Apriori Algorithm --weak5

Step 1- Import required libraries

```
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.special import comb
from itertools import combinations, permutations
from apyori import apriori as apr
from mlxtend.frequent_patterns import apriori, association_rules
import scipy as sp
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import seaborn as sns
from mlxtend.preprocessing import TransactionEncoder
```

```
C:\Users\dodda\anaconda3\lib\site-packages\seaborn\rcmod.py:82:
DeprecationWarning: distutils Version classes are deprecated. Use
packaging.version instead.
  if LooseVersion(mpl.__version__) >= "3.0":
C:\Users\dodda\anaconda3\lib\site-packages\setuptools\_distutils\
version.py:351: DeprecationWarning: distutils Version classes are
deprecated. Use packaging.version instead.
  other = LooseVersion(other)
```

```
pip install apyori
```

```
Collecting apyori
  Downloading apyori-1.1.2.tar.gz (8.6 kB)
Building wheels for collected packages: apyori
  Building wheel for apyori (setup.py): started
  Building wheel for apyori (setup.py): finished with status 'done'
  Created wheel for apyori: filename=apyori-1.1.2-py3-none-any.whl
size=5974
sha256=1a8c9f8e462d6e6584ff344863721dbab45fe212311bcfd3ca8d1917e7b1933
5
  Stored in directory: c:\users\dodda\appdata\local\pip\cache\wheels\
32\2a\54\10c595515f385f3726642b10c60bf788029e8f3a1323e3913a
Successfully built apyori
Installing collected packages: apyori
Successfully installed apyori-1.1.2
Note: you may need to restart the kernel to use updated packages.
```

```
df = pd.read_csv("5_my_movies.csv")
df
```

```
              V1             V2             V3          V4     V5  \
0    Sixth Sense          LOTR1  Harry Potter1  Green Mile  LOTR2
1      Gladiator        Patriot     Braveheart         NaN    NaN
2          LOTR1          LOTR2            NaN         NaN    NaN
3      Gladiator        Patriot    Sixth Sense         NaN    NaN
4      Gladiator        Patriot    Sixth Sense         NaN    NaN
5      Gladiator        Patriot    Sixth Sense         NaN    NaN
6  Harry Potter1  Harry Potter2            NaN         NaN    NaN
7      Gladiator        Patriot            NaN         NaN    NaN
8      Gladiator        Patriot    Sixth Sense         NaN    NaN
9    Sixth Sense           LOTR      Gladiator  Green Mile    NaN

   Sixth Sense  Gladiator  LOTR1  Harry Potter1  Patriot  LOTR2  \
0            1          0      1              1        0      1
1            0          1      0              0        1      0
2            0          0      1              0        0      1
3            1          1      0              0        1      0
4            1          1      0              0        1      0
5            1          1      0              0        1      0
6            0          0      0              1        0      0
7            0          1      0              0        1      0
8            1          1      0              0        1      0
9            1          1      0              0        0      0

   Harry Potter2  LOTR  Braveheart  Green Mile
0              0     0           0           1
1              0     0           1           0
2              0     0           0           0
3              0     0           0           0
4              0     0           0           0
5              0     0           0           0
6              1     0           0           0
7              0     0           0           0
8              0     0           0           0
9              0     1           0           1
```

Step 2- Load, visualize and explore the dataset

```
df1 = df.iloc[:,5:]
df1.head()
```

```
   Sixth Sense  Gladiator  LOTR1  Harry Potter1  Patriot  LOTR2  \
0            1          0      1              1        0      1
1            0          1      0              0        1      0
2            0          0      1              0        0      1
3            1          1      0              0        1      0
4            1          1      0              0        1      0

   Harry Potter2  LOTR  Braveheart  Green Mile
```

```
0                0     0           0          1
1                0     0           1          0
2                0     0           0          0
3                0     0           0          0
4                0     0           0          0

df1 = df.iloc[:,5:]
df1.head()

   Sixth Sense  Gladiator  LOTR1  Harry Potter1  Patriot  LOTR2  \
0            1          0      1              1        0      1
1            0          1      0              0        1      0
2            0          0      1              0        0      1
3            1          1      0              0        1      0
4            1          1      0              0        1      0

   Harry Potter2  LOTR  Braveheart  Green Mile
0              0     0           0           1
1              0     0           1           0
2              0     0           0           0
3              0     0           0           0
4              0     0           0           0
```

Step 3- Clean the data set

```
df1.isnull().sum()

Sixth Sense      0
Gladiator        0
LOTR1            0
Harry Potter1    0
Patriot          0
LOTR2            0
Harry Potter2    0
LOTR             0
Braveheart       0
Green Mile       0
dtype: int64

df1

   Sixth Sense  Gladiator  LOTR1  Harry Potter1  Patriot  LOTR2  \
0            1          0      1              1        0      1
1            0          1      0              0        1      0
2            0          0      1              0        0      1
3            1          1      0              0        1      0
4            1          1      0              0        1      0
5            1          1      0              0        1      0
6            0          0      0              1        0      0
7            0          1      0              0        1      0
```

```
8           1          1      0              0        1      0
9           1          1      0              0        0      0

   Harry Potter2  LOTR  Braveheart  Green Mile
0              0     0           0           1
1              0     0           1           0
2              0     0           0           0
3              0     0           0           0
4              0     0           0           0
5              0     0           0           0
6              1     0           0           0
7              0     0           0           0
8              0     0           0           0
9              0     1           0           1

#Setting different thresholds
confidence = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
```

