Introduction to Python for Data Visualization

Hello, and welcome to the Introduction to Python for Data Visualization workshop. In this tutorial, we will learn how to create simple visualizations using the matplotlib and pandas libraries.

Importing Libraries and Assigning Aliases

To begin, we have to import the matplotlib and pandas libraries into the Jupyter notebook. This is done using **import** followed by the name of the library you wish to import. Completing this step at the beginning of your code ensures that the libraries are loaded and ready for use.

We will also assign each library an alias. Matplotlib will be given the alias "plt" and pandas will be given the alias "pd." Using the aliases will allow you to access matplotlib and pandas without having to type in the full name each time.

In [ ]:
```
import matplotlib.pyplot as plt
import pandas as pd
```

Creating datasets using lists

Let's say you were interested in visualizing the average scores for a final exam in a chemistry class from 2010-2015. In this case, we have two types of data: the year and exam score.

To create a list, use square brackets ([ ]) and seperate each value with a comma. Next, assign each list to an object, which will be used to "store" our data and be used in the visualization.

In the example below, there are two objects: years and test_scores. Each object was assigned a list using the **=** sign.

The names of the objects should be meaningful so that someone looking at your code for the first time can quickly interpret what the objects contain. Avoid names that are ambiguous (x, numbers, etc.).

Important: When using more than one array for visualization, make sure the number of values in each array is the same. For example, if you have five values in the first array, make sure you have five values in the second array. Otherwise, you will get an error.

In [3]:
```
years = [2010, 2011, 2012, 2013, 2014, 2015]
test_scores = [95, 92, 97, 75, 88, 90]
```

Creating a line graph with Matplotlib

The first kind of visualization we will create is a line graph. Using our two objects, we can use **plt.plot** to create a graph of the average test scores over time. When creating a line graph, the first object will be the x-axis and the second object will serve as the y-axis.

plt.plot(x,y)

To see the line graph you created, use **plt.show()**.

In [4]:
```
plt.plot(years, test_scores)
plt.show()
```

Customizing your graph

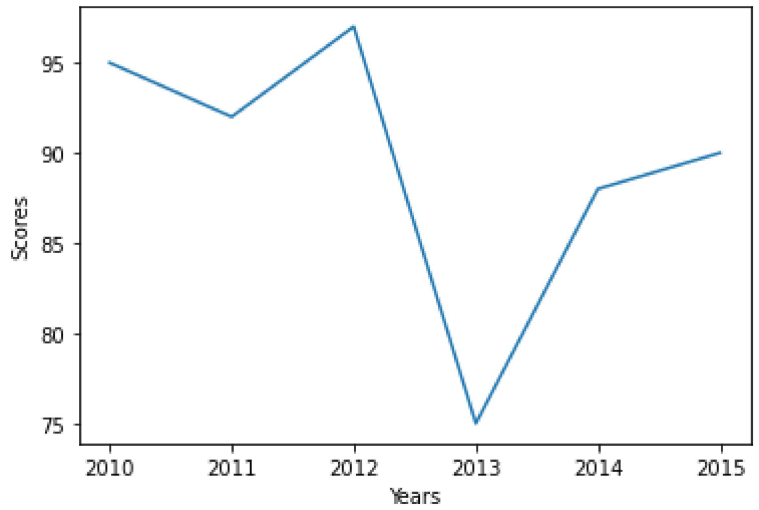
Congrats, you just made your first visualization in Python! However, it does not tell us much. What do the numbers on each axis represent? What if you wanted to add a title? What if you hate the default line color and want to change it to something else? Finally, what if you wanted to change the scaling of the graph?

Axis labels

To begin, let's add some labels to our axis. To do this, we use **plt.xlabel()** and **plt.ylabel()**. In the paratheses, include the desired labels as a string ('').

In this case, we want to label our x-axis as 'Years' and our y-axis as 'Scores'. Therefore, we would input **plt.xlabel('Years')** and **plt.ylabel('Scores')** in addition to our existing code in order to display the labels.

