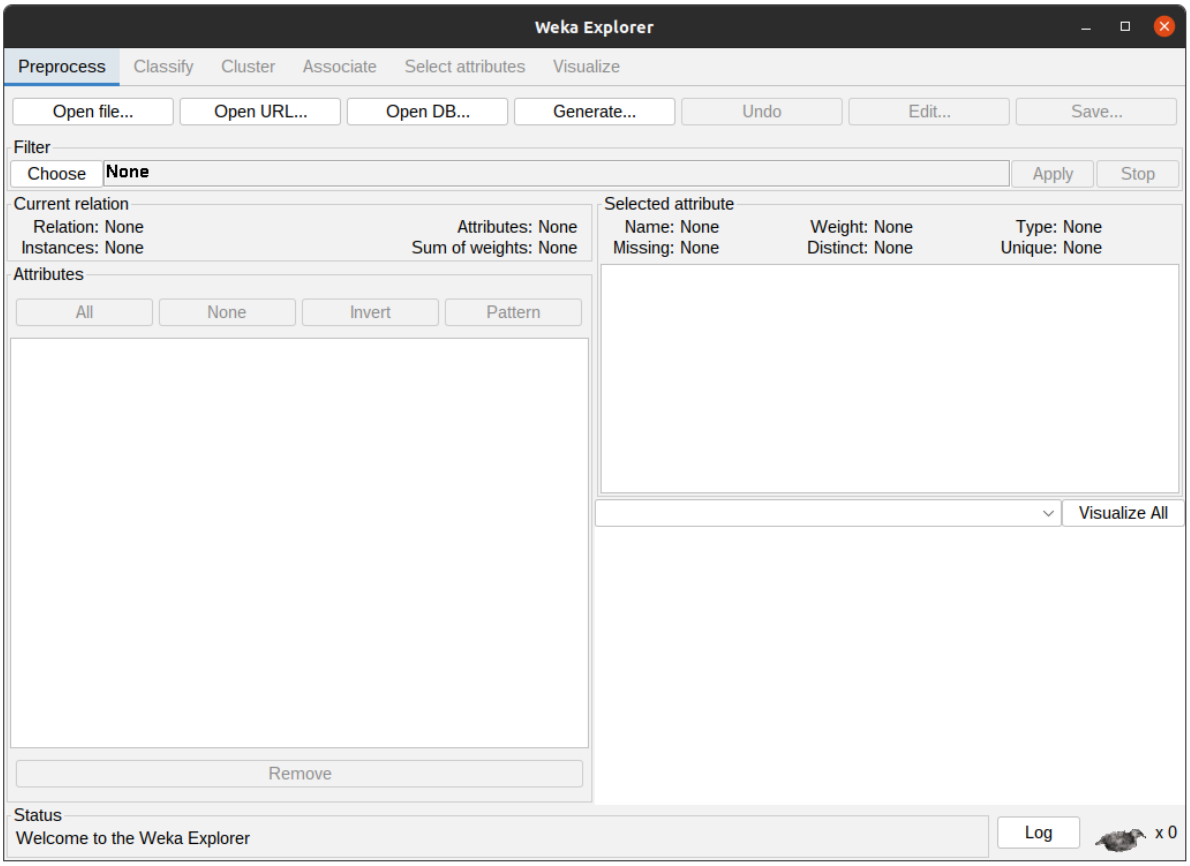
Experiment – 1

What is Weka?

WEKA - an open source software provides tools for data preprocessing, implementation of several Machine Learning algorithms, and visualisation tools so that you can develop machine learning techniques and apply them to real-world data mining problems. Weka is a collection of machine learning algorithms for data mining tasks. It contains tools for data preparation, classification, regression, clustering, association rules mining, and visualisation.

Purpose of weka

- We can start with the raw data collected from the field. This data may contain several null values and irrelevant fields. You use the data preprocessing tools provided in WEKA to cleanse the data.
- Then, you would save the preprocessed data in your local storage for applying ML algorithms.
- Next, depending on the kind of ML model that you are trying to develop you would select one of the options such as **Classify, Cluster**, or **Associate**. The **Attributes Selection** allows the automatic selection of features to create a reduced dataset.
- Note that under each category, WEKA provides the implementation of several algorithms. You would select an algorithm of your choice, set the desired parameters and run it on the dataset.
- Then, WEKA would give you the statistical output of the model processing. It provides you a visualisation tool to inspect the data.
- The various models can be applied on the same dataset. You can then compare the outputs of different models and select the best that meets your purpose.
- Thus, the use of WEKA results in a quicker development of machine learning models on the whole.



AIM: To write a program on processing of a dataset

DESCRIPTION:

Preprocessing

Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

Need of Data Preprocessing

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

- Getting the dataset
- Importing libraries
- Importing datasets
- Finding Missing Data
- Encoding Categorical Data
- Splitting dataset into training and test set
- Feature scaling

PROGRAM:

```
#Data Import

import pandas as pd

df = pd.read_csv("Automobile_data.csv")

#type of csv file

type(df)

pandas.core.frame.DataFrame

df

    index      company   body-style  wheel-base  length engine-type  \
0       0  alfa-romero  convertible        88.6   168.8        dohc
1       1  alfa-romero  convertible        88.6   168.8        dohc
2       2          NaN    hatchback        94.5   171.2        ohcv
3       3         audi        sedan        99.8   176.6         ohc
4       4         audi        sedan        99.4   176.6         ohc
..    ...          ...          ...         ...     ...         ...
56     81          NaN        sedan        97.3   171.7         ohc
57     82   volkswagen        sedan        97.3   171.7         ohc
58     86   volkswagen        sedan        97.3   171.7         ohc
59     87        volvo        sedan       104.3   188.8         ohc
60     88        volvo        wagon       104.3   188.8         ohc

   num-of-cylinders  horsepower  average-mileage    price
0              four         111               21  13495.0
1              four         111               21  16500.0
```

```
2           six      154          19  16500.0
3          four      102          24  13950.0
4          five      115          18  17450.0
..          ...      ...         ...      ...
56         four       85          27   7975.0
57         four       52          37   7995.0
58         four      100          26   9995.0
59         four      114          23  12940.0
60         four      114          23  13415.0

[61 rows x 10 columns]
```

df.shape

(61, 10)

Data Selection

df['price']

```
0     13495.0
1     16500.0
2     16500.0
3     13950.0
4     17450.0
       ...
56     7975.0
57     7995.0
58     9995.0
59    12940.0
60    13415.0
Name: price, Length: 61, dtype: float64
```

df.columns

```
Index(['index', 'company', 'body-style', 'wheel-base', 'length', 'engine-type',
       'num-of-cylinders', 'horsepower', 'average-mileage', 'price'],
      dtype='object')
```

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61 entries, 0 to 60
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   index             61 non-null     int64
 1   company           57 non-null     object
 2   body-style        61 non-null     object
 3   wheel-base        58 non-null     float64
 4   length            59 non-null     float64
 5   engine-type       61 non-null     object
 6   num-of-cylinders  60 non-null     object
```

```
 7   horsepower       61 non-null     int64
 8   average-mileage   61 non-null     int64
 9   price             58 non-null     float64
dtypes: float64(3), int64(3), object(4)
memory usage: 4.9+ KB
```

```
df.loc[3]
```

```
index                    3
company               audi
body-style           sedan
wheel-base            99.8
length               176.6
engine-type            ohc
num-of-cylinders      four
horsepower             102
average-mileage         24
price              13950.0
Name: 3, dtype: object
```

```
df.loc[:8]
```

```
   index      company   body-style  wheel-base  length engine-type  \
0      0  alfa-romero  convertible        88.6   168.8        dohc
1      1  alfa-romero  convertible        88.6   168.8        dohc
2      2          NaN    hatchback        94.5   171.2        ohcv
3      3         audi        sedan        99.8   176.6         ohc
4      4         audi        sedan        99.4   176.6         ohc
5      5         audi        sedan        99.8   177.3         ohc
6      6         audi        wagon       105.8   192.7         ohc
7      9          NaN        sedan       101.2   176.8         ohc
8     10          bmw        sedan         NaN   176.8         ohc

  num-of-cylinders  horsepower  average-mileage    price
0             four         111               21  13495.0
1             four         111               21  16500.0
2              six         154               19  16500.0
3             four         102               24  13950.0
4             five         115               18  17450.0
5             five         110               19  15250.0
6             five         110               19  18920.0
7             four         101               23  16430.0
8             four         101               23  16925.0
```

```
df.head()
```

```
   index      company   body-style  wheel-base  length engine-type  \
0      0  alfa-romero  convertible        88.6   168.8        dohc
1      1  alfa-romero  convertible        88.6   168.8        dohc
2      2          NaN    hatchback        94.5   171.2        ohcv
3      3         audi        sedan        99.8   176.6         ohc
4      4         audi        sedan        99.4   176.6         ohc
```

```
  num-of-cylinders  horsepower  average-mileage    price
0             four         111               21  13495.0
1             four         111               21  16500.0
2              six         154               19  16500.0
3             four         102               24  13950.0
4             five         115               18  17450.0
```

df.tail()

```
    index     company body-style  wheel-base  length engine-type  \
56     81         NaN      sedan        97.3   171.7         ohc
57     82  volkswagen      sedan        97.3   171.7         ohc
58     86  volkswagen      sedan        97.3   171.7         ohc
59     87       volvo      sedan       104.3   188.8         ohc
60     88       volvo      wagon       104.3   188.8         ohc

   num-of-cylinders  horsepower  average-mileage    price
56             four          85               27   7975.0
57             four          52               37   7995.0
58             four         100               26   9995.0
59             four         114               23  12940.0
60             four         114               23  13415.0
```

Clean data and update the CSV file

df.isnull()

```
    index  company  body-style  wheel-base  length  engine-type  \
0   False    False       False       False   False        False
1   False    False       False       False   False        False
2   False     True       False       False   False        False
3   False    False       False       False   False        False
4   False    False       False       False   False        False
..    ...      ...         ...         ...     ...          ...
56  False     True       False       False   False        False
57  False    False       False       False   False        False
58  False    False       False       False   False        False
59  False    False       False       False   False        False
60  False    False       False       False   False        False

    num-of-cylinders  horsepower  average-mileage  price
0              False       False            False  False
1              False       False            False  False
2              False       False            False  False
3              False       False            False  False
4              False       False            False  False
..               ...         ...              ...    ...
56             False       False            False  False
57             False       False            False  False
58             False       False            False  False
59             False       False            False  False
```

```
60            False      False           False  False

[61 rows x 10 columns]
```

df.isnull()

```
   index  company  body-style  wheel-base  length  engine-type  \
0   False    False       False       False   False        False
1   False    False       False       False   False        False
2   False     True       False       False   False        False
3   False    False       False       False   False        False
4   False    False       False       False   False        False
..    ...      ...         ...         ...     ...          ...
56  False     True       False       False   False        False
57  False    False       False       False   False        False
58  False    False       False       False   False        False
59  False    False       False       False   False        False
60  False    False       False       False   False        False

   num-of-cylinders  horsepower  average-mileage  price
0             False       False            False  False
1             False       False            False  False
2             False       False            False  False
3             False       False            False  False
4             False       False            False  False
..              ...         ...              ...    ...
56            False       False            False  False
57            False       False            False  False
58            False       False            False  False
59            False       False            False  False
60            False       False            False  False

[61 rows x 10 columns]
```

df.isnull().any()

```
index               False
company              True
body-style          False
wheel-base           True
length               True
engine-type         False
num-of-cylinders     True
horsepower          False
average-mileage     False
price                True
dtype: bool
```

df.isnull().values

```
array([[False, False, False, False, False, False, False, False, False,
        False],
```

```
[False, False, False, False, False, False, False, False, False,
 False],
[False,  True, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False,  True, False, False, False, False, False, False, False,
 False],
[False, False, False,  True, False, False, False, False, False,
 False],
[False, False, False,  True, False, False, False, False, False,
 False],
[False, False, False,  True, False, False, False, False, False,
 False],
[False, False, False, False,  True, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False,  True, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False,  True, False, False,
  True],
[False, False, False, False, False, False, False, False, False,
  True],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
```

```
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False,  True, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
  True],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
[False, False, False, False, False, False, False, False, False,
 False],
```

```
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False,  True, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False],
       [False, False, False, False, False, False, False, False, False,
        False]])
```

```
df.isnull().values.any()
```

```
True
```

```
#gives the missing values of all columns
df.isnull().sum()
```

```
index              0
company            4
body-style         0
wheel-base         3
length             2
engine-type        0
num-of-cylinders   1
horsepower         0
average-mileage    0
price              3
dtype: int64
```

```
df.head()
```

```
   index      company   body-style  wheel-base  length engine-type  \
0      0  alfa-romero  convertible        88.6   168.8        dohc
1      1  alfa-romero  convertible        88.6   168.8        dohc
2      2          NaN    hatchback        94.5   171.2        ohcv
3      3         audi        sedan        99.8   176.6         ohc
4      4         audi        sedan        99.4   176.6         ohc

  num-of-cylinders  horsepower  average-mileage    price
0             four         111               21  13495.0
1             four         111               21  16500.0
```

```
2            six       154              19  16500.0
3           four       102              24  13950.0
4           five       115              18  17450.0
```

```
df['price'].fillna(df['price'].mean(),inplace=True)
df["num-of-cylinders"].value_counts()
```

```
four      38
six       11
five       5
eight      3
three      1
twelve     1
two        1
Name: num-of-cylinders, dtype: int64
```

```
df.isnull().sum()
```

```
index               0
company             4
body-style          0
wheel-base          3
length              2
engine-type         0
num-of-cylinders    1
horsepower          0
average-mileage     0
price               0
dtype: int64
```

```
df['num-of-cylinders'].fillna(value="six",inplace=True)
```

```
df["num-of-cylinders"].value_counts()
#df.isnull().sum()
```

```
four      38
six       12
five       5
eight      3
three      1
twelve     1
two        1
Name: num-of-cylinders, dtype: int64
```

```
df.isnull().sum()
```

```
index               0
company             4
body-style          0
wheel-base          3
length              2
engine-type         0
num-of-cylinders    0
```

```
horsepower          0
average-mileage     0
price               0
dtype: int64
```

df.isnull().sum()

```
index               0
company             4
body-style          0
wheel-base          3
length              2
engine-type         0
num-of-cylinders    0
horsepower          0
average-mileage     0
price               0
dtype: int64
```

df['length'].fillna(df['length'].mean(),inplace=True)

df.isnull().sum()

```
index               0
company             4
body-style          0
wheel-base          3
length              0
engine-type         0
num-of-cylinders    0
horsepower          0
average-mileage     0
price               0
dtype: int64
```

df.to_csv("Auto.csv")

df.head()

```
   index      company   body-style  wheel-base  length engine-type  \
0      0  alfa-romero  convertible        88.6   168.8        dohc
1      1  alfa-romero  convertible        88.6   168.8        dohc
2      2          NaN    hatchback        94.5   171.2        ohcv
3      3         audi        sedan        99.8   176.6         ohc
4      4         audi        sedan        99.4   176.6         ohc

  num-of-cylinders  horsepower  average-mileage    price
0             four         111               21  13495.0
1             four         111               21  16500.0
2              six         154               19  16500.0
3             four         102               24  13950.0
4             five         115               18  17450.0
```

```
df1=df[['company','price']][df.price==df['price'].max()]
print(df1)
```

```
         company    price
35  mercedes-benz  45400.0
```

```
dft=df.groupby('body-style')
dft
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000001D98597F4F0>
```

```
dtyot=dft.get_group('sedan')

print(dtyot)
```

```
    index        company body-style  wheel-base      length engine-type  \
3       3           audi      sedan        99.8  176.600000         ohc
4       4           audi      sedan        99.4  176.600000         ohc
5       5           audi      sedan        99.8  177.300000         ohc
7       9            NaN      sedan       101.2  176.800000         ohc
8      10            bmw      sedan         NaN  176.800000         ohc
9      11            bmw      sedan         NaN  176.800000         ohc
10     13            bmw      sedan         NaN  189.000000         ohc
11     14            bmw      sedan       103.5  173.015254         ohc
12     15            bmw      sedan       110.0  197.000000         ohc
15     18      chevrolet      sedan        94.5  158.800000         ohc
19     28          honda      sedan        96.5  175.400000         ohc
20     29          honda      sedan        96.5  169.100000         ohc
21     30          isuzu      sedan        94.3  170.700000         ohc
22     31          isuzu      sedan        94.5  155.900000         ohc
23     32          isuzu      sedan        94.5  155.900000         ohc
24     33         jaguar      sedan       113.0  199.600000        dohc
25     34         jaguar      sedan       113.0  199.600000        dohc
26     35         jaguar      sedan       102.0  191.700000        ohcv
31     43          mazda      sedan       104.9  175.000000         ohc
32     44  mercedes-benz      sedan       110.0  190.900000         ohc
34     46  mercedes-benz      sedan       120.9  208.100000        ohcv
38     51     mitsubishi      sedan        96.3  172.400000         ohc
39     52            NaN      sedan        96.3  172.400000         ohc
40     53         nissan      sedan        94.5  165.300000         ohc
41     54         nissan      sedan        94.5  165.300000         ohc
42     55         nissan      sedan        94.5  165.300000         ohc
44     57         nissan      sedan       100.4  184.600000        ohcv
55     80     volkswagen      sedan        97.3  171.700000         ohc
56     81            NaN      sedan        97.3  171.700000         ohc
57     82     volkswagen      sedan        97.3  171.700000         ohc
58     86     volkswagen      sedan        97.3  171.700000         ohc
59     87          volvo      sedan       104.3  188.800000         ohc

   num-of-cylinders  horsepower  average-mileage    price
3              four         102               24  13950.0
4              five         115               18  17450.0
```

```
5           five       110             19  15250.0
7           four       101             23  16430.0
8           four       101             23  16925.0
9           six        121             21  20970.0
10          six        182             16  30760.0
11          six        182             16  41315.0
12          six        182             15  36880.0
15          four        70             38   6575.0
19          four       101             24  12945.0
20          four       100             25  10345.0
21          four        78             24   6785.0
22          six         70             38  15387.0
23          four        70             38  15387.0
24          six        176             15  32250.0
25          six        176             15  35550.0
26        twelve       262             13  36000.0
31          four        72             31  18344.0
32          five       123             22  25552.0
34         eight       184             14  40960.0
38          four        88             25   6989.0
39          four        88             25   8189.0
40          four        55             45   7099.0
41          four        69             31   6649.0
42          four        69             31   6849.0
44          six        152             19  13499.0
55          four        52             37   7775.0
56          four        85             27   7975.0
57          four        52             37   7995.0
58          four       100             26   9995.0
59          four       114             23  12940.0
```

```
sdf=pd.read_csv('Auto.csv')
sdf=sdf.sort_values(by=['price'],ascending=False)

sdf.head()
```

```
    Unnamed: 0  index        company   body-style  wheel-base      length  \
35          35     47  mercedes-benz      hardtop       112.0  199.200000
11          11     14            bmw        sedan       103.5  173.015254
34          34     46  mercedes-benz        sedan       120.9  208.100000
46          46     62        porsche  convertible        89.5  168.900000
12          12     15            bmw        sedan       110.0  197.000000

   engine-type num-of-cylinders  horsepower  average-mileage    price
35        ohcv            eight         184               14  45400.0
11         ohc              six         182               16  41315.0
34        ohcv            eight         184               14  40960.0
46        ohcf              six         207               17  37028.0
12         ohc              six         182               15  36880.0
```

