Data Science Nigeria: Introductory Machine Learning Training

Data Visualization

MATPLOTLIB

This is a Python library for visualization, others include

- Seaborn
- ggplot
- Bokeh
- Plotly
- Pygal
- Altair

Terms

Some common terms in Matpliotlib implementation are:

Figure: It is a whole figure which may contain one or more than one axes (plots). You can think of a Figure as a canvas that contains plots.

Axes: It is what we generally think of as a plot. A Figure can contain many axes. It contains two or three (in the case of 3D) Axis objects. Each axis has a title, an x-label, and a y-label.

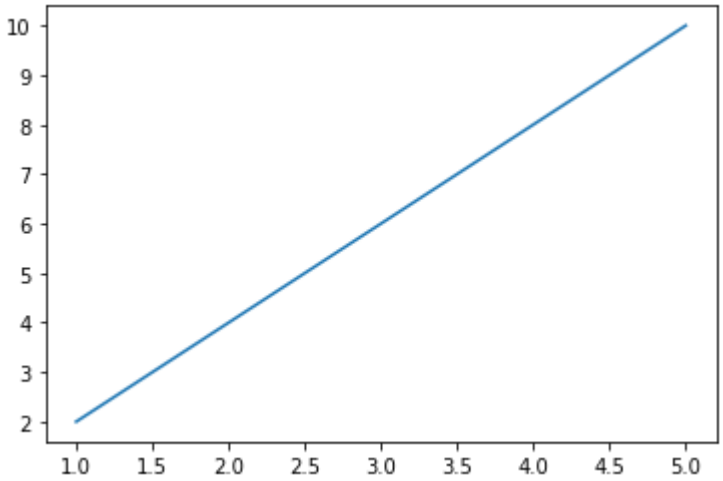
Axis: They are the number line like objects and take care of generating the graph limits.

In [6]:

```
import matplotlib.pyplot as plt
import numpy as np
```

In [7]:

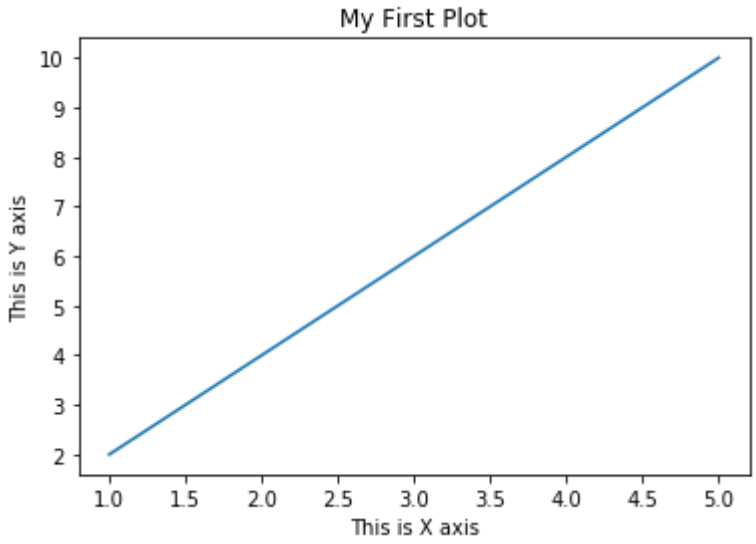
```
plt.plot([1,2,3,4,5], [2,4,6,8,10])

plt.show()
```



Let us add the title, and name x-axis and y-axis using methods title(), xlabel() and ylabel() respectively.

In [3]:

