INDEX

PRACTICAL 1

1. **Read the numeric data from .CSV file and use some basic operation on it.**

Step 1: import pandas as pd
Step 2: x=pd.Series()
Step 3: import array as ar
Step 4: y = ar.array('i',[5,10,25,30,35,40])
Step 5: x = pd.Series(y)
Step 6: print (x)

OUTPUT:

```
0     5
1    10
2    25
3    30
4    35
5    40
dtype: int64
```

DICTIONARY

Step 1: mylist=[10,20,30,40,50]
Step 2: x=pd.Series(mylist)
Step 3: print(x)

OUTPUT :

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

LIST:

Step 1: mylist1=[100,200,150,250,300]
mylist2=[50,100,150,200,250]
sr1=pd.Series(mylist1)
sr2=pd.Series(mylist2)
Step 2: print(sr1.add(sr2))

OUTPUT:

```
0    150
1    300
2    300
3    450
4    550
dtype: int64
```

---

print(sr1.sub(sr2))

```
0     50
1    100
2      0
3     50
4     50
dtype: int64
```

print(sr1.divide(sr2))

```
0    2.00
1    2.00
2    1.00
3    1.25
4    1.20
dtype: float64
```

print(sr1//sr2)

```
0    2
1    2
2    1
3    1
4    1
dtype: int64
```

print(sr1.multiply(sr2))

```
0     5000
1    20000
2    22500
3    50000
4    75000
dtype: int64
```

print(sr1%sr2)

```
0     0
1     0
2     0
3    50
4    50
dtype: int64
```

print(sr1.gt(sr2))

```
0     True
1     True
2    False
3     True
4     True
dtype: bool
```

.CSV FILE:

Step 1: import numpy as np

Step 2: np.zeros((13,14))

Step 3: np.ones((3,5))

Step 4: np.arange(10,100,3)

Step 5: import pandas as pd

Step 6: data=pd.read_csv("healthcare_patient_journey.csv")

Step 7: print(data.head())

Step 8: print(data.tail())

OUTPUT:

	patient_id	age	gender	chronic_condition	admission_type	department	wait_time_min	length_of_stay_days	procedures_count	medication_count	complications	discharge_status	readmitted_30d	total_cost_€	satisfaction_score
0	1	69	male	0	scheduled	Neurology	41	2	0	3	1	referred	1	1440	2
1	2	38	male	0	emergency	Oncology	17	3	1	2	0	recovered	0	2060	3
2	3	81	male	0	scheduled	Neurology	40	2	3	2	0	recovered	0	2110	3
3	4	67	female	1	emergency	ER	7	4	5	9	0	recovered	0	4070	3
4	5	88	male	1	emergency	Cardiology	34	3	7	5	0	recovered	1	3800	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2995	2996	40	male	0	emergency	Cardiology	41	1	2	1	0	recovered	0	1330	3
2996	2997	21	male	1	scheduled	Polyclinic	54	3	3	2	0	recovered	0	2560	2
2997	2998	72	male	0	scheduled	Neurology	38	1	3	2	0	recovered	0	1660	2
2998	2999	33	female	0	scheduled	Neurology	28	2	4	2	0	recovered	0	2360	2
2999	3000	59	male	1	emergency	Polyclinic	43	5	2	5	0	recovered	1	3450	3

3000 rows × 15 columns

```
print(data.head())
```

```
   patient_id  age  gender  chronic_condition admission_type  department  \
0           1   69    male                  0      scheduled   Neurology
1           2   38    male                  0      emergency    Oncology
2           3   81    male                  0      scheduled   Neurology
3           4   67  female                  1      emergency          ER
4           5   88    male                  1      emergency  Cardiology

   wait_time_min  length_of_stay_days  procedures_count  medication_count  \
0             41                    2                 0                 3
1             17                    3                 1                 2
2             40                    2                 3                 2
3              7                    4                 5                 9
4             34                    3                 7                 5

   complications discharge_status  readmitted_30d  total_cost_€  \
0              1         referred               1          1440
1              0        recovered               0          2060
2              0        recovered               0          2110
3              0        recovered               0          4070
4              0        recovered               1          3800

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

