Ex No: 01 **Statistical basics for Data Science**

Date:

Aim:

To perform a statistical basics for data science using python in google colab.

Procedure:

Step 1: Open Google colab.

Step 2: Import the required libraries.

Step 3: Perform the statistical basics such as Mean, Median, and Mode

Step 4: Run the code

Step 5: End

Implementation:

```
import pandas as pd

d={'Names':pd.Series(['kavi','ram','janu','madhu','ruby','dinesh','kavya','sasmi','vinoth','jegan']),

   'Ages':pd.Series([32,41,28,54,35,26,23,33,38,40])}

df=pd.DataFrame(d)

print("Mean values in the distribution:")

print(df.mean())

print("*************************************")

print("Median vaules in the distribution:")

print(df.median())

print("*************************************")
```

```
print("Mode values in the distribution:")

print(df.mode())

print("*************************************")

print("Standard Deviation")

print(df.std())

print("*************************************")
```

Output:

```
Mean values in the distribution:
Ages    35.0
dtype: float64
*************************************
Median vaules in the distribution:
Ages    34.0
dtype: float64
*************************************
Mode values in the distribution:
    Names  Ages
0  dinesh    23
1    janu    26
2   jegan    28
3    kavi    32
4   kavya    33
5   madhu    35
6     ram    38
7    ruby    40
8   sasmi    41
9  vinoth    54
*************************************
Standard Deviation
Ages    8.931841
dtype: float64
```

CLASS PERFORMANCE	
RECORD	
VIVA	
TOTAL	

Result:

Thus, the statistical basics for data science were performed successfully and the output is verified.

Ex No: 2a Implement K-Means Algorithm

Aim:

To implement K-means algorithm for the given dataset using python in Google colab.

Procedure:

Step 1: Open Google colab.

Step 2: Import the required libraries.

Step 3: Read the dataset (train,test)

Step 4: Perform the K-means algorithm and run the code

Step 5: End

Implementation:

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
train=pd.read_csv(r"C:\Users\NIKHIL\Downloads\train (1).csv")
```

test=pd.read_csv(r"C:\Users\NIKHIL\Downloads\test.csv")

print("-------Train set -----")

print(train.head())

print("\n")

print("-------Test set----- ")

print(test.head())

```
-------Train set------
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S


-------Test set------
   PassengerId  Pclass                                          Name     Sex  \
0          892       3                              Kelly, Mr. James    male
1          893       3              Wilkes, Mrs. James (Ellen Needs)  female
2          894       2                     Myles, Mr. Thomas Francis    male
3          895       3                              Wirz, Mr. Albert    male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked
0  34.5      0      0   330911   7.8292   NaN        Q
1  47.0      1      0   363272   7.0000   NaN        S
2  62.0      0      0   240276   9.6875   NaN        Q
3  27.0      0      0   315154   8.6625   NaN        S
```

print("-------Train set -----")

print(train.describe())

print("\n")

print("-------Test set----- ")

print(test.describe())

```
-------Train set------
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200


-------Test set------
       PassengerId      Pclass         Age       SibSp       Parch        Fare
count   418.000000  418.000000  332.000000  418.000000  418.000000  417.000000
mean   1100.500000    2.265550   30.272590    0.447368    0.392344   35.627188
std     120.810458    0.841838   14.181209    0.896760    0.981429   55.907576
min     892.000000    1.000000    0.170000    0.000000    0.000000    0.000000
25%     996.250000    1.000000   21.000000    0.000000    0.000000    7.895800
50%    1100.500000    3.000000   27.000000    0.000000    0.000000   14.454200
75%    1204.750000    3.000000   39.000000    1.000000    0.000000   31.500000
max    1309.000000    3.000000   76.000000    8.000000    9.000000  512.329200
```

print(train.columns.values)

```
['PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch'
 'Ticket' 'Fare' 'Cabin' 'Embarked']
```

train.isna().head()

Out[9]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	True	False

test.isna().head()

Out[10]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	True	False
2	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	True	False
4	False	False	False	False	False	False	False	False	False	True	False

print("-------Train set -----")

```
print(train.isnull().sum())

print("\n")

print("-------Test set----- ")

print(test.isnull().sum())
```

```
*****In the train set*****
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64


*****In the test set*****
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

```
train.fillna(train.mean(),inplace=True)

test.fillna(test.mean(),inplace=True)

print(train.isna().sum())
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

print(test.isna().sum())

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          327
Embarked         0
dtype: int64
```

train['Ticket'].head()

```
Out[16]: 0           A/5 21171
         1            PC 17599
         2    STON/O2. 3101282
         3              113803
         4              373450
         Name: Ticket, dtype: object
```

train['Cabin'].head()

```
Out[17]: 0     NaN
         1     C85
         2     NaN
         3    C123
         4     NaN
         Name: Cabin, dtype: object
```

train[['Sex','Survived']].groupby(['Sex'], as_index=False).mean().sort_values(by='Survived', ascending=False)

Out[19]:

	Sex	Survived
0	female	0.742038
1	male	0.188908

train[['Pclass','Survived']].groupby(['Pclass'], as_index=False).mean().sort_values(by='Survived', ascending=False)

Out[18]:

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

train[["SibSp", "Survived"]].groupby(['SibSp'], as_index=False).mean().sort_values(by='Survived', ascending=False)

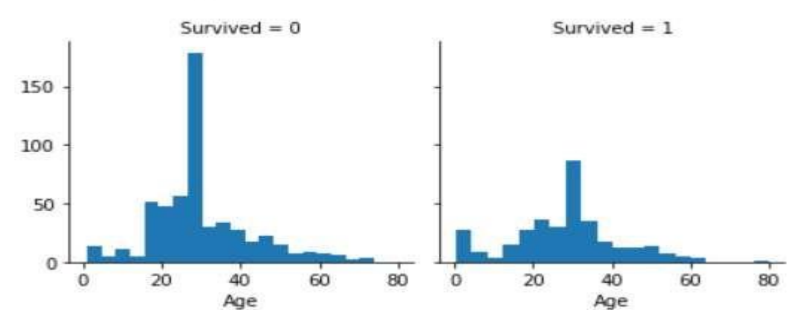
Out[21]:

	SibSp	Survived
1	1	0.535885
2	2	0.464286
0	0	0.345395
3	3	0.250000
4	4	0.166667
5	5	0.000000
6	8	0.000000

g = sns.FacetGrid(train,col='Survived')

g.map(plt.hist,'Age', bins = 20)

Out[22]: <seaborn.axisgrid.FacetGrid at 0x7fce52598fd0>



train.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          891 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
labelEncoder = LabelEncoder()

labelEncoder.fit(train['Sex'])

labelEncoder.fit(test['Sex'])

train['Sex'] = labelEncoder.transform(train['Sex'])

test['Sex'] = labelEncoder.transform(test['Sex'])

train = train.drop(['Name','Ticket', 'Cabin','Embarked'], axis=1)

test = test.drop(['Name','Ticket', 'Cabin','Embarked'], axis=1)

train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Sex          891 non-null    int64
 4   Age          891 non-null    float64
 5   SibSp        891 non-null    int64
 6   Parch        891 non-null    int64
 7   Fare         891 non-null    float64
dtypes: float64(2), int64(6)
memory usage: 55.8 KB
```

