PROGRAM NO: 1

AIM :

Write a program to create an element-wise comparison (greater, greater than or equal to, less, less than or equal to).

CODE :

```
import numpy as np
x = np.array([3,5,1,2,3])
y = np.array([2,5,3,2,1])
print("Array A")
print(x)
print("\nArray B")
print(y)
print("\nA>B")
print(np.greater(x, y))
print("\nA>=B")
print(np.greater_equal(x, y))
print("\nA<B")
print(np.less(x, y))
print("\nA<=B")
print(np.less_equal(x, y))
```

OUTPUT :

Array A

[3 5 1 2 3]

Array B

[2 5 3 2 1]

A>B

[True False False False True]

A>=B

[True True False True True]

A<B

[False False True False False]

A<=B

[False True True True False]

PROGRAM NO: 2

AIM :

Write a program to find the transpose of a matrix.

CODE :

```
import numpy as np
b = np.array([[1,2,3,4],[4,5,6,7]])
print(b)
print(b.transpose())
```

OUTPUT :

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

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

PROGRAM NO: 3

AIM :

Write a program to find the rank of a matrix.

CODE :

```
import numpy as np
def find_rank(matrix):
  return np.linalg.matrix_rank(matrix)
rows = int(input("Enter the number of rows: "))
columns = int(input("Enter the number of columns: "))
matrix = []
print("Enter the elements row wise : ")
for i in range(rows):
  a = []
  for j in range(columns):
    a.append(int(input()))
  matrix.append(a)
rank = find_rank(matrix)
print("The Given Matrix: \n",matrix)
print("Rank of the matrix is : ",rank)
```

OUTPUT :

Enter the number of rows: 3

Enter the number of columns: 3

Enter the elements row wise :

3

5

6

7

8

9

1

2

5

The Given Matrix:

[[3, 5, 6], [7, 8, 9], [1, 2, 5]]

Rank of the matrix is : 3

PROGRAM NO: 4

AIM :

Write a python program to generate the series of dates from 1st May, 2021 to 12th May, 2021 (both inclusive).

CODE :

```
import pandas as pd

sr = pd.Series(pd.date_range('2021-05-01','2021-05-12',freq='D'))

print(sr.to_string(index=False))
```

OUTPUT :

```
2021-05-01
2021-05-02
2021-05-03
2021-05-04
2021-05-05
2021-05-06
2021-05-07
2021-05-08
2021-05-09
2021-05-10
2021-05-11
2021-05-12
```

PROGRAM NO: 5

AIM :

Given a dataframe with custom indexing, write a python program to convert the index to default indexing and display it.

data = {'Name': ['e','a','a','b','c','d'] , 'Age': [1,2,1,3,3,4] , 'Rank': [0,1,2,3,4,5]}

index = ['a1', 'b1', 'c1', 'd1', 'e1','f1']

CODE :

```
import pandas as pd

data = {'Name': ['e','a','a','b','c','d'],

            'Age': [1,2,1,3,3,4],

            'Rank': [0,1,2,3,4,5]}

index = ['a1', 'b1', 'c1', 'd1', 'e1','f1']

df = pd.DataFrame(data,index)

df.reset_index(inplace=True,drop=True)

print(df.to_string())
```

OUTPUT :

```
Name  Age  Rank

0    e    1    0

1    a    2    1

2    a    1    2

3    b    3    3

4    c    3    4

5    d    4    5
```

PROGRAM NO: 6

AIM :

Given is a dataframe showing name, occupation, salary of people. Write a python program to find the average salary per occupation.

details = {'Name' : ['a','b','c','d','e'] , 'Occupation' : ['A1','A1','A1','B1','B1'] , 'Salary' : [20,30,40,27,23] }

CODE :

```
import pandas as pd

details = { 'Name' : ['a','b','c','d','e'] , 'Occupation' : ['A1','A1','A1','B1','B1'] , 'Salary' :
[20,30,40,27,23], }

data = pd.DataFrame(details)

print(data)

occ_avg_salary = data.groupby('Occupation')['Salary'].mean()

print("Average salary for each occupation:")

print(occ_avg_salary)
```

OUTPUT :

```
Name Occupation  Salary
0    a        A1      20
1    b        A1      30
2    c        A1      40
3    d        B1      27
4    e        B1      23
Average salary for each occupation:
Occupation
A1    30.0
B1    25.0
Name: Salary, dtype: float64
```

PROGRAM NO: 7

AIM :

Write a program to draw a scatter plot comparing two subject marks of Mathematics and Science. Use marks of 10 students.

math_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]

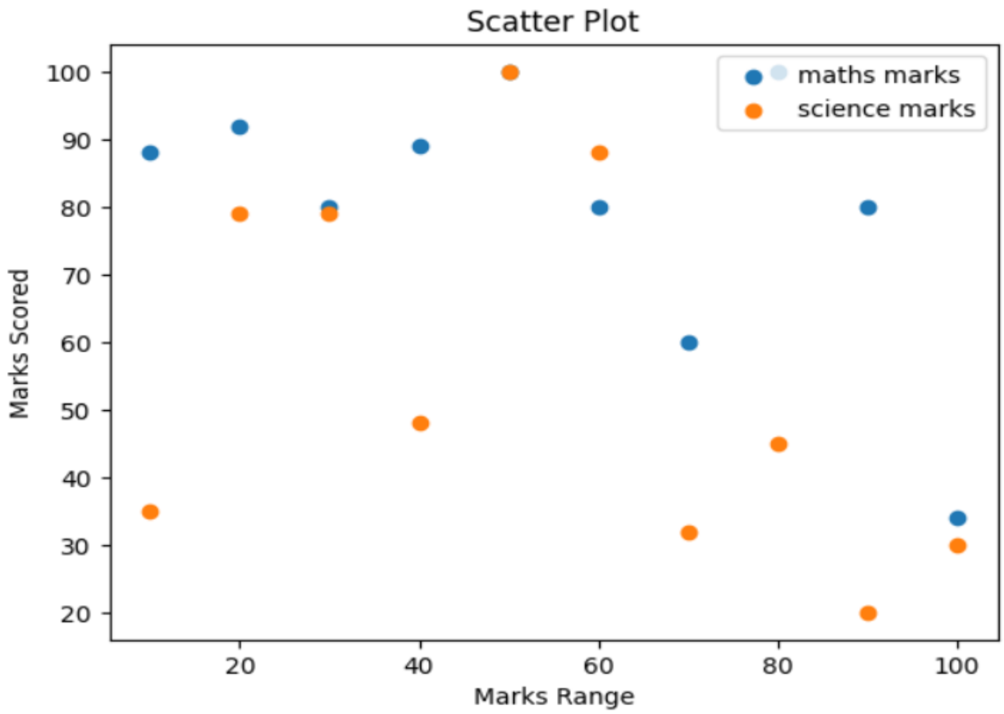
science_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]

marks_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

CODE :

```
import matplotlib.pyplot as plt

x=[10,20,30,40,50,60,70,80,90,100]

m=[88,92,80,89,100,80,60,100,80,34]

s=[35,79,79,48,100,88,32,45,20,30]

plt.scatter(x,m,label='maths marks')

plt.scatter(x,s,label='science marks')

plt.legend(loc='upper right')

plt.xlabel('Marks Range')

plt.ylabel('Marks Scored')

plt.title('Scatter Plot')

plt.show()
```

OUTPUT :


Scatter Plot
maths marks
science marks
Marks Scored
Marks Range
100
90
80
70
60
50
40
30
20
20
40
60
80
100

PROGRAM NO: 8

AIM :

Write a program to create bar plot of scores by group and gender. Use multiple X values on the same chart for men and women.

Sample Data:

