Python for Data Analysts

Overview

Welcome to my course: **Python for Data Analysts**. The purpose of this course is to give you the skills to get started using Python like a Data Analyst. It doesn't teach you how to be a developer and I skip over or lightly cover any concepts that I think are less important to getting started. The idea is to just teach you what you need to get started as a Data Analyst and get you motivated enough to start learning Python in depth by yourself as you need to.

Coding is a difficult skill to pick up and doesn't come naturally to a lot of people including myself. If you are just learning how to code then I highly recommend you set up your Python environment the same way I did. Development environments can be complicated even for experienced individuals.

Instructions are highlighted

I've released a YouTube video to accompany this document and **highly** recommend that you follow along with that as well. You can find it on my YouTube channel here:

Also make sure you download the datasets necessary for this tutorial here.

One more thing. If this is your first time coding or you're just starting your journey, don't worry! You will probably need to go through this at least twice in order to start understanding

What is Python and What does it help Data Analysts do?

Python is a high-level, object-oriented programming language.

- High-level: What you program in Python is abstracted away from the actual operations a computer performs to get there. This means that programming is easier to understand
- Object-Oriented: Your code is designed around the idea of Objects. We'll be going over what Objects are shortly

It is one of the most popular languages used by Data Analysts and Scientists because it is open-source, very flexible, easy to learn, and VERY well-supported. That last point is very important because many languages have all the prior features, but because Python has gained critical mass in the Analytics industry, almost everyone who designs an application to be used by Data Analysts or Data Scientists will design it with Python compatibility. Aspects of Analytics that are well supported by Python and their respective supported libraries include:

- **Connecting to Data**
    - SQL Alchemy: Connect to SQL Databases
    - PySpark: Connect to Big Data
- **Exploring Data**
    - SciPy: Run statistical analyses on data
    - Pandas: Explore data stored in tables
- **Modeling Data**
    - skLearn: Machine Learning
    - Tensorflow: Artificial Intelligence
- **Visualizing and Presenting Data**
    - Dash/Plotly/Streamlit: Make full web-applications

Let's get started:

Preparation

You'll need three things to get started with this course:

- Python: This will get the Python programming language + interpreter on your computer
- Text Editor: This will help you edit and deploy your code. We'll be using VSCode
- Jupyter Notebook: This is a program used by Data Analysts to easily explore data using Python

Installing Python through conda

We'll be installing Python using a program called Conda. Conda was designed for Data Analysts and makes it easy to manage all the extra programs you'll need to organize as you develop your skills.

Conda can be downloaded through a visual interface called Anaconda or a Command-Line interface called MiniConda

- I highly recommend using MiniConda instead of Anaconda because it's way lighter on your system. I've had major issues getting Anaconda to work well on my computer and recommend against it unless you know exactly what you need it for

Download the appropriate version of MiniConda from the official website

If you have a Windows computer and want to know whether you have 32-Bit or 64-Bit version of the operating system, check here: https://support.microsoft.com/en-us/windows/32-bit-and-64-bit-windows-frequently-asked-questions-c6ca9541-8dce-4d48-0415-94a3faa2e13d

Download VSCode

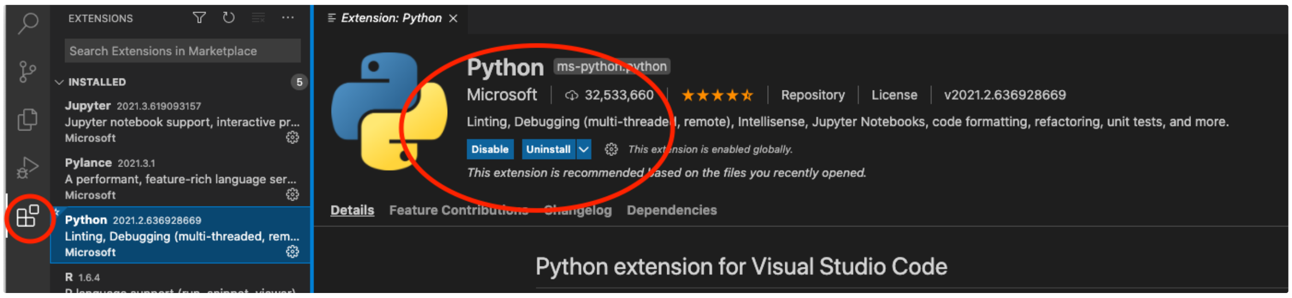
- I prefer to use a text editor instead of an IDE because they're just much simpler and have all the functionality a beginning Data Analyst needs
- VSCode is a very well supported editor made by Microsoft

Download the appropriate version of VSCode from the official website

Jupyter Notebooks

- Install the Python and Jupyter extensions
- I recommend using Visual Studio Code that way if you want to make .py files it's easy
- Code Cells, inline plots

Open VSCode and click on the "Extensions" button on the left side of the toolbar. Use the search bar to download the "Python" and "Jupyter" extensions from the marketplace.



Jupyter 2021.3.619093157
Jupyter notebook support, interactive pr...
Microsoft

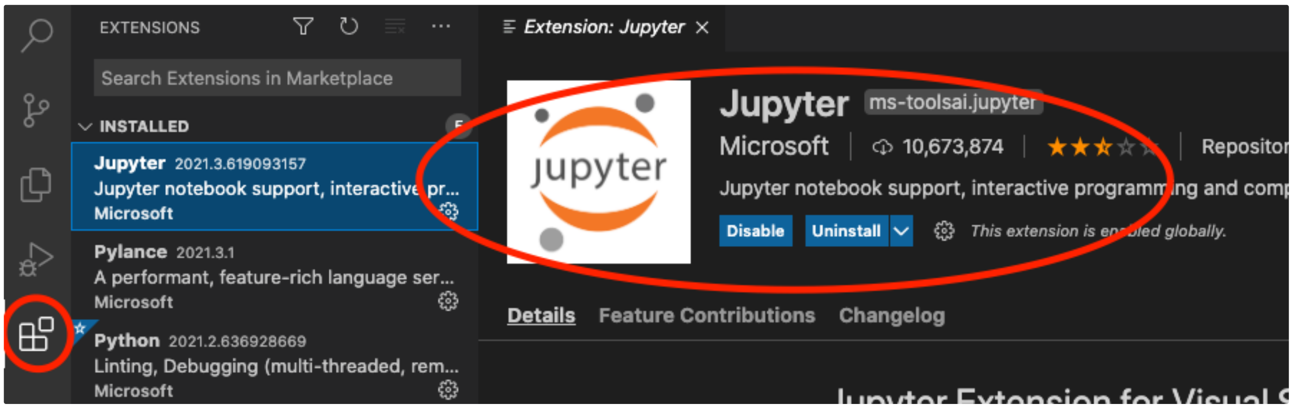
Pylance 2021.3.1
A performant, feature-rich language ser...
Microsoft

Python 2021.2.636928669
Linting, Debugging (multi-threaded, rem...
Microsoft

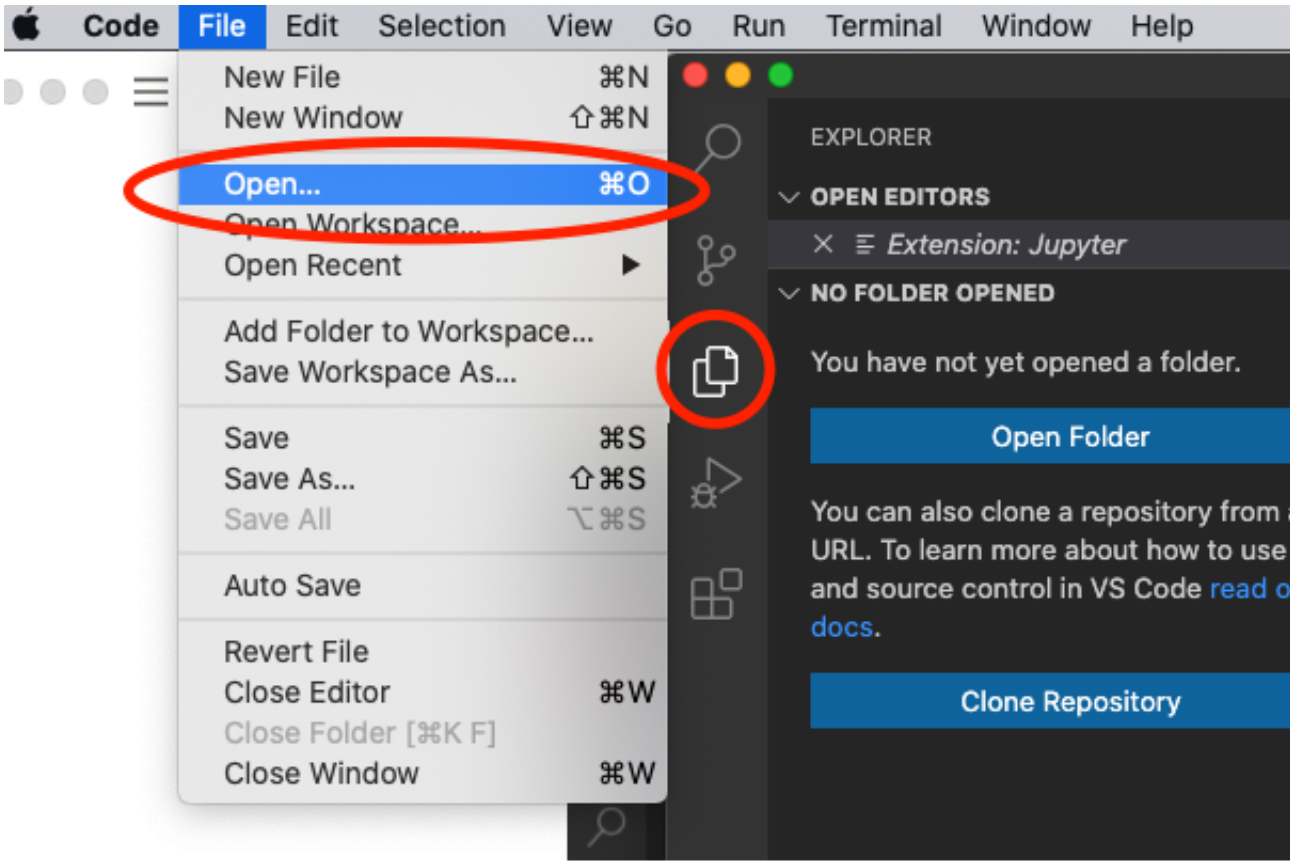
R 1.6.4

Python ms-python.python
Microsoft | 32,533,660 | ★★★★★ | Repository | License | v2021.2.636928669
Linting, Debugging (multi-threaded, remote), Intellisense, Jupyter Notebooks, code formatting, refactoring, unit tests, and more.
This extension is enabled globally.
This extension is recommended based on the files you recently opened.

Python extension for Visual Studio Code


EXTENSIONS
Search Extensions in Marketplace
INSTALLED
Jupyter 2021.3.619093157
Jupyter notebook support, interactive pr...
Microsoft
Pylance 2021.3.1
A performant, feature-rich language ser...
Microsoft
Python 2021.2.636928669
Linting, Debugging (multi-threaded, rem...
Microsoft
Extension: Jupyter
Jupyter ms-toolsai.jupyter
Microsoft 10,673,874
Jupyter notebook support, interactive programming and comp
Disable
Uninstall
This extension is enabled globally.
Details
Feature Contributions
Changelog

Create a folder on your computer that you want to work out of and navigate to that folder by clicking on the "Explorer" tab on the toolbar and use the "File" → "Open" (or "Open Folder" on Windows" to navigate to the folder you've just created.



If at any point through the tutorial you find that getting your computer up and running with Python isn't working as expected, then feel free to use Google's free online cloud programming interface at this website: https://colab.research.google.com/

Basic Python

Python Files

Python files normally use the `.py` file extension. If you see this then you know you're looking at a Python file. The files that we'll be working with are called Jupyter Notebooks and have the file extension `.ipynb` (this is because they used to be called IPython Notebooks).

Jupyter Notebooks

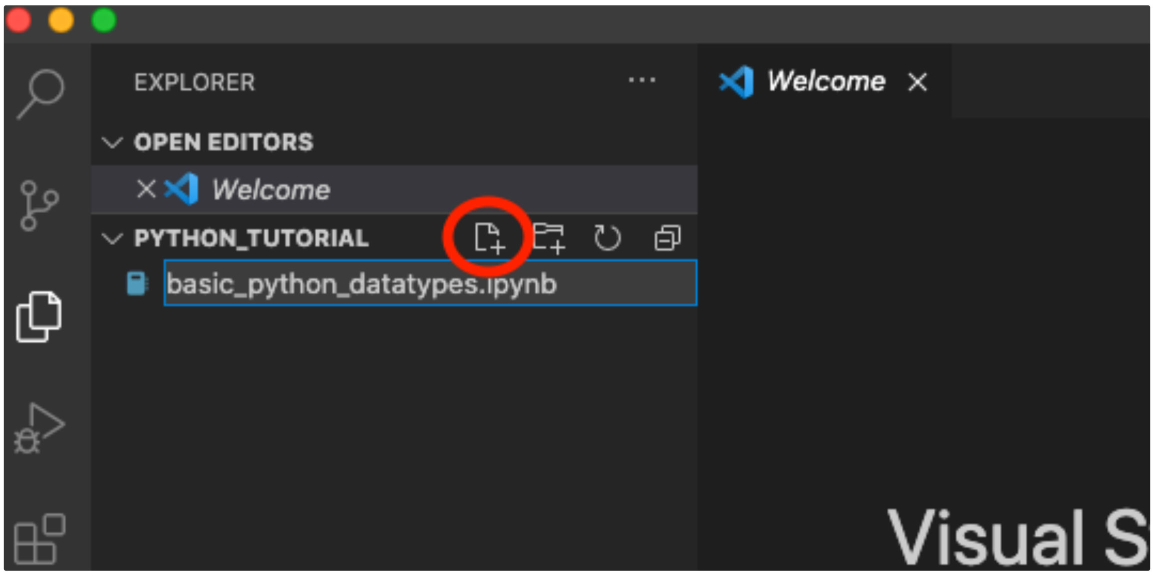
Jupyter Notebooks are files that allow you to combine code, comments, and visualizations all into one file. They are a key way that Data Analysts and Scientists interact with their data and my preferred way to programming.

Code is stored in something called a cell which can be used to run code in chunks which can make development much easier.

Here is an example Notebook that was created by the people who created Jupyter Notebooks.

We're ready to get started!