```
df.describe()
```

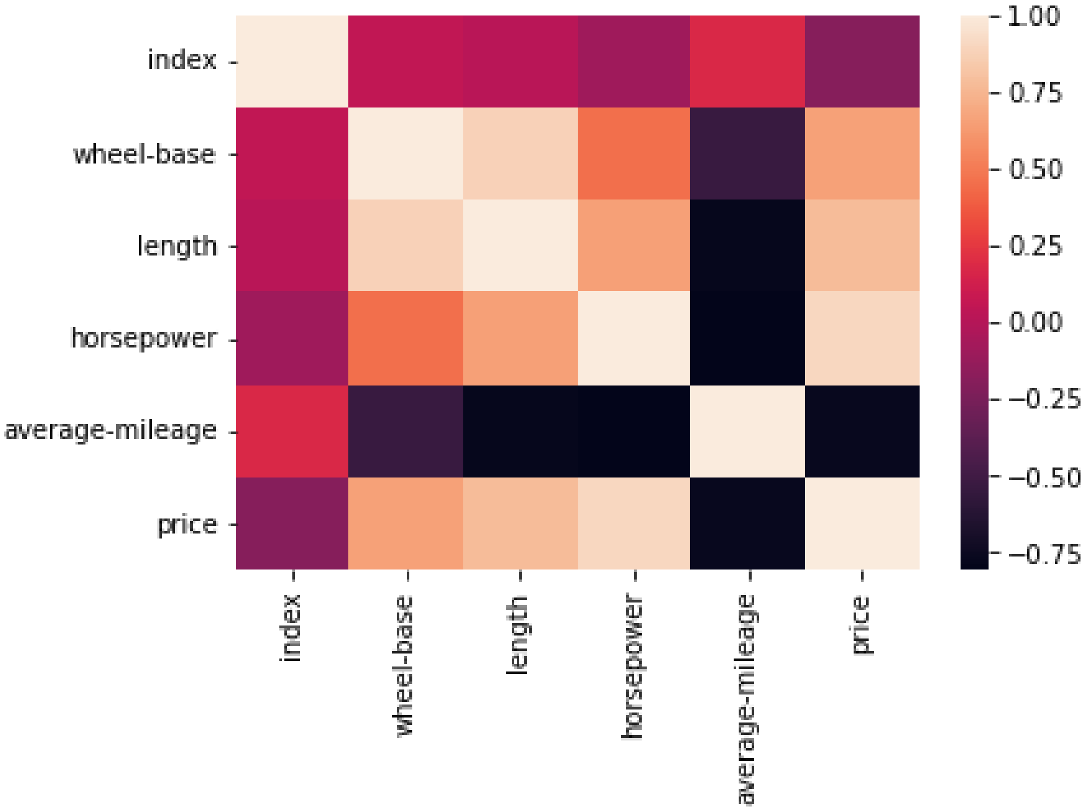
```
       index  wheel-base      length  horsepower  average-mileage  \
count  61.000000   58.000000   61.000000   61.000000        61.000000
mean   40.885246   98.301724  173.015254  107.852459        25.803279
std    25.429706    6.798981   13.612602   53.524398         8.129821
min     0.000000   88.400000  141.100000   48.000000        13.000000
25%    18.000000   94.500000  165.300000   68.000000        19.000000
50%    39.000000   96.000000  171.700000  100.000000        25.000000
75%    61.000000  101.000000  176.800000  123.000000        31.000000
max    88.000000  120.900000  208.100000  288.000000        47.000000

             price
count     61.00000
mean   15387.00000
std    11033.62446
min     5151.00000
25%     6849.00000
50%    12940.00000
75%    17450.00000
max    45400.00000
```

df.corr()

```
                    index  wheel-base    length  horsepower  average-mileage  \
index            1.000000    0.046964  0.015570   -0.093809         0.176037
wheel-base       0.046964    1.000000  0.879011    0.455915        -0.538118
length           0.015570    0.879011  1.000000    0.652709        -0.781407
horsepower      -0.093809    0.455915  0.652709    1.000000        -0.808804
average-mileage  0.176037   -0.538118 -0.781407   -0.808804         1.000000
price           -0.197470    0.658375  0.778277    0.901707        -0.770217

                    price
index           -0.197470
wheel-base       0.658375
length           0.778277
horsepower       0.901707
average-mileage -0.770217
price            1.000000
```

```
import seaborn as sns
sns.heatmap(df.corr())
```

<AxesSubplot:>




```
from sklearn.preprocessing import LabelEncoder
Labelencoder_X = LabelEncoder()

X=df.iloc[:,:].values
print(X[:,1])
print(type(X[:,1]))
X[:,1]=Labelencoder_X.fit_transform(X[:,1])
```

```
['alfa-romero' 'alfa-romero' nan 'audi' 'audi' 'audi' 'audi' nan 'bmw'
 'bmw' 'bmw' 'bmw' 'bmw' 'chevrolet' 'chevrolet' 'chevrolet' 'dodge'
 'dodge' 'honda' 'honda' 'honda' 'isuzu' 'isuzu' 'isuzu' 'jaguar' 'jaguar'
 'jaguar' 'mazda' 'mazda' 'mazda' 'mazda' 'mazda' 'mercedes-benz'
 'mercedes-benz' 'mercedes-benz' 'mercedes-benz' 'mitsubishi' 'mitsubishi'
 'mitsubishi' nan 'nissan' 'nissan' 'nissan' 'nissan' 'nissan' 'porsche'
 'porsche' 'porsche' 'toyota' 'toyota' 'toyota' 'toyota' 'toyota' 'toyota'
 'toyota' 'volkswagen' nan 'volkswagen' 'volkswagen' 'volvo' 'volvo']
<class 'numpy.ndarray'>
```

```
print(X)
print(X[:,1])
```

```
[[0 0 'convertible' 88.6 168.8 'dohc' 'four' 111 21 13495.0]
 [1 0 'convertible' 88.6 168.8 'dohc' 'four' 111 21 16500.0]
 [2 16 'hatchback' 94.5 171.2 'ohcv' 'six' 154 19 16500.0]
 [3 1 'sedan' 99.8 176.6 'ohc' 'four' 102 24 13950.0]
 [4 1 'sedan' 99.4 176.6 'ohc' 'five' 115 18 17450.0]
 [5 1 'sedan' 99.8 177.3 'ohc' 'five' 110 19 15250.0]
 [6 1 'wagon' 105.8 192.7 'ohc' 'five' 110 19 18920.0]
```

```
[9 16 'sedan' 101.2 176.8 'ohc' 'four' 101 23 16430.0]
[10 2 'sedan' nan 176.8 'ohc' 'four' 101 23 16925.0]
[11 2 'sedan' nan 176.8 'ohc' 'six' 121 21 20970.0]
[13 2 'sedan' nan 189.0 'ohc' 'six' 182 16 30760.0]
[14 2 'sedan' 103.5 nan 'ohc' 'six' 182 16 41315.0]
[15 2 'sedan' 110.0 197.0 'ohc' 'six' 182 15 36880.0]
[16 3 'hatchback' 88.4 141.1 'l' 'three' 48 47 5151.0]
[17 3 'hatchback' 94.5 155.9 'ohc' 'four' 70 38 6295.0]
[18 3 'sedan' 94.5 158.8 'ohc' 'four' 70 38 6575.0]
[19 4 'hatchback' 93.7 nan 'ohc' 'four' 68 31 6377.0]
[20 4 'hatchback' 93.7 157.3 'ohc' 'four' 68 31 6229.0]
[27 5 'wagon' 96.5 157.1 'ohc' 'four' 76 30 7295.0]
[28 5 'sedan' 96.5 175.4 'ohc' 'four' 101 24 12945.0]
[29 5 'sedan' 96.5 169.1 'ohc' 'four' 100 25 10345.0]
[30 6 'sedan' 94.3 170.7 'ohc' 'four' 78 24 6785.0]
[31 6 'sedan' 94.5 155.9 'ohc' nan 70 38 nan]
[32 6 'sedan' 94.5 155.9 'ohc' 'four' 70 38 nan]
[33 7 'sedan' 113.0 199.6 'dohc' 'six' 176 15 32250.0]
[34 7 'sedan' 113.0 199.6 'dohc' 'six' 176 15 35550.0]
[35 7 'sedan' 102.0 191.7 'ohcv' 'twelve' 262 13 36000.0]
[36 8 'hatchback' 93.1 159.1 'ohc' 'four' 68 30 5195.0]
[37 8 'hatchback' 93.1 159.1 'ohc' 'four' 68 31 6095.0]
[38 8 'hatchback' 93.1 159.1 'ohc' 'four' 68 31 6795.0]
[39 8 'hatchback' 95.3 169.0 'rotor' 'two' 101 17 11845.0]
[43 8 'sedan' 104.9 175.0 'ohc' 'four' 72 31 18344.0]
[44 9 'sedan' 110.0 190.9 'ohc' 'five' 123 22 25552.0]
[45 9 'wagon' 110.0 190.9 'ohc' 'five' 123 22 28248.0]
[46 9 'sedan' 120.9 208.1 'ohcv' 'eight' 184 14 40960.0]
[47 9 'hardtop' 112.0 199.2 'ohcv' 'eight' 184 14 45400.0]
[49 10 'hatchback' 93.7 157.3 'ohc' 'four' 68 37 5389.0]
[50 10 'hatchback' 93.7 157.3 'ohc' 'four' 68 31 6189.0]
[51 10 'sedan' 96.3 172.4 'ohc' 'four' 88 25 6989.0]
[52 16 'sedan' 96.3 172.4 'ohc' 'four' 88 25 8189.0]
[53 11 'sedan' 94.5 165.3 'ohc' 'four' 55 45 7099.0]
[54 11 'sedan' 94.5 165.3 'ohc' 'four' 69 31 6649.0]
[55 11 'sedan' 94.5 165.3 'ohc' 'four' 69 31 6849.0]
[56 11 'wagon' 94.5 170.2 'ohc' 'four' 69 31 7349.0]
[57 11 'sedan' 100.4 184.6 'ohcv' 'six' 152 19 13499.0]
[61 12 'hardtop' 89.5 168.9 'ohcf' 'six' 207 17 34028.0]
[62 12 'convertible' 89.5 168.9 'ohcf' 'six' 207 17 37028.0]
[63 12 'hatchback' 98.4 175.7 'dohcv' 'eight' 288 17 nan]
[66 13 'hatchback' 95.7 158.7 'ohc' 'four' 62 35 5348.0]
[67 13 'hatchback' 95.7 158.7 'ohc' 'four' 62 31 6338.0]
[68 13 'hatchback' 95.7 158.7 'ohc' 'four' 62 31 6488.0]
[69 13 'wagon' 95.7 169.7 'ohc' 'four' 62 31 6918.0]
[70 13 'wagon' 95.7 169.7 'ohc' 'four' 62 27 7898.0]
[71 13 'wagon' 95.7 169.7 'ohc' 'four' 62 27 8778.0]
[79 13 'wagon' 104.5 187.8 'dohc' 'six' 156 19 15750.0]
[80 14 'sedan' 97.3 171.7 'ohc' 'four' 52 37 7775.0]
[81 16 'sedan' 97.3 171.7 'ohc' 'four' 85 27 7975.0]
```

```
 [82 14 'sedan' 97.3 171.7 'ohc' 'four' 52 37 7995.0]
 [86 14 'sedan' 97.3 171.7 'ohc' 'four' 100 26 9995.0]
 [87 15 'sedan' 104.3 188.8 'ohc' 'four' 114 23 12940.0]
 [88 15 'wagon' 104.3 188.8 'ohc' 'four' 114 23 13415.0]]
[0 0 16 1 1 1 1 16 2 2 2 2 2 3 3 3 4 4 5 5 5 6 6 6 7 7 7 8 8 8 8 8 9 9 9 9
 10 10 10 16 11 11 11 11 11 12 12 12 13 13 13 13 13 13 13 14 16 14 14 15
 15]
```

```
en=LabelEncoder()
```

```
en.fit_transform(["Delhi","Bom","Hyd","Ama"])
```

```
array([2, 1, 3, 0], dtype=int64)
```

```
dum=pd.get_dummies(X[:,2])
```

```
print(dum)
```

```
   convertible  hardtop  hatchback  sedan  wagon
0            1        0          0      0      0
1            1        0          0      0      0
2            0        0          1      0      0
3            0        0          0      1      0
4            0        0          0      1      0
..         ...      ...        ...    ...    ...
56           0        0          0      1      0
57           0        0          0      1      0
58           0        0          0      1      0
59           0        0          0      1      0
60           0        0          0      0      1

[61 rows x 5 columns]
```

```
df.columns
```

```
Index(['index', 'company', 'body-style', 'wheel-base', 'length', 'engine-type',
       'num-of-cylinders', 'horsepower', 'average-mileage', 'price'],
      dtype='object')
```

```
df['body-style'].value_counts()
```

```
sedan          32
hatchback      15
wagon           9
convertible     3
hardtop         2
Name: body-style, dtype: int64
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
data={'price':[492,282,487,519,514]}
p=pd.DataFrame(data)
mnorm=MinMaxScaler()
```

```
scaled=mnorm.fit_transform(p)
pf=pd.DataFrame(scaled)
pf
```

	0
0	0.886076
1	0.000000
2	0.864979
3	1.000000
4	0.978903

Experiment – 2

AIM: To write a program on Linear Regression.

DESCRIPTION:
Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable. It fits a straight line or surface that minimizes the discrepancies between predicted and actual output values

PROGRAM:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
```

```
data = pd.read_csv("C:/Users/91995/ML Lab/Lab-2/Salary_Data.csv").values
data
```

```
array([[1.10000e+00, 3.93430e+04],
       [1.30000e+00, 4.62050e+04],
       [1.50000e+00, 3.77310e+04],
       [2.00000e+00, 4.35250e+04],
       [2.20000e+00, 3.98910e+04],
       [2.90000e+00, 5.66420e+04],
       [3.00000e+00, 6.01500e+04],
       [3.20000e+00, 5.44450e+04],
       [3.20000e+00, 6.44450e+04],
       [3.70000e+00, 5.71890e+04],
       [3.90000e+00, 6.32180e+04],
       [4.00000e+00, 5.57940e+04],
       [4.00000e+00, 5.69570e+04],
       [4.10000e+00, 5.70810e+04],
       [4.50000e+00, 6.11110e+04],
       [4.90000e+00, 6.79380e+04],
       [5.10000e+00, 6.60290e+04],
       [5.30000e+00, 8.30880e+04],
       [5.90000e+00, 8.13630e+04],
       [6.00000e+00, 9.39400e+04],
       [6.80000e+00, 9.17380e+04],
       [7.10000e+00, 9.82730e+04],
       [7.90000e+00, 1.01302e+05],
       [8.20000e+00, 1.13812e+05],
       [8.70000e+00, 1.09431e+05],
       [9.00000e+00, 1.05582e+05],
       [9.50000e+00, 1.16969e+05],
```

```
       [9.60000e+00, 1.12635e+05],
       [1.03000e+01, 1.22391e+05],
       [1.05000e+01, 1.21872e+05]])
```

```
x = data[:, 0].reshape(-1, 1)
x.shape
```

```
(30, 1)
```

```
y = data[:, 1]
y.shape
```

```
(30,)
```

```
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(x, y)
```

```
from sklearn.linear_model import LinearRegression

alg = LinearRegression()
alg.fit(xtrain, ytrain)
```

```
LinearRegression()
```

```
ypred = alg.predict(xtest)
ypred
```

```
array([115562.43269133, 110899.11672721,  68929.27305015,  47478.01961521
,
        74525.25220709,  93178.51606356, 100639.82160615,  76390.57859274
])
```

```
alg.score(xtrain, ytrain)
```

```
0.95954093674679
```

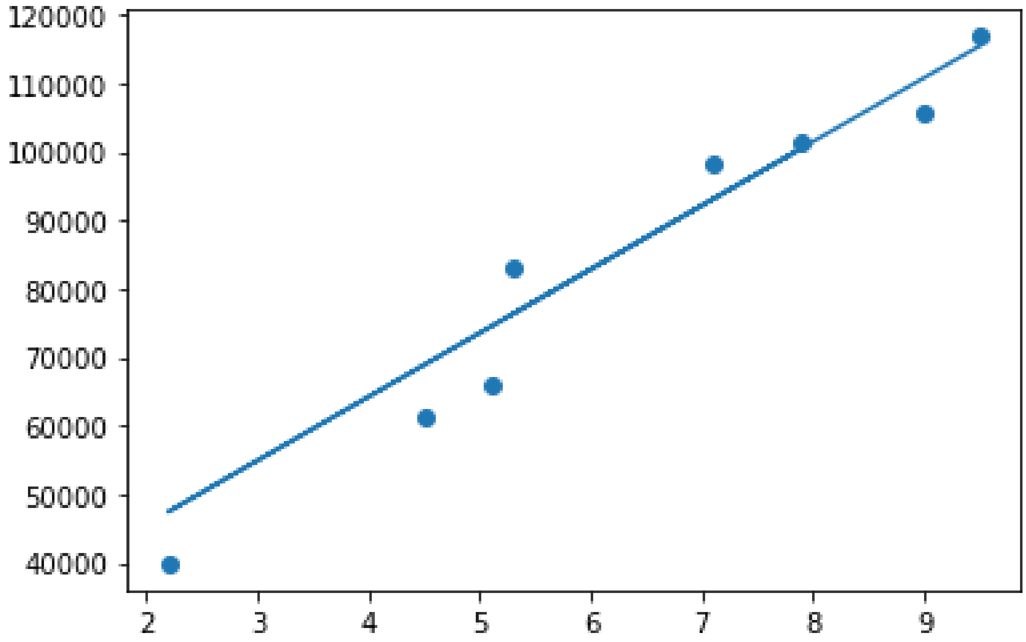
```
alg.score(xtest, ytest)
```

```
0.939705920726265
```

```
from sklearn.metrics import mean_squared_error

mean_squared_error(ytest, ypred, squared=False)
```

```
6045.374511658176
```

```
plt.scatter(xtest, ytest)
plt.plot(xtest, ypred)
plt.show()
```


120000
110000
100000
90000
80000
70000
60000
50000
40000
2
3
4
5
6
7
8
9

Experiment – 3

AIM: To write a program to find the classification metrics.

DESCRIPTION:
Classification is a process of categorizing a given set of data into classes, It can be performed on both structured or unstructured data. The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories.

classification metric is a number that measures the performance that your machine learning model when it comes to assigning observations to certain classes

PROGRAM:

```
from sklearn import datasets
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()

x_train, x_test, y_train, y_test = train_test_split(iris.data, iris.targe
t, random_state=1)

clf = LogisticRegression()
clf.fit(x_train, y_train)

C:\Users\91995\anaconda3\lib\site-packages\sklearn\linear_model\_logistic
.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression
  n_iter_i = _check_optimize_result(

LogisticRegression()

y_train_pred = clf.predict(x_train)

y_test_pred = clf.predict(x_test)

from sklearn.metrics import confusion_matrix

confusion_matrix(y_train, y_train_pred)

array([[37,  0,  0],
       [ 0, 32,  2],
       [ 0,  0, 41]], dtype=int64)
```

```
confusion_matrix(y_test, y_test_pred)
```

```
array([[13,  0,  0],
       [ 0, 15,  1],
       [ 0,  0,  9]], dtype=int64)
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

acc = accuracy_score(y_test, y_test_pred)
print(‘Accuracy:’,acc)
recall = recall_score(y_test, y_test_pred, average='weighted')
print(‘Recall:’,recall)
f1 = f1_score(y_test, y_test_pred, average='weighted')
print(‘f1 score:’,f1)
```

```
Accuracy: 0.9736842105263158
Recall: 0.9736842105263158
f1 score: 0.9739522830846216
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       1.00      0.94      0.97        16
           2       0.90      1.00      0.95         9

    accuracy                           0.97        38
   macro avg       0.97      0.98      0.97        38
weighted avg       0.98      0.97      0.97        38
```

Experiment– 4

AIM: To write a program on types of linear regression

DESCRIPTION:

Types of Linear Regression:

1. Simple Linear Regression
2. Multiple Linear Regression

In Simple Linear Regression, we try to find the relationship between a single independent variable (input) and a corresponding dependent variable (output)**.** This can be expressed in the form of a straight line.

The same equation of a line can be re-written as:

$Y = \beta_0 + \beta_1 X$

In Multiple Linear Regression, we try to find the relationship between 2 or more independent variables (inputs) and the corresponding dependent variable (output). The independent variables can be continuous or categorical.

