EXPERIMENT – 1

AIM:

Compute Central Tendency Measures and Measure of Dispersion

```
import statistics as st
from math import isnan
from itertools import filterfalse

data = [20.7,float('NaN'), 19.2, 18.3, float('NaN'), 14.4]
print(data)
clean = list(filterfalse(isnan, data))
print("Cleaned data:", clean)
print('median =', st.median(clean))
print('median low =', st.median_low(clean))
print('median high =', st.median_high(clean))
print(f'{st.mode([1, 1, 2, 3, 3, 3, 4])}')
print(f'{st.mode(["red", "blue", "blue", "red", "green", "red", "red"])}')
print(f'{st.multimode('aabbbbbccddddeeeffffgg')}')

data1 = [6,7,10,13,14,14,18,19,22,24]
print("variance =", st.variance(data))
mean = st.mean(data1)
print('variance=',st.variance(data1,mean))
X = [3,5,6,6,7,8,12,14,15,19]
Y = [6,7,7,13,16,15,17,20,24,27]
print('covariance = ',st.covariance(X,Y))
print('correlation = ',st.correlation(X,Y))
```

```
[20.7, nan, 19.2, 18.3, nan, 14.4]
Cleaned data: [20.7, 19.2, 18.3, 14.4]
median = 18.75
median low = 18.3
median high = 19.2
3
red
['b', 'd']
variance = nan
variance= 36.67777777777778
covariance =  35.333333333333336
correlation =  0.9443522018799386
```

EXPERIMENT – 2

AIM:

2A) Study of Python Basic Libraries: Statistics, Math, NumPy.

```
import numpy as np
arr = np.array([11,42,35])
print("Array with Rank 1: \n",arr)
arr2 = np.array([[1,2,3],[4,5,6],[7,8,9]])
print("Array with Rank 2: \n",arr2)
arr3 = np.array((1,6,4))
print("Array created using passed tuple:\n",arr3)
```

```
Array with Rank 1:
 [11 42 35]
Array with Rank 2:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Array created using passed tuple:
 [1 6 4]
```

```
arr = np.array([[-2, 3, -4, 1],
 [ 4, -1, 2, 0],
 [ 3, -3, 1, -2],
 [-1, 5, 0, 4]]
)
print("Initial array:\n",arr)
sliced_arr = arr[:2,::2]
print("Array with first 2 rows and alternate columns(0 and 2):\n",sliced_arr)
Index_arr = arr[[1, 2, 0, 3],[2,1,0,1]]
print("Elements at indices(1,2),(2,1),(0,0),(3,1):\n",Index_arr)
```

```
Initial array:
 [[-2  3 -4  1]
 [ 4 -1  2  0]
 [ 3 -3  1 -2]
 [-1  5  0  4]]
Array with first 2 rows and alternate columns(0 and 2):
 [[-2 -4]
 [ 4  2]]
Elements at indices(1,2),(2,1),(0,0),(3,1):
 [ 2 -3 -2  5]
```

```
a = np.array([[1,2],[3,4]])
b = np.array([[5,7],[9,8]])
print("Adding 1 to every element:\n",a+1)
print("Subtracting 2 from each element:\n",b-2)
```

```
Adding 1 to every element:
 [[2 3]
 [4 5]]
Subtracting 2 from each element:
 [[3 5]
 [7 6]]
```

```
s = "MACHINE LEARNING"
print(s)
print(s[6])
print(s[-1])
print(s[-3])
print(s[1:4])
print(s[-5:-1])
print(s[-2:-7:-2])
```

```
MACHINE LEARNING
E
G
I
ACH
RNIN
NNA
```

```
x = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
a1 = x.flatten()
a1[2] = 25
print(x)
print(a1)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
[ 1  2 25  4  5  6  7  8  9 10 11 12]
```

```
a2 = x.ravel()
a2[0] = 55
print(x)
print(a2)
```

```
[[55  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
[55  2  3  4  5  6  7  8  9 10 11 12]
```

```
print(np.zeros((2,4)))
print(np.ones((3,4,2)))
print(np.empty((3,4)))
np.arange(10,30,5)
```

```
[[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.]]]
[[9.62185643e-312 3.16202013e-322 0.00000000e+000 0.00000000e+000]
 [1.18831764e-312 3.22003211e-057 1.39471268e+165 1.68882068e-051]
 [1.42112079e+161 9.51921895e+169 3.76061223e+179 1.40487214e+165]]
```

```
a = np.array([20,30,40,50])
b = np.arange(4)
print('a =',a)
print('b =',b)
c = a - b
print('c =',c)
print('b**2 =',b**2)
print("a<35 =",a<35)
```

```
a = [20 30 40 50]
b = [0 1 2 3]
c = [20 29 38 47]
b**2 = [0 1 4 9]
a<35 = [ True  True False False]
```

```
A = np.array([[1,1],[0,1]])
B = np.array([[2,0],[3,4]])
print(A*B)
print(A@B)
```

```
[[2 0]
 [0 4]]
[[5 4]
 [3 4]]
```

2B)