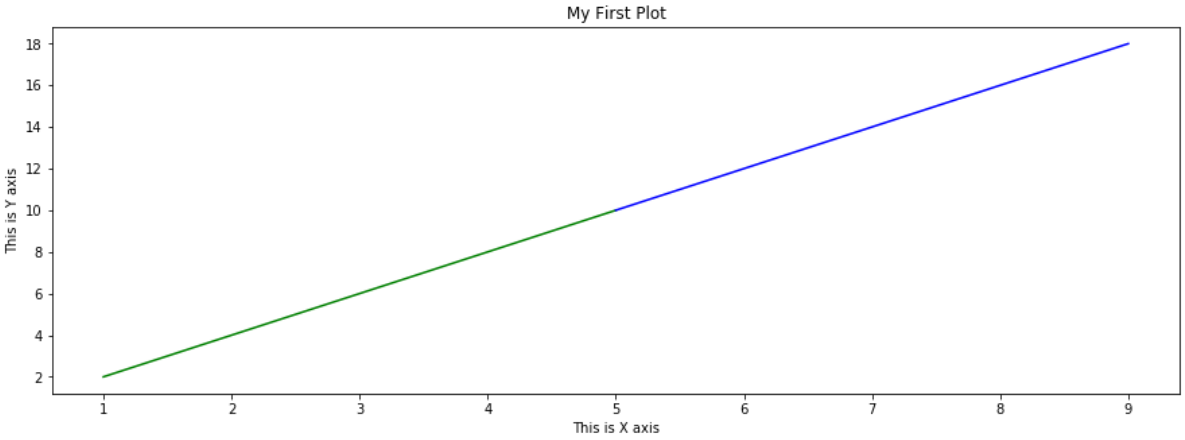
```
plt.plot([1,2,3,4,5], [2,4,6,8,10])
plt.title('My First Plot')
plt.xlabel("This is X axis")
plt.ylabel("This is Y axis")
plt.show()
```



We can also specify the size of the figure using method figure() and passing the values as a tuple representing the rows and columns

In [4]:

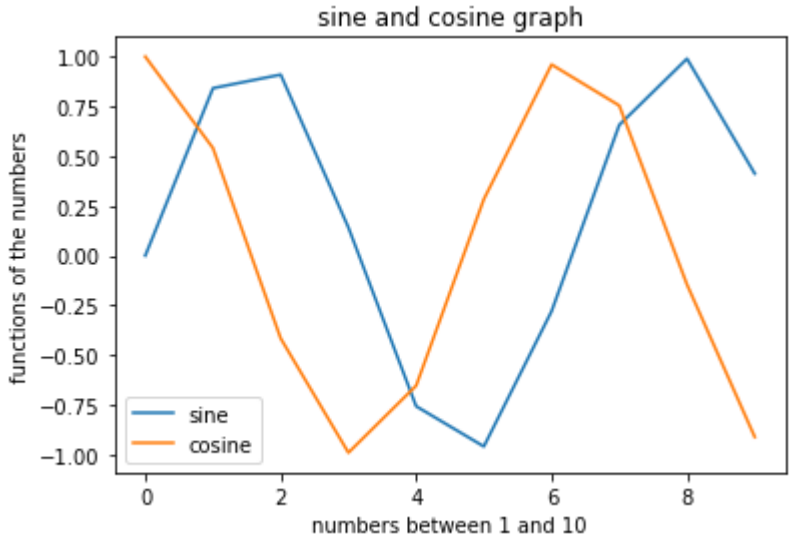
```
plt.figure(figsize= (15,5))
plt.plot([1,2,3,4,5], [2,4,6,8,10],'g', [5,6,7,8,9], [10,12,14,16,18], 'b')
plt.title('My First Plot')
plt.xlabel("This is X axis")
plt.ylabel("This is Y axis")
plt.show()
```



In [8]:

```
x = np.arange(0, 10)
print (x)

y = np.sin (x)
print (y)

z = np.cos (x)
print (z)

plt.plot(x,y, x,z)
plt.xlabel('numbers between 1 and 10')
plt.ylabel('functions of the numbers')
plt.title('sine and cosine graph')
plt.legend(['sine','cosine'])
plt.show()
```

```
[0 1 2 3 4 5 6 7 8 9]
[ 0.          0.84147098  0.90929743  0.14112001 -0.7568025  -0.95892427
 -0.2794155   0.6569866   0.98935825  0.41211849]
[ 1.          0.54030231 -0.41614684 -0.9899925  -0.65364362  0.28366219
  0.96017029  0.75390225 -0.14550003 -0.91113026]
```

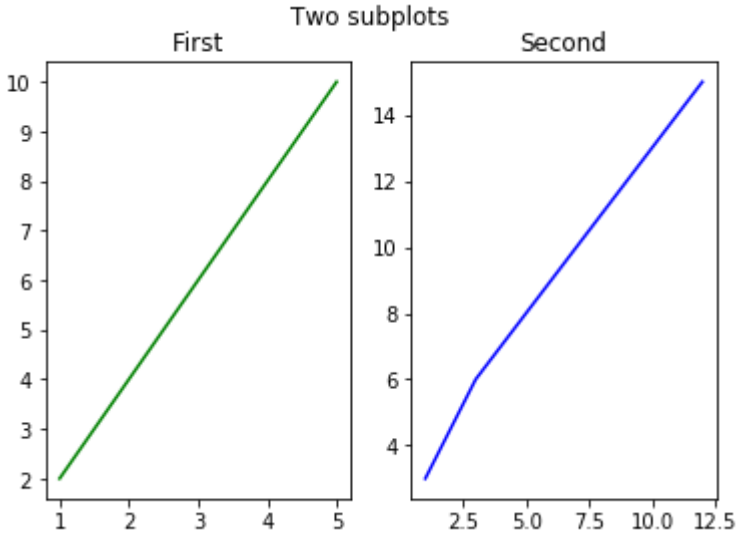


Subplot

With the use of subplot method, we can create many subplots in a figure. The subplot() method takes three arguments viz are nrows, ncols, and index.

