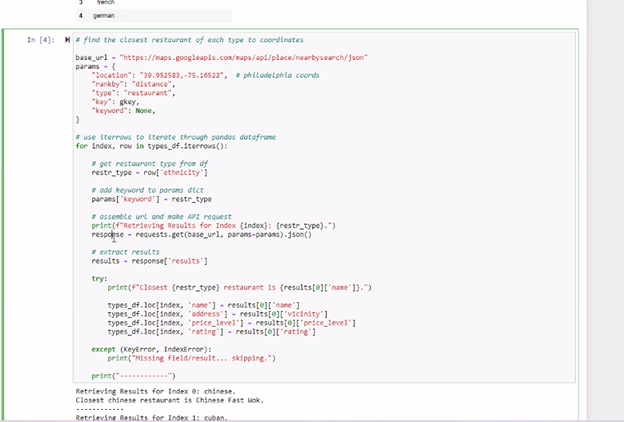
****

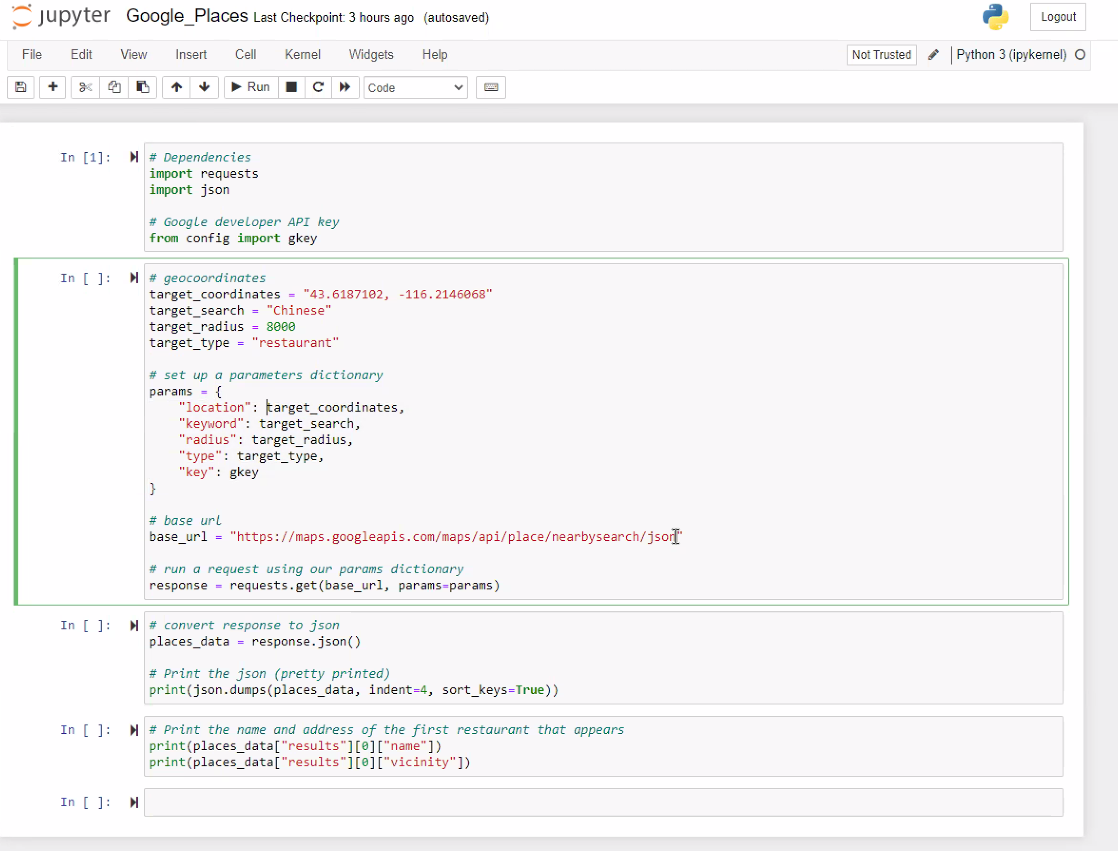
****

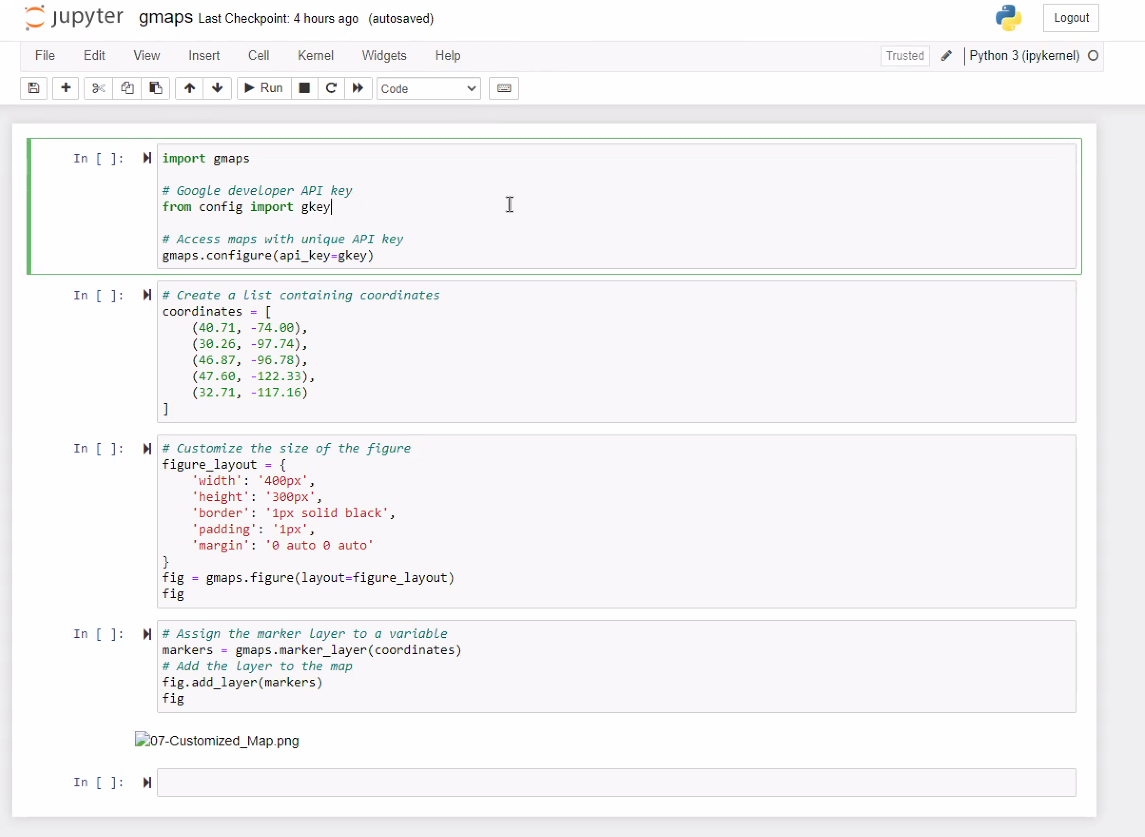
****

****

****



****

****

**Module 6.2.6**

Let's use pseudocode to map out, at a high level, how we will get the weather data for each city for the website.

We will need to do the following:

1. Import our dependencies and initialize counters and an empty list that will hold the weather data.
2. Loop through the cities list.
3. Group the cities in sets of 50 to log the process as we find the weather data for each city.
   * Two counters will be needed here: one to log the city count from 1 to 50, and another for the sets.
4. Build the city\_url or endpoint for each city.
5. Log the URL and the record and set numbers.
6. Make an API request for each city.
7. Parse the JSON weather data for the following:
   * City, country, and date
   * Latitude and longitude
   * Maximum temperature
   * Humidity
   * Cloudiness
   * Wind speed
8. Add the data to a list in a dictionary format and then convert the list to a DataFrame.

Module 6 Code Errors

6.5.1

## Install the gmaps Dependency

## # The below code all starts with a $. This makes the code invalid / wrong syntax

$ conda install -c conda-forge gmaps

$ jupyter nbextension enable --py --sys-prefix widgetsnbextension

$ pip install gmaps

$ jupyter nbextension enable --py --sys-prefix gmaps

6.5.2

# Get the maximum temperature.

max\_temp = city\_data\_df["Max Temp"]

temps = []

for temp in max\_temp:

temps.append(max(temp, 0))

