**Getting acquainted with DataFrames in Python**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

When it comes to exploring data with Python, DataFrames makes analyzing and manipulating data for analysis easy. This article will take a look at some of its ins and outs when it comes to working with them.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



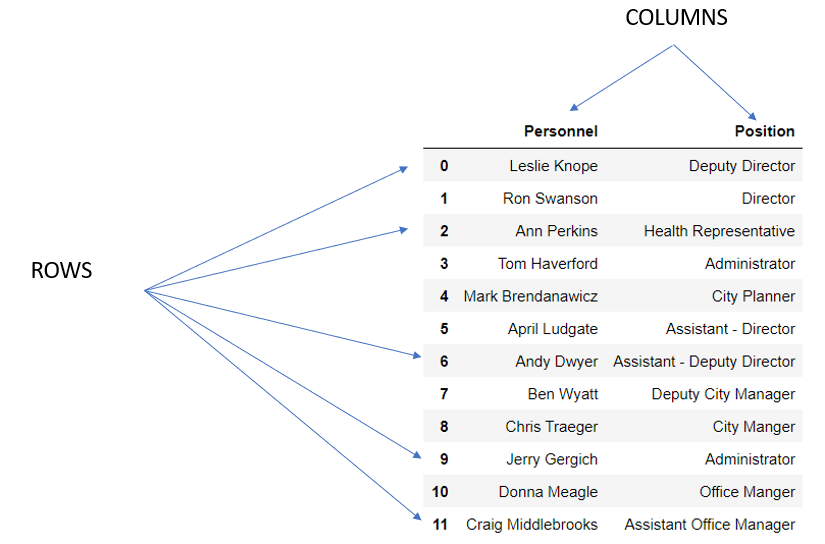
Python is a powerful tool when it comes to working with data. Qualities like its scalability and variety of libraries for data analysis and data science applications often comes to mind. However, what’s often underappreciated but is highly values, is its ease in data manipulation with its expressive and flexible data structures. One of these structures is a DataFrame.

**What’s a DataFrame?**

To start, it’s important to know that there are a variety of different structures that data can be configured as whereby these individual qualities provides value in certain context. For the majority of cases, most data are in [tabular form](https://papl.cs.brown.edu/2016/intro-tabular-data.html) (i.e., data structured into rows representing a single entry). You are likely already familiar with this if you’ve ever worked with an Excel spreadsheet or a SQL table. Aggregates of each of these rows that represents a given data entry and its properties are formed into a 2-dimentional structure where titled columns consist of values of the same property.

These structures have several unique qualities:

1. Rows = representing a singular data entry point
2. Columns = corresponding to a grouping relating to a singular quality of each given data point that are usually titled
3. Index = a unique identifier for each data entry



Although these can be named differently depending on the programming language or application tool that is being used, these structures are called DataFrames in Python. The principal library used in working with these structures is [Pandas](https://pandas.pydata.org/).

**How do you make a DataFrame?**

When it comes to creating a DataFrame, it can either be imported from an external file or created from scratch on Python.

METHOD 1- IMPORT DATA FROM A FILE

In the real world, a dataset is often read into Python via an external source that curated it. These datasets can be found in multiple types of files, but are most commonly in the form of comma separated value files (CSVs). Fortunately in the Pandas library, it has a function that works to convert the data in this format into a DataFrame called [pandas.read\_csv()](https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html). The only major argument that it requires is a pathway that outlines where the file exists.