Create a new file using this button in the folder you just created and call it `basic_python_datatypes.ipynb`



Objects

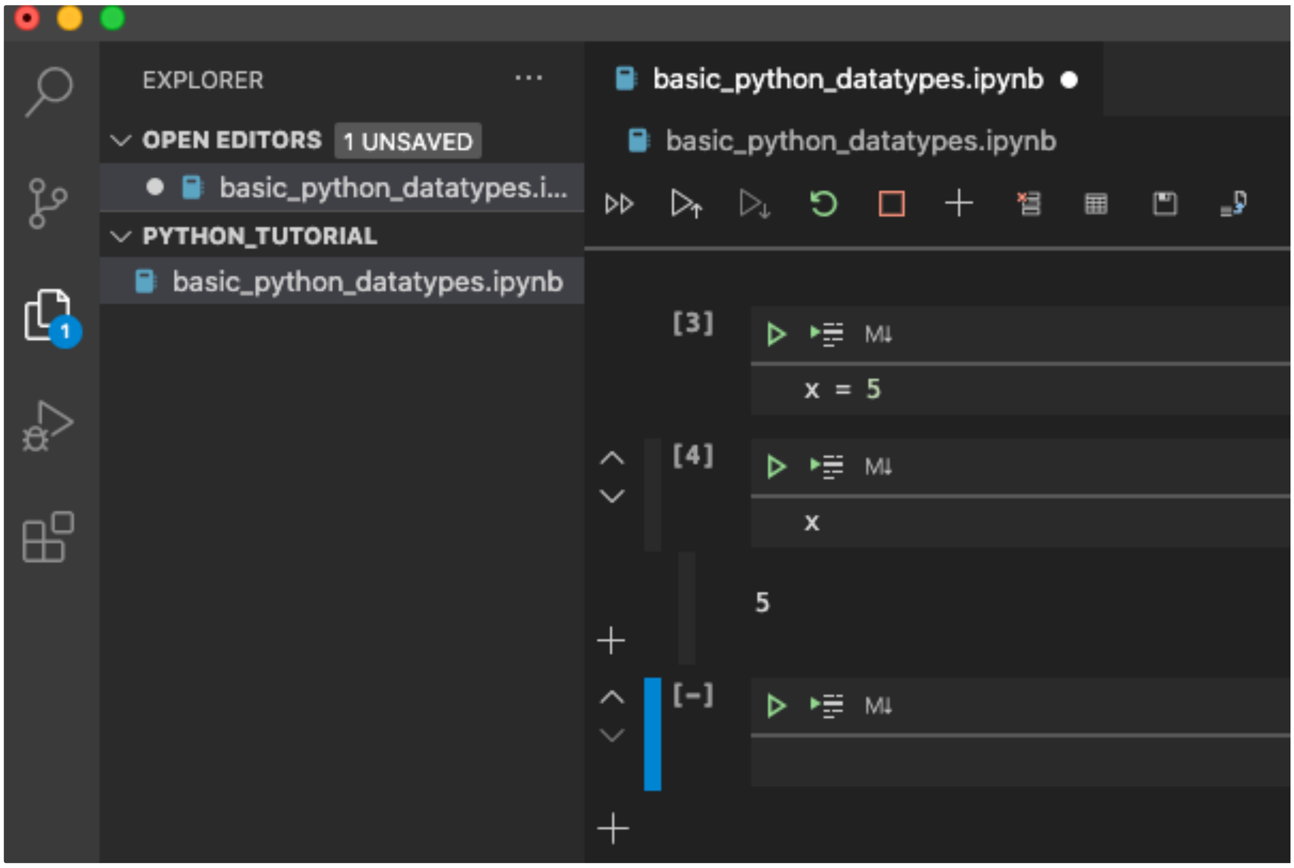
Everything in Python is an object. Objects are the core of the language and are what you use to do almost anything in Python. Every object has an ID, type, and value in memory. The actual technicalities of objects are worth learning but are beyond the scope of this course.

Variables

In Python you don't need to declare a variable the same way you need to in other languages. A variable is created when you assign a value to it.

In our code editor, let's type `x = 5` and hit "Shift + Enter".

Congratulations, you've declared a variable and run Python code! Type `x` in the next code cell and run the cell "Shift + Enter"


EXPLORER
OPEN EDITORS 1 UNSAVED
basic_python_datatypes.i...
PYTHON_TUTORIAL
basic_python_datatypes.ipynb
basic_python_datatypes.ipynb

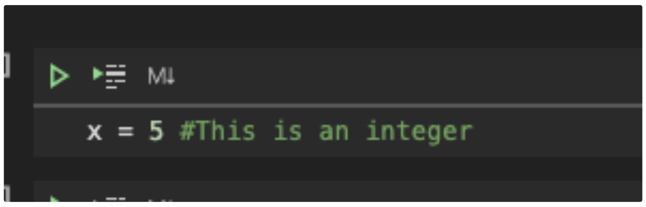
```
[3]  x = 5

[4]  x

5

[-]
```

You'll see that you assigned the value "5" in one cell and run the code in the next one.

Commenting in Python

Comments are a great way of documenting your code and explaining why you did certain things. You can comment using `#` to comment stuff in Python.

You can hit "CMD + /" ("CTRL + /" on Windows) on any line to instantly comment it

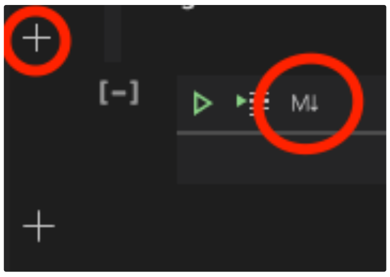


```
x = 5 #This is an integer
```

Markdown Cells

In Jupyter Notebooks you have two basic cell types that you can use:

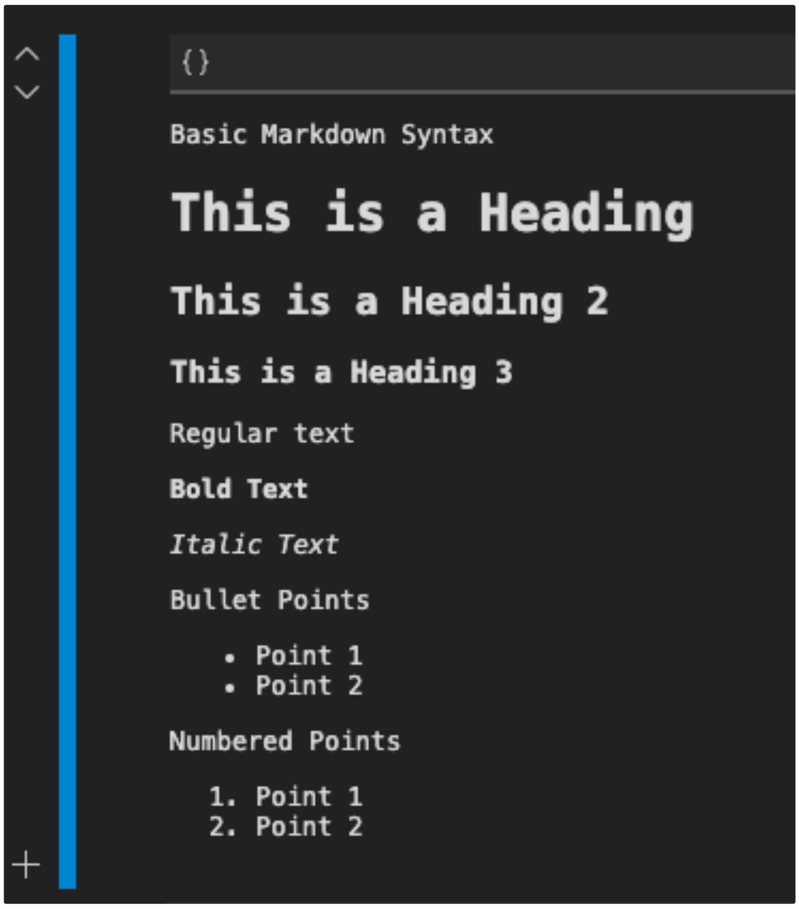
- Code Cells - Allow you to run code
- Markdown Cells - Allow you to run Markdown which is a text publishing tool that's super easy to use



You can create a new cell using the "plus" button highlighted above, and change the cell from a Code cell to a Markdown Cell by clicking on the "M↓" button highlighted above. (If you can't type in the cell then double tap the bar to the right of the "{}"

Markdown allows you to use special syntax in order to make your text pop. Copy and paste the text below into the Markdown cell you just created and then hit "Shift + Enter"

```
Basic Markdown Syntax # This is a Heading ## This is a Heading 2
### This is a Heading 3 Regular text **Bold Text** *Italic Text*
Bullet Points * Point 1 * Point 2 Numbered Points 1. Point 1 2.
Point 2
```


Basic Markdown Syntax

This is a Heading

This is a Heading 2

This is a Heading 3

Regular text

Bold Text

Italic Text

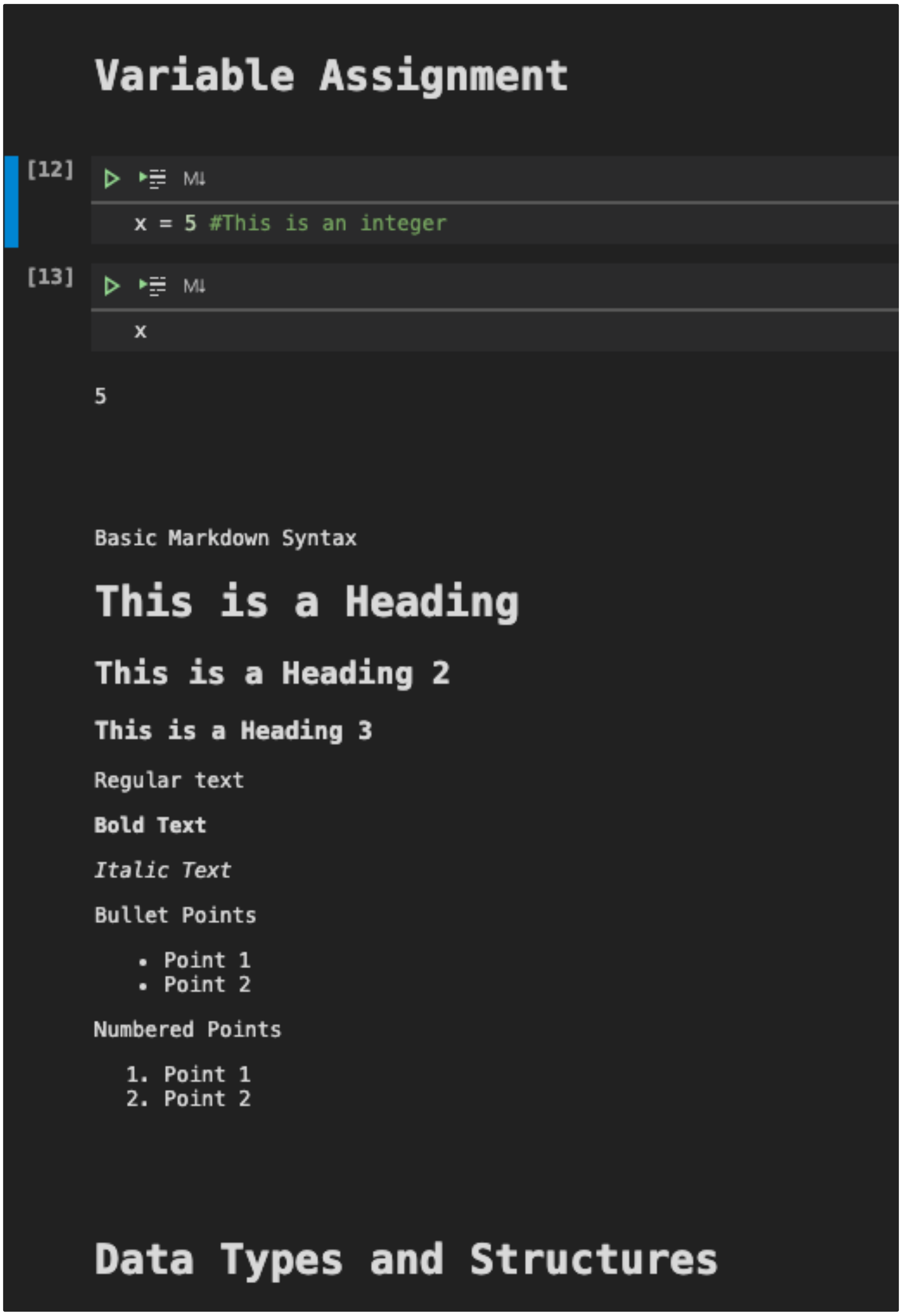
Bullet Points

- Point 1
- Point 2

Numbered Points

1. Point 1
2. Point 2

Let's create a new Markdown cell to label the concept of variable assignment and move it to the top using the arrows highlighted. Then let's create a new Markdown cell and at the bottom for Data Types and Structures. Your file should look like the screenshot below:



Variable Assignment

```
[12]
x = 5 #This is an integer
```

```
[13]
x
```

```
5
```

Basic Markdown Syntax

This is a Heading

This is a Heading 2

This is a Heading 3

Regular text

Bold Text

Italic Text

Bullet Points

- Point 1
- Point 2

Numbered Points

1. Point 1
2. Point 2

Data Types and Structures

Data Types/Structures

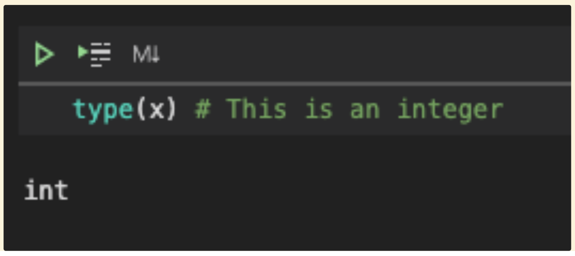
Data structures are the foundation of many programming languages. This is very important to understand and will underpin everything else we do for the rest of the course.

- **Primitive**

- Integers: Numeric integer data from at least -2,147,483,648 through 2,147,483,647
  type(x)

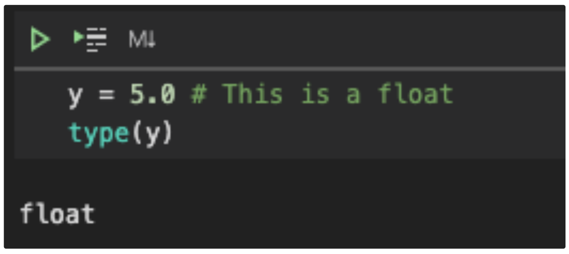
  ```
  type(x)
  ```

  - We created an integer (called an int) called `x` earlier in the tutorial. You can check the type of the variable with the code below. You can also comment that it's an integer.

    ```
    type(x) # This is an integer
    ```
    ```
    int
    ```

- Floats: This stands for "floating point number" and basically covers real decimal numbers

  ```
  y = 5.0 # This is a float
  type(y)
  ```
  ```
  float
  ```

- Strings: This is any alphanumeric data enclosed in quotations

  ```
  z = "I am learning Python" # A very basic string a = "5" #
  The number "5" as a string a = '5' # Single quotations to c
  reate a string a_two = "'5'" # Surround one type of quotati
  on with another one to make that quotation part of the stri
  ng b = """This is a multiline string""" # Triple quotes to
  denote a multiline string c = "Escaping a character \''" #
  Use a "\" to "escape a character" meaning that you'll print
  that character as is. This is useful for when you need to u
  se special characters like single quotes d = "this is how y
  ou create a \n newline, which is kind of like hitting the e
  nter key" # Use "\n" to denote a newline print(z) # Use the
  "print" function to look at all of these outputs print(a) p
  rint(a_two) print(b) print(c) print(d)
  ```

```
# Here are some character escapes that you should keep in m
ind \n # This is the same as hitting the "enter" key \t # T
his is the same as hitting the "tab" key
```

- Slicing: This is an operation that allows you to access parts of a sequential element in Python. Python uses something called 0-based indexing meaning that the "first" element of a sequence is actually the "0th" element

```
print(z[0]) # This gets the first element of the string
"I am learning Python" print(z[0:10]) # This gets the e
lements from 0 through 10, including 0 but excluding 10
```

- Boolean: True-False values that correlate to 1 and 0

```
boolean = True # This is a boolean not_boolean = "True" #
This is a string not a boolean x = 1 y = "1" x == y # This
gets us a Boolean
```

- You'll notice we used a `==` and not a `=`. The `==` is the equality operator and denotes that two items are equal to one another. This is opposed to the `=` which is used to assign a value to a variable

- **Non-primitive data structures**
  - List / Arrays: Python does not have an Array type per say, but something called a List that more or less serves the same function

```
a_list = [1, 2, 3, 4, 5, "asdas"] # This is a list
a_list[:3] # They can be sliced the same way strings are
a_list[::-1] # Here is an easy way to reverse a list
```

- Notice how you can put just about anything in a list: ints, floats, strings etc. You surround a list of items with brackets and separate them with commas

- Tuple: (pronounced "tupp-el") This is the same thing as a list but the elements inside cannot be changed. I very rarely use them and their mostly the result of outputs from other Python libraries

```
a_tuple = (1, 2, 3, 4, 5, "asdas") # This is a tuple a_tupl
e[:3] # They can be sliced the same way strings are a_tuple
[::-1] # Here is an easy way to reverse a tuple
```

    - They can be sliced the same way lists are

- Dictionary: These are key-value pairs and can be quite useful when we start working with tables

```
a_dictionary = {"key":"value", "multiple_values":["more", "
than", "one", "value", "per", "key"]} # This is a dictionar
y a_dictionary.get("multiple_values") # This gets the key w
ith a certain value a_dictionary["key"] # This is another w
ay of finding keys
```

- Set: A Set is an unordered, unchangeable way to store multiple variables without duplicates

```
a_set = {"Value", "first", "third"}
```

Operations

This section was heavily inspired by https://www.w3schools.com/python/python_operators.asp, a tremendous resource for all questions about Python.

