Experiment No: 1

Date:

Aim: To Download, install and explore the features of Numpy,Scipy,Jupyter,Statsmodels and Pandas Packages.

Requirements: PC, Internet.

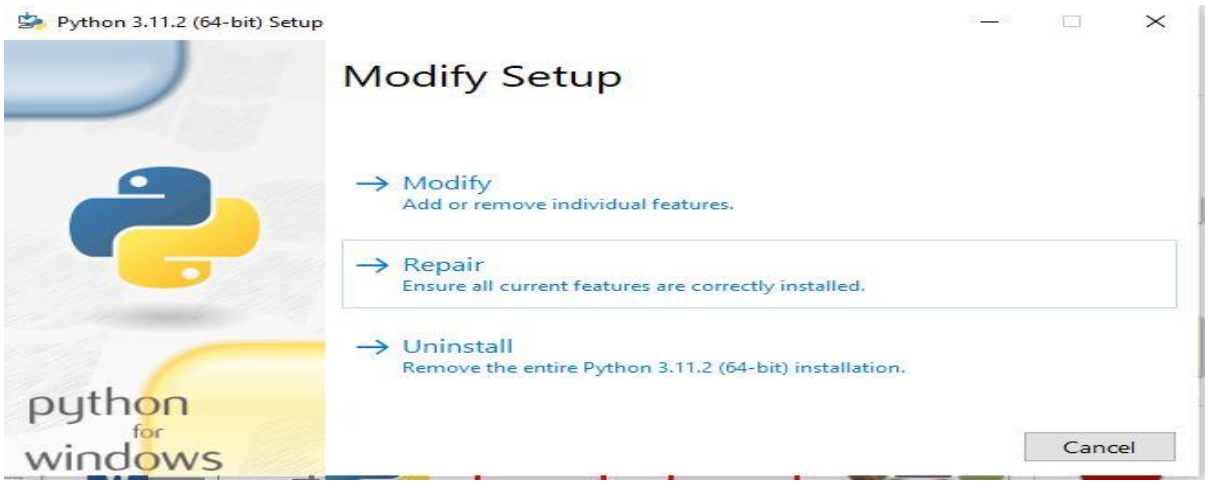
Procedure:

1 Open google Chrome, type www.python.org download. So download the latest version 3.11. 3 of python.

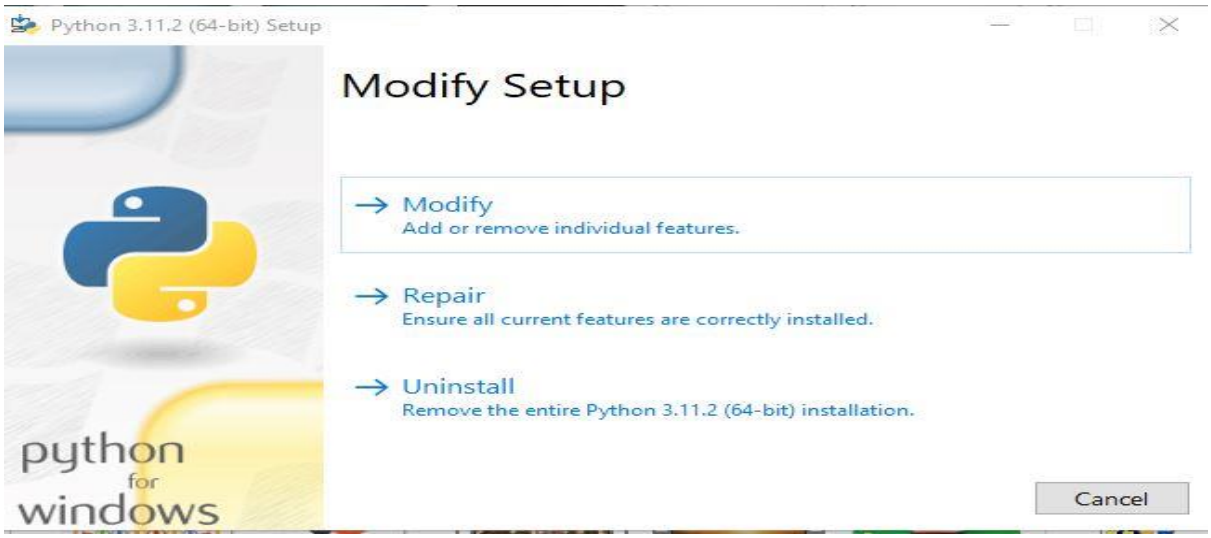
2 Once it is downloaded, make up the basic installation to get access to IDLE.

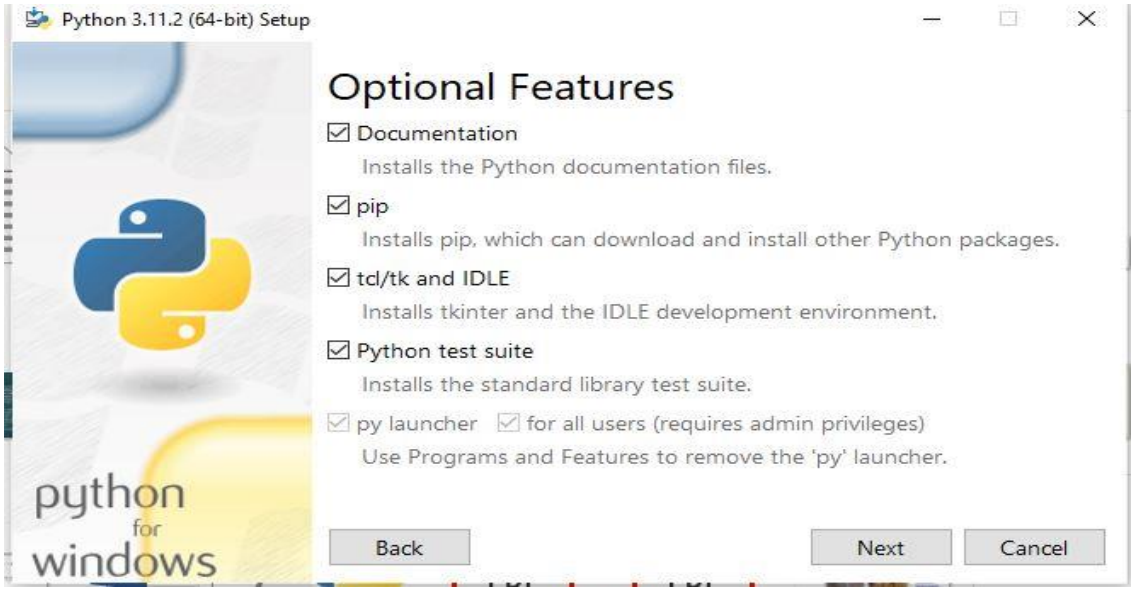
3. Then after, In the File manager, go to python amd setup file. It shows options like Modify, repair, and uninstall.

Click on Repair.



4. After the repair process was done, Click on modify. It displays some optional features, Click on Next and Then Install. Modify process will be complete.



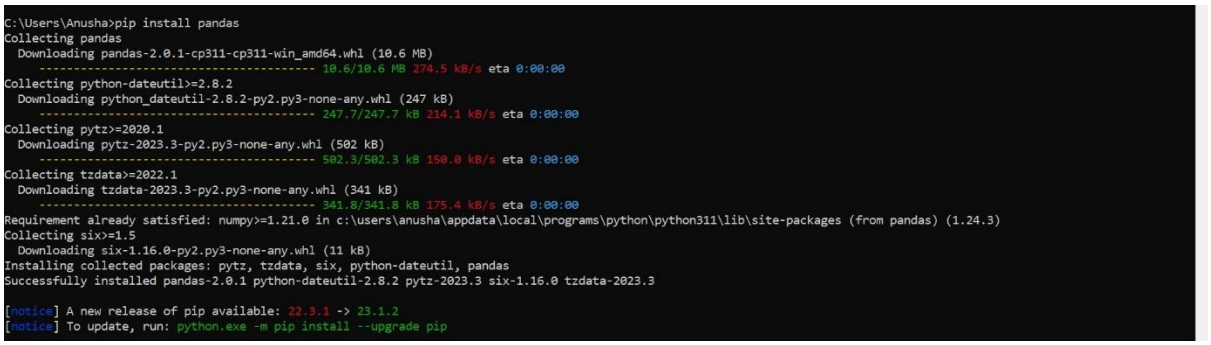


5. Now, Click Windows logo button +R, and type cmd. This takes you to the Command Prompt. A Black Screen will be displayed.

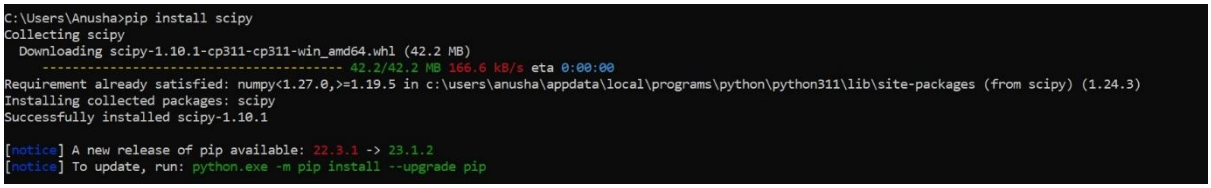
To install Numpy , give **pip install numpy** in the command prompt and click enter.