AIM: SciPy Implementation

```
from scipy import special as sp
from scipy import linalg
import numpy as np
matrix = np.array([[11, 14, 31],
[23, 15, 25],
[16, 34, 27]])
print("Matrix:\n", matrix)
det = linalg.det(matrix)
print("Determinant =", det)
a = sp.exp10(2)
b = sp.exp10(3)
c = sp.sindg(90)
print("a =", a)
print("b =", b)
print("c =", c)
inv = linalg.inv(matrix)
print("Inverse of matrix:\n", inv)
```

```
Matrix:
 [[11 14 31]
 [23 15 25]
 [16 34 27]]
Determinant = 8813.0
a = 100.0
b = 1000.0
c = 1.0
Inverse of matrix:
 [[-0.05049359  0.07670487 -0.01304891]
 [-0.02507659 -0.02258028  0.04969931]
 [ 0.06150006 -0.01702031 -0.01781459]]
```

```
from scipy import linalg
import numpy as np
arr = np.array([[9, 2],[3, 7]])
eg_val, eg_vect = linalg.eig(arr)
print("Eigenvalues:", eg_val)
print("Eigenvectors:\n", eg_vect)
```

```
Eigenvalues: [10.64575131+0.j  5.35424869+0.j]
Eigenvectors:
 [[ 0.7721779  -0.48096517]
 [ 0.6354064   0.8767397 ]]
```

```python
from scipy import integrate
f = lambda x: x**2
# Integrate from 0 to 1
result = integrate.quad(f, 0, 1)
print("Integration of x^2 from 0 to 1 =", result)
from math import sqrt
f = lambda x, y: 64 * x * y
p = lambda x: 0
q = lambda y: sqrt(1 - 2 * y**2)
integration = integrate.dblquad(f, 0, 2/4, p, q)
print("Double integration result =", integration)
```

```
Integration of x^2 from 0 to 1 = (0.33333333333333337, 3.700743415417189e-15)
Double integration result = (3.0, 9.657432734515774e-14)
```

EXPERIMENT – 3

AIM:

3A) Study of Python Libraries for ML Applications: Pandas

```
import pandas as pd
s = pd.Series(["Jpn","Eng","Frc","Spn"])
type(s)
```

```
pandas.core.series.Series
```

```
s = pd.Series(["Jpn","Eng","Frc","Spn"],index = ["a","b","c","d"])
s
```

```
a    Jpn
b    Eng
c    Frc
d    Spn
dtype: object
```

```
EmployeeData = {
    'ID' : [101,102,103],
    'name' : ['Daniel','Butler','Morgan'],
    'age' : [34,32,29],
    'city' : ['Goa','Agra','Delhi']
}
empDB = pd.DataFrame(EmployeeData)
empDB
```

	ID	name	age	city
0	101	Daniel	34	Goa
1	102	Butler	32	Agra
2	103	Morgan	29	Delhi

```
empDB = pd.DataFrame(EmployeeData,index = ['a','b','c'])
print(empDB)
```

```
    ID    name  age   city
a  101  Daniel   34    Goa
b  102  Butler   32   Agra
c  103  Morgan   29  Delhi
```

```
ColumnData = ['ID','Emp','Salary','Exp']
df = pd.DataFrame(columns = ColumnData)
df
```

ID	Emp	Salary	Exp

```
df1 = pd.DataFrame()
df1['id'] = [1,2,3]
df1['emp'] = ['A','B','C']
df1['salary'] = [1000,2000,3000]
df1['exp'] = [2,3,4]
df1
```

	id	emp	salary	exp
0	1	A	1000	2
1	2	B	2000	3
2	3	C	3000	4

```
df3 = pd.DataFrame(columns = ['emp','salary','grade'],index = ['a','b','c'])
df3
```

	emp	salary	grade
a	NaN	NaN	NaN
b	NaN	NaN	NaN
c	NaN	NaN	NaN

```
df3['emp'] = ['Michael','Lorenzo','Alex']
df3
```

	emp	salary	grade
a	Michael	NaN	NaN
b	Lorenzo	NaN	NaN
c	Alex	NaN	NaN

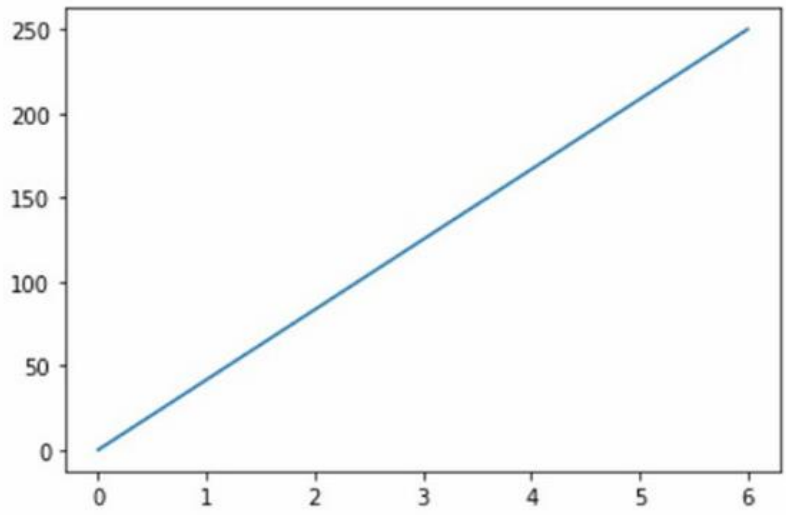
```
df = pd.read_csv("Iris.csv")
df.head(4)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa

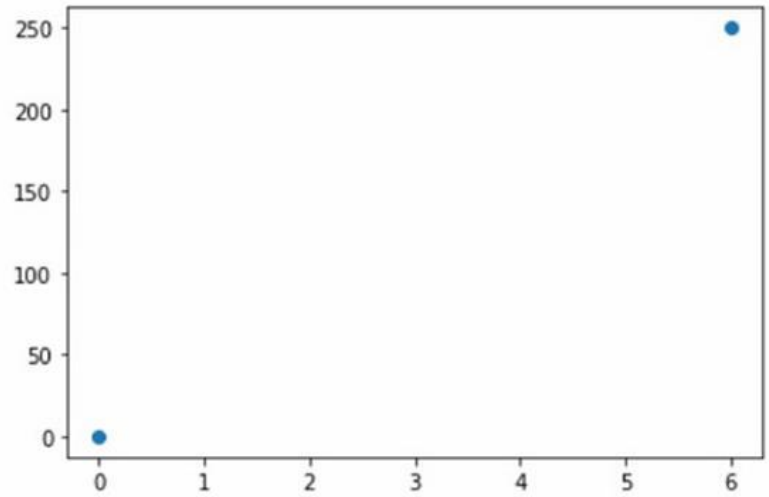
3B)

AIM: Implementation of Matplotlib

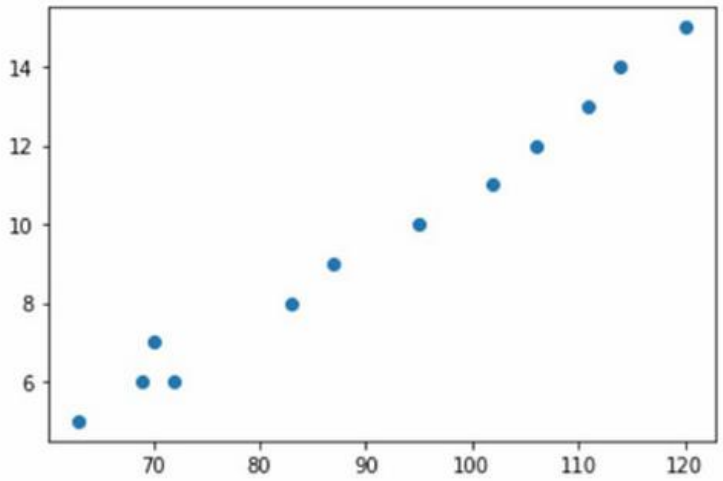
```
xpoints = np.array([0,6])
ypoints = np.array([0,250])
plt.plot(xpoints,ypoints)
plt.show()
```



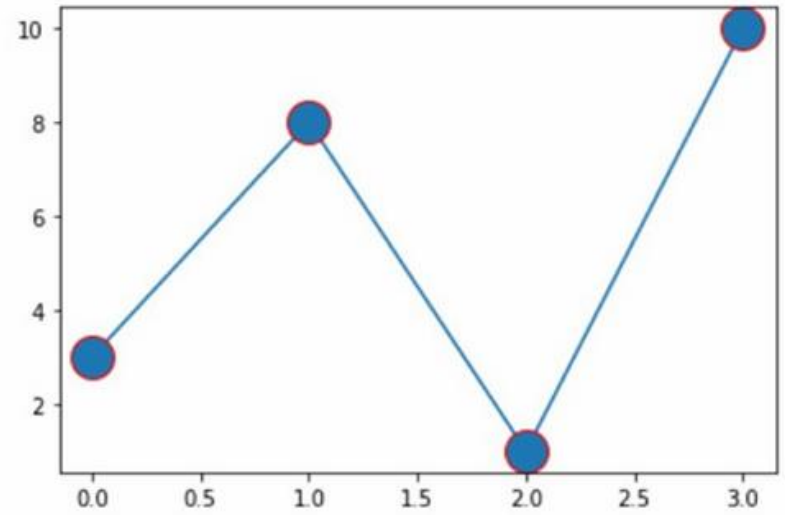