Arithmetic Operators

```
add = 2 + 2 add_str = "str" + "strr" add_list = [1, 2, 3, 4, 5] +
[1, 2] sub = 3 - 1 mult = 3 * 3 mult_str = "str" * 3 mult_list =
[1, 2, 3] * 3 div = 2 / 2 modulo = 10%3 # Gives you the the remai
nder of a division operation floor_division = 7 // 2 # Gives you
the largest multiple, in this case it's 3 print(add) print(add_st
r) print(add_list) print(sub) print(mult) print(mult_str) print(m
ult_list) print(div) print(modulo) print(floor_division)
```

Assignment Operators

You can use these to quickly change the values of variables. You can see more information on these at https://www.w3schools.com/python/python_operators.asp

Comparison Operators

Comparison operators help us compare different values to one another. They are very useful in conditional statements and can help you write more logically complex code.

```
3 == 3 # Equals 3 != 4 # Does not equal 3 > 4 # Greater than 3 <
5 # Less than 2 >= 2 # Greater than or equal to 4 <= 4 # Less tha
n or equal to
```

Bitwise Operators

These are from the Python website: https://wiki.python.org/moin/BitwiseOperators

You'll see us use them to filter tables in pandas later in the course.

x << y: Returns x with the bits shifted to the left by y places (and new bits on the right-hand-side are zeros). This is the same as multiplying x by 2**y.

x >> y: Returns x with the bits shifted to the right by y places. This is the same as //'ing x by 2**y

x & y: Does a "bitwise and". Each bit of the output is 1 if the corresponding bit of x AND of y is 1, otherwise it's 0.

x | y: Does a "bitwise or". Each bit of the output is 0 if the corresponding bit of x AND of y is 0, otherwise it's 1.

~ x: Returns the complement of x - the number you get by switching each 1 for a 0 and each 0 for a 1. This is the same as -x - 1.

x ^ y: Does a "bitwise exclusive or". Each bit of the output is the same as the corresponding bit in x if that bit in y is 0, and it's the complement of the bit in x if that bit in y is 1.

Logical Operators

These are used to string together different comparison operators. We'll be using this when we're filtering tables

```
3 == 3 and 2 <= 5 # Returns true if both operations are true 3 ==
3 or 2 <= 1 # Returns true if one operation is true not(3 == 3 an
d 2 <= 5) # Returns the opposite (False in this case)
```

Membership Operations

These check to see if an object is a member of a collection of objects.

```
"T" in "TRDFSADFASD" "z" not in ["a", 123, 14, 41, 41, 4]
```

Functions

If you find yourself writing the same code over and over again you might need a function. A function basically allows to you create a code snippet that takes inputs and produces outputs.

```
def function_name(x, y, z): return x + y + z # The "return" tells
Python what to output from a function function_name(x = 1, y = 2,
z = 3) # This is how we call a function function_name(1, 2, 3) #
You can also call it like this by letting Python assume your argu
ments are in order function_name(1, 2, z = 4) # You can also not
specify earlier arguments, but specify later ones function_name(x
= 1, 2, 3) # This is not valid (Python doesn't know what argument
s to match 2 and 3 with
```

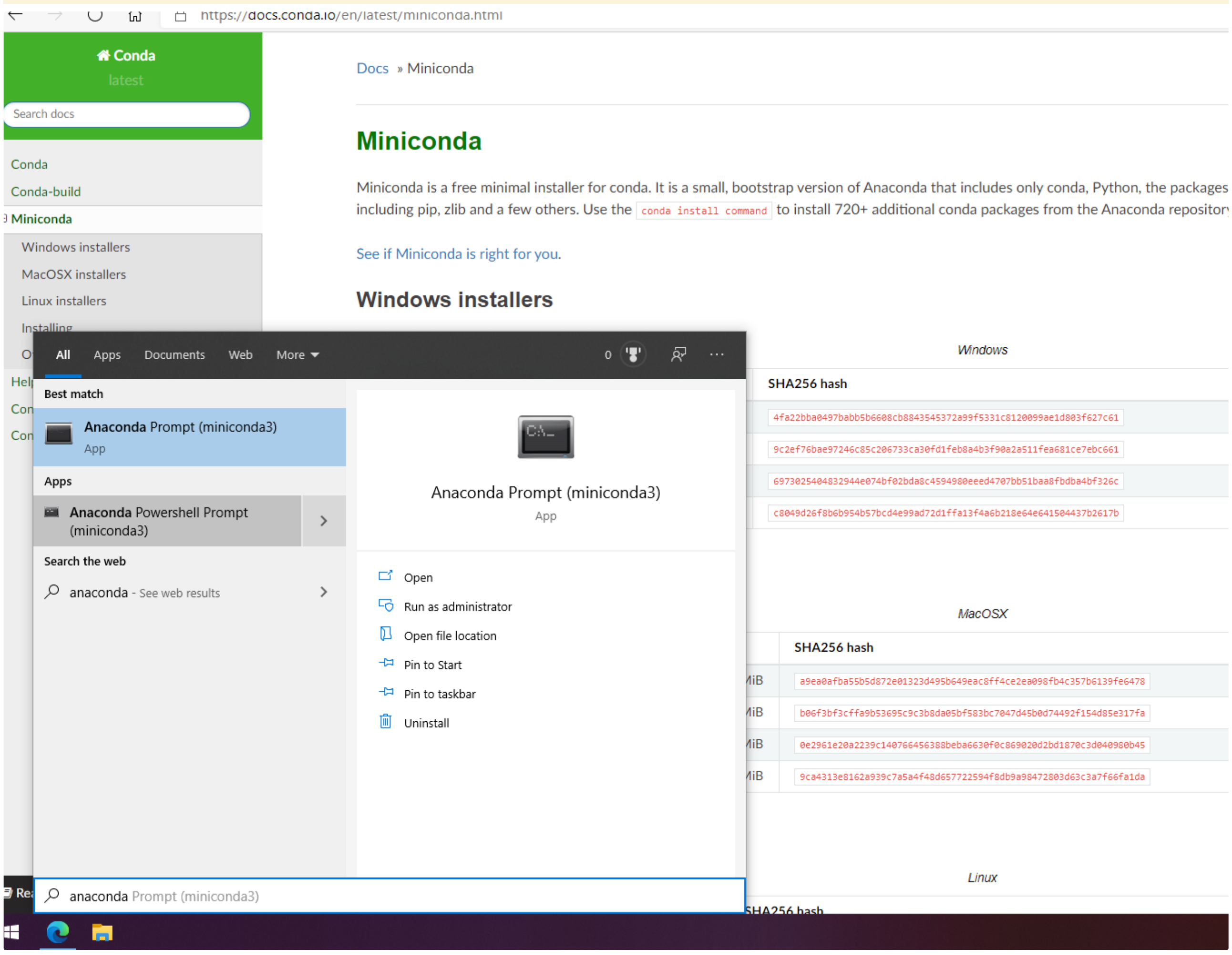
The inputs `x, y, z` are what we called arguments.

Libraries and Installing new libraries

One of the reasons that Python is such a great programming language is that people are constantly creating things called Libraries. These are large lumps of code that you can call in order to execute very complicated operations. If you wanted to run a machine learning algorithm for example, you'd install a library in your computer and call that library.

Windows:

Open up "Anaconda Prompt" using the Windows start menu

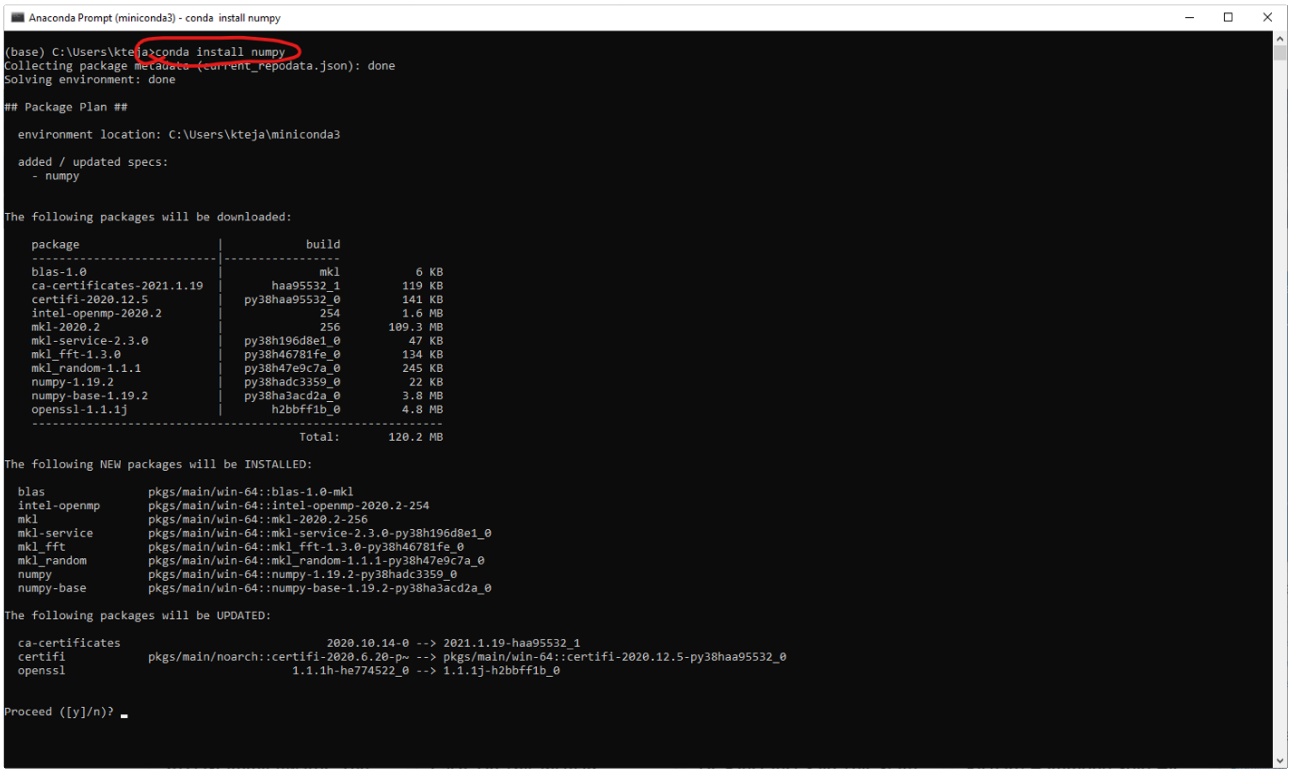


Miniconda is a free minimal installer for conda. It is a small, bootstrap version of Anaconda that includes only conda, Python, the packages including pip, zlib and a few others. Use the `conda install command` to install 720+ additional conda packages from the Anaconda repositor

See if Miniconda is right for you.

Type in:

```
conda install numpy
```



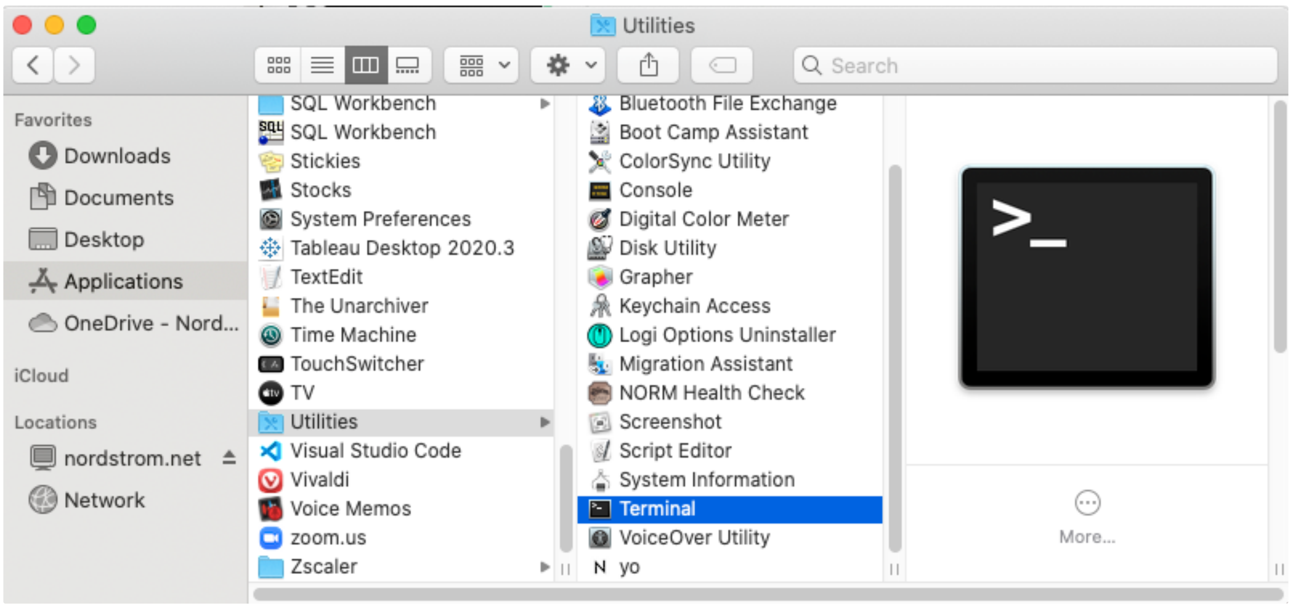
```
Anaconda Prompt (miniconda3) - conda  install numpy

(base) C:\Users\kteja>conda install numpy
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\kteja\miniconda3

  added / updated specs:
    - numpy


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    blas-1.0                   |              mkl           6 KB
    ca-certificates-2021.1.19  |       haa95532_1         119 KB
    certifi-2020.12.5          |   py38haa95532_0         141 KB
    intel-openmp-2020.2        |              254         1.6 MB
    mkl-2020.2                 |              256       109.3 MB
    mkl-service-2.3.0          |   py38h196d8e1_0          47 KB
    mkl_fft-1.3.0              |   py38h46781fe_0         134 KB
    mkl_random-1.1.1           |   py38h47e9c7a_0         245 KB
    numpy-1.19.2               |   py38hadc3359_0          22 KB
    numpy-base-1.19.2          |   py38ha3acd2a_0         3.8 MB
    openssl-1.1.1j             |       h2bbff1b_0         4.8 MB
    ------------------------------------------------------------
                                           Total:       120.2 MB

The following NEW packages will be INSTALLED:

  blas               pkgs/main/win-64::blas-1.0-mkl
  intel-openmp       pkgs/main/win-64::intel-openmp-2020.2-254
  mkl                pkgs/main/win-64::mkl-2020.2-256
  mkl-service        pkgs/main/win-64::mkl-service-2.3.0-py38h196d8e1_0
  mkl_fft            pkgs/main/win-64::mkl_fft-1.3.0-py38h46781fe_0
  mkl_random         pkgs/main/win-64::mkl_random-1.1.1-py38h47e9c7a_0
  numpy              pkgs/main/win-64::numpy-1.19.2-py38hadc3359_0
  numpy-base         pkgs/main/win-64::numpy-base-1.19.2-py38ha3acd2a_0

The following packages will be UPDATED:

  ca-certificates                          2020.10.14-0 --> 2021.1.19-haa95532_1
  certifi            pkgs/main/noarch::certifi-2020.6.20-p~ --> pkgs/main/win-64::certifi-2020.12.5-py38haa95532_0
  openssl                             1.1.1h-he774522_0 --> 1.1.1j-h2bbff1b_0


Proceed ([y]/n)? _
```

Type `y` and hit "Enter" and Conda should start installing the package.