One pathway may be from the web (*i.e., from an API or a GitHub repository*)

```

import pandas as pd

import numpy as np

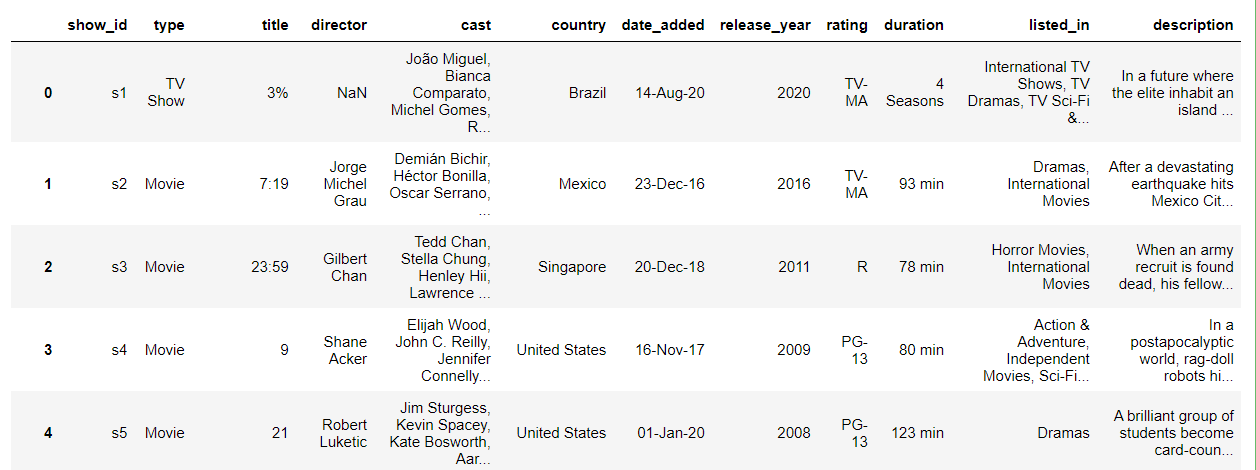
# While not necessarily the case, you’ll often need to load the numpy library when working with the pandas library

url = ”<https://raw.githubusercontent.com/Vibe1990/Netflix-Project/main/netflix_title.csv>”

pd.read\_csv(url)

# When providing the URL pathway, this will need to be in the form of a raw string, otherwise it will result in an error

```



Alternatively, if a file were instead stored on your computer in a working directory, then the path would be adjusted accordingly. In this process, we could use [either the relative or full path](https://docs.oracle.com/javase/tutorial/essential/io/path.html#:~:text=A%20path%20is%20either%20relative,required%20to%20locate%20the%20file.&text=A%20relative%20path%20needs%20to,foo%20is%20a%20relative%20path.) in specifying the pathway to retrieve a given file as the function is able to decipher the difference between the two without an issue.

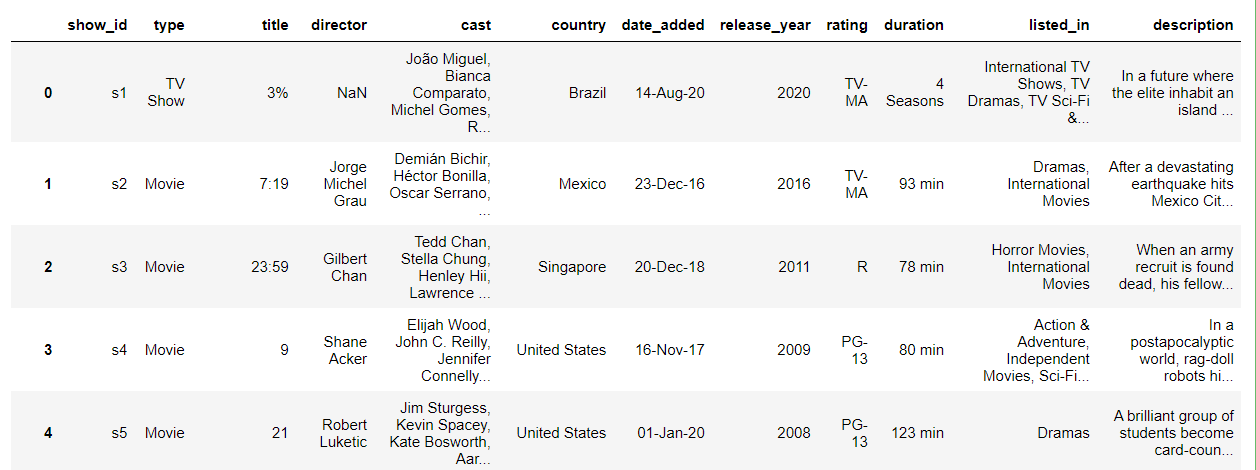
```

# Assuming you've set up your notebook to have the desired working directory set

%cd # WORKING DIRECTORY HERE

pd.read\_csv(“netflix\_title.csv”)

```



Although CSV files are the most common, there are a number of different functions that are available in Pandas to read in file of various types into a data frame that operates with the same general process:

|  |  |
| --- | --- |
| **File type** | **Function in Pandas** |
| [JSON](https://www.json.org/json-en.html) | [read\_json()](https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#io-json-reader) |
| [HTML](https://en.wikipedia.org/wiki/HTML) | [read\_html()](https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#io-read-html) |
| [XML](https://www.w3.org/standards/xml/core) | [read\_xml()](https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#io-read-xml) |
| [SQL](https://en.wikipedia.org/wiki/SQL) | [read\_sql](https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#io-sql)() |
| [Excel](https://en.wikipedia.org/wiki/Microsoft_Excel) | [read\_excel](https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#io-excel-reader)() |