In [9]:

```
plt.subplot(1,2,1) # 1 row, 2 columns, index 1
plt.plot([1,2,3,4,5], [2,4,6,8,10], 'g')
plt.title('First')

plt.subplot(1,2,2) # 1 row, 2 columns, index 2
plt.plot([1,3,6,9,12], [3,6,9,12,15], 'b')
plt.title('Second')

plt.subplots_adjust(hspace=0.4) # for adjusting spaces inbetween subplots

plt.suptitle('Two subplots')
plt.show()
```



Other Platforms for data visualization include

- powerBI
- Excel
- tableau

e.t.c

Class Work

Load a datset and visualise the relationship between a feature and the target variable

Data Preprocessing

In [10]:

```
import pandas as pd
import numpy as np
```

Dropping Columns in a DataFrame

Often, you'll find that not all the categories of data in a dataset are useful to you. Pandas provide a handy way of removing unwanted columns or rows from a DataFrame with the drop() function. Let's look at a simple example where we drop a number of columns from a DataFrame.

In [11]:

```
df = pd.read_csv('datasets/BL-Flickr-Images-Book.csv')
df.head()
```

Out[11]:

	Identifier	Edition Statement	Place of Publication	Date of Publication	Publisher	Title	Author	Contributors
0	206	NaN	London	1879 [1878]	S. Tinsley & Co.	Walter Forbes. [A novel.] By A. A	A. A.	FORBES, Walter.
1	216	NaN	London; Virtue & Yorston	1868	Virtue & Co.	All for Greed. [A novel. The dedication signed...	A., A. A.	BLAZE DE BURY, Marie Pauline Rose - Baroness
2	218	NaN	London	1869	Bradbury, Evans & Co.	Love the Avenger. By the author of "All for Gr...	A., A. A.	BLAZE DE BURY, Marie Pauline Rose - Baroness
3	472	NaN	London	1851	James Darling	Welsh Sketches, chiefly ecclesiastical, to the...	A., E. S.	Appleyard, Ernest Silvanus.
4	480	A new edition, revised, etc.	London	1857	Wertheim & Macintosh	[The World in which I live, and my place in it...	A., E. S.	BROOME, John Henry.

In [12]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8287 entries, 0 to 8286
Data columns (total 15 columns):
Identifier                8287 non-null int64
Edition Statement         773 non-null object
Place of Publication      8287 non-null object
Date of Publication       8106 non-null object
Publisher                 4092 non-null object
Title                     8287 non-null object
Author                    6509 non-null object
Contributors              8287 non-null object
Corporate Author          0 non-null float64
Corporate Contributors    0 non-null float64
Former owner              1 non-null object
Engraver                  0 non-null float64
Issuance type             8287 non-null object
Flickr URL                8287 non-null object
Shelfmarks                8287 non-null object
dtypes: float64(3), int64(1), object(11)
memory usage: 971.3+ KB
```

the information in the following columns are reductant or not needed, so we would be dropping them from the data set using the **drop()**

In [ ]:

```
to_drop = ['Edition Statement',
           'Corporate Author',
          'Corporate Contributors',
           'Former owner',
           'Engraver',
           'Contributors',
           'Issuance type',
           'Shelfmarks']

# inplace agrument in the drop method makes the changes permanent in the dataset
df.drop(to_drop, inplace=True, axis=1)
```

In [14]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8287 entries, 0 to 8286
Data columns (total 7 columns):
Identifier              8287 non-null int64
Place of Publication    8287 non-null object
Date of Publication     8106 non-null object
Publisher               4092 non-null object
Title                   8287 non-null object
Author                  6509 non-null object
Flickr URL              8287 non-null object
dtypes: int64(1), object(6)
memory usage: 453.3+ KB
```

Missing Values

Every dataset has an amount of noise (unwanted or unneeded datapoint) in them, and it is our job as a data scientist to remove them from the dataset.