Then Numpy package will installed.

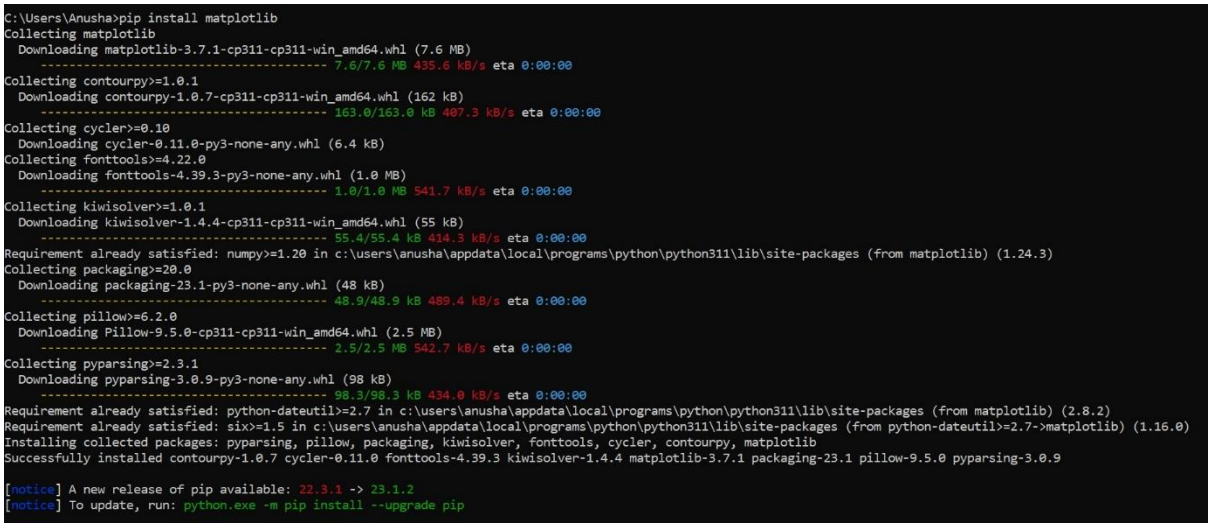
To install pandas, give **pip install pandas** in command prompt followed by enter. Then the Pandas package will be installed.



To install scipy, give **pip install scipy** in command prompt followed by enter. Then the Scipy package will be installed.



To install matplotlib, give **pip install matplotlib** in command prompt followed by enter. Then matplotlib package will be installed.



Result: Hence, The python packages like numpy, scipy, pandas, matplotlib are successfully installed and their features are explored.

Experiment No: 2

Date:

Aim: To write a program to perform Exploratory Data analysis (EDA) for Classification using Pandas and Matplotlib.

Requirements: Jupyter notebook, Sample Data set.

Procedure:

1 Firstly, open jupyter notebook using command prompt or anaconda navigator. Upload the sample dataset i.e.; car_prices.csv file in the notebook.

2. Now, open a new Python3 Kernel and perform the Data analysis.

Program code:

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(color_codes=True)
```

In [2]:
```
df = pd.read_csv('car_prices.csv')
df.head()
```

Out[2]:

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	Vehicle Style	highway MPG	city mpg	Popularit
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact	Coupe	26	19	391
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Convertible	28	19	391
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	Coupe	28	20	391
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Coupe	28	18	391
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible	28	18	391

In [3]:
```
df.dtypes
```

Out[3]:
```
Make                object
Model               object
Year                 int64
Engine Fuel Type    object
Engine HP          float64
Engine Cylinders   float64
Transmission Type   object
Driven_Wheels       object
Number of Doors    float64
Market Category     object
Vehicle Size        object
Vehicle Style       object
highway MPG          int64
city mpg             int64
Popularity           int64
MSRP                 int64
dtype: object
```

In [4]:
```
df = df.drop(['Engine Fuel Type', 'Market Category', 'Vehicle Style', 'Popularity', 'Number of Doors', 'Vehicle Size'], axis=1)
df.head(5)
```

Out[4]:

	Make	Model	Year	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	highway MPG	city mpg	MSRP
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

In [5]:
```
df = df.rename(columns={"Engine HP": "HP", "Engine Cylinders": "Cylinders", "Transmission Type": "Transmission", "Driven_Wheels"
df.head(5)
```

Out[5]:

	Make	Model	Year	HP	Cylinders	Transmission	Drive Mode	MPG-H	MPG-C	Price
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

In [6]:
```
df.shape
```

Out[6]:
```
(11914, 10)
```

In [7]:
```
duplicate_rows_df = df[df.duplicated()]
print("Number of duplicate rows: ", duplicate_rows_df.shape)
```

```
Number of duplicate rows:  (989, 10)
```

In [8]:
```
df = df.drop_duplicates()
df.head(5)
```

Out[8]:

	Make	Model	Year	HP	Cylinders	Transmission	Drive Mode	MPG-H	MPG-C	Price
0	BMW	1 Series M	2011	335.0	6.0	MANUAL	rear wheel drive	26	19	46135
1	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	19	40650
2	BMW	1 Series	2011	300.0	6.0	MANUAL	rear wheel drive	28	20	36350
3	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	29450
4	BMW	1 Series	2011	230.0	6.0	MANUAL	rear wheel drive	28	18	34500

In [9]:
```
print(df.isnull().sum())
```

```
Make              0
Model             0
Year              0
HP               69
Cylinders        30
Transmission      0
Drive Mode        0
MPG-H             0
MPG-C             0
Price             0
dtype: int64
```

In [10]:
```
df = df.dropna()
df.shape
```

Out[10]: (10827, 10)

In [11]:
```
df.describe()
```

Out[11]:

	Year	HP	Cylinders	MPG-H	MPG-C	Price
count	10827.000000	10827.000000	10827.000000	10827.000000	10827.000000	1.082700e+04
mean	2010.896370	254.553062	5.691604	26.308119	19.327607	4.249325e+04
std	7.029534	109.841537	1.768551	7.504652	6.643567	6.229451e+04
min	1990.000000	55.000000	0.000000	12.000000	7.000000	2.000000e+03
25%	2007.000000	173.000000	4.000000	22.000000	16.000000	2.197250e+04
50%	2015.000000	240.000000	6.000000	25.000000	18.000000	3.084500e+04
75%	2016.000000	303.000000	6.000000	30.000000	22.000000	4.330000e+04
max	2017.000000	1001.000000	16.000000	354.000000	137.000000	2.065902e+06

In [12]:
```
plt.figure(figsize=(10,5))
sns.heatmap(df.corr(),annot=True)
```

Out[12]: <AxesSubplot:>

Result: Hence, Exploratory Data Analysis is performed for Classification using pandas and matplotlib.

Experiment No: 3

Date:

Aim: To Read data from text files, Excel and the web and exploring various commands for doing descriptive analytics on the Iris data set.

Requirements: Jupyter notebook, Iris Data set from web.

Procedure:

1. Download Iris data set from the web and upload it in the Jupyter notebook.

2. Now, Open new python kernel and perform required operations.

Program Code:

In [11]:
```
import pandas as pd
```

In [12]:
```
#Reading the CSV file
df=pd.read_csv('iris.csv')
#printing top 5 rows
df.head()
```

Out[12]:

	sepallength	sepalwidth	petallength	petalwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [13]:
```
df.shape
```

Out[13]: (150, 5)

In [14]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   sepallength  150 non-null    float64
 1   sepalwidth   150 non-null    float64
 2   petallength  150 non-null    float64
 3   petalwidth   150 non-null    float64
 4   class        150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [15]:
```
df.describe()
```

Out[15]:

	sepallength	sepalwidth	petallength	petalwidth
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [16]:
```
df.isnull().sum()
```

Out[16]:
```
sepallength    0
sepalwidth     0
petallength    0
petalwidth     0
class          0
dtype: int64
```

In [18]:
```
data=df.drop_duplicates(subset='class',)
data
```

Out[18]:

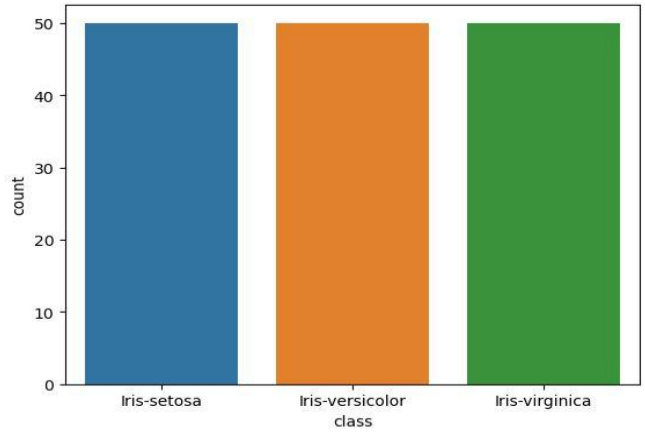
	sepallength	sepalwidth	petallength	petalwidth	class
0	5.1	3.5	1.4	0.2	Iris-setosa
50	7.0	3.2	4.7	1.4	Iris-versicolor
100	6.3	3.3	6.0	2.5	Iris-virginica

In [21]:
```
df.value_counts('class')
```

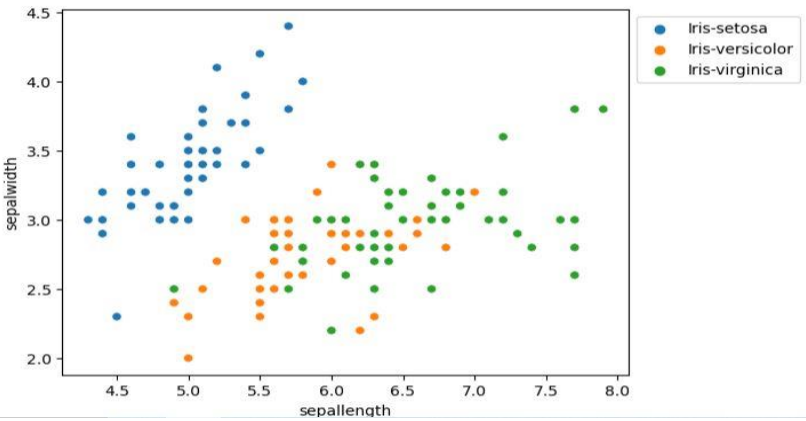
Out[21]:
```
class
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