METHOD 2 – CREATING FROM SCRATCH

While not the most commonly performed method, you can certainly create a data frame from scratch by inputting data. This is accomplished with the [pandas.DataFrame()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html#pandas.DataFrame) function, which takes its data input argument and converts it into a data frame. The pandas.DataFrame() function is quite robust in that it can take in a variety of different data inputs such as:

1. Nothing – just will make an empty DataFrame that you can populate with data later

```

data = pd.DataFrame()

data

```

1. A dictionary of [ndarrays](https://www.geeksforgeeks.org/numpy-ndarray/) / [lists](https://www.tutorialspoint.com/python/python_lists.htm)

```

Pawnee\_city\_hall = {

"Personnel": ["Leslie Knope", "Ron Swanson", "Ann Perkins", "Tom Haverford", "Mark Brendanawicz", "April Ludgate", "Andy Dwyer", "Ben Wyatt", "Chris Traeger","Jerry Gergich", "Donna Meagle", "Craig Middlebrooks"],

"Position": ["Deputy Director", "Director", "Health Representative", "Administrator", "City Planner", "Assistant - Director", "Assistant - Deputy Director", "Deputy City Manager", "City Manger", "Administrator", "Office Manger", "Assistant Office Manager"]

}

pd.DataFrame(Pawnee\_city\_hall)

```



1. A dictionary of [series](https://pandas.pydata.org/docs/reference/api/pandas.Series.html) (*a 1-dimensional array of data with an axis label*)

```

personnel = pd.Series(["Leslie Knope", "Ron Swanson", "Ann Perkins", "Tom Haverford", "Mark Brendanawicz", "April Ludgate", "Andy Dwyer", "Ben Wyatt", "Chris Traeger","Jerry Gergich", "Donna Meagle", "Craig Middlebrooks"])

position = pd.Series(["Deputy Director", "Director", "Health Representative", "Administrator", "City Planner", "Assistant - Director", "Assistant - Deputy Director", "Deputy City Manager", "City Manger", "Administrator", "Office Manger", "Assistant Office Manager"])

pd.DataFrame({"Names": personnel, "Job": position})

```



1. A list of lists

```

data = [

["Leslie Knope", "Deputy Director"],

["Ron Swanson", "Director"],

["Tom Haverford", "Administrator"],

["April Ludgate", "Assistant-Director"],

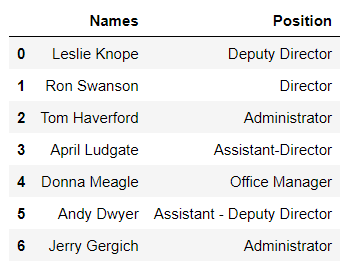
["Donna Meagle", "Office Manager"],

["Andy Dwyer", "Assistant - Deputy Director"],

['Jerry Gergich', "Administrator"]]

pd.DataFrame(data, columns = ["Names", "Position"])

```



1. A list of dictionaries

```

raptors = [{"Player": "Pascal Siakim", "PPG": 23.7, "College": "University of New Mexico", "is\_starting\_five": True},

{"Player": "Fred VanVleet", "PPG": 20.1, "College":"Whicita State", "is\_starting\_five": True},

{"Player": "Scottie Barnes", "PPG": 15.1, "College": "Florida State", "is\_starting\_five": True},

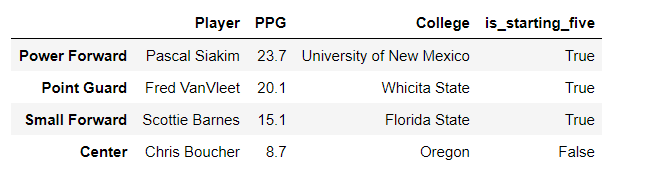
{"Player": "Chris Boucher", "PPG": 8.7, "College": "Oregon", "is\_starting\_five": False}

]

raptors = pd.DataFrame(raptors, index = ['Power Forward', "Point Guard", "Small Forward", "Center"])

raptors

```



***NOTE:*** the panda.DataFrame function also has the index and column argument that’s used to name the row index and column titles respectively.

