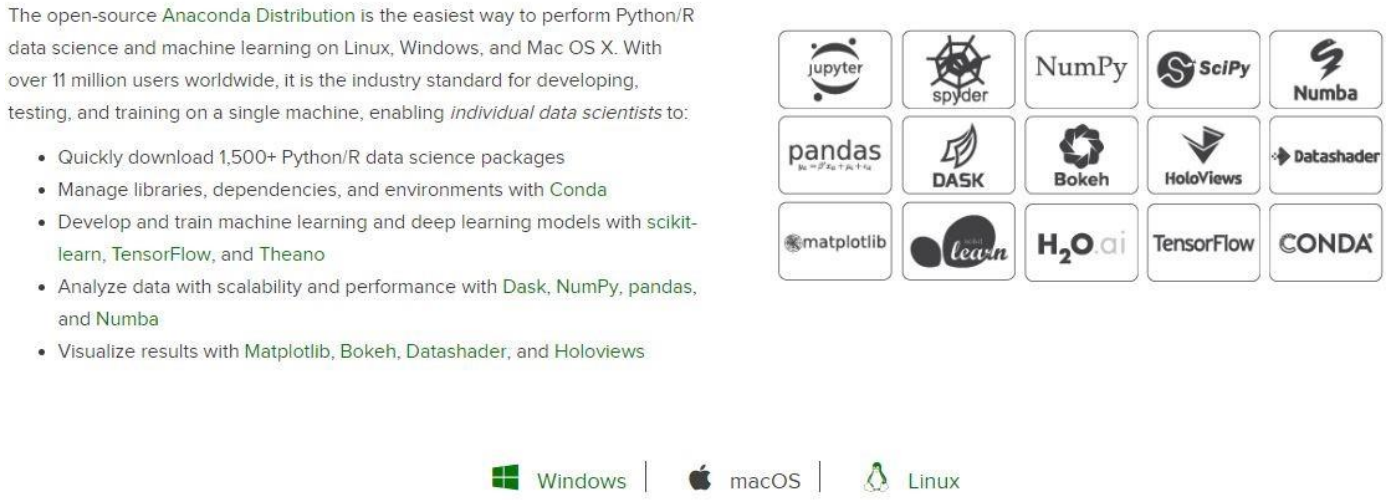
Day 2

Introduction to Pandas and Numpy

- Installation and Environment Setup
- Libraries
  - Numpy, Pandas, Matplotlib basics
  - Data Preprocessing
    - Handling Missing Data
    - Encoding Categorical Variable
- Introduction to Regression
- Homework

Installation and Environment Setup

For this training, we will be using one of the most popular framework used for data science and machine learning known as **Anaconda**. You can download the correct version of anaconda here (https://www.anaconda.com/distribution/)as it relates to your op erating system – Windows, Mac or Linux.



- Why Python?

Python is a general-purpose, beginner friendly language, which can be used to build virtually anything. It can be used for web development, desktop app development, gaming, data analysis, Artificial intellingence, scientific computing etc. Increasing community of python users is another reason for choosing PYTHON

Remember,

> The larger a community, the more likely you'd get help and the more people will be building useful tools to ease the process of development.

Required Libraries

- Numpy
- Pandas
- Matplotlib
- Scikit Learn

NUMPY

This is a fundamental Package for scientific computing for manipulation of multi-dimensional arrays and matrices. It is particularly useful for linear algebra, Fourier transform, random number simulation etc

Matrices are rectangular array of numbers, symbols and expressions arranged in rows and columns. The numbers, symbols or expressions in the matrix are called its entries or its elements. The horizontal and vertical lines of entries in a matrix are called rows and columns, respectively. Its operations inclue addition, subtraction, multiplication

The first step is to import numpy library into the active notebook

In [1]:

```
import numpy
```

To shorten the length of any library, a better alternative is to instantiate the library with a shorter name, as in, In [2]:

```
import numpy as np
```

With this, each time numpy is required on this active notebook, **np** will be used instead

In [3]:

```
#creating a 1 dimensional array

x = np.array([1, 2, 3, 4, 5])
y = np.array([9, 10])
print(x)
print('The shape of X is' , x.shape)

print(y)
print('The shape of Y is' , y.shape)
```

```
[1 2 3 4 5]
The shape of X is (5,)
[ 9 10]
The shape of Y is (2,)
```