# above code does not work. It did not eliminate negative numbers for max\_temp

**Module 6 challenge**

# Used [cmoconnor17](https://github.com/cmoconnor17) / [Module6\_Challenge](https://github.com/cmoconnor17/Module6_Challenge)

**Module 7 Structured Query Language (SQL, pronounced “Sequel”)**

Class 1 – up to lesson 3.2

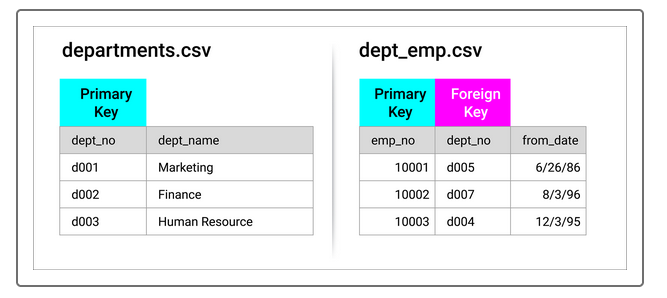
Pewlett Hackard is a large company with several thousand employees. It is looking toward the future in two ways. First, it is offering retirement packages to employees who meet certain criteria and starting to think about which positions will need to filled in the near future. Th number of retirements in the near future will leave thousands of job openings. What would happen to a company if they didn’t look ahead and prepare for this many vacancies? It probably wouldn’t be pretty. Bobby is an up and coming HR analyst whose task it is to perform employee research. Specifically, he need to find answers to the following questions, Who will be retiring in the next few years and how many positions will the company need to fill? This analysis will help future-proof the company by generating a list of all employees eligible for the retirement package. The employee data Bobby needs is only available in the form of six CSV files because the company has been mainly using Excel and VBA to work with their data. But now they have finally decided to update their methods and instead use SQL. My task was to help Bobby build an employee database with SQL by applying your data modeling, engineering and analysis skills.