It looks really complicated but this is essentially doing the same thing when a normal application installs. The advantage of using Conda like this is that it will help make sure that all of your packages are compatible with one another.

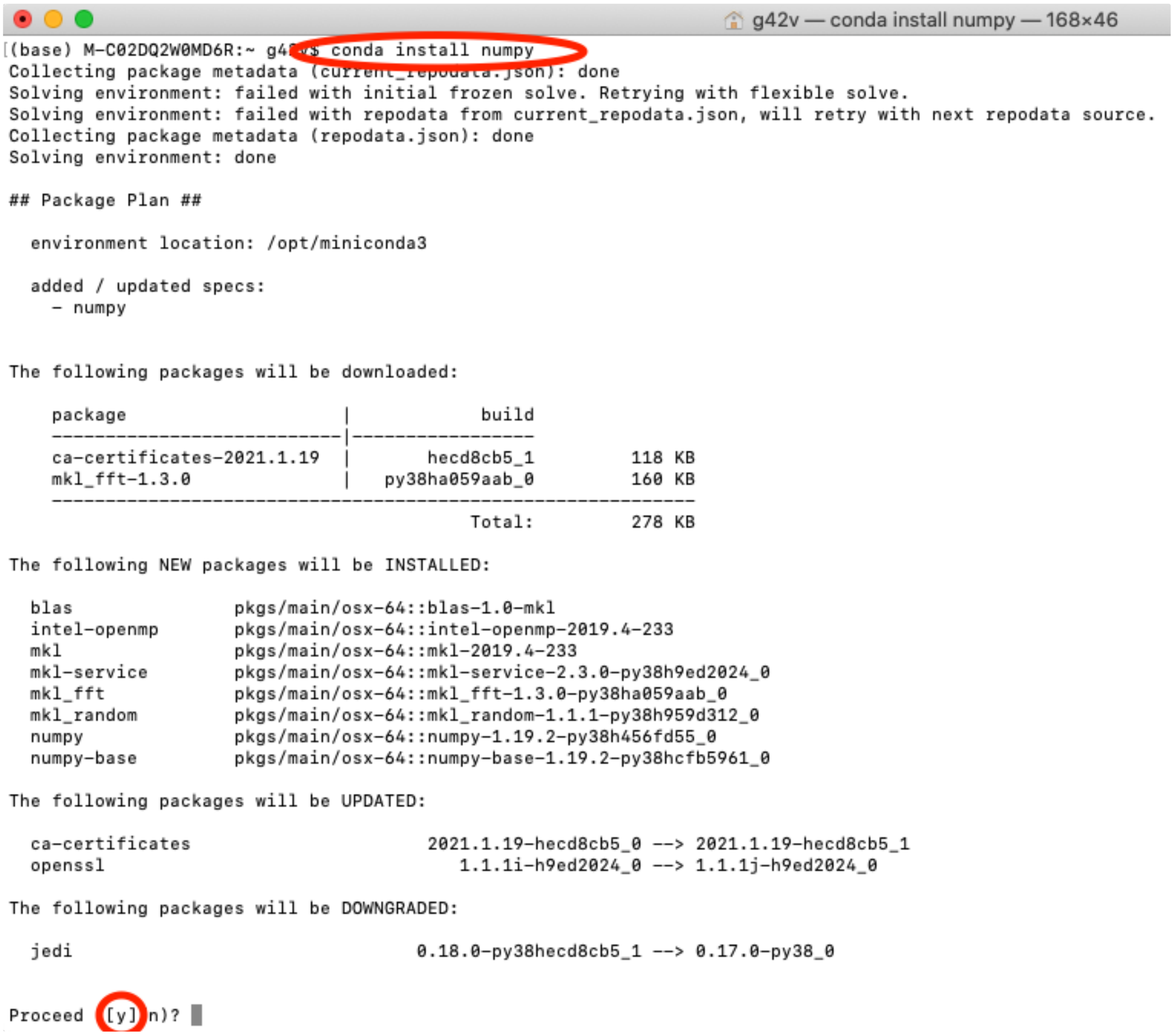
Mac:

Open up your Terminal. You can access it under the Utilities folder in the Applications folder.



Type in:

```
conda install numpy
```



```
[(base) M-C02DQ2W0MD6R:~ g42v$ conda install numpy
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /opt/miniconda3

  added / updated specs:
    - numpy


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2021.1.19  |       hecd8cb5_1         118 KB
    mkl_fft-1.3.0              |   py38ha059aab_0         160 KB
    ------------------------------------------------------------
                                           Total:         278 KB

The following NEW packages will be INSTALLED:

  blas               pkgs/main/osx-64::blas-1.0-mkl
  intel-openmp       pkgs/main/osx-64::intel-openmp-2019.4-233
  mkl                pkgs/main/osx-64::mkl-2019.4-233
  mkl-service        pkgs/main/osx-64::mkl-service-2.3.0-py38h9ed2024_0
  mkl_fft            pkgs/main/osx-64::mkl_fft-1.3.0-py38ha059aab_0
  mkl_random         pkgs/main/osx-64::mkl_random-1.1.1-py38h959d312_0
  numpy              pkgs/main/osx-64::numpy-1.19.2-py38h456fd55_0
  numpy-base         pkgs/main/osx-64::numpy-base-1.19.2-py38hcfb5961_0

The following packages will be UPDATED:

  ca-certificates                     2021.1.19-hecd8cb5_0 --> 2021.1.19-hecd8cb5_1
  openssl                                1.1.1i-h9ed2024_0 --> 1.1.1j-h9ed2024_0

The following packages will be DOWNGRADED:

  jedi                               0.18.0-py38hecd8cb5_1 --> 0.17.0-py38_0


Proceed ([y]/n)?
```

Type `y` and hit "Enter" and Conda should start installing the package.

It looks really complicated but this is essentially doing the same thing when a normal application installs. The advantage of using Conda like this is that it will help make sure that all of your packages are compatible with one another.

Importing a Library

When we want to use a non-default library, we'll need to import it

```
import numpy as np # the "as np" lets us call Numpy by just typin
g "np"
```

Methods

Methods are basically functions that are a part of classes. You call them in Python using a `.`. The highlighted section below is the method we're calling on the list class.

```
test = [1, 2, 3, 4] test.append(3) # This calls a method from the
"list" class
```

Control Statements

Control statements generally allow us to repeat operations in Python. They're very useful and definitely worth remembering.

- For Loops

  ```
  the_list = range(3, 10) # range creates a sequence of values b
  etween the two arguments for i in the_list: # This is how you
  start a for loop i += 2 # "i" represents the value of the sequ
  ence you're on print(i)
  ```

- While Loops

  ```
  x = 0 while x < 5: # The condition that has to be true for thi
  s loop to continue x +=1 # These are the calculations that are
  performed each loop print(x)
  ```

- If - Then

  ```
  x = 3 y = 5 if x == y: print("x equals y") elif x > y: print("
  x is greater than y") else: print("x is less than y")
  ```

- Try - Except: Try will help you block errors from your code

  ```
  try: 2 / 0 # You can't divide by 0 and this will throw an exce
  ption except ZeroDivisionError: print("Can't divide by 0")
  ```

- With
  - The best explanation of the `with` statement I've seen comes from https://www.geeksforgeeks.org/with-statement-in-python/
    - [The] with statement helps avoiding bugs and leaks by ensuring that a resource is properly released when the code using the resource is completely executed. The with statement is popularly used with file streams, as shown above and with Locks, sockets, subprocesses and telnets etc.
  - A common usage of the `with` statement is to read files. You want to use this statement in order to make sure that the filestream is closed after the file is opened

    ```
    with open('dog_breeds.txt', 'rb') as reader: print("File ha
    s been read")
    ```

- FAQ
  - https://www.reddit.com/r/learnpython/wiki/faq

NumPy

The tutorial on NumPy is heavily borrowed from the NumPy website; https://numpy.org/devdocs/user/quickstart.html

NumPy (num-pie) or "Numerical Python" is a Python library that undergirds almost all of Python's advanced mathematical and statistical functions. You might not use it too much directly but you'll be interacting with it constantly through other libraries.

The core of NumPy are the NumPy arrays which are kind of like very advanced Python Lists.

```
import numpy as np a = np.array([2, 3, 4]) # Make sure you use a list to create an array, the below is wrong b = np.array(2, 3, 4)
```

In the code block above, `a` is a 1-dimensional array.

```
c = np.array([2, 3, 4, 5, 6, 2]).reshape(2, 3)
```

In the code block above, `c` is a 2-dimensional array, basically a table.

```
d = np.array([2, 3, 4, 5, 6, 2, 3, 5]).reshape(2, 2, 2)
```

In the code block above `d` is a 3-dimensional array

NumPy Calculations

```
e = np.array([1, 2, 3, 4, 5, 6]) print(e.sum()) # Sum the array print(e.min()) # Find the minimum value in an array print(e.max()) # Find the maximum value in an array print(e.mean()) # Find the arithmetic mean of an array
```

Special NumPy Arrays

These are some useful arrays that you can quickly create in NumPy. Don't worry too much about learning them now but it'll be useful to have a cheatsheet of them nearby.

```
np.zeros(5) # Creates an array of zeros with 5 elements np.linspa
ce(0, 6, 10) # Creates an array of values from 0 through 6 in 10
even spaces np.random.rand(3) # Three random values between 0 and
1 np.random.randint(1, 1000) # A random integer between 1 and 100
0 np.arange(25) # Like the "Range" function we learned earlier, t
his is gives you a range of integers between 1 - 25
```

Pandas

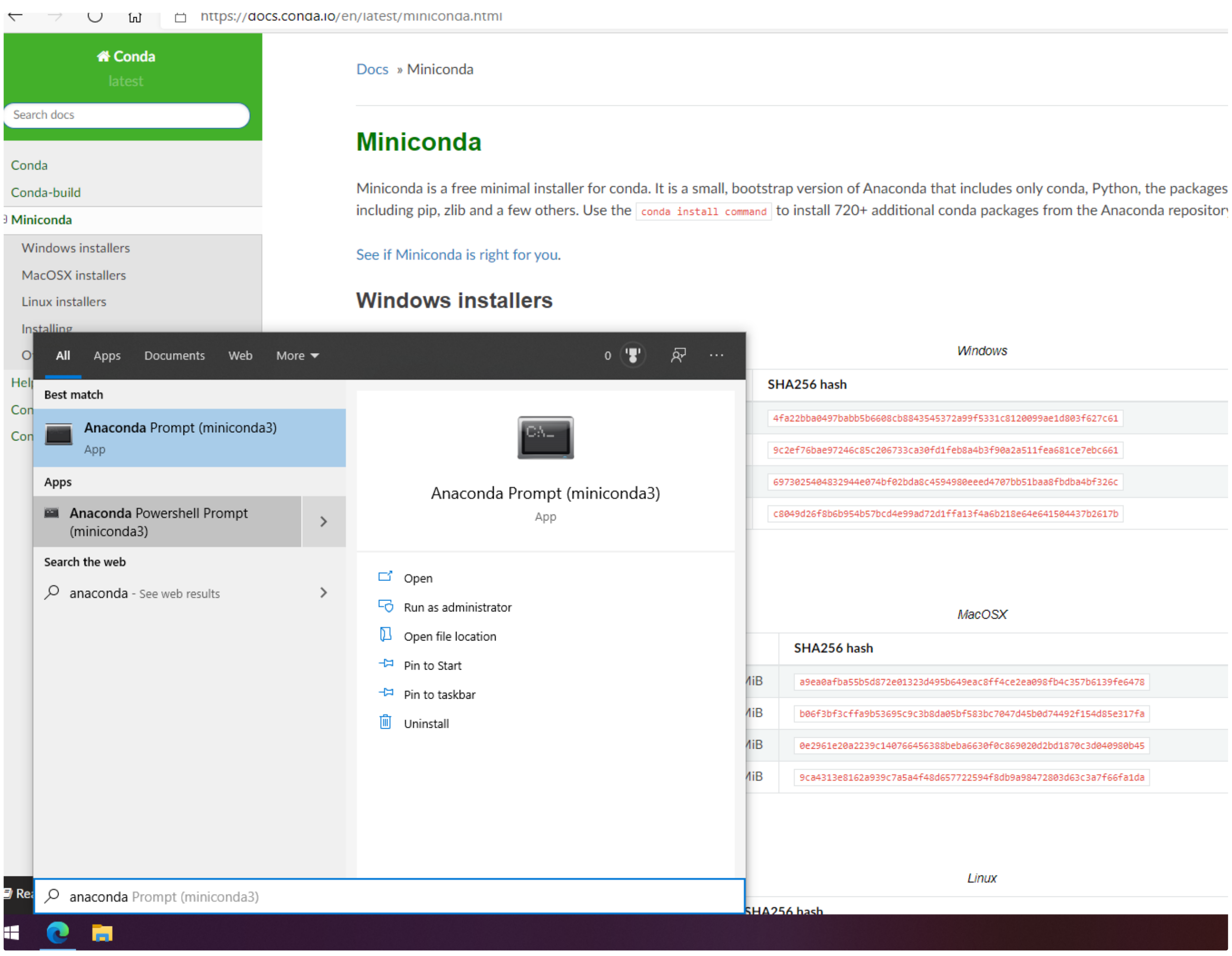
Congratulations! You've completed all of the preliminary work to understand Python. As a Data Analyst, I've had to know how to do all of the functions above, but I typically apply Pandas more directly in my work. This is where the meat and potatoes of Data Analysis lies. To see me use Pandas to solve a real-world problem check out the video linked here.

Pandas or "Panel Data" is the core library to handle structured data in Python. Structured data is just data in a tabular format (like in Excel). First let's install Pandas. We will install it the same way we installed NumPy.

Install Pandas

Windows:

Open up "Anaconda Prompt" using the Windows start menu


https://docs.conda.io/en/latest/miniconda.html

Conda
latest
Search docs

Conda
Conda-build
Miniconda
Windows installers
MacOSX installers
Linux installers
Installing

Docs » Miniconda

Miniconda

Miniconda is a free minimal installer for conda. It is a small, bootstrap version of Anaconda that includes only conda, Python, the packages
including pip, zlib and a few others. Use the `conda install command` to install 720+ additional conda packages from the Anaconda repositor

See if Miniconda is right for you.