In [4]:

```
# Creating a 2D arrays
z = np.array([[1, 2], [3, 4]])

print(z)
print('The shape of Z is' , z.shape)
```

```
[[1 2]
 [3 4]]
The shape of Z is (2, 2)
```

In [5]:

```
# creating a multidimensional array

w = np.array([[[1,2,3],[4,5,6],[7,8,9]],[[10,11,12],[13,14,15],[16,17,18]],[[19,20,21],[
print(w, '\n')
print('The shape of W is' , w.shape)
```

```
[[[ 1  2  3]
  [ 4  5  6]
  [ 7  8  9]]

 [[10 11 12]
  [13 14 15]
  [16 17 18]]

 [[19 20 21]
  [22 23 24]
  [26 27 28]]]

The shape of W is (3, 3, 3)
```

Numpy Functions

Numpy has built-in functions for creating arrays. These includes:

arrange:

reshape

zeros ones

full

linspace random

> The dimensions (no of rows and column) are passed as parameters to the function.

In [6]:

```
#arrange is Used to create arrays with values in a specified range.

A10 = np.arange(10)
A10
print(A10.shape)
```

(10,)

In [7]:

```
#To change a scalar matrix to vextor

B10 = A10.reshape(-1,1)

print ( A10, '\n', B10)
print ("The shape of 1D array X = " , B10.shape)
```

```
[0 1 2 3 4 5 6 7 8 9]
 [[0]
 [1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
The shape of 1D array X =  (10, 1)
```

In [8]:

```
A10 = B10.reshape(2,5)

print ( A10)
print ("The shape of 1D array X = " , A10.shape)
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
The shape of 1D array X =  (2, 5)
```

> Note: The new dimension must be compatible with the old one

In [9]:

```
#zeros is used to create an array filled with zeros.

np_Zeros = np.zeros((2,3))

np_Zeros
```

Out[9]:

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [10]:

```
#ones is used to create an array filled with ones
np_Ones = np.ones((2,3))

np_Ones
```

Out[10]:

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

In [11]:

```
 #function creates a n * n array filled with a specified given value.
np_full = np.full((2,3), 4)

np_full
```

Out[11]:

```
array([[4, 4, 4],
       [4, 4, 4]])
```

In [12]:

```
#The eye function lets you create a n * n diagonal matrix
np_eye = np.eye(3,6)

np_eye
```

Out[12]:

```
array([[1., 0., 0., 0., 0., 0.],       [0., 1., 0., 0., 0., 0.],
    [0., 0., 1., 0., 0., 0.]])
```

In [13]:

```
#linspace returns evenly spaced numbers over a specified interval
np_linspace = np.linspace(0, 10, num = 6)

np_linspace
```

Out[13]:

```
array([ 0.,  2.,  4.,  6.,  8., 10.])
```

In [14]:

```
#This creates an array filled with random values between 0 and 1
np_rand = np.random.random_sample((2,3))

np_rand1 = np.random.rand(2,3)

X = np.random.randint(10, size=(5,3))

print(np_rand)

print(np_rand1)
X
```

```
[[0.28608689 0.58308977 0.96682239]
 [0.29172351 0.82030534 0.41377409]] [[0.12963402
0.59496657 0.17656489]  [0.76165033 0.14181669
0.17240588]]
```

Out[14]:

```
array([[3, 9, 7],       [3, 2, 3],
    [7, 4, 6],
    [1, 1, 8],
    [4, 9, 9]])
```

In

Accessing elements of Numpy array

> To access an element in a two-dimensional array, you need to specify an index for both the row and the column.