```
xpoints = np.array([0,6])
ypoints = np.array([0,250])
plt.plot(xpoints,ypoints,'o')
plt.show()
```



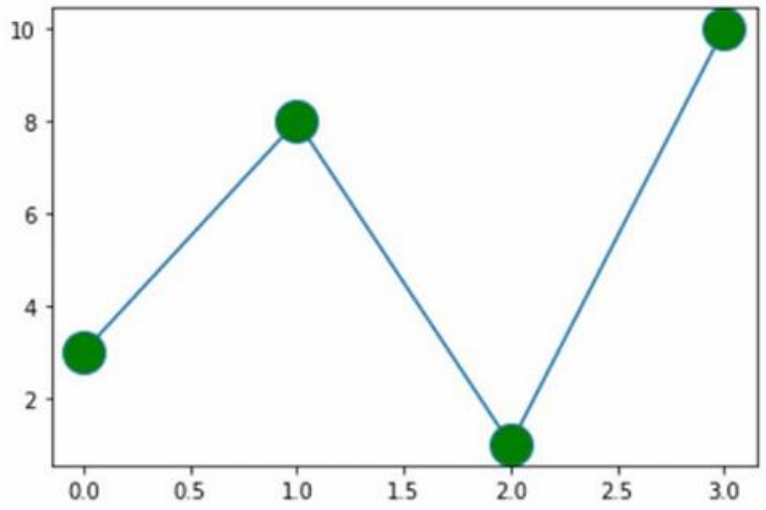
```
xpoints = np.array([63, 69, 72, 70, 83, 87, 95, 102, 106, 111, 114, 120])
ypoints = np.array([5, 6, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])
plt.plot(xpoints,ypoints,'o')
plt.show()
```



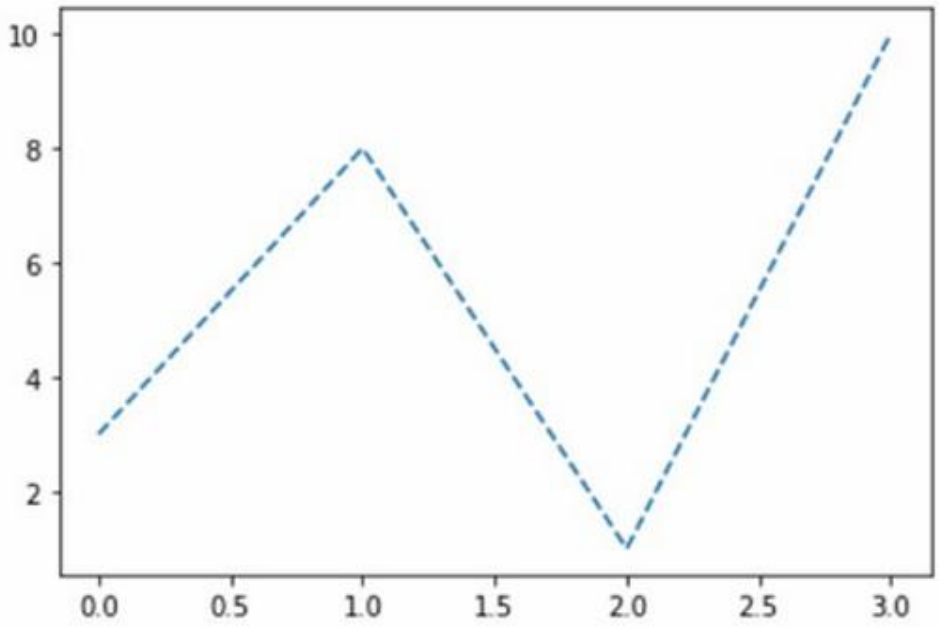