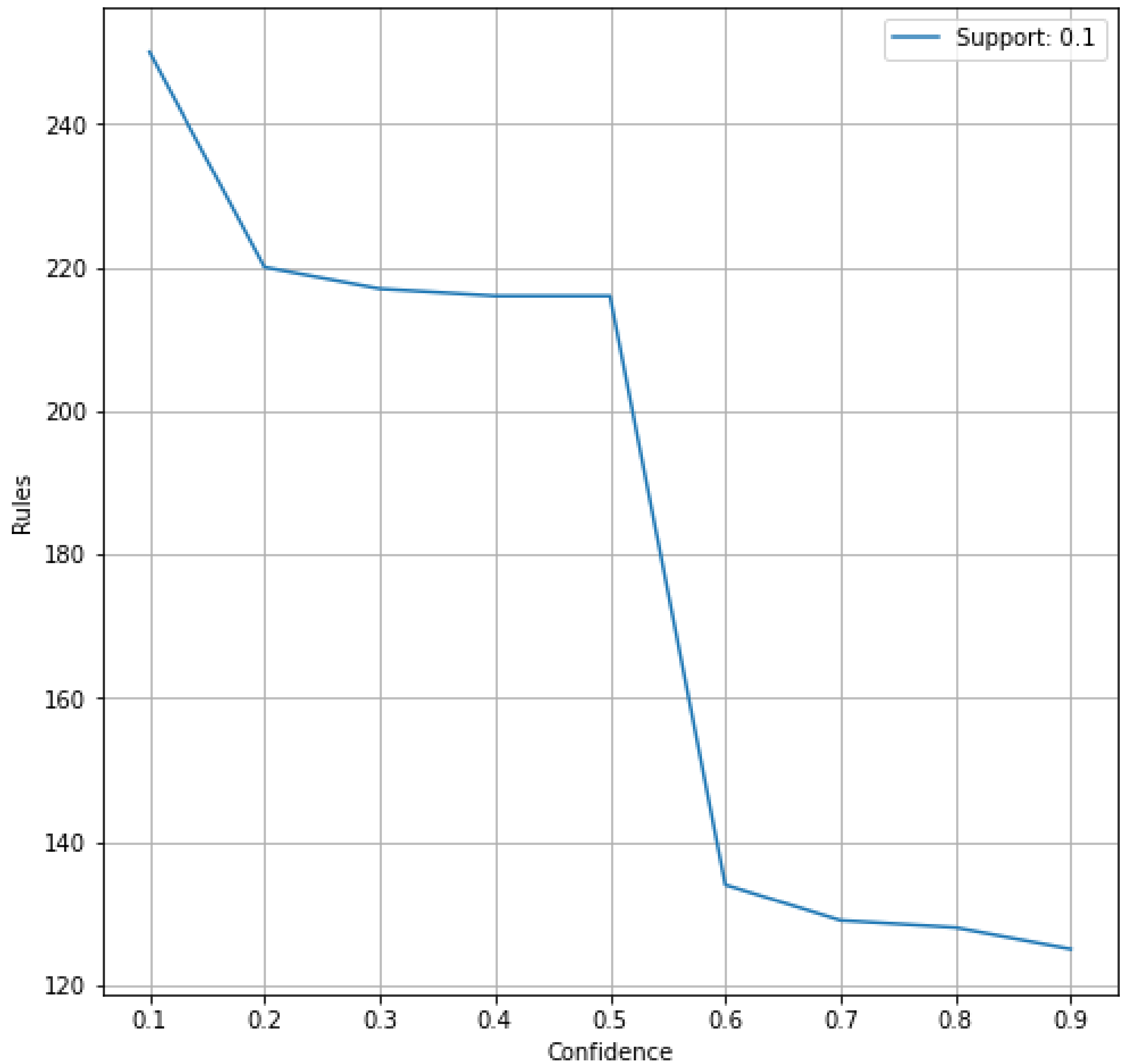
Step - 4 Generating Association Rules

```
def gen_rules(df,confidence,support):
    ap = {}
    for i in confidence:
        ap_i =apriori(df1,support,True)
        rule= association_rules(ap_i,min_threshold=i)
        ap[i] = len(rule.antecedents)
    return pd.Series(ap).to_frame("Support: %s"%support)

confs = []
ap_i = gen_rules(df1,confidence=confidence,support=0.1)
confs.append(ap_i)
all_conf = pd.concat(confs,axis=1)

C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
```

```
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
```

Step - 5 Visuvalizing Association Rules with different support and confidence thresholds

```
all_conf.plot(figsize=(8,8),grid=True)
plt.ylabel('Rules')
plt.xlabel('Confidence')
plt.show()
```