In [22]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

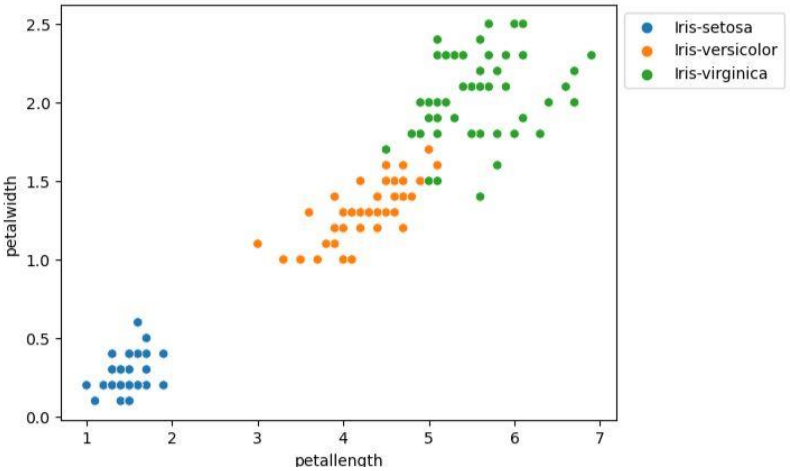
In [23]:
```
sns.countplot(x='class',data=df,)
plt.show()
```



In [24]:
```
sns.scatterplot(x='sepallength',y='sepalwidth',hue='class',data=df,)
plt.legend(bbox_to_anchor=(1,1),loc=2)
plt.show()
```



In [25]:
```
sns.scatterplot(x='petallength',y='petalwidth',hue='class',data=df,)
plt.legend(bbox_to_anchor=(1,1),loc=2)
plt.show()
```



Result: Hence, Exploratory data analysis is performed on Iris Data Set.

Experiment No: 4a

Date:

Aim: To read different types of Data sets(.txt,.csv) from Web and disk and writing in file in specific disk location.

Requirements: Sample .txt file, Sample .csv file.

- **READING A (.TXT) FILE :**

Procedure:

1 Create a sample .txt or It can be downloaded from the web. Upload it in Jupyter notebook and read the file.

For Example: Sample .txt

It contains:

Data science is the study of data to extract meaningful insights for business. It is a multidisciplinary approach that combines principles and practices from the fields of mathematics, statistics, artificial intelligence, and computer engineering to analyze large amounts of data.

Program Code:

```
In [2]: # read text file
        with open('Sample.txt','r') as f:
            print(f.read())
```

```
Data science is the study of data to extract meaningful insights for business.
It is a multidisciplinary approach that combines principles and practices from the fields of mathematics, statistics, artificia
l intelligence, and computer engineering to analyze large amounts of data.
```

```
In [3]: #read text file
        with open('Sample.txt','r') as f:
            print(f.readline())
```

```
Data science is the study of data to extract meaningful insights for business.
```

```
In [4]: with open('Sample.txt','r') as f:
            print(f.readlines())
```

```
['Data science is the study of data to extract meaningful insights for business. \n', 'It is a multidisciplinary approach that
combines principles and practices from the fields of mathematics, statistics, artificial intelligence, and computer engineering
to analyze large amounts of data.']
```

- **READING A CSV FILE:**

Procedure:

**Download sample csv file from the web. Upload it in Jupyter Notebook and read .csv file.

For example: Create Products.csv

Id, Product, Price

1, Pen, 10

2, Pencil, 5

3, Eraser, 2

4, Notebook, 40

5, Stapler, 60

Program Code:

```
In [5]: import pandas as pd
        #read csv file into a dataframe
        df=pd.read_csv('Products.csv')
        #displaying DataFrame
        print(df)
```

```
   Id    Product   Price
0   1        Pen      10
1   2     Pencil       5
2   3     Eraser       2
3   4   Notebook      40
4   5    Stapler      60
```

Result: Hence, the .txt,.csv files that are loaded from the web are read using python, Jupyter Notebook

Experiment No: 4b

Date:

Aim: To read Excel files using Python

Requirements: Jupyter Notebook

Procedure:

Download Sample Excel sheet from the web. Then upload it in the Jupyter Notebook and perform the operations to read the excel file.

For Example: world.xlsx

Program Code:

```
In [3]: import pandas as pd
        #read excel file into a dataFrame
        df=pd.read_excel('world.xlsx')
        #print values
        print(df)
```

```
   0 First Name  Last Name  Gender        Country  Age        Date    Id
0  1      Dulce      Abril  Female  United States   32  15/10/2017  1562
1  2       Mara  Hashimoto  Female  Great Britain   25  16/08/2016  1582
2  3     Philip       Gent    Male         France   36  21/05/2015  2587
3  4   Kathleen     Hanner  Female  United States   25  15/10/2017  3549
4  5    Nereida    Magwood  Female  United States   58  16/08/2016  2468
5  6     Gaston      Brumm    Male  United States   24  21/05/2015  2554
6  7       Etta       Hurn  Female  Great Britain   56  15/10/2017  3598
7  8    Earlean     Melgar  Female  United States   27  16/08/2016  2456
8  9   Vincenza    Weiland  Female  United States   40  21/05/2015  6548
```

```
In [4]: xl=pd.ExcelFile('world.xlsx')
        xl.sheet_names
```

```
Out[4]: ['Sheet1']
```

```
In [7]: df=pd.read_excel('world.xlsx')
        df
```

Out[7]:

	0	First Name	Last Name	Gender	Country	Age	Date	Id
0	1	Dulce	Abril	Female	United States	32	15/10/2017	1562
1	2	Mara	Hashimoto	Female	Great Britain	25	16/08/2016	1582
2	3	Philip	Gent	Male	France	36	21/05/2015	2587
3	4	Kathleen	Hanner	Female	United States	25	15/10/2017	3549
4	5	Nereida	Magwood	Female	United States	58	16/08/2016	2468
5	6	Gaston	Brumm	Male	United States	24	21/05/2015	2554
6	7	Etta	Hurn	Female	Great Britain	56	15/10/2017	3598
7	8	Earlean	Melgar	Female	United States	27	16/08/2016	2456
8	9	Vincenza	Weiland	Female	United States	40	21/05/2015	6548

Result: Hence, The excel file is read using Python

Experiment No : 5

Date :

Aim: To write a python program to find basic descriptive statistics using summary, str,quartile function on mtcars datasets.

Requirements: Jupyter notebook, and mtcars dataset from the web.

Procedure: Download mtcars dataset from the web. and upload it the Jupyter notebook. Perform different operations using summary, str,quartile etc;

Program code:

In [1]:
```
%matplotlib inline

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

In [3]:
```
mtcars = pd.read_csv("mtcars.csv")
mtcars =  mtcars.rename(columns={'Unnamed: 0': 'model'})
mtcars.index = mtcars.model
del mtcars["model"]