Review of Data files

**Primary keys**: unique identifier for their dataset, i.e. each row in the table. **dept\_no, emp\_no,**

**Join:** linking the tables through the relationship between tables

**Foreign keys:** references another dataset's primary key.

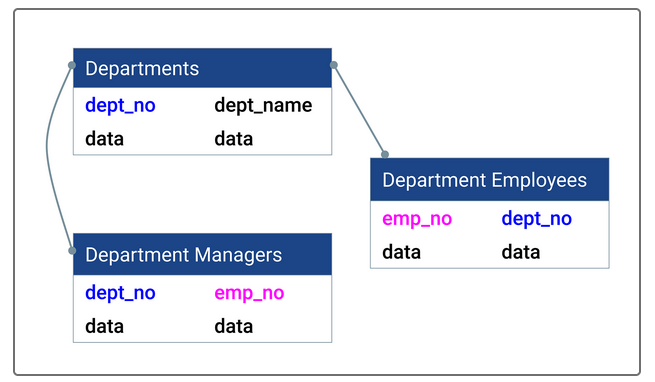
****

## Entity Relationship Diagrams (ERDs)

An **entity relationship diagram (ERD)** is a type of flowchart that highlights different tables and their relationships to each other. The ERD does not include any actual data, but it does capture the following pertinent information from each CSV file:

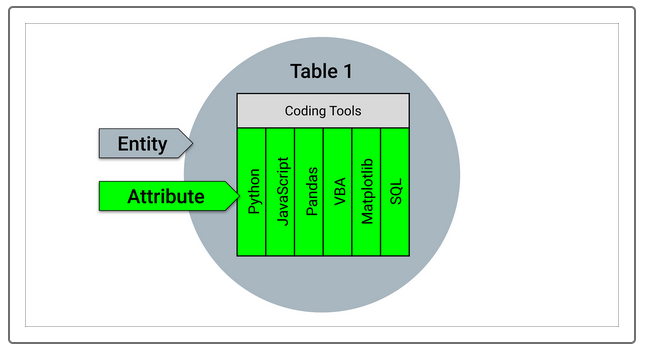
* Primary keys
* Foreign keys
* Data types for each column

The ERD also shows the flow of information from one table to another, as captured in the image below:



In addition to creating new databases, ERDs are used to document existing databases. The visual representation of the tables gives a deeper understanding of the data and the database as a whole.

When creating a diagram, we need to fully understand all of the data being inserted. Database components include tables, known as **entities,** with data, known as **attributes.**

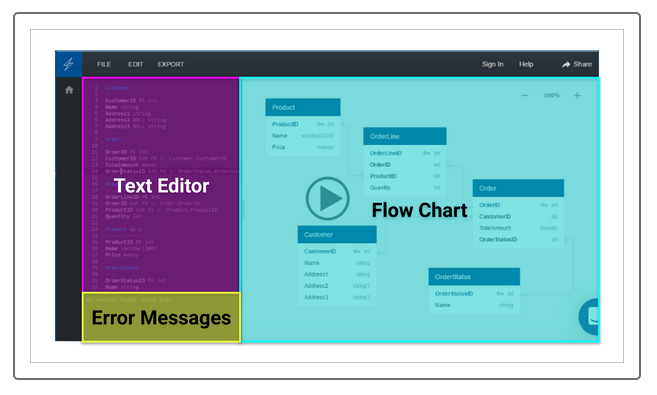


Data types include Booleans, integers, and varying characters (i.e., within a string).

There are three types of ERDs: **conceptual, logical,** and **physical.** Each one builds upon the other—you need the conceptual ERD to build a logical ERD to build a physical ERD. We'll learn how to create ERDs later in this module.

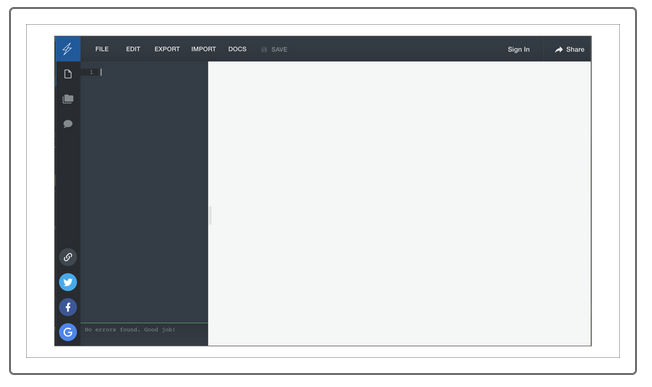
# **7.1.4 Use the Quick Database Diagrams Tools**

Another step is to create a map of the database. This map will show us each table in the database and the flow of data from one table to another. That's right, we're going to make a fancy flowchart. This provides us with an easy reference to the data without actually accessing it. This is called "modeling the data," and we can get started by creating a diagram with an online tool, instead of trying to make one from scratch. Our flow chart will help us navigate through the relationships more easily than if we had all six CSV files open side-by-side.  
  