```
plt.plot(ypoints,marker = 'o',ms = 20,mec = 'r')
plt.show()
```



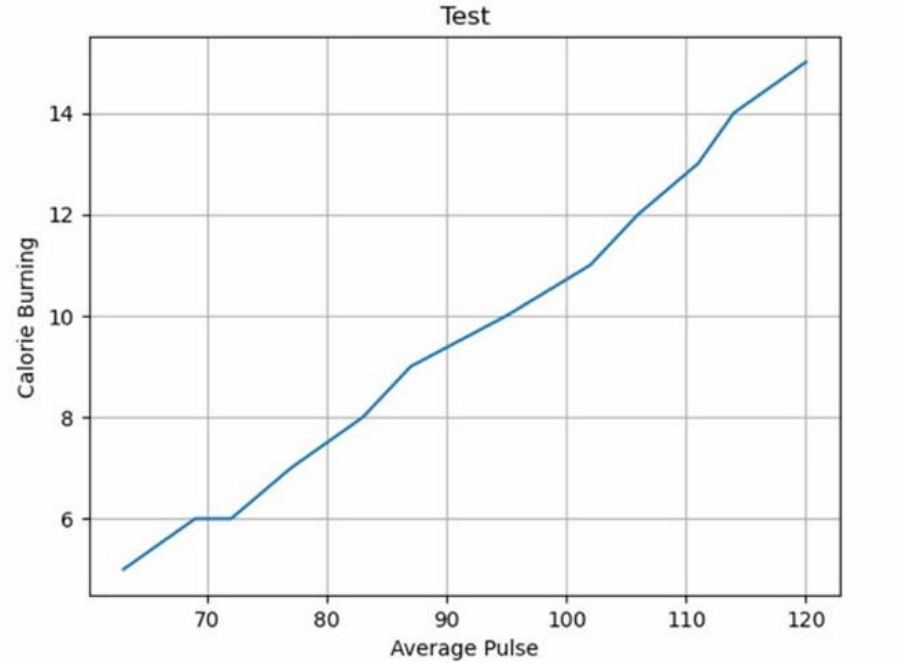
```
plt.plot(ypoints,marker = 'o',ms = 20,mfc = 'g')
plt.show()
```



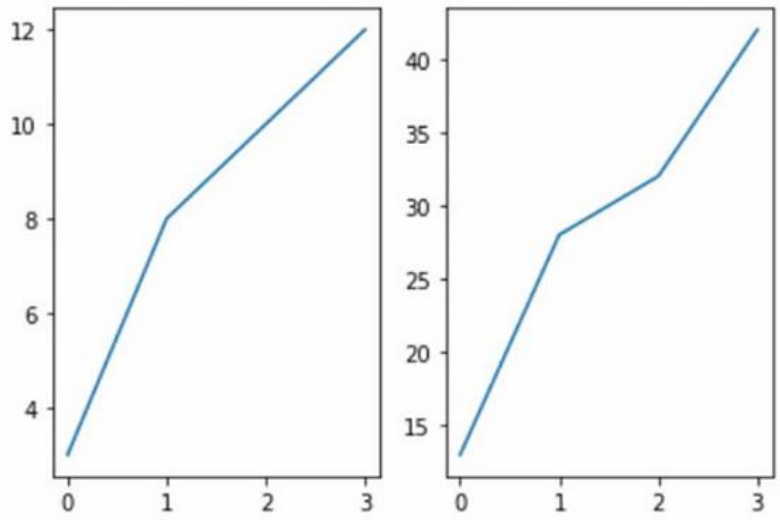

```
plt.plot(ypoints,linestyle = 'dashed')
plt.show()
```



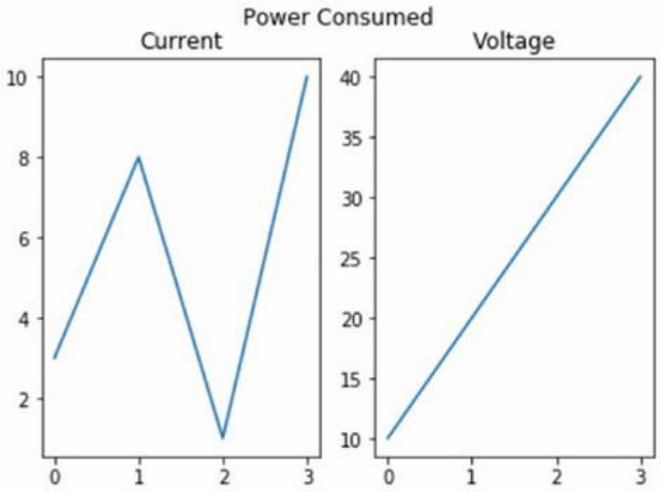
[27]:
```
y = np.array([63, 69, 72, 77, 83, 87, 95, 102, 106, 111, 114, 120])
x = np.array([5, 6, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15])
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burning")
plt.title("Test")
plt.plot(y,x)
plt.grid()
plt.show()
```