The equation that describes how the predicted values of y is related to p independent variables is called as Multiple Linear Regression equation :

$Y= \beta_0 + \beta_1 x1 + \beta_2 x2+ \ldots\ldots. B_p xp$

PROGRAM:

1.Simple Linear Regression

```
[7]: import pandas as pd

df = pd.read_csv('/home/student/176/weight-height.csv')
df
```

[7]:

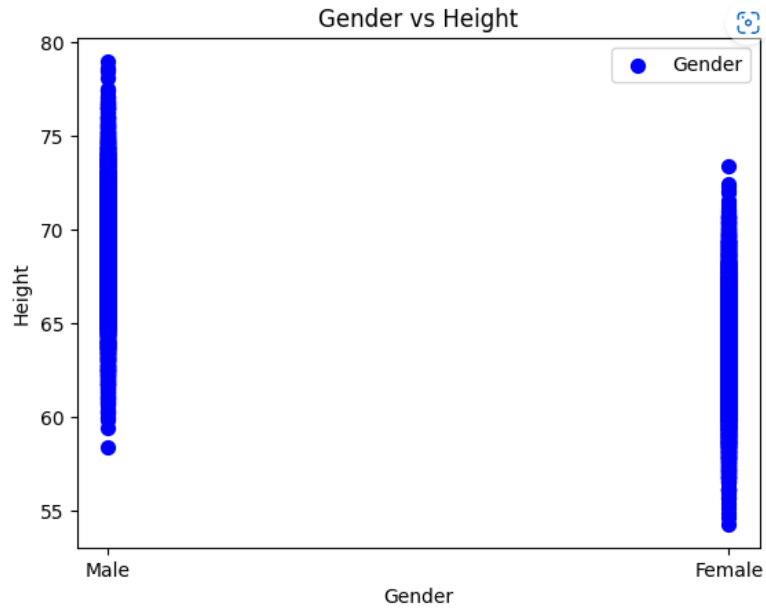
	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801
...	...	...	...
9995	Female	66.172652	136.777454
9996	Female	67.067155	170.867906
9997	Female	63.867992	128.475319
9998	Female	69.034243	163.852461
9999	Female	61.944246	113.649103

10000 rows × 3 columns

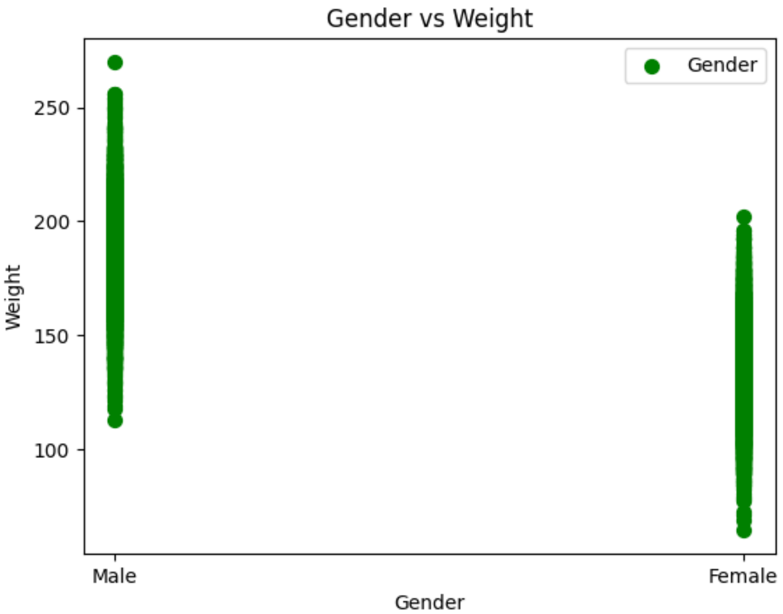
[9]:
```
data = df.values
data
```

t[9]:
```
array([['Male', 73.847017017515, 241.893563180437],
       ['Male', 68.7819040458903, 162.3104725213],
       ['Male', 74.1101053917849, 212.7408555565],
       ...,
       ['Female', 63.8679922137577, 128.475318784122],
       ['Female', 69.0342431307346, 163.852461346571],
       ['Female', 61.9442458795172, 113.649102675312]], dtype=object)
```

[19]:
```
x = data[:, 0]
y = data[:, 1]
plt.scatter(x,y,label='Gender',color='Blue',s=50)
plt.xlabel('Gender')
plt.ylabel('Height')
plt.title('Gender vs Height')
plt.legend()
plt.show()
```



[18]:
```
import matplotlib.pyplot as plt

x = data[:, 0]
y = data[:, 2]
plt.scatter(x,y,label='Gender',color='Green',s=50)
plt.xlabel('Gender')
plt.ylabel('Weight')
plt.title('Gender vs Weight')
plt.legend()
plt.show()
```



[14]:
```
df.corr()
```

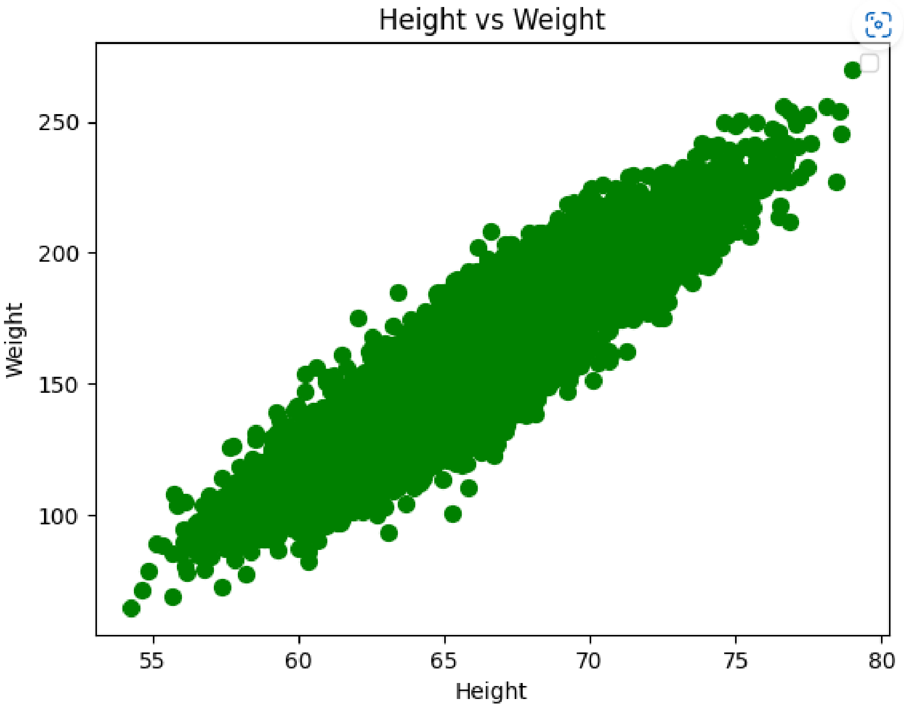
```
/tmp/ipykernel_14477/1134722465.py:
ture version, it will default to Fa
  df.corr()
```

t[14]:

	Height	Weight
Height	1.000000	0.924756
Weight	0.924756	1.000000

[26]:
```
x = data[:, 1]
y = data[:, 2]
plt.scatter(x,y,color='Green',s=50)
plt.xlabel('Height')
plt.ylabel('Weight')
plt.title('Height vs Weight')
plt.legend()
plt.show()
print(x)
print(y)
```

```
No artists with labels found to put in legend.  Note that artists whose label sta
() is called with no argument.
```




```
[73.847017017515 68.7819040458903 74.1101053917849 ... 63.8679922137577
 69.0342431307346 61.9442458795172]
[241.893563180437 162.3104725213 212.7408555565 ... 128.475318784122
 163.852461346571 113.649102675312]
```

[29]:
```
x = x.reshape(-1, 1)
print(x)
print(y)
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

```
[[73.847017017515]
 [68.7819040458903]
 [74.1101053917849]
 ...
 [63.8679922137577]
 [69.0342431307346]
 [61.9442458795172]]
[241.893563180437 162.3104725213 212.7408555565 ... 128.475318784122
 163.852461346571 113.649102675312]
```

```
[31]: from sklearn.linear_model import LinearRegression

      alg = LinearRegression()
      alg.fit(x_train, y_train)
```

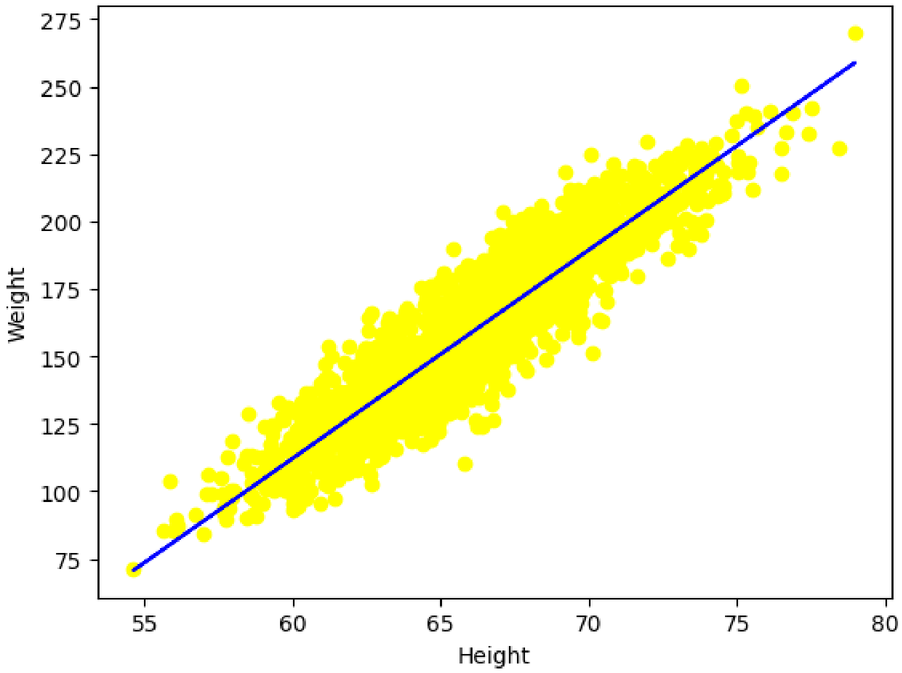
```
[31]: LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
[32]: y_pred = alg.predict(x_test)
      y_pred
```

```
[32]: array([180.02257827, 140.63528449, 185.92637087, ..., 192.17165879,
             208.30063231, 178.83252792])
```

```
[41]: plt.scatter(x_test, y_test, color='yellow')
      plt.plot(x_test, y_pred, color='blue')
      plt.xlabel("Height")
      plt.ylabel('Weight')
      plt.show()
```




```
[52]: import numpy as np

      print('Coefficients:', alg.coef_)

      # The mean squared error
      print(f"Mean squared error: {np.mean((y_pred - y_test) ** 2):.2f}")

      # Explained variance score: 1 is perfect prediction
      print('Variance score: %.2f'% alg.score(x_test, y_test))
```

```
Coefficients: [7.71867935]
Mean squared error: 148.73
Variance score: 0.86
```

2.Multiple Linear Regression

```
[12]: import numpy as numpy
      import pandas as pd
      import matplotlib.pyplot as plt
      from sklearn.datasets import load_iris
```

```
[4]: iris = load_iris()
     iris
```

```
        [4.6, 3.2, 1.4, 0.2],
        [5.3, 3.7, 1.5, 0.2],
        [5. , 3.3, 1.4, 0.2],
        [7. , 3.2, 4.7, 1.4],
        [6.4, 3.2, 4.5, 1.5],
        [6.9, 3.1, 4.9, 1.5],
        [5.5, 2.3, 4. , 1.3],
        [6.5, 2.8, 4.6, 1.5],
        [5.7, 2.8, 4.5, 1.3],
        [6.3, 3.3, 4.7, 1.6],
        [4.9, 2.4, 3.3, 1. ],
        [6.6, 2.9, 4.6, 1.3],
        [5.2, 2.7, 3.9, 1.4],
        [5. , 2. , 3.5, 1. ],
        [5.9, 3. , 4.2, 1.5],
        [6. , 2.2, 4. , 1. ],
        [6.1, 2.9, 4.7, 1.4],
        [5.6, 2.9, 3.6, 1.3],
        [6.7, 3.1, 4.4, 1.4],
        [5.6, 3. , 4.5, 1.5]
```

```
[5]: x = iris.data
     y = iris.target
```

```
[9]: from sklearn.model_selection import train_test_split

     x_train, x_test, y_train, y_test = train_test_split(x, y, random_state = 0)
```

```
[10]: from sklearn.linear_model import LinearRegression

      alg = LinearRegression()
      alg.fit(x_train, y_train)
```

```
[10]: LinearRegression()
```

```
[11]: y_pred = alg.predict(x_test)
      y_pred
```

```
[11]: array([ 2.07872867,  0.9662282 , -0.16117412,  1.82229476, -0.03749929,
              2.28704244, -0.03604989,  1.30986735,  1.27147131,  1.10781204,
              1.59744796,  1.299921  ,  1.23731195,  1.32145191,  1.34954356,
             -0.11133487,  1.36886386,  1.2542803 ,  0.03401222, -0.05014733,
              1.82644819,  1.42764369,  0.09995305,  0.04048737,  1.59299693,
             -0.1147503 ,  0.15857194,  1.17003517,  0.9301028 ,  0.10397109,
              1.74160045,  1.45830398, -0.07070034,  1.62994357,  2.00546549,
              1.27901229, -0.04419114,  1.59151965])
```

Experiment – 5

AIM: To write a program for Gradient Descent and demonstrate types of linear regression

DESCRIPTION:
Gradient descent (GD) is an iterative first-order optimisation algorithm used to find a local minimum/maximum of a given function. This method is commonly used in *machine learning* (ML) and *deep learning*(DL) to minimise a cost/loss function (e.g. in a linear regression).

PROGRAM:

```
[1]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
```

```
[2]: data = pd.read_csv("C:/Users/91995/ML Lab/Lab-2/Salary_Data.csv").values
     data
```

```
[2]: array([[1.10000e+00, 3.93430e+04],
            [1.30000e+00, 4.62050e+04],
            [1.50000e+00, 3.77310e+04],
            [2.00000e+00, 4.35250e+04],
            [2.20000e+00, 3.98910e+04],
            [2.90000e+00, 5.66420e+04],
            [3.00000e+00, 6.01500e+04],
            [3.20000e+00, 5.44450e+04],
            [3.20000e+00, 6.44450e+04],
            [3.70000e+00, 5.71890e+04],
            [3.90000e+00, 6.32180e+04],
            [4.00000e+00, 5.57940e+04],
            [4.00000e+00, 5.69570e+04],
            [4.10000e+00, 5.70810e+04],
            [4.50000e+00, 6.11110e+04],
            [4.90000e+00, 6.79380e+04],
            [5.10000e+00, 6.60290e+04],
            [5.30000e+00, 8.30880e+04],
            [5.90000e+00, 8.13630e+04],
            [6.00000e+00, 9.39400e+04],
            [6.80000e+00, 9.17380e+04],
            [7.10000e+00, 9.82730e+04],
            [7.90000e+00, 1.01302e+05],
            [8.20000e+00, 1.13812e+05],
            [8.70000e+00, 1.09431e+05],
            [9.00000e+00, 1.05582e+05],
            [9.50000e+00, 1.16969e+05],
            [9.60000e+00, 1.12635e+05],
            [1.03000e+01, 1.22391e+05],
            [1.05000e+01, 1.21872e+05]])
```

```
[9]: x= data[:, 0]
     x
```

```
[9]: array([ 1.1,  1.3,  1.5,  2. ,  2.2,  2.9,  3. ,  3.2,  3.2,  3.7,  3.9,
             4. ,  4. ,  4.1,  4.5,  4.9,  5.1,  5.3,  5.9,  6. ,  6.8,  7.1,
             7.9,  8.2,  8.7,  9. ,  9.5,  9.6, 10.3, 10.5])
```

```
[10]: y = data[:, 1]
      y
```

```
[10]: array([ 39343.,  46205.,  37731.,  43525.,  39891.,  56642.,  60150.,
               54445.,  64445.,  57189.,  63218.,  55794.,  56957.,  57081.,
               61111.,  67938.,  66029.,  83088.,  81363.,  93940.,  91738.,
               98273., 101302., 113812., 109431., 105582., 116969., 112635.,
              122391., 121872.])
```

```
[48]: from sklearn.linear_model import LinearRegression

      alg = LinearRegression()
      alg.fit(x.reshape(-1, 1), y)
```

```
[48]: LinearRegression()
```

```
[52]: alg.intercept_
```

```
[52]: 25792.200198668717
```

```
[58]: alg.coef_
```

```
[58]: array([9449.96232146])
```

```
[61]: def mean_squared_error(y_true, y_predicted):
          cost = np.sum((y_true-y_predicted)**2) / len(y_true)
          return cost

      def gradient_descent(x, y, iterations = 1000, learning_rate = 0.02, stopping_threshold = 1e-6):

          current_weight = 0.1
          current_bias = 0.01
          n = float(len(x))

          costs = []
          weights = []
          previous_cost = None

          for i in range(iterations):
              y_predicted = (current_weight * x) + current_bias

              current_cost = mean_squared_error(y, y_predicted)

              previous_cost = current_cost

              costs.append(current_cost)
              weights.append(current_weight)

              weight_derivative = -(2/n) * sum(x * (y-y_predicted))
              bias_derivative = -(2/n) * sum(y-y_predicted)

              current_weight = current_weight - (learning_rate * weight_derivative)
              current_bias = current_bias - (learning_rate * bias_derivative)

              print(f"Iteration {i+1}: Cost {current_cost}, Weight {current_weight}, Bias {current_bias}")

          return current_weight, current_bias
```

```
[62]: estimated_weight, eatimated_bias = gradient_descent(x, y, iterations=500)
      print(f"Estimated Weight: {estimated_weight}\nEstimated Bias: {eatimated_bias}")
```

```
Iteration 484: Cost 31304495.311911035, Weight 9507.922120807743, Bias 25401.624494300133
Iteration 485: Cost 31303930.092406902, Weight 9507.431726242132, Bias 25404.929133119123
Iteration 486: Cost 31303374.397020176, Weight 9506.94548087705, Bias 25408.20581157703
Iteration 487: Cost 31302828.065266732, Weight 9506.463349606342, Bias 25411.45476624488
Iteration 488: Cost 31302290.93936652, Weight 9505.985297620895, Bias 25414.676231692083
Iteration 489: Cost 31301762.864198122, Weight 9505.511290406102, Bias 25417.87044050337
Iteration 490: Cost 31301243.687254086, Weight 9505.041293739394, Bias 25421.037623295593
Iteration 491: Cost 31300733.258596588, Weight 9504.575273687751, Bias 25424.178008734358
Iteration 492: Cost 31300231.430814426, Weight 9504.113196605256, Bias 25427.291823550546
Iteration 493: Cost 31299738.05898028, Weight 9503.655029130674, Bias 25430.379292556685
Iteration 494: Cost 31299253.000608914, Weight 9503.200738185036, Bias 25433.440638663178
Iteration 495: Cost 31298776.115615983, Weight 9502.750290969247, Bias 25436.47608289439
Iteration 496: Cost 31298307.266277596, Weight 9502.303654961732, Bias 25439.48584440462
Iteration 497: Cost 31297846.317190632, Weight 9501.860797916068, Bias 25442.4701404939
Iteration 498: Cost 31297393.13523346, Weight 9501.421687858683, Bias 25445.429186623714
Iteration 499: Cost 31296947.58952759, Weight 9500.986293086513, Bias 25448.363196432532
Iteration 500: Cost 31296509.551399965, Weight 9500.554582164748, Bias 25451.272381751245
Estimated Weight: 9500.554582164748
Estimated Bias: 25451.272381751245
```

```
[63]: # Making predictions using estimated parameters
      y_pred = estimated_weight*x + eatimated_bias
      y_pred
```

```
[63]: array([ 35901.88242213,  37801.99333857,  39702.104255  ,  44452.38154608,
         46352.49246251,  53002.88067003,  53952.93612825,  55853.04704468,
         55853.04704468,  60603.32433576,  62503.43525219,  63453.49071041,
         63453.49071041,  64403.54616863,  68203.76800149,  72003.98983436,
         73904.10075079,  75804.21166722,  81504.54441652,  82454.59987474,
         90055.04354047,  92905.20991512, 100505.65358085, 103355.8199555 ,
        108106.09724658, 110956.26362123, 115706.54091232, 116656.59637053,
        123306.98457805, 125207.09549448])
```

Types of Linear Regression:

1. Simple Linear Regression
2. Multiple Linear Regression

In Simple Linear Regression, we try to find the relationship between a single independent variable (input) and a corresponding dependent variable (output). This can be expressed in the form of a straight line.
The same equation of a line can be re-written as:

$Y = \beta_0 + \beta_1 X$

In Multiple Linear Regression, we try to find the relationship between 2 or more independent variables (inputs) and the corresponding dependent variable (output). The independent variables can be continuous or categorical.
The equation that describes how the predicted values of y is related to p independent variables is called as Multiple Linear Regression equation :

$Y= \beta_0 + \beta_1 x1 + \beta_2 x2+\ldots\ldots. B_p xp$

PROGRAM:

1.Simple Linear Regression