mtcars.head()
```

Out[3]:

model	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2

In [4]:
```
mtcars.mean()
```

Out[4]:
```
mpg      20.090625
cyl       6.187500
disp    230.721875
hp      146.687500
drat      3.596563
wt        3.217250
qsec     17.848750
vs        0.437500
am        0.406250
gear      3.687500
carb      2.812500
dtype: float64
```

In [5]:
```
mtcars.mean(axis=1)
```

Out[5]:
```
model
Mazda RX4              29.907273
Mazda RX4 Wag          29.981364
Datsun 710             23.598182
Hornet 4 Drive         38.739545
Hornet Sportabout      53.664545
Valiant                35.049091
Duster 360             59.720000
Merc 240D              24.634545
Merc 230               27.233636
Merc 280               31.860000
Merc 280C              31.787273
Merc 450SE             46.430909
Merc 450SL             46.500000
Merc 450SLC            46.350000
Cadillac Fleetwood     66.232727
Lincoln Continental    66.058545
Chrysler Imperial      65.972273
Fiat 128               19.440909
Honda Civic            17.742273
Toyota Corolla         18.814091
Toyota Corona          24.888636
Dodge Challenger       47.240909
AMC Javelin            46.007727
Camaro Z28             58.752727
Pontiac Firebird       57.379545
Fiat X1-9              18.928636
Porsche 914-2          24.779091
Lotus Europa           24.880273
Ford Pantera L         60.971818
Ferrari Dino           34.508182
Maserati Bora          63.155455
Volvo 142E             26.262727
dtype: float64
```

In [6]:
```
mtcars.median()
```

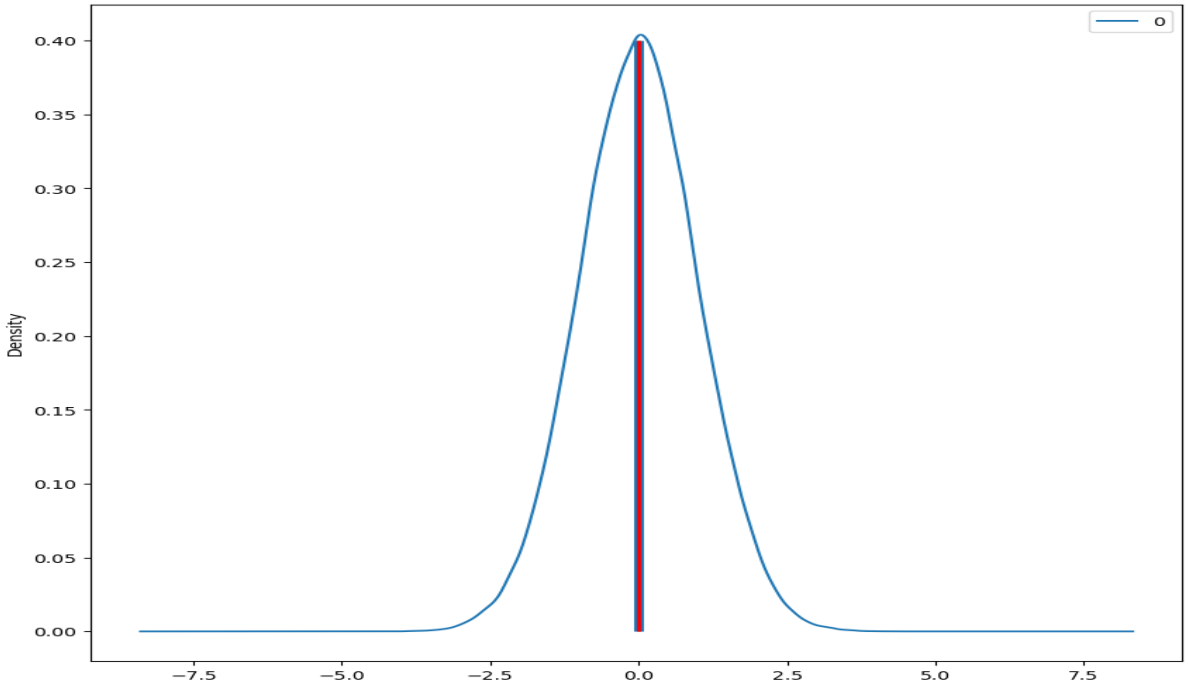
Out[6]:
```
mpg      19.200
cyl       6.000
disp    196.300
hp      123.000
drat      3.695
wt        3.325
qsec     17.710
vs        0.000
am        0.000
gear      4.000
carb      2.000
dtype: float64
```

In [7]:
```
norm_data = pd.DataFrame(np.random.normal(size=100000))

norm_data.plot(kind="density",
               figsize=(10,10));


plt.vlines(norm_data.mean(),     # Plot black line at mean
           ymin=0,
           ymax=0.4,
           linewidth=5.0);

plt.vlines(norm_data.median(),   # Plot red line at median
           ymin=0,
           ymax=0.4,
           linewidth=2.0,
           color="red");
```

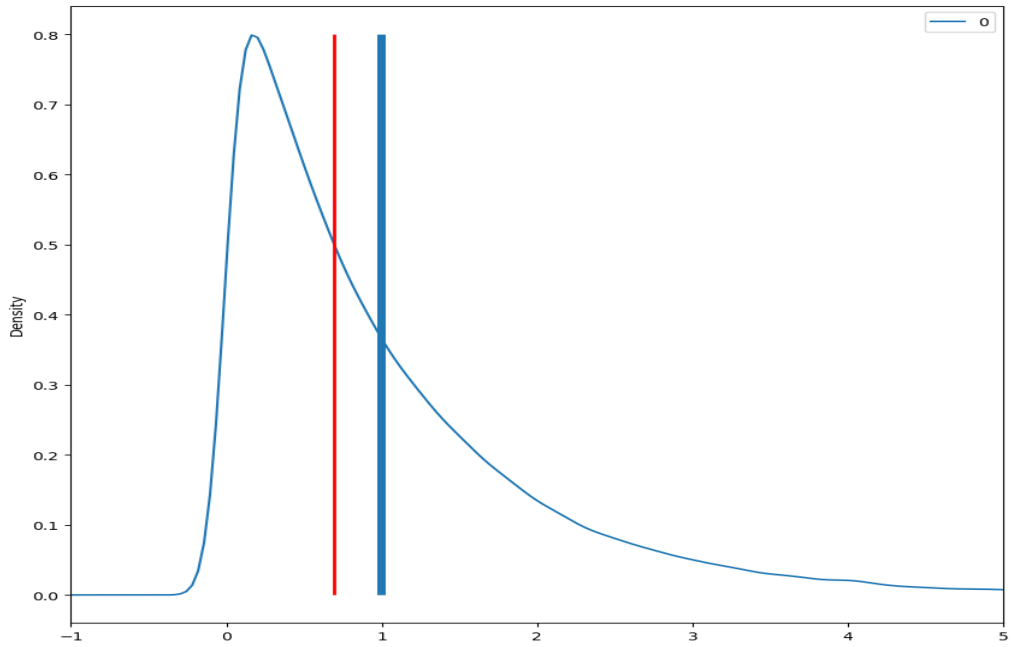


In [8]:
```
skewed_data = pd.DataFrame(np.random.exponential(size=100000))

skewed_data.plot(kind="density",
                 figsize=(10,10),
                 xlim=(-1,5));


plt.vlines(skewed_data.mean(),     # Plot black line at mean
           ymin=0,
           ymax=0.8,
           linewidth=5.0);

plt.vlines(skewed_data.median(),   # Plot red line at median
           ymin=0,
           ymax=0.8,
           linewidth=2.0,
           color="red");
```

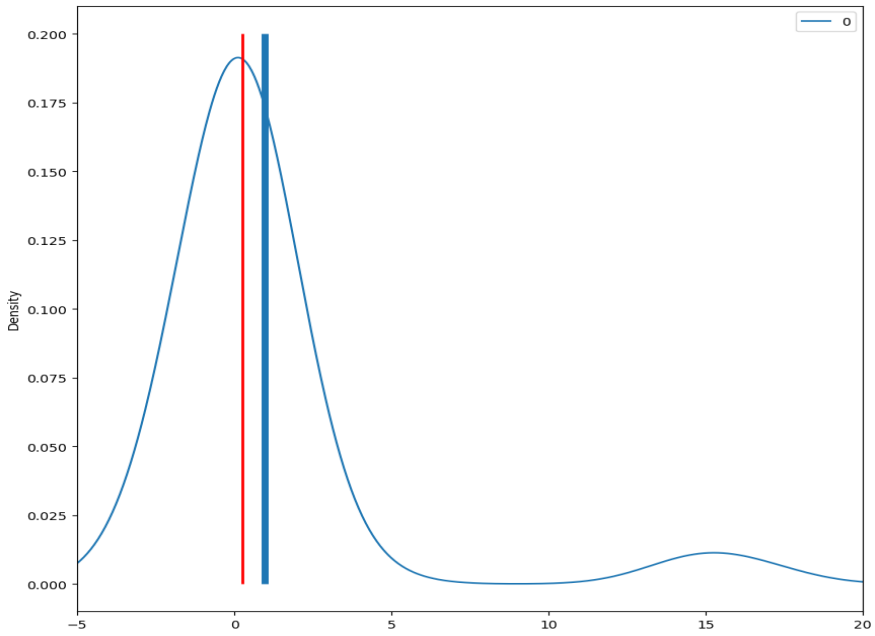


In [9]:
```
norm_data = np.random.normal(size=50)
outliers = np.random.normal(15, size=3)
combined_data = pd.DataFrame(np.concatenate((norm_data, outliers), axis=0))

combined_data.plot(kind="density",
              figsize=(10,10),
              xlim=(-5,20));


plt.vlines(combined_data.mean(),     # Plot black line at mean
           ymin=0,
           ymax=0.2,
           linewidth=5.0);

plt.vlines(combined_data.median(),   # Plot red line at median
           ymin=0,
           ymax=0.2,
           linewidth=2.0,
           color="red");
```



In [10]:
```
mtcars.mode()
```

Out[10]:

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
0	10.4	8.0	275.8	110.0	3.07	3.44	17.02	0.0	0.0	3.0	2.0
1	15.2	NaN	NaN	175.0	3.92	NaN	18.90	NaN	NaN	NaN	4.0
2	19.2	NaN	NaN	180.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	21.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	21.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	22.8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	30.4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [11]:
```
max(mtcars["mpg"]) - min(mtcars["mpg"])
```

Out[11]:
```
23.5
```

In [12]:
```
five_num = [mtcars["mpg"].quantile(0),
            mtcars["mpg"].quantile(0.25),
            mtcars["mpg"].quantile(0.50),
            mtcars["mpg"].quantile(0.75),
            mtcars["mpg"].quantile(1)]