```
x = np.array([0,1,2,3])
y = np.array([3,8,10,12])
plt.subplot(1,2,1)
plt.plot(x,y)
x = np.array([0,1,2,3])
y = np.array([13,28,32,42])
plt.subplot(1,2,2)
plt.plot(x,y)
plt.show()
```

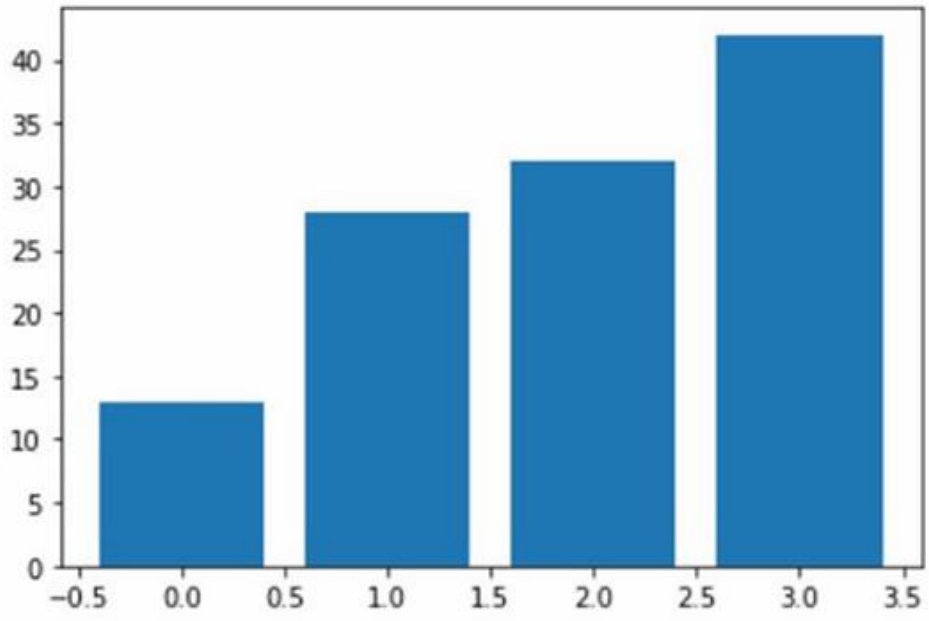



```
x1 = np.array([0, 1, 2, 3])
y1 = np.array([3, 8, 1, 10])
plt.subplot(1, 2, 1)
plt.plot(x1, y1)
plt.title("Current")
x2 = np.array([0, 1, 2, 3])
y2 = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)
plt.plot(x2, y2)
plt.title("Voltage")
plt.suptitle("Power Consumed")
plt.show()
```



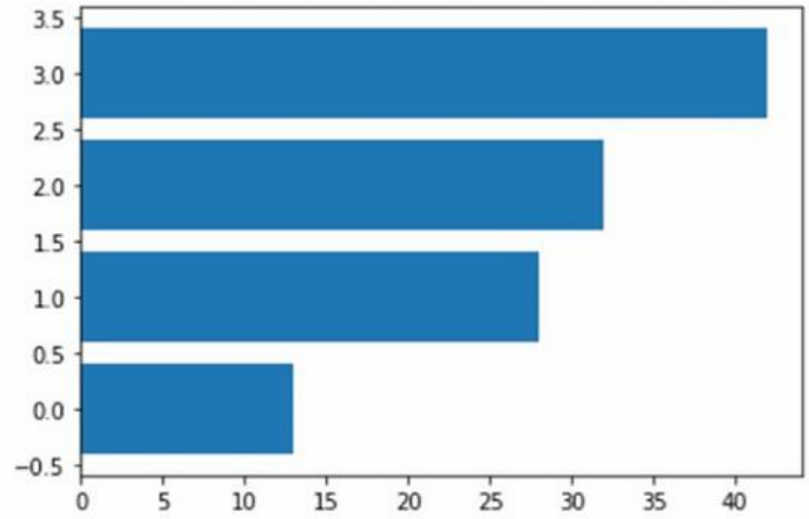
```
plt.bar(x,y)
```

```
<BarContainer object of 4 artists>
```



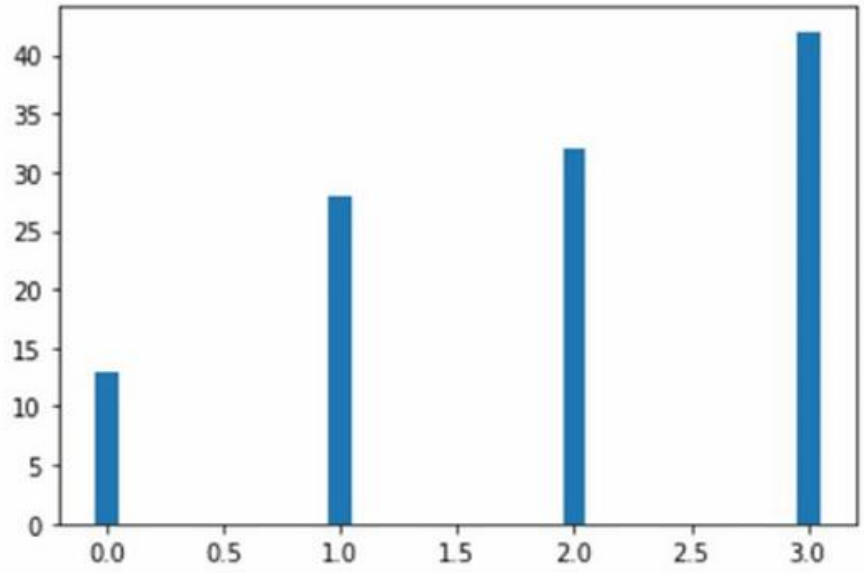

```
plt.barh(x,y)
```

```
<BarContainer object of 4 artists>
```



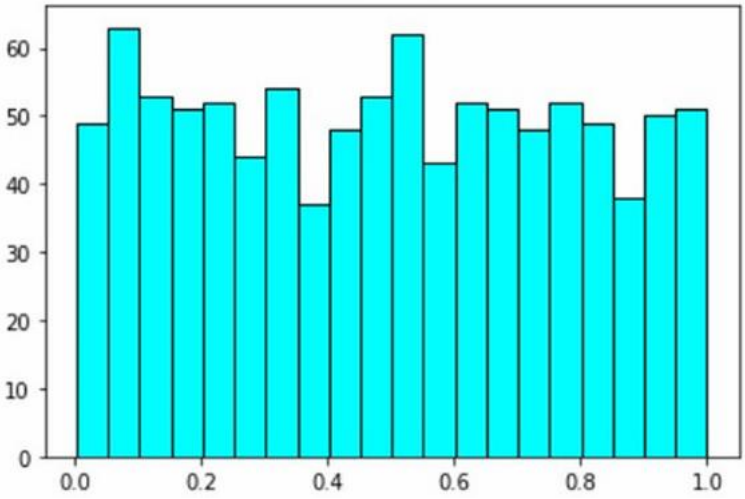
```
plt.bar(x,y,width = 0.1)
```