```
[7]: import pandas as pd

     df = pd.read_csv('/home/student/176/weight-height.csv')
     df
```

[7]:

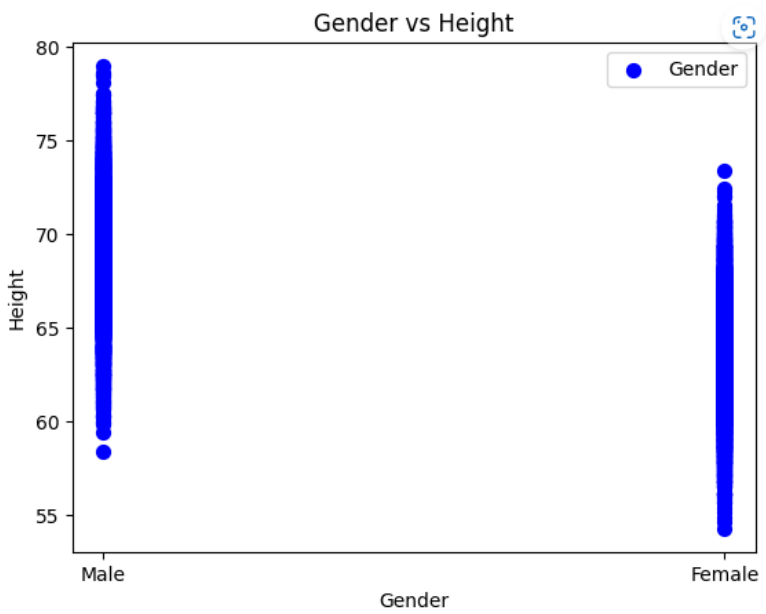
	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801
...	...	...	...
9995	Female	66.172652	136.777454
9996	Female	67.067155	170.867906
9997	Female	63.867992	128.475319
9998	Female	69.034243	163.852461
9999	Female	61.944246	113.649103

10000 rows × 3 columns

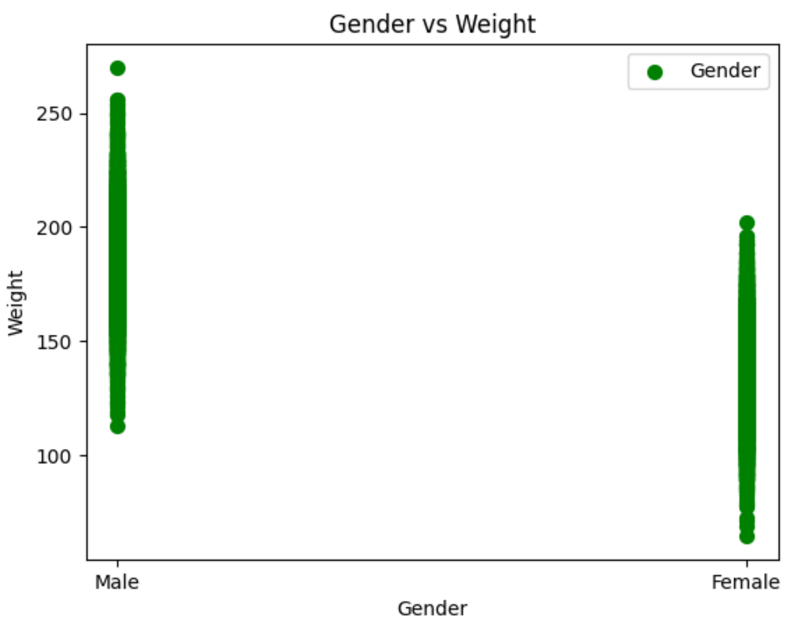
```
[9]: data = df.values
     data
```

```
t[9]: array([['Male', 73.847017017515, 241.893563180437],
             ['Male', 68.7819040458903, 162.3104725213],
             ['Male', 74.1101053917849, 212.7408555565],
             ...,
             ['Female', 63.8679922137577, 128.475318784122],
             ['Female', 69.0342431307346, 163.852461346571],
             ['Female', 61.9442458795172, 113.649102675312]], dtype=object)
```

[19]:
```
x = data[:, 0]
y = data[:, 1]
plt.scatter(x,y,label='Gender',color='Blue',s=50)
plt.xlabel('Gender')
plt.ylabel('Height')
plt.title('Gender vs Height')
plt.legend()
plt.show()
```



[18]:
```
import matplotlib.pyplot as plt

x = data[:, 0]
y = data[:, 2]
plt.scatter(x,y,label='Gender',color='Green',s=50)
plt.xlabel('Gender')
plt.ylabel('Weight')
plt.title('Gender vs Weight')
plt.legend()
plt.show()
```



[14]:
```
df.corr()
```

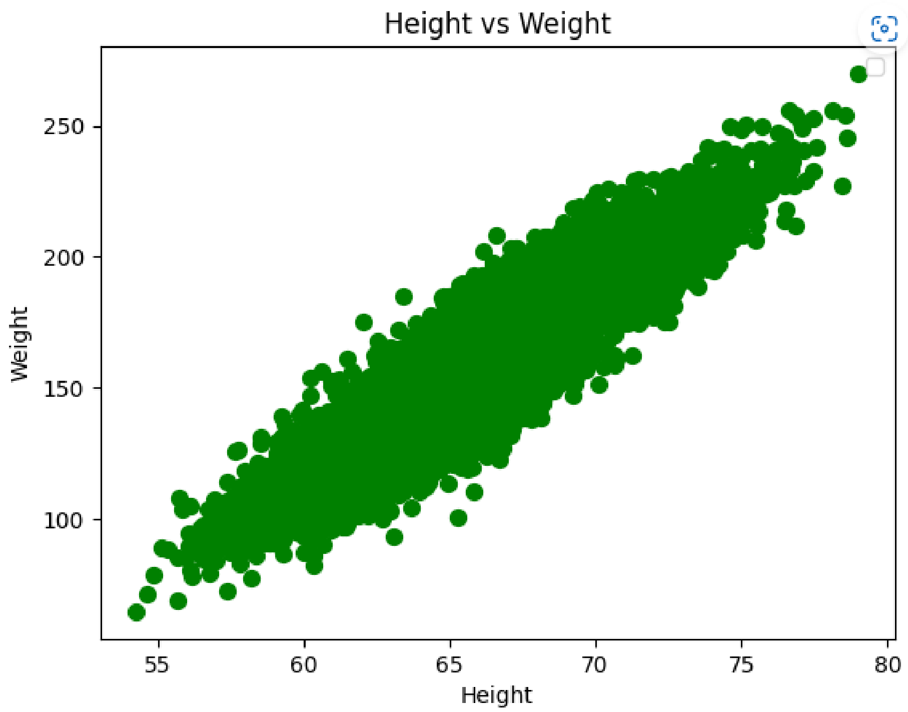
```
/tmp/ipykernel_14477/1134722465.py:
ture version, it will default to Fa
  df.corr()
```

t[14]:

	Height	Weight
Height	1.000000	0.924756
Weight	0.924756	1.000000

[26]:
```
x = data[:, 1]
y = data[:, 2]
plt.scatter(x,y,color='Green',s=50)
plt.xlabel('Height')
plt.ylabel('Weight')
plt.title('Height vs Weight')
plt.legend()
plt.show()
print(x)
print(y)
```

```
No artists with labels found to put in legend.  Note that artists whose label sta
() is called with no argument.
```




```
[73.847017017515 68.7819040458903 74.1101053917849 ... 63.8679922137577
 69.0342431307346 61.9442458795172]
[241.893563180437 162.3104725213 212.7408555565 ... 128.475318784122
 163.852461346571 113.649102675312]
```

[29]:
```
x = x.reshape(-1, 1)
print(x)
print(y)
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
```

```
[[73.847017017515]
 [68.7819040458903]
 [74.1101053917849]
 ...
 [63.8679922137577]
 [69.0342431307346]
 [61.9442458795172]]
[241.893563180437 162.3104725213 212.7408555565 ... 128.475318784122
 163.852461346571 113.649102675312]
```

[31]:
```
from sklearn.linear_model import LinearRegression

alg = LinearRegression()
alg.fit(x_train, y_train)
```

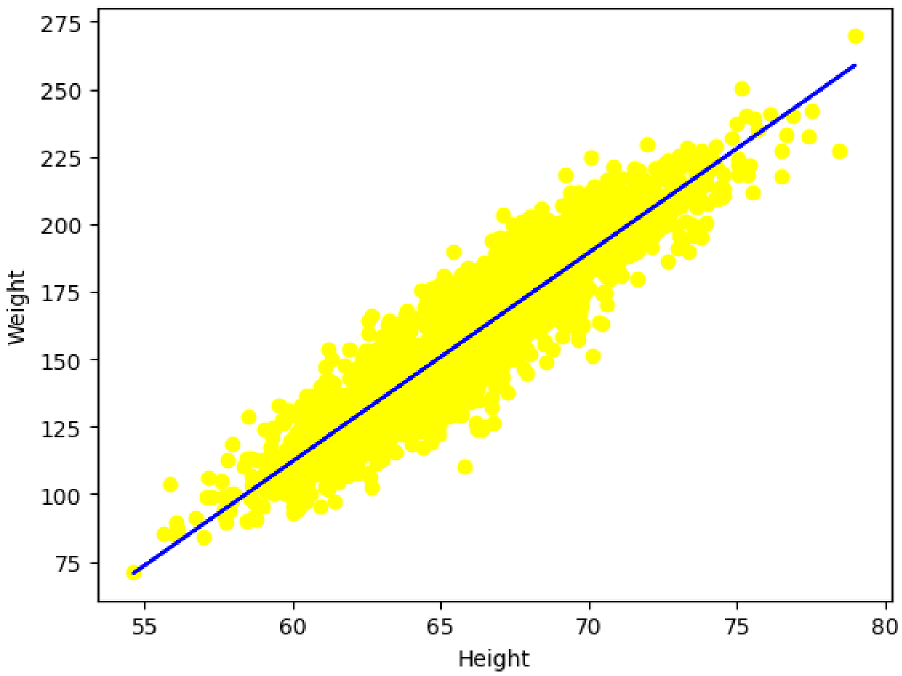
[31]:
```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

[32]:
```
y_pred = alg.predict(x_test)
y_pred
```

[32]:
```
array([180.02257827, 140.63528449, 185.92637087, ..., 192.17165879,
       208.30063231, 178.83252792])
```

[41]:
```
plt.scatter(x_test, y_test, color='yellow')
plt.plot(x_test, y_pred, color='blue')
plt.xlabel("Height")
plt.ylabel('Weight')
plt.show()
```



```
[52]: import numpy as np

      print('Coefficients:', alg.coef_)

      # The mean squared error
      print(f"Mean squared error: {np.mean((y_pred - y_test) ** 2):.2f}")

      # Explained variance score: 1 is perfect prediction
      print('Variance score: %.2f'% alg.score(x_test, y_test))

      Coefficients: [7.71867935]
      Mean squared error: 148.73
      Variance score: 0.86
```

2.Multiple Linear Regression

```
[12]: import numpy as numpy
      import pandas as pd
      import matplotlib.pyplot as plt
      from sklearn.datasets import load_iris

[4]:  iris = load_iris()
      iris

             [4.6, 3.2, 1.4, 0.2],
             [5.3, 3.7, 1.5, 0.2],
             [5. , 3.3, 1.4, 0.2],
             [7. , 3.2, 4.7, 1.4],
             [6.4, 3.2, 4.5, 1.5],
             [6.9, 3.1, 4.9, 1.5],
             [5.5, 2.3, 4. , 1.3],
             [6.5, 2.8, 4.6, 1.5],
             [5.7, 2.8, 4.5, 1.3],
             [6.3, 3.3, 4.7, 1.6],
             [4.9, 2.4, 3.3, 1. ],
             [6.6, 2.9, 4.6, 1.3],
             [5.2, 2.7, 3.9, 1.4],
             [5. , 2. , 3.5, 1. ],
             [5.9, 3. , 4.2, 1.5],
             [6. , 2.2, 4. , 1. ],
             [6.1, 2.9, 4.7, 1.4],
             [5.6, 2.9, 3.6, 1.3],
             [6.7, 3.1, 4.4, 1.4],
             [5.6, 3. , 4.5, 1.5]

[5]:  x = iris.data
      y = iris.target

[9]:  from sklearn.model_selection import train_test_split

      x_train, x_test, y_train, y_test = train_test_split(x, y, random_state = 0)

[10]: from sklearn.linear_model import LinearRegression

      alg = LinearRegression()
      alg.fit(x_train, y_train)

[10]: LinearRegression()
```

```
[11]: y_pred = alg.predict(x_test)
      y_pred
```

```
[11]: array([ 2.07872867,  0.9662282 , -0.16117412,  1.82229476, -0.03749929,
              2.28704244, -0.03604989,  1.30986735,  1.27147131,  1.10781204,
              1.59744796,  1.299921  ,  1.23731195,  1.32145191,  1.34954356,
             -0.11133487,  1.36886386,  1.2542803 ,  0.03401222, -0.05014733,
              1.82644819,  1.42764369,  0.09995305,  0.04048737,  1.59299693,
             -0.1147503 ,  0.15857194,  1.17003517,  0.9301028 ,  0.10397109,
              1.74160045,  1.45830398, -0.07070034,  1.62994357,  2.00546549,
              1.27901229, -0.04419114,  1.59151965])
```

Experiment – 6

AIM: To write a program for Logistic Regression

DESCRIPTION:
It is a type of statistical model, often used for classification and predictive analytics. Logistic regression estimates the probability of an event occurring, It is used when the dependent variable(target) is categorical.

PROGRAM:

```
[2]: from sklearn import datasets
     from sklearn.linear_model import LogisticRegression
```

```
23]: df = datasets.load_breast_cancer()
     df
```

```
23]: {'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
             1.189e-01],
            [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
             8.902e-02],
            [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
             8.758e-02],
            ...,
            [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
             7.820e-02],
            [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
             1.240e-01],
            [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
             7.039e-02]]),
  'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
        0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
        1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
        1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
        0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
        0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0,
        1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
        1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
        1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1]),
```

```
[3]: X = df.data
     y = df.target
```

```
[4]: from sklearn.model_selection import train_test_split

     x_train, x_test, y_train, y_test = train_test_split(X, y)

     clf = LogisticRegression()
     clf.fit(x_train, y_train)
```

```
C:\Users\91995\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Conve
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
:[4]: LogisticRegression()
```

```
[5]: y_pred = clf.predict(x_test)
     y_pred
```

```
:[5]: array([1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
             1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
             0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0,
             1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
             1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1,
             1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1])
```



```
[5]: y_pred = clf.predict(x_test)
     y_pred
```

```
:[5]: array([1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
             1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
             0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0,
             1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
             1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
             1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1,
             1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1])
```



```
[7]: from sklearn.metrics import precision_score, accuracy_score, mean_absolute_error

     acc = accuracy_score(y_pred, y_test)
     print('Accuracy:',acc)
```

```
Accuracy: 0.9300699300699301
```

```
[8]: pre = precision_score(y_pred, y_test)
     print('Precision:',pre)
```

```
Precision: 0.9494949494949495
```

```
[9]: mae = mean_absolute_error(y_pred, y_test)
     print('Mean Absolute error:',mae)
```

```
Mean Absolute error: 0.06993006993006994
```

Experiment – 7

AIM: To write a program for Naïve Bayes Classifier

DESCRIPTION:
Naive Bayes classifier is classification algorithm based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e., every pair of features being classified is independent of each other.

PROGRAM:

[80]:
```
import pandas as pd
import numpy as np
```

[81]:
```
df = pd.read_csv("buys_computer.csv")
df
```

t[81]:

	age	income	student	credit_rating	buys_computer
0	<=30	high	no	fair	no
1	<=30	high	no	excellent	no
2	31..40	high	no	fair	yes
3	>40	medium	no	fair	yes
4	>40	low	yes	fair	yes
5	>40	low	yes	excellent	no
6	31..40	low	yes	excellent	yes
7	<=30	medium	no	fair	no
8	<=30	low	yes	fair	yes
9	>40	medium	yes	fair	yes
10	<=30	medium	yes	excellent	yes
11	31..40	medium	no	excellent	yes
12	31..40	high	yes	fair	yes
13	>40	medium	no	excellent	no

[83]:
```
from sklearn.preprocessing import LabelEncoder

en = LabelEncoder()
df['age'] = en.fit_transform(df['age'])
df['income'] = en.fit_transform(df['income'])
df['student'] = en.fit_transform(df['student'])
df['credit_rating'] = en.fit_transform(df['credit_rating'])
df['buys_computer'] = en.fit_transform(df['buys_computer'])
```

[84]:
```
df
```

[84]:

	age	income	student	credit_rating	buys_computer
0	1	0	0	1	0
1	1	0	0	0	0
2	0	0	0	1	1
3	2	2	0	1	1
4	2	1	1	1	1
5	2	1	1	0	0
6	0	1	1	0	1
7	1	2	0	1	0
8	1	1	1	1	1
9	2	2	1	1	1
10	1	2	1	0	1
11	0	2	0	0	1
12	0	0	1	1	1
13	2	2	0	0	0

[85]:
```
data= df.values
```

[86]:
```
x_train = data[:, :-1]
x_train
```

:[86]:
```
array([[1, 0, 0, 1],
       [1, 0, 0, 0],
       [0, 0, 0, 1],
       [2, 2, 0, 1],
       [2, 1, 1, 1],
       [2, 1, 1, 0],
       [0, 1, 1, 0],
       [1, 2, 0, 1],
       [1, 1, 1, 1],
       [2, 2, 1, 1],
       [1, 2, 1, 0],
       [0, 2, 0, 0],
       [0, 0, 1, 1],
       [2, 2, 0, 0]])
```

[88]:
```
x_test = np.array([[0, 2, 1, 1]])
x_test
```

:[88]: array([[0, 2, 1, 1]])

[89]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x_train, y_train)

y_pred = gnb.predict(x_test)
```

[91]:
```
y_pred
```

:[91]: array([1])

[92]:
```
from sklearn import tree

model = tree.DecisionTreeClassifier()
model = model.fit(x_train, y_train)
predicted_value = model.predict(x_test)
predicted_value
```

:[92]: array([1])

Experiment – 8

AIM: To write a program for Decision Tree

DESCRIPTION:
Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

PROGRAM:

```
[2]: from sklearn.datasets import load_iris
     from sklearn import tree
```

```
[3]: iris = load_iris()
     iris
```

```
t[3]: {'data': array([[5.1, 3.5, 1.4, 0.2],
              [4.9, 3. , 1.4, 0.2],
              [4.7, 3.2, 1.3, 0.2],
              [4.6, 3.1, 1.5, 0.2],
              [5. , 3.6, 1.4, 0.2],
              [5.4, 3.9, 1.7, 0.4],
              [4.6, 3.4, 1.4, 0.3],
              [5. , 3.4, 1.5, 0.2],
              [4.4, 2.9, 1.4, 0.2],
              [4.9, 3.1, 1.5, 0.1],
              [5.4, 3.7, 1.5, 0.2],
              [4.8, 3.4, 1.6, 0.2],
              [4.8, 3. , 1.4, 0.1],
              [4.3, 3. , 1.1, 0.1],
              [5.8, 4. , 1.2, 0.2],
              [5.7, 4.4, 1.5, 0.4],
              [5.4, 3.9, 1.3, 0.4],
              [5.1, 3.5, 1.4, 0.3],
              [5.7, 3.8, 1.7, 0.3],
```

```
[4]: from sklearn.model_selection import train_test_split

     X, y = iris.data, iris.target
     X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

```
[13]: clf1 = tree.DecisionTreeClassifier(criterion='entropy', random_state=1)
      clf1 = clf1.fit(X_train, y_train)
```

```
[14]: y_pred = clf1.predict(X_test)
      y_pred
```

```
[14]: array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1,
             2, 0, 2, 1, 0, 0, 1, 2, 1, 2, 1, 2, 2, 0, 1, 0])
```

```
[15]: from sklearn.metrics import accuracy_score

      acc = accuracy_score(y_test, y_pred)
      print('Accuracy score:',acc)

      Accuracy score: 0.9736842105263158
```

Experiment – 9

AIM: To write a program to demonstrate ensemble methods

DESCRIPTION:

Bagging, the short form for bootstrap aggregating, is mainly applied in classification and *regression*. It increases the accuracy of models through decision trees, which reduces variance to a large extent. The reduction of variance increases accuracy, eliminating overfitting, which is a challenge to many predictive models.
Boosting is an ensemble technique that learns from previous predictor mistakes to make better predictions in the future. The technique combines several weak base learners to form one strong learner, thus significantly improving the predictability of models. Boosting works by arranging weak learners in a sequence, such that weak learners learn from the next learner in the sequence to create better predictive models.
Stacking, another ensemble method, is often referred to as stacked generalization. This technique works by allowing a training algorithm to ensemble several other similar learning algorithm predictions. Stacking has been successfully implemented in regression, density estimations, distance learning, and classifications. It can also be used to measure the error rate involved during bagging

PROGRAM:

```
[65]: import pandas as pd
      from sklearn.datasets import load_breast_cancer
```

```
[66]: cancer = load_breast_cancer()
      cancer
```

```
t[66]: {'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
                 1.189e-01],
                [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
                 8.902e-02],
                [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
                 8.758e-02],
                ...,
                [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
                 7.820e-02],
                [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
                 1.240e-01],
                [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
                 7.039e-02]]),
        'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
                1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
                1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
```

[67]:
```
df = pd.DataFrame(cancer.data, columns=cancer.feature_names)
df
```

[67]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	wor smoothne
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	184.60	2019.0	0.162
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	158.80	1956.0	0.123
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	152.50	1709.0	0.144
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	98.87	567.7	0.209
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	152.20	1575.0	0.137
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	166.10	2027.0	0.141
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	155.00	1731.0	0.116
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	126.70	1124.0	0.113
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	184.60	1821.0	0.165
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	59.16	268.6	0.089

569 rows × 30 columns

[68]:
```
df.isnull().sum()
```

[69]:
```
x = df.values
x
```

[70]:
```
y = cancer.target
y
```

[71]:
```
cols = ['Ensemble method', 'Accuracy']
summary = []
```

[72]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=1)
```

BAGGING

[73]:
```
#Bagging
name = 'bagging'
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics import accuracy_score

model = BaggingClassifier()
model.fit(x_train, y_train)
```

[73]:
```
BaggingClassifier()
```

[74]:
```
y_pred = model.predict(x_test)
acc = accuracy_score(y_pred, y_test)
print('Accuracy:',acc)
summary.append([name, acc])
```

```
Accuracy: 0.965034965034965
```

GRADIENT BOOSTING

[75]:
```
#Gradient Boosting
name = 'Gradient Boostng'
from sklearn.ensemble import GradientBoostingClassifier

model = GradientBoostingClassifier()
model.fit(x_train, y_train)
```

[75]:
```
GradientBoostingClassifier()
```

[76]:
```
y_pred = model.predict(x_test)
acc = accuracy_score(y_pred, y_test)
print('Accuracy:',acc)
summary.append([name, acc])
```

```
Accuracy: 0.965034965034965
```

STACKING

[77]:
```
#Stacing
name = 'Stacking'
from sklearn.ensemble import StackingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

base_learners = [
    ('rf_1', RandomForestClassifier()),
    ('rf_2', KNeighborsClassifier(n_neighbors=5))
]

model = StackingClassifier(estimators = base_learners)
model.fit(x_train, y_train)
```

[77]:
```
StackingClassifier(estimators=[('rf_1', RandomForestClassifier()),
                               ('rf_2', KNeighborsClassifier())])
```

[78]:
```
y_pred = model.predict(x_test)
acc = accuracy_score(y_pred, y_test)
print('Accuracy:',acc)
summary.append([name, acc])
```

```
Accuracy: 0.958041958041958
```

COMPARISON

[79]:
```
summary = pd.DataFrame(summary, columns=cols)
summary
```

t[79]:

	Ensemble method	Accuracy
0	bagging	0.965035
1	Gradient Boostng	0.965035
2	Stacking	0.958042

CONCLUSION:

From the experimented results for Breast Cancer dataset, we can infer that both Bagging and Boosting gave identical results with accuracy 0.96

Experiment – 10

AIM: To write a program to demonstrate SVM

DESCRIPTION:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

PROGRAM:

```
[65]: import pandas as pd
      from sklearn.datasets import load_breast_cancer
```

```
[66]: cancer = load_breast_cancer()
      cancer
```

```
t[66]: {'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
                 1.189e-01],
                [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
                 8.902e-02],
                [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
                 8.758e-02],
                ...,
                [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
                 7.820e-02],
                [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
                 1.240e-01],
                [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
                 7.039e-02]]),
         'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
                0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
                1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
                1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
                1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
```

```
[67]: df = pd.DataFrame(cancer.data, columns=cancer.feature_names)
      df
```

[67]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	worst perimeter	worst area	wor smoothne
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	25.380	17.33	184.60	2019.0	0.162
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	24.990	23.41	158.80	1956.0	0.123
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	23.570	25.53	152.50	1709.0	0.144
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	14.910	26.50	98.87	567.7	0.209
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	22.540	16.67	152.20	1575.0	0.137
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	25.450	26.40	166.10	2027.0	0.141
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	23.690	38.25	155.00	1731.0	0.116
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	18.980	34.12	126.70	1124.0	0.113
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	25.740	39.42	184.60	1821.0	0.165
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	9.456	30.37	59.16	268.6	0.089

569 rows × 30 columns

```
[68]: df.isnull().sum()
```

```
[69]: x = df.values
      x
```

```
[70]: y = cancer.target
      y
```

```
[24]: cols = ['SVM', 'Accuracy']
      summary = []
```

```
[25]: from sklearn.model_selection import train_test_split
      x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=1)
```

SVM - linear

```
[26]: #SVM Linear
      name = 'SVM-linear'
      from sklearn.svm import SVC
      from sklearn.metrics import accuracy_score

      model = SVC(kernel='linear', probability=True)
      model.fit(x_train, y_train)
```

```
Out[26]: SVC(kernel='linear', probability=True)
```

```
[27]: y_pred = model.predict(x_test)
      acc = accuracy_score(y_pred, y_test)
      print('Accuracy:',acc)
      summary.append([name, acc])
```

```
Accuracy: 0.9370629370629371
```

SVM – rbf

[28]:
```python
#SVM RBF
name = 'SVM-rbf'
from sklearn.model_selection import GridSearchCV

# pg = {'C':[0.1,1,10,100,1000], 'gamma':[1,0.1,0.01,0.001,0.0001]}

# grid = GridSearchCV(model, param_grid=pg, cv=10)
# grid.fit(x_train, y_train)

# print('Best Hyperparameter: ',grid.best_params_)

model = SVC(kernel='rbf', C=1000, probability=True)
model.fit(x_train, y_train)
```

Out[28]:
```
SVC(C=1000, probability=True)
```

[29]:
```python
y_pred = model.predict(x_test)
acc = accuracy_score(y_pred, y_test)
print('Accuracy:',acc)
summary.append([name, acc])
```

```
Accuracy: 0.951048951048951
```

SVM – poly

[30]:
```python
#SVM Poly
name = 'SVM-poly'

model = SVC(kernel='poly', probability=True, degree=3)
model.fit(x_train, y_train)
```

Out[30]:
```
SVC(kernel='poly', probability=True)
```

[31]:
```python
y_pred = model.predict(x_test)
acc = accuracy_score(y_pred, y_test)
print('Accuracy:',acc)
summary.append([name, acc])
```

```
Accuracy: 0.9020979020979021
```

[32]:
```python
summary = pd.DataFrame(summary, columns=cols)
summary
```

Out[32]:

	SVM	Accuracy
0	SVM-linear	0.937063
1	SVM-rbf	0.951049
2	SVM-poly	0.902098

CONCLUSION:

From the experimented results for Breast Cancer dataset, we can infer that SVM -RBF gave better results with accuracy 0.95

Experiment – 11

AIM: To do a case study on classification methods

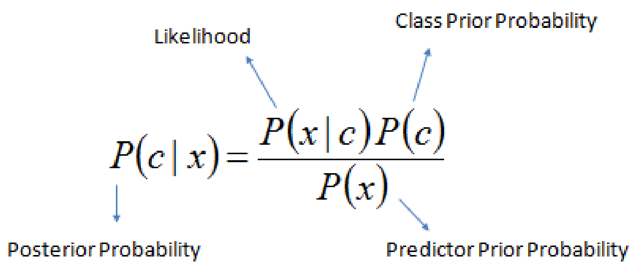
DESCRIPTION:

LOGISTIC REGRESSION

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable,where the two values are labeled "0" and "1".

NAIVE BAYES

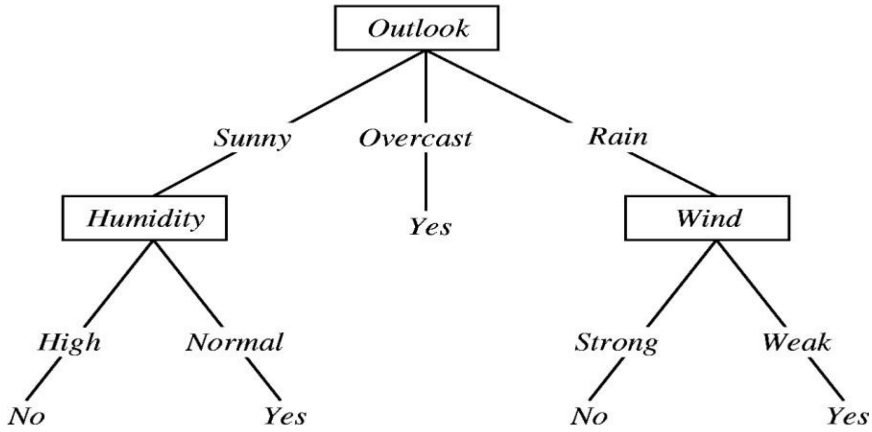
Naive bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Naive bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, naive bayes is known to outperform even highly sophisticated classification methods. Bayes theorem provides a way of calculating posterior probability p(c|x) from p(c), p(x) and p(x|c). Look at the equation below:



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

DECISION TREE

A decision tree is a flowchart-like structure in which each internal node represents a test on a feature (e.g. whether a coin flip comes up heads or tails) , each leaf node represents a class label (decision taken after computing all features) and branches represent conjunctions of features that lead to those class labels. The paths from root to leaf represent classification rules. Below diagram illustrate the basic flow of decision tree for decision making with labels (Rain(Yes), No Rain(No))



ENSEMBLE METHODS(BAGGING ,BOOSTING, STACKING)

BAGGING

Bootstrap Aggregation (or Bagging for short) is a simple and very powerful ensemble method.

An ensemble method is a technique that combines the predictions from multiple machine learning algorithms together to make more accurate predictions than any individual model.

Bootstrap Aggregation is a general procedure that can be used to reduce the variance for those algorithms that have high variance. An algorithm that has high variance are decision trees, like classification and regression trees (CART).

Decision trees are sensitive to the specific data on which they are trained. If the training data is changed (e.g, a tree is trained on a subset of the training data) the resulting decision tree can be quite different and in turn the predictions can be quite different.

Bagging is the application of the Bootstrap procedure to a high-variance machine learning algorithm, typically decision trees.

Let us assume we have a sample dataset of 1000 instances (x) and we are using the CART algorithm. Bagging of the CART algorithm would work as follows.

1. Create many (e.g, 100) random sub-samples of our dataset with replacement.
2. Train a CART model on each sample.
3. Given a new dataset, calculate the average prediction from each model.

BOOSTING

Boosting is used to create a collection of predictors. In this technique, learners are learned sequentially with early learners fitting simple models to the data and then analysing data for errors. Consecutive trees (random sample) are fit and at every step, the goal is to improve the accuracy from the prior tree. When an input is misclassified by a hypothesis, its weight is increased so that next hypothesis is more likely to classify it correctly. This process converts weak learners into better performing mode

BOOSTING STEPS

Draw a random subset of training samples d1 without replacement from the training set D to train a weak learner C1

- Draw second random training subset d2 without replacement from the training set and add 50 percent of the samples that were previously falsely classified/misclassified to train a weak learner C2
- Find the training samples d3 in the training set D on which C1 and C2 disagree to train a third weak learner C3
- Combine all the weak learners via majority voting.

STACKING

Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor. The base level models are trained based on a complete training set, then the meta-model is trained on the outputs of the base level model as features. The base level often consists of different learning algorithms and therefore stackingensembles are often heterogeneous. The algorithm below summarizes stacking.

Algorithm Stacking

Input: training data $D = \{x_i, y_i\}_{i=1}^{m}$
Ouput: ensemble classifier H
Step 1: learn base-level classifiers
for $t = 1$ to T **do**
    learn h_t based on D
end for
Step 2: construct new data set of predictions
for $i = 1$ to m **do**
    $D_h = \{x'_i, y_i\}$, where $x'_i = \{h_1(x_i), ..., h_T(x_i)\}$
end for
Step 3: learn a meta-classifier
learn H based on D_h
return H

SVM

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labelled training data for each category, they can categorize new text.

More formally, a support-vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier

PROGRAM:

```
[123]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve
```

```
[124]: adv = pd.read_csv('advertising.txt', sep=',')
df = adv.copy()
df
```

[124]:

	Daily Time Spent on Site	Age	Area Income	Daily Internet Usage	Ad Topic Line	City	Male	Country	Timestamp	Clicked on Ad
0	68.95	35	61833.90	256.09	Cloned 5thgeneration orchestration	Wrightburgh	0	Tunisia	2016-03-27 00:53:11	0
1	80.23	31	68441.85	193.77	Monitored national standardization	West Jodi	1	Nauru	2016-04-04 01:39:02	0
2	69.47	26	59785.94	236.50	Organic bottom-line service-desk	Davidton	0	San Marino	2016-03-13 20:35:42	0
3	74.15	29	54806.18	245.89	Triple-buffered reciprocal time-frame	West Terrifurt	1	Italy	2016-01-10 02:31:19	0
4	68.37	35	73889.99	225.58	Robust logistical utilization	South Manuel	0	Iceland	2016-06-03 03:36:18	0
...	...	...	...	...	...	...	...	...	...	...
995	72.97	30	71384.57	208.58	Fundamental modular algorithm	Duffystad	1	Lebanon	2016-02-11 21:49:00	1
996	51.30	45	67782.17	134.42	Grass-roots cohesive monitoring	New Darlene	1	Bosnia and Herzegovina	2016-04-22 02:07:01	1
997	51.63	51	42415.72	120.37	Expanded intangible solution	South Jessica	1	Mongolia	2016-02-01 17:24:57	1
998	55.55	19	41920.79	187.95	Proactive bandwidth-monitored policy	West Steven	0	Guatemala	2016-03-24 02:35:54	0
999	45.01	26	29875.80	178.35	Virtual 5thgeneration emulation	Ronniemouth	0	Brazil	2016-06-03 21:43:21	1

1000 rows × 10 columns

PREPROCESSING:

```
[125]: #check if any misssing values
df.isnull().sum()
```

```
[125]: Daily Time Spent on Site    0
Age                         0
Area Income                 0
Daily Internet Usage        0
Ad Topic Line               0
City                        0
Male                        0
Country                     0
Timestamp                   0
Clicked on Ad               0
dtype: int64
```

```
[126]: print(df.iloc[:,-1].value_counts())
```

```
0    500
1    500
Name: Clicked on Ad, dtype: int64
```

[127]:
```
#normalization
from sklearn.preprocessing import MinMaxScaler
cols = ['Daily Time Spent on Site', 'Age', 'Area Income', 'Daily Internet Usage']
```

[128]:
```
sc = MinMaxScaler()
x = sc.fit_transform(df[cols])
x
```

[128]:
```
array([[0.61788203, 0.38095238, 0.73047247, 0.916031  ],
       [0.80962094, 0.28571429, 0.83137522, 0.53874561],
       [0.62672106, 0.16666667, 0.69920032, 0.7974331 ],
       ...,
       [0.32347442, 0.76190476, 0.43395874, 0.09438189],
       [0.39010709, 0.        , 0.4264012 , 0.50351132],
       [0.2109468 , 0.16666667, 0.24247537, 0.4453929 ]])
```

[129]:
```
x= pd.DataFrame(x)
x['Male'] = df['Male']
x['Country'] = df['Country']
x
```

[129]:

	0	1	2	3	Male	Country
0	0.617882	0.380952	0.730472	0.916031	0	Tunisia
1	0.809621	0.285714	0.831375	0.538746	1	Nauru
2	0.626721	0.166667	0.699200	0.797433	0	San Marino
3	0.706272	0.238095	0.623160	0.854280	1	Italy
4	0.608023	0.380952	0.914568	0.731323	0	Iceland
...	...	...	...	...	...	...
995	0.686215	0.261905	0.876310	0.628405	1	Lebanon
996	0.317865	0.619048	0.821302	0.179441	1	Bosnia and Herzegovina
997	0.323474	0.761905	0.433959	0.094382	1	Mongolia
998	0.390107	0.000000	0.426401	0.503511	0	Guatemala
999	0.210947	0.166667	0.242475	0.445393	0	Brazil

1000 rows × 6 columns

[132]:
```
#label encoding
from sklearn.preprocessing import LabelEncoder

en = LabelEncoder()
x['Country'] = en.fit_transform(x['Country'])
```

[133]:
```
x
```

[133]:

	0	1	2	3	Male	Country
0	0.617882	0.380952	0.730472	0.916031	0	215
1	0.809621	0.285714	0.831375	0.538746	1	147
2	0.626721	0.166667	0.699200	0.797433	0	184
3	0.706272	0.238095	0.623160	0.854280	1	103
4	0.608023	0.380952	0.914568	0.731323	0	96
...	...	...	...	...	...	...
995	0.686215	0.261905	0.876310	0.628405	1	116
996	0.317865	0.619048	0.821302	0.179441	1	26
997	0.323474	0.761905	0.433959	0.094382	1	140
998	0.390107	0.000000	0.426401	0.503511	0	85
999	0.210947	0.166667	0.242475	0.445393	0	28

1000 rows × 6 columns

[134]:
```
#target
y = df.iloc[:,-1]
y
```

[135]:
```
x = x.values
print(x)
y = y.values
print(y)
```

TRAINING:

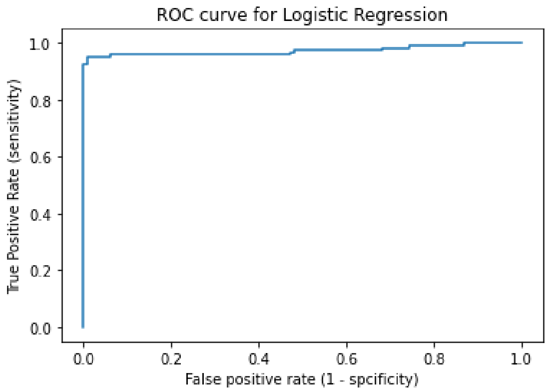
[136]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=1)
```

[137]:
```
all_metrics = ['Classifier', 'Accuracy', 'Precision', 'Recall', 'Specificity', 'AUC']
summary= []
```

LOGISTIC REGRESSION

```
[138]: #Logistic Regression
       name = 'Logistic Regression'
       from sklearn.linear_model import LogisticRegression

       model = LogisticRegression()
       model.fit(x_train, y_train)

[138]: LogisticRegression()
```

```
[139]: from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve, roc_auc_score

       y_pred = model.predict(x_test)
       acc = accuracy_score(y_pred, y_test)
       pre = precision_score(y_pred, y_test)
       rec = recall_score(y_pred, y_test, pos_label = 1)
       spe = recall_score(y_pred, y_test, pos_label = 0)
       print(f'{name}:')
       print('Accuracy:',acc)
       print('Precision:',pre)
       print('Recall:',rec)
       print('Specificity:',spe)

       pred_prob= model.predict_proba(x_test)[:, 1]
       fpr, tpr, thresholds= roc_curve(y_test, pred_prob)
       plt.plot(fpr, tpr)
       plt.title('ROC curve for '+name)
       plt.xlabel('False positive rate (1 - spcificity)')
       plt.ylabel('True Positive Rate (sensitivity)')

       auc = roc_auc_score(y_test, y_pred)
       print('AUC is:',auc)

       summary.append([name, acc, pre, rec, spe, auc])

       Logistic Regression:
       Accuracy: 0.956
       Precision: 0.9090909090909091
       Recall: 1.0
       Specificity: 0.9214285714285714
       AUC is: 0.9545454545454546
```