Although the same data types are used in the above examples (strings), data frames can consist of a variety of different data types such as integers, floats, lists, datetimes, bools, list, etc.

```

lakers = {

"player": ['Lebron James', Russell Westbrook', 'Anthony Davis', 'Dwight Howard', 'Avery Bradley', 'DeAndre Jordan', 'Carmelo Anthony', 'Austin Reaves', 'Kent Bazemore', 'Malik Monk', 'Stanley Johnson', 'Trevor Ariza', 'Wayne Ellington', 'Talen Horton-Tucker'],

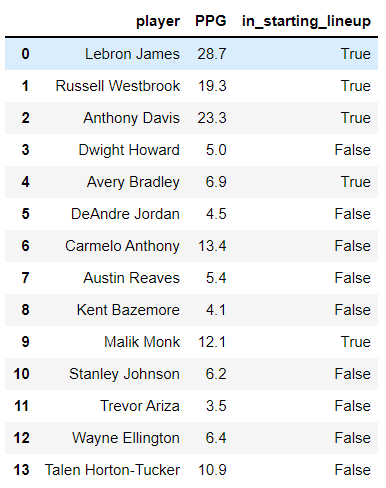
"PPG": [28.7, 19.3, 23.3, 5, 6.9, 4.5, 13.4, 5.4, 4.1, 12.1, 6.2, 3.5, 6.4, 10.9],

"in\_starting\_lineup": [True, True, True, False, True, False, False, False, False, True, False, False, False, False,]

}

pd.DataFrame(lakers)

```



**Exploring a DataFrame**

Since Python is an object-oriented programming language, creating a data frame means creating an object of the DataFrame class. This also means that there are a number of different attributes that we can explore and methods that can be applied to the data frame. While these are more often utilized in situations where you aren’t familiar of the dataset (*say from importing it from somewhere*), they are nonetheless useful.

Whenever a dataset is loaded into Python as a DataFrame, it’s best to take a look at its structure. There are a number of different attributes that can provide that info such as:

* [DataFrame.shape](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.shape.html#pandas.DataFrame.shape): returns a tuple indicating the number of rows and columns of the DataFrame
* [DataFrame.size](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.size.html#pandas.DataFrame.size): returns an integer value of the number of datapoints in the dataset

```

print(netflix.shape)

print(netflix.size)

```



If you were to explore the axes of the DataFrame, you may do so by having an array return the listed columns and index via [DataFrame.columns](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.columns.html#pandas.DataFrame.columns) and [DataFrame.index](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.index.html#pandas.DataFrame.index).

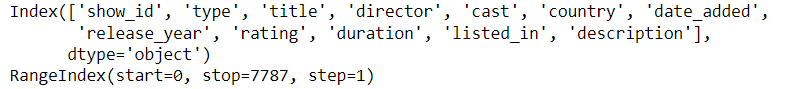
```

print(netflix.columns)

print(netflix.index)

print(netflix.dtypes)

```

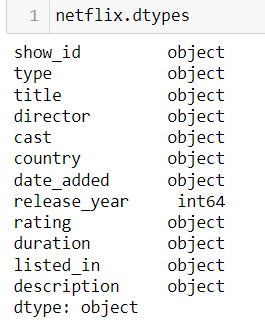


On the other hand, it may be useful to look at the different types of data that makes up the dataset. In these situations, the [DataFrame.dtypes](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dtypes.html#pandas.DataFrame.dtypes) is used.

```

netflix.dtypes

```



**Manipulating a DataFrame**

After getting a handle of what the make up of the DataFrame is, it’s time to do some real work! Principally, this involves manipulating it as part of the data cleaning and data wrangling process, just prior to the actual analysis. Now there are a number of basic operations that should be in everyone’s repertoire but the first one is being able to access and isolate a given segment of a DataFrame.