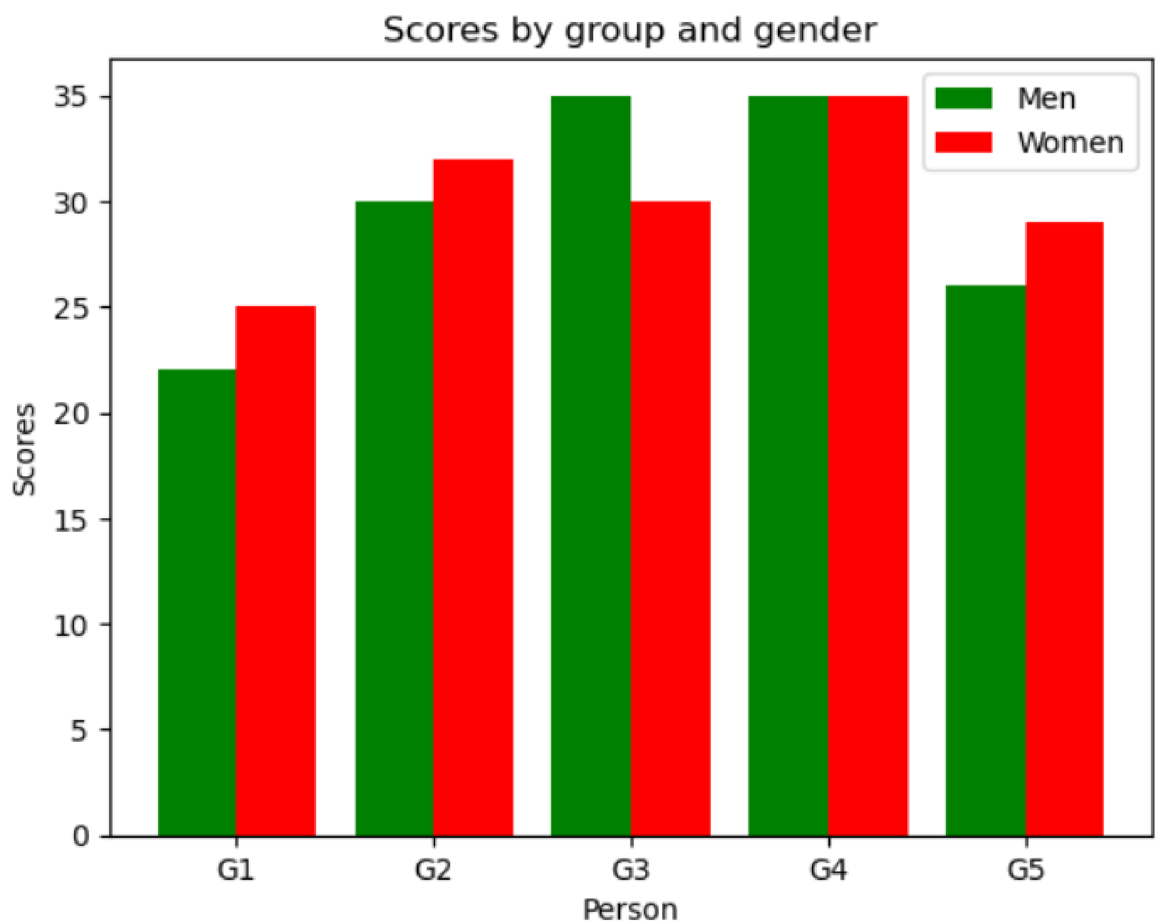
Means (men) = (22, 30, 35, 35, 26)

Means (women) = (25, 32, 30, 35, 29)

CODE :

```
import numpy as np
import matplotlib.pyplot as plt
y1=[22,30,35,35,26]
y2=[25,32,30,35,29]
x_labels = ['G1','G2','G3','G4','G5']
x1 = np.arange(5)
width = 0.40
plt.bar(x1-0.2,y1,color='green',width=width,label='Men')
plt.bar(x1+0.2,y2,color='red',width=width,label='Women')
plt.xticks(x1,x_labels)
plt.xlabel("Person")
plt.ylabel("Scores")
plt.legend()
plt.title("Scores by group and gender")
plt.show()
```

OUTPUT :


Scores by group and gender
Men
Women
Scores
35
30
25
20
15
10
5
0
G1
G2
G3
G4
G5
Person

PROGRAM NO: 9

AIM :

Write a program to create a pie chart of the popularity of programming Languages.

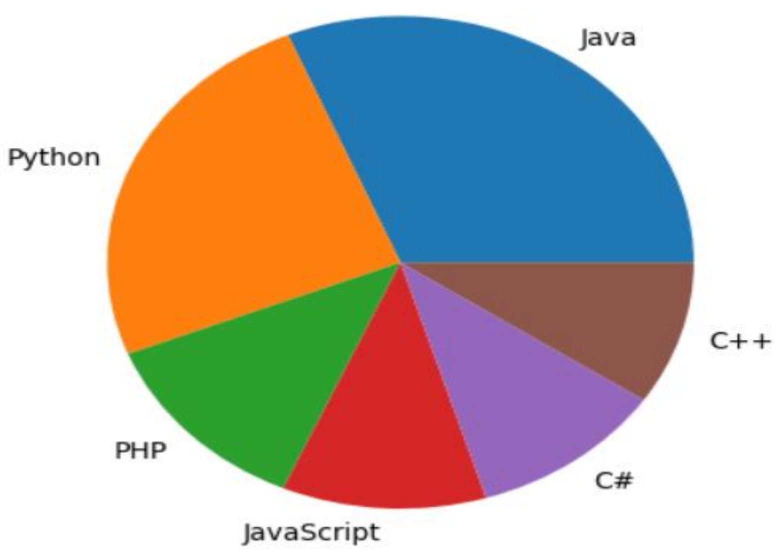
Programming languages: Java Python PHP JavaScript C# C++

Popularity ``````: 22.2 17.6 8.8 8 7.7 6.7

CODE :

```
import matplotlib.pyplot as plt

import numpy as np

y=np.array([22.2,17.6,8.8,8,7.7,6.7])

mylabels = ["Java","Python","PHP","JavaScript","C#","C++"]

plt.pie(y,labels=mylabels)

plt.show()
```

OUTPUT :



PROGRAM NO: 10

AIM :

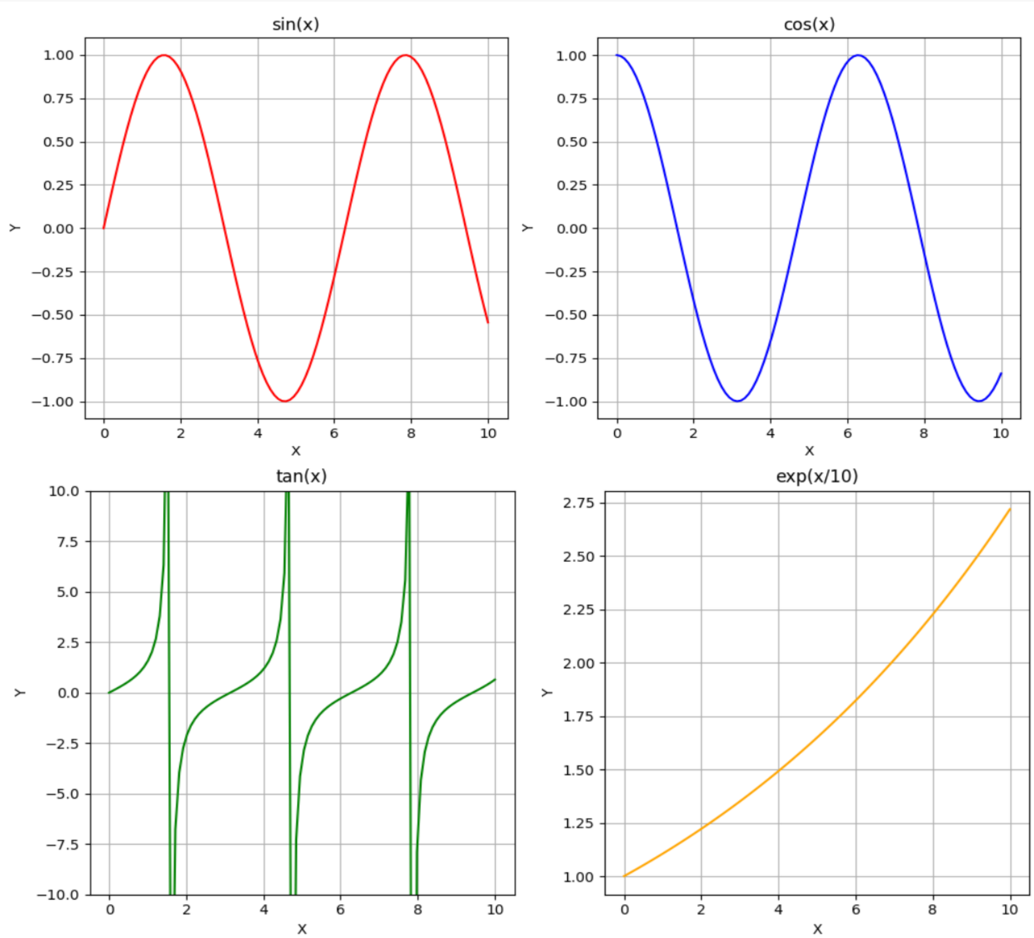
Write a python program to create multiple plots.