```
X = np.array(train.drop(['Survived'], 1).astype(float))
```

y = np.array(train['Survived'])

kmeans = KMeans(n_clusters=2)

kmeans.fit(X)

```
Out[30]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
                n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
                random_state=None, tol=0.0001, verbose=0)
```

correct = 0

for i in range(len(X)):

predict_me = np.array(X[i].astype(float))

predict_me = predict_me.reshape(-1, len(predict_me))

prediction = kmeans.predict(predict_me)

if prediction[0] == y[i]:

correct += 1

print(correct/len(X))

```
0.49158249158249157
```

kmeans = kmeans = KMeans(n_clusters=2, max_iter=600, algorithm = 'auto')

kmeans.fit(X)

```
Out[32]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=600,
                n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
                random_state=None, tol=0.0001, verbose=0)
```

scaler = MinMaxScaler()

X_scaled = scaler.fit_transform(X)

kmeans.fit(X_scaled)

```
Out[35]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=600,
         n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
         random_state=None, tol=0.0001, verbose=0)
```

correct = 0

for i in range(len(X)):

predict_me = np.array(X[i].astype(float))

predict_me = predict_me.reshape(-1, len(predict_me))

prediction = kmeans.predict(predict_me)

if prediction[0] == y[i]:

correct += 1

print(correct/len(X))

```
0.6262626262626263
```

CLASS PERFORMANCE	
RECORD	
VIVA	
TOTAL	

Result:

The above code has been executed and the output is verified.

EX NO: 2b **Implement K-Mediods Algorithm**

DATE:

Aim:

To perform a K-Mediods algorithm for the given dataset using python in Google colab.

Procedure:

Step 1: Open Google colab.

Step 2: Import the required libraries.

Step 3: Read the dataset (train dataset, test dataset)

Step 4: Perform the K-Mediods algorithm and run the code

Step 5: End

Implementation:

```
pip install scikit-learn-extra
import pandas as pd
from sklearn_extra.cluster import KMedoids
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

```
train=pd.read_csv(r"C:\Users\NIKHIL\Downloads\train (1).csv")

test=pd.read_csv(r"C:\Users\NIKHIL\Downloads\test.csv")

print("-------Train set -----")

print(train.head())

print("\n")

print("-------Test set----- ")

print(test.head())
```

```
-------Train set------
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S


-------Test set------
   PassengerId  Pclass                                          Name     Sex  \
0          892       3                              Kelly, Mr. James    male
1          893       3              Wilkes, Mrs. James (Ellen Needs)  female
2          894       2                     Myles, Mr. Thomas Francis    male
3          895       3                              Wirz, Mr. Albert    male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked
0  34.5      0      0   330911   7.8292   NaN        Q
1  47.0      1      0   363272   7.0000   NaN        S
2  62.0      0      0   240276   9.6875   NaN        Q
3  27.0      0      0   315154   8.6625   NaN        S
```

```
print("-------Train set -----")

print(train.describe())

print("\n")

print("-------Test set----- ")
```

print(test.describe())

```
-------Train set------
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200

-------Test set------
       PassengerId      Pclass         Age       SibSp       Parch        Fare
count   418.000000  418.000000  332.000000  418.000000  418.000000  417.000000
mean   1100.500000    2.265550   30.272590    0.447368    0.392344   35.627188
std     120.810458    0.841838   14.181209    0.896760    0.981429   55.907576
min     892.000000    1.000000    0.170000    0.000000    0.000000    0.000000
25%     996.250000    1.000000   21.000000    0.000000    0.000000    7.895800
50%    1100.500000    3.000000   27.000000    0.000000    0.000000   14.454200
75%    1204.750000    3.000000   39.000000    1.000000    0.000000   31.500000
max    1309.000000    3.000000   76.000000    8.000000    9.000000  512.329200
```

print(train.columns.values)

```
['PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch'
 'Ticket' 'Fare' 'Cabin' 'Embarked']
```

train.isna().head()

Out[9]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	True	False

test.isna().head()

Out[10]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	True	False
2	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	True	False
4	False	False	False	False	False	False	False	False	False	True	False

```
print("-------Train set ----- ")

print(train.isnull().sum())

print("\n")

print("-------Test set----- ")

print(test.isnull().sum())
```

```
*****In the train set*****
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64


*****In the test set*****
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

```
train.fillna(train.mean(),inplace=True)

test.fillna(test.mean(),inplace=True)

print(train.isna().sum())
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

print(test.isna().sum())

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          327
Embarked         0
dtype: int64
```

train['Ticket'].head()

```
Out[16]: 0           A/5 21171
         1            PC 17599
         2    STON/O2. 3101282
         3              113803
         4              373450
         Name: Ticket, dtype: object
```

train['Cabin'].head()

```
Out[17]: 0     NaN
         1     C85
         2     NaN
         3    C123
         4     NaN
         Name: Cabin, dtype: object
```

train[['Sex','Survived']].groupby(['Sex'], as_index=False).mean().sort_values(by='Survived', ascending=False)

Out[19]:

	Sex	Survived
0	female	0.742038
1	male	0.188908

train[['Pclass','Survived']].groupby(['Pclass'], as_index=False).mean().sort_values(by='Survived', ascending=False)

Out[18]:

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

train[["SibSp", "Survived"]].groupby(['SibSp'], as_index=False).mean().sort_values(by='Survived', ascending=False)

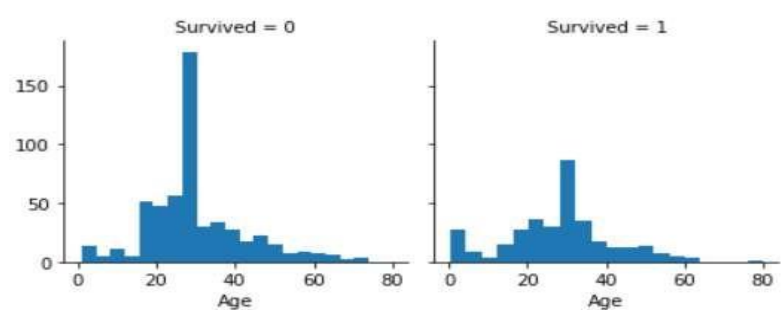
Out[21]:

	SibSp	Survived
1	1	0.535885
2	2	0.464286
0	0	0.345395
3	3	0.250000
4	4	0.166667
5	5	0.000000
6	8	0.000000

g = sns.FacetGrid(train,col='Survived')

g.map(plt.hist,'Age', bins = 20)

Out[22]: <seaborn.axisgrid.FacetGrid at 0x7fce52598fd0>



train.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          891 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
labelEncoder = LabelEncoder()

labelEncoder.fit(train['Sex'])

labelEncoder.fit(test['Sex'])

train['Sex'] = labelEncoder.transform(train['Sex'])

test['Sex'] = labelEncoder.transform(test['Sex'])

train = train.drop(['Name','Ticket', 'Cabin','Embarked'], axis=1)

test = test.drop(['Name','Ticket', 'Cabin','Embarked'], axis=1)

train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Sex          891 non-null    int64  
 4   Age          891 non-null    float64
 5   SibSp        891 non-null    int64  
 6   Parch        891 non-null    int64  
 7   Fare         891 non-null    float64
dtypes: float64(2), int64(6)
memory usage: 55.8 KB
```

```
kmedoids = KMedoids(n_clusters=2, random_state=0).fit(X)

correct = 0

for i in range(len(X)):
```