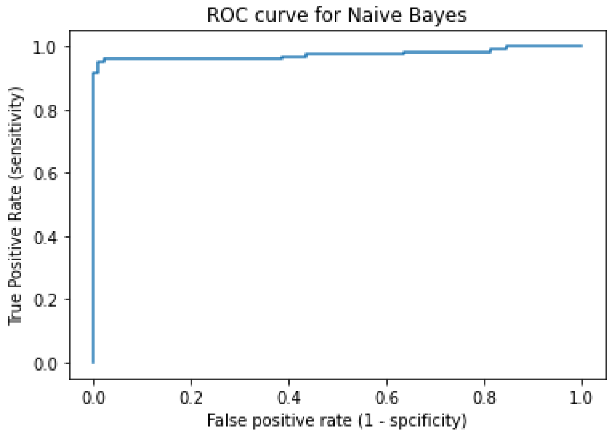


NAÏVE BAYES

```
[140]: #Naive Bayes
       name='Naive Bayes'
       from sklearn.naive_bayes import GaussianNB
       gnb = GaussianNB()
       gnb.fit(x_train, y_train)

[140]: GaussianNB()
```

```
[141]: y_pred = gnb.predict(x_test)

       acc = accuracy_score(y_pred, y_test)
       pre = precision_score(y_pred, y_test)
       rec = recall_score(y_pred, y_test, pos_label = 1)
       spe = recall_score(y_pred, y_test, pos_label = 0)
       print(f'{name}:')
       print('Accuracy:',acc)
       print('Precision:',pre)
       print('Recall:',rec)
       print('Specificity:',spe)

       pred_prob= gnb.predict_proba(x_test)[:, 1]
       fpr, tpr, thresholds= roc_curve(y_test, pred_prob)
       plt.plot(fpr, tpr)
       plt.title('ROC curve for '+name)
       plt.xlabel('False positive rate (1 - spcificity)')
       plt.ylabel('True Positive Rate (sensitivity)')

       auc = roc_auc_score(y_test, y_pred)
       print('AUC is:',auc)

       summary.append([name, acc, pre, rec, spe, auc])
```

```
Naive Bayes:
Accuracy: 0.96
Precision: 0.9256198347107438
Recall: 0.9911504424778761
Specificity: 0.9343065693430657
AUC is: 0.9589339483631238
```



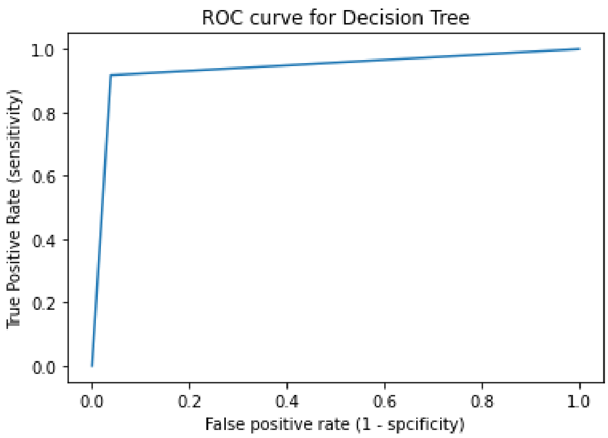
DECISION TREE

```
[142]: #Decision Tree
from sklearn import tree

name = 'Decision Tree'
clf = tree.DecisionTreeClassifier(criterion='entropy')
clf.fit(x_train, y_train)
```

```
t[142]: DecisionTreeClassifier(criterion='entropy')
```

```
[143]: y_pred = clf.predict(x_test)

acc = accuracy_score(y_pred, y_test)
pre = precision_score(y_pred, y_test)
rec = recall_score(y_pred, y_test, pos_label = 1)
spe = recall_score(y_pred, y_test, pos_label = 0)
print(f'{name}:')
print('Accuracy:',acc)
print('Precision:',pre)
print('Recall:',rec)
print('Specificity:',spe)

pred_prob= clf.predict_proba(x_test)[:, 1]
fpr, tpr, thresholds= roc_curve(y_test, pred_prob)
plt.plot(fpr, tpr)
plt.title('ROC curve for '+name)
plt.xlabel('False positive rate (1 - spcificity)')
plt.ylabel('True Positive Rate (sensitivity)')

auc = roc_auc_score(y_test, y_pred)
print('AUC is:',auc)

summary.append([name, acc, pre, rec, spe, auc])
```

```
Decision Tree:
Accuracy: 0.94
Precision: 0.9173553719008265
Recall: 0.9568965517241379
Specificity: 0.9253731343283582
AUC is: 0.9392978409891729
```



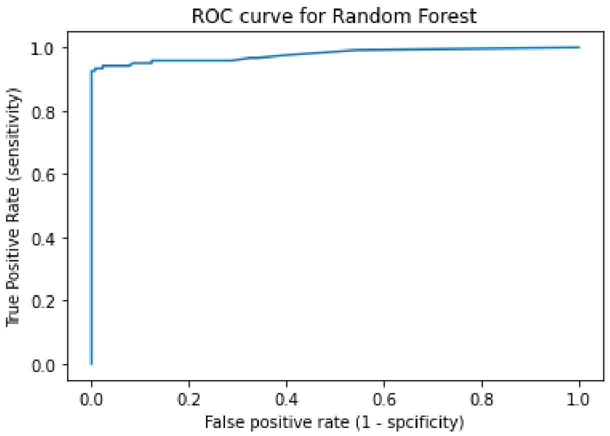
RANDOM FOREST

[144]:
```
#Random Forest
name = 'Random Forest'
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
model.fit(x_train, y_train)
```

t[144]:
```
RandomForestClassifier()
```

[145]:
```
y_pred = model.predict(x_test)
acc = accuracy_score(y_pred, y_test)
pre = precision_score(y_pred, y_test)
rec = recall_score(y_pred, y_test, pos_label = 1)
spe = recall_score(y_pred, y_test, pos_label = 0)
print(f'{name}:')
print('Accuracy:',acc)
print('Precision:',pre)
print('Recall:',rec)
print('Specificity:',spe)

pred_prob= model.predict_proba(x_test)[:, 1]
fpr, tpr, thresholds= roc_curve(y_test, pred_prob)
plt.plot(fpr, tpr)
plt.title('ROC curve for '+name)
plt.xlabel('False positive rate (1 - spcificity)')
plt.ylabel('True Positive Rate (sensitivity)')

auc = roc_auc_score(y_test, y_pred)
print('AUC is:',auc)

summary.append([name, acc, pre, rec, spe, auc])
```

```
Random Forest:
Accuracy: 0.96
Precision: 0.9256198347107438
Recall: 0.9911504424778761
Specificity: 0.9343065693430657
AUC is: 0.9589339483631238
```

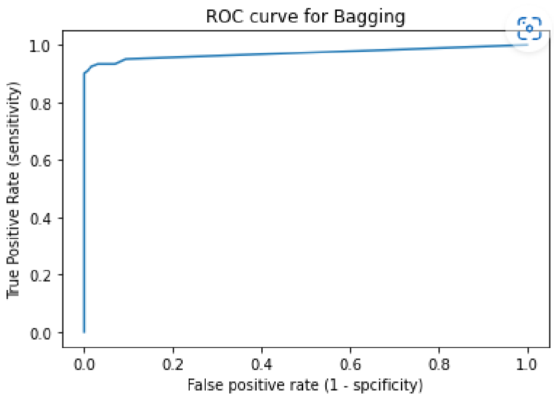


BAGGING

[146]:
```
#Bagging
name = 'Bagging'
from sklearn.ensemble import BaggingClassifier

model = BaggingClassifier()
model.fit(x_train, y_train)
```

[146]: `BaggingClassifier()`

```
Bagging:
Accuracy: 0.952
Precision: 0.9090909090909091
Recall: 0.990990990990991
Specificity: 0.920863309352518
AUC is: 0.9506694855532065
```

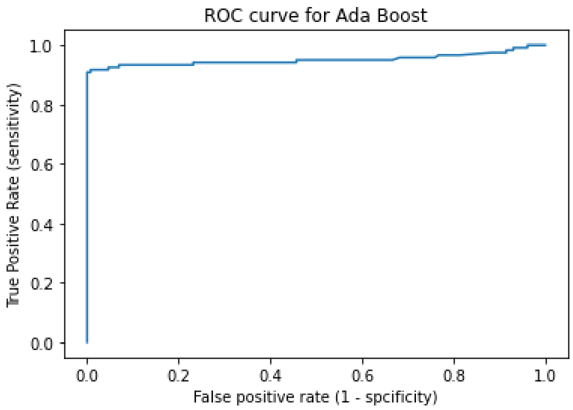


ADA BOOST

[148]:
```
#Ada Boost
name = 'Ada Boost'
from sklearn.ensemble import AdaBoostClassifier

model = AdaBoostClassifier()
model.fit(x_train, y_train)
```

[148]: `AdaBoostClassifier()`

```
Ada Boost:
Accuracy: 0.952
Precision: 0.9090909090909091
Recall: 0.990990990990991
Specificity: 0.920863309352518
AUC is: 0.9506694855532065
```

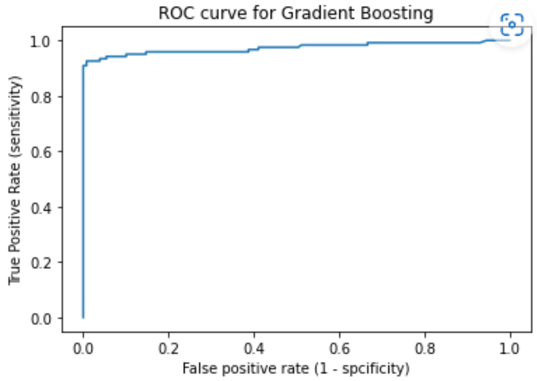


GRADIENT BOOSTING

[150]:
```
#Gradient Boost
name = 'Gradient Boosting'
from sklearn.ensemble import GradientBoostingClassifier

model = GradientBoostingClassifier()
model.fit(x_train, y_train)
```

[150]:
```
GradientBoostingClassifier()
```

```
Gradient Boosting:
Accuracy: 0.952
Precision: 0.9090909090909091
Recall: 0.990990990990991
Specificity: 0.920863309352518
AUC is: 0.9506694855532065
```

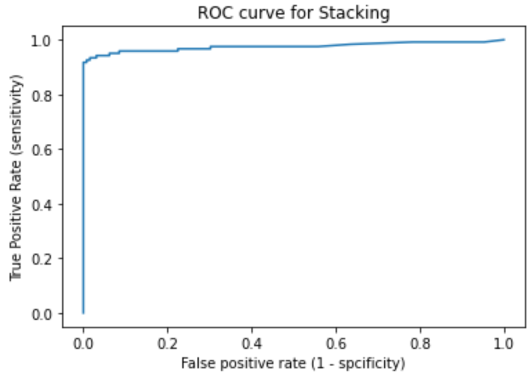


STACKING

[152]:
```
#Stacking
name = 'Stacking'
from sklearn.ensemble import StackingClassifier
from sklearn.neighbors import KNeighborsClassifier

base_learners = [
    ('rf_1', RandomForestClassifier()),
    ('rf_2', KNeighborsClassifier(n_neighbors=5))
]

model = StackingClassifier(estimators = base_learners)
model.fit(x_train, y_train)
```

t[152]:
```
StackingClassifier(estimators=[('rf_1', RandomForestClassifier()),
                               ('rf_2', KNeighborsClassifier())])
```

```
Stacking:
Accuracy: 0.96
Precision: 0.9256198347107438
Recall: 0.9911504424778761
Specificity: 0.9343065693430657
AUC is: 0.9589339483631238
```

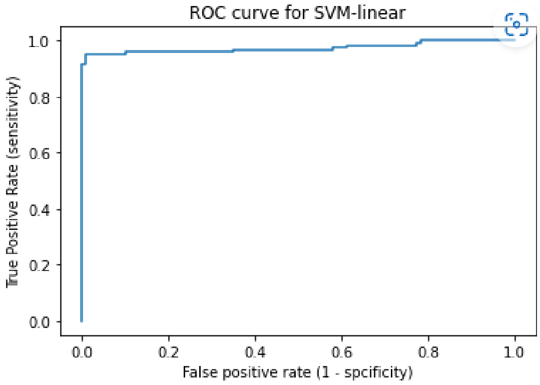


SVM-LINEAR

```
[154]: #SVM linear
       name = 'SVM-linear'
       from sklearn.svm import SVC

       model = SVC(kernel='linear', probability=True)
       model.fit(x_train, y_train)
```

```
[154]: SVC(kernel='linear', probability=True)
```

```
SVM-linear:
Accuracy: 0.956
Precision: 0.9090909090909091
Recall: 1.0
Specificity: 0.9214285714285714
AUC is: 0.9545454545454546
```



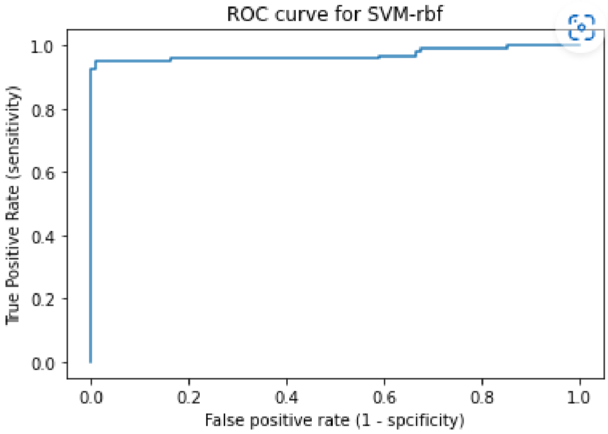
SV-RBF

```
[156]: #SVM RBF
       name = 'SVM-rbf'
       from sklearn.svm import SVC
```

```
[157]: model = SVC(kernel='rbf', C=1000, probability=True)
       model.fit(x_train, y_train)
```

```
[157]: SVC(C=1000, probability=True)
```

```
SVM-rbf:
Accuracy: 0.944
Precision: 0.8842975206611571
Recall: 1.0
Specificity: 0.9020979020979021
AUC is: 0.9421487603305785
```



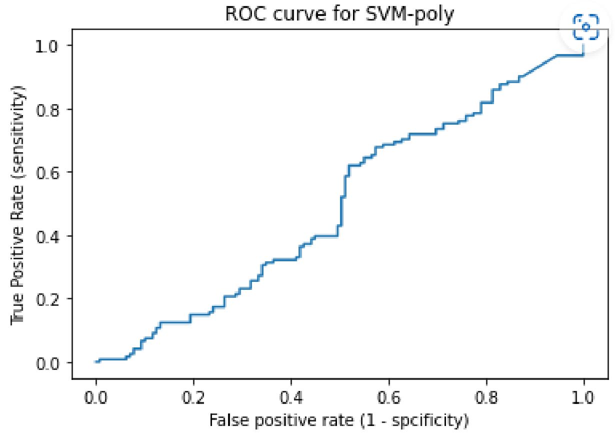
SVM- POLY

[159]:
```
#SVM Poly
name = 'SVM-poly'
from sklearn.svm import SVC
```

[160]:
```
model = SVC(kernel='poly', probability=True)
model.fit(x_train, y_train)
```

[160]:
```
SVC(kernel='poly', probability=True)
```

```
SVM-poly:
Accuracy: 0.504
Precision: 0.17355371900826447
Recall: 0.4666666666666667
Specificity: 0.5121951219512195
AUC is: 0.49375360369017873
```



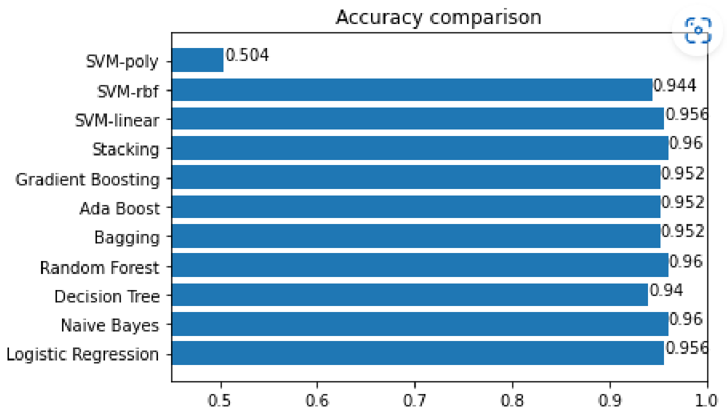
COMPARISON

[162]:
```
summary = pd.DataFrame(summary, columns=all_metrics)
summary
```

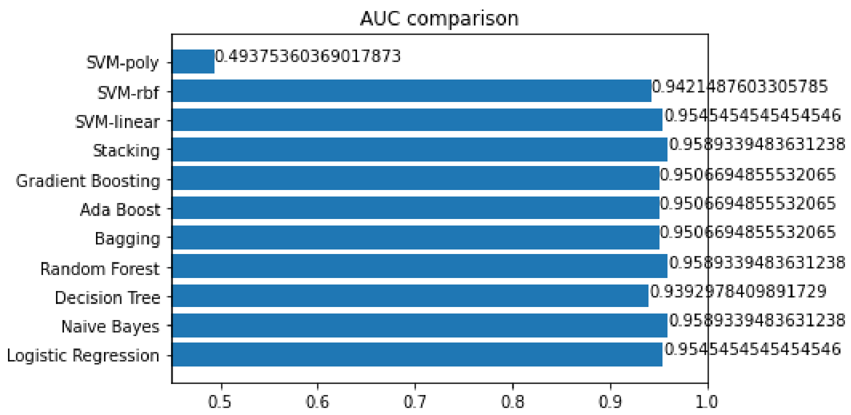
[162]:

	Classifier	Accuracy	Precision	Recall	Specificity	AUC
0	Logistic Regression	0.956	0.909091	1.000000	0.921429	0.954545
1	Naive Bayes	0.960	0.925620	0.991150	0.934307	0.958934
2	Decision Tree	0.940	0.917355	0.956897	0.925373	0.939298
3	Random Forest	0.960	0.925620	0.991150	0.934307	0.958934
4	Bagging	0.952	0.909091	0.990991	0.920863	0.950669
5	Ada Boost	0.952	0.909091	0.990991	0.920863	0.950669
6	Gradient Boosting	0.952	0.909091	0.990991	0.920863	0.950669
7	Stacking	0.960	0.925620	0.991150	0.934307	0.958934
8	SVM-linear	0.956	0.909091	1.000000	0.921429	0.954545
9	SVM-rbf	0.944	0.884298	1.000000	0.902098	0.942149
10	SVM-poly	0.504	0.173554	0.466667	0.512195	0.493754

```
[176]: plt.barh(summary.iloc[:, 0], summary.iloc[:, 1])
       plt.xlim([0.45, 1.0])
       plt.title('Accuracy comparison')
       for ind, value in enumerate(summary.iloc[:, 1]):
           plt.text(value, ind, value)
       plt.show()
```




```
[177]: plt.barh(summary.iloc[:, 0], summary.iloc[:, 5])
       plt.xlim([0.45, 1.0])
       plt.title('AUC comparison')
       for ind, value in enumerate(summary.iloc[:, 5]):
           plt.text(value, ind, value)
       plt.show()
```



CONCLUSION:

We have tried different classifications methods. Based on the experimented results, we can say that the Random Forest has the highest AUC score 0.9589.

Experiment – 12

AIM: To write a program to demonstrate k-means, k-nearest

DESCRIPTION:
K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these.