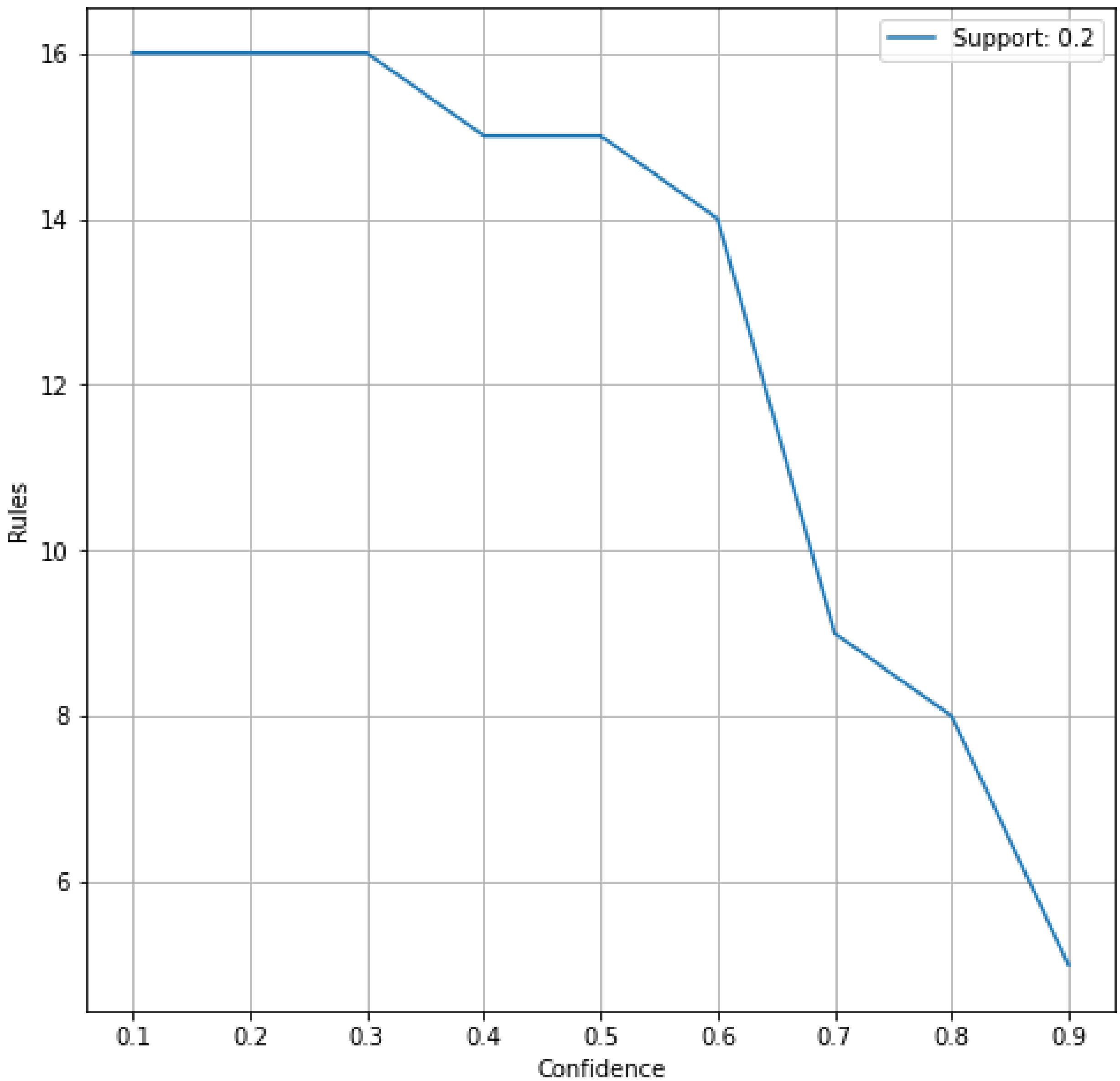



```
confs = []
ap_i = gen_rules(df1,confidence=confidence,support=0.2)
confs.append(ap_i)
all_conf = pd.concat(confs,axis=1)
```

```
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
```

```
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
C:\Users\dodda\anaconda3\lib\site-packages\mlxtend\frequent_patterns\
fpcommon.py:111: DeprecationWarning: DataFrames with non-bool types
result in worse computationalperformance and their support might be
discontinued in the future.Please use a DataFrame with bool type
  warnings.warn(
```

```
all_conf.plot(figsize=(8,8),grid=True)
plt.ylabel('Rules')
plt.xlabel('Confidence')
plt.show()
```



weak3-multiple linear regression-

```
# Importing Required files

import numpy as np
import pandas as pd
from sklearn import linear_model

# Importing the dataset
df = pd.read_csv("data.csv")

df
```

```
           Car       Model  Volume  Weight  CO2
0       Toyoty        Aygo    1000     790   99
1   Mitsubishi  Space Star    1200    1160   95
2        Skoda      Citigo    1000     929   95
3         Fiat         500     900     865   90
4         Mini      Cooper    1500    1140  105
5           VW         Up!    1000     929  105
6        Skoda       Fabia    1400    1109   90
7     Mercedes     A-Class    1500    1365   92
8         Ford      Fiesta    1500    1112   98
9         Audi          A1    1600    1150   99
10     Hyundai         I20    1100     980   99
11      Suzuki       Swift    1300     990  101
12        Ford      Fiesta    1000    1112   99
13       Honda       Civic    1600    1252   94
14      Hundai         I30    1600    1326   97
15        Opel       Astra    1600    1330   97
16         BMW           1    1600    1365   99
17       Mazda           3    2200    1280  104
18       Skoda       Rapid    1600    1119  104
19        Ford       Focus    2000    1328  105
20        Ford      Mondeo    1600    1584   94
21        Opel    Insignia    2000    1428   99
22    Mercedes     C-Class    2100    1365   99
23       Skoda     Octavia    1600    1415   99
24       Volvo         S60    2000    1415   99
25    Mercedes         CLA    1500    1465  102
26        Audi          A4    2000    1490  104
27        Audi          A6    2000    1725  114
28       Volvo         V70    1600    1523  109
29         BMW           5    2000    1705  114
30    Mercedes     E-Class    2100    1605  115
31       Volvo        XC70    2000    1746  117
32        Ford       B-Max    1600    1235  104
33         BMW         216    1600    1390  108
34        Opel      Zafira    1600    1405  109
35    Mercedes         SLK    2500    1395  120
```

```python
# selecting dependent and independent variables
X = df[['Weight', 'Volume']]
y = df['CO2']

X
```

```
   Weight  Volume
0     790    1000
1    1160    1200
2     929    1000
3     865     900
4    1140    1500
```

```
5      929    1000
6     1109    1400
7     1365    1500
8     1112    1500
9     1150    1600
10     980    1100
11     990    1300
12    1112    1000
13    1252    1600
14    1326    1600
15    1330    1600
16    1365    1600
17    1280    2200
18    1119    1600
19    1328    2000
20    1584    1600
21    1428    2000
22    1365    2100
23    1415    1600
24    1415    2000
25    1465    1500
26    1490    2000
27    1725    2000
28    1523    1600
29    1705    2000
30    1605    2100
31    1746    2000
32    1235    1600
33    1390    1600
34    1405    1600
35    1395    2500
```

y

```
0      99
1      95
2      95
3      90
4     105
5     105
6      90
7      92
8      98
9      99
10     99
11    101
12     99
13     94
14     97
15     97
```

```
16     99
17    104
18    104
19    105
20     94
21     99
22     99
23     99
24     99
25    102
26    104
27    114
28    109
29    114
30    115
31    117
32    104
33    108
34    109
35    120
Name: CO2, dtype: int64
```

```
#fitting the model
regr = linear_model.LinearRegression()
regr.fit(X, y)
```

```
LinearRegression()
```

```
#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300]])

print(predictedCO2)
```

```
[107.2087328]
```

```
C:\Users\dodda\anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

week2-logistic regression.-