```
<BarContainer object of 4 artists>
```




```
data = np.random.random(1000)
plt.hist(data,bins = 20,color = "cyan", edgecolor = "black")
```

```
(array([49., 63., 53., 51., 52., 44., 54., 37., 48., 53., 62., 43., 52.,
        51., 48., 52., 49., 38., 50., 51.]),
 array([0.00218999, 0.05207762, 0.10196525, 0.15185287, 0.2017405 ,
        0.25162813, 0.30151576, 0.35140339, 0.40129102, 0.45117865,
        0.50106628, 0.55095391, 0.60084154, 0.65072917, 0.7006168 ,
        0.75050443, 0.80039206, 0.85027969, 0.90016732, 0.95005495,
        0.99994257]),
 <a list of 20 Patch objects>)
```



```
plt.pie(y)
```

```
([<matplotlib.patches.Wedge at 0x1d92eb78248>,
  <matplotlib.patches.Wedge at 0x1d92eb78b88>,
  <matplotlib.patches.Wedge at 0x1d92eb7c048>,
  <matplotlib.patches.Wedge at 0x1d92eb7cc88>],
 [Text(1.0313589124935578, 0.382490252973989, ''),
  Text(0.10501489634580775, 1.094975740162986, ''),
  Text(-1.0995895670519726, 0.030046364679530268, ''),
  Text(0.451980867449008, -1.0028525791261855, '')])
```

Experiment:4

AIM: Write a python program to implement Simple Linear Regression

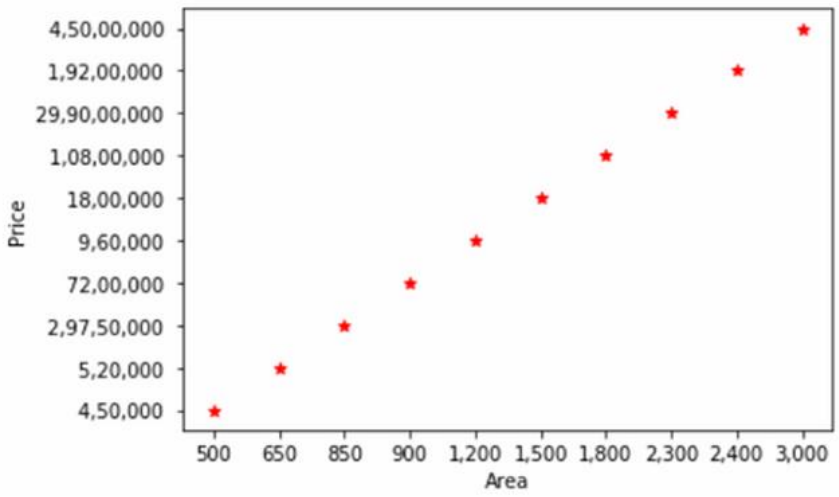
```
import pandas as pd
df = pd.read_csv('areaprice.csv')
```

```
df
```

	Area	Price
0	500	450000
1	650	520000
2	850	29750000
3	900	7200000
4	1200	9600000
5	1500	18000000
6	1800	10800000
7	2300	299000000
8	2400	19200000
9	3000	45000000

```
plt.xlabel('Area')
plt.ylabel('Price')
plt.scatter(df['Area'],df['Price'],color = 'r',marker="*")
```

```
<matplotlib.collections.PathCollection at 0x1d932447cc8>
```



```
from sklearn import linear_model
reg = linear_model.LinearRegression()
reg.fit(df[['Area']], df['Price'])
```

```
LinearRegression
LinearRegression()
```

```
print(reg.coef_)
```

```
[46716.11842105]
```

```
print(reg.predict(np.array([[10000]])))
```

```
[4.38087845e+08]
```

```
reg.intercept_
```

```
-29073338.81578946
```

```
d = pd.read_csv('House Price India.csv')
```

```
d.head(10)
```

```
d.head(10)
```

	grade of the house	Area of the house(excluding basement)	Area of the basement	Price
0	10	3370	280	2380000
1	8	1910	1010	1400000
2	8	2910	0	1200000
3	9	3310	0	838000
4	8	1880	830	805000
5	9	1700	900	790000
6	10	3660	0	785000
7	8	1550	690	750000
8	8	1440	950	750000
9	7	1300	900	698000

```
dataset = pd.read_csv('House Price India.csv')
X = dataset.iloc[:,:-1]
y = dataset.iloc[:,1].values
```

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=1/3,random_state = 0)
```

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
y_pred = regressor.predict(X_test)
```

```
y_pred
```

```
array([1440., 4270., 1010., ..., 1380., 1380., 2240.])
```

Experiment:5

AIM: To implement a python program on Multiple Linear Regression for House Price prediction using sklearn.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LinearRegression
```

```
USAhousing = pd.read_csv("house_prices.csv")
print(USAhousing.head())
```

```
    price  grade  yr_built  sqft_living
0  221900      7      1955         1340
1  538000      7      1951         1690
2  180000      6      1933         2720
3  604000      7      1965         1360
4  510000      8      1987         1800
```

```
X = USAhousing[['grade','yr_built','sqft_living']]
y = USAhousing['price']
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size = 0.25,random_state=0)
lm = LinearRegression()
lm.fit(X_train,y_train)
predictions = lm.predict(X_test)
```

```
plt.scatter(y_test,predictions)
plt.xlabel('Actual Prices')
plt.ylabel('Predicted Prices')
plt.title('Actual v/s Predicted Prices')
plt.show()
```


```
print("MAE (Mean Absolute Error):",metrics.mean_absolute_error(y_test,predictions))
print("MSE (Mean Squared Error):",metrics.mean_squared_error(y_test,predictions))
print("RMSE (Root Mean Squared Error):",np.sqrt(metrics.mean_absolute_error(y_test,predictions)))
```

```
MAE (Mean Absolute Error): 152379.93440558165
MSE (Mean Squared Error): 60492564745.971436
RMSE (Root Mean Squared Error): 390.3587252842975
```