Windows installers

Windows

SHA256 hash
4fa22bba0497babb5b6608cb8843545372a99f5331c8120099ae1d803f627c61
9c2ef76bae97246c85c206733ca30fd1feb8a4b3f90a2a511fea681ce7ebc661
6973025404832944e074bf02bda8c4594980eeed4707bb51baa8fbdba4bf326c
c8049d26f8b6b954b57bcd4e99ad72d1ffa13f4a6b218e64e641504437b2617b

MacOSX

	SHA256 hash
MiB	a9ea0afba55b5d872e01323d495b649eac8ff4ce2ea098fb4c357b6139fe6478
MiB	b06f3bf3cffa9b53695c9c3b8da05bf583bc7047d45b0d74492f154d85e317fa
MiB	0e2961e20a2239c140766456388beba6630f0c869020d2bd1870c3d040980b45
MiB	9ca4313e8162a939c7a5a4f48d657722594f8db9a98472803d63c3a7f66fa1da

Linux

SHA256 hash

All Apps Documents Web More

Best match
Anaconda Prompt (miniconda3)
App

Apps
Anaconda Powershell Prompt (miniconda3)

Search the web
anaconda - See web results

Anaconda Prompt (miniconda3)
App

Open
Run as administrator
Open file location
Pin to Start
Pin to taskbar
Uninstall

anaconda Prompt (miniconda3)

Type in:

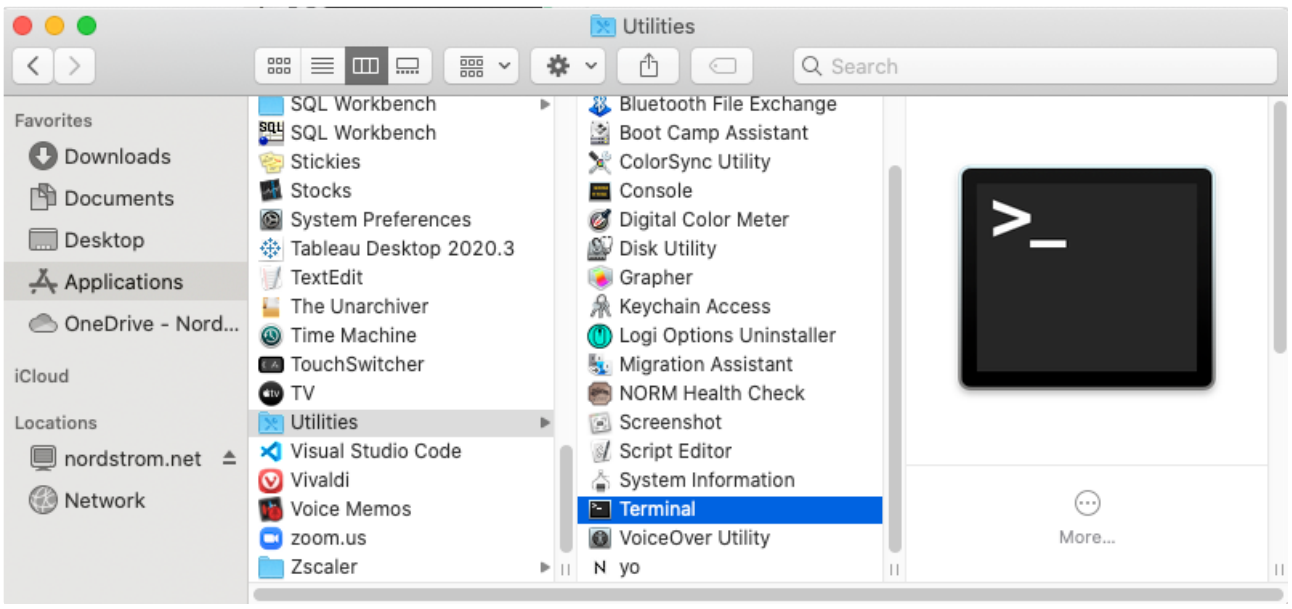
```
conda install pandas
```

Type `y` and hit "Enter" and Conda should start installing the package.

It looks really complicated but this is essentially doing the same thing when a normal application installs. The advantage of using Conda like this is that it will help make sure that all of your packages are compatible with one another.

Mac:

Open up your Terminal. You can access it under the Utilities folder in the Applications folder.



Type in:

```
conda install pandas
```

Type `y` and hit "Enter" and Conda should start installing the package.

It looks really complicated but this is essentially doing the same thing when a normal application installs. The advantage of using Conda like this is that it will help make sure that all of your packages are compatible with one another.

Import Pandas

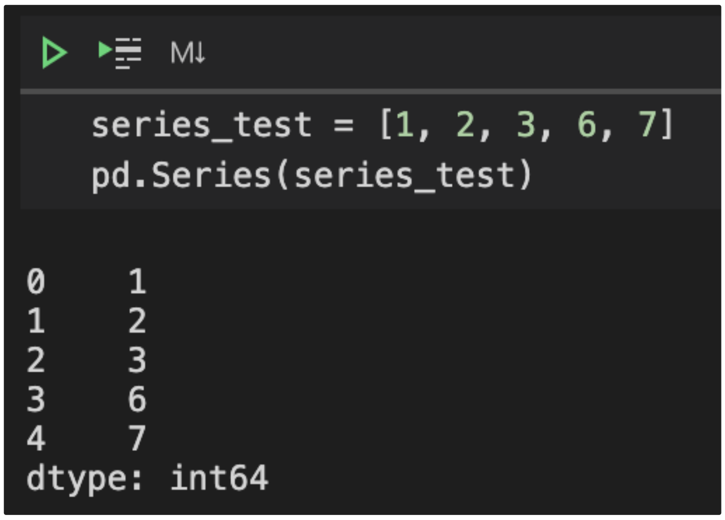
Like before, we need to import the library into our code editor. We'll alias it as `pd`

```
import pandas as pd
```

Series

The core of Pandas is the DataFrame. A DataFrame is basically a table and is composed of Series. A Series is a set of values that are indexed. Try the code below.

```
series_test = [1, 2, 3, 6, 7] print(pd.Series(series_test))
```



```
series_test = [1, 2, 3, 6, 7]
pd.Series(series_test)
```

```
0    1
1    2
2    3
3    6
4    7
dtype: int64
```

You'll see the output is "two" columns. The index on the left and the data on the right. The index is "kind-of" a column, but is better understood as the "address" of data in Pandas. We'll be working a lot with indices in the future.

DataFrames

Although we've almost entirely created our own data throughout this process, you'll usually work with data that already exists. This is where the `read_***` method of Pandas comes in. This allows you to read in data from any number of data sources including:

- CSVs
- Excel
- Stata
- SAS
- SPSS
- SQL
- Big Query
- ORC
- and much more!

For this tutorial we'll be sticking mostly to reading CSV's because there usually are special setups you need to get right to read data from SQL databases.

A CSV stands for "comma-separated values" and is a text format that separates each record with a comma and each line with a newline ("enter" key). It is part of a family of formats that include tab-separated values where the commas are separated by tabs, and pipe-separated values where the commas are separated by pipes (|). The commas, tabs, and pipes are what we call "delimiters".

In the folder you downloaded, there should be a file called "simple_csv.csv". Let's import that.

```
import os # This will be used to tell us where the file is import
pandas as pd pwd = os.getcwd() # This creates a string of the fol
der this Python Script is stored in filepath = pwd + "/simple_csv
.csv" # This creates a string that is the filepath to the simple_
csv file first_import = pd.read_csv(filepath) # This reads the cs
v into Python first_import
```

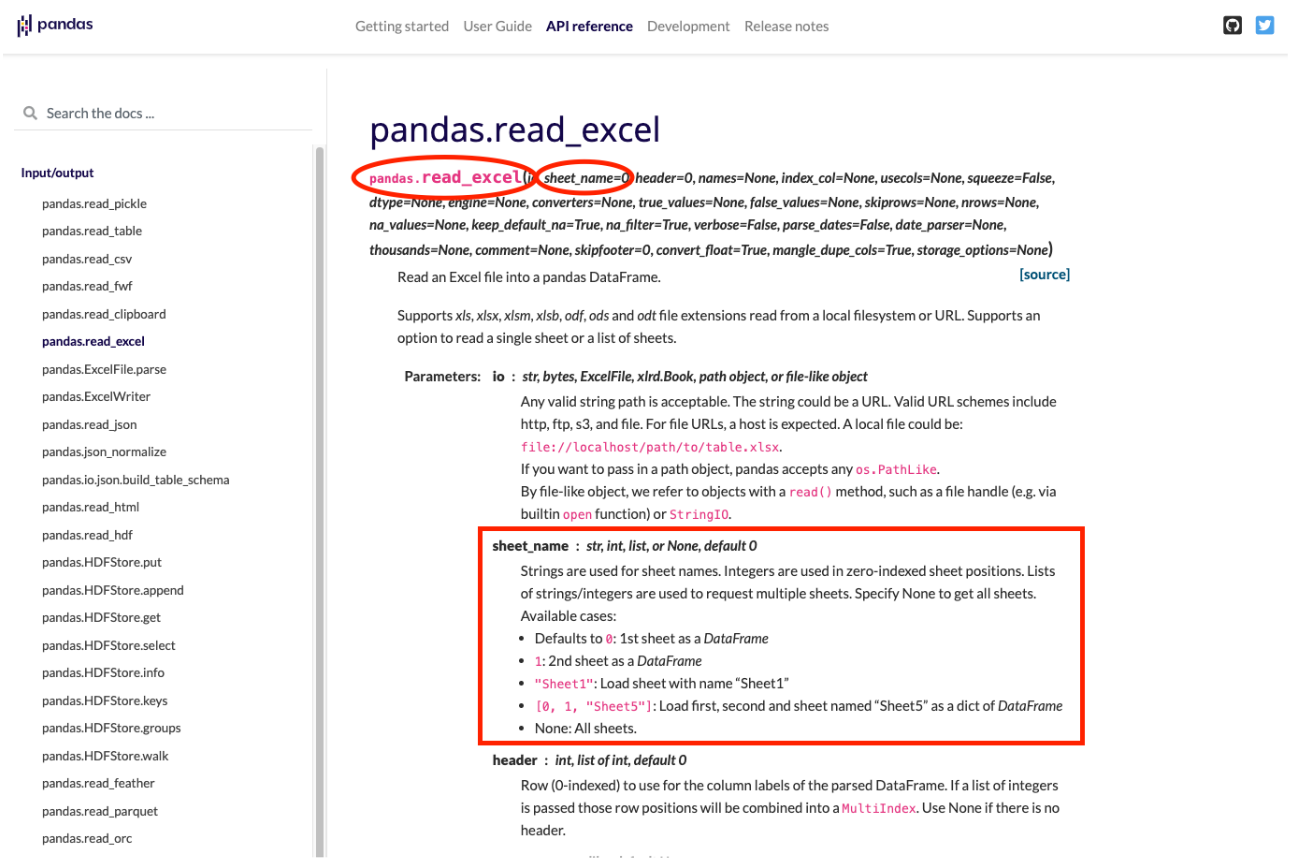
You'll notice that `pd.read_csv()` looks kind of like a function in that we are inserting arguments into a set of parentheses in order to tell Python what to do. Most of Pandas operates like this with you calling methods with arguments that you specify based on what you need. This is why it's very useful to be able to read documentation to figure out how you can specify other settings in a method.

Let's say we want to import an Excel file. Unlike csv's, Excel files have sheets that we need to specify so Pandas knows what sheet to import. So how do we figure out what the argument to specify a sheet is? Let's look at the documentation.

pandas.read_excel - pandas 1.2.3 documentation

Read an Excel file into a pandas DataFrame. Supports xls, xlsx, xlsm, xlsb, odf, ods and odt file extensions read from a local filesystem or URL. Supports an option to read a single sheet or a list of

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_excel.html



pandas.read_excel

`pandas.read_excel`(*io, sheet_name=0, header=0, names=None, index_col=None, usecols=None, squeeze=False, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skiprows=None, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, parse_dates=False, date_parser=None, thousands=None, comment=None, skipfooter=0, convert_float=True, mangle_dupe_cols=True, storage_options=None*) [source]

Read an Excel file into a pandas DataFrame.

Supports *xls*, *xlsx*, *xlsm*, *xlsb*, *odf*, *ods* and *odt* file extensions read from a local filesystem or URL. Supports an option to read a single sheet or a list of sheets.

Parameters: **io** : ***str, bytes, ExcelFile, xlrd.Book, path object, or file-like object***

Any valid string path is acceptable. The string could be a URL. Valid URL schemes include http, ftp, s3, and file. For file URLs, a host is expected. A local file could be: `file://localhost/path/to/table.xlsx`.
If you want to pass in a path object, pandas accepts any `os.PathLike`.
By file-like object, we refer to objects with a `read()` method, such as a file handle (e.g. via builtin `open` function) or `StringIO`.

sheet_name : ***str, int, list, or None, default 0***

Strings are used for sheet names. Integers are used in zero-indexed sheet positions. Lists of strings/integers are used to request multiple sheets. Specify None to get all sheets.
Available cases:

- Defaults to `0`: 1st sheet as a *DataFrame*
- `1`: 2nd sheet as a *DataFrame*
- `"Sheet1"`: Load sheet with name "Sheet1"
- `[0, 1, "Sheet5"]`: Load first, second and sheet named "Sheet5" as a dict of *DataFrame*
- None: All sheets.

header : ***int, list of int, default 0***

Row (0-indexed) to use for the column labels of the parsed DataFrame. If a list of integers is passed those row positions will be combined into a `MultiIndex`. Use None if there is no header.

You'll see here, if you want to read a specific sheet from an Excel file then you'll need to use the argument `sheet_name`

```
excel_import = pd.read_excel(filepath, sheet_name="Sheet1")
```

Most programming languages and libraries have too much functionality for anyone to memorize. This is why learning how to find and read documentation is very important.

Columns

Looking at the `first_import` variable we've created, let's see if we can perform some very basic transformations on it.

```
first_import.Column1 first_import["Column1"] # You can use either of these techniques to specify a column in a table. I prefer the second one because it allows your column names to have spaces in them
```

Let's see if we can create a column and drop (delete) another column.

```
first_import["New_Column"] = 1 # You can specify a uniform value for every row first_import["New_Column2"] = range(29) # You can pass a list of values that is as long as the DataFrame first_import["Adding Stuff"] = first_import["Column3"] + first_import["Column1"] # You can take a couple of columns and add their values together first_import = first_import.drop(columns="Column2") first_import.drop(columns="Column3", inplace=True) # The "drop" method creates a copy of the DataFrame. If you want to change the original DataFrame either do what we did in the line above, or use the "inplace" argument first_import
```

Rows

Rows can be a bit more complicated to deal with in Pandas. You'll be using the `loc` and `iloc` functions a lot.

The `loc` method allows us to use the names of columns to specify them.