five_num
```

Out[12]:
```
[10.4, 15.425, 19.2, 22.8, 33.9]
```

In [13]:
```
mtcars["mpg"].describe()
```

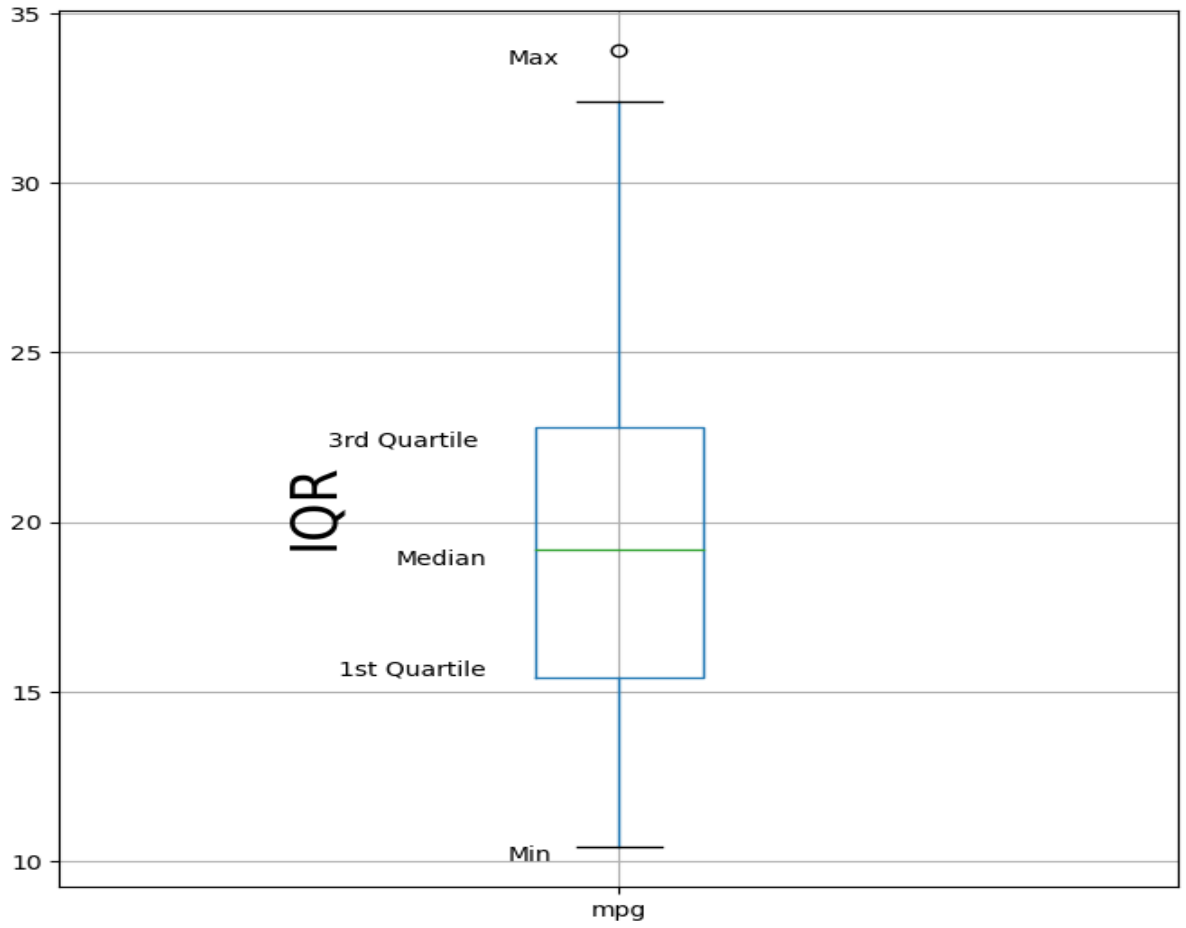
Out[13]:
```
count    32.000000
mean     20.090625
std       6.026948
min      10.400000
25%      15.425000
50%      19.200000
75%      22.800000
max      33.900000
Name: mpg, dtype: float64
```

In [14]:
```
mtcars["mpg"].quantile(0.75) - mtcars["mpg"].quantile(0.25)
```

Out[14]: 7.375

In [15]:
```
mtcars.boxplot(column="mpg",
               return_type='axes',
               figsize=(8,8))

plt.text(x=0.74, y=22.25, s="3rd Quartile")
plt.text(x=0.8, y=18.75, s="Median")
plt.text(x=0.75, y=15.5, s="1st Quartile")
plt.text(x=0.9, y=10, s="Min")
plt.text(x=0.9, y=33.5, s="Max")
plt.text(x=0.7, y=19.5, s="IQR", rotation=90, size=25);
```



In [16]:
```
mtcars["mpg"].var()
```

Out[16]: 36.32410282258065

In [17]:
```
mtcars["mpg"].std()
```

Out[17]: 6.026948052089105

In [18]:
```
abs_median_devs = abs(mtcars["mpg"] - mtcars["mpg"].median())

abs_median_devs.median() * 1.4826
```

Out[18]: 5.411490000000001

In [19]:
```
mtcars["mpg"].skew()
```

Out[19]: 0.6723771376290805

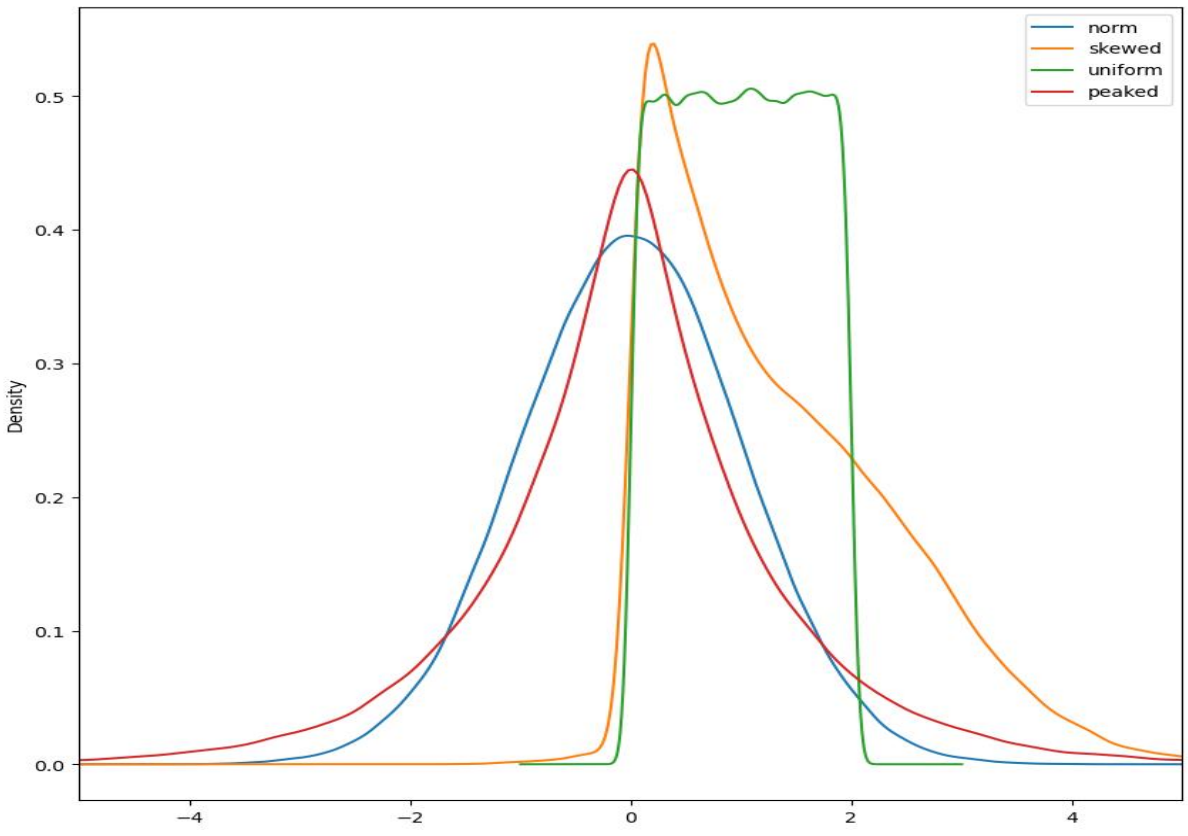
In [20]:
```
mtcars["mpg"].kurt()
```

Out[20]: -0.0220062914240855

In [21]:
```
norm_data = np.random.normal(size=100000)
skewed_data = np.concatenate((np.random.normal(size=35000)+2,
                              np.random.exponential(size=65000)),
                              axis=0)
uniform_data = np.random.uniform(0,2, size=100000)
peaked_data = np.concatenate((np.random.exponential(size=50000),
                              np.random.exponential(size=50000)*(-1)),
                              axis=0)

data_df = pd.DataFrame({"norm":norm_data,
                        "skewed":skewed_data,
                        "uniform":uniform_data,
                        "peaked":peaked_data})
```

In [22]:
```
data_df.plot(kind="density",
             figsize=(10,10),
             xlim=(-5,5));
```



```
In [23]: data_df.skew()

Out[23]: norm      -0.005625
         skewed     0.999534
         uniform   -0.002307
         peaked    -0.011550
         dtype: float64

In [24]: data_df.kurt()