PROGRAM:

1. K-Means

```
[11]: import pandas as pd
      import numpy as np
      import matplotlib.pyplot as plt
```

```
[3]: df = pd.read_csv('clustering.csv')
     df
```

t[3]:

	x	y
0	2	10
1	2	5
2	8	4
3	5	8
4	5	7
5	6	4
6	1	2
7	4	9

```
[5]: from sklearn.cluster import KMeans
```

```
[6]: X = df.values
     X
```

```
t[6]: array([[ 2, 10],
             [ 2,  5],
             [ 8,  4],
             [ 5,  8],
             [ 5,  7],
             [ 6,  4],
             [ 1,  2],
             [ 4,  9]])
```

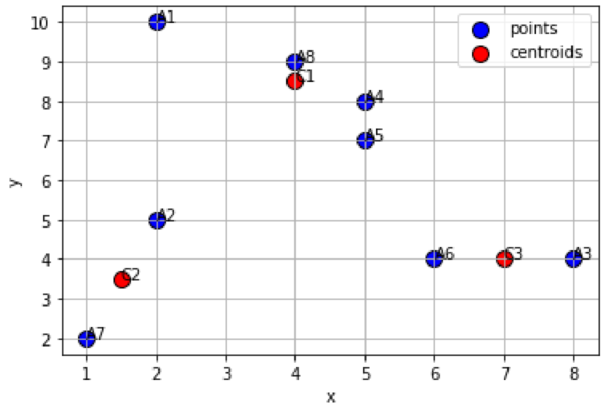
```
[8]: kmeans = KMeans(n_clusters=3, random_state=0)
     kmeans.fit(X)
```

```
[8]: KMeans(n_clusters=3, random_state=0)
```

```
10]: centroids = kmeans.cluster_centers_
     centroids
```

```
10]: array([[4. , 8.5],
            [1.5, 3.5],
            [7. , 4. ]])
```

```
[51]: fig, ax = plt.subplots()

      ax.scatter(
          X[:, 0], X[:, 1],
          s=100,c='blue', edgecolor='black',
          label='points'
      )

      ax.scatter(
          kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
          s=100,c='red', edgecolor='black',
          label='centroids'
      )

      pl = ['A1','A2','A3','A4','A5','A6','A7','A8']
      for i, txt in enumerate(pl):
          ax.annotate(txt, (X[i,0], X[i,1]))

      cl = ['C1','C2','C3']
      for i, txt in enumerate(cl):
          ax.annotate(txt, (kmeans.cluster_centers_[i, 0], kmeans.cluster_centers_[i, 1]))

      plt.xlabel('x')
      plt.ylabel('y')
      plt.legend()
      plt.grid()
      plt.show()
```



2. Nearest Neighbors

[11]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[3]:
```
df = pd.read_csv('clustering.csv')
df
```

t[3]:

	x	y
0	2	10
1	2	5
2	8	4
3	5	8
4	5	7
5	6	4
6	1	2
7	4	9

[5]:
```
from sklearn.cluster import KMeans
```

[6]:
```
X = df.values
X
```

t[6]:
```
array([[ 2, 10],
       [ 2,  5],
       [ 8,  4],
       [ 5,  8],
       [ 5,  7],
       [ 6,  4],
       [ 1,  2],
       [ 4,  9]])
```

:
```
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=3)
neigh.fit(X)
```

[38]:
```
NearestNeighbors(n_neighbors=3)
```

:
```
x_test = [[3, 7]]
print(neigh.kneighbors(x_test))
```

```
(array([[2.        , 2.23606798, 2.23606798]]), array([[4, 3, 1]], dtype=int64))
```

Experiment – 13

AIM: To write a program to demonstrate Agglomerative and DBSCAN

DESCRIPTION:

Agglomerative:

It is a type of Hierarchical clustering. Initially consider every data point as an individual Cluster and at every step, merge the nearest pairs of the cluster. (It is a bottom-up method). At first every data set is considered as individual entity or cluster. At every iteration, theclusters merge with different clusters until one cluster is formed.

DBSCAN:

Clusters are dense regions in the data space, separated by regions of the lower density of points. The **DBSCAN algorithm** is based on this intuitive notion of "clusters" and "noise". The key idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points

PROGRAM:

1. Agglomerative

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

```
df = pd.read_csv('clustering.csv')
df
```

3]:

	x	y
0	2	10
1	2	5
2	8	4
3	5	8
4	5	7
5	6	4
6	1	2
7	4	9

```
X = df.values
```

```
from sklearn.cluster import AgglomerativeClustering

model= AgglomerativeClustering(n_clusters=3, affinity='euclidean')
model.fit(df)
```

15]: AgglomerativeClustering(n_clusters=3)

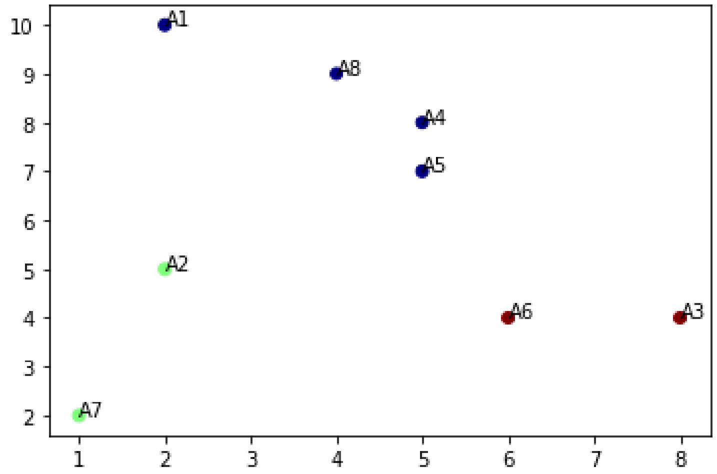
```
df['cluster'] = model.labels_
df
```

16]:

	x	y	cluster
0	2	10	0
1	2	5	1
2	8	4	2
3	5	8	0
4	5	7	0
5	6	4	2
6	1	2	1
7	4	9	0

```
fig, ax = plt.subplots()

pl = ['A1','A2','A3','A4','A5','A6','A7','A8']
for i, txt in enumerate(pl):
    ax.annotate(txt, (X[i,0], X[i,1]))

ax.scatter(df['x'], df['y'], c = df['cluster'], cmap='jet')
```

.7]: <matplotlib.collections.PathCollection at 0x23e9c041f40>



2. DBSCAN

']:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

:]:
```
df = pd.read_csv('clustering.csv')
df
```

t[18]:

	x	y
0	2	10
1	2	5
2	8	4
3	5	8
4	5	7
5	6	4
6	1	2
7	4	9

']:
```
X = df.values
```

```
from sklearn.cluster import DBSCAN

model = DBSCAN(eps=3, min_samples=2)
model.fit(X)
```

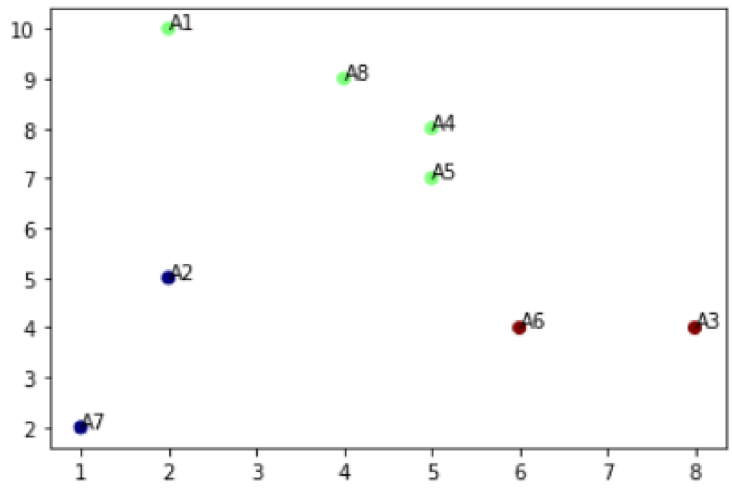
20]:
```
DBSCAN(eps=3, min_samples=2)
```

```
df['cluster'] = model.labels_
model.labels_
```

21]:
```
array([ 0, -1,  1,  0,  0,  1, -1,  0], dtype=int64)
```

```
fig, ax = plt.subplots()

pl = ['A1','A2','A3','A4','A5','A6','A7','A8']
for i, txt in enumerate(pl):
    ax.annotate(txt, (X[i,0], X[i,1]))

ax.scatter(df['x'], df['y'], c = df['cluster'], cmap='jet')
```

28]: <matplotlib.collections.PathCollection at 0x21c8ec02820>



Experiment – 14

AIM: To do a case study on clustering methods

DESCRIPTION:

UNSUPERVISED LEARNING ALGORITHM

1. KMEANS Algorithm
   - Use Medical Expenses dataset from folder and try to form clusters using K means Algorithm.
   - Figure out if any preprocessing such as scaling would help.
   - Draw elbow plot and from that figure out optimal value of k.

2. Hierarchical Algorithm
   - Use Medical Expenses dataset from folder and try to form clusters using Hierarchical algorithm.
   - Figure out if any preprocessing such as scaling would help.
   - Draw dendrogram for different linkage methods like single (min), complete (max), ward and from that figure out the number of clusters.
   - Draw a comparison table on different linkage metrics.

3. DBSCAN
   - Use Medical Expenses dataset from folder and try to form clusters using DBSCAN Algorithm.
   - Figure out if any preprocessing such as scaling would help.
   - Draw knee plot and from that figure out optimal value of epsilon and minimum point.

CLUSTERING

Clustering is one of the most common exploratory data analysis technique used to get an intuition about the structure of the data. It can be defined as the task of identifying subgroups in the data such that data points in the same subgroup (cluster) are very similar while data points in different clusters are very different. In other words, we try to find homogeneous subgroups within the data such that datapoints in each cluster are as similar as possible according to a similarity measure such as Euclidean-based distance or correlation-based distance. The decision of which similarity measure to use is application specific.

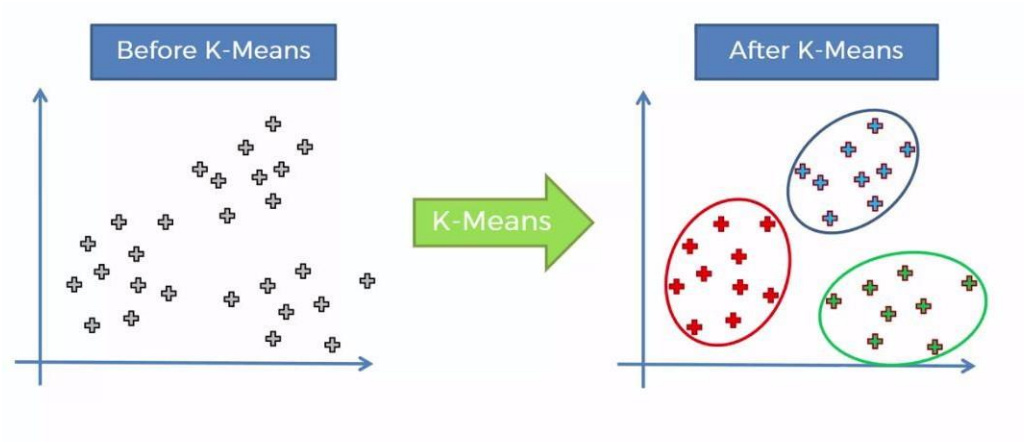
Unlike supervised learning, clustering is considered an unsupervised learning method since we do not have the ground truth to compare the output of the clustering algorithm to the true labels to evaluate its performance. We only want to try to investigate the structure of the data by grouping the data points into distinct subgroups.

KMEANS ALGORITHM

K-means algorithm is an iterative algorithm that tries to partition the dataset into Kpre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the intra-cluster data points as similar aspossible while also keeping the clusters as different (far) as possible. It assigns datapoints to a cluster such that the sum of the squared distance between the data pointsand the cluster’s centroid (arithmetic mean of all the data points that belong to thatcluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

The way k-means algorithm works is as follows:

1. Specify number of clusters *K*.
2. Initialize centroids by first shuffling the dataset and then randomly selecting *K* data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e., assignment of datapoints to clusters is not changing.
    - Compute the sum of the squared distance between data points and allcentroids.
    - Assign each data point to the closest cluster (centroid).
    - Compute the centroids for the clusters by taking the average of all data pointsthat belong to each cluster.



Contrary to supervised learning where we have the ground truth to evaluate the model’s performance, clustering analysis does not have a solid evaluation metric that we can use to evaluate the outcome of different clustering algorithms. Moreover,since k-means requires k as an input and does not learn it from data, there is no right answer in terms of the number of clusters that we should have in any problem.

Two metrics that may give us some intuition about k are :

- Elbow method
- Silhouette analysis

HIERARCHICAL ALGORITHM

A Hierarchical clustering method works via grouping data into a tree of clusters. Hierarchical clustering begins by treating every data point as a separate cluster. Then, it repeatedly executes the subsequent steps:

- Identify the 2 clusters which can be closest together, and
- Merge the 2 maximum comparable clusters. We need to continue these stepsuntil all the clusters are merged.

In Hierarchical Clustering, the aim is to produce a hierarchical series of nested clusters. A diagram called Dendrogram (A Dendrogram is a tree-like diagram that statistics the sequences of merges or splits) graphically represents this hierarchy and is an inverted tree that describes the order in which factors are merged (bottom-up view) or cluster are break up (top-down view).

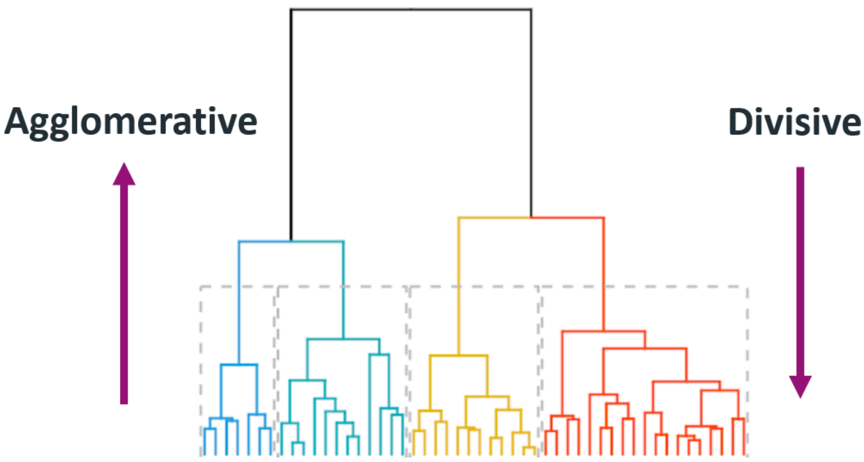
The basic method to generate hierarchical clustering are:

1. **Agglomerative:**

   Initially consider every data point as an individual Cluster and at every step, merge the nearest pairs of the cluster. (It is a bottom-up method). At first every data set is considered as individual entity or cluster. At every iteration, theclusters merge with different clusters until one cluster is formed.

2. **Divisive:**

   We can say that the Divisive Hierarchical clustering is precisely the opposite of the Agglomerative Hierarchical clustering. In Divisive Hierarchical clustering, we consider all the data points as a single cluster and in every iteration, we separatethe data points from the clusters which aren't comparable. In the end, we are leftwith N clusters.



DBSCAN

Density-Based Clustering refers to unsupervised learning methods that identifydistinctive groups/clusters in the data, based on the idea that a cluster in data spaceis a contiguous region of high point density, separated from other such clusters bycontiguous regions of low point density.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a base algorithm for density-based clustering. It can discover clusters of different shapes and sizes from a large amount of data, which is containing noise and outliers.

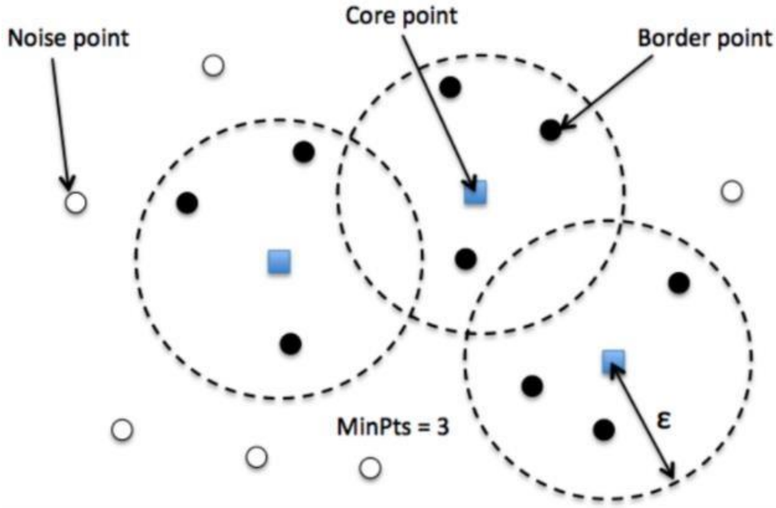
The DBSCAN algorithm uses two parameters:

- minPts: The minimum number of points (a threshold) clustered together for aregion to be considered dense.
- eps (ε): A distance measure that will be used to locate the points about any

  point.

These parameters can be understood if we explore two concepts called Density Reachability and Density Connectivity. Reachability in terms of density establishes apoint to be reachable from another if it lies within a particular distance (eps) from it.Connectivity, on the other hand, involves a transitivity-based chaining-approach todetermine whether points are in a particular cluster. For example, p and q points could be connected if p->r->s->t->q, where a->b means b is in the neighbourhood ofa.

There are three types of points after the DBSCAN clustering is complete:



- Core — This is a point that has at least m points within distance n from itself.
- Border — This is a point that has at least one Core point at a distance n.
- Noise — This is a point that is neither a Core nor a Border. And it has less thanm points within distance n from itself.

PROGRAM

IMPORTING

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
df=pd.read_csv("medExpenses.csv")
```

UNDERSTANDING THE DATASET

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

Out[3]: (33, 3)

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

Out[4]:

	Unnamed: 0	familysize	expenses
0	1	2	7.15
1	2	3	6.93
2	3	3	7.57
3	4	5	6.10
4	5	4	10.30

In [5]:
```
df=df.drop("Unnamed: 0",axis=1)
```

In [6]:
```
df.head()
```

Out[6]:

	familysize	expenses
0	2	7.15
1	3	6.93
2	3	7.57
3	5	6.10
4	4	10.30

In [8]:
```
df=df.reindex(columns=['expenses','familysize'])
```

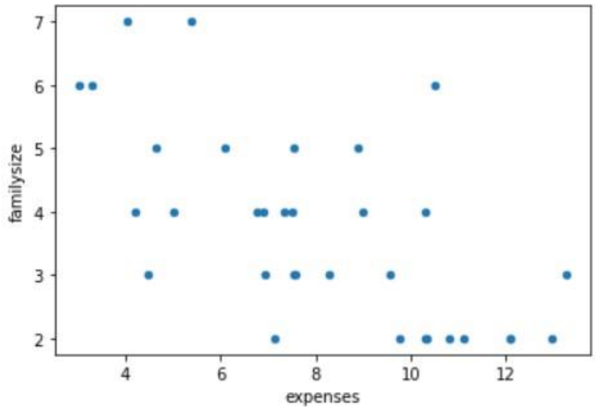
In [9]:
```
df.head()
```

Out[9]:

	expenses	familysize
0	7.15	2
1	6.93	3
2	7.57	3
3	6.10	5
4	10.30	4

```
In [10]: df.plot.scatter(x="expenses",y="familysize")
```

Out[10]: <AxesSubplot:xlabel='expenses', ylabel='familysize'>



SCALING THE DATA

```
In [11]: from sklearn.preprocessing import StandardScaler
```

```
In [12]: sc=StandardScaler()
         scaled_df=sc.fit_transform(df)
         df=pd.DataFrame(scaled_df,columns=["expenses","familysize"])
```

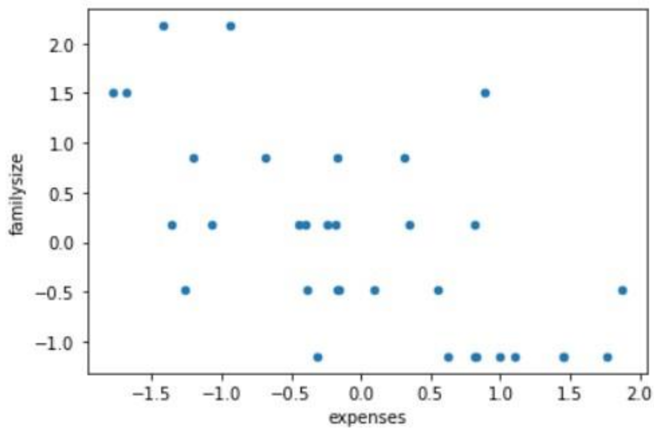
```
In [13]: df.head()
```

Out[13]:

	expenses	familysize
0	-0.311503	-1.149231
1	-0.390083	-0.483887
2	-0.161488	-0.483887
3	-0.686541	0.846802
4	0.813610	0.181458