predict_me = np.array(X[i].astype(float))

predict_me = predict_me.reshape(-1, len(predict_me))

prediction = kmedoids.predict(predict_me)

if prediction[0] == y[i]:

correct += 1

print(correct/len(X))

```
0.49158249158249157
```

kmedoids = kmedoids = KMedoids(n_clusters=2, max_iter=600)

kmedoids.fit(X)

```
KMedoids(max_iter=600, n_clusters=2)
```

scaler = MinMaxScaler()

X_scaled = scaler.fit_transform(X)

kmedoids.fit(X_scaled)

```
KMedoids(max_iter=600, n_clusters=2)
```

correct = 0

for i in range(len(X)):

predict_me = np.array(X[i].astype(float))

predict_me = predict_me.reshape(-1, len(predict_me))

prediction = kmedoids.predict(predict_me)

if prediction[0] == y[i]:

correct += 1

print(correct/len(X))

```
0.6161616161616161
```

kmedoids.inertia_

```
647.3786187283088
```

CLASS PERFORMANCE	
RECORD	
VIVA	
TOTAL	

Result:

The above code has been executed and the output is verified.

EX NO: 3 **Implement Agglomerative Clustering**

DATE:

Aim:

To implement an Agglomerative clustering for the given dataset using python in jupyter notebook.

Procedure:

Step 1: Open Google colab.

Step 2: Import the required libraries.

Step 3: Read the dataset (Mall customer's dataset)

Step 4: Perform the Aggloramative clustering and run the code

Step 5: End

Implementation:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

data=pd.read_csv(r"C:\Users\NIKHIL\Downloads\Mall_Customers.csv")

data.head()

Out[4]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

X = data.iloc[:, [3, 4]].values

import scipy.cluster.hierarchy as sch

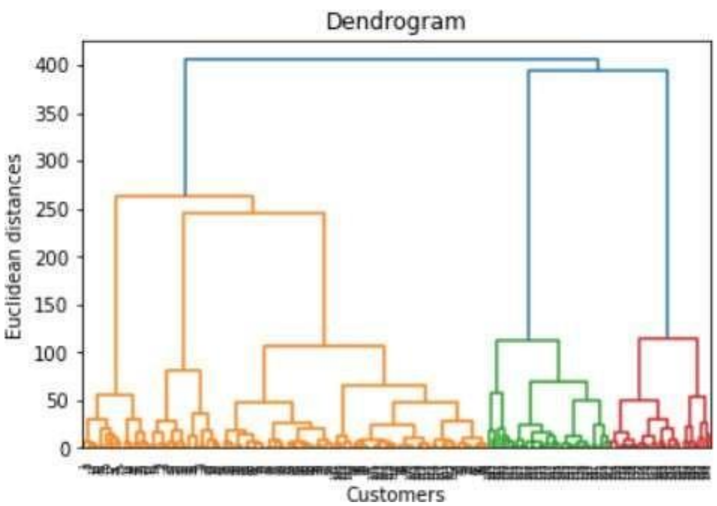
dendro = sch.dendrogram(sch.linkage(X, method = 'ward'))

plt.title('Dendrogram')

plt.xlabel('Customers')

plt.ylabel('Euclidean distances')

plt.show()



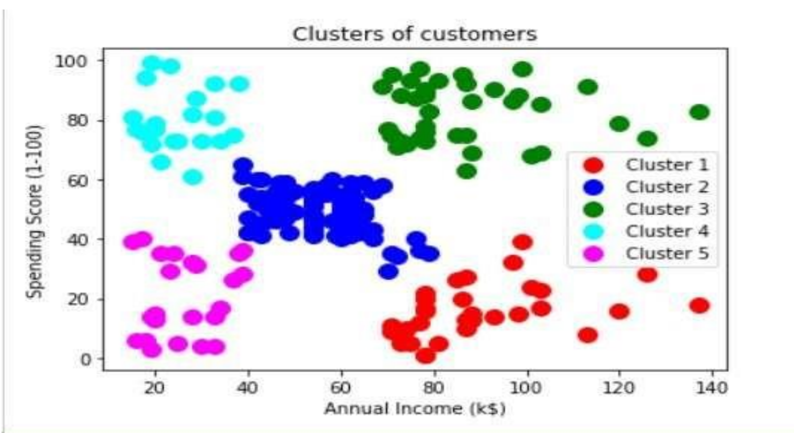
from sklearn.cluster import AgglomerativeClustering

hc = AgglomerativeClustering(n_clusters = 5, affinity = 'euclidean', linkage = 'ward')

y_hc = hc.fit_predict(X)

```
[4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4 3 4
 3 4 3 4 3 4 1 4 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 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 1 2 0 2 1 2
 0 2 0 2 0 2 0 2 0 2 0 2 1 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2 0
 2 0 2 0 2 0 2 0 2 0 2 0 2 0 2]
```

```
print(y_hc)

plt.scatter(X[y_hc == 0, 0], X[y_hc == 0, 1], s = 100, c = 'red', label = 'Cluster 1')
plt.scatter(X[y_hc == 1, 0], X[y_hc == 1, 1], s = 100, c = 'blue', label = 'Cluster 2')
plt.scatter(X[y_hc == 2, 0], X[y_hc == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
plt.scatter(X[y_hc == 3, 0], X[y_hc == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
plt.scatter(X[y_hc == 4, 0], X[y_hc == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```



CLASS PERFORMANCE	
RECORD	
VIVA	
TOTAL	

Result:

The Above code has been completetered and got verified.

EX NO: 4 Implement K-Nearest Neighbor Algorithm

Aim:

To implement a K-Nearest Neighbor algorithm for the given dataset using python in Google colab.

Procedure:

Step 1: Open Google colab.

Step 2: Import the required libraries.

Step 3: Read the dataset (fruit data with colours dataset)

Step 4: Perform the K-Nearest Neighbor algorithm and run the code

Step 5: End

Implementation:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

data=pd.read_table(r"C:\Users\NIKHIL\Downloads\fruit_data_with_colors.txt")

data.head()
```