CODE :

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,10,100)
y1 = np.sin(x)
y2 = np.cos(x)
y3 = np.tan(x)
y4 = np.exp(x/10)
fig,axs = plt.subplots(2,2,figsize=(10,10))
axs[0,0].plot(x,y1,color='red')
axs[0,0].set_title('sin(x)')
axs[0,0].set_xlabel('X')
axs[0,0].set_ylabel('Y')
axs[0,0].grid()
axs[0,1].plot(x,y2,color='blue')
axs[0,1].set_title('cos(x)')
axs[0,1].set_xlabel('X')
axs[0,1].set_ylabel('Y')
axs[0,1].grid()
axs[1,0].plot(x,y3,color='green')
axs[1,0].set_title('tan(x)')
axs[1,0].set_xlabel('X')
axs[1,0].set_ylabel('Y')
```

```
axs[1,0].grid()
axs[1,0].set_ylim(-10,10)
axs[1,1].plot(x,y4,color='orange')
axs[1,1].set_title('exp(x/10)')
axs[1,1].set_xlabel('X')
axs[1,1].set_ylabel('Y')
axs[1,1].grid()
plt.tight_layout()
plt.show()
```

OUTPUT :



PROGRAM NO: 11

AIM :

Write a program to implement data visualisation for Iris dataset using matplotlib and seaborn.

CODE :

```
import matplotlib.pyplot as plt

import matplotlib.lines as mlines

import csv

import math

import numpy as np

import pandas as pd

import seaborn as sns

from pandas.plotting import parallel_coordinates

from pandas.plotting import andrews_curves

import plotly.express as p


df = pd.read_csv("C:/Users/crana/Downloads/iris - iris.csv")

df.info()
```

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

```
df.describe()
```

	sepal.length	sepal.width	petal.length	petal.width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

df.head()

	sepal.length	sepal.width	petal.length	petal.width	variety
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
2	4.7	3.2	1.3	0.2	Setosa
3	4.6	3.1	1.5	0.2	Setosa
4	5.0	3.6	1.4	0.2	Setosa

df.isnull()

	sepal.length	sepal.width	petal.length	petal.width	variety
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
...	...	...	...	...	...
145	False	False	False	False	False
146	False	False	False	False	False
147	False	False	False	False	False
148	False	False	False	False	False
149	False	False	False	False	False

df.isna().sum()

	0
sepal.length	0
sepal.width	0
petal.length	0
petal.width	0
variety	0

dtype: int64

df.value_counts('variety')

	count
variety	
Setosa	50
Versicolor	50
Virginica	50

dtype: int64

Histogram

plt.figure(figsize=(10, 7))

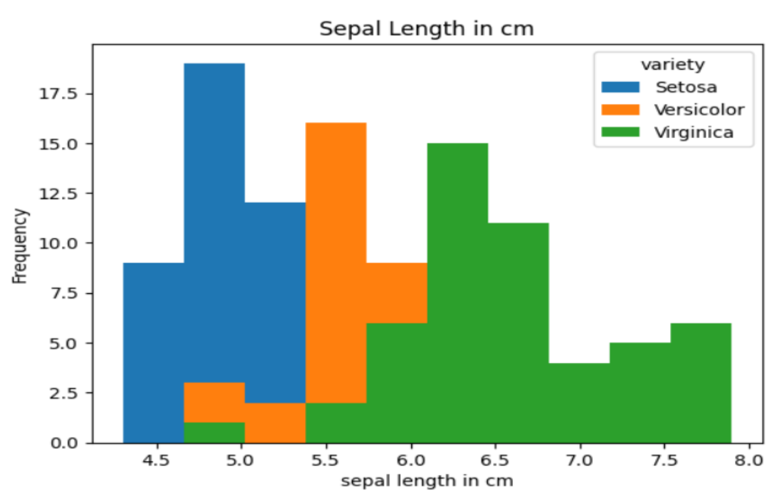
x = pd.DataFrame({"sepallength":df['sepal.length'],"variety":df['variety']})

x.pivot(columns="variety",values="sepallength").plot.hist()

plt.title("Sepal Length in cm")

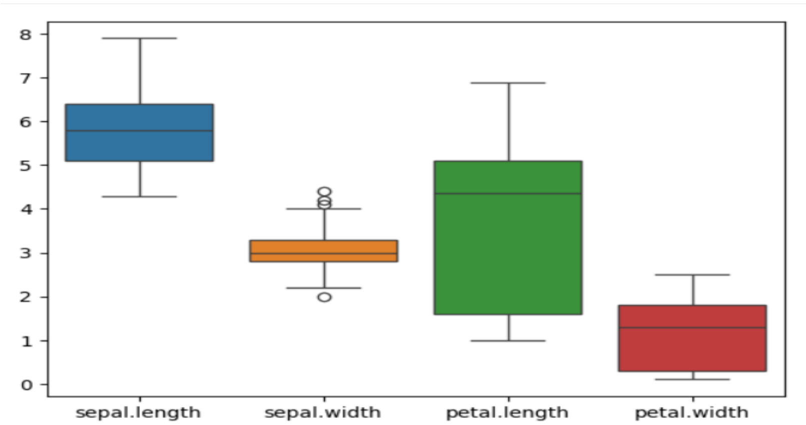
plt.xlabel("sepal length in cm")

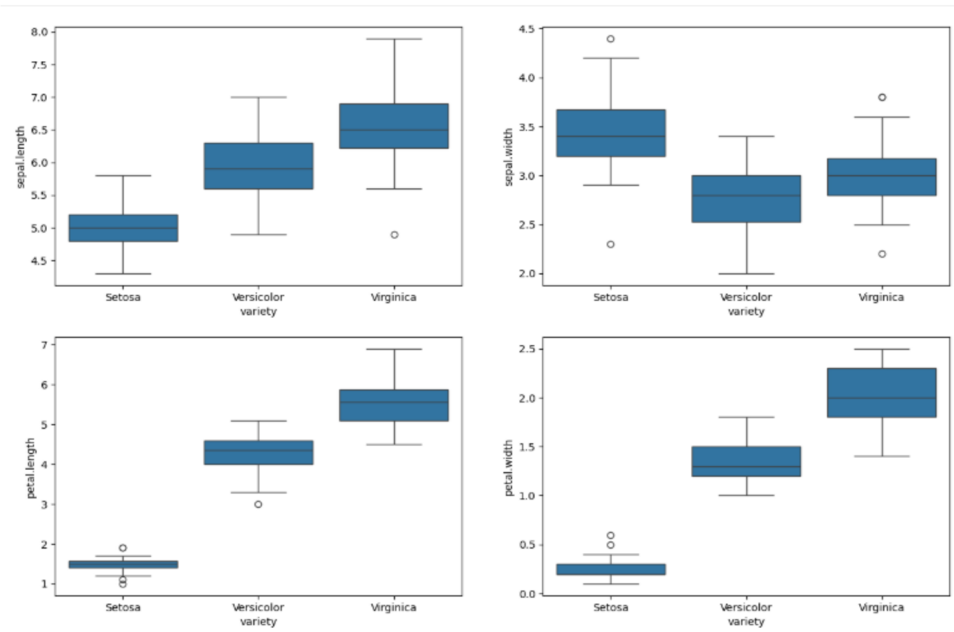
plt.ylabel("Frequency")



Quartile(Box) Plot

```
sns.boxplot(data=df)
plt.show()