Using an online tool called Quick Database Diagrams ("Quick DBD" for short), we'll help Bobby start by familiarizing ourselves with the webpage, then create a conceptual ERD.

Now you can view our canvas, split into two sections: a text editor and a flowchart workspace. Like the example previously shown, we'll build our tables using the text editor (on the left), and they'll display as a flow chart on the right.

As we build our tables, the flow chart will update, revealing the connections between the tables. Also, if we make a mistake, it quickly becomes apparent: the message box below the text editor will display any errors as they occur.

Instead of altering the sample, we'll go ahead and delete the text in the text editor to clear the canvas for our own use.



# **7.1.5 Create ERDs**

**We** are ready to map our data. By now, we're comfortable with what we're working with. We have an idea of the data connections, and we're ready to use our new tool: Quick DBD.  
  
There are several ways to refer to the map we're about to create. It's also called a flowchart, an entity relationship diagram, and a schema. We'll be using all of these terms in this module, though "ERD" is the most specific.  
  
There are three forms of ERDs: conceptual, logical, and physical. We'll start by helping Bobby with the most basic of the three, the conceptual diagram. As we add more information to our tables, such as data types and keys, we'll advance through the more complex diagrams.

## Conceptual Diagrams

A **conceptual diagram** is an ERD in its simplest form. To create one, we only need two things: a table name and column headers.

It's simple because we're creating just the concept of the diagram. By covering only the basics, it's easier to capture the main points. If we tried to capture everything at once (data types, location of the primary and foreign keys, etc.), we're more likely to overlook a crucial item.

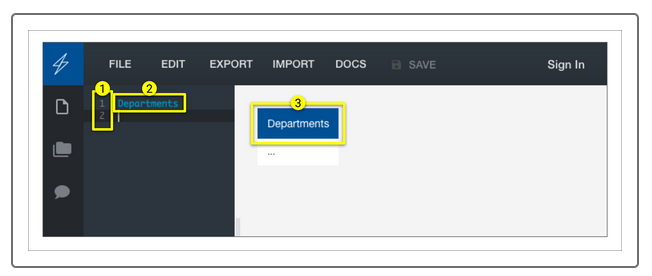
Returning to the Quick DBD website, let's create the blueprint (or schema) for our first table. The schema generates the actual diagram, and we're creating schema instead of coding in the text editor.

**important**

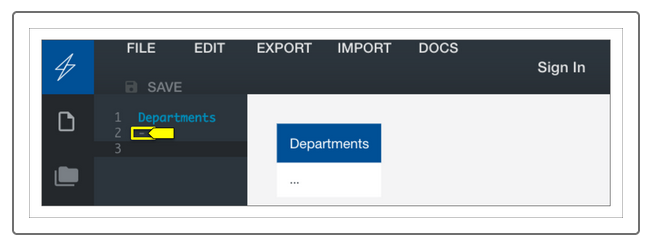
**Schema** is a term that will come up often while working in SQL and its extensions (such as MySQL, PostgreSQL, and many others). It references the design of the database, and specifically how the tables and their relationships are mapped out.

We'll use departments.csv for the first table, so let's review the worksheet again, as we'll need to know the column names.

The first thing to create in the text editor is the name of the new table. Enter "Departments" on the first line, then hit Enter. Read the following "About Quick Database Diagrams" user notes.

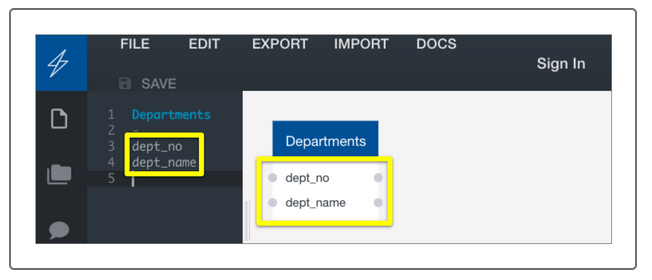


Below the table name, insert a hyphen to signify we're ready to start adding data.



Now add placeholders for your data. Remember, for a conceptual diagram, don't load all of the details. This diagram should only represent the relationships between the tables, so all we need is the name of each column. There are two column headers in departments.csv: dept\_no and dept\_name. Add those to our schema.

On the next blank line in the text editor, input "dept\_no" and hit Enter. On the next line, type "dept\_name" and hit Enter again. You'll see that the small table has grown a bit!



Did you notice "int" to the right of our table data? That's the default data type assigned by Quick DBD. It's there because we didn't specify what data type each of the headers represent.

Create a table for the next CSV, dept\_emp.csv, by following these steps:

1. After dept\_no, skip one line to leave it blank.
2. On the next line, start the new table by entering its name "Dept\_Emp."
3. On the next line, enter a hyphen to signify a new table, ready for data.
4. On the following lines, enter a column header for each column name.

**important**

The syntax is important when creating schema in Quick DBD. When creating tables, make sure to use underscores ( \_ ) instead of spaces. A space within a table or column name will generate an error.

Notice the second table has appeared next to the first—this is the beginning of our flow chart.