Out[6]:

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79

```
data.isnull().sum()
```

```
Out[10]: fruit_label      0
         fruit_name       0
         fruit_subtype    0
         mass             0
         width            0
         height           0
         color_score      0
         dtype: int64
```

D_df=data.fillna(0)

D_df.head()

Out[14]:

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79

predct = dict(zip(data.fruit_label.unique(), data.fruit_name.unique()))

predct

```
Out[16]: {1: 'apple', 2: 'mandarin', 3: 'orange', 4: 'lemon'}
```

data['fruit_name'].value_counts()

```
apple       19
orange      19
lemon       16
mandarin     5
Name: fruit_name, dtype: int64
```

apple_data=data[data['fruit_name']=='apple']

orange_data=data[data['fruit_name']=='orange']

lemon_data=data[data['fruit_name']=='lemon']

mandarin_data=data[data['fruit_name']=='mandarin']

apple_data.head()

Out[20]:

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
8	1	apple	braeburn	178	7.1	7.8	0.92
9	1	apple	braeburn	172	7.4	7.0	0.89

orange_data.head()

Out[22]:

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
24	3	orange	spanish_jumbo	342	9.0	9.4	0.75
25	3	orange	spanish_jumbo	356	9.2	9.2	0.75
26	3	orange	spanish_jumbo	362	9.6	9.2	0.74
27	3	orange	selected_seconds	204	7.5	9.2	0.77
28	3	orange	selected_seconds	140	6.7	7.1	0.72

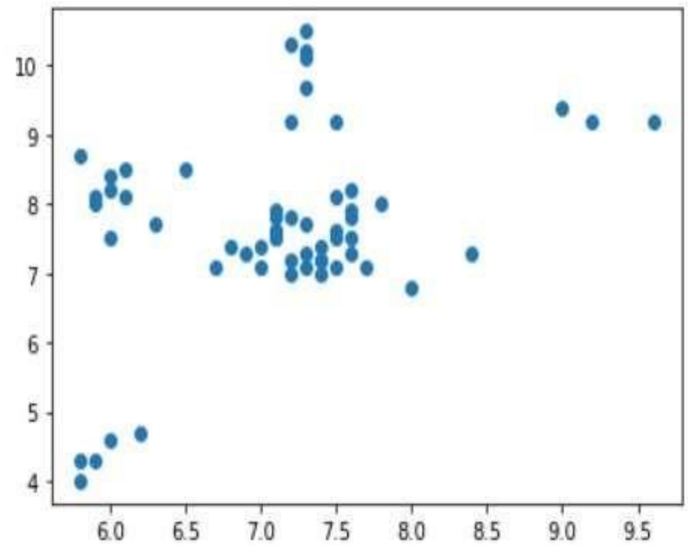
lemon_data.head()

Out[23]:

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
43	4	lemon	spanish_belsan	194	7.2	10.3	0.70
44	4	lemon	spanish_belsan	200	7.3	10.5	0.72
45	4	lemon	spanish_belsan	186	7.2	9.2	0.72
46	4	lemon	spanish_belsan	216	7.3	10.2	0.71
47	4	lemon	spanish_belsan	196	7.3	9.7	0.72

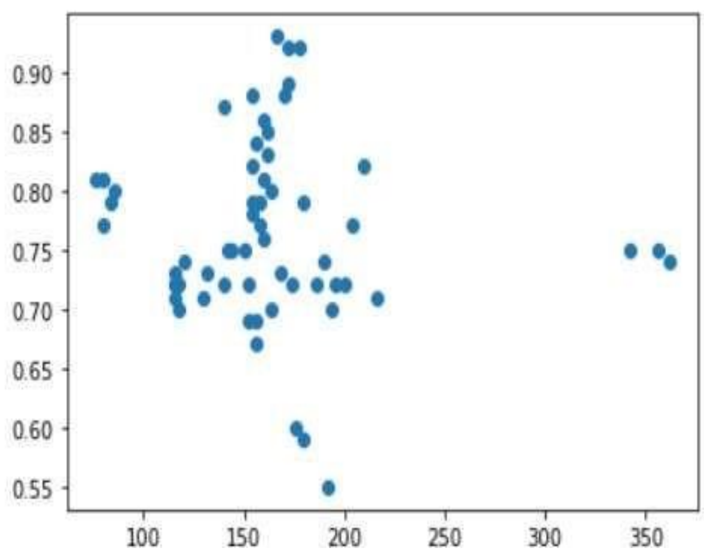
plt.scatter(data['width'],data['height'])

```
Out[24]: <matplotlib.collections.PathCollection at 0x1b4692484c0>
```



plt.scatter(data['mass'],data['color_score'])

```
Out[25]: <matplotlib.collections.PathCollection at 0x1b4695859a0>
```

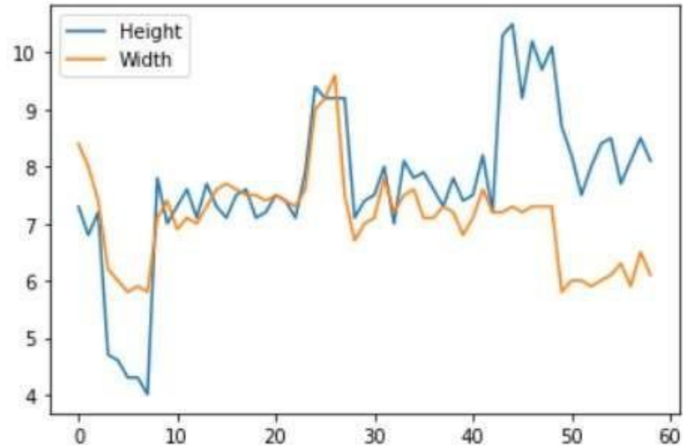


plt.plot(data['height'],label='Height')

plt.plot(data['width'],label='Width')

plt.legend()

Out[26]: <matplotlib.legend.Legend at 0x1b4695ecc10>



from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

X=data[['mass','width','height']]

Y=data['fruit_label']

X_train,X_test,y_train,y_test=train_test_split(X,Y,random_state=0)

X_train.describe()

Out[29]:

	mass	width	height
count	44.000000	44.000000	44.000000
mean	159.090909	7.038636	7.643182
std	53.316876	0.835886	1.370350
min	76.000000	5.800000	4.000000
25%	127.500000	6.175000	7.200000
50%	157.000000	7.200000	7.600000
75%	172.500000	7.500000	8.250000
max	356.000000	9.200000	10.500000

X_test.describe()