figure,axis = plt.subplots(2,2,figsize=(15,10))
sns.boxplot(x="variety",y="sepal.length",data=df,ax=axis[0,0])
sns.boxplot(x="variety",y="sepal.width",data=df,ax=axis[0,1])
sns.boxplot(x="variety",y="petal.length",data=df,ax=axis[1,0])
sns.boxplot(x="variety",y="petal.width",data=df,ax=axis[1,1])
plt.show()
```





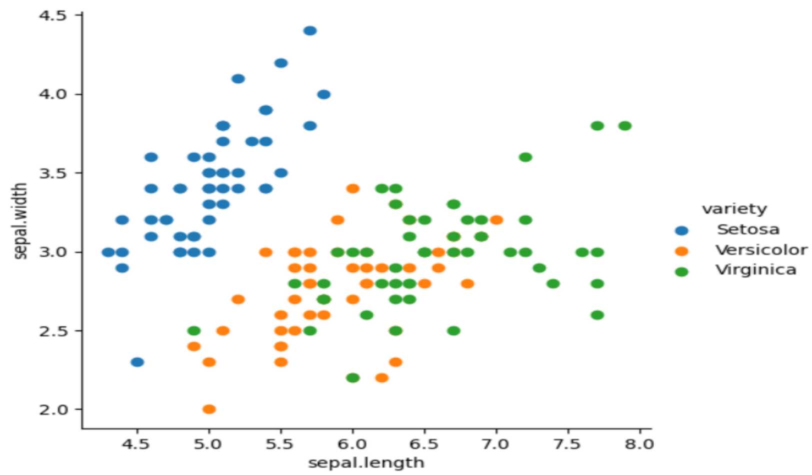
Scatter Plot

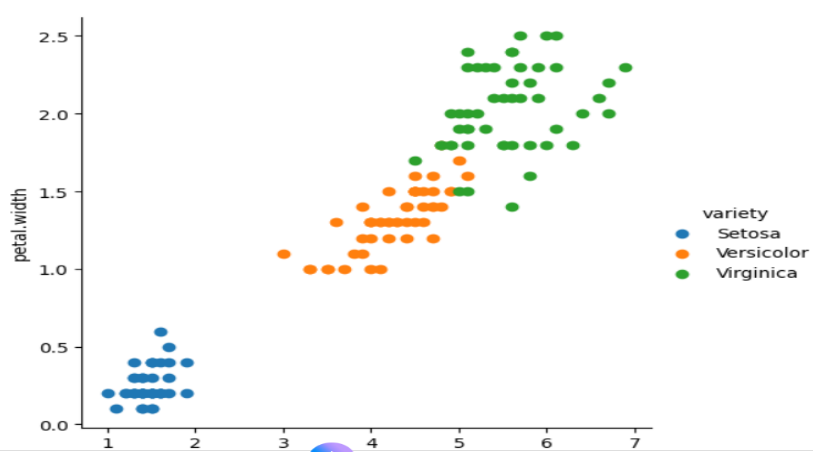
sns.FacetGrid(df,hue="variety",height=5).map(plt.scatter,"sepal.length","sepal.width").add_legend()

plt.show()

sns.FacetGrid(df,hue="variety",height=5).map(plt.scatter,"petal.length","petal.width").add_legend()

plt.show()





Scatter Multiple

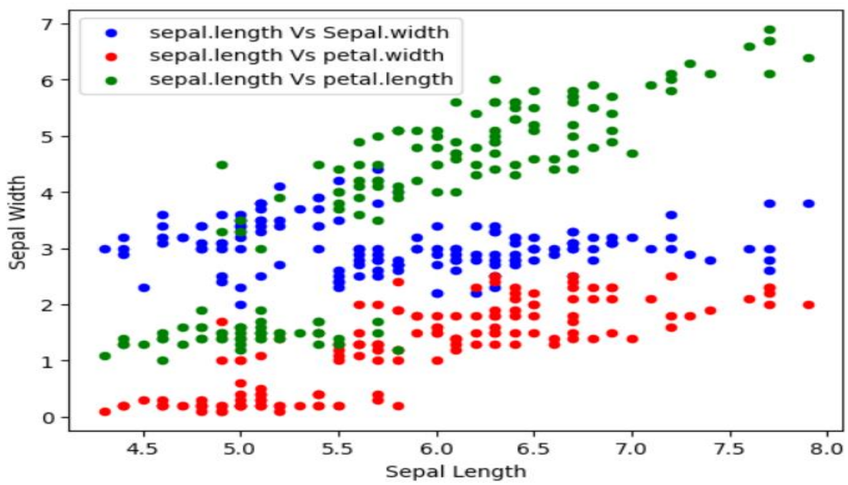
ax = df.plot(kind="scatter",x="sepal.length",y="sepal.width",color="blue",label="sepal.length Vs Sepal.width") df.plot(kind="scatter",x="sepal.length",y="petal.width",color="red",label="sepal.length Vs petal.width",ax=ax)

df.plot(kind="scatter",x="sepal.length",y="petal.length",color="green",label="sepal.length Vs petal.length",ax=ax)

ax.set_xlabel("Sepal Length")

ax.set_ylabel("Sepal Width")

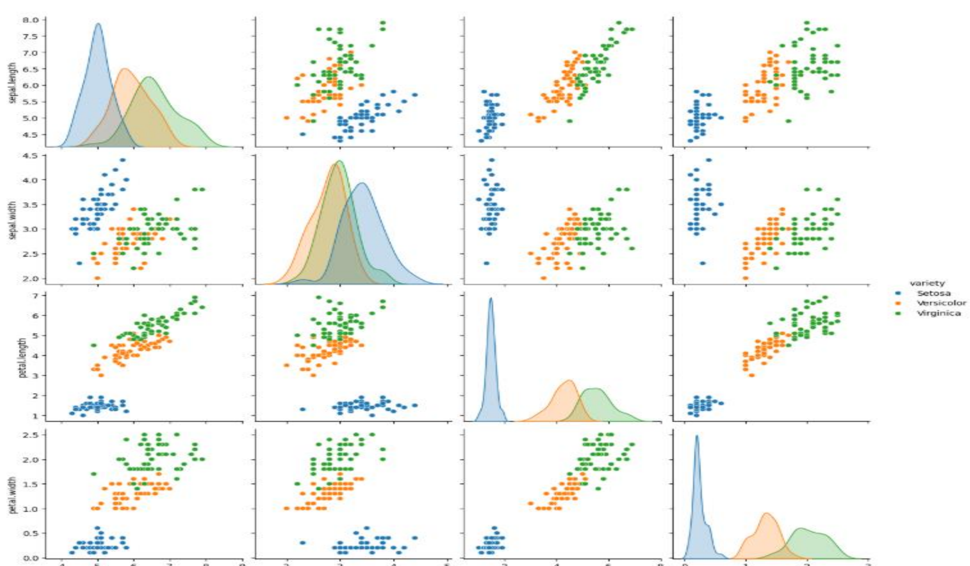
plt.show()



Scatter Matrix

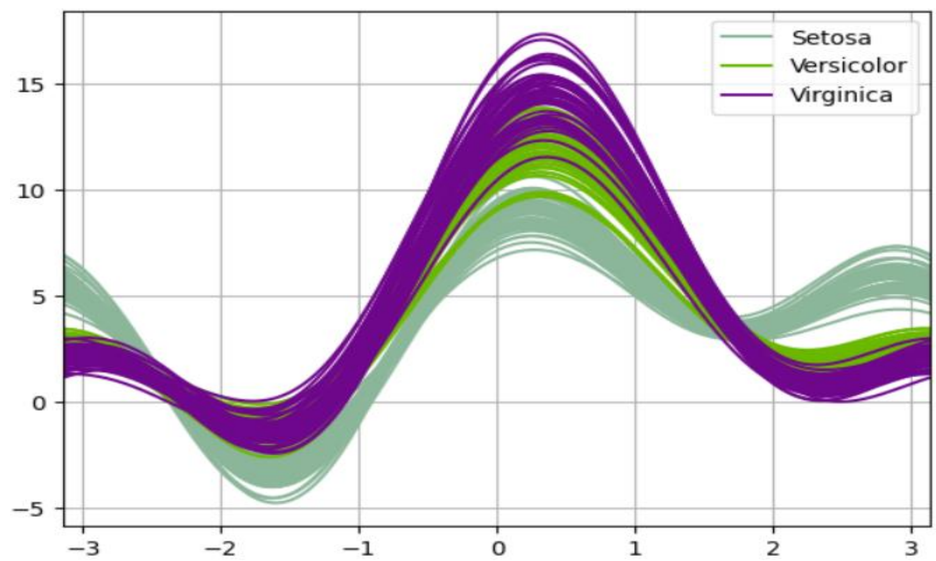
sns.pairplot(df,hue="variety",height=3)

plt.show()



Andrews Curves

```
andrews_curves(df, "variety")

plt.show()
```



PROGRAM NO: 12

AIM :

Write a program to implement K-Nearest Neighbour(KNN) algorithm to predict the name of the flower for given sample using Iris dataset.

CODE :

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
iris = load_iris()
x = iris.data
y = iris.target
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=1)
c_knn = KNeighborsClassifier(n_neighbors=3)
c_knn.fit(x_train,y_train)
y_pred = c_knn.predict(x_test)
print("Accuracy : ",metrics.accuracy_score(y_test,y_pred))
sample = [[2,3,2,5]]
pred = c_knn.predict(sample)
pred_v = [iris.target_names[p] for p in pred]
print(pred_v)
```

OUTPUT :

```
Accuracy :  0.9777777777777777
[np.str_('versicolor')]
```

PROGRAM NO: 13

AIM :

Write a program to implement K-Nearest Neighbour(KNN) algorithm to predict whether the cancer is Malignant or Benign for given sample using Cancer dataset.

CODE :

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score

data = pd.read_csv("C:/Users/Student/Downloads/cancer.csv")
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perir
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	

5 rows × 32 columns

```
data.drop("id",axis=1)
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	...	radius_worst	texture_wors
0	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	...	25.380	17.3
1	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	...	24.990	23.4
2	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	...	23.570	25.5
3	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	...	14.910	26.5
4	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	...	22.540	16.6
...	...	...	...	...	...	...	...	...	...	...	...	...	...
564	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	...	25.450	26.4
565	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	...	23.690	38.2
566	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	...	18.980	34.1
567	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	...	25.740	39.4
568	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	...	9.456	30.3

569 rows × 31 columns

data['diagnosis'].unique()

```
array(['M', 'B'], dtype=object)
```

label_encoder = LabelEncoder()

data['diagnosis'] = label_encoder.fit_transform(data['diagnosis'])

data.head()

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perim
0	842302	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	
1	842517	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	
2	84300903	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	
3	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	
4	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	

train , test = train_test_split(data, test_size=0.3)

trainX = train[data.columns[2:-1]]

trainY = train[data.columns[1]]

testX = test[data.columns[2:-1]]

testY = test[data.columns[1]]

c_knn = KNeighborsClassifier(n_neighbors=3)

c_knn.fit(trainX.values,trainY.values)

y_pred = c_knn.predict(testX.values)

print("Accuracy : ",accuracy_score(testY.values,y_pred))

```
Accuracy :  0.9239766081871345
```

sample = [[15.78,17.89,103.6,781,0.0971,0.1292,0.09954,0.06606,0.1842,0.06082,0.5058,0.9849,3.564,54.16,0.005771,0.04061,0.02791,0.01282,0.02008,0.004144,20.42,27.28,136.5,1299,0.1396,0.5609,0.3965,0.181,0.3792]]

pred = c_knn.predict(sample)

```
res = pred[0]
if res==0:
    print("Benign")
if res==1:
    print("Malignant")
```

```
Malignant
```

PROGRAM NO: 14

AIM :

Write a program to implement Naive Bayes algorithm to predict the name of the flower for given sample using Iris dataset.

CODE :

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB


X,y = load_iris(return_X_y=True)

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.5,random_state=0)

gnb=GaussianNB()

y_pred = gnb.fit(X_train,y_train).predict(X_test)

print(y_pred)

x_new = [[5,5,4,4]]

y_new = gnb.fit(X_train,y_train).predict(x_new)

print("Predicted output for [[5,5,4,4]] : ",y_new)

print("Naive Bayes score : ",gnb.score(X_test,y_test))
```

OUTPUT :

```
[2 1 0 2 0 2 0 1 1 1 1 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 1 1 1 2 0 2 0 0 1 2 2 1 2 1 2 1 1 2 1 1 2 1 2 1 0 2 1 1 1 1 2 0 0 2 1 0 0
 1]
Predicted output for [[5,5,4,4]] :  [2]
Naive Bayes score :  0.9466666666666667
```

PROGRAM NO: 15

AIM :

Write a program to implement Naïve Bayes algorithm to predict whether the cancer is Malignant or Benign for given sample using cancer dataset.