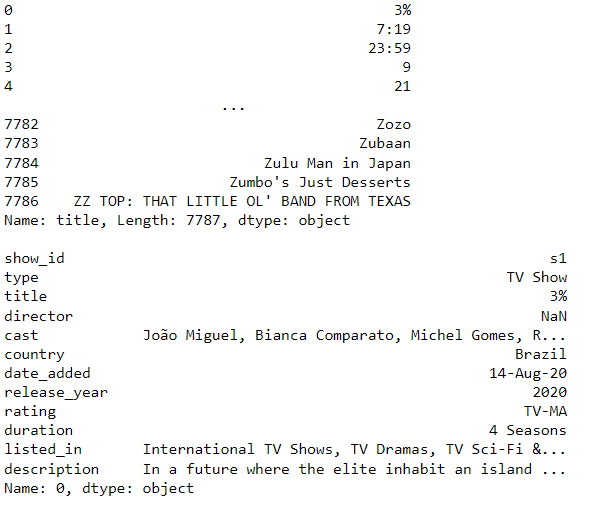
For segmenting a DataFrame, it is accomplished using either the [DataFrame.loc](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.loc.html#pandas.DataFrame.loc) attribute or [DataFrame.iloc](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html#pandas.DataFrame.iloc) attribute where the input dictate which rows or columns are extracted ([rows: columns]). If a column needs to be isolated, then the process would be to use square brackets with the name of the given column.

```

print(netflix[‘title’]) # extract the title column

print(netflix.loc[0]) # extract the first row of the dataframe

```



In the event that you need to extract multiple row or columns, then we can do so by using the [slice method](https://realpython.com/lessons/indexing-and-slicing/) which involves using a “:” that indicates a continuous range with the end range being exclusive (i.e., not included) or by inputting criteria within square brackets in a similar manner as [indexing with Boolean with NumPy](https://numpy.org/doc/stable/user/quickstart.html#indexing-with-boolean-arrays).

```

# Select the first two row of the raptors DataFrame

raptors.iloc[0:2]

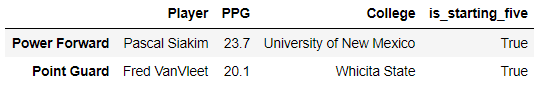
# Select the last three columns of the raptors DataFrame

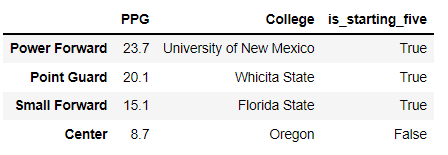
raptors.iloc[:, 1:4]

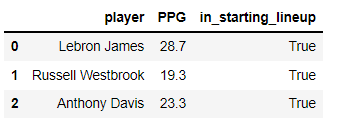
# Select players that averaged more than 15 PPG on the Laker DataFrame

lakers[lakers[‘PPG’] > 15]

```







While the above examples are simplistic, it is possible to make it more powerful and sophisticated with the use of operators such as AND (&), OR (|), NOT (!=) or EQUAL TO (==).

```

print(ufc\_champs[(ufc\_champs['country'] == "Brazil") & (ufc\_champs['weightclass'] == "Light Heavyweight")])

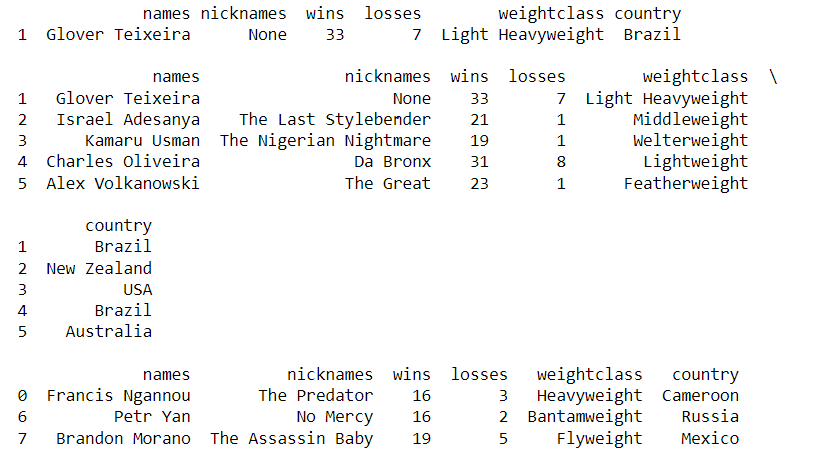
print("")

print(ufc\_champs[(ufc\_champs['country'] == "Brazil") | (ufc\_champs['losses'] < 2)])

print("")

print(ufc\_champs[(ufc\_champs['wins'] < 20) & (ufc\_champs["country"] != "USA")])