Out[30]:

	mass	width	height
count	15.000000	15.00000	15.000000
mean	174.933333	7.30000	7.840000
std	60.075508	0.75119	1.369463
min	84.000000	6.00000	4.600000
25%	146.000000	7.10000	7.250000
50%	166.000000	7.20000	7.600000
75%	185.000000	7.45000	8.150000
max	362.000000	9.60000	10.300000

knn=KNeighborsClassifier()

knn.fit(X_train,y_train)

```
Out[32]: KNeighborsClassifier()
```

knn.score(X_test,y_test)

```
Out[33]: 0.5333333333333333
```

prediction1=knn.predict([['90','5.3','7']])

predct[prediction1[0]]

```
C:\Users\SUJITHA\anaconda3\lib\site-packages\sklearn\utils\validation.py:72: FutureWarning: Beginning in version 0.22, arrays o
f bytes/strings will be converted to decimal numbers if dtype='numeric'. It is recommended that you convert the array to a floa
t dtype before using it in scikit-learn, for example by using your_array = your_array.astype(np.float64).
  return f(**kwargs)
C:\Users\SUJITHA\anaconda3\lib\site-packages\sklearn\utils\validation.py:72: FutureWarning: Beginning in version 0.22, arrays o
f bytes/strings will be converted to decimal numbers if dtype='numeric'. It is recommended that you convert the array to a floa
t dtype before using it in scikit-learn, for example by using your_array = your_array.astype(np.float64).
  return f(**kwargs)
Out[34]: 'mandarin'
```

prediction2=knn.predict([['120','8.3','5']])

predct[prediction2[0]]

```
C:\Users\SUJITHA\anaconda3\lib\site-packages\sklearn\utils\validation.py:72: FutureWarning: Beginning in version 0.22, arrays o
f bytes/strings will be converted to decimal numbers if dtype='numeric'. It is recommended that you convert the array to a floa
t dtype before using it in scikit-learn, for example by using your_array = your_array.astype(np.float64).
  return f(**kwargs)
C:\Users\SUJITHA\anaconda3\lib\site-packages\sklearn\utils\validation.py:72: FutureWarning: Beginning in version 0.22, arrays o
f bytes/strings will be converted to decimal numbers if dtype='numeric'. It is recommended that you convert the array to a floa
t dtype before using it in scikit-learn, for example by using your_array = your_array.astype(np.float64).
  return f(**kwargs)
Out[35]: 'lemon'
```

CLASS PERFORMANCE	
RECORD	
VIVA	
TOTAL	

Result:

The above code has been executed and the output is verified.

EX NO: 5 Implement Decision Tree Algorithm

Aim:

To implement a decision tree algorithm for the given dataset using python in Google colab.

Procedure:

Step 1: Open Google colab.

Step 2: Import the required libraries.

Step 3: Read the dataset (balance scale dataset)

Step 4: Perform the decision tree algorithm and run the code

Step 5: End

Implementation:

```
import numpy as np

import pandas as pd

from sklearn.metrics import confusion_matrix

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from sklearn.metrics import classification_report

from sklearn.tree import DecisionTreeClassifier

def importdata():

balance_data = pd.read_csv(

'https://archive.ics.uci.edu/ml/machine-learning-'+

'databases/balance-scale/balance-scale.data',
```

```
sep= ',', header = None)

# Printing the dataswet shape

print ("Dataset Length: ", len(balance_data))

print ("Dataset Shape: ", balance_data.shape)

# Printing the dataset obseravtions

print ("Dataset: ",balance_data.head())

return balance_data
```

```
Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
3  R  1  1  1  4
4  R  1  1  1  5
```

```
# Function to split the dataset

def splitdataset(balance_data):

# Separating the target variable

X = balance_data.values[:, 1:5]

Y = balance_data.values[:, 0]

# Splitting the dataset into train and test

X_train, X_test, y_train, y_test = train_test_split(

X, Y, test_size = 0.3, random_state = 100)

return X, Y, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.

def train_using_gini(X_train, X_test, y_train):

# Creating the classifier object

clf_gini = DecisionTreeClassifier(criterion = "gini",
```

```
random_state = 100,max_depth=3, min_samples_leaf=5)
# Performing training
clf_gini.fit(X_train, y_train)
return clf_gini
# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):
# Decision tree with entropy
clf_entropy = DecisionTreeClassifier(
criterion = "entropy", random_state = 100,
max_depth = 3, min_samples_leaf = 5)
# Performing training
clf_entropy.fit(X_train, y_train)
return clf_entropy
# Function to make predictions
def prediction(X_test, clf_object):
# Predicton on test with giniIndex
y_pred = clf_object.predict(X_test)
print("Predicted values:")
print(y_pred)
return y_pred
# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):
print("Confusion Matrix: ",
confusion_matrix(y_test, y_pred))
```

print ("Accuracy : ",

accuracy_score(y_test,y_pred)*100)

print("Report : ",

classification_report(y_test, y_pred))

```
Confusion Matrix:  [[ 0  6  7]
 [ 0 67 18]
 [ 0 19 71]]
Accuracy :  73.40425531914893
Report :               precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.73      0.79      0.76        85
           R       0.74      0.79      0.76        90

    accuracy                           0.73       188
   macro avg       0.49      0.53      0.51       188
weighted avg       0.68      0.73      0.71       188
```

Driver code

def main():

Building Phase

data = importdata()

X, Y, X_train, X_test, y_train, y_test = splitdataset(data)

clf_gini = train_using_gini(X_train, X_test, y_train)

clf_entropy = tarin_using_entropy(X_train, X_test, y_train)

Operational Phase

print("Results Using Gini Index:")

```
Results Using Gini Index:
Predicted values:
['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']
```

Prediction using gini

y_pred_gini = prediction(X_test, clf_gini)

cal_accuracy(y_test, y_pred_gini)

print("Results Using Entropy:")

```
Results Using Entropy:
Predicted values:
['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
 'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'
 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
 'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
 'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']
Confusion Matrix:  [[ 0  6  7]
 [ 0 63 22]
 [ 0 20 70]]
Accuracy :  70.74468085106383
Report :               precision    recall  f1-score   support

           B       0.00      0.00      0.00        13
           L       0.71      0.74      0.72        85
           R       0.71      0.78      0.74        90

    accuracy                           0.71       188
   macro avg       0.47      0.51      0.49       188
weighted avg       0.66      0.71      0.68       188
```

Prediction using entropy

y_pred_entropy = prediction(X_test, clf_entropy)

```
cal_accuracy(y_test, y_pred_entropy)
# Calling main function
if__name__== '__main__':
main()
```

CLASS PERFORMANCE	
RECORD	
VIVA	
TOTAL	

Result:

The above code has been executed and the output is verified successfully in Jupyter notebook.

EX NO: 6 Implement Naive Bayes Algorithm

Aim:

To perform a Naive Bayes algorithm for the given dataset using python in Google colab.

Procedure:

Step 1: Open google colab.

Step 2: Import the required libraries.