CODE :

```
!pip install mlxtend

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

import matplotlib.pyplot as plt

from sklearn import metrics

from sklearn.metrics import confusion_matrix

from mlxtend.plotting import plot_confusion_matrix

from sklearn import preprocessing


df = pd.read_csv("C:/Users/Student/Downloads/cancer.csv")

df
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	pe
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	...	25.380	17.33	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	24.990	23.41	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	23.570	25.53	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	14.910	26.50	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	22.540	16.67	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	...	25.450	26.40	
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	...	23.690	38.25	
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	...	18.980	34.12	
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	...	25.740	39.42	
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	...	9.456	30.37	

569 rows × 32 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      569 non-null    int64
 1   diagnosis               569 non-null    object
 2   radius_mean             569 non-null    float64
 3   texture_mean            569 non-null    float64
 4   perimeter_mean          569 non-null    float64
 5   area_mean               569 non-null    float64
 6   smoothness_mean         569 non-null    float64
 7   compactness_mean        569 non-null    float64
 8   concavity_mean          569 non-null    float64
 9   concave points_mean     569 non-null    float64
 10  symmetry_mean           569 non-null    float64
 11  fractal_dimension_mean  569 non-null    float64
 12  radius_se               569 non-null    float64
 13  texture_se              569 non-null    float64
 14  perimeter_se            569 non-null    float64
 15  area_se                 569 non-null    float64
 16  smoothness_se           569 non-null    float64
 17  compactness_se          569 non-null    float64
 18  concavity_se            569 non-null    float64
 19  concave points_se       569 non-null    float64
 20  symmetry_se             569 non-null    float64
 21  fractal_dimension_se    569 non-null    float64
```

df.head()

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	perir
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	

5 rows × 32 columns

df['diagnosis'].unique()

```
array(['M', 'B'], dtype=object)
```

label_encoder = preprocessing.LabelEncoder()

df['diagnosis'] = label_encoder.fit_transform(df['diagnosis'])

df['diagnosis']

```
0      1
1      1
2      1
3      1
4      1
      ..
564    1
565    1
566    1
567    1
568    0
Name: diagnosis, Length: 569, dtype: int64
```

df.columns

```
Index(['id', 'diagnosis', 'radius_mean', 'texture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

df.describe()

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_wo
count	5.690000e+02	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	...	569.000000	569.000
mean	3.037183e+07	0.372583	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	...	16.269190	25.677
std	1.250206e+08	0.483918	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	...	4.833242	6.146
min	8.670000e+03	0.000000	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	...	7.930000	12.020
25%	8.692180e+05	0.000000	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	...	13.010000	21.080
50%	9.060240e+05	0.000000	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	...	14.970000	25.410
75%	8.813129e+06	1.000000	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	...	18.790000	29.720
max	9.113205e+08	1.000000	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	...	36.040000	49.540

8 rows × 32 columns

train,test = train_test_split(df,test_size = 0.3)

trainX = train[df.columns[2:-1]]

trainY = train[df.columns[1]]

testX = test[df.columns[2:-1]]

testY = test[df.columns[1]]

gnb = GaussianNB()

clf = gnb.fit(trainX.values,trainY.values)

y_pred=clf.predict(testX.values)

print("Number of mislabelled points out of a total %d points : %d" %(testX.shape[0] , (testY.values != y_pred).sum()))

```
Number of mislabelled points out of a total 171 points : 9
```

testY

y_pred

```
testY

445    0
252    1
438    0
352    1
538    0
      ..
260    1
385    1
158    0
32     1
12     1
Name: diagnosis, Length: 171, dtype: int64
```

```
y_pred

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

print('Accuracy of the NB Classifier is : ',metrics.accuracy_score(y_pred,testY)*100)

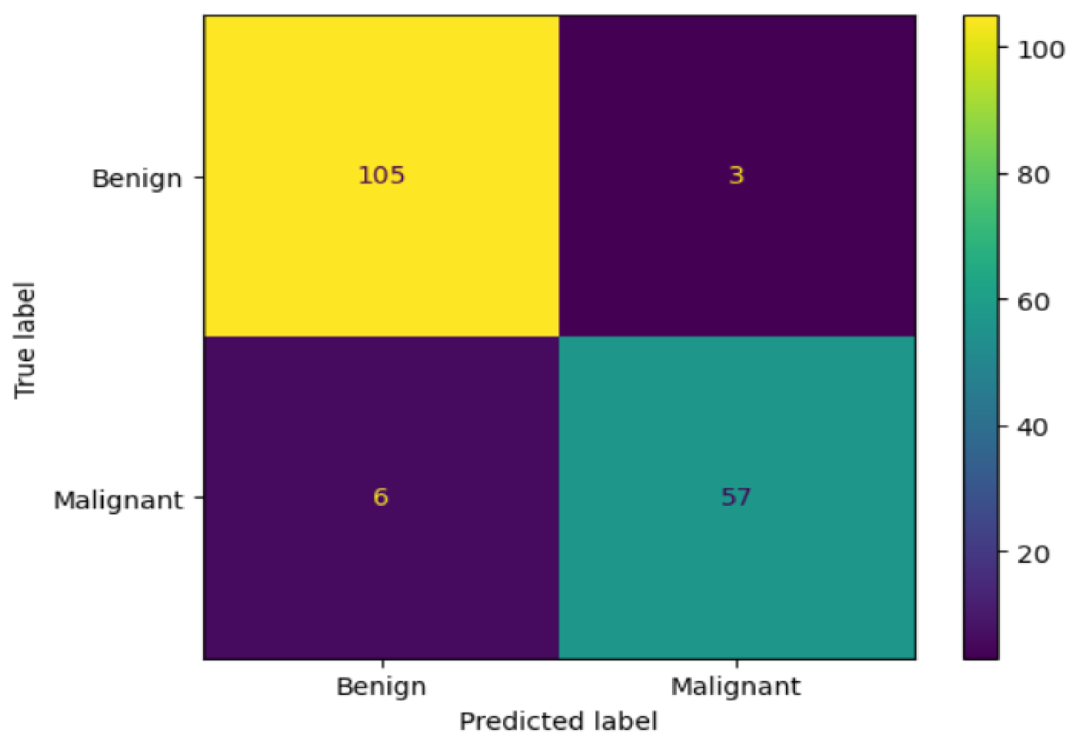
```
Accuracy of the NB Classifier is :  94.73684210526315
```

cm = confusion_matrix(testY.values,y_pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = cm,display_labels = ['Benign','Malignant'])

cm_display.plot()

plt.show()



```
sample =
[[11.94,18.24,75.71,437.6,0.08261,0.04751,0.01972,0.01349,0.1868,0.0611,0.2273,0.6329,1.52,
17.47,0.00721,0.00838,0.01311,0.008,0.01996,0.002635,13.1,21.33,83.67,527.2,0.1144,0.08906,
0.09203,0.06296,0.2785]]

pred = clf.predict(sample)

res = pred[0]

if res==0:

    print('Benign')

if res==1:

    print('Malignant')
```

Benign

PROGRAM NO: 16

AIM :

Write a program to implement decision trees using Iris dataset and find the accuracy of the algorithm.