```
print(data.tail())
```

```
      patient_id  age  gender  chronic_condition admission_type  department  \
2995        2996   40    male                  0      emergency  Cardiology
2996        2997   21    male                  1      scheduled  Polyclinic
2997        2998   72    male                  0      scheduled   Neurology
2998        2999   33  female                  0      scheduled   Neurology
2999        3000   59    male                  1      emergency  Polyclinic

      wait_time_min  length_of_stay_days  procedures_count  medication_count  \
2995             41                    1                 2                 1
2996             54                    3                 3                 2
2997             38                    1                 3                 2
2998             28                    2                 4                 2
2999             43                    5                 2                 5

      complications discharge_status  readmitted_30d  total_cost_€  \
2995              0        recovered               0          1330
2996              0        recovered               0          2560
2997              0        recovered               0          1660
2998              0        recovered               0          2360
2999              0        recovered               1          3450

      satisfaction_score
2995                   3
2996                   2
2997                   2
2998                   2
2999                   3
```

PRACTICAL 2

2. Write a program to demonstrate the working of the decision tree algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

Step 1: pip install scikit-learn
Step 2: import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
Step 3: data = {
'Outlook': ['Sunny','Sunny','Overcast','Rain','Rain','Rain','Overcast'],
'Temperature': ['Hot','Hot','Hot','Mild','Cool','Cool','Cool'],
'Humidity': ['High','High','High','High','Normal','Normal','Normal'],
'Wind': ['Weak','Strong','Weak','Weak','Weak','Strong','Strong'],
'PlayTennis': ['No','No','Yes','Yes','Yes','No','Yes']
}

df = pd.DataFrame(data)
print(df)
Step 4: le = LabelEncoder()

df['Outlook'] = le.fit_transform(df['Outlook'])
df['Temperature'] = le.fit_transform(df['Temperature'])
df['Humidity'] = le.fit_transform(df['Humidity'])
df['Wind'] = le.fit_transform(df['Wind'])
df['PlayTennis'] = le.fit_transform(df['PlayTennis'])

print(df)
Step 5: X = df[['Outlook','Temperature','Humidity','Wind']]
y = df['PlayTennis']
Step 6: model = DecisionTreeClassifier()
model.fit(X, y)
Step 7: new_sample = [[2, 2,1,1]] # encoded values
result = model.predict(new_sample)

print("Prediction:", result)
Step 8: if result == 1:
print("Play Tennis: YES")
else:
print("Play Tennis: NO")

OUTPUT:

	Outlook	Temperature	Humidity	Wind	PlayTennis
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes

	Outlook	Temperature	Humidity	Wind	PlayTennis
0	2	1	0	1	0
1	2	1	0	0	0
2	0	1	0	1	1
3	1	2	0	1	1
4	1	0	1	1	1
5	1	0	1	0	0
6	0	0	1	0	1

```
]: if result == 1:
       print("Play Tennis: YES")
   else:
       print("Play Tennis: NO")
```

```
Play Tennis: NO
```

PRACTICAL 3

3. Write a program to demonstrate the working of the Random Forest algorithm.

Step 1: pip install scikit-learn
Step 2: from sklearn.ensemble import RandomForestClassifier
Step 3: X=[
[1,20],
[2,30],
[3,40],
[4,50],
[5,60],
[6,70]
]
y=[0,0,0,1,1,1]
Step 4: model=RandomForestClassifier(n_estimators=5)
Step 5: model.fit(X,y)
Step 6: prediction= model.predict([[3,51]])
Step 7: if prediction==1:
print("Student will pass")
else:
print("Student will fail")
Step 8: import pandas as pd
Step 9: from sklearn.ensemble import RandomForestClassifier
Step 10: data={
'Experience':[1,2,3,4,5],
'Salary':['Low','Low','High','High','High']
}
Step 11: df=pd.DataFrame(data)
Step 12: df['Salary']=df['Salary'].map({'Low':0,'High':1})
Step 13: X= df[['Experience']]
y=df['Salary']
Step 14: model = RandomForestClassifier(n_estimators=3)
Step 15: model.fit(X,y)
Step 16: prediction=model.predict([[3]])
Step 17: if prediction==1:
print("High Salary")
else:
print("Low Salary")

OUTPUT:

RandomForestClassifier

▼ Parameters

n_estimators	3
criterion	'gini'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	'sqrt'
max_leaf_nodes	None
min_impurity_decrease	0.0
bootstrap	True
oob_score	False
n_jobs	None
random_state	None
verbose	0
warm_start	False
class_weight	None
ccp_alpha	0.0
max_samples	None
monotonic_cst	None