```
# Importing Required files

import numpy as np
import pandas as pd
from sklearn import linear_model
```

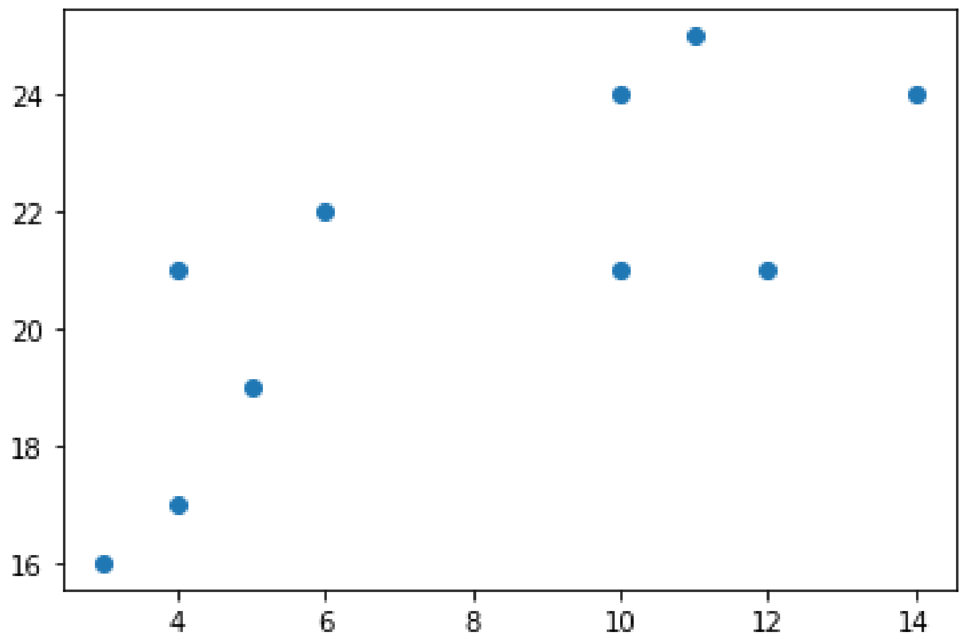
```
#X represents the size of a tumor in centimeters.
X = np.array([3.78, 2.44, 2.09, 0.14, 1.72, 1.65, 4.92, 4.37, 4.96,
4.52, 3.69, 5.88]).reshape(-1,1)

#Note: X has to be reshaped into a column from a row for the
LogisticRegression() function to work.
#y represents whether or not the tumor is cancerous (0 for "No", 1 for
"Yes").
y = np.array([0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

X

array([[3.78],
       [2.44],
       [2.09],
       [0.14],
       [1.72],
       [1.65],
       [4.92],
       [4.37],
       [4.96],
       [4.52],
       [3.69],
       [5.88]])

y

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

#Fitting the model
logr = linear_model.LogisticRegression()
logr.fit(X,y)

LogisticRegression()

#predict if tumor is cancerous where the size is 3.46mm:
predicted = logr.predict(np.array([3.46]).reshape(-1,1))
predicted

array([0])
```

We have predicted that a tumor with a size of 3.46mm will not be cancerous.

weak6-Perform clustering using k-means clustering algorithm-

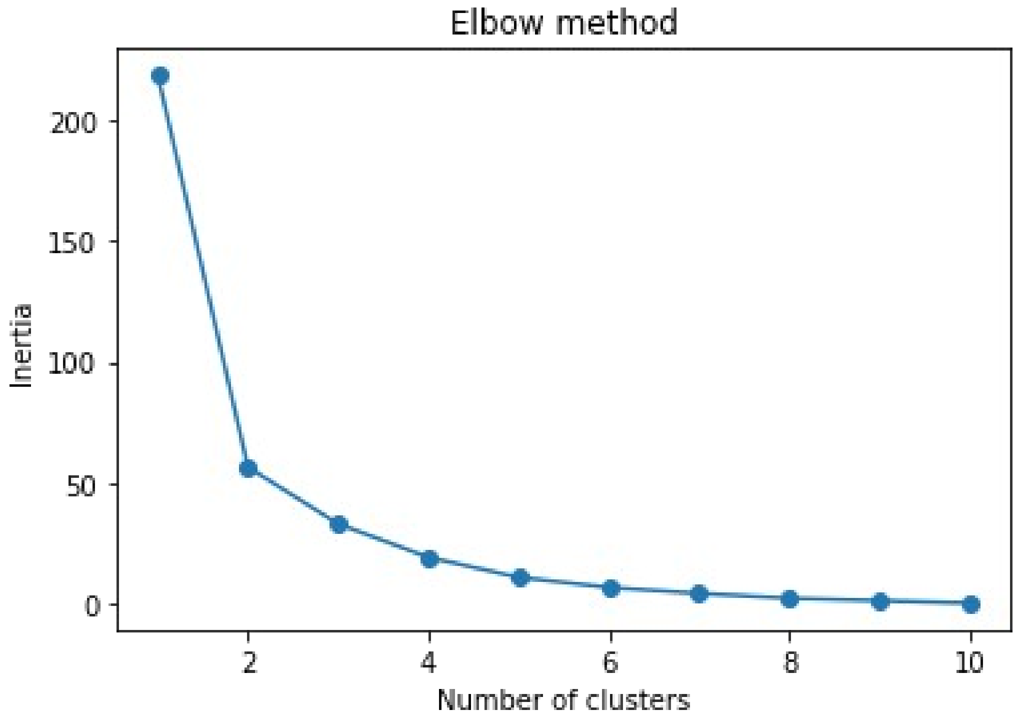
```
#1.Start by visualizing some data points:
import matplotlib.pyplot as plt
```