CODE :

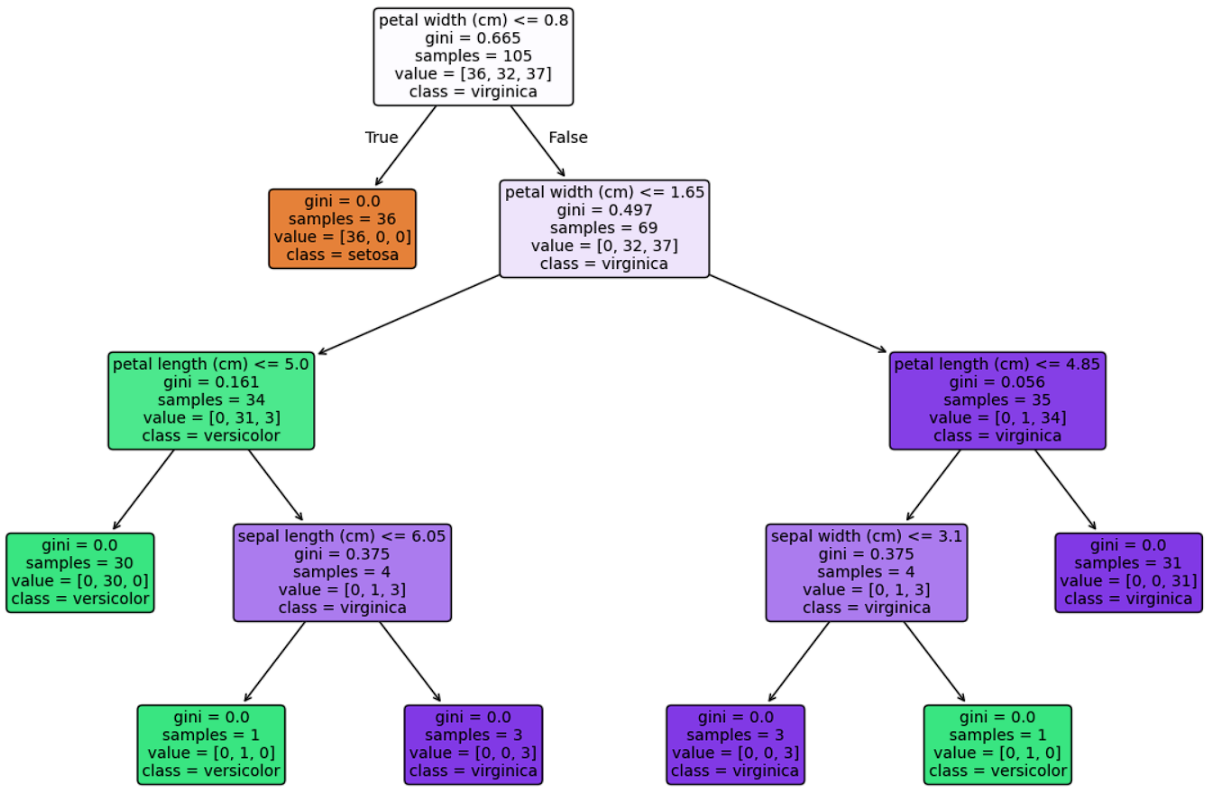
```
from sklearn.datasets import load_iris
from sklearn import metrics
from sklearn import tree
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

iris = load_iris()
x=iris.data
y = iris.target
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3,random_state = 1)

clf = DecisionTreeClassifier()
clf = clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
print("Accuracy : " , metrics.accuracy_score(y_test,y_pred))
```

```
Accuracy :  0.9555555555555556
```

plt.figure(figsize=(15,10))

tree.plot_tree(clf,fontsize = 10,filled = True, rounded = True,class_names=iris.target_names ,feature_names=iris.feature_names)

plt.show()



PROGRAM NO: 17

AIM :

Write a program to implement decision trees using Iris dataset and find the accuracy of the algorithm (using entropy).

CODE :

```
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

from sklearn import metrics
from sklearn.metrics import confusion_matrix
from mlxtend.plotting import plot_confusion_matrix

x,y = load_iris(return_X_y=True)
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=0)

dtree = DecisionTreeClassifier(criterion="entropy")

clf = dtree.fit(x_train,y_train)
y_pred = clf.predict(x_test)
print("Number of mislabelled points out of a total %d points : %d " %(x_test.shape[0],(y_test != y_pred).sum()))
```

```
Number of mislabelled points out of a total 45 points : 1
```

y_test

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

y_pred

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

print("Accuracy of the Decision tree Classifier is : ",metrics.accuracy_score(y_pred,y_test)*100
)

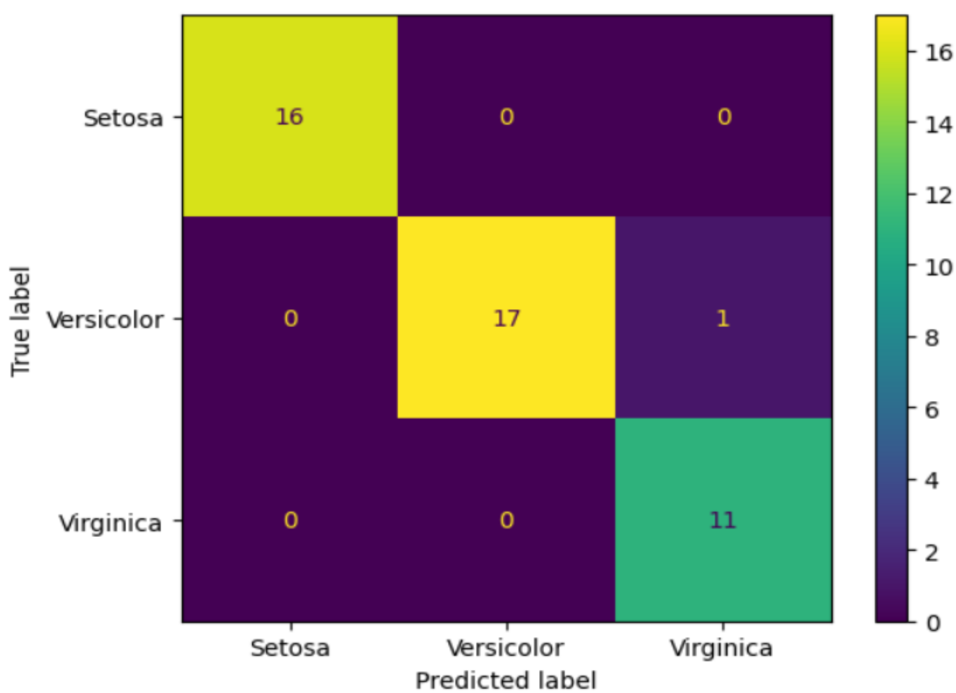
```
Accuracy of the Decision tree Classifier is :  97.77777777777777
```

cm = confusion_matrix(y_test,y_pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = cm, display_labels =
["Setosa","Versicolor","Virginica"])

cm_display.plot()

plt.show()



```
res = clf.predict([[5.1,3.5,1.4,0.2]])
if res==0:
    print("Setosa")
if res==1:
    print("Versicolor")
if res==2:
    print("Virginica")
```

```
Setosa
```

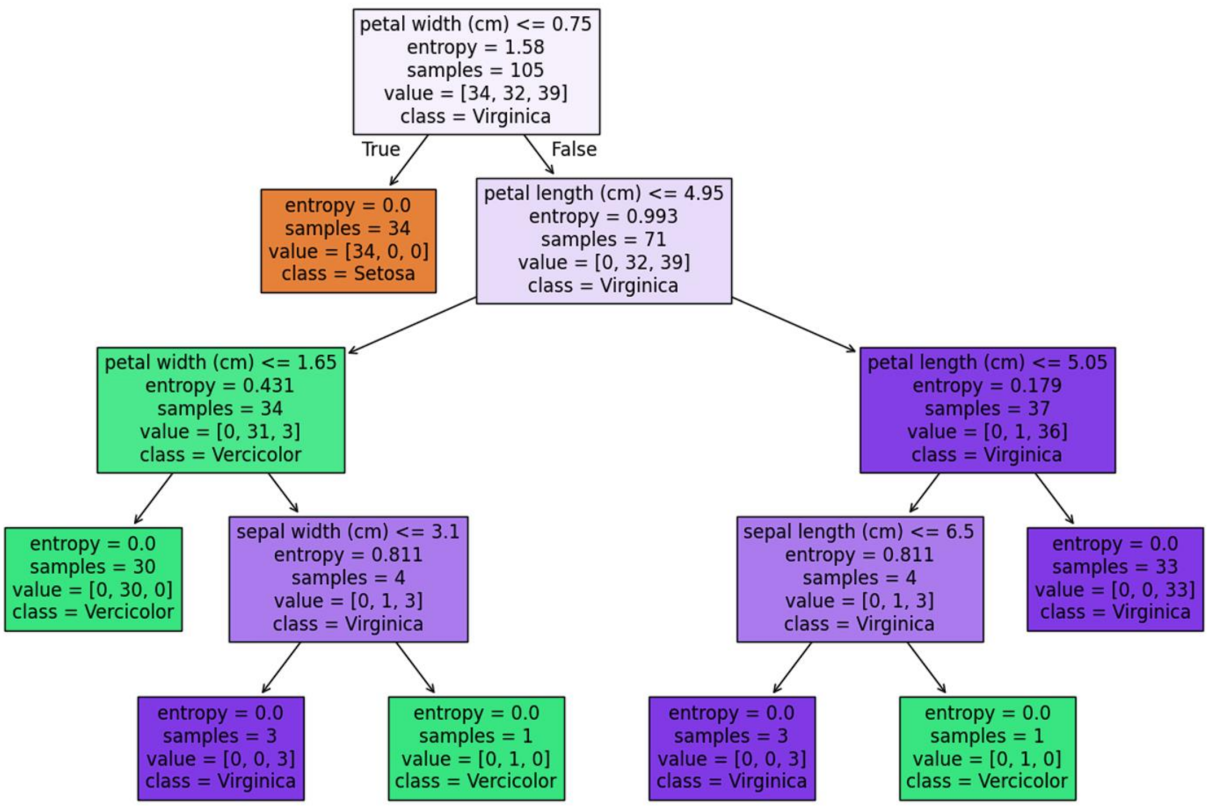
```
res = clf.predict([[5.2,3.2,6.3,7.6]])
if res==0:
    print("Setosa")
if res==1:
    print("Versicolor")
if res==2:
    print("Virginica")
```

```
Virginica
```

```
res = clf.predict([[3.4,5.5,2.3,4.2]])
if res==0:
    print("Setosa")
if res==1:
    print("Versicolor")
if res==2:
    print("Virginica")
```

```
Versicolor
```

```
from sklearn import tree
fig = plt.figure(figsize=(20,20))
_ =
tree.plot_tree(clf,feature_names=load_iris().feature_names,class_names=['Setosa','Vercicolor','Vi
rginica'],filled=True)
```



PROGRAM NO: 18

AIM :

Write a program to implement linear and multiple regression techniques using any standard dataset available in the public domain and evaluate its performance.