You structure a `loc` query like below:

`dataframe.loc[from_row : to_row , [list_of_columns]]`

```
first_import.loc[:, "Column1"] # Select all rows in Column1 first_import.loc[0:4, "Column1"] # Select rows 0 through 4 INCLUSIVE (different from other Python indices) in Column1 first_import.loc[:, ["Column1", "New_Column2"]] # Select all rows in Column1 and Column2
```

An `iloc` query is based off of normal numerical indexing but operates mostly the same way:

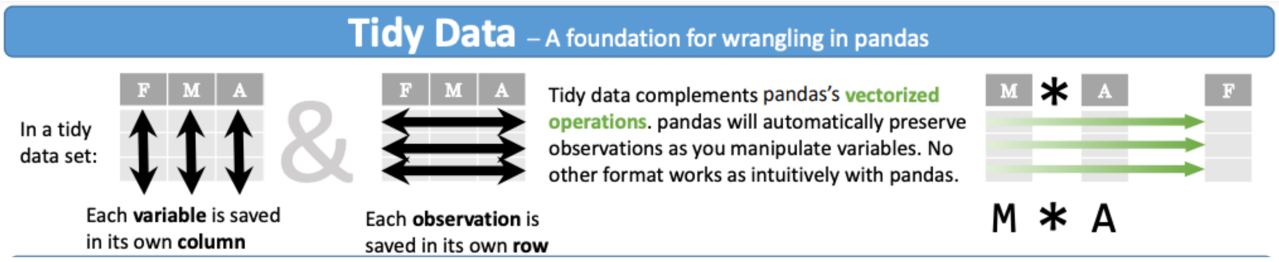
```
first_import.iloc[:, 1] # Select all rows in Column1 first_import.iloc[0:4, 1] # Select rows 0 through 4 excluding 4 (different from other Python indices) in Column1 first_import.iloc[:, 0:2] # Select all rows in Column1 and Column2
```

Let's start working with some real data. Import the dataset `netflix_titles.csv`. This dataset was created by Shivam Bansal at https://www.kaggle.com/shivamb/netflix-shows

```
netflix = pd.read_csv(pwd + "/netflix_titles.csv") netflix
```

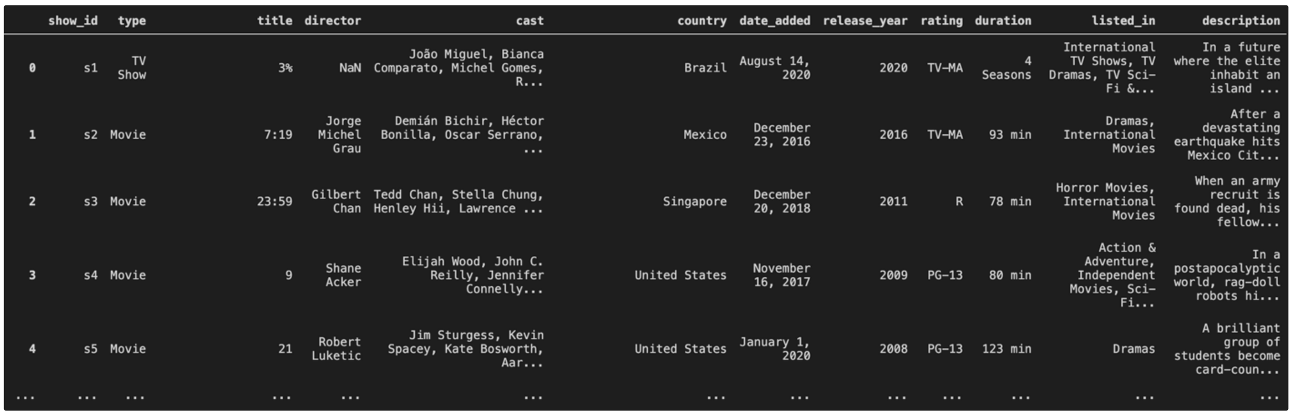
Taking a look at the data you can see it's very thorough and clean. It's what we call Tidy Data. The chart below from the pandas website explains it very well:

https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf


Tidy Data – A foundation for wrangling in pandas
In a tidy data set:
F M A
&
F M A
M * A
F
M * A
Each variable is saved in its own column
Each observation is saved in its own row

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

The data we imported looks something like this:



	show_id	type	title	director	cast	country	date_added	release_year	rating	duration	listed_in	description
0	s1	TV Show	3%	NaN	João Miguel, Bianca Comparato, Michel Gomes, R...	Brazil	August 14, 2020	2020	TV-MA	4 Seasons	International TV Shows, TV Dramas, TV Sci-Fi &...	In a future where the elite inhabit an island ...
1	s2	Movie	7:19	Jorge Michel Grau	Demián Bichir, Héctor Bonilla, Oscar Serrano, ...	Mexico	December 23, 2016	2016	TV-MA	93 min	Dramas, International Movies	After a devastating earthquake hits Mexico Cit...
2	s3	Movie	23:59	Gilbert Chan	Tedd Chan, Stella Chung, Henley Hii, Lawrence ...	Singapore	December 20, 2018	2011	R	78 min	Horror Movies, International Movies	When an army recruit is found dead, his fellow...
3	s4	Movie	9	Shane Acker	Elijah Wood, John C. Reilly, Jennifer Connelly...	United States	November 16, 2017	2009	PG-13	80 min	Action & Adventure, Independent Movies, Sci-Fi...	In a postapocalyptic world, rag-doll robots hi...
4	s5	Movie	21	Robert Luketic	Jim Sturgess, Kevin Spacey, Kate Bosworth, Aar...	United States	January 1, 2020	2008	PG-13	123 min	Dramas	A brilliant group of students become card-coun...
...	...	...	...	...	...	...	...	...	...	...	...	...

List out Columns

This will list all of the columns in the dataset

```
netflix.columns
```

Basic Descriptive Statistics

If we want to run some descriptive statistics on all the numerical columns in the dataset use

```
netflix.describe()
```

See Unique Values in a Column

If you want to see the unique values in a column just do this

```
netflix["type"].unique()
```

Filtering DataFrames

Let's try and find only the movies made in Singapore. We're going to be using something called subsetting by boolean conditions.

```
netflix[netflix["country"] == "Singapore"]
```

Let's see if we can use multiple conditions. We'll be using those bitwise operators we mentioned earlier.

```
netflix[(netflix["country"] == "Singapore") & (netflix["rating"]
== "TV-MA")]
```

It is important that you enclose each case in parentheses.

DateTimes

As a Data Analyst, Dates and Times will be one of the most challenging datatypes you'll encounter. Different systems will store them differently, and there will often be different formats of dates in the same database. I could make an entire lesson on just Dates and Times so we'll just stick with the basics here.

```
netflix["computer_date"] = pd.to_datetime(netflix["date_added"])
```

We ended up with a couple of `NaT` values which are the datetime equivalents of `NaN`. Let's fill those in. We'll have to create a `datetime` variable using the `datetime` package.

```
import datetime netflix["computer_date"].fillna(datetime.datetime
(2020, 1,1), inplace=True)
```

I also want to tell you about something called Unix Time. You'll encounter it in a lot of datasets, it's basically a way to express a date and time as an integer. It counts up 1 for every second since January 1st, 1970, known as the Unix Epoch.

```
netflix["unix_time"] = netflix["computer_date"].astype(int)
```

Splitting Columns

Sometimes we'll encounter a DataFrame column with data we want to split on some delimiter. Let's try and split our "date_added" column into two columns that way we can compare the year media shows up on Netflix to when it was released.

In this case we'll want to split on a comma, and looking through the data, we know that we'll end up with two different columns.
We can split this data using a `str.split` method. We can also specify the two columns ahead of time like so.

```
netflix[["Date Part 1", "Date Part 2"]] = netflix["date_added"].s
tr.split(", ", expand=True)
```

`expand=True` is used to create new columns instead of creating a list of all the split elements in one column.

Renaming Columns

I named the year column we just split off `Date Part 2` on purpose. We will now rename it.

```
netflix.rename(columns={"Date Part 2": "Year"}, inplace=True)
```

This is another one of those methods that we can tell to change the DataFrame `inplace`. If you call the DataFrame now you'll notice that there is a `Year` column at the end.

Dealing with Null Values - Fill Na

Often you'll find `NaN` data in your DataFrame. This means "Not a Number" and is similar to a "Null" value. You often don't want this in your data, let's practice getting rid of it.

```
netflix["cast"].fillna(value="no cast", inplace=True)
```

You can also input a list into the `value` argument to gain more granular control of what you're switching your `NaN`s for.

Applying a Function to Every Row

Next, I'd like to count the number of genres that a film is listed in. This can be done multiple ways, but I'm going to use something called a lambda function and the `apply` method to accomplish this.

As a quick aside, a lambda function allows you to write a function in one line.

```
the_adder = lambda x: x + 210101
```

In this situation, the argument is `x` and you perform the calculations with whatever is tot he right of the colon.

We can use this to apply logic to every row of a DataFrame using either the `map` or the `apply` method.

The `map` method will pass each element of a Series (remember a series is basically a column of a DataFrame) to the lambda function. The `apply` method will pass an entire row to the lambda function. In short this means that if you want to perform calculations using multiple columns of data, use the `apply` method, otherwise use the `map` method.

```
netflix["genre_count"] = netflix["listed_in"].map(lambda x: len(x
.split(",")))
```

What Pandas is doing here, is it's passing each element of the Series `netflix["listed_in"]` to the lambda function as `x`. We then perform the logic with the code after the colon.

As an example of using the `apply` method let's try and count the number of cast members and genres.

```
netflix["nonsensical_columns"] = netflix.apply(lambda x: len(x["c
ast"].split(",")) + len(x["listed_in"].split(",")), axis = 1)
```

GroupBy, Aggregation, and Sorting Values

Oftentimes you'll want to perform aggregations on your data. In this case, you'll want to use `groupby`s and aggregations. Let's count the number of movies per country and sort the list in descending order.

```
netflix.groupby("country")["show_id"].count()
```

The above code will output something called a `groupby` object. This isn't too useful, so we'll need to turn the data back into a DataFrame using a method called `reset_index()`.

```
netflix.groupby("country")["show_id"].count().reset_index()
```

Now, let's sort our values.

```
netflix_movies_by_country = netflix.groupby("country")["show_id"]
.count().reset_index().sort_values(by="show_id", ascending=False)
```

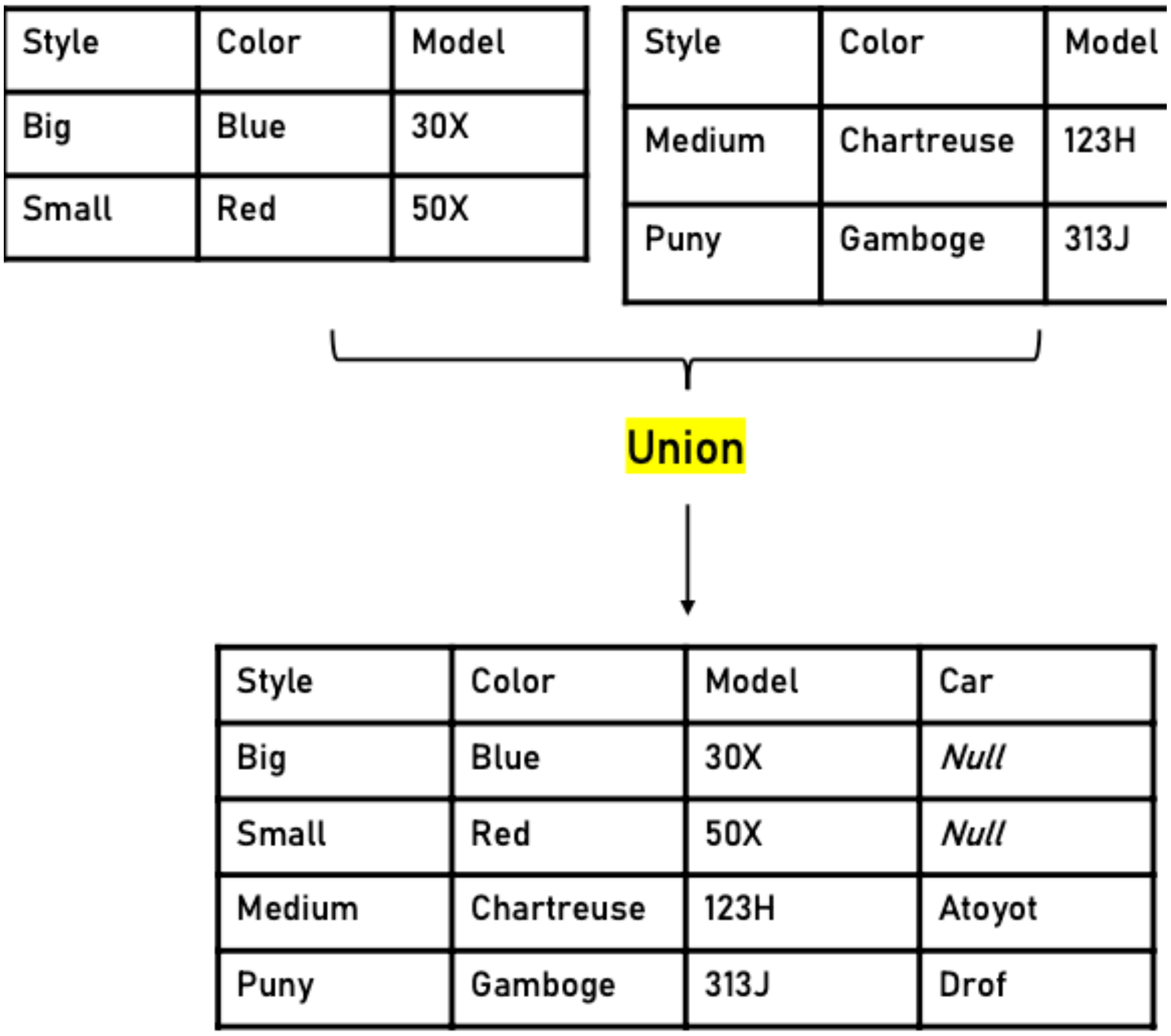
As you can see, Pandas makes it very easy to just string together commands to transform DataFrames. This is one of the great things about Pandas. You can use other aggregations if you'd like also.

Joins and Unions

Oftentimes we'll want to enrich our data by adding more data from other datasets to it. There are two major ways we can combine our datasets together: (The following visualizations will be coming from my FREE Tableau Course)

Unions

Put simply, a union (SQL Union) is the process of stacking two tables on top of one another. You will usually do this when your data is split up into multiple sections like an excel spreadsheet of a year’s sales split by month.



Style	Color	Model
Big	Blue	30X
Small	Red	50X

Style	Color	Model
Medium	Chartreuse	123H
Puny	Gamboge	313J

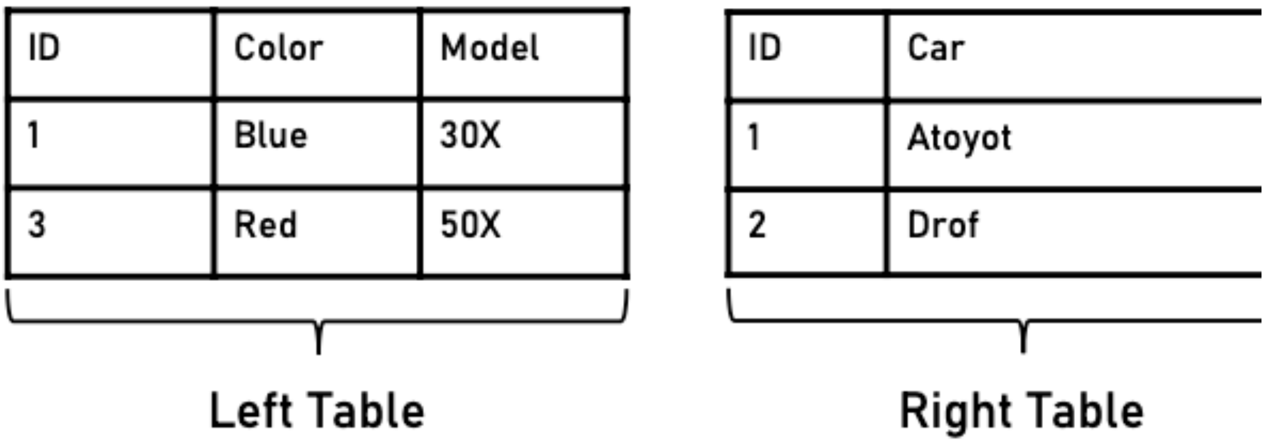
Style	Color	Model	Car
Big	Blue	30X	*Null*
Small	Red	50X	*Null*
Medium	Chartreuse	123H	Atoyot
Puny	Gamboge	313J	Drof

```
second_dataset = pd.read_csv(pwd + "/netflix_titles_second.csv")
new_dataset = pd.concat([netflix, second_dataset])
```

In the above code we Unioned the datasets by using the `pd.concat` method. This is an easy way to combine a bunch of datasets together all at once. You can use this with a for loop for example to combine a bunch of datasets together. I use this when querying APIs and getting the results as DataFrames, to combine all of the data at once.

Joins

Joins combine two tables horizontally. For a join, like a Union you have to have at least two tables, what we call our Left Table and our Right Table. You (mostly) have to have at least one matching column between the two tables, and you will match rows from these columns. The most common way to visualize the types of Joins are through Venn Diagrams.