In [5]:
```
plt.plot(years, test_scores)
plt.xlabel('Years')
plt.ylabel('Scores')
plt.show()
```

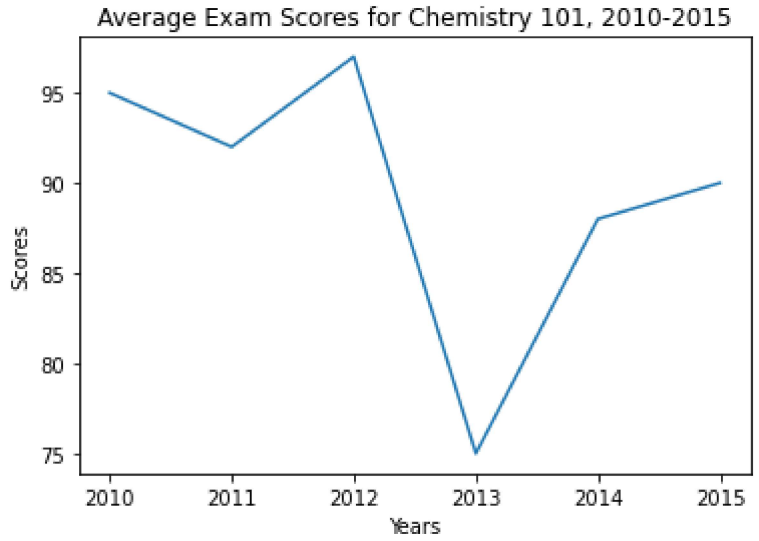


Title

Now, let's add a title to our graph. To do this, input **plt.title()** after adding the x and y-axis labels. In the parentheses, input the desired title as a string.

For this example, input the title as 'Average Exam Scores for Chemistry 101, 2010-2015.'

In [6]:
```
plt.plot(years, test_scores)
plt.xlabel('Years')
plt.ylabel('Scores')
plt.title('Average Exam Scores for Chemistry 101, 2010-2015')
plt.show()
```



Line color

Next, let's change the color of the line. To do this, we're going to add an argument to plt.plot(). This tells Python that we want to change an aspect of our graph (in this case, the color of the line). After adding the objects you wish to be visualized, add the argument **color='()'** with the desired color as a string. This color can be the name of the color, the hexidecimal value, or RGB groupings.

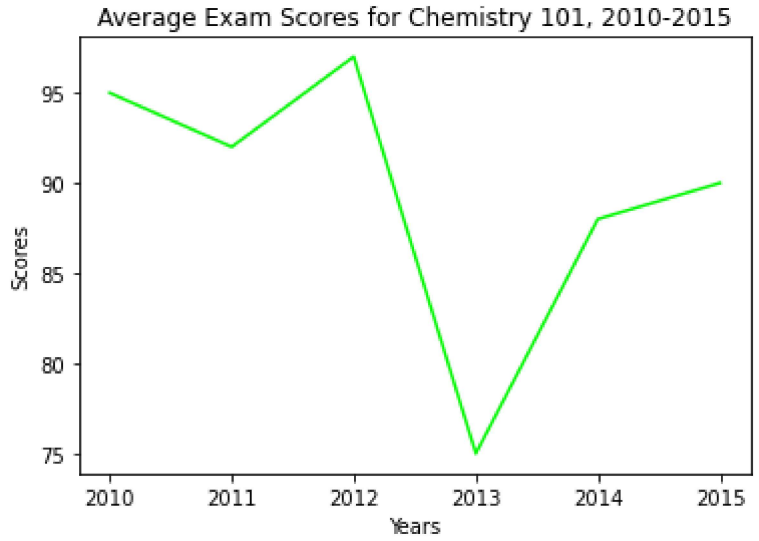
Important: If you're inputing RGB groupings, use parentheses instead of the singular quotation marks. Python processes RGB groupings as integers, not strings.

plt.plot(x,y, color='purple')

plt.plot(x,y, color='#008000')

plt.plot(x,y, color= (0,1,0))

In [8]:
```
plt.plot(years, test_scores, color=(0,1,0))
plt.xlabel('Years')
plt.ylabel('Scores')
plt.title('Average Exam Scores for Chemistry 101, 2010-2015')
plt.show()
```



Scaling

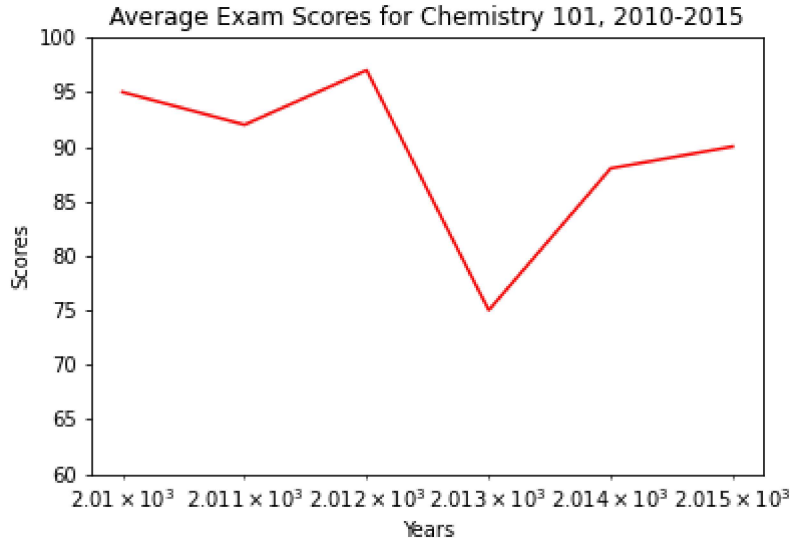
Finally, let's adjust the scaling. There are two methods for adjusting scaling depending on the needs of your graph.