These tables aren't locked into place and can be rearranged for a cleaner look. To move a table, left click on the table name and drag to place. It's a useful feature because as we continue to add tables to the diagram the screen can become cluttered, and it allows you to organize the tables as you see fit.

The next step in building a diagram is to assign data types, which transitions our conceptual ERD to a logical ERD.

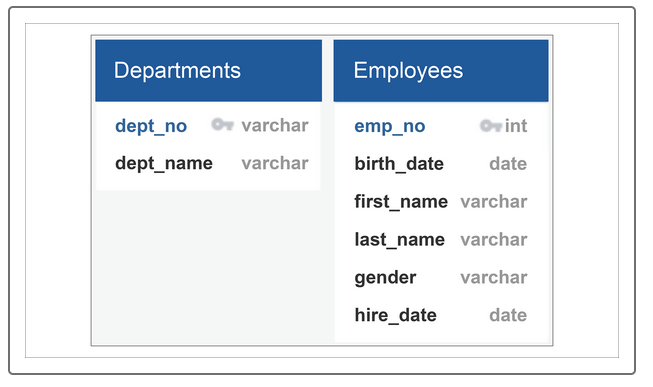
## Logical Diagrams

**Logical diagrams** contain all of the same information that a conceptual diagram does, but the table is updated to include data types and primary keys.

Returning to the Quick DBD webpage, let's update our schema. Because we already took an initial look at the worksheet, we have already identified the primary key and know what type of data we're working with. Using the following syntax, update our Departments schema:

* Add "varchar pk" to dept\_no.
* Add "varchar" to dept\_name.

We use [varchar (Links to an external site.)](https://en.wikipedia.org/wiki/Varchar) in these columns because the fields contain characters of varying length. Adding "pk" in the schema next to column indicates that column as a primary key. The table updated to reflect the changes in the text editor. A key symbol appears next to the dept\_no line, indicating that it is the table's primary key, and varchar is added to indicate its type.



The relationship diagrams are beginning to really take shape. Next, add foreign keys and physically map the relationships between them.

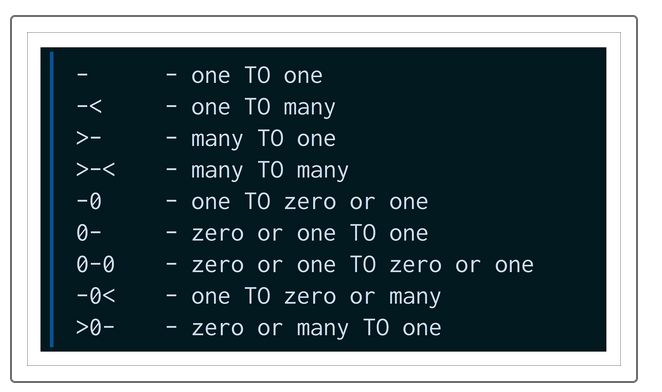
Varchar – variable character field type

PK – primary key

FK – foreign key

## Physical Diagrams

**Physical diagrams** portray the physical relationship, or how the data is connected, between each table. There are several different relationships available to keep in mind when making these connections, as shown below:



The relationships are fairly self-explanatory, too. For example, a **one-to-one** relationship means that one row, or record, in a table can be referenced to one other record in another table. For example, an employee will have a single employee number.

A **one-to-many** relationship, on the other hand, means that a row (entity) is referenced in two or more rows in another table within the database. For instance, an employee's number may be referenced in several other locations in a table that records employee advancement.

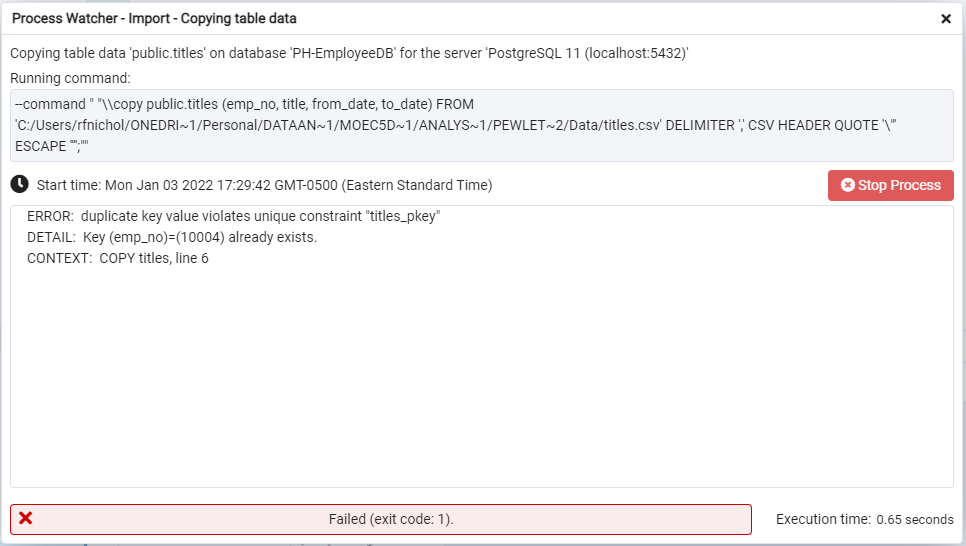
#### Practicing Using Entity Relationship Diagrams