In [17]:

```
# firstly, we must know the number of missing values in each column in the dataset
df.isnull().sum()
```

Out[17]:

```
Identifier                0
Place of Publication      0
Date of Publication     181
Publisher              4195
Title                     0
Author                 1778
Flickr URL                0
dtype: int64
```

In [18]:

```
# percentage of missing value, this will help us to understan if we are to drop a partic
# or fill in the missing value
df.isnull().sum() * 100 / len(df)
```

Out[18]:

```
Identifier              0.000000
Place of Publication    0.000000
Date of Publication     2.184144
Publisher              50.621455
Title                   0.000000
Author                 21.455291
Flickr URL              0.000000
dtype: float64
```

Dealing with missing values

Impute Missing Values

Imputing refers to using a model to replace missing values.

There are many options we could consider when replacing a missing value, for example:

A constant value that has meaning within the domain, such as 0, distinct from all other values. A value from another randomly selected record. A mean, median or mode value for the column. A value estimated by another predictive model.

or you can delete the row or column base on the nature of the dataset

pandas.fillna() can be used to fill in value into a missing column

In [ ]:

```
```

Rename columns

In [19]:

```
olympics_df = pd.read_csv('Datasets/olympics.csv')
olympics_df.head()
```

Out[19]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	
0	NaN	? Summer	01 !	02 !	03 !	Total	? Winter	01 !	02 !	03 !	Total	? Games	01 !	02 !	03 !	Comb
1	Afghanistan (AFG)	13	0	0	2	2	0	0	0	0	0	13	0	0	2	
2	Algeria (ALG)	12	5	2	8	15	3	0	0	0	0	15	5	2	8	
3	Argentina (ARG)	23	18	24	28	70	18	0	0	0	0	41	18	24	28	
4	Armenia (ARM)	5	1	2	9	12	6	0	0	0	0	11	1	2	9	

In [20]:

```
# the argument header=1 is to start from index 0 as the header and ingnor anything above
olympics_df = pd.read_csv('Datasets/olympics.csv', header=1)
olympics_df.head()
```

Out[20]:

	Unnamed: 0	? Summer	01 !	02 !	03 !	Total	? Winter	01 !.1	02 !.1	03 !.1	Total.1	? Games	01 !.2	02 !.2	03 !.2	Co
0	Afghanistan (AFG)	13	0	0	2	2	0	0	0	0	0	13	0	0	2	
1	Algeria (ALG)	12	5	2	8	15	3	0	0	0	0	15	5	2	8	
2	Argentina (ARG)	23	18	24	28	70	18	0	0	0	0	41	18	24	28	
3	Armenia (ARM)	5	1	2	9	12	6	0	0	0	0	11	1	2	9	
4	Australasia (ANZ) [ANZ]	2	3	4	5	12	0	0	0	0	0	2	3	4	5	

In [21]:

```
new_names =  {'Unnamed: 0': 'Country',
              '? Summer': 'Summer Olympics',
               '01 !': 'Gold',
               '02 !': 'Silver',
               '03 !': 'Bronze',
               '? Winter': 'Winter Olympics',
               '01 !.1': 'Gold.1',
               '02 !.1': 'Silver.1',
               '03 !.1': 'Bronze.1',
               '? Games': '# Games',
               '01 !.2': 'Gold.2',
               '02 !.2': 'Silver.2',
               '03 !.2': 'Bronze.2'}
```

In [22]:

```
# rename the header
olympics_df.rename(columns=new_names, inplace=True)
```

In [23]:

```
olympics_df.head()
```

Out[23]:

	Country	Summer Olympics	Gold	Silver	Bronze	Total	Winter Olympics	Gold.1	Silver.1	Bronze.1	Total.1
0	Afghanistan (AFG)	13	0	0	2	2	0	0	0	0	0
1	Algeria (ALG)	12	5	2	8	15	3	0	0	0	0
2	Argentina (ARG)	23	18	24	28	70	18	0	0	0	0
3	Armenia (ARM)	5	1	2	9	12	6	0	0	0	0
4	Australasia (ANZ) [ANZ]	2	3	4	5	12	0	0	0	0	0

Class Work

Clean the dataset "titanic_train.csv" in the file folder

- load in the dataset using pandas framework
- check for missing values
- replace the missing values with either mean or mode
- and some visualization would be nice if added

Assignment

- 1
- 2
- 3

In [ ]:

```

```