CODE :

```
import matplotlib.pyplot as plt

import numpy as np

from sklearn import datasets, linear_model

from sklearn.metrics import mean_squared_error, r2_score


df = datasets.load_diabetes()

df['feature_names']
```

```
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

```
df
```

```
{'data': array([[ 0.03807591,  0.05068012,  0.06169621, ..., -0.00259226,
          0.01990749, -0.01764613],
        [-0.00188202, -0.04464164, -0.05147406, ..., -0.03949338,
         -0.06833155, -0.09220405],
        [ 0.08529891,  0.05068012,  0.04445121, ..., -0.00259226,
          0.00286131, -0.02593034],
        ...,
        [ 0.04170844,  0.05068012, -0.01590626, ..., -0.01107952,
         -0.04688253,  0.01549073],
        [-0.04547248, -0.04464164,  0.03906215, ...,  0.02655962,
          0.04452873, -0.02593034],
        [-0.04547248, -0.04464164, -0.0730303 , ..., -0.03949338,
         -0.00422151,  0.00306441]]),
 'target': array([151.,  75., 141., 206., 135.,  97., 138.,  63., 110., 310., 101.,
         69., 179., 185., 118., 171., 166., 144.,  97., 168.,  68.,  49.,
         68., 245., 184., 202., 137.,  85., 131., 283., 129.,  59., 341.,
         87.,  65., 102., 265., 276., 252.,  90., 100.,  55.,  61.,  92.,
        259.,  53., 190., 142.,  75., 142., 155., 225.,  59., 104., 182.,
        128.,  52.,  37., 170., 170.,  61., 144.,  52., 128.,  71., 163.,
        150.,  97., 160., 178.,  48., 270., 202., 111.,  85.,  42., 170.,
        200., 252., 113., 143.,  51.,  52., 210.,  65., 141.,  55., 134.,
         42., 111.,  98., 164.,  48.,  96.,  90., 162., 150., 279.,  92.,
         83., 128., 102., 302., 198.,  95.,  53., 134., 144., 232.,  81.,
        104.,  59., 246., 297., 258., 229., 275., 281., 179., 200., 200.,
        173., 180.,  84., 121., 161.,  99., 109., 115., 268., 274., 158.,
        107.,  83., 103., 272.,  85., 280., 336., 281., 118., 317., 235.,
         60., 174., 259., 178., 128.,  96., 126., 288.,  88., 292.,  71.,
        197., 186.,  25.,  84.,  96., 195.,  53., 217., 172., 131., 214.,
         59.,  70., 220., 268., 152.,  47.,  74., 295., 101., 151., 127.,
        237., 225.,  81., 151., 107.,  64., 138., 185., 265., 101., 137.,
        143., 141.,  79., 292., 178.,  91., 116.,  86., 122.,  72., 129.,
        142.,  90., 158.,  39., 196., 222., 277.,  99., 196., 202., 155.,
         77., 191.,  70.,  73.,  49.,  65., 263., 248., 296., 214., 185.,
         78.,  93., 252., 150.,  77., 208.,  77., 108., 160.,  53., 220.,
        154., 259.,  90., 246., 124.,  67.,  72., 257., 262., 275., 177.,
```

Linear regression

diabetes_X,diabetes_y = datasets.load_diabetes(return_X_y = True)

diabetes_X.shape

```
(442, 10)
```

diabetes_y.shape

```
(442,)
```

diabetes_X = diabetes_X[:, np.newaxis, 2]

diabetes_X.shape

```
(442, 1)
```

diabetes_X_train = diabetes_X[:-20]

diabetes_X_test = diabetes_X[-20:]

diabetes_y_train = diabetes_y[:-20]

diabetes_y_test = diabetes_y[-20:]

regr = linear_model.LinearRegression()

regr.fit(diabetes_X_train,diabetes_y_train)

```
▾ LinearRegression
LinearRegression()
```

diabetes_y_pred = regr.predict(diabetes_X_test)

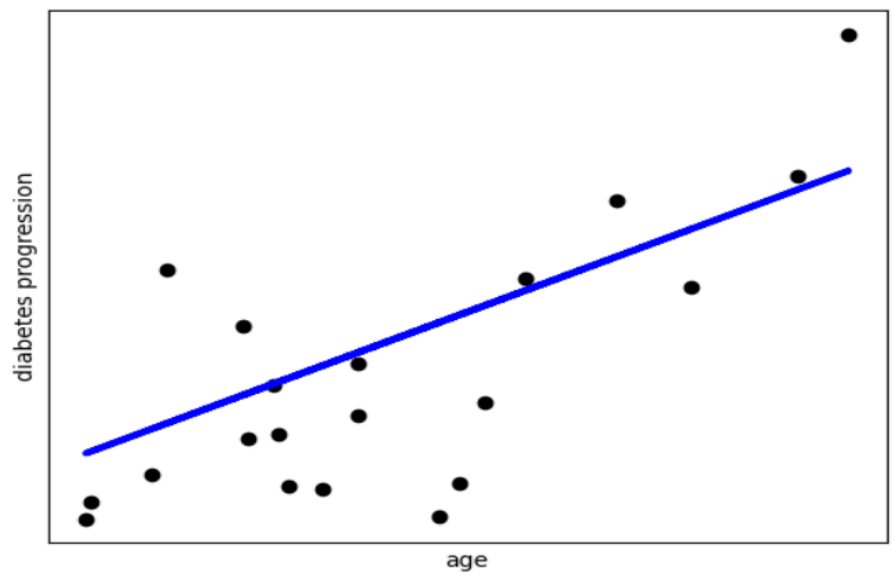
print("Coefficients: \n",regr.coef_)

print("Mean Squared error: %.2f" % mean_squared_error(diabetes_y_test,diabetes_y_pred))

print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test,diabetes_y_pred))

```
Coefficients:
 [938.23786125]
Mean Squared error: 2548.07
Coefficient of determination: 0.47
```

```
plt.scatter(diabetes_X_test,diabetes_y_test,color="black")
plt.plot(diabetes_X_test,diabetes_y_pred,color="blue", linewidth=3)
plt.xlabel("age")
plt.ylabel("diabetes progression")
plt.xticks(())
plt.yticks(())
plt.show()
```



Multiple Regression

```
diabetes_X,diabetes_y = datasets.load_diabetes(return_X_y = True)
diabetes_X.shape
diabetes_X = diabetes_X[:, [0,2]]
diabetes_X.shape
```

```
(442, 2)
```

```
diabetes_X
```

```
array([[ 0.03807591,  0.06169621],
       [-0.00188202, -0.05147406],
       [ 0.08529891,  0.04445121],
       [-0.08906294, -0.01159501],
       [ 0.00538306, -0.03638469],
       [-0.09269548, -0.04069594],
       [-0.04547248, -0.04716281],
       [ 0.06350368, -0.00189471],
       [ 0.04170844,  0.06169621],
       [-0.07090025,  0.03906215],
       [-0.09632802, -0.08380842],
       [ 0.02717829,  0.01750591],
       [ 0.01628068, -0.02884001],
       [ 0.00538306, -0.00189471],
       [ 0.04534098, -0.02560657],
       [-0.05273755, -0.01806189],
       [-0.00551455,  0.04229559],
       [ 0.07076875,  0.01211685],
       [-0.0382074 , -0.0105172 ],
       [-0.02730979, -0.01806189],
       [-0.04910502, -0.05686312],
       [-0.0854304 , -0.02237314],
       [-0.0854304 , -0.00405033],
       [ 0.04534098,  0.06061839],
       [-0.06363517,  0.03582872],
       [-0.06726771, -0.01267283],
       [-0.10722563, -0.07734155],
       [-0.02367725,  0.05954058],
       [ 0.05260606, -0.02129532],
       [ 0.06713621, -0.00620595],
       [-0.06000263,  0.04445121],
       [-0.02367725, -0.06548562],
       [ 0.03444337,  0.12528712],
       [ 0.03081083, -0.05039625],
       [ 0.01628068, -0.06332999],
       [ 0.04897352, -0.03099563],
```

diabetes_X_train = diabetes_X[:-20]

diabetes_X_test = diabetes_X[-20:]

diabetes_y_train = diabetes_y[:-20]

diabetes_y_test = diabetes_y[-20:]

regr = linear_model.LinearRegression()

regr.fit(diabetes_X_train,diabetes_y_train)