Step 3: Read the dataset (Iris)

Step 4: Perform the naive bayes algorithm and run the code

Step 5: End

Implementation:

```
import pandas as pd
import numpy as np
from sklearn import datasets
iris = datasets.load_iris() # importing the dataset
iris.data # showing the iris 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],
       [5.1, 3.8, 1.5, 0.3],
       [5.4, 3.4, 1.7, 0.2],
       [5.1, 3.7, 1.5, 0.4],
       [4.6, 3.6, 1. , 0.2],
       [5.1, 3.3, 1.7, 0.5],
       [4.8, 3.4, 1.9, 0.2],
       [5. , 3. , 1.6, 0.2],
       [5. , 3.4, 1.6, 0.4],
       [5.2, 3.5, 1.5, 0.2],
       [5.2, 3.4, 1.4, 0.2],
```

```
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

```
X=iris.data #assign the data to the X
y=iris.target #assign the target/flower type to the y
print (X.shape)
print (y.shape)
```

```
(150, 4)
(150,)
```

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,random_state=9

) #Split the dataset

from sklearn.naive_bayes import GaussianNB

nv = GaussianNB() # create a classifier

nv.fit(X_train,y_train) # fitting the data

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

from sklearn.metrics import accuracy_score

y_pred = nv.predict(X_test) # store the prediction data

```
1.0
```

accuracy_score(y_test,y_pred) # calculate the accuracy

CLASS PERFORMANCE	
RECORD	
VIVA	
TOTAL	

Result:

The above code has been executed and the output is verified.

EX NO: 7 Implement Support Vector Machine

Aim:

To implement support vector machine algorithm for the given dataset using python in google colab.

Procedure:

Step 1: Open google colab.

Step 2: Import the required libraries.

Step 3: Implement support vector machine algorithm

Step 4: Perform the naive bayes algorithm and run the code

Step 5: End

Implementation:

```
#Import scikit-learn dataset library
from sklearn import datasets

#Load dataset
cancer = datasets.load_breast_cancer()
# print the names of the 13 features
print("Features: ", cancer.feature_names)

# print the label type of cancer('malignant' 'benign')
print("Labels: ", cancer.target_names)
```

```
Features:  ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
Labels:  ['malignant' 'benign']
```

print data(feature)shape

cancer.data.shape

(569, 30)

print the cancer data features (top 5 records)

print(cancer.data[0:5])

```
[[1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
  1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
  6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
  1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
  4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 1.326e+03 8.474e-02 7.864e-02 8.690e-02
  7.017e-02 1.812e-01 5.667e-02 5.435e-01 7.339e-01 3.398e+00 7.408e+01
  5.225e-03 1.308e-02 1.860e-02 1.340e-02 1.389e-02 3.532e-03 2.499e+01
  2.341e+01 1.588e+02 1.956e+03 1.238e-01 1.866e-01 2.416e-01 1.860e-01
  2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 1.203e+03 1.096e-01 1.599e-01 1.974e-01
  1.279e-01 2.069e-01 5.999e-02 7.456e-01 7.869e-01 4.585e+00 9.403e+01
  6.150e-03 4.006e-02 3.832e-02 2.058e-02 2.250e-02 4.571e-03 2.357e+01
  2.553e+01 1.525e+02 1.709e+03 1.444e-01 4.245e-01 4.504e-01 2.430e-01
  3.613e-01 8.758e-02]
 [1.142e+01 2.038e+01 7.758e+01 3.861e+02 1.425e-01 2.839e-01 2.414e-01
  1.052e-01 2.597e-01 9.744e-02 4.956e-01 1.156e+00 3.445e+00 2.723e+01
  9.110e-03 7.458e-02 5.661e-02 1.867e-02 5.963e-02 9.208e-03 1.491e+01
  2.650e+01 9.887e+01 5.677e+02 2.098e-01 8.663e-01 6.869e-01 2.575e-01
  6.638e-01 1.730e-01]
 [2.029e+01 1.434e+01 1.351e+02 1.297e+03 1.003e-01 1.328e-01 1.980e-01
  1.043e-01 1.809e-01 5.883e-02 7.572e-01 7.813e-01 5.438e+00 9.444e+01
  1.149e-02 2.461e-02 5.688e-02 1.885e-02 1.756e-02 5.115e-03 2.254e+01
  1.667e+01 1.522e+02 1.575e+03 1.374e-01 2.050e-01 4.000e-01 1.625e-01
  2.364e-01 7.678e-02]]
```

print the cancer labels (0:malignant, 1:benign)

print(cancer.target)

```
[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
 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]
```

# Import train_test_split function
from sklearn.model_selection import train_test_split

Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(cancer.data, cancer.target, test_
size=0.3,random_state=109) # 70% training and 30% test
#Import svm model
from sklearn import svm

#Create a svm Classifier
clf = svm.SVC(kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

# Model Accuracy: how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

```
Accuracy: 0.9649122807017544
```

```
# Model Precision: what percentage of positive tuples are labeled as such?
print("Precision:",metrics.precision_score(y_test, y_pred))

# Model Recall: what percentage of positive tuples are labelled as such?
print("Recall:",metrics.recall_score(y_test, y_pred))
```

```
Precision: 0.9811320754716981
Recall: 0.9629629629629629
```

CLASS PERFORMANCE	
RECORD	
VIVA	
TOTAL	

Result:

The above code has been executed and the output is verified.

EX NO: 8 Implement Multilayer Perceptron

Aim:

To implement multilayer perceptron using python in google colab.

Procedure:

Step 1: Open google colab.

Step 2: Import the required libraries.

Step 3: Read the Data set(Iris)

Step 4: Implement the code

Step 5: End

Implementation:

```
import numpy as np
import pandas as pd
iris = pd.read_csv('/content/drive/MyDrive/Iris.csv')
iris = iris.sample(frac=1).reset_index(drop=True)
X = iris[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']]
X = np.array(X)
X[:5]
```

```
array([[5.8, 2.6, 4. , 1.2],
       [6.6, 2.9, 4.6, 1.3],
       [5.1, 3.8, 1.6, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [7.2, 3. , 5.8, 1.6]])
```

```
from sklearn.preprocessing import OneHotEncoder
one_hot_encoder = OneHotEncoder(sparse=False)
```