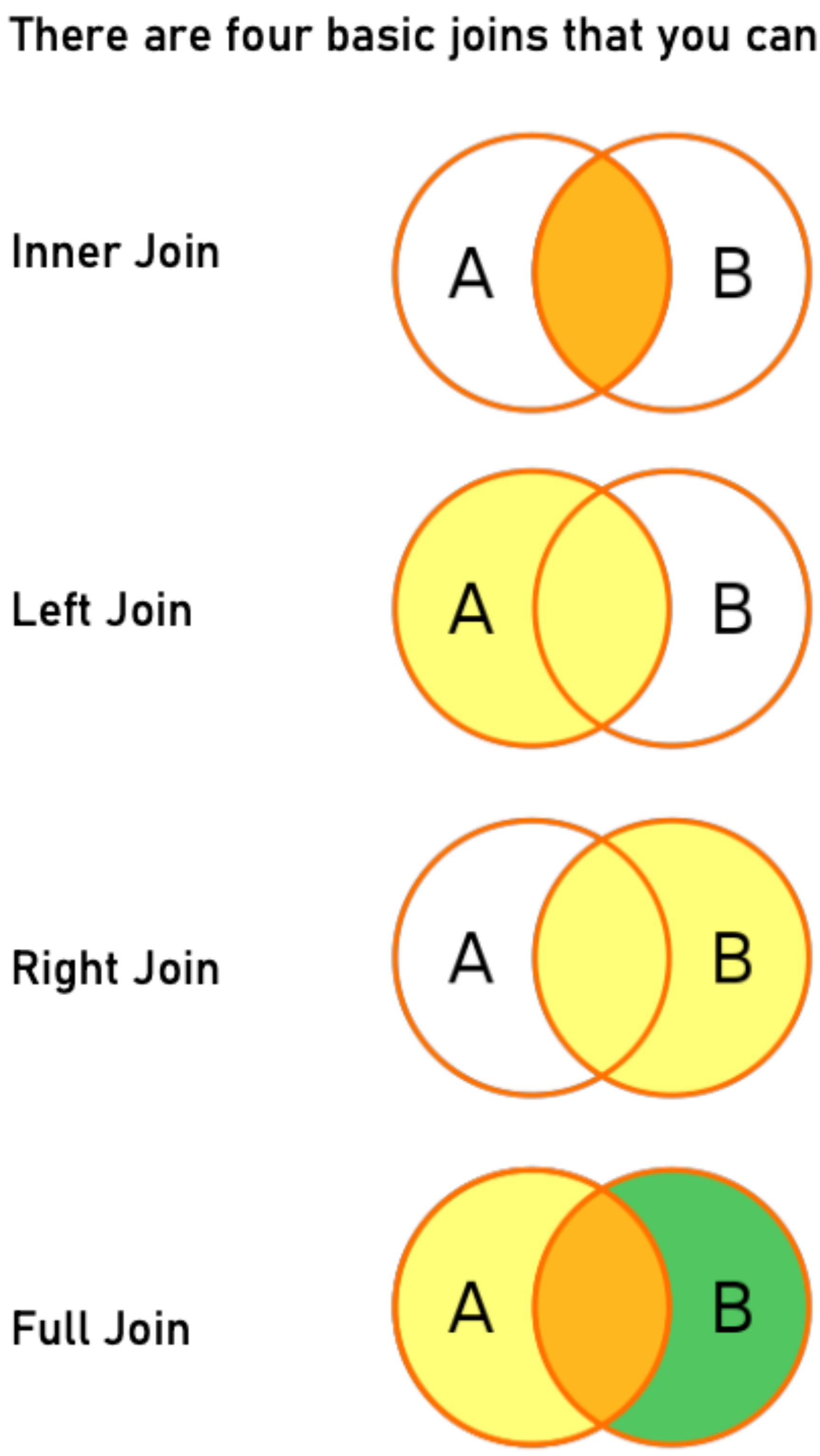


ID	Color	Model
1	Blue	30X
3	Red	50X

Left Table

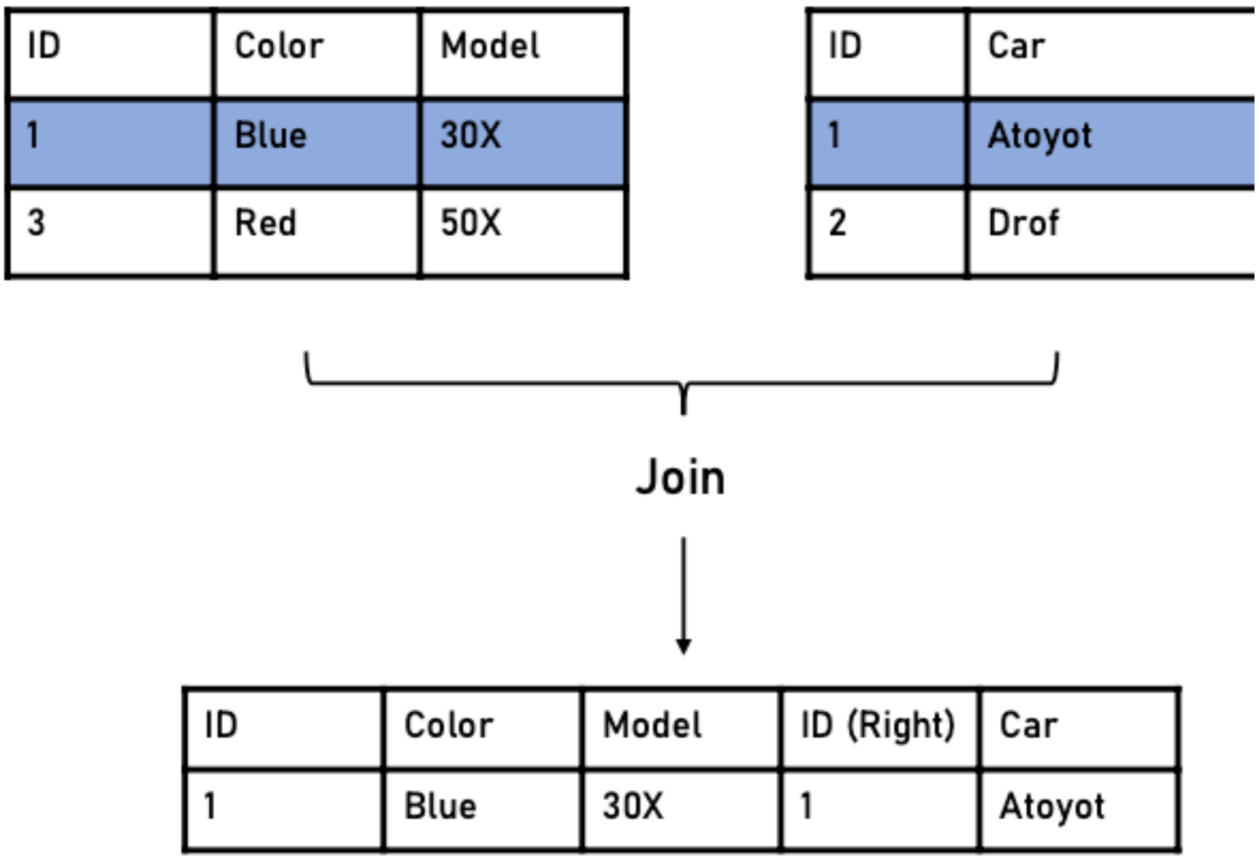
ID	Car
1	Atoyot
2	Drof

Right Table



You'll mostly be sticking to Left and Inner Joins. It's worth your time to learn more about Joins because they are some of the most powerful tools you can use to manipulate data. I use Joins basically every single day in my work. For this course we're going to stick with relatively simple Joins.

We're now going to do something called an Inner Join on the [ID] column which will only output exact matches from the [ID] column in our output.



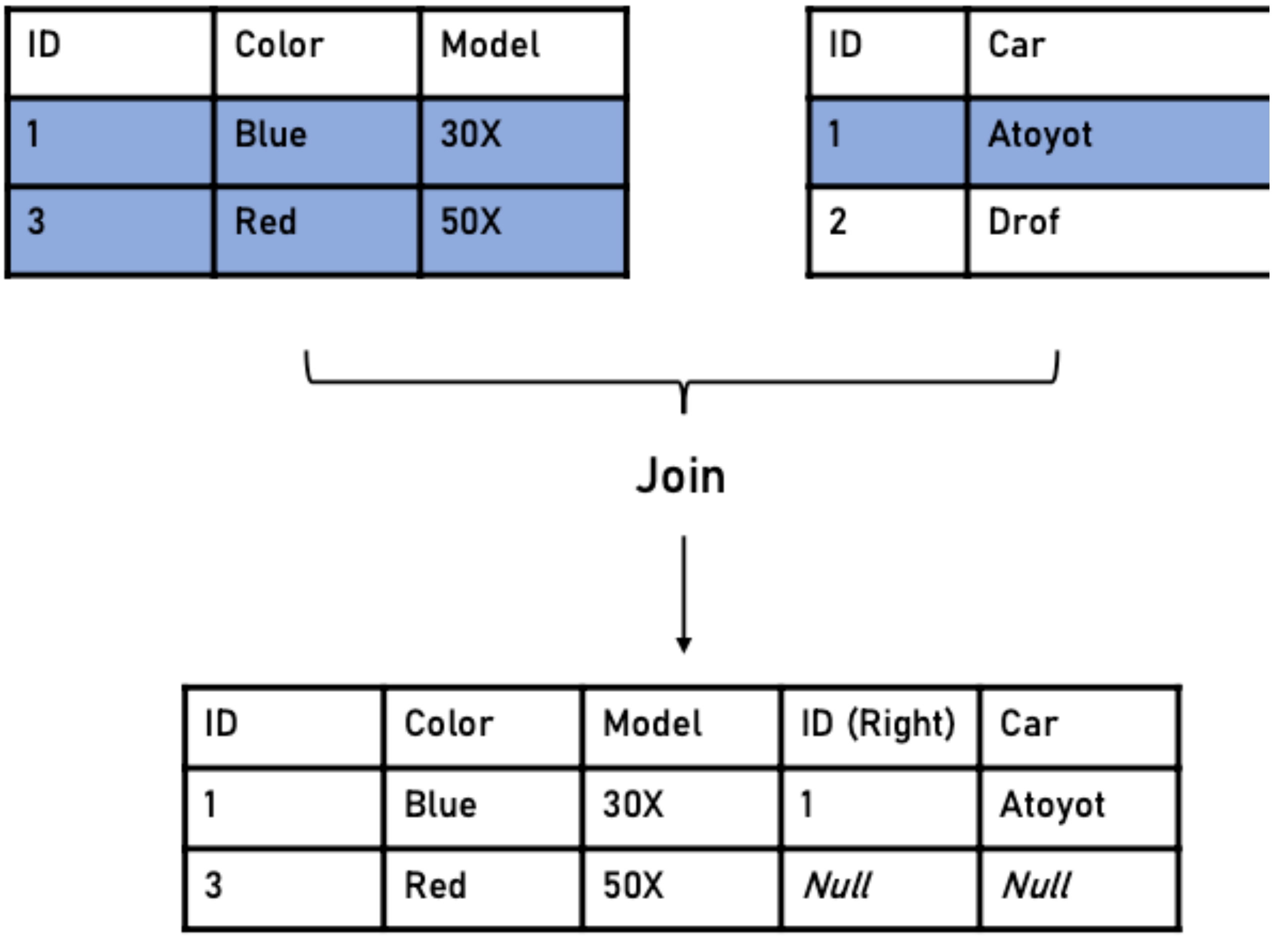
ID	Color	Model
1	Blue	30X
3	Red	50X

ID	Car
1	Atoyot
2	Drof

ID	Color	Model	ID (Right)	Car
1	Blue	30X	1	Atoyot

Inner Join

A Left Join keeps all of the data from your Left table and whatever matches from the Right table.



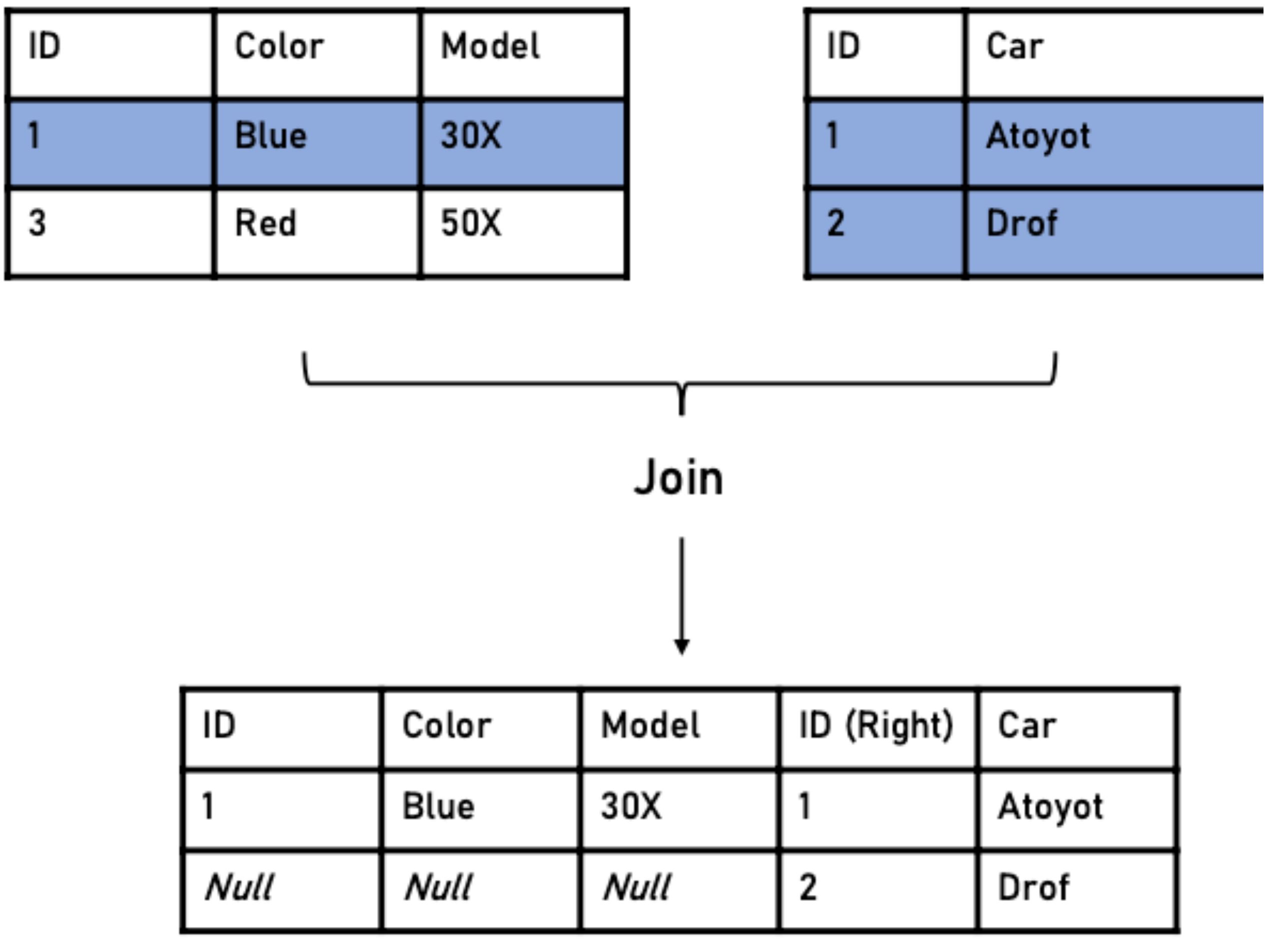
ID	Color	Model
1	Blue	30X
3	Red	50X

ID	Car
1	Atoyot
2	Drof

ID	Color	Model	ID (Right)	Car
1	Blue	30X	1	Atoyot
3	Red	50X	*Null*	*Null*

Left Join

A Right Join does the exact opposite and keeps everything from your Right table while only bringing in the matches from the Left table.