```
x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

plt.scatter(x, y)
plt.show()
```

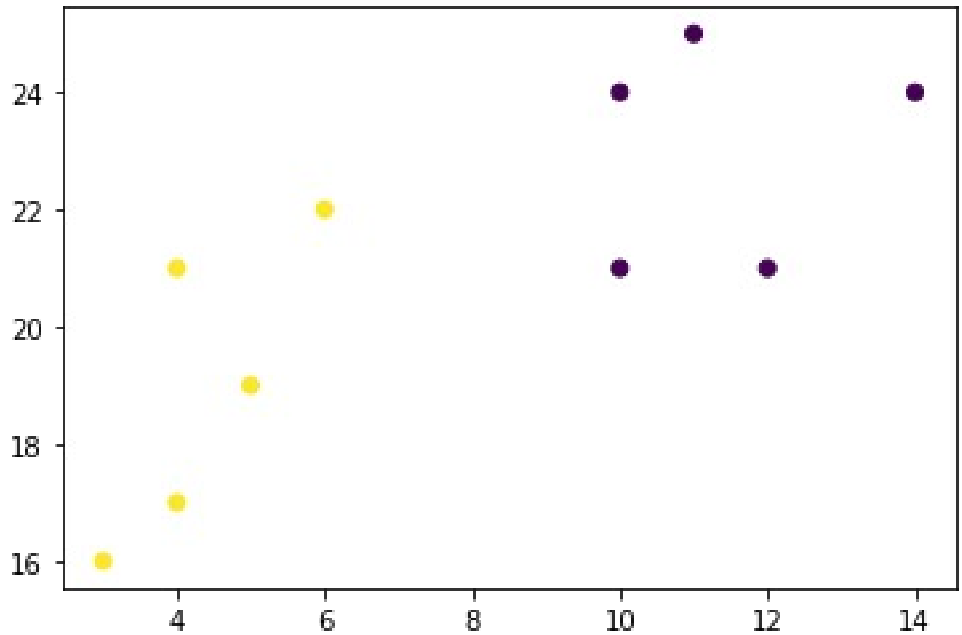


```
#2.Now we utilize the elbow method to visualize the intertia for
different values of K:
from sklearn.cluster import KMeans

data = list(zip(x, y))
inertias = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(data)
    inertias.append(kmeans.inertia_)

plt.plot(range(1,11), inertias, marker='o')
plt.title('Elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.show()

C:\Users\dodda\anaconda3\lib\site-packages\sklearn\cluster\
_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak on
Windows with MKL, when there are less chunks than available threads.
You can avoid it by setting the environment variable
```

```
OMP_NUM_THREADS=1.
  warnings.warn(
```




```
#3.The elbow method shows that 2 is a good value for K, so we retrain
and visualize the result:
kmeans = KMeans(n_clusters=2)
kmeans.fit(data)

plt.scatter(x, y, c=kmeans.labels_)
plt.show()
```



weak7-Perform Principle Component Analysis and then perform clustering-

Step 1- Import required libraries

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
import seaborn as sns
from sklearn.decomposition import PCA
import scipy.cluster.hierarchy as sch
```

Step 2- Load, visualize and explore the dataset

```
df = pd.read_csv("7_wine.csv")
df.head()

   Type  Alcohol  Malic   Ash  Alcalinity  Magnesium  Phenols
Flavanoids  \
0     1    14.23   1.71  2.43        15.6        127     2.80
3.06
1     1    13.20   1.78  2.14        11.2        100     2.65
2.76
```

```
2     1    13.16   2.36  2.67         18.6        101      2.80
3.24
3     1    14.37   1.95  2.50         16.8        113      3.85
3.49
4     1    13.24   2.59  2.87         21.0        118      2.80
2.69

   Nonflavanoids  Proanthocyanins  Color   Hue  Dilution  Proline
0           0.28             2.29   5.64  1.04      3.92     1065
1           0.26             1.28   4.38  1.05      3.40     1050
2           0.30             2.81   5.68  1.03      3.17     1185
3           0.24             2.18   7.80  0.86      3.45     1480
4           0.39             1.82   4.32  1.04      2.93      735
```

```
y= df['Type']
df1 = df.iloc[:, 1:]
df1.head()
```

```
   Alcohol  Malic   Ash  Alcalinity  Magnesium  Phenols  Flavanoids  \
0    14.23   1.71  2.43        15.6        127     2.80        3.06
1    13.20   1.78  2.14        11.2        100     2.65        2.76
2    13.16   2.36  2.67        18.6        101     2.80        3.24
3    14.37   1.95  2.50        16.8        113     3.85        3.49
4    13.24   2.59  2.87        21.0        118     2.80        2.69

   Nonflavanoids  Proanthocyanins  Color   Hue  Dilution  Proline
0           0.28             2.29   5.64  1.04      3.92     1065
1           0.26             1.28   4.38  1.05      3.40     1050
2           0.30             2.81   5.68  1.03      3.17     1185
3           0.24             2.18   7.80  0.86      3.45     1480
4           0.39             1.82   4.32  1.04      2.93      735
```

```
y #actual clusters
```

```
0      1
1      1
2      1
3      1
4      1
      ..
173    3
174    3
175    3
176    3
177    3
Name: Type, Length: 178, dtype: int64
```

Step 3- Feature Scaling

```python
# normalizing the data
df_norm = StandardScaler().fit_transform(df1)

df_norm
```