```



Aside from filtering out a DataFrame or segmenting it, it is also possible to use the DataFrame.iloc() and DataFrame.loc() attributes in changing a particular value as well.

```

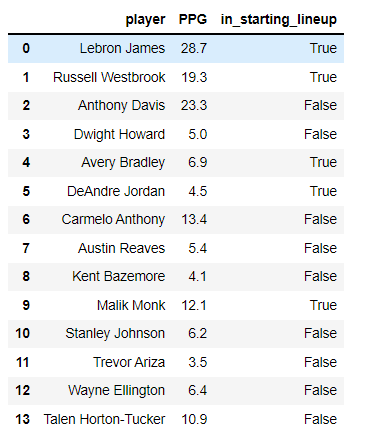
# The Lakers made a change in the starting 5 lineup where we replace Anthony Davis with DeAndre Jordan

lakers.iloc[2,2] = False

lakers.loc[5,"in\_starting\_lineup"] = True

lakers

```



In some situations, it may be necessary to insert or delete data from a DataFrame. To insert or delete a row, it can be accomplish with the [append()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.append.html) and [drop()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html#pandas.DataFrame.drop) methods. With the append method, you would use a [panda.Series](https://pandas.pydata.org/docs/reference/api/pandas.Series.html) object that matches the dimensions of a DataFrame as the argument for the function. For the drop method, all that needs to be stated is the index/column label within the DataFrame that wound need to be dropped. It should be noted that with both methods, it does contain the axis argument that specify whether to add or drop a row or column

```

# Say we create a DataFrame consisting of UFC champions that held the title at the end of 2021 with their monikers and win-loss record

ufc\_champs = {

"names": ['Francis Ngannou', "Glover Teixeira",

"Israel Adesanya", "Kamaru Usman",

"Charles Oliveira", "Alex Volkanowski",

'Aljamain Sterling', "Brandon Morano",

"Julianna Pena", "Valentina Shevchenko", "Rose Namajunas"],

"nicknames": ['The Predator', None, "The Last Stylebender",

"The Nigerian Nightmare", "Da Bronx", "The Great",

"Funk Master", "The Assassin Baby", "The Venezulean Vixen",

"The Bullet", "Thug"],

"wins": [16,33,21,19,31,23,20,19,11,22,11],

"losses": [3,7,1,1,8,1,3,5,4,3,4],

"weightclass": ['Heavyweight', "Light Heavyweight", "Middleweight",

'Welterweight', "Lightweight", "Featherweight",

'Bantamweight', "Flyweight", "Bantamweight", "Flyweight", "Strawweight"]

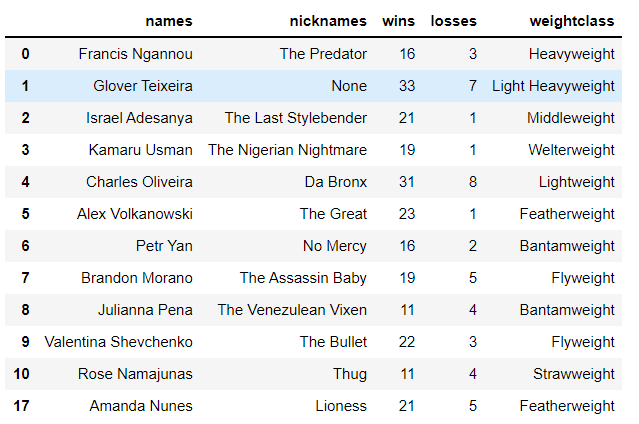
}

ufc\_champs = pd.DataFrame(ufc\_champs)

# In making this DataFrame, we forgot to include the Women’s Featherweight champ

ufc\_champs.append(pd.Series(data = [“Amanda Nunes”, “Lioness”, 21, 5, “Featherweight”], index = ufc\_champs.columns, name = 17))

```



```

ufc\_champs.drop([0], axis = 0) # drops first row

```



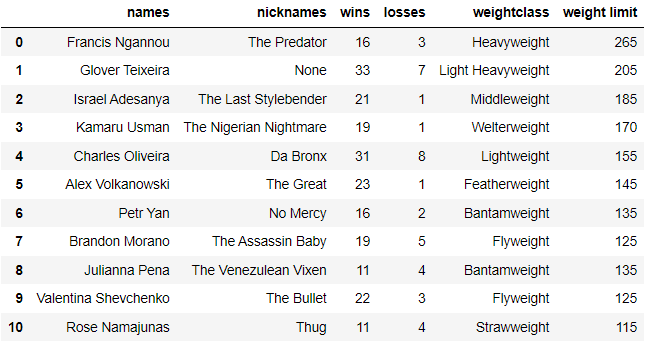
In the case for adding a column, the process would be similar to that of adding an item into a dictionary.

```

weight\_limit = [265,205,185,170,155,145,135,125,135,125,115]

ufc\_champs[‘weight\_limit’] = weight\_limit

```



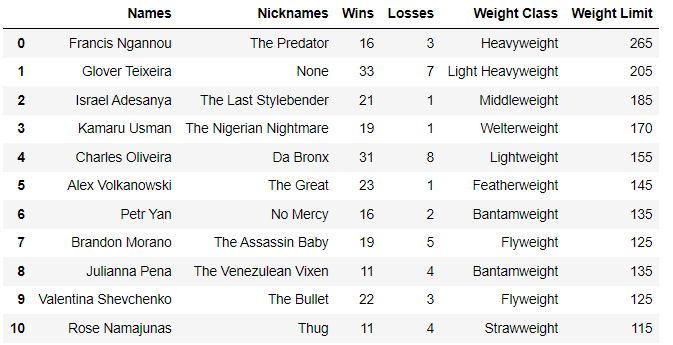
Sometimes with data sets, the labels used in identifying a column may not be accurate describe its property. To change these labels, we can use the [DataFrame.rename()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.rename.html#pandas.DataFrame.rename) method which takes in the index argument (specifies the labels for the index in a dictionary-like format), columns argument (specifies the labels for the columns in a dictionary-like format), and the inplace argument that determines whether to return a new DataFrame.

```

# Let’s say we need to rename the columns of the UFC champ DataFrame by capitalizing the labels

ufc\_champs.rename(columns = {“names”: ”Names”, “nicknames”: “Nicknames”, “wins”: “Wins”, “losses”: “Losses”, “weightclass”:”Weight Class” , “weight limit” : “Weight Limit”})

```



Lastly, there may be some cases where the current makeup of the data set needs to be reshaped in order to make it suitable for data analysis. While it could certainly be possible to manually remake another DataFrame, it’ll be easier to transform it. In Pandas, there are three different transformation functions that can be used to reshape the DataFrame:

METHOD 1 – PIVOTING

With this transformation, it is essentially taking a longer format DataFrame and making it broader. Often this is a result of having a unique identifier repeated along multiple rows for each subsequent entry. One method to derive a newly formatted DataFrame is by using [DataFrame.pivot](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.pivot.html). This method requires defining which of the data columns will be used as the new index and index as well as values for the DataFrame.

```

# Say we are creating a DataFrame that maps out the voting for the season MVP of the 2021-2022 NBA season from 5 different sports journalist/reporter/pundits in sports media

mvp\_vote\_2022 = pd.DataFrame({

"Voter": ['Kenny Smith', 'Kenny Smith', 'Kenny Smith', 'Kenny Smith', 'Kenny Smith',

"Charles Barkley", "Charles Barkley", "Charles Barkley", "Charles Barkley", "Charles Barkley",

"Ernie Johnson", "Ernie Johnson", "Ernie Johnson", "Ernie Johnson", "Ernie Johnson",

"Michael Wilbon", "Michael Wilbon", "Michael Wilbon", "Michael Wilbon", "Michael Wilbon",

"Doris Burke", "Doris Burke", "Doris Burke", "Doris Burke", "Doris Burke"

],

"Player": ['Steph Curry', 'Lebron James', 'Chris Paul', 'Kevin Durant', 'Giannis Antetokounmpo',

'Steph Curry', 'Nikola Jokic', 'Giannis Antetokounmpo', 'Chris Paul', 'DeMar DeRozan',

'Steph Curry', 'Giannis Antetokounmpo', 'Kevin Durant', 'Nikola Jokic', 'Joel Embid',

'Kevin Durant', 'Giannis Antetokounmpo', 'Nikola Jokic', 'Steph Curry', 'DeMar DeRozan',

'Kevin Durant', 'Giannis Antetokounmpo', 'Steph Curry', 'Nikola Jokic', 'DeMar DeRozan',],

"Placing": [1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5]

})

# Say if we need to switch the index to indicate voters as the index values and columns with the MVP placings from each voter

mvp\_vote\_2022.pivot(index = “Voter”, columns = “Placing”)

```



With the DataFrame.pivot method, it requires that you cannot have rows with duplicate values for a given column. If this uniqueness isn’t guaranteed, an alternative approach would be to use the [DataFrame.pivot\_table()](https://pandas.pydata.org/docs/reference/api/pandas.pivot_table.html) function instead. With this approach, it requires arguments used to specify the index, column, and values. Uniquely, this function also has an additional argument “aggfunc” (*default is numpy.mean*) which passes a function to aggregate the values of a DataFrame.

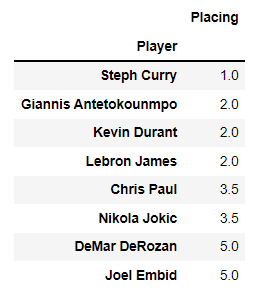
```

# Create a new DataFrame that shows the median placing for NBA MVP

mvp\_vote\_2022.pivot\_table(index = “Player”, values = “Placing”, aggfunc = np.median).sort\_values(by=”Placing”)

# the .sort\_values() method is used to arrange the DataFrame by some existing variable in either ascending or descending order

```



METHOD 2 – STACKING/UNSTACKING

Sometimes a DataFrame may have multiple indices that’ll look something like this:

```

array = [["Month 1", "Month 2", "Month 3", "Month 4"],

['Squat', "Squat", "Squat", "Squat"]]

array2 = [['Arnold', "Arnold","Larry", "Larry"],

["Before", "After", "Before", "After"]]

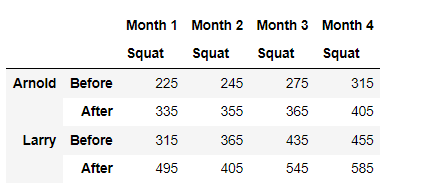
values = [[225, 245, 275, 315], [335, 355, 365, 405], [315, 365, 435, 455], [495, 405, 545, 585]]

values = np.array(values)

effect\_of\_ped = pd.DataFrame(data = values, index = array2, columns = array)

effect\_of\_ped

```

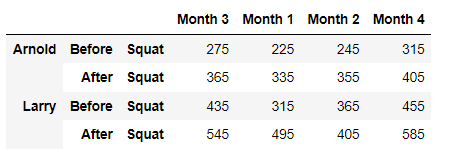


It is often difficult to be able to make sense of the data or address it for analysis. So, functions such as [stack()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.stack.html#pandas.DataFrame.stack) or [unstack()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.unstack.html#pandas.DataFrame.unstack) makes it possible to make it longer or broader, respectively.

```

effect\_of\_ped.stack()

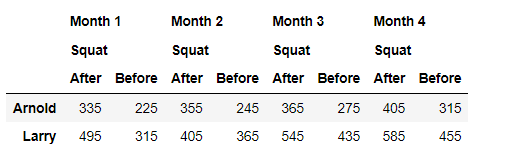
```



```

effect\_of\_ped.unstack()

```



METHOD 3 – MELTING

Best considered as unpivoting a DataFrame, it works by essentially making a wide format DataFrame to a long format. This usually occurs when more than one column works as an identifier for a given analysis. In order to transform the DataFrame to a longer format, we’ll need to use the [DataFrame.melt()](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.melt.html#pandas.DataFrame.melt) function, which requires establishing which columns are to be used as the identifier variable and the columns to “unpivot” to correspond to the values for said identifier.

```

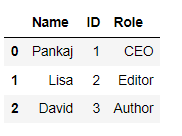
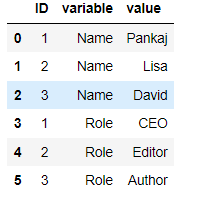
data = pd.DataFrame({"Name": ["Pankaj", "Lisa", "David"], "ID": [1, 2, 3], "Role": ["CEO", "Editor", "Author"]})

data

# Let's say the identifier is ID and the values would be Name + Role

pd.melt(data, id\_vars = ["ID"], value\_vars = ['Name', 'Role'])

```

**CONCLUSION**

So far, we’ve only scratched the surface of DataFrames. There are many more functions and methods that can operate on these data structures within Python to gain deeper insights into your data. You can find these in the [Pandas DataFrame reference guide](https://pandas.pydata.org/docs/reference/frame.html) . However, a great place to start that dive is with the [Pandas and NumPy Fundamentals course on Data Quest](https://app.dataquest.io/course/pandas-fundamentals). Once you get a handle on it, dive deeper with working with data in Python in some of the other courses in the [Data Analyst pathway](https://www.dataquest.io/path/data-analyst/).