Out[24]: norm       0.009454
         skewed     1.273474
         uniform   -1.201107
         peaked     3.034400
         dtype: float64
```

Result: Hence, Analysis operations are done for mtcars dataset using summary, str, quartile functions.

Experiment No: 7a

Date:

Aim: To perform Univariate analysis (Frequency, mean, mode, Standard deviation , Skewness and kurtosis) Using the diabetes, UCI and Pima Indians Diabetes dataset.

Requirements: Pima Indians Diabetes dataset, Jupyter notebook

Procedure:

1. To perform univariate analysis on the Pima Indians Diabetes dataset using Python, you can follow the steps below. First, make sure you have the necessary packages installed, such as pandas and scipy. You can install them using pip if needed.

In this code, we load the dataset from the provided from Kaggle.com and assign appropriate column names. Then, we calculate the frequency of the "Outcome" variable using the `value_counts()` function. Next, we calculate the mean, median, mode, variance, standard deviation, skewness, and kurtosis of all the variables in the dataset using various functions from pandas and scipy. stats.

Finally, Print the results. Hence Univariate analysis on Pima Indians diabetes dataset will be done.

Program code:

In [3]:
```
import pandas as pd
from scipy.stats import skew,kurtosis
column_names=["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","DiabetesPedigreeFunction","Age","Outcome"]
data=pd.read_csv("pima indians.csv",names=column_names)
```

In [4]:
```
data.head()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [7]:
```
#To calculate Frequency
frequency=data["Outcome"].value_counts()
#To calculate mean
mean=data.mean()
#calculate median
median=data.median()
#Print results
print("Frequency:\n", frequency)
print("\nMean:\n", mean)
print("\nMedian:\n", median)
```

```
Frequency:
 0    500
1    268
Name: Outcome, dtype: int64

Mean:
 Pregnancies                 3.845052
Glucose                   120.894531
BloodPressure              69.105469
SkinThickness              20.536458
Insulin                    79.799479
BMI                        31.992578
DiabetesPedigreeFunction    0.471876
Age                        33.240885
Outcome                     0.348958
dtype: float64

Median:
 Pregnancies                 3.0000
Glucose                   117.0000
BloodPressure              72.0000
SkinThickness              23.0000
Insulin                    30.5000
BMI                        32.0000
DiabetesPedigreeFunction    0.3725
Age                        29.0000
Outcome                     0.0000
dtype: float64
```

In [9]:
```
#calculate mode
mode=data.mode().iloc[0]
#calculate variance
variance=data.var()
#caalculate standard seviation
std_deviation=data.std()
#Print results
print("\nMode:\n", mode)
print("\nVariance:\n", variance)
print("\nStandard Deviation:\n", std_deviation)
```

```
Mode:
 Pregnancies                 1.000
Glucose                    99.000
BloodPressure              70.000
SkinThickness               0.000
Insulin                     0.000
BMI                        32.000
DiabetesPedigreeFunction    0.254
Age                        22.000
Outcome                     0.000
Name: 0, dtype: float64

Variance:
 Pregnancies                 11.354056
Glucose                   1022.248314
BloodPressure              374.647271
SkinThickness              254.473245
Insulin                  13281.180078
BMI                         62.159984
DiabetesPedigreeFunction     0.109779
Age                        138.303046
Outcome                      0.227483
dtype: float64

Standard Deviation:
 Pregnancies                  3.369578
Glucose                     31.972618
BloodPressure               19.355807
SkinThickness               15.952218
Insulin                    115.244002
BMI                          7.884160
DiabetesPedigreeFunction     0.331329
Age                         11.760232
Outcome                      0.476951
dtype: float64
```

In [10]:
```
#Calculate skewness
skewness=data.skew()
#Calculate kurtosis
kurt=data.kurtosis()
#Print results
print("\nSkewness:\n", skewness)
print("\nKurtosis:\n", kurt)
```

```
Skewness:
 Pregnancies                0.901674
Glucose                     0.173754
BloodPressure              -1.843608
SkinThickness               0.109372
Insulin                     2.272251
BMI                        -0.428982
DiabetesPedigreeFunction    1.919911
Age                         1.129597
Outcome                     0.635017
dtype: float64

Kurtosis:
 Pregnancies                0.159220
Glucose                     0.640780
BloodPressure               5.180157
SkinThickness              -0.520072
Insulin                     7.214260
BMI                         3.290443
DiabetesPedigreeFunction    5.594954
Age                         0.643159
Outcome                    -1.600930
dtype: float64
```

Result:

Hence, Univariate analysis is performed using python packages pandas and scipy on Pima Indians diabetes dataset.

Experiment No: 7b

Date:

Aim: To perform Bivariate analysis (logistic and linear regression modelling) using diabetes dataset.

Requirements: Diabetes dataset, Jupyter notebook

Procedure: In this program, we first load the dataset using the provided URL and define the column names. Then, we split the data into features (`X`) and the target variable (`y`). Next, we split the data into training and testing sets using 80% for training and 20% for testing.

We then create an instance of the `Linear Regression` class and fit the model using the training data. After that, we make predictions on the test data and calculate the mean squared error (MSE) as a measure of the model's performance.

Similarly, we create an instance of the `Logistic Regression` class, fit the model using the training data, and make predictions on the test data. We calculate the accuracy of the logistic regression model using the `accuracy_score` function.

Program Code:

In [2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, accuracy_score

# Load the dataset
column_names = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction", "Age",
df = pd.read_csv("pima indians.csv", names=column_names)
```

In [5]:
```
# Split the data into features and target
X = df.drop("Outcome", axis=1)
y = df["Outcome"]
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [6]:
```
# Linear regression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
linear_predictions = linear_model.predict(X_test)
linear_mse = mean_squared_error(y_test, linear_predictions)
print("Linear Regression MSE:", linear_mse)
```

```
Linear Regression MSE: 0.171045272808501
```

In [7]:
```
# Logistic regression
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
logistic_predictions = logistic_model.predict(X_test)
logistic_accuracy = accuracy_score(y_test, logistic_predictions)
print("Logistic Regression Accuracy:", logistic_accuracy)
```

```
Logistic Regression Accuracy: 0.7467532467532467
```

Result: Hence Bivariate analysis includes Linear regression and Logistic regression is performed on Pima Indians Diabetes dataset using python packages like pandas and sklearn.

Problem No: 10

Date:

Aim: To develop a python program to build Linear Regression model and Logistic regression model and predict the numerical quantities.

Requirements: Sample data set (Pima Indians diabetes dataset), Python.

Procedure:

1. In this program, we first load the dataset from the provided URL and split it into features (`X`) and the target variable (`y`). We then split the data into training and testing sets using an 80-20 split.

2. Next, we build a linear regression model by creating an instance of the `Linear Regression` class and fit the model using the training data. We then use the model to predict numerical quantities on the test data (`X_test`) and store the predictions in `linear_predictions`.

3. Similarly, we build a logistic regression model by creating an instance of the `LogisticRegression` class and fit the model using the training data. We then use the model to predict numerical quantities on the test data (`X_test`) and store the predictions in `logistic_predictions`.

4. Finally, we print the predictions for both the linear regression and logistic regression models.

Program code:

In [2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression

# Load the dataset
column_names = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI", "DiabetesPedigreeFunction", "Age",
df = pd.read_csv("pima indians.csv", names=column_names)

# Split the data into features and target
X = df.drop("Outcome", axis=1)
y = df["Outcome"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [3]:
```
# Build linear regression model
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
# Predict numerical quantities using linear regression model
linear_predictions = linear_model.predict(X_test)
print("Linear Regression Predictions:")
print(linear_predictions)
```

```
Linear Regression Predictions:
[ 0.33550028  0.23809869  0.1510522   0.2401365   0.48142376  0.45257375
 -0.17450469  0.60662287  0.52417796  0.70476953  0.32360466  0.85290601
  0.38466612  0.36056948  0.09946712  0.41539557  0.17869123  0.07782301
  0.80730861  0.51299477  0.28090594  0.08303057  0.5099157   0.11381771
  0.51325022  0.82528549  0.17892718 -0.0594202   0.28338572  0.16407949
  0.83851225  0.80737515  0.68154389  0.7649502   0.56140297  0.62123131
  1.06134554  0.30990775  0.51752336  0.63691482  0.07075333  0.57757007
  0.55015462  0.37541745 -0.07644182  0.50119208  0.59600162  0.27464761
  0.42477995  0.9941898   0.00969584  0.61763578  0.73395288  0.31090975
  0.13456812 -0.02536316  0.71219147 -0.30518218  0.41994556  0.67869594
  0.66891428  0.3798452   0.2956646   0.288035    0.06813053  0.55464338
  0.01368504  0.6272007  -0.02033281  0.6372293   0.61928494  0.07019372
  0.26388322  0.14080565  0.12425109  0.50054317  0.24772661  0.21027229
  0.18419241  0.28346361  0.60206367  0.19720081  0.04718638  0.39163459
  0.31373787  0.75789609  0.82549769  0.35944228  0.1723114   0.0957888
  0.05894136  0.277268   -0.35746245  0.52802473  0.48569971  0.57670079
  0.40681613  0.16649133  0.56927171  0.09451543  0.6570335   0.03311435
  0.68073803  0.48441106  0.58967882  0.27055501  0.33149868  0.66512401
  0.17581258  0.51566149  0.13045166  0.38010107 -0.0949753   0.65582849
  0.23302651  0.3716743   0.68391471  0.28174341  0.05450268  0.53690397
  0.04284507  0.33357357  0.30472023  0.10053203  0.33006507  0.44782371
  0.02663058  0.82020965  1.03616317  0.66672645  0.6518381   0.77042295
  0.11555357  0.44926623  0.72795331  0.15230489  0.21288603  0.76637265
  0.72722441 -0.20395979  0.12946513 -0.02149655  0.27508285  0.39903148
  0.15993455  0.33468331  0.20438069 -0.12662191  0.43170733  0.68158975
  0.163167    0.4815615   0.30101739  0.26110909]
```

In [4]:
```
# Build logistic regression model
logistic_model = LogisticRegression()
logistic_model.fit(X_train, y_train)
# Predict numerical quantities using logistic regression model
logistic_predictions = logistic_model.predict(X_test)
print("Logistic Regression Predictions:")
print(logistic_predictions)
```

```
Logistic Regression Predictions:
[0 0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 1 1 0 0 1 0 1 1 0 0 0 0 1 1 1 1 1 1
 0 1 1 0 1 1 0 0 1 1 0 0 1 0 1 1 0 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 1 1 0 0 0
 0 1 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 1 1 1 0 0 1 0 1 0 1 1 1 0 0 1 0 1 0
 0 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0
 0 1 0 0 0 0]
```

Result: Hence, a python is developed to build a Linear and Logistic Regression model and the numerical quantities are predicted for Pima Indians diabetes dataset.