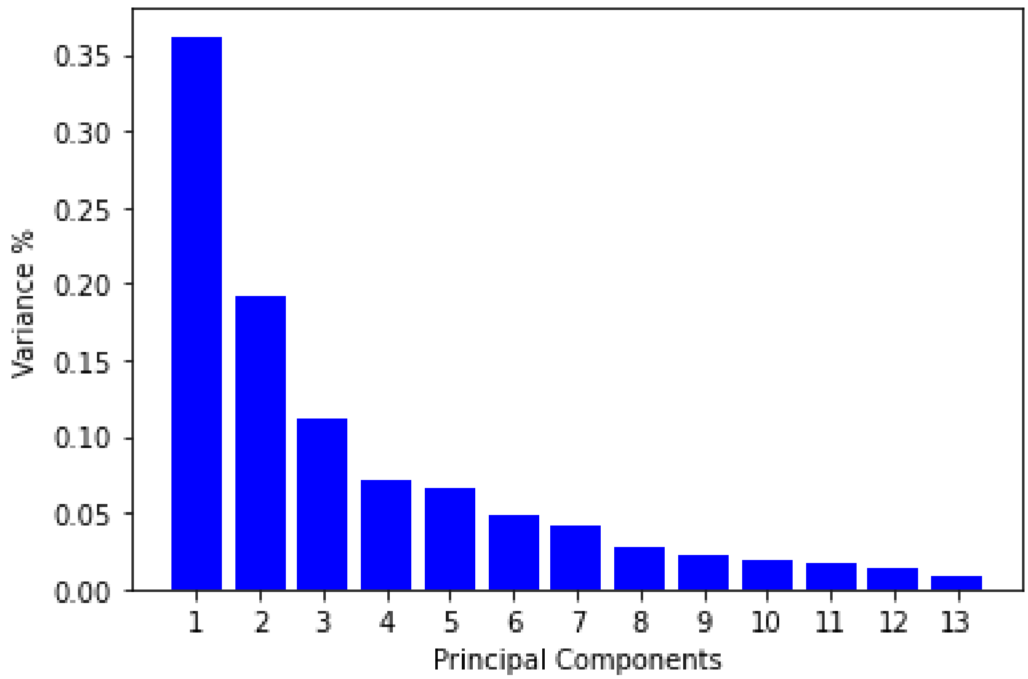
```
array([[ 1.51861254, -0.5622498 ,  0.23205254, ...,  0.36217728,
         1.84791957,  1.01300893],
       [ 0.24628963, -0.49941338, -0.82799632, ...,  0.40605066,
         1.1134493 ,  0.96524152],
       [ 0.19687903,  0.02123125,  1.10933436, ...,  0.31830389,
         0.78858745,  1.39514818],
       ...,
       [ 0.33275817,  1.74474449, -0.38935541, ..., -1.61212515,
        -1.48544548,  0.28057537],
       [ 0.20923168,  0.22769377,  0.01273209, ..., -1.56825176,
        -1.40069891,  0.29649784],
       [ 1.39508604,  1.58316512,  1.36520822, ..., -1.52437837,
        -1.42894777, -0.59516041]])
```

Step 4- Dimensionality Reduction with PCA

```python
# Applying PCA function
pca = PCA(n_components=13)
principalComponents = pca.fit_transform(df_norm)

PC = range(1, pca.n_components_+1)
plt.bar(PC, pca.explained_variance_ratio_, color='blue')
plt.xlabel('Principal Components')
plt.ylabel('Variance %')
plt.xticks(PC)
```

```
([<matplotlib.axis.XTick at 0x21183ef3c10>,
  <matplotlib.axis.XTick at 0x21183ef3be0>,
  <matplotlib.axis.XTick at 0x21183ef3310>,
  <matplotlib.axis.XTick at 0x21183f450a0>,
  <matplotlib.axis.XTick at 0x21183f45550>,
  <matplotlib.axis.XTick at 0x21183f45be0>,
  <matplotlib.axis.XTick at 0x21183f4a370>,
  <matplotlib.axis.XTick at 0x21183f4aac0>,
  <matplotlib.axis.XTick at 0x21183f52250>,
  <matplotlib.axis.XTick at 0x21183f529a0>,
  <matplotlib.axis.XTick at 0x21183f52610>,
  <matplotlib.axis.XTick at 0x21183f4a5e0>,
  <matplotlib.axis.XTick at 0x21183f550d0>],
 [Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
  Text(0, 0, ''),
```

```
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, ''),
 Text(0, 0, '')])
```