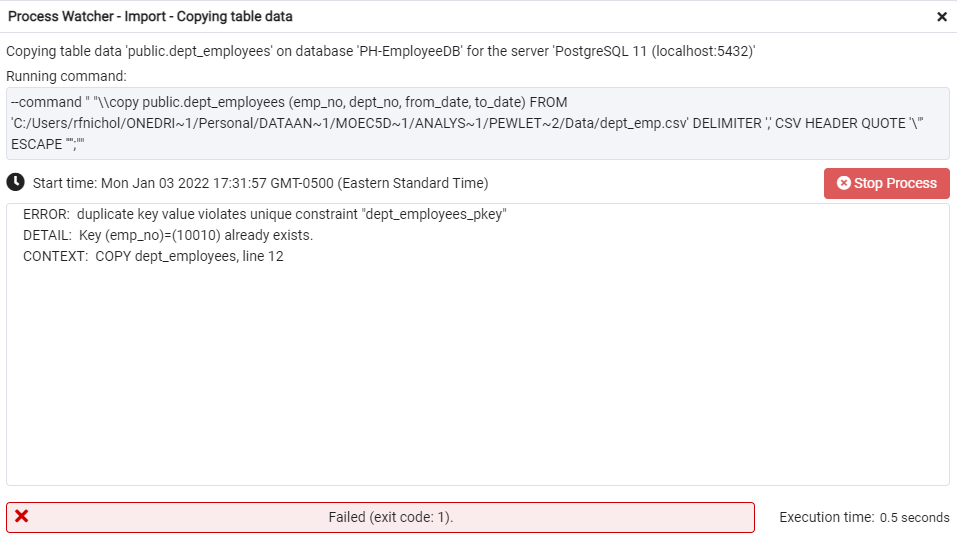
One benefit to building out each form of the diagram (conceptual, logical, and physical) is that it gives us more exposure to the data and its layout. Working with it often leads to more familiarity, which in turn helps with creating a map of the relationships. Even with the exposure and familiarity, it can still be confusing, so let's keep practicing.

Take a look at the Managers table in our text editor. We already know that the dept\_no is also the primary key for the Departments table, and because it's referencing another table, the dept\_no in the Managers table is a foreign key as well as a primary key. Add that designation to the schema by adding "fk" right after "pk" in that schema.

Now that we have the foreign key identified, we can create a one-to-one relationship between the Managers and Departments tables.

**Importing files into Database**

****



**Notes 7.1**

* The activities in this class will complement Lessons **7.0.1: Exploring Databases with SQL** through **7.3.3: Joins in Action**. The students will benefit from these activities if they‘ve progressed through these lessons, which cover the following concepts, techniques, and tasks:
* Create an ERD from CSV files
* Create a database on pgAdmin
* Create tables in a database using the ERD
* Import CSV files into database tables
* Create and run queries
* Use joins to combine records from two or more tables
* Create a new table from a query
* Export data from a table into a CSV file

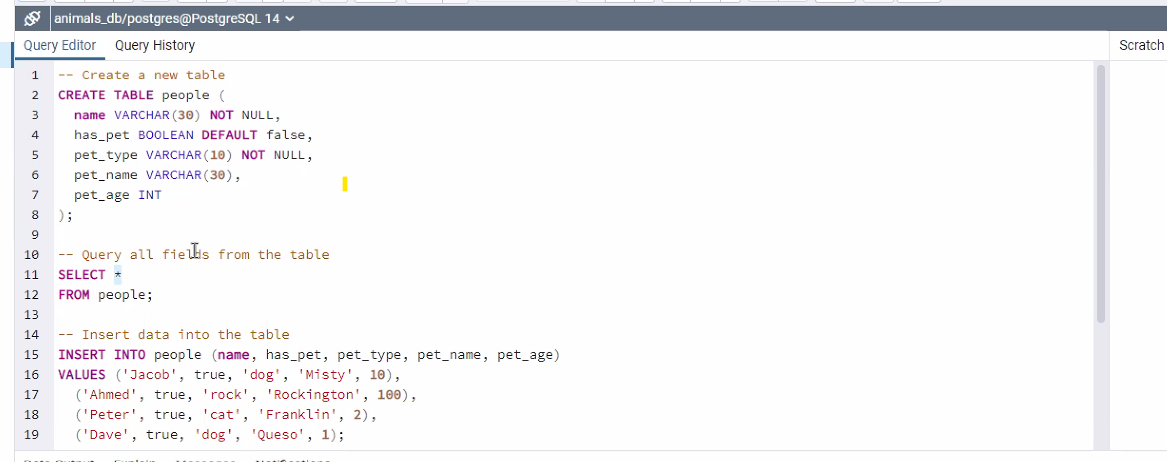
**What might learners struggle with?**  
Learners have historically enjoyed the shift to SQL, but you may have some folks who have trouble tying this technology together with what they have already learned. Be prepared to talk through the connective tissue between databases and what they've already learned by providing context and examples.