Problem No: 11

Date:

Aim: To import data from web storage and To perform Logistic Regression to compute relation between variables that are affecting admission of student in a instance based on his/her GRE score, GPA obtained by the student, and also To check model is fit or not.

Requirements: Sample data set , Python Software.

Procedure:

In this program, we start by loading the dataset from the provided URL using `pd.read_csv`. Then, we rename the columns for convenience.

Next, we preprocess the data by converting the "Rank" variable to a categorical variable using `as type ("category"). We create dummy variables for the "Rank" variable using `pd.get_dummies` and drop the first dummy column to avoid multicollinearity issues. We then concatenate the original data frame with the dummy variables.

The independent variables (GRE, GPA, and rank dummies) are stored in the `X` data frame, and the dependent variable (admission) is stored in the `y` series.

We perform logistic regression using `sm.Logit` and obtain the result using the `fit` method. We print the summary of the logistic regression model using `result_Summary ()`.

Finally, we check the model's fit by predicting the admission probabilities for the same dataset. We round the predictions to 0 or 1, and calculate the accuracy by comparing the predicted classes with the actual admission values.

Program code:

Problem No: 14 a

Date:

Aim: To find the data distributions using box and scatter plot for a sample Data Frame.

Requirements: Python

Procedure:

In this program, we have a sample data frame called `data` with columns 'A', 'B', and 'C' to demonstrate the various plots.

Box Plot: We use `boxplot` function to create a box plot of the data.

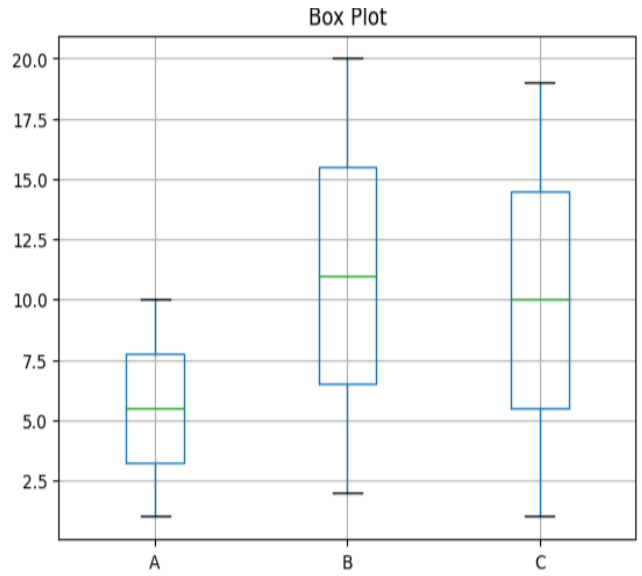
Scatter Plot: We use `scatter` function to create a scatter plot between columns 'A' and 'B'.

This `plt.title`, `plt.xlabel`, and `plt.ylabel` are used to set the titles and labels for the plots, and `plt.show` is used to display the plots.

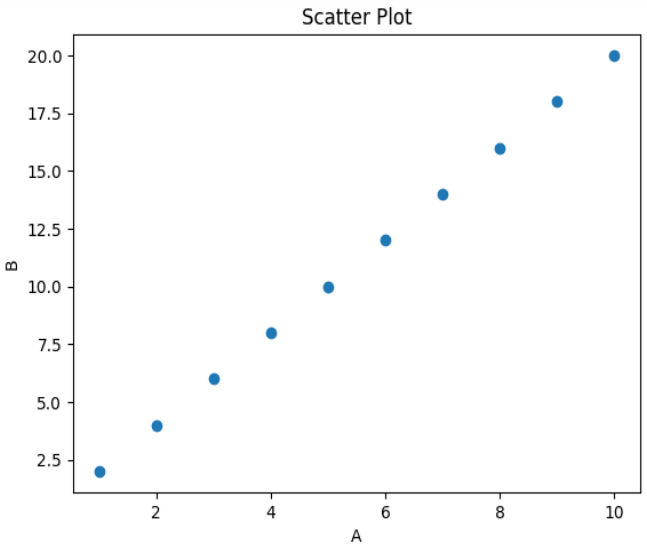
Program Code:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
# Sample data
data = pd.DataFrame({'A': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                     'B': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
                     'C': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]})
```

In [2]:
```
# Box plot
data.boxplot()
plt.title("Box Plot")
plt.show()
```



In [3]:
```
# Scatter plot
plt.scatter(data['A'], data['B'])
plt.title("Scatter Plot")
plt.xlabel("A")
plt.ylabel("B")
plt.show()
```



Result: Hence, The Data Distribution is done using Box plot and scatter plots for sample Data frame.

Problem No: 14b

Date:

Aim: To Find outliers using plot for sample Data Frame.

Requirements: Python **(**Jupyter notebook**)**

Procedure:

In this program, we have a sample data frame called `data` with columns 'A', 'B', and 'C' to demonstrate the various plots.

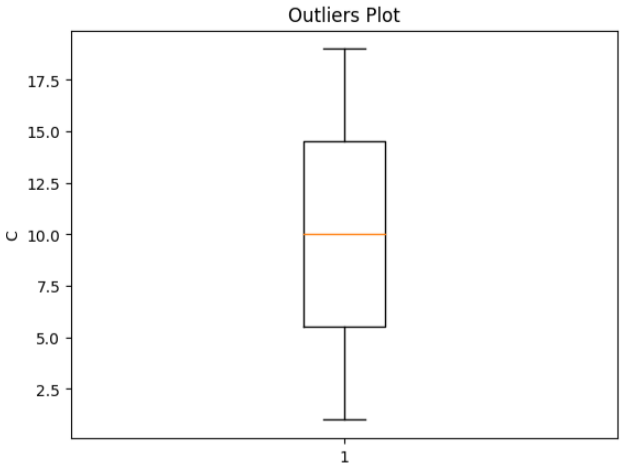
-Outliers Plot: We use `boxplot` function to create a box plot specifically for column 'C' to identify any outliers.

`plt.title`, `plt.xlabel`, and `plt.ylabel` are used to set the titles and labels for the plots, and `plt.show` is used to display the plots.

Program code:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
# Sample data
data = pd.DataFrame({'A': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                     'B': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
                     'C': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]})
```

In [8]:
```
# Outliers plot
plt.boxplot(data['C'])
plt.title("Outliers Plot")
plt.ylabel("C")
plt.show()
```



Result: Hence, outliers are found using plot for sample Data frame.

Problem No: 14c

Date:

Aim: To plot the Histogram, Barchart , Pie chart on sample data.

Requirements: Python (Jupyter notebook)

Procedure:

In this program, we have a sample data frame called `data` with columns 'A', 'B', and 'C' to demonstrate the various plots.

*Histogram: We use `hist` function to create a histogram for column 'A'.

*Bar Chart: We use `bar` function to create a bar chart between columns 'A' and 'B'.

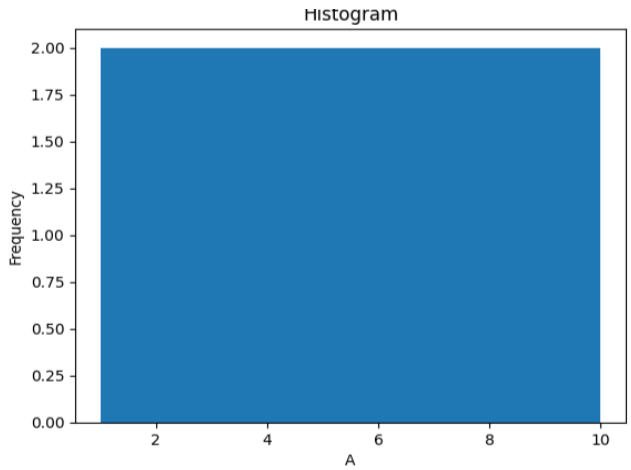
*Pie Chart: We use `pie` function to create a pie chart with custom labels and sizes.

`plt.title`, `plt.xlabel`, and `plt.ylabel` are used to set the titles and labels for the plots, and `plt.show` is used to display the plots.