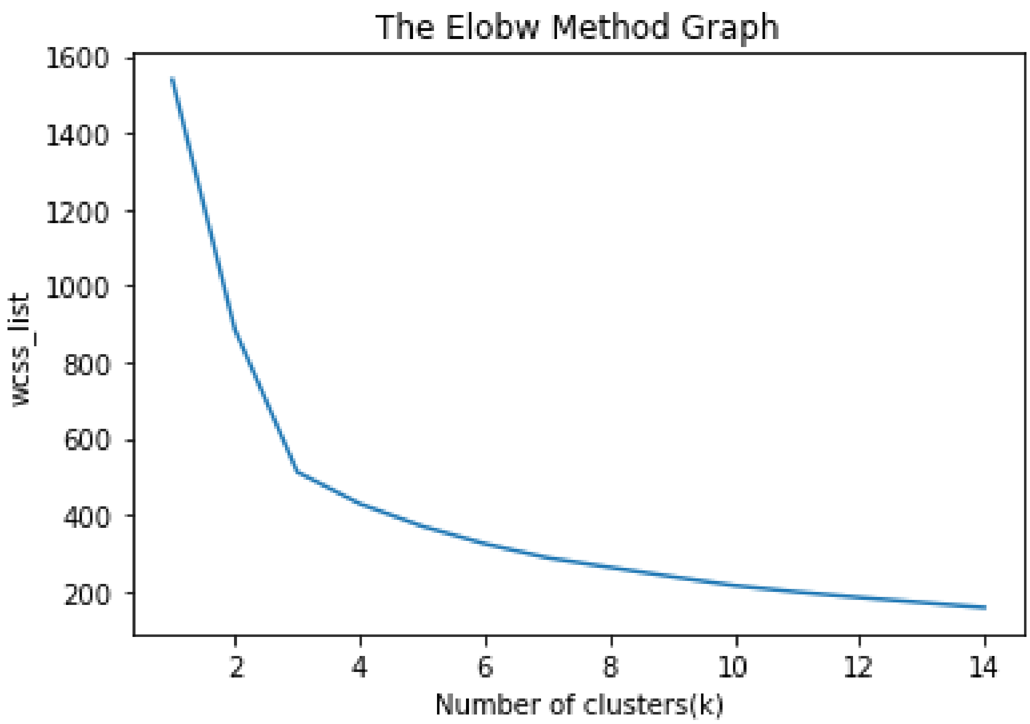

```
pca.explained_variance_ratio_

array([0.36198848, 0.1920749 , 0.11123631, 0.0706903 , 0.06563294,
       0.04935823, 0.04238679, 0.02680749, 0.02222153, 0.01930019,
       0.01736836, 0.01298233, 0.00795215])

PCA_components = pd.DataFrame(principalComponents)

PCA_components

          0         1         2         3         4         5
6   \
0    3.316751 -1.443463 -0.165739 -0.215631  0.693043 -0.223880
0.596427
1    2.209465  0.333393 -2.026457 -0.291358 -0.257655 -0.927120
0.053776
2    2.516740 -1.031151  0.982819  0.724902 -0.251033  0.549276
0.424205
3    3.757066 -2.756372 -0.176192  0.567983 -0.311842  0.114431 -
0.383337
```

```
4    1.008908 -0.869831  2.026688 -0.409766  0.298458 -0.406520
0.444074
..        ...       ...       ...       ...       ...       ...
...
173 -3.370524 -2.216289 -0.342570  1.058527 -0.574164 -1.108788
0.958416
174 -2.601956 -1.757229  0.207581  0.349496  0.255063 -0.026465
0.146894
175 -2.677839 -2.760899 -0.940942  0.312035  1.271355  0.273068
0.679235
176 -2.387017 -2.297347 -0.550696 -0.688285  0.813955  1.178783
0.633975
177 -3.208758 -2.768920  1.013914  0.596903 -0.895193  0.296092
0.005741

            7         8         9        10        11        12
0    0.065139  0.641443  1.020956 -0.451563  0.540810 -0.066239
1    1.024416 -0.308847  0.159701 -0.142657  0.388238  0.003637
2   -0.344216 -1.177834  0.113361 -0.286673  0.000584  0.021717
3    0.643593  0.052544  0.239413  0.759584 -0.242020 -0.369484
4    0.416700  0.326819 -0.078366 -0.525945 -0.216664 -0.079364
..        ...       ...       ...       ...       ...       ...
173 -0.146097 -0.022498 -0.304117  0.139228  0.170786 -0.114427
174 -0.552427 -0.097969 -0.206061  0.258198 -0.279431 -0.187371
175  0.047024  0.001222 -0.247997  0.512492  0.698766  0.072078
176  0.390829  0.057448  0.491490  0.299822  0.339821 -0.021866
177 -0.292914  0.741660 -0.117969 -0.229964 -0.188788 -0.323965

[178 rows x 13 columns]
```

Step 5- We have to find the optimal K value for clustering the data. Now we are using the Elbow method to find the optimal K value.

```
wcss = []
for i in range(1, 15):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state =
42)
    kmeans.fit(PCA_components.iloc[:,:3])
    wcss.append(kmeans.inertia_)

C:\Users\dodda\anaconda3\lib\site-packages\sklearn\cluster\
_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak on
Windows with MKL, when there are less chunks than available threads.
You can avoid it by setting the environment variable
OMP_NUM_THREADS=1.
  warnings.warn(

plt.plot(range(1, 15), wcss)
plt.title('The Elobw Method Graph')
plt.xlabel('Number of clusters(k)')
```

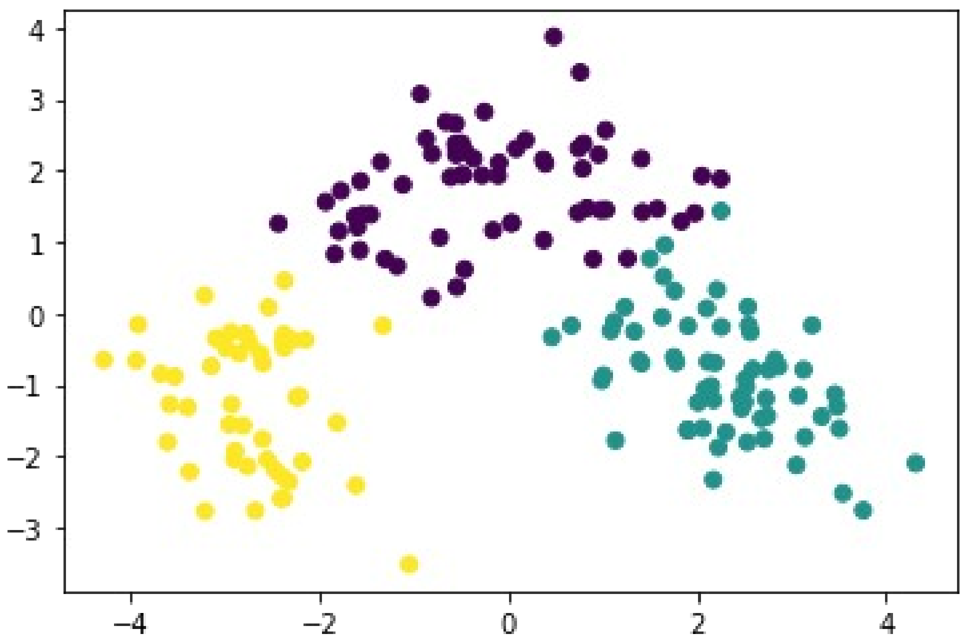
```
plt.ylabel('wcss_list')
plt.show()
```




```
model = KMeans(n_clusters=3)
labels=model.fit_predict(PCA_components.iloc[:,:2])

labels

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

Step-7: Visualizing the Clusters

```
plt.scatter(PCA_components[0], PCA_components[1], c=labels)
plt.show()
```



Step-8: Comparing actual Cluster No. with predicted Cluster No

```
df_r=pd.DataFrame({'Actual':y, 'Predicted':labels})
df_r.head()
```

	Actual	Predicted
0	1	1
1	1	1
2	1	1
3	1	1
4	1	1

weak8-Prepare a Classification model using decision tree Classifier.-