Experiment:6

AIM: To implement a Python program for Hyperparameter Tuning on a Decision Tree Classifier.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
df = pd.read_csv("D:/ml/PlayTennis.csv")
```

```
X = df.drop('PlayTennis', axis=1)
y = df['PlayTennis']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
hyperparameters = [
    {'criterion': 'gini', 'max_depth': None, 'min_samples_split': 2, 'min_samples_leaf': 1},
    {'criterion': 'entropy', 'max_depth': 4, 'min_samples_split': 2, 'min_samples_leaf': 1},
    {'criterion': 'gini', 'max_depth': 6, 'min_samples_split': 5, 'min_samples_leaf': 2},
    {'criterion': 'entropy', 'max_depth': 8, 'min_samples_split': 10, 'min_samples_leaf': 4},

]
best_accuracy = 0
best_params = None
best_tree = None
```

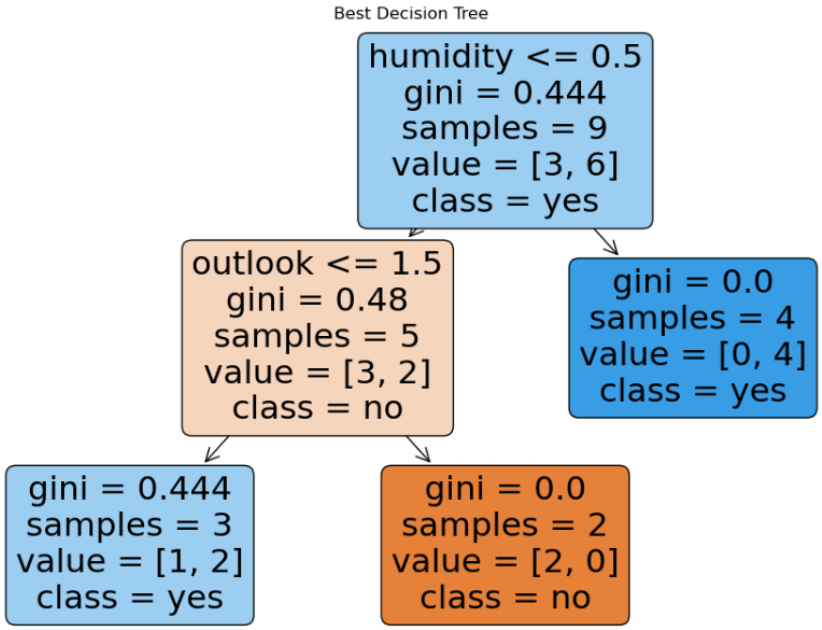
```
for params in hyperparameters:
    tree = DecisionTreeClassifier(**params, random_state=42)
    tree.fit(X_train, y_train)
    y_pred = tree.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Parameters: {params}, Accuracy: {accuracy:.4f}")
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_params = params
        best_tree = tree
```

```
Parameters: {'criterion': 'gini', 'max_depth': None, 'min_samples_split': 2, 'min_samples_leaf': 1}, Accuracy: 0.6000
Parameters: {'criterion': 'entropy', 'max_depth': 4, 'min_samples_split': 2, 'min_samples_leaf': 1}, Accuracy: 0.6000
Parameters: {'criterion': 'gini', 'max_depth': 6, 'min_samples_split': 5, 'min_samples_leaf': 2}, Accuracy: 0.8000
Parameters: {'criterion': 'entropy', 'max_depth': 8, 'min_samples_split': 10, 'min_samples_leaf': 4}, Accuracy: 0.6000
```

```
print(f"\nBest Parameters: {best_params}, Best Accuracy: {best_accuracy:.4f}")
```

```
Best Parameters: {'criterion': 'gini', 'max_depth': 6, 'min_samples_split': 5, 'min_samples_leaf': 2}, Best Accuracy: 0.8000
```

```
plt.figure(figsize=(12, 8))
plot_tree(best_tree,filled=True,feature_names=list(X.columns),class_names=list(label_encoders['PlayTennis'].classes_),rounded=True)
plt.title('Best Decision Tree')
plt.show()
y_pred_best = best_tree.predict(X_test)
print("Best Decision Tree - Classification Report:")
print(classification_report(y_test, y_pred_best,
target_names=label_encoders['PlayTennis'].classes_))
print("Best Decision Tree - Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_best))
print("Best Decision Tree - Accuracy Score:")
print(accuracy_score(y_test, y_pred_best))
```




```
Best Decision Tree - Classification Report:
              precision    recall  f1-score   support

          no       1.00      0.50      0.67         2
         yes       0.75      1.00      0.86         3

    accuracy                           0.80         5
   macro avg       0.88      0.75      0.76         5
weighted avg       0.85      0.80      0.78         5

Best Decision Tree - Confusion Matrix:
[[1 1]
 [0 3]]
Best Decision Tree - Accuracy Score:
0.8
```

Experiment:7

AIM: To implement a Python program for K-Nearest Neighbors (KNN) Classifier.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,classification_report
iris=load_iris()
data=iris.data
target=iris.target
```

```
X_train, X_test, y_train, y_test = train_test_split(data,target,test_size=0.33,random_state=42)
clf=KNeighborsClassifier(n_neighbors=5)
clf.fit(X_train,y_train)
predict=clf.predict(X_test)
print(f"Scikit-learn KNN classifier accuracy:{accuracy_score(y_test,predict)}")
print(classification_report(y_test,predict))
```

```
Scikit-learn KNN classifier accuracy:0.98
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.94      1.00      0.97        15
           2       1.00      0.94      0.97        16

    accuracy                           0.98        50
   macro avg       0.98      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50
```

23261A6660 **16/10/2025**

Experiment:8

AIM: To implement a Python program for Support Vector Classifier (SVC).