```
if prediction==1:
    print("High Salary")
else:
    print("Low Salary")
```

```
High Salary
```

PRACTICAL 4

4. **Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.**

Step 1: import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score

Step 2: data=pd.read_csv("tennis-1.csv")

Step 3: print("Dataset:\n",data)

Step 4: le=LabelEncoder()

Step 5: data['Outlook']=le.fit_transform(data['Outlook'])

data['Temperature']=le.fit_transform(data['Temperature'])

data['Humidity']=le.fit_transform(data['Humidity'])

data['Wind']=le.fit_transform(data['Wind'])

data['PlayTennis']=le.fit_transform(data['PlayTennis'])

Step 6: X= data [['Outlook','Temperature','Humidity','Wind']]

Y=data['PlayTennis']

Step 7: X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.5)

Step 8: model = GaussianNB()

Step 9: model.fit(X_train, Y_train)

Step 10: prediction = model.predict(X_test)

Step 11: accuracy=accuracy_score(Y_test,prediction)

Step 12: print("\nPrediction:",prediction)

Step 13: print("\nAccuracy:",accuracy*100,"%")

OUTPUT:

```
Dataset:
      Outlook Temperature Humidity    Wind PlayTennis
0       Sunny         Hot     High    Weak         No
1       Sunny         Hot     High  Strong         No
2    Overcast         Hot     High    Weak        Yes
3        Rain        Mild     High    Weak        Yes
4        Rain        Cool   Normal    Weak        Yes
5        Rain        Cool   Normal  Strong         No
6    Overcast        Cool   Normal  Strong        Yes
7       Sunny        Mild     High    Weak         No
8       Sunny        Cool   Normal    Weak        Yes
9        Rain        Mild   Normal    Weak        Yes
10      Sunny        Mild   Normal  Strong        Yes
11   Overcast        Mild     High  Strong        Yes
12   Overcast         Hot   Normal    Weak        Yes
13       Rain        Mild     High  Strong         No
```

```
print("\nPrediction:",prediction)
```

```
Prediction: [1 1 0 0 0 0 1]
```

```
print("\nAccuracy:",accuracy*100,"%")
```

```
Accuracy: 71.42857142857143 %
```

PRACTICAL 5

5. Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set.

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score

# Step 1: Training data
documents = [
    "win free prize",
    "claim money now",
    "meeting at office",
    "project submission today"
]

labels = ["Spam", "Spam", "NotSpam", "NotSpam"]

# Step 2: Convert text into numbers
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(documents)


# Step 3: Train model
model = MultinomialNB()
model.fit(X, labels)

# Step 4: Test data
test_docs = ["free money", "claim project"]
actual_labels = ["Spam", "Spam"]

X_test = vectorizer.transform(test_docs)

# Step 5: Prediction
predicted_labels = model.predict(X_test)

print("Predicted Output:", predicted_labels)


# Step 6: Performance metrics
accuracy = accuracy_score(actual_labels, predicted_labels)
```

```
precision = precision_score(actual_labels, predicted_labels, pos_label="Spam",
average='binary')
recall = recall_score(actual_labels, predicted_labels, pos_label="Spam", average='binary')

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

OUTPUT:

```
print("Predict Output:", predicted_labels)
```

```
Predict Output: ['Spam' 'NotSpam']
```

```
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
```

```
Accuracy: 0.5
Precision: 1.0
Recall: 0.5
```

PRACTICAL 6

6. Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API

Step 1: import pandas as pd

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
```

Step 2: # Sample heart dataset

```
data = pd.DataFrame({
    'Age': [45, 50, 39, 60, 48, 35, 55, 42],
    'BP': [130, 140, 120, 150, 135, 118, 145, 125],
    'Cholesterol': [230, 250, 200, 270, 240, 190, 260, 210],
    'ChestPain': [1, 1, 0, 1, 1, 0, 1, 0],
    'HeartDisease': [1, 1, 0, 1, 1, 0, 1, 0]
})
```

Step 3: # Input and output

```
X = data[['Age', 'BP', 'Cholesterol', 'ChestPain']]
y = data['HeartDisease']
```

Step 4: # Split data

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)
```

Step 5: # Create model

```
model = GaussianNB()
```

Step 6: # Train model

```
model.fit(X_train, y_train)
```

Step 7: # Prediction

predicted = model.predict(X_test)

Step 8: # Performance

accuracy = accuracy_score(y_test, predicted)

precision = precision_score(y_test, predicted)

recall = recall_score(y_test, predicted)

print("Predicted:", predicted)

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

OUTPUT:

```
Predicted: [1 0]
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
```

PRACTICAL 7

7. Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

Step 1: from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

import pandas as pd

Step 2: iris = load_iris()

Step 3: df = pd.DataFrame(iris.data, columns=iris.feature_names)

df['target'] = iris.target

df

Step 4: X = iris.data

y = iris.target

Step 5: X_train, X_test, y_train, y_test = train_test_split(

X, y, test_size=0.2, random_state=42

)

Step 6: model = KNeighborsClassifier