```
#Three lines to make our compiler able to draw:
import sys
import matplotlib
matplotlib.use('Agg')

import pandas as pd
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
```

```
import matplotlib.pyplot as plt


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

print(df)
```

```
    Age  Experience  Rank Nationality   Go
0    36          10     9          UK   NO
1    42          12     4         USA   NO
2    23           4     6           N   NO
3    52           4     4         USA   NO
4    43          21     8         USA  YES
5    44          14     5          UK   NO
6    66           3     7           N  YES
7    35          14     9          UK  YES
8    52          13     7           N  YES
9    35           5     9           N  YES
10   24           3     5         USA   NO
11   18           3     7          UK  YES
12   45           9     9          UK  YES
```

```
d = {'UK': 0, 'USA': 1, 'N': 2}
df['Nationality'] = df['Nationality'].map(d)
d = {'YES': 1, 'NO': 0}
df['Go'] = df['Go'].map(d)

print(df)
```

```
    Age  Experience  Rank  Nationality  Go
0    36          10     9            0   0
1    42          12     4            1   0
2    23           4     6            2   0
3    52           4     4            1   0
4    43          21     8            1   1
5    44          14     5            0   0
6    66           3     7            2   1
7    35          14     9            0   1
8    52          13     7            2   1
9    35           5     9            2   1
10   24           3     5            1   0
11   18           3     7            0   1
12   45           9     9            0   1
```

```
features = ['Age', 'Experience', 'Rank', 'Nationality']

X = df[features]
y = df['Go']

X
```

```
    Age  Experience  Rank  Nationality
0    36          10     9            0
1    42          12     4            1
2    23           4     6            2
3    52           4     4            1
4    43          21     8            1
5    44          14     5            0
6    66           3     7            2
7    35          14     9            0
8    52          13     7            2
9    35           5     9            2
10   24           3     5            1
11   18           3     7            0
12   45           9     9            0
```

```
y
```

```
0     0
1     0
2     0
3     0
4     1
5     0
6     1
7     1
8     1
9     1
10    0
11    1
12    1
Name: Go, dtype: int64
```

```
dtree = DecisionTreeClassifier()
dtree = dtree.fit(X, y)

tree.plot_tree(dtree, feature_names=features)
```

```
[Text(0.3333333333333333, 0.875, 'Rank <= 6.5\ngini = 0.497\nsamples =
13\nvalue = [6, 7]'),
 Text(0.16666666666666666, 0.625, 'gini = 0.0\nsamples = 5\nvalue =
[5, 0]'),
 Text(0.5, 0.625, 'Experience <= 9.5\ngini = 0.219\nsamples = 8\nvalue
= [1, 7]'),
 Text(0.3333333333333333, 0.375, 'gini = 0.0\nsamples = 4\nvalue = [0,
4]'),
 Text(0.6666666666666666, 0.375, 'Experience <= 11.5\ngini = 0.375\
nsamples = 4\nvalue = [1, 3]'),
 Text(0.5, 0.125, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.8333333333333334, 0.125, 'gini = 0.0\nsamples = 3\nvalue = [0,
3]')]
```

weak9-Prepare a Classification model using Navie Bayes Classifier-

Step-1:Assigning features and label variables

```
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast'
,'Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mil
d','Mild','Mild','Hot','Mild']

play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Y
es','Yes','No']
```

Step-2: Encoding the data

```
# Import LabelEncoder
from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
wheather_encoded=le.fit_transform(weather)
print(wheather_encoded)

[2 2 0 1 1 1 0 2 2 1 2 0 0 1]

# Converting string labels into numbers
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)
print("Temp:",temp_encoded)
print("Play:",label)

Temp: [1 1 1 2 0 0 0 2 0 2 2 2 1 2]
Play: [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

Step-3:Combinig weather and temp into single listof tuples

```
features=zip(wheather_encoded,temp_encoded)
final=list(features)

final

[(2, 1),
 (2, 1),
 (0, 1),
 (1, 2),
```

```
 (1, 0),
 (1, 0),
 (0, 0),
 (2, 2),
 (2, 0),
 (1, 2),
 (2, 2),
 (0, 2),
 (0, 1),
 (1, 2)]
```

Step-4: Fitting the model and predicting the classfier

```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
model = GaussianNB()

# Train the model using the training sets
model.fit(final,label)

#Predict Output
predicted= model.predict([[0,0]]) # 0:Overcast, 2:Mild
print("Predicted Value:", predicted)

Predicted Value: [1]
```

#Here, 1 indicates that players can 'play'.

```
weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast'
,'Sunny','Sunny',
'Rainy','Sunny','Overcast','Overcast','Rainy']
temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mil
d','Mild','Mild','Hot','Mild']

play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Y
es','Yes','No']
# Import LabelEncoder
from sklearn import preprocessing
#creating labelEncoder
le = preprocessing.LabelEncoder()
# Converting string labels into numbers.
wheather_encoded=le.fit_transform(weather)
print("Weather:",wheather_encoded)
# Converting string labels into numbers
temp_encoded=le.fit_transform(temp)
label=le.fit_transform(play)
print("Temp:",temp_encoded)
```

```python
print("Play:",label)
features=zip(wheather_encoded,temp_encoded)
final=list(features)
print("Features are:", final)
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
model = GaussianNB()

# Train the model using the training sets
model.fit(final,label)

#Predict Output
predicted= model.predict([[0,2]]) # 0:Overcast, 2:Mild
print("Predicted Value:", predicted)
```

```
Weather: [2 2 0 1 1 1 0 2 2 1 2 0 0 1]
Temp: [1 1 1 2 0 0 0 2 0 2 2 2 1 2]
Play: [0 0 1 1 1 0 1 0 1 1 1 1 1 0]
Features are: [(2, 1), (2, 1), (0, 1), (1, 2), (1, 0), (1, 0), (0, 0),
(2, 2), (2, 0), (1, 2), (2, 2), (0, 2), (0, 1), (1, 2)]
Predicted Value: [1]
```