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import pandas as pd

wine = load_wine()

df = pd.DataFrame(wine.data, columns=wine.feature_names)
df['target'] = wine.target

print("First five rows of the dataset:\n")
print(df.head())

X = wine.data
y = wine.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

model = SVC(kernel='linear')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=wine.target_names))
```

First five rows of the dataset:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	\
0	14.23	1.71	2.43	15.6	127.0	2.80	
1	13.20	1.78	2.14	11.2	100.0	2.65	
2	13.16	2.36	2.67	18.6	101.0	2.80	
3	14.37	1.95	2.50	16.8	113.0	3.85	
4	13.24	2.59	2.87	21.0	118.0	2.80	

	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	\
0	3.06	0.28	2.29	5.64	1.04	
1	2.76	0.26	1.28	4.38	1.05	
2	3.24	0.30	2.81	5.68	1.03	
3	3.49	0.24	2.18	7.80	0.86	
4	2.69	0.39	1.82	4.32	1.04	

	od280/od315_of_diluted_wines	proline	target
0	3.92	1065.0	0
1	3.40	1050.0	0
2	3.17	1185.0	0
3	3.45	1480.0	0
4	2.93	735.0	0

Accuracy: 0.9814814814814815

Confusion Matrix:

```
 [[19  0  0]
 [ 0 20  1]
 [ 0  0 14]]
```

Classification Report:

	precision	recall	f1-score	support
class_0	1.00	1.00	1.00	19
class_1	1.00	0.95	0.98	21
class_2	0.93	1.00	0.97	14
accuracy			0.98	54
macro avg	0.98	0.98	0.98	54
weighted avg	0.98	0.98	0.98	54

Experiment:9

AIM: To implement a Python program for Logistic Regression.

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import pandas as pd

wine = load_wine()

df = pd.DataFrame(wine.data, columns=wine.feature_names)
df['target'] = wine.target
df['target_name'] = df['target'].apply(lambda x: wine.target_names[x])

print("First five rows of the dataset:\n")
print(df.head())

X = wine.data
y = wine.target
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("\nAccuracy:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=wine.target_names))

sample = [[13.2, 2.7, 2.1, 20.0, 88.0, 2.5, 2.7, 0.26, 1.5, 4.3, 1.0, 2.8, 750.0]]
predicted_class = model.predict(sample)

print("\nPredicted class index:", predicted_class)
print("Predicted Wine Type:", wine.target_names[predicted_class[0]])
```

```
First five rows of the dataset:

   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0    14.23        1.71  2.43               15.6      127.0           2.80
1    13.20        1.78  2.14               11.2      100.0           2.65
2    13.16        2.36  2.67               18.6      101.0           2.80
3    14.37        1.95  2.50               16.8      113.0           3.85
4    13.24        2.59  2.87               21.0      118.0           2.80

   flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
4        2.69                  0.39             1.82             4.32  1.04

   od280/od315_of_diluted_wines  proline  target target_name
0                          3.92   1065.0       0     class_0
1                          3.40   1050.0       0     class_0
2                          3.17   1185.0       0     class_0
3                          3.45   1480.0       0     class_0
4                          2.93    735.0       0     class_0

Accuracy: 1.0

Confusion Matrix:
 [[19  0  0]
 [ 0 21  0]
 [ 0  0 14]]

Classification Report:
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        19
     class_1       1.00      1.00      1.00        21
     class_2       1.00      1.00      1.00        14

    accuracy                           1.00        54
   macro avg       1.00      1.00      1.00        54
weighted avg       1.00      1.00      1.00        54


Predicted class index: [0]
Predicted Wine Type: class_0
```

Experiment: 10

AIM : To implement K-means Clustering

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_breast_cancer
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score, adjusted_rand_score

cancer = load_breast_cancer()
X = cancer.data
y = cancer.target

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
kmeans.fit(X_scaled)
predicted_labels = kmeans.labels_
```

```
print("-- Clustering Evaluation --")
sil_score = silhouette_score(X_scaled, predicted_labels)
print(f"Silhouette Score: {sil_score:.3f}")
ari_score = adjusted_rand_score(y, predicted_labels)
print(f"Adjusted Rand Index (ARI): {ari_score:.3f}\n")

plot_data = pd.DataFrame(X_scaled, columns=cancer.feature_names)
plot_data['Predicted Cluster'] = predicted_labels
plot_data['True Label'] = y


pca = PCA(n_components=2)
pca_result = pca.fit_transform(X_scaled)
plot_data['PCA1'] = pca_result[:, 0]
plot_data['PCA2'] = pca_result[:, 1]


plt.figure(figsize=(16, 7))
```

```
plt.subplot(1, 2, 1)
sns.scatterplot(
    data=plot_data,
    x='PCA1',
    y='PCA2',
    hue='Predicted Cluster',
    palette='viridis',
    s=100,
    alpha=0.8
)
centers = kmeans.cluster_centers_
centers_pca = pca.transform(centers)
plt.scatter(centers_pca[:, 0], centers_pca[:, 1], c='red', s=250, marker='X', label='Centroids')
plt.title('K-Means Clustering Results (on Breast Cancer Data)')
plt.legend()
```

```
plt.subplot(1, 2, 2)
sns.scatterplot(
    data=plot_data,
    x='PCA1',
    y='PCA2',
    hue='True Label',
    palette='deep',
    s=100,
    alpha=0.8
)
plt.title('Actual Breast Cancer Classes (Ground Truth)')
plt.legend()


plt.suptitle('K-Means Clustering vs. Ground Truth on Breast Cancer Dataset')
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

```
-- Clustering Evaluation --
Silhouette Score: 0.343
Adjusted Rand Index (ARI): 0.654
```