```
Y = iris.Species
Y = one_hot_encoder.fit_transform(np.array(Y).reshape(-1, 1))
Y[:5]
```

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

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.15)
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.1
)


def NeuralNetwork(X_train, Y_train, X_val=None, Y_val=None, epochs=10, n
odes=[], lr=0.15):
    hidden_layers = len(nodes) - 1
    weights = InitializeWeights(nodes)

    for epoch in range(1, epochs+1):
        weights = Train(X_train, Y_train, lr, weights)

        if(epoch % 20 == 0):
            print("Epoch {}".format(epoch))
            print("Training Accuracy:{}".format(Accuracy(X_train, Y_train, weight
s)))
            if X_val.any():
                print("Validation Accuracy:{}".format(Accuracy(X_val, Y_val, weigh
ts)))
```

```
    return weights
def InitializeWeights(nodes):
    """Initialize weights with random values in [-1, 1] (including bias)"""
    layers, weights = len(nodes), []

    for i in range(1, layers):
        w = [[np.random.uniform(-1, 1) for k in range(nodes[i-1] + 1)]
              for j in range(nodes[i])]
        weights.append(np.matrix(w))

    return weights
def ForwardPropagation(x, weights, layers):
    activations, layer_input = [x], x
    for j in range(layers):
        activation = Sigmoid(np.dot(layer_input, weights[j].T))
        activations.append(activation)
        layer_input = np.append(1, activation) # Augment with bias

    return activations


def BackPropagation(y, activations, weights, layers):
    outputFinal = activations[-1]
    error = np.matrix(y - outputFinal) # Error at output

    for j in range(layers, 0, -1):
        currActivation = activations[j]
```

```
        if(j > 1):
            # Augment previous activation
            prevActivation = np.append(1, activations[j-1])
        else:
            # First hidden layer, prevActivation is input (without bias)
            prevActivation = activations[0]

        delta = np.multiply(error, SigmoidDerivative(currActivation))
        weights[j-1] += lr * np.multiply(delta.T, prevActivation)
        w = np.delete(weights[j-1], [0], axis=1) # Remove bias from weights
        error = np.dot(delta, w) # Calculate error for current layer

    return weights
def Train(X, Y, lr, weights):
     layers = len(weights)
    for i in range(len(X)):
         x, y = X[i], Y[i]
        x = np.matrix(np.append(1, x)) # Augment feature vector

        activations = ForwardPropagation(x, weights, layers)
        weights = BackPropagation(y, activations, weights, layers)

    return weights
def Sigmoid(x):
    return 1 / (1 + np.exp(-x))

def SigmoidDerivative(x):
    return np.multiply(x, 1-x)
def Predict(item, weights):
```

```
    layers = len(weights)
    item = np.append(1, item) # Augment feature vector

    ##_Forward Propagation_##
    activations = ForwardPropagation(item, weights, layers)

    outputFinal = activations[-1].A1
    index = FindMaxActivation(outputFinal)

    # Initialize prediction vector to zeros
    y = [0 for i in range(len(outputFinal))]
    y[index] = 1 # Set guessed class to 1
def Accuracy(X, Y, weights):
    """Run set through network, find overall accuracy"""
    correct = 0

    for i in range(len(X)):
        x, y = X[i], list(Y[i])
        guess = Predict(x, weights)

        if(y == guess):
            # Guessed correctly
            correct += 1

    return correct / len(X)
def Accuracy(X, Y, weights):
    """Run set through network, find overall accuracy"""
    correct = 0
```

```
    for i in range(len(X)):
        x, y = X[i], list(Y[i])
        guess = Predict(x, weights)

        if(y == guess):
            # Guessed correctly
            correct += 1

    return correct / len(X)
f = len(X[0]) # Number of features
o = len(Y[0]) # Number of outputs / classes

layers = [f, 5, 10, o] # Number of nodes in layers
lr, epochs = 0.15, 100

weights = NeuralNetwork(X_train, Y_train, X_val, Y_val, epochs=epochs, node
s=layers, lr=lr);
```

```
Epoch 20
Training Accuracy:0.7105263157894737
Validation Accuracy:0.46153846153846156
Epoch 40
Training Accuracy:0.9385964912280702
Validation Accuracy:1.0
Epoch 60
Training Accuracy:0.9649122807017544
Validation Accuracy:1.0
Epoch 80
Training Accuracy:0.9385964912280702
Validation Accuracy:1.0
Epoch 100
Training Accuracy:0.9298245614035088
Validation Accuracy:1.0
```

```
print("Testing Accuracy: {}".format(Accuracy(X_test, Y_test, weights)))
```

```
Testing Accuracy: 0.9565217391304348
```

CLASS PERFORMANCE	
RECORD	
VIVA	
TOTAL	

Result:

The above code has been executed and the output is verified

EX NO: 9 Implement Bagging Methods

DATE:

Aim:

To implement bagging methods using python in google colab.

Procedure:

Step 1: Open google colab.

Step 2: Import the required libraries.

Step 3: Implement Bagging methods

Step 4: Run the code

Step 5: End

Implementation:

```
# check scikit-learn version
import sklearn
print(sklearn._version_)
```

```
0.22.2.post1
```

```
from sklearn.datasets import make_classification
# define dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15,
 n_redundant=5, random_state=5)
# summarize the dataset
print(X.shape, y.shape)
```

```
(1000, 20) (1000,)
```

```
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import BaggingClassifier
# define dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15,
 n_redundant=5, random_state=5)
# define the model
model = BaggingClassifier()
# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1,
error_score='raise')
# report performance
print('Accuracy: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
Accuracy: 0.860 (0.043)
```

```
# make predictions using bagging for classification
from sklearn.datasets import make_classification
from sklearn.ensemble import BaggingClassifier
# define dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=15,
 n_redundant=5, random_state=5)
# define the model
model = BaggingClassifier()
# fit the model on the whole dataset
model.fit(X, y)
```

```
# make a single prediction
row = [[-4.7705504,-1.88685058,-0.96057964,2.53850317,-6.5843005,3.45711
663,-7.46225013,2.01338213,-0.45086384,-1.89314931,-2.90675203,-0.212145
68,-0.9623956,3.93862591,0.06276375,0.33964269,4.0835676,1.31423977,-2.1
7983117,3.1047287]]
yhat = model.predict(row)
print('Predicted Class: %d' % yhat[0])
```

```
Predicted Class: 1
```

```
# test regression dataset
from sklearn.datasets import make_regression
# define dataset
X, y = make_regression(n_samples=1000, n_features=20, n_informative=15, no
ise=0.1, random_state=5)
# summarize the dataset
print(X.shape, y.shape)
```

```
(1000, 20) (1000,)
```

```
# evaluate bagging ensemble for regression
from numpy import mean
from numpy import std
from sklearn.datasets import make_regression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedKFold
from sklearn.ensemble import BaggingRegressor
# define dataset
X, y = make_regression(n_samples=1000, n_features=20, n_informative=15, no
ise=0.1, random_state=5)
```

```
# define the model
model = BaggingRegressor()
# evaluate the model
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, y, scoring='neg_mean_absolute_error', cv
=cv, n_jobs=-1, error_score='raise')
# report performance
print('MAE: %.3f (%.3f)' % (mean(n_scores), std(n_scores)))
```

```
MAE: -100.406 (10.336)
```