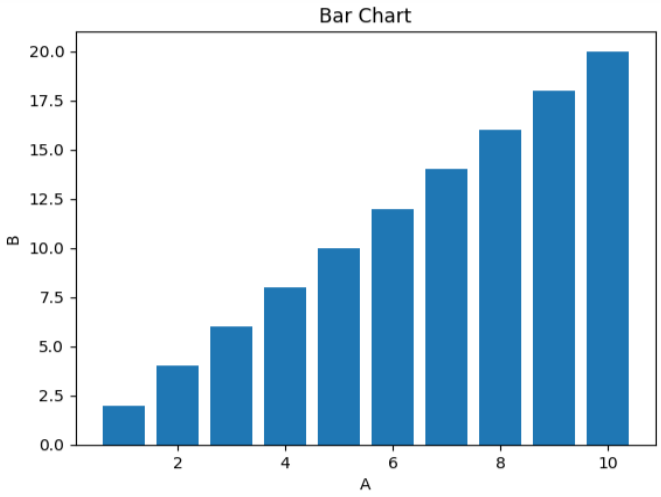
Program Code:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
# Sample data
data = pd.DataFrame({'A': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                     'B': [2, 4, 6, 8, 10, 12, 14, 16, 18, 20],
                     'C': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]})
```

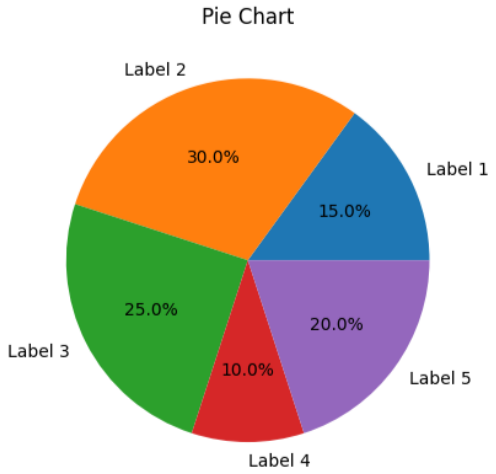
In [5]:
```
# Histogram
plt.hist(data['A'], bins=5)
plt.title("Histogram")
plt.xlabel("A")
plt.ylabel("Frequency")
plt.show()
```



In [6]:
```
# Bar chart
plt.bar(data['A'], data['B'])
plt.title("Bar Chart")
plt.xlabel("A")
plt.ylabel("B")
plt.show()
```



In [7]:
```
# Pie chart
labels = ['Label 1', 'Label 2', 'Label 3', 'Label 4', 'Label 5']
sizes = [15, 30, 25, 10, 20]
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.title("Pie Chart")
plt.show()
```



Result: Hence, The Histogram, Bar chart, Pie chart are drawn for sample Data Frame.

Problem No: 15

Date:

Aim: To visualize the Geographic Data with Basemap.

Requirements: Python (jupyter notebook)

Description:

This is a basic example of visualizing geographic data using Basemap. You can customize the map projection, add more map features, and plot various types of data on the map. Basemap provides a wide range of functionality for working with geographic data, including shapefiles, annotations, and advanced map projections.

Basemap is deprecated as of Matplotlib 3.3, and its development has been stopped. It is recommended to use Cartopy, a successor to Basemap, for creating maps with Matplotlib.

Procedure:

1. Firstly, Installation of basemap using pip should be done.
2. Then, to import necessary libraries required for basemap visualization.
3. Then after,Create a Basemap instance.Later, add the map features like drawing coastlines, filling continents with color, drawing country boundaries, drawing meridians and parallels, and adding map title.
4. Nextly, Plot the data points on map by converting the coordinates to map projection.
5. Finally, Display the map using plt.show () function.

Program Code:

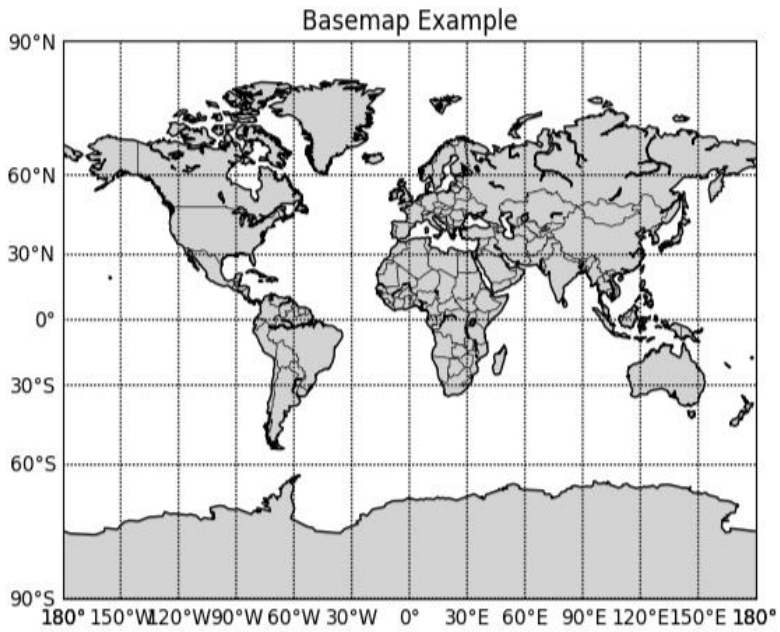
1. Basemap installation with pip command.

```
C:\Users\LENOVO>pip install basemap
Collecting basemap
  Downloading basemap-1.3.7-cp311-cp311-win_amd64.whl (486 kB)
     ---------------------------------------- 486.8/486.8 kB 116.9 kB/s eta 0:00:00
Collecting basemap-data<1.4,>=1.3.2 (from basemap)
  Downloading basemap_data-1.3.2-py2.py3-none-any.whl (30.5 MB)
     ---------------------------------------- 30.5/30.5 MB 376.0 kB/s eta 0:00:00
Collecting pyshp<2.4,>=1.2 (from basemap)
  Downloading pyshp-2.3.1-py2.py3-none-any.whl (46 kB)
     ---------------------------------------- 46.5/46.5 kB 165.7 kB/s eta 0:00:00
Requirement already satisfied: matplotlib<3.8,>=1.5 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from basemap) (3.7.1)
Collecting pyproj<3.6.0,>=1.9.3 (from basemap)
  Downloading pyproj-3.5.0-cp311-cp311-win_amd64.whl (5.1 MB)
     ---------------------------------------- 5.1/5.1 MB 356.4 kB/s eta 0:00:00
Requirement already satisfied: numpy<1.25,>=1.22 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from basemap) (1.24.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (1.0.7
)
Requirement already satisfied: cycler>=0.10 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (4.39
.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (1.4.
4)
Requirement already satisfied: packaging>=20.0 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (23.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (3.0.9
)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from matplotlib<3.8,>=1.5->basemap) (2
.8.2)
Requirement already satisfied: certifi in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from pyproj<3.6.0,>=1.9.3->basemap) (2022.12.7)
Requirement already satisfied: six>=1.5 in c:\users\lenovo\appdata\local\programs\python\python311\lib\site-packages (from python-dateutil>=2.7->matplotlib<3.8,>=1.5->b
asemap) (1.16.0)
Installing collected packages: pyshp, pyproj, basemap-data, basemap
Successfully installed basemap-1.3.7 basemap-data-1.3.2 pyproj-3.5.0 pyshp-2.3.1
```

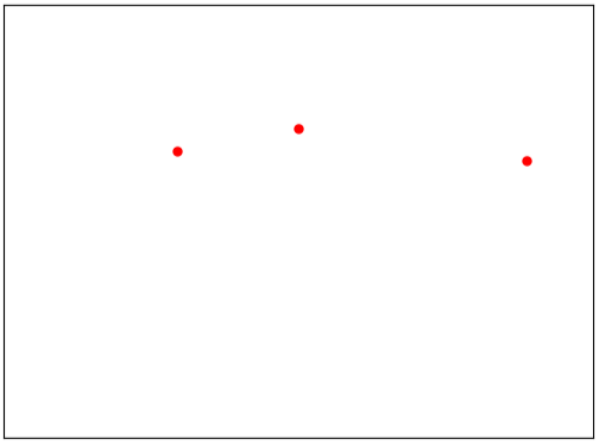
In [2]:
```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
# Define the map boundaries
lon_min, lon_max = -180, 180
lat_min, lat_max = -90, 90
# Create a Basemap instance
m = Basemap(llcrnrlon=lon_min,llcrnrlat=lat_min,urcrnrlon=lon_max,urcrnrlat=lat_max,projection='mill')
```

In [4]:
```
# Draw coastlines
m.drawcoastlines()
# Fill continents with color
m.fillcontinents(color='lightgray')
# Draw country boundaries
m.drawcountries()
# Draw meridians and parallels
m.drawmeridians(range(-180, 181, 30), labels=[0, 0, 0, 1])
m.drawparallels(range(-90, 91, 30), labels=[1, 0, 0, 0])
# Add title
plt.title("Basemap Example")
```

Out[4]:
```
Text(0.5, 1.0, 'Basemap Example')
```



In [8]:
```
# Coordinates of data points
lons = [-74.0059, -0.1276, 139.6917]
lats = [40.7128, 51.5074, 35.6895]
# Convert coordinates to map projection
x, y = m(lons, lats)
# Plot data points
m.plot(x, y, 'ro', markersize=5)
```



Result: Hence, The visualization of geographic data is done on Basemap.