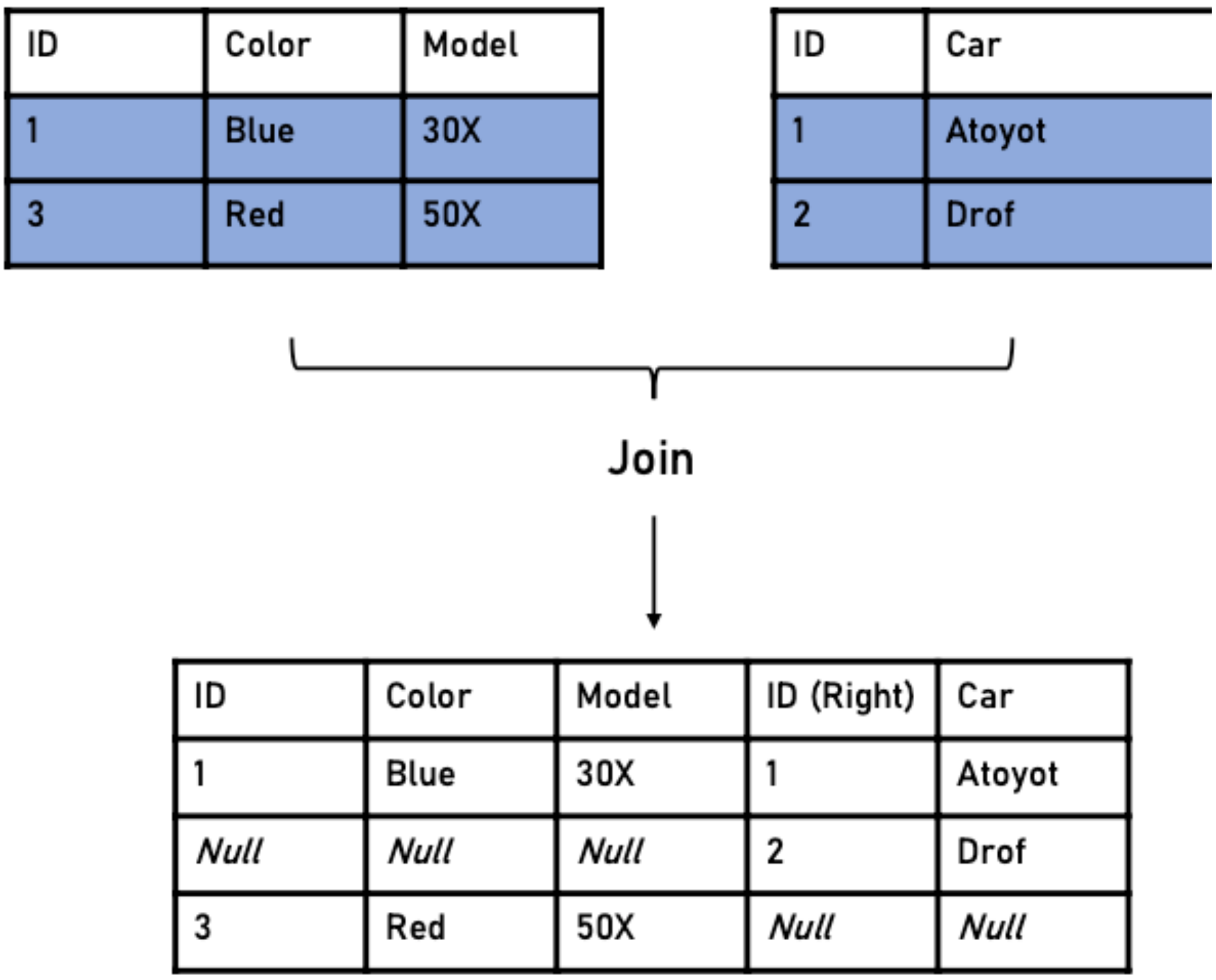
ID	Color	Model
1	Blue	30X
3	Red	50X

ID	Car
1	Atoyot
2	Drof

ID	Color	Model	ID (Right)	Car
1	Blue	30X	1	Atoyot
Null	*Null*	*Null*	2	Drof

Right Join

A Full Join brings in everything from both tables and matches whatever will match from the columns you specify.



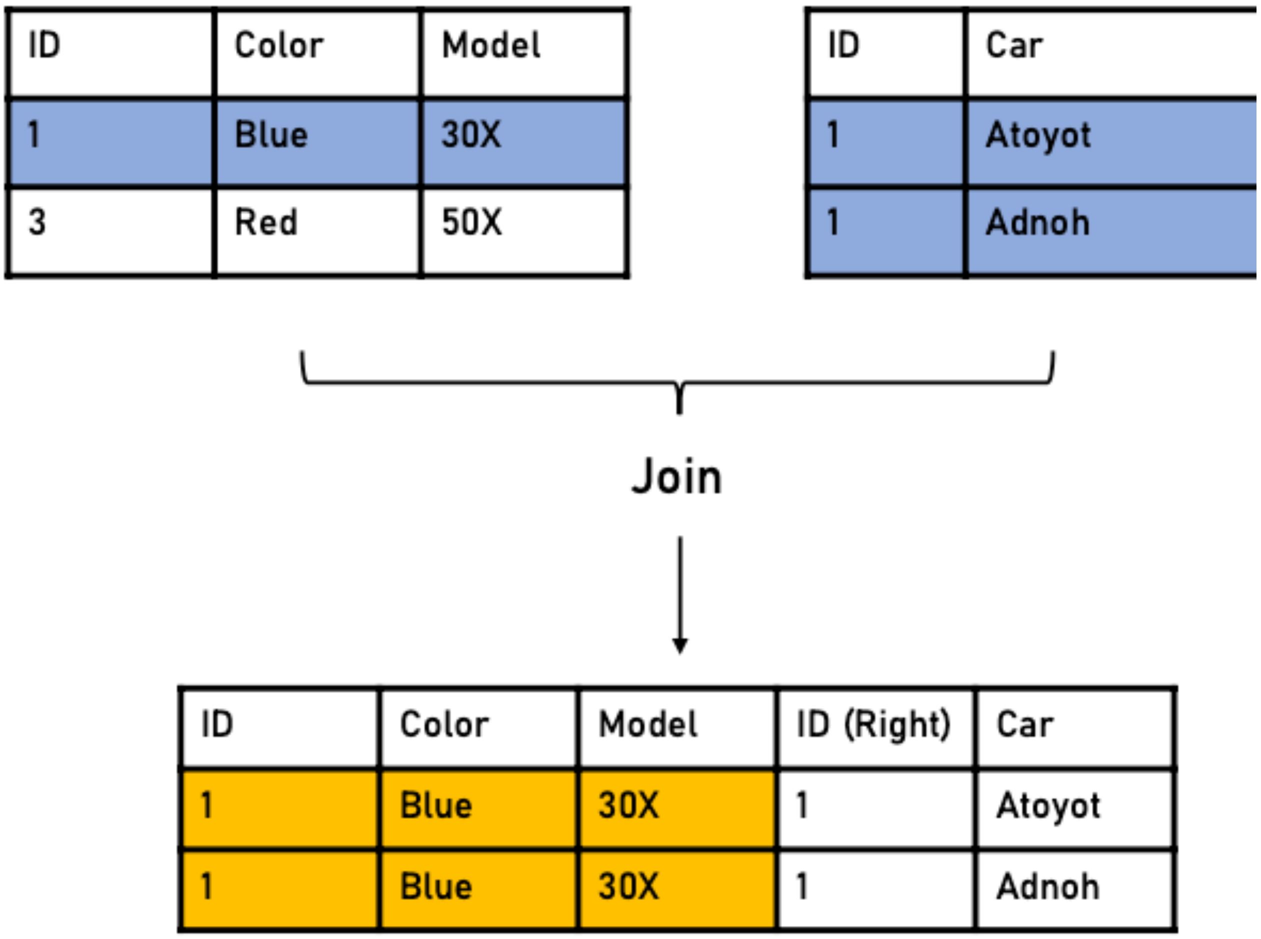
ID	Color	Model
1	Blue	30X
3	Red	50X

ID	Car
1	Atoyot
2	Drof

ID	Color	Model	ID (Right)	Car
1	Blue	30X	1	Atoyot
Null	*Null*	*Null*	2	Drof
3	Red	50X	*Null*	*Null*

Common Join Gotchas

Joins can get a bit tricky because of the potential for gotchas when joining two tables. The most common one is row duplication where you accidentally duplicate rows because the columns you're matching on have multiple potential matches. In the example below we're going to try an Inner Join. You'll notice the columns in Orange were duplicated.



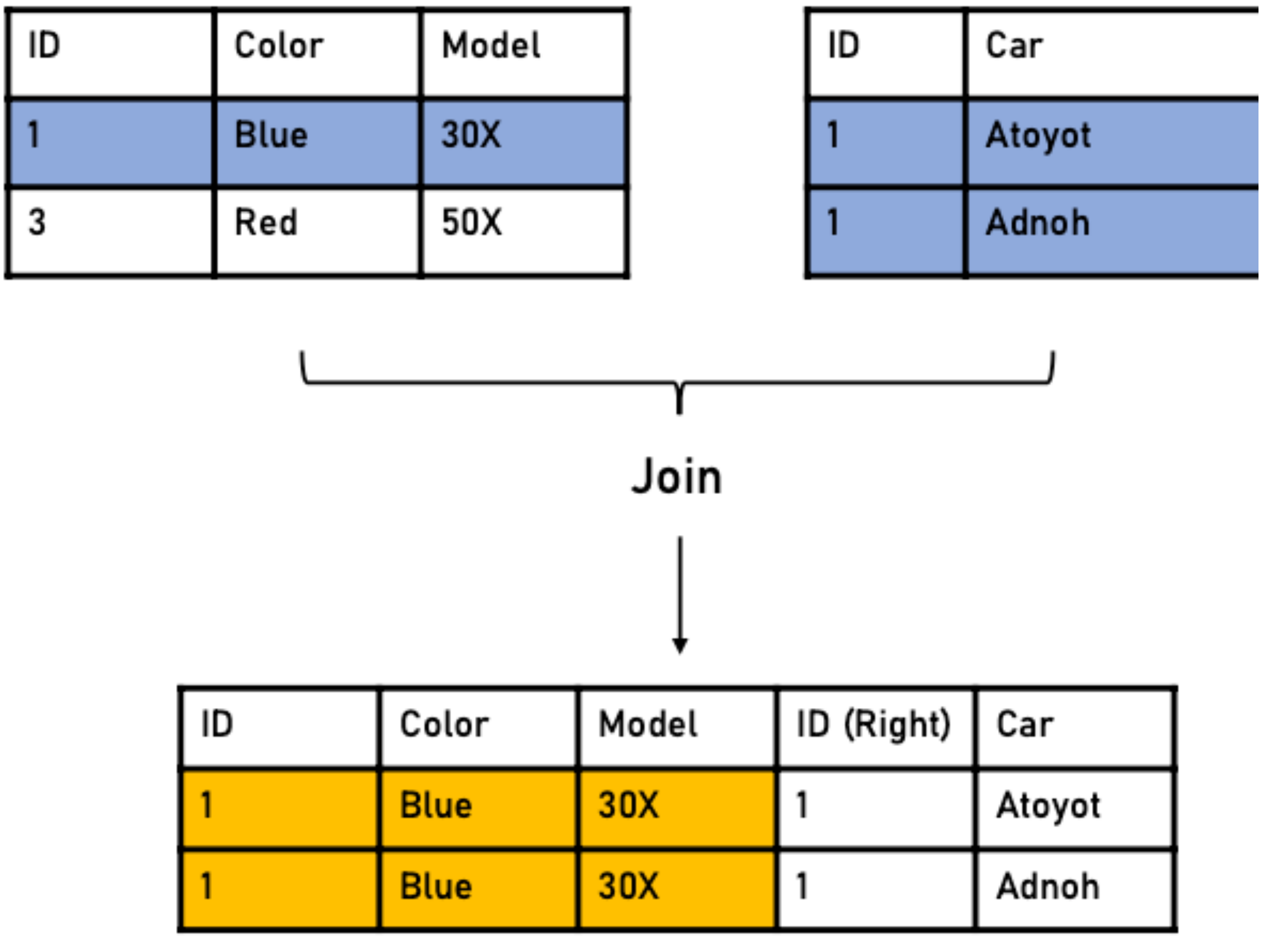
ID	Color	Model
1	Blue	30X
3	Red	50X

ID	Car
1	Atoyot
1	Adnoh

ID	Color	Model	ID (Right)	Car
1	Blue	30X	1	Atoyot
1	Blue	30X	1	Adnoh

Join duplication "error"

This isn’t an error per se but it is something to watch out for as it can cause you to duplicate data you don’t intend to duplicate.



ID	Color	Model
1	Blue	30X
3	Red	50X

ID	Car
1	Atoyot
1	Adnoh

ID	Color	Model	ID (Right)	Car
1	Blue	30X	1	Atoyot
1	Blue	30X	1	Adnoh

Join duplication "error"

Let's combine our data using a left join with the `netflix_movies_by_country` data we made earlier.

```
netflix_merged = pd.merge(left = new_dataset, right = netflix_mov
ies_by_country, how ="inner", left_on = ["country"], right_on = [
"country"])
```

Pivot and Melt

Pivot

Oftentimes you'll want to pivot or "melt" data. Pivoting data takes it from the "long" format that we are used to and puts it in a "wide" format that might be easier to read. Here's an example.

```
pivot_table = netflix.pivot_table(index="country", columns="type"
, values= "title", aggfunc='count', fill_value=0).reset_index()
```

```
type                                          country  Movie  TV Show
   0                                        Argentina     34       16
   1  Argentina, Brazil, France, Poland, Germany, De...      1        0
   2                                 Argentina, Chile      1        0
   3                           Argentina, Chile, Peru      1        0
   4                                Argentina, France      1        0
 ...                                              ...    ...      ...
 676                                        Venezuela      1        0
 677                              Venezuela, Colombia      1        0
 678                                          Vietnam      5        0
 679                                     West Germany      1        0
 680                                         Zimbabwe      1        0

681 rows × 3 columns
```

You typically want to pivot data when you want to present it to individuals as it will be easier to digest.

Melt

Oftentimes I receive data from clients and coworkers that's in a "wide" format that I'd like to convert to a "long" format in order to more easily analyze it in Python. Since the data we have is already in the long format, I'll use the example provided on the Pandas website to illustrate how to transform it.

```
df = pd.DataFrame({'A': {0: 'a', 1: 'b', 2: 'c'}, 'B': {0: 1, 1:
3, 2: 5}, 'C': {0: 2, 1: 4, 2: 6}}) df.melt(id_vars=['A'], value_
vars=['B'], var_name='myVarname', value_name='myValname')
```

Summary

Congratulations! You've reached the end of the course. You've learned a lot and I hope that I explained everything in a way that was easy to follow. Don't feel bad if you need to go through the content again, this stuff can be very difficult to understand on the first go. This course should give you a good overview to get you started with using Python for Data Analysis. After you finish this, I have a video here where I use many of the skills taught in this course to solve a real world issue with real (anonymized) data. The data is all available to you so you can follow along.

These skills are very useful to anyone who works regularly with data and once you master them you can comfortably charge in the upper five-figures for these skills and by combining these with knowledge of another tool or area like Tableau or an industry, you can comfortably charge six-figures.

This is just the start of your journey, so if you need any help please feel free to let me know!