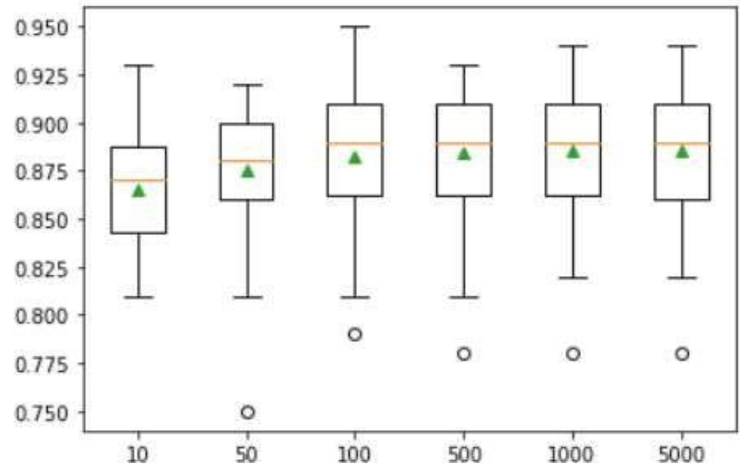
```
# bagging ensemble for making predictions for regression
from sklearn.datasets import make_regression
from sklearn.ensemble import BaggingRegressor
# define dataset
X, y = make_regression(n_samples=1000, n_features=20, n_informative=15, no
ise=0.1, random_state=5)
# define the model
model = BaggingRegressor()
# fit the model on the whole dataset
model.fit(X, y)
# make a single prediction
row = [[0.88950817,-0.93540416,0.08392824,0.26438806,-0.52828711,-1.2110
2238,-0.4499934,1.47392391,-0.19737726,-0.22252503,0.02307668,0.2695327
6,0.03572757,-0.51606983,-0.39937452,1.8121736,-0.00775917,-0.02514283,-
0.76089365,1.58692212]]
yhat = model.predict(row)
print('Prediction: %d' % yhat[0])
```

```
Prediction: -207
```

```
# explore bagging ensemble number of trees effect on performance
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.ensemble import BaggingClassifier
from matplotlib import pyplot

# get the dataset
def get_dataset():
  X, y = make_classification(n_samples=1000, n_features=20, n_informative=1
5, n_redundant=5, random_state=5)
  return X, y

# get a list of models to evaluate
def get_models():
  models = dict()
  # define number of trees to consider
  n_trees = [10, 50, 100, 500, 500, 1000, 5000]
  for n in n_trees:
    models[str(n)] = BaggingClassifier(n_estimators=n)
  return models

# evaluate a given model using cross-validation
def evaluate_model(model, X, y):
  # define the evaluation procedure
```

```
  cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
  # evaluate the model and collect the results
  scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
  return scores

# define dataset
X, y = get_dataset()
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
  # evaluate the model
  scores = evaluate_model(model, X, y)
  # store the results
  results.append(scores)
  names.append(name)
  # summarize the performance along the way
  print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
```

```
>10 0.865 (0.032)
>50 0.875 (0.036)
>100 0.882 (0.035)
>500 0.884 (0.036)
>1000 0.885 (0.036)
>5000 0.885 (0.036)
```



CLASS PERFORMANCE	
RECORD	
VIVA	
TOTAL	

Result:

The above code has been executed and the output is verified.

EX NO: 10 **Implement Boosting Methods**

Aim:

To implement boosting methods using python in google colab.

Procedure:

Step 1: Open google colab.

Step 2: Import the required libraries.

Step 3: Implement Boosting methods

Step 4: Run the code

Step 5: End

Implementation:

```
from sklearn.datasets import load_boston
boston = load_boston()
print(boston.keys())
```

```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

```
print(boston.data.shape)
```

```
(506, 13)
```

```
print(boston.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

```
import pandas as pd
```

data = pd.DataFrame(boston.data)

data.columns = boston.feature_names

data.head()

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

data['PRICE'] = boston.target

data.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  PRICE    506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

data.describe()

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.653063	22.532806
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.141062	9.197104
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.730000	5.000000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.950000	17.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.360000	21.200000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.955000	25.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.970000	50.000000

```
import xgboost as xgb
from sklearn.metrics import mean_squared_error
import pandas as pd
import numpy as np
X, y = data.iloc[:,:-1],data.iloc[:,-1]
data_dmatrix = xgb.DMatrix(data=X,label=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, random_st
ate=123)
xg_reg = xgb.XGBRegressor(objective ='reg:linear', colsample_bytree = 0.3, lea
rning_rate = 0.1,
                max_depth = 5, alpha = 10, n_estimators = 10)
xg_reg.fit(X_train,y_train)

preds = xg_reg.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, preds))
print("RMSE: %f" % (rmse))
```

```
RMSE: 10.449300
```

```
params = {"objective":"reg:linear",'colsample_bytree': 0.3,'learning_rate': 0.1,
                'max_depth': 5, 'alpha': 10}
```

```
cv_results = xgb.cv(dtrain=data_dmatrix, params=params, nfold=3,
                    num_boost_round=50,early_stopping_rounds=10,metrics="rmse",
as_pandas=True, seed=123)
cv_results.head()
```

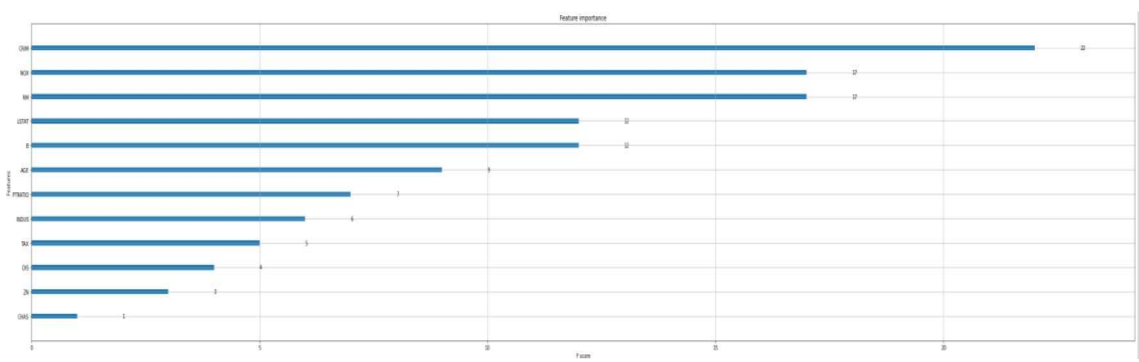
	train-rmse-mean	train-rmse-std	test-rmse-mean	test-rmse-std
0	21.679234	0.025626	21.677613	0.050617
1	19.772293	0.016054	19.773320	0.020571
2	18.049563	0.058904	18.065856	0.082746
3	16.430113	0.013615	16.492199	0.020919
4	15.025977	0.062132	15.132644	0.073104

```
print((cv_results["test-rmse-mean"]).tail(1))
```

```
49    3.975679
Name: test-rmse-mean, dtype: float64
```

```
import matplotlib.pyplot as plt


xgb.plot_tree(xg_reg,num_trees=0)
plt.rcParams['figure.figsize'] = [50, 10]
plt.show()
```

```
xgb.plot_importance(xg_reg)
plt.rcParams['figure.figsize'] = [5, 5]
plt.show()
```



CLASS PERFORMANCE	
RECORD	
VIVA	
TOTAL	

Result:

The above code has been executed and the output is verified.