**What's the relevance of PostgreSQL?**  
Databases will be crucial to your work as data professionals, and PostgreSQL, or Postgres, is a free, open-source object-relational database management system that is very popular in some data careers.

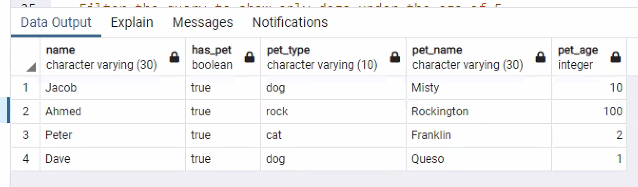
**Notes 7.2**  
The activities in this class will complement Lessons **7.3.4: Use Count, Group By, and Order By** through **7.3.6: Create a Tailored List**. The students will benefit from these activities if they‘ve progressed through these lessons, which cover the following concepts, techniques, and tasks:

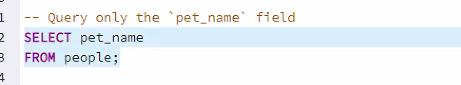
* Use the COUNT() function in a SQL query
* Group and order data in a SQL query
* Use joins to combine records from two or more tables
* Filter SQL queries using multiple joins with the WHERE clause

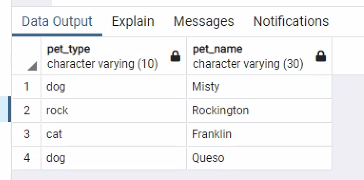
**Class 7.1**



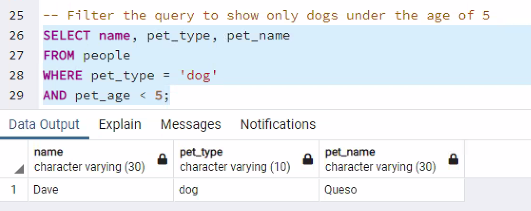




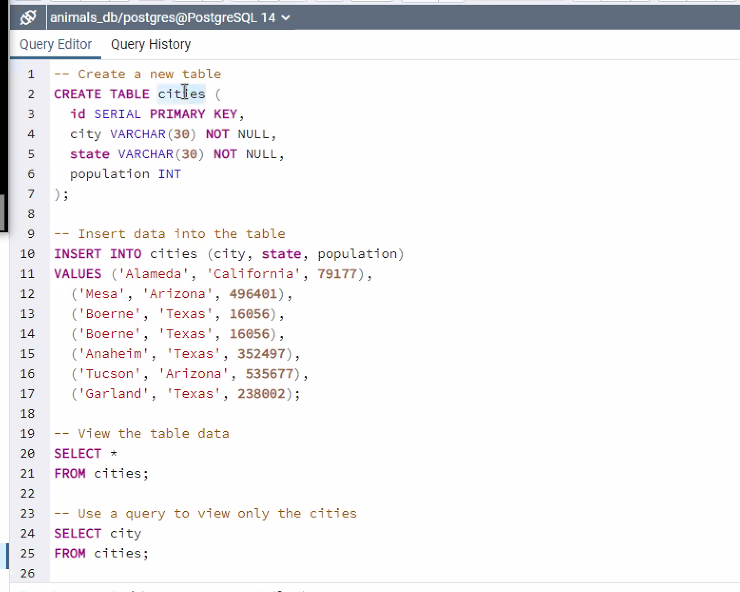


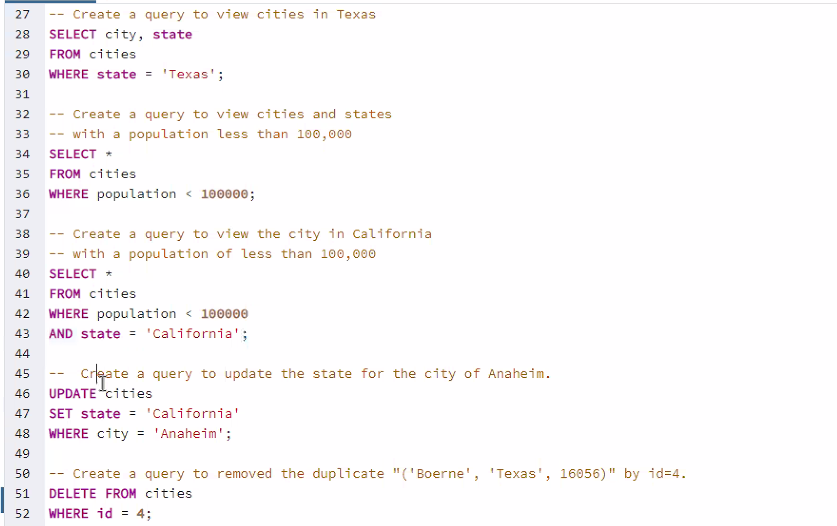


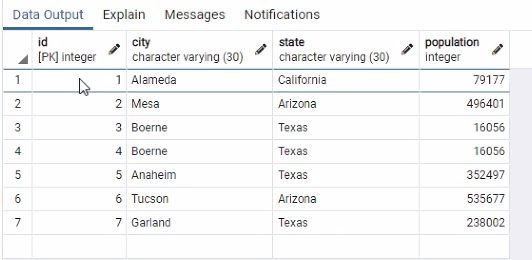




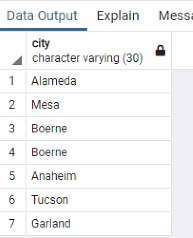
Another table – cities



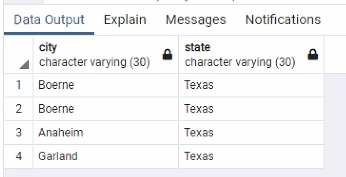




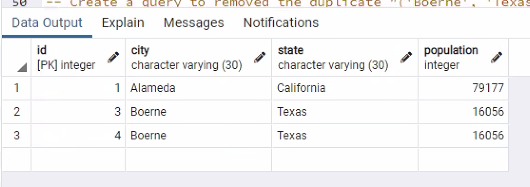
City query



Cities in TX

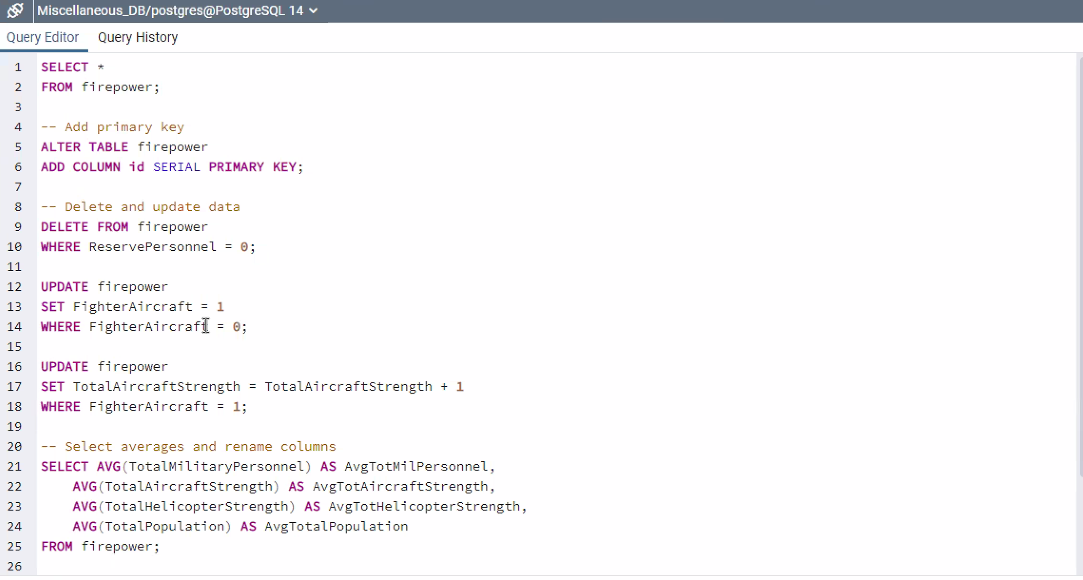


Cities less than 100K









Joins

Use Inner and Left Joins – mostly in real world

Cross joins provide highest number of records. Right joins are rarely used unless specifically called for

Instructions Hide and Seek

1. Create a new table in the Miscellaneous\_DB database called wordassociation.
2. Import the data from the WordAssociation\_AC.csv file in the Resources folder.
3. Create a query in which the data in the word1 column is stone.
4. Create a query that collects all rows in which the author is within the range 0–10.
5. Create a query that searches for any rows that have pie in their word1 or word2 columns.