[15]:

```
#Row 1, column 0 gives a scalar
z[1,0]
```

Out[15]:

3

In [16]:

```
#or

p = z[1][0]

p
```

Out[16]:

3

In [17]:

```
p = (z[0:1, 0])
p
```

Out[17]: array([1])

Numpy Attributes

Array attributes reflect information that is intrinsic to the array itself. Generally, accessing an array through its attributes allows you to get and sometimes set intrinsic properties of the array without creating a new array. The exposed attributes are the core parts of an array and only some of them can be reset meaningfully without creating a new array

Some commonly used attributes are:

- Shape: indicates the size of an array
- Size: returns the total number of elements in the NumPy array
- Dtype: returns the type of elements in the array, i.e., int64, character

In [18]:

```
print ("The Dtype of elements in array X= "  , x.dtype)

print ("The shape of ND array W= " , w.dtype)
```

In

The Dtype of elements in array X= int32
The shape of ND array W= int32

[19]:

```
print ("The shape of 1D array X = ", x.shape) print ("The shape of 2D
array Z = ", z.shape) print ("The shape of ND array W = ", w.shape)
print ("The shape of arange A10 = ", A10.shape)
```

The shape of 1D array X = (5,)
The shape of 2D array Z = (2, 2)
The shape of ND array W = (3, 3, 3)
The shape of arange A10 = (2, 5)

In [20]:

```
print ("The shape of ND array W = ", w.size) print ("The shape of
arange A10 = ", A10.size)
```

The shape of ND array W = 27
The shape of arange A10 = 10

Numpy array math operations

In [21]:

```
x = np.array([[1,2,3],[4,5,6]])
y = np.array([[2,2,2],[3,3,3]])
z = np.array([1,2,3])
```

In [22]:

```
#Transpose a matrix

x.T
```

Out[22]:

```
array([[1, 4],
       [2, 5],
       [3, 6]])
```

In [23]:

```
#Elementwise addittion

print (x+y)
print (np.add(x,y))
```

```
[[3 4 5]
 [7 8 9]] [[3 4 5]
 [7 8 9]]
```

In [24]:

```
#Elementwise Subtraction

print (x-y)
print (np.subtract(x,y))
```

```
[[-1  0  1]
 [1 2 3]] [[-1 0 1]
 [1 2 3]]
```

In [25]:

```
#Elementwise Multiplication

print (x*z)
print (np.multiply(x,z))
```

```
[[ 1  4  9]
 [4 10 18]] [[ 1  4  9]
 [4 10 18]]
```

In [26]:

```
##Elementwise Division

print (x/y)
print (np.divide(x,y))
```

```
[[0.5        1.         1.5       ]
 [1.33333333 1.66666667 2.        ]] [[0.5      1.       1.5      ]
 [1.33333333 1.66666667 2.       ]]
```

In [27]:

```
# Inner product of vectors
print(np.dot(x, z), "\n")
```

```
[14 32]
```

PYTHON PANDAS

This is a multidimensional data structures and analysis tool for manipulating numerical

> Note: Rows represent **observations** while columns represent **input features**

Pandas Data Type

Recognised pandas data type includes:

In

- **object:** To represent text **int64:**
- Integer values

- **float64:** Floating point numbers **Category:** List of
- text values **bool:** True or false values **datetime64:**
- Date and time values **timedelta:** Difference
- between two datetimes
-

In [29]:

```
import pandas as pd
```

Ways to create pandas dataframe

In [30]:

```
# initialize list of lists
data = [['Ayo', 10], ['Imran', 15], ['Chucks', 14]]

# Create the pandas DataFrame from the list and adding column headers
df = pd.DataFrame(data, columns = ['Name', 'Age'])

# print dataframe.
df
```

Out[30]:

	Name	Age
0	Ayo	10
1	Imran	15
2	Chucks	14

In [31]:

```
# Create the pandas DataFrame from the dictionary of narray list
#Example 1:
# initialize list of lists
data = {'Name': ['Ayo', 'Imran','Chucks'] ,'Age':[10, 15, 14]}

# Create the pandas DataFrame from the list and adding column headers
df = pd.DataFrame(data)

# print dataframe.
df
```

Out[31]:

	Age	Name

In [34]:

0 10 Ayo

1 15 Imran

2 14 Chucks

[32]:

```
#Example 2:

#Population and area (km/square) of some states in Nigeria and their capital

dict_data = {"State": ["Abia", "Adamawa", "Lagos", "Osun", "Rivers"],
       "Capital": ["Umuahia", "Yola", "Ikeja", "Osogbo", "Portharcourt"],
       "area": [6320, 36917, 3345, 9251, 11077],
       "population" : [2845380, 3178950, 9113605, 3416959, 5198605] }

df = pd.DataFrame(dict_data)

df
```

Out[32]:

	Capital	State	area	population
0	Umuahia	Abia	6320	2845380
1	Yola	Adamawa	36917	3178950
2	Ikeja	Lagos	3345	9113605
3	Osogbo	Osun	9251	3416959
4	Portharcourt	Rivers	11077	5198605

In [34]:

```
df.dtypes
```

Out[34]:

Capital object State object
area int64 population
int64 dtype: object

In [35]:

ZIP

```
# pandas Datadaframe from lists using zip.

# List1
Name = ['Ayo', 'Imran','Chucks', 'judith']

# List2
Age = [25, 30, 26, 22]

# get the list of tuples from two list and merge them by using zip().
list_of_tuples = list(zip(Name, Age))

# Converting lists of tuples into pandas Dataframe.
df = pd.DataFrame(list_of_tuples , columns = ['Name', 'Age'])

# Print data.
df
```

Out[35]:

	Name	Age
0	Ayo	25
1	Imran	30
2	Chucks	26
3	judith	22

SERIES

A Series represents a single column in memory, which is either independent or belongs to a Pandas DataFrame.

In [36]:

```
# Pandas Dataframe from Dicts of series.

import pandas as pd

# Intialise data to Dicts of series.
series_data = {"State": pd.Series(["Abia", "Adamawa", "Lagos", "Osun", "Rivers"]),
       "Capital": pd.Series(["Umuahia", "Yola", "Ikeja", "Osogbo", "Portharcourt"]),
       "area": pd.Series([6320, 36917, 3345, 9251, 11077]),
       "population": pd.Series([2845380, 3178950, 9113605, 3416959, 5198605]) }

# creates Dataframe.
df = pd.DataFrame(series_data)

# print the data.
df
```

Out[36]:

	Capital	State	area	population
0	Umuahia	Abia	6320	2845380
1	Yola	Adamawa	36917	3178950
2	Ikeja	Lagos	3345	9113605
3	Osogbo	Osun	9251	3416959
4	Portharcourt	Rivers	11077	5198605

External source -

CSV Another way to create a DataFrame is by importing a csv file using pd.read_csv

```
csv_df = pd.read_csv('Data/2006.csv')

csv_df
```

Out[37]:

	STATES	AREA (km2)	Population
0	Abia State	6320	2845380
1	Adamawa State	36917	3178950
2	Akwa Ibom State	7081	3178950
3	Anambra State	4844	4177828
4	Bauchi State	45837	4653066
5	Bayelsa State	10773	1704515
6	Benue State	34059	4253641
7	Borno State	70898	4171104
8	Cross River	20156	2892988

In [37]:

9	Delta State	17698	4112445
10	Ebonyi State	5670	2176947
11	Edo State	17802	3233366
12	Ekiti State	6353	2398957
13	Enugu State	7161	3267837
14	FCT	7315	1405201
15	Gombe State	18768	2365040
16	Imo State	5530	3927563
17	Jigawa State	23154	4361002
18	Kaduna State	46053	6113503
19	Kano State	20131	9401288
20	Katsina State	24192	5801584
21	Kebbi State	36800	3256541
22	Kogi State	29833	3314043
23	Kwara State	36825	2365353
24	Lagos State	3345	9113605
25	Nasarawa State	27117	1869377
26	Niger State	76363	3954772
27	Ogun State	16762	3751140
28	Ondo State	15500	3460877
29	Osun State	9251	3416959
30	Oyo State	28454	5580894
31	Plateau State	30913	3206531
	STATES	**AREA (km2)**	**Population**
32	Rivers State	11077	5198605
33	Sokoto State	25973	3702676
34	Taraba State	54473	2294800
35	Yobe State	45502	2321339
36	Zamfara State	39762	3278873

EXCEL- XLSX

In [38]:

```
Excel_df = pd.read_excel('Data/2006.xlsx' )

Excel_df
```

Out[38]:

	STATES	AREA (km2)	Population
0	Abia State	6320	2845380
1	Adamawa State	36917	3178950
2	Akwa Ibom State	7081	3178950
3	Anambra State	4844	4177828
4	Bauchi State	45837	4653066
5	Bayelsa State	10773	1704515
6	Benue State	34059	4253641
7	Borno State	70898	4171104
8	Cross River	20156	2892988
9	Delta State	17698	4112445
10	Ebonyi State	5670	2176947
11	Edo State	17802	3233366
12	Ekiti State	6353	2398957
13	Enugu State	7161	3267837
14	FCT	7315	1405201
15	Gombe State	18768	2365040
16	Imo State	5530	3927563
17	Jigawa State	23154	4361002
18	Kaduna State	46053	6113503
19	Kano State	20131	9401288
20	Katsina State	24192	5801584
21	Kebbi State	36800	3256541
22	Kogi State	29833	3314043
23	Kwara State	36825	2365353
24	Lagos State	3345	9113605
25	Nasarawa State	27117	1869377
26	Niger State	76363	3954772
27	Ogun State	16762	3751140
28	Ondo State	15500	3460877
29	Osun State	9251	3416959
30	Oyo State	28454	5580894
31	Plateau State	30913	3206531

In

	STATES	AREA (km2)	Population
32	Rivers State	11077	5198605
33	Sokoto State	25973	3702676
34	Taraba State	54473	2294800
35	Yobe State	45502	2321339
36	Zamfara State	39762	3278873

In [39]:

```
#By default, if no length is specified, it returns the first 5 rows
print(csv_df.head(), '\n')

#This returns the first 5 rows in Population Column
print(csv_df['Population' ].head())
```

```
      STATES   AREA (km2)    Population
0   Abia State      6320    2845380
1   Adamawa State     36917    3178950
2   Akwa Ibom State      7081    3178950
3   Anambra State      4844    4177828
4   Bauchi State      45837    4653066
0   2845380
1   3178950
2   3178950
3   4177828
4   4653066
Name: Population, dtype: int64
```

In [40]:

```
#By default, if no length is specified, it returns the last 5 rows
print(csv_df.tail(), '\n')

#This returns the last 5 rows in Population Column
print(csv_df['Population' ].tail())
```

```
      STATES   AREA (km2)    Population
32  Rivers State      11077    5198605
33  Sokoto State      25973    3702676
34  Taraba State      54473    2294800
35  Yobe State      45502    2321339
36  Zamfara State      39762    3278873
32  5198605
33  3702676
34  2294800
35  2321339
36  3278873
Name: Population, dtype: int64
```

[41]:

In

```
#For summary of descriptive statistics of the dataframe
csv_df.describe()
```

Out[41]:

	AREA (km2)	Population
count	37.000000	3.700000e+01
mean	24990.864865	3.775879e+06 **std**
	18243.870444	1.726418e+06 **min**
	3345.000000	1.405201e+06 **25%**
	9251.000000	2.845380e+06
50%	20156.000000	3.314043e+06
75%	36800.000000	4.177828e+06 **max**
	76363.000000	9.401288e+06

In [42]:

```
#To include summary of descriptive statistics of non numeric columns of the dataframe csv_df.describe(include='all')
```

Out[42]:

	STATES	AREA (km2)	Population
count	37	37.000000	3.700000e+01
unique	37	NaN	NaN
top	Ekiti State	NaN	NaN
freq	1	NaN	NaN
mean	NaN	24990.864865	3.775879e+06 **std** NaN
	18243.870444	1.726418e+06 **min** NaN 3345.000000	
	1.405201e+06 **25%** NaN 9251.000000 2.845380e+06		
50%	NaN	20156.000000	3.314043e+06
75%	NaN	36800.000000	4.177828e+06 **max** NaN
	76363.000000	9.401288e+06	

In [43]:

```
csv_df['Population'].mean()
```

In

Out[43]:

3775879.4594594594

Other descriptive statistics functions are:

- count() Number of non-null observations
- sum() Sum of values
- mean() Mean of Values
- median() Median of Values
- mode() Mode of values
- std() Standard Deviation of the Values
- min() Minimum Value
- max() Maximum Value
- abs() Absolute Value
- prod() Product of Values
- cumsum() Cumulative Sum
- cumprod() Cumulative Product

> Note: Functions like abs(), cumprod() throw exception when the DataFrame contains character or string data because such operations cannot be performed.