The first method is putting limits on the values for the axis. To do this, we use **plt.xlim([num1, num2])** and **plt.ylim([num1, num2])**. This tells Python to set the scale to the designated limits.

The second method is used for fitting a graph to a particular scale, most commonly the logorithmic scale. To do this, use **plt.xscale('')** and **plt.yscale('')** to scale the graph according to a specific scale. For example, if we wanted the graph to be scaled to the logorithmic scale, use **plt.xscale('log')** and **plt.yscale('log')**.

Below, we will use **plt.xlim()** and **plt.ylim()** to scale our axis.

In [11]:
```
plt.plot(years, test_scores, color='#FF0000')
plt.xscale('log')
plt.ylim([60,100])
plt.xlabel('Years')
plt.ylabel('Scores')
plt.title('Average Exam Scores for Chemistry 101, 2010-2015')
plt.show()
```



Loading and manipulating a .csv file with pandas

Likely, you will be using Python to visualize larger datasets. If you keep your data contained within a .csv file, you can use the pandas library to load, read, and manipulate the dataset for visualization.

Let's say that we were interested in conducting an exploratory analysis of a large datasets of films (insert authors/citation). This particular dataset contains information on 1600 films, including the year, length, lead actors, director, and popularity.

To load the CSV file and store it for visualization, we will create an object ('df'). This stands for dataframe and is a common abbreviation when working with pandas. Then, input **df = pd.read_csv("film.csv")**. This will read and store the csv within 'df'.

Important: Sometimes, UTF-encoding errors, incorrect delimiters, and issues with the file directory can cause pandas not to load the data and give you an error. There are a few workarounds for this depending on the specific issue, but usually you need to add additional arguments to pd.read_csv(). The workaround below was for fixing an incorrect delimiter and a UTF-8 encoding issue present with this dataset.

To view the first few rows of the data set, input **df.head()**.

In [ ]:
```
df = pd.read_csv("film.csv", delimiter=';', encoding = "ISO-8859-1")
df.head()
```

Grouping data and creating a bar graph

For our exploratory film analysis, let's determine how many films of each subject (genre) are in this dataset. The best visualization for this would be a bar graph. In order to accomplish this, we need to tell Python to group the films according to subject.

To do this, input **df.groupby()** and input the desired grouping as a string. In this case, our string will be "Subject."

To get a count and visualize it within a bar graph, add **.size()** and **.plot(kind='bar')** to the end of df.groupby('subject').

In [ ]:
```
df.groupby('Subject').size().plot(kind='barh')
plt.xlabel('Subject')
plt.ylabel('Number of Movies')
plt.title('Number of Movies by Subject')
plt.show()
```

Creating a scatter plot

Finally, let's create a scatter plot to see if there is a correlation between the length of the film and its popularity.

First, let's create a dataframe with values for length and popularity for the first ten films in the dataset. To accomplish this we can use an object to store a "list of lists". Think of this as rows within a spreadsheet, with each set of two values representing a row with two columns.

In [ ]:
```
data = ([111, 68], [113,68], [104,70], [122, 6], [94,14], [140, 68], [101, 14], [99, 0], [104, 32], [149, 81])
```

Next, use **pd.DataFrame()** to create the dataframe. To differentiate the two columns, we will add the columns argument with the names of the columns as strings within square brackets.

To view the newly created dataframe, input **df**.

In [ ]:
```
df= pd.DataFrame(data, columns=['Length', 'Popularity'])
df
```

To create a scatter plot with pandas, input **df.plot.scatter(x='', y='')**. For this exercise, the x-axis will be the length of the movie and the y-axis will be the popularity score, so we will input the names of the columns as strings within the x and y parameters.

In [ ]:
```
df.plot.scatter(x='Length', y='Popularity')
plt.title("Length of Movie vs. Popularity")
plt.show()
```

Additional Resources

Congrats, you successfully completed the Introduction to Python for Data Visualization workshop!

Below are additional resources related to the libraries discussed in the workshop.

Matplotlib Documentation and User Guide

https://matplotlib.org/stable/index.html (https://matplotlib.org/stable/index.html)

Pandas User Guide

https://pandas.pydata.org/ (https://pandas.pydata.org/)