Bonus

* Import WordAssociation\_BC.csv to the wordassociation table explore filtering on the source column.
* Create a query that will collect all rows with a source of BC.
* Create a query that will collect all rows with a source of BC and an author range between 333 and 335.

[Xavier Jackson](https://app.slack.com/team/U02LW76BVL3)  [8:37 PM](https://osuvirtdatapt-wnh9099.slack.com/archives/C02KYL1LWSC/p1641260255115000)

Joining the NBA  
In this activity, you will be using joins to query NBA player seasonal statistics.Instructions

1. Create a new database named NBA\_DB and create two new tables with pgAdmin named players and seasons\_stats.
2. Copy the code from schema.sql to create the tables, and then import the corresponding data Players.csv and Seasons\_Stats.csv. Note: Remember to refresh the database; newly created tables will not immediately appear.
3. Perform joins that will generate the following outputs:

Class 7.2

Gregarious Aggregates  
In this activity, you will practice writing queries with aggregate functions with grouping and adding aliases.Instructions  
Use aggregate functions as you run queries to answer the following questions. You will have to search the internet for some of them. Try to use aliases for more informative column headings.

1. What is the average cost to rent a film in the Sakila stores?
2. What is the average rental cost of films by rating? On average, what is the cheapest rating of films to rent? What is the most expensive?
3. How much would it cost to replace all films in the database?
4. How much would it cost to replace all films in each ratings category?
5. How long is the longest movie in the database? How long is the shortest movie?

Hint: Consult the Postgres documentation on [aggregate functions](https://www.postgresql.org/docs/9.5/functions-aggregate.html) for a summary of the available functions.