```
▾ LinearRegression
LinearRegression()
```

diabetes_y_pred = regr.predict(diabetes_X_test)

print("Coefficients: \n",regr.coef_)

print("Intercept: \n",regr.intercept_)

print("Mean Squared error: %.2f" % mean_squared_error(diabetes_y_test,diabetes_y_pred))

print("Coefficient of determination: %.2f" % r2_score(diabetes_y_test,diabetes_y_pred))

```
Coefficients:
 [139.20420118 912.45355549]
Intercept:
 152.87670001405584
Mean Squared error: 2596.60
Coefficient of determination: 0.46
```

diabetes_X_test.shape

```
(20, 2)
```

x = diabetes_X_test[:,0]

y = diabetes_X_test[:,1]

#z = diabetes_X_test[:,2]

diabetes_X_test

```
array([[-0.07816532,  0.07786339],
       [ 0.0090156 , -0.03961813],
       [ 0.00175052,  0.01103904],
       [-0.07816532, -0.04069594],
       [ 0.03081083, -0.03422907],
       [-0.03457486,  0.00564998],
       [ 0.04897352,  0.08864151],
       [-0.04183994, -0.03315126],
       [-0.00914709, -0.05686312],
       [ 0.07076875, -0.03099563],
       [ 0.0090156 ,  0.05522933],
       [-0.02730979, -0.06009656],
       [ 0.01628068,  0.00133873],
       [-0.01277963, -0.02345095],
       [-0.05637009, -0.07410811],
       [ 0.04170844,  0.01966154],
       [-0.00551455, -0.01590626],
       [ 0.04170844, -0.01590626],
       [-0.04547248,  0.03906215],
       [-0.04547248, -0.0730303 ]])
```

plt.style.use('default')

fig = plt.figure(figsize=(12,4))

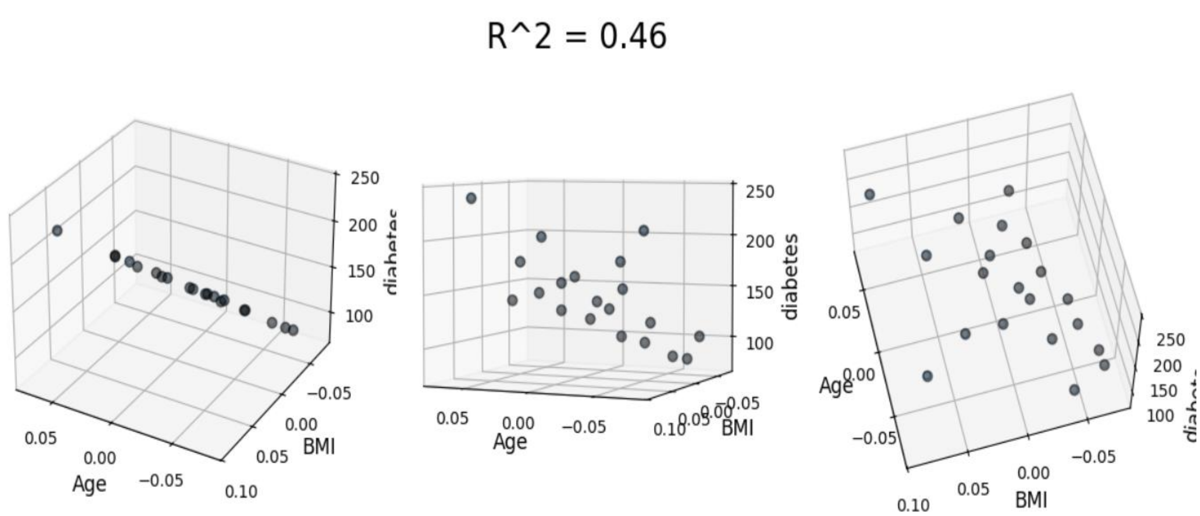
ax1 = fig.add_subplot(131,projection = '3d')

ax2 = fig.add_subplot(132,projection = '3d')

ax3 = fig.add_subplot(133,projection = '3d')

axes = [ax1,ax2,ax3]

```
for ax in axes:
    ax.plot(x,y, diabetes_y_pred,color = 'k', zorder = 15, linestyle = 'none',marker = 'o',alpha = 0.5)
    ax.scatter(x.flatten(),y.flatten(),diabetes_y_pred,facecolor=(0,0,0,0), s=20,edgecolor = 
'#70b3f0')
    ax.set_xlabel('Age',fontsize = 12)
    ax.set_ylabel('BMI',fontsize = 12)
    ax.set_zlabel('diabetes',fontsize = 12)
    ax.locator_params(nbins=4,axis='x')
    ax.locator_params(nbins=5,axis='x')
    ax1.view_init(elev=28,azim=120)
    ax2.view_init(elev=4,azim=114)
    ax3.view_init(elev=60,azim=165)
    fig.suptitle('R^2 = %.2f' % r2_score(diabetes_y_test,diabetes_y_pred),fontsize=20)
    fig.tight_layout()
```



PROGRAM NO: 19

AIM :

Write a program to implement Iris flower classification using Support vector machine.

CODE :

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.svm import SVC

iris = load_iris()
x = iris.data
y = iris.target
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.3,random_state=1)

classifier = SVC(kernel='linear',random_state=0)
classifier.fit(x_train,y_train)
y_pred = classifier.predict(x_test)

print("Acccuracy : ",metrics.accuracy_score(y_test,y_pred))
```

```
Acccuracy :  1.0
```

```
sample = [[1,1,1,2]]
pred = classifier.predict(sample)
pred_v = [iris.target_names[p] for p in pred]
print(pred_v)
```

```
[np.str_('setosa')]
```

PROGRAM NO: 20

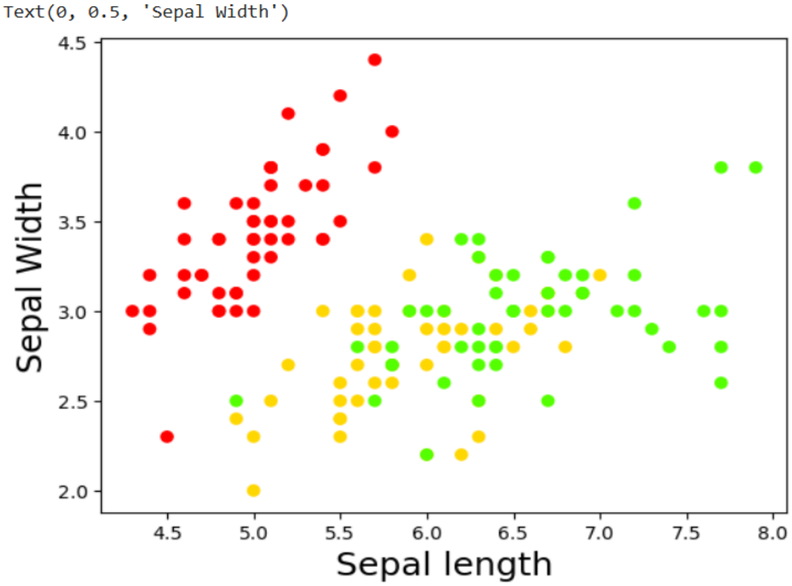
AIM :

Write a program to implement k-means clustering technique using any standard dataset available in the public domain.

CODE :

```
from sklearn import datasets
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans

iris = datasets.load_iris()
X = iris.data[:, :2]
y = iris.target
plt.scatter(X[:,0],X[:,1],c=y,cmap='prism')
plt.xlabel('Sepal length',fontsize=18)
plt.ylabel('Sepal Width',fontsize=18)
```

Text(0, 0.5, 'Sepal Width')



km = KMeans(n_clusters=3,init='k-means++',n_init=10,max_iter=300,tol=0.0001, verbose=0, random_state=21, copy_x=True, algorithm="elkan")

km.fit(X)

centers = km.cluster_centers_

print(centers)

new_labels = km.labels_

print(new_labels)

print(y)

```
[[6.81276596 3.07446809]
 [5.77358491 2.69245283]
 [5.006      3.428     ]]
[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 2 2 0 0 0 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
 0 0 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0
 0 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 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 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 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 2]
```

fig, axes = plt.subplots(1,2,figsize=(16,8))

axes[0].scatter(X[:,0],X[:,1],c=y,cmap='prism',edgecolor='k',s=75)

axes[1].scatter(X[:,0],X[:,1],c=new_labels,cmap='jet',edgecolor='k',s=75)

axes[0].set_xlabel('Sepal length',fontsize=12)

axes[0].set_ylabel('Sepal Width',fontsize=12)

axes[1].set_xlabel('Sepal length',fontsize=12)

axes[1].set_ylabel('Sepal Width',fontsize=12)

axes[0].tick_params(direction='in',length=10,width=5,colors='k',labelsize=15)

axes[1].tick_params(direction='in',length=10,width=5,colors='k',labelsize=15)

axes[0].set_title('Actual',fontsize=18)

axes[1].set_title('Predicted',fontsize=18)

```
Text(0.5, 1.0, 'Predicted')
```