```
In [14]: df.plot.scatter(x="expenses",y="familysize")
```

Out[14]: <AxesSubplot:xlabel='expenses', ylabel='familysize'>



KMEANS Algorithm

In [15]:
```
from sklearn.cluster import KMeans
model=KMeans(n_clusters=2)
model.fit(df)
```

Out[15]:
```
KMeans(n_clusters=2)
```

In [16]:
```
model.inertia_
# Inertia: Sum of distances of samples to their closest cluster center
```

Out[16]:
```
29.348159372311844
```

In [17]:
```
model=KMeans(n_clusters=3)
model.fit(df)
```

Out[17]:
```
KMeans(n_clusters=3)
```

In [18]:
```
model.inertia_
```

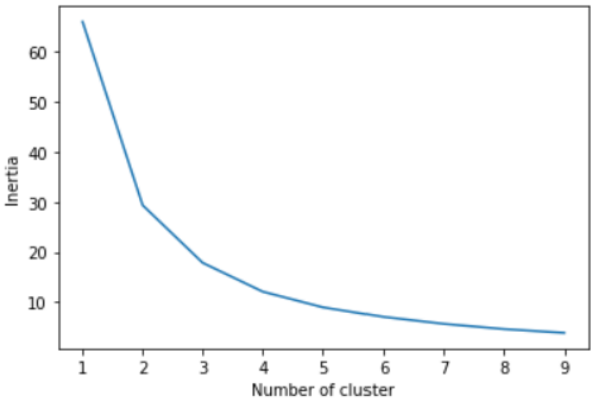
Out[18]:
```
17.833498987534917
```

In [19]:
```
sse={}
for k in range(1, 10):
    model=KMeans(n_clusters=k)
    model.fit(df)
    sse[k]=model.inertia_
```

In [20]:
```
sse
```

Out[20]:
```
{1: 66.0,
 2: 29.348159372311844,
 3: 17.833498987534917,
 4: 12.068806683169074,
 5: 8.943908378480092,
 6: 7.015767176966719,
 7: 5.628275420524087,
 8: 4.588835793812496,
 9: 3.83937146598356}
```

In [21]:
```
plt.figure()
plt.plot(list(sse.keys()),list(sse.values()))
plt.xlabel("Number of cluster")
plt.ylabel("Inertia")
plt.show()
```



In [22]:
```
model=KMeans(n_clusters=3)
model.fit(df)
predicted=model.predict(df)
```

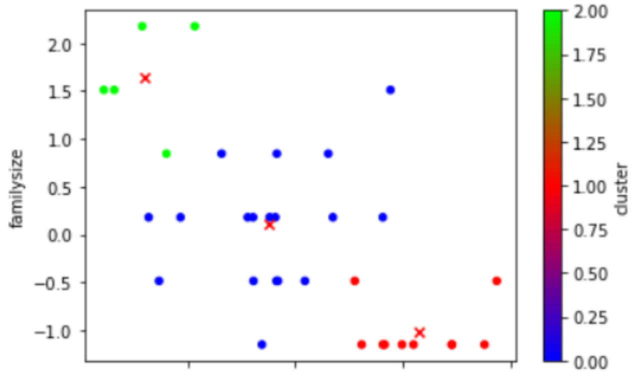
In [23]:
```
df['cluster']=predicted
print(df['cluster'].value_counts())
```

```
0    18
1    10
2     5
Name: cluster, dtype: int64
```

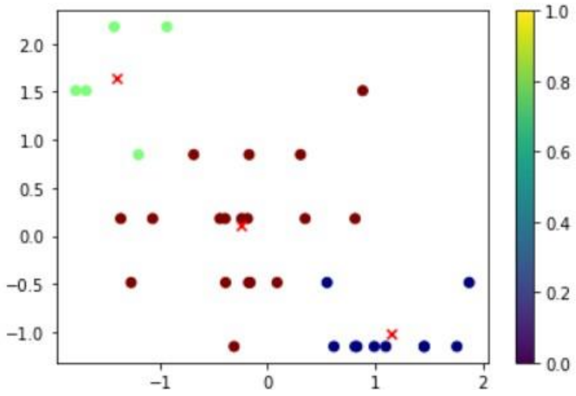
In [25]:
```
centers=model.cluster_centers_
print(centers)
```

```
[[-0.24562365  0.10753038]
 [ 1.14542952 -1.01616208]
 [-1.4066139   1.64521479]]
```

In [26]:
```
df.plot.scatter(x="expenses",y="familysize",c='cluster',colormap='brg')
plt.scatter(centers[:,0],centers[:,1],marker="x",color='r')
plt.show()
```



In [27]:
```
x=df["expenses"]
y=df["familysize"]
c=df["cluster"]

plt.scatter(x,y,c=c,cmap="jet")
plt.scatter(centers[:,0],centers[:,1],marker="x",color='r')
plt.colorbar()
plt.show()
```



Hierarchical Algorithm

In [29]:
```
df.head()
```

Out[29]:

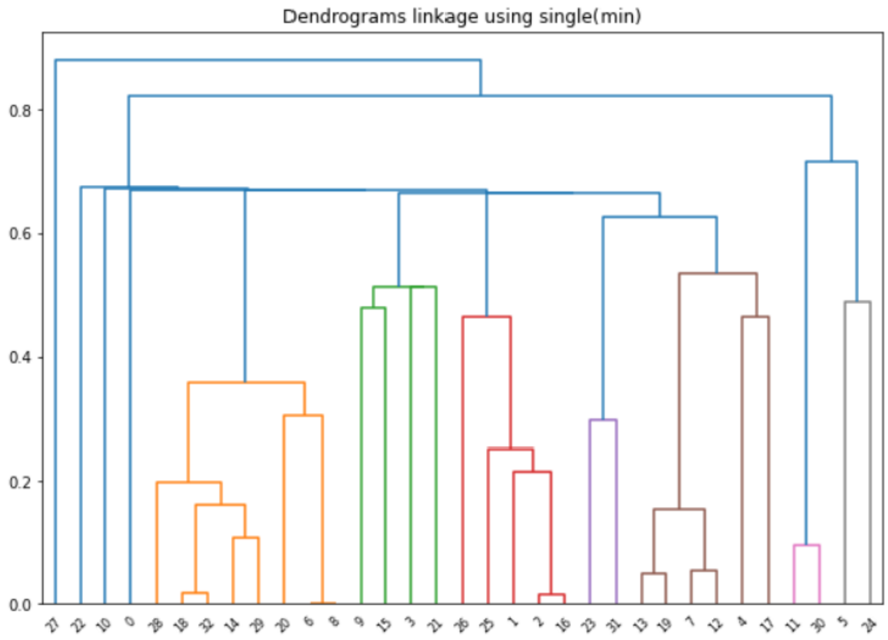
	expenses	familysize	cluster
0	-0.311503	-1.149231	0
1	-0.390083	-0.483887	0
2	-0.161488	-0.483887	0
3	-0.686541	0.846802	0
4	0.813610	0.181458	0

In [30]:
```
df=df.drop("cluster",axis=1)
df.head()
```

Out[30]:

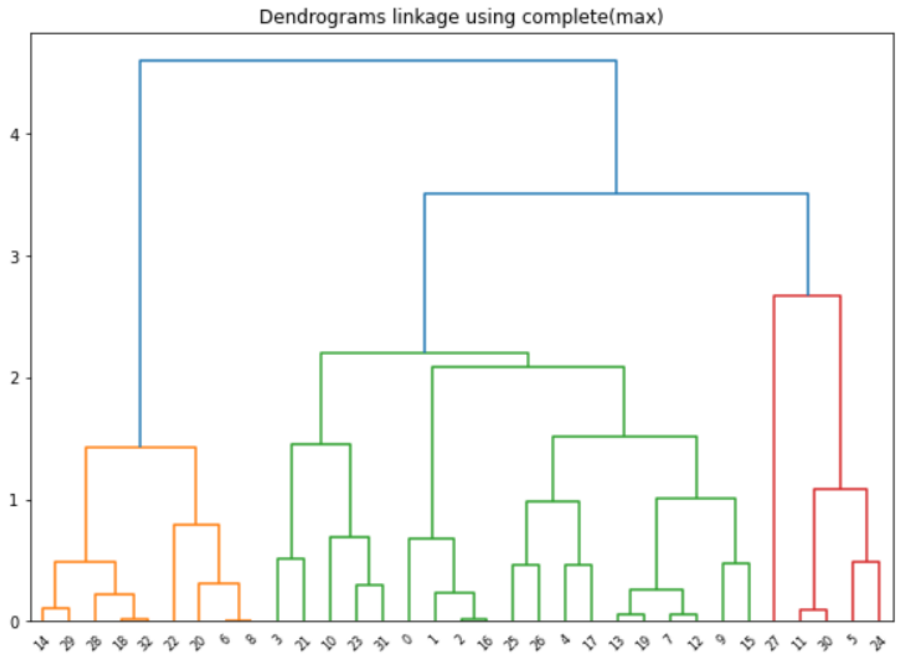
	expenses	familysize
0	-0.311503	-1.149231
1	-0.390083	-0.483887
2	-0.161488	-0.483887
3	-0.686541	0.846802
4	0.813610	0.181458

In [31]:
```
import scipy.cluster.hierarchy as shc
plt.figure(figsize=(10, 7))
plt.title("Dendrograms linkage using single(min)")
dend = shc.dendrogram(shc.linkage(df, method='single'))
```



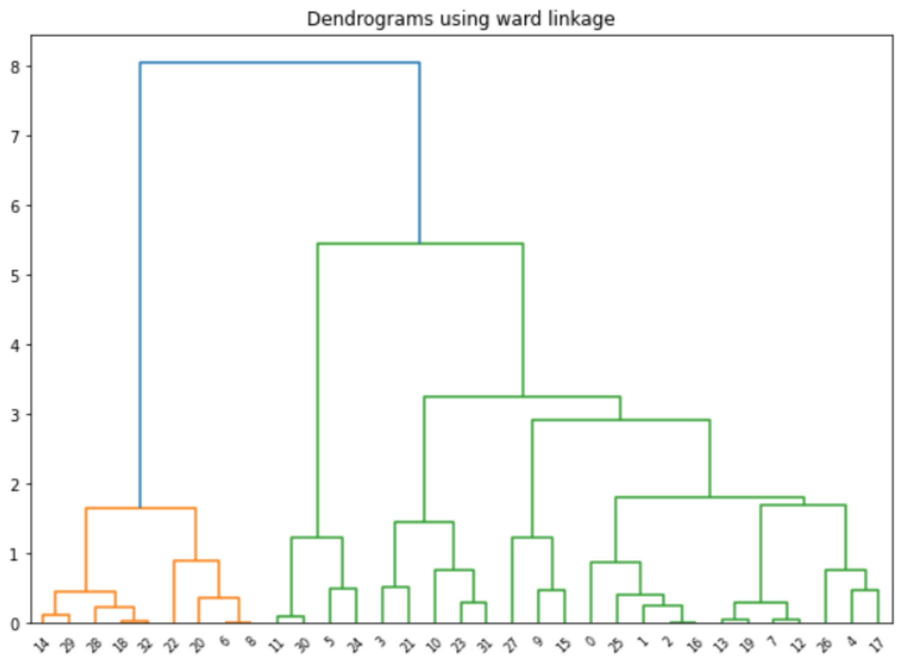
In [32]:

```
plt.figure(figsize=(10, 7))
plt.title("Dendrograms linkage using complete(max)")
dend = shc.dendrogram(shc.linkage(df, method='complete'))
```



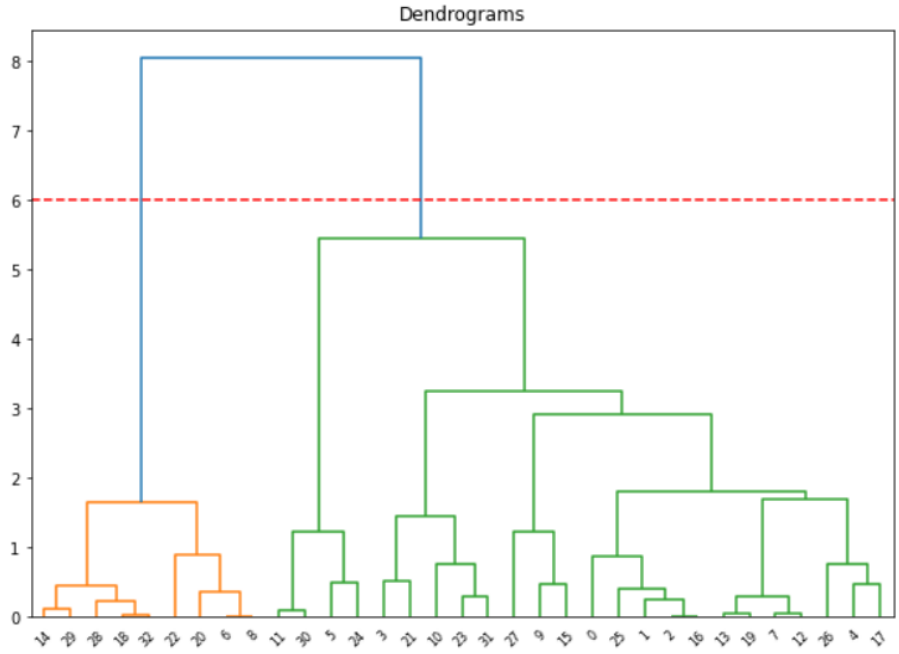
In [33]:

```
plt.figure(figsize=(10, 7))
plt.title("Dendrograms using ward linkage")
model=shc.dendrogram(shc.linkage(df,method='ward'))
```



The x-axis contains the samples and y-axis represents the distance between thesesamples. The vertical line with maximum distance is the blue line and hence we candecide a threshold of 6 and cut the dendrogram

In [34]:
```
plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
plt.axhline(y=6,color='r',linestyle='--')
dend = shc.dendrogram(shc.linkage(df,method='ward'))
```



We have two clusters as this line cuts the dendrogram at two points. Let us nowapply hierarchical clustering for 2 clusters.

Agglomerative Clustering

In [35]:
```
from sklearn.cluster import AgglomerativeClustering
model=AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
model.fit(df)
```

Out[35]: AgglomerativeClustering()

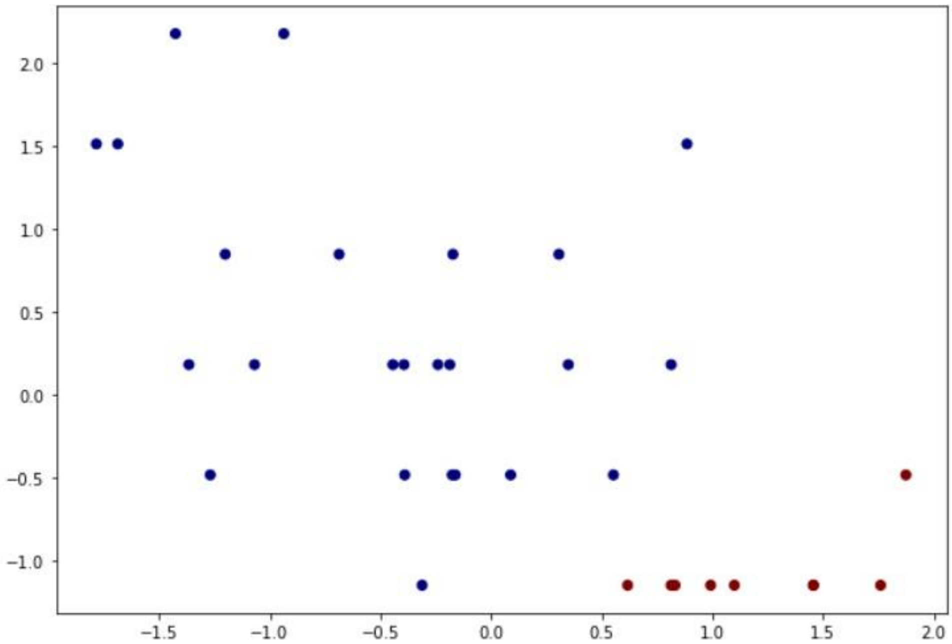
In [36]:
```
df['cluster']=model.fit_predict(df)
```

In [37]:
```
df.head()
```

Out[37]:

	expenses	familysize	cluster
0	-0.311503	-1.149231	0
1	-0.390083	-0.483887	0
2	-0.161488	-0.483887	0
3	-0.686541	0.846802	0
4	0.813610	0.181458	0

```
In [38]: plt.figure(figsize=(10, 7))
         plt.scatter(df['expenses'], df['familysize'],c=df['cluster'],cmap='jet')

Out[38]: <matplotlib.collections.PathCollection at 0x24f813502b0>
```



Dendrograms cannot tell us how many clusters we should have

A common mistake people make when reading dendrograms is to assume that the shape of the dendrogram gives a clue as to how many clusters exist. In the example above, the (incorrect) interpretation is that the dendrogram shows that there are two clusters, as the distance between the clusters (the vertical segments of the dendrogram) are highest between two and three clusters.

Interpretation of this kind is justified only when the ultra-metric tree inequality holds, which, as mentioned above, is very rare. In general, it is a mistake to use dendrograms as a tool for determining the number of clusters in data. Where thereis an obviously "correct" number of clusters this will often be evident in a dendrogram. However, dendrograms often suggest a correct number of clusters when there is no real evidence to support the conclusion.

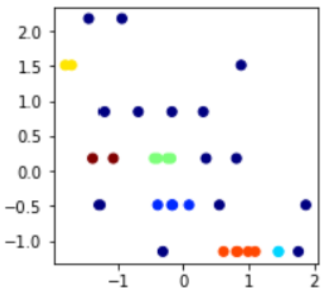
DBSCAN

In [39]:
```
df.head()
```
Out[39]:

	expenses	familysize	cluster
0	-0.311503	-1.149231	0
1	-0.390083	-0.483887	0
2	-0.161488	-0.483887	0
3	-0.686541	0.846802	0
4	0.813610	0.181458	0

In [40]:
```
df=df.drop('cluster',axis=1)
df.head()
```
Out[40]:

	expenses	familysize
0	-0.311503	-1.149231
1	-0.390083	-0.483887
2	-0.161488	-0.483887
3	-0.686541	0.846802
4	0.813610	0.181458

In [41]:
```
from sklearn.cluster import DBSCAN
model=DBSCAN(eps=0.3,min_samples=2)
model.fit(df)
```
Out[41]:
```
DBSCAN(eps=0.3, min_samples=2)
```
In [42]:
```
model.labels_
```
Out[42]:
```
array([-1,  0,  0, -1, -1, -1,  1,  2,  1, -1, -1,  3,  2,  2,  4, -1,  0,
       -1,  4,  2, -1, -1, -1,  5, -1,  0, -1, -1,  4,  4,  3,  5,  4],
      dtype=int64)
```
In [43]:
```
df['cluster']=model.labels_
```
In [44]:
```
plt.figure(figsize=(3,3))
plt.scatter(df['expenses'], df['familysize'],c=df['cluster'],cmap='jet')
```
Out[44]:
```
<matplotlib.collections.PathCollection at 0x24f812b4e80>
```


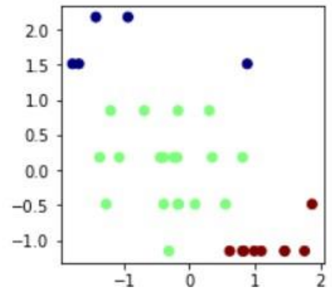
In [45]:
```
df=df.drop('cluster',axis=1)
df.head()
```

Out[45]:

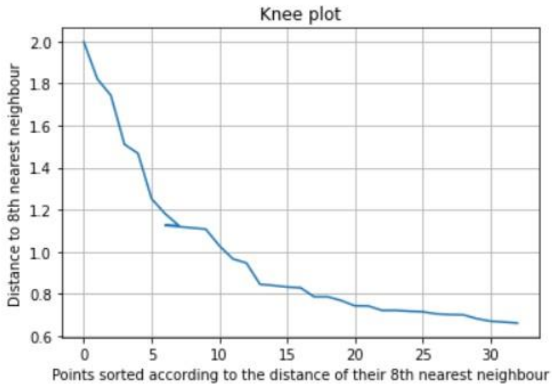
	expenses	familysize
0	-0.311503	-1.149231
1	-0.390083	-0.483887
2	-0.161488	-0.483887
3	-0.686541	0.846802
4	0.813610	0.181458

In [47]:
```
model=DBSCAN(eps=0.7,min_samples=5)
model.fit(df)
df['cluster']=model.labels_
plt.figure(figsize=(3,3))
plt.scatter(df['expenses'], df['familysize'],c=df['cluster'],cmap='jet')
```

Out[47]: <matplotlib.collections.PathCollection at 0x24f811682b0>



In [46]:
```
from sklearn.neighbors import NearestNeighbors
ns=8
nbrs=NearestNeighbors(n_neighbors=ns)
nbrs.fit(df)
distances,indices=nbrs.kneighbors(df)
distances=sorted(distances[:,ns-1],reverse=True)
plt.plot(indices[:,0],distances)
plt.xlabel('Points sorted according to the distance of their 8th nearest neighbour')
plt.ylabel('Distance to 8th nearest neighbour')
plt.title('Knee plot')
plt.grid()
```



CONCLUSION

From the above experiment, we come to know the different ways to form clusters from the given Medical Expenses dataset. Data Scaling also plays a huge role in theformation of the clusters properly.