In [44]:

```
#To show the features in the dataset
csv_df.columns
```

Out[44]:

Index(['STATES', 'AREA (km2) ', 'Population'], dtype='object')

In [45]:

```
#To show even more information about the dataset

csv_df.info()
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37 entries, 0 to 36 Data columns
(total 3 columns):
STATES 37 non-null object AREA (km2)
37 non-null int64 Population 37 non-null
int64 dtypes: int64(2), object(1) memory usage:
968.0+ bytes

Pandas Idexing

There are several ways to index a Pandas DataFrame. These are:

Square bracket notation: One of the easiest ways to do this is by using square bracket notation.

In

Loc and iloc: loc is label-based, which means that you have to specify rows and columns based on their row and column labels. iloc is integer index based, so you have to specify rows and columns by their integer index Dot (.) notation:

[46]:

```
#Square bracket notation to access all observations of selected features

# Print out states column as Pandas Series
print(csv_df['STATES'])

# Print out state column as Pandas DataFrame
print(csv_df[['STATES']])

# Print out DataFrame with states and population columns
print(csv_df[['STATES', 'Population']])
```

```
0          Abia State
1       Adamawa State
2     Akwa Ibom State
3      Anambra State
4       Bauchi State
5       Bayelsa State
6        Benue State
7        Borno State
8        Cross River
9        Delta State
10       Ebonyi State
11          Edo State
12       Ekiti State
13       Enugu State
14               FCT
15       Gombe State
16         Imo State
17       Jigawa State
18       Kaduna State
```

Note: A single square bracket will output a pandas series while a double square bracket outputs a pandas dataframe

In [47]:

```
#To access features of selected observations (rows) from a DataFrame, square bracket can

# Print out first 4 observations
print(csv_df[0:4], '\n')

# Print out fifth, sixth, and seventh observation
print(csv_df[4:6])
```

```
          STATES  AREA (km2)   Population
0     Abia State        6320      2845380
1  Adamawa State       36917      3178950
2  Akwa Ibom State      7081      3178950
```

In

3	Anambra State	4844	4177828
STATES	AREA (km2)	Population	
4	Bauchi State	45837	4653066
5	Bayelsa State	10773	1704515

[48]:

```
#Using Loc and Iloc

#since the dataset contains no label-based index, we can only use interger based iloc

# Print out observation for the third state
print(csv_df.iloc[2])

# Print out observation for the 4th and 5th state
print(csv_df.iloc[3:5])
```

```
STATES          Akwa Ibom State
AREA (km2)              7081
Population           3178950
Name: 2, dtype: object
      STATES  AREA (km2)  Population
3   Anambra State        4844    4177828
4   Bauchi State        45837    4653066
```

Deleting features/rows in datasets

In [49]:

```
# Drop rows of column called population
df = csv_df.drop(['Population' ], axis = 1)

print (df)
```

```
             STATES  AREA (km2)
0        Abia State          6320
1     Adamawa State         36917
2   Akwa Ibom State          7081
3     Anambra State          4844
4      Bauchi State         45837
5     Bayelsa State         10773
6       Benue State         34059
7       Borno State         70898
8       Cross River         20156
9       Delta State         17698
10     Ebonyi State          5670
11        Edo State         17802
12      Ekiti State          6353
13      Enugu State          7161
14              FCT          7315
15      Gombe State         18768
16        Imo State          5530
17     Jigawa State         23154
```

#using del function del

df['Population'] print(df)

using pop function

df.pop('Population') print (df)
d pop(opu at o) p t (d)

ADDITING TO DATASET

[52]:

```
# adding more features to datset

df['Population' ] = csv_df['Population' ]
df
```

Out[52]:

	STATES	AREA (km2)	Population
0	Abia State	6320	2845380
1	Adamawa State	36917	3178950
2	Akwa Ibom State	7081	3178950
3	Anambra State	4844	4177828
4	Bauchi State	45837	4653066
5	Bayelsa State	10773	1704515
6	Benue State	34059	4253641
7	Borno State	70898	4171104
8	Cross River	20156	2892988
9	Delta State	17698	4112445
10	Ebonyi State	5670	2176947
11	Edo State	17802	3233366
12	Ekiti State	6353	2398957
13	Enugu State	7161	3267837
14	FCT	7315	1405201
15	Gombe State	18768	2365040
16	Imo State	5530	3927563
17	Jigawa State	23154	4361002
18	Kaduna State	46053	6113503
19	Kano State	20131	9401288
20	Katsina State	24192	5801584
21	Kebbi State	36800	3256541
22	Kogi State	29833	3314043
23	Kwara State	36825	2365353
24	Lagos State	3345	9113605
25	Nasarawa State	27117	1869377
26	Niger State	76363	3954772
27	Ogun State	16762	3751140
28	Ondo State	15500	3460877
29	Osun State	9251	3416959
30	Oyo State	28454	5580894

31	Plateau State	30913	3206531
32	Rivers State	11077	5198605
33	Sokoto State	25973	3702676
34	Taraba State	54473	2294800
35	Yobe State	45502	2321339
36	Zamfara State	39762	3278873

STATES **AREA (km2)**

Population

Changing Data type of Pandas datafram and pandas series

[53]:

```
#changing the dtype of features for Series object

df['Population' ] = df['Population' ].astype('float')

df

#or with the use of downcasting

pd.to_numeric(df['Population' ], downcast='integer' )
```

Out[53]:

```
0   2845380
1   3178950
2   3178950
3   4177828
4   4653066
5   1704515
6   4253641
7   4171104
8   2892988
9   4112445
10  2176947
11  3233366
12  2398957
13  3267837
14  1405201
15  2365040
16  3927563
17  4361002
18  6113503
19  9401288
20  5801584
21  3256541
22  3314043
23  2365353
24  9113605
25  1869377
26  3954772
27  3751140
28  3460877
29  3416959
30  5580894
31  3206531
32  5198605
33  3702676
34  2294800
35  2321339
36  3278873
```

Name: Population, dtype: int32 In [54]:

#changing the dtype of features for pandas dataframe

```
df[['Population', 'AREA (km2) ']] = df[['Population', 'AREA (km2) ']].astype(float) df[['Population', 'AREA (km2) ']]
```

Out[54]:

	Population	AREA (km2)
0	2845380.0	6320.0
1	3178950.0	36917.0
2	3178950.0	7081.0
3	4177828.0	4844.0
4	4653066.0	45837.0
5	1704515.0	10773.0
6	4253641.0	34059.0
7	4171104.0	70898.0
8	2892988.0	20156.0
9	4112445.0	17698.0
10	2176947.0	5670.0
11	3233366.0	17802.0
12	2398957.0	6353.0
13	3267837.0	7161.0
14	1405201.0	7315.0
15	2365040.0	18768.0
16	3927563.0	5530.0
17	4361002.0	23154.0
18	6113503.0	46053.0
19	9401288.0	20131.0
20	5801584.0	24192.0
21	3256541.0	36800.0
22	3314043.0	29833.0
23	2365353.0	36825.0
24	9113605.0	3345.0
25	1869377.0	27117.0
26	3954772.0	76363.0
27	3751140.0	16762.0
28	3460877.0	15500.0
29	3416959.0	9251.0
30	5580894.0	28454.0

	Population	AREA (km2)
31	3206531.0	30913.0
32	5198605.0	11077.0
33	3702676.0	25973.0
34	2294800.0	54473.0
35	2321339.0	45502.0
36	3278873.0	39762.0

In [55]:

```
#Adding a new column using the existing columns in DataFrame

df['AreaPopu' ]=df['AREA (km2) ' ]+df['Population' ]

df.columns
```

Out[55]:

```
Index(['STATES', 'AREA (km2) ', 'Population', 'AreaPopu'], dtype='object')
```

> Note: the new feature column must be of the same dimension existing columns

Pandas Method

Sorting

Pandas sorting could be done either by using index or value

In [56]:

```
df.sort_index(inplace = True, ascending = False)

df
```

Out[56]:

	STATES	AREA (km2)	Population	AreaPopu
36	Zamfara State	39762.0	3278873.0	3318635.0
35	Yobe State	45502.0	2321339.0	2366841.0
34	Taraba State	54473.0	2294800.0	2349273.0
33	Sokoto State	25973.0	3702676.0	3728649.0
32	Rivers State	11077.0	5198605.0	5209682.0
31	Plateau State	30913.0	3206531.0	3237444.0
30	Oyo State	28454.0	5580894.0	5609348.0
29	Osun State	9251.0	3416959.0	3426210.0
28	Ondo State	15500.0	3460877.0	3476377.0
27	Ogun State	16762.0	3751140.0	3767902.0

In [57]:

```
# sorting data frame by values. The argument will use column name
#axis -> 0 means sorting will be done row-wise
#ascending -> False
df.sort_values("STATES", axis = 0, ascending = False,
                 inplace = True)
df
```

Out[57]:

	STATES	AREA (km2)	Population	AreaPopu
36	Zamfara State	39762.0	3278873.0	3318635.0
35	Yobe State	45502.0	2321339.0	2366841.0
34	Taraba State	54473.0	2294800.0	2349273.0
33	Sokoto State	25973.0	3702676.0	3728649.0
32	Rivers State	11077.0	5198605.0	5209682.0
31	Plateau State	30913.0	3206531.0	3237444.0
30	Oyo State	28454.0	5580894.0	5609348.0
29	Osun State	9251.0	3416959.0	3426210.0
28	Ondo State	15500.0	3460877.0	3476377.0
27	Ogun State	16762.0	3751140.0	3767902.0
26	Niger State	76363.0	3954772.0	4031135.0
25	Nasarawa State	27117.0	1869377.0	1896494.0
24	Lagos State	3345.0	9113605.0	9116950.0
23	Kwara State	36825.0	2365353.0	2402178.0
22	Kogi State	29833.0	3314043.0	3343876.0
21	Kebbi State	36800.0	3256541.0	3293341.0
20	Katsina State	24192.0	5801584.0	5825776.0
19	Kano State	20131.0	9401288.0	9421419.0
18	Kaduna State	46053.0	6113503.0	6159556.0
17	Jigawa State	23154.0	4361002.0	4384156.0
16	Imo State	5530.0	3927563.0	3933093.0
15	Gombe State	18768.0	2365040.0	2383808.0
14	FCT	7315.0	1405201.0	1412516.0
13	Enugu State	7161.0	3267837.0	3274998.0
12	Ekiti State	6353.0	2398957.0	2405310.0
11	Edo State	17802.0	3233366.0	3251168.0
10	Ebonyi State	5670.0	2176947.0	2182617.0
9	Delta State	17698.0	4112445.0	4130143.0
8	Cross River	20156.0	2892988.0	2913144.0

	STATES	AREA (km2)	Population	AreaPopu
6	Benue State	34059.0	4253641.0	4287700.0
5	Bayelsa State	10773.0	1704515.0	1715288.0
4	Bauchi State	45837.0	4653066.0	4698903.0
3	Anambra State	4844.0	4177828.0	4182672.0
2	Akwa Ibom State	7081.0	3178950.0	3186031.0
1	Adamawa State	36917.0	3178950.0	3215867.0
0	Abia State	6320.0	2845380.0	2851700.0

Pandas DataFrame String operations

Method	Description
lower()	Converts strings in the Series/Index to lower case.
upper()	Converts strings in the Series/Index to upper case.
len()	Computes string length
strip()	Helps strip whitespace(including newline) from each string in the Series/index from both the sides
split(' ')	Splits each string with the given pattern.
cat(sep=' ')	Concatenates the series/index elements with given separator.
get_dummies()	Returns the DataFrame with One-Hot Encoded values.
contains(pattern)	Returns a Boolean value True for each element if the substring contains in the element, else False.
replace(a,b)	Replaces the value a with the value b.
repeat(value)	Repeats each element with specified number of times.
count(pattern)	Returns count of appearance of pattern in each element.
startswith(pattern)	Returns true if the element in the Series/Index starts with the pattern.
endswith(pattern)	Returns true if the element in the Series/Index ends with the pattern.
find(pattern)	Returns the first position of the first occurrence of the pattern.
findall(pattern)	Returns a list of all occurrence of the pattern.
swapcase	Swaps the case lower/upper.
islower()	Checks whether all characters in each string in the Series/Index in lower case or not. Returns Boolean
isupper()	Checks whether all characters in each string in the Series/Index in upper case or not. Returns Boolean.
isnumeric()	Checks whether all characters in each string in the Series/Index are numeric. Returns Boolean.

In [ ]:

for further reading:

https://pbpython.com/pandas_dtypes.html (https://pbpython.com/pandas_dtypes.html)

https://en.wikipedia.org/wiki/Matrix_(mathematics) (https://en.wikipedia.org/wiki/Matrix_(mathematics))

https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/

(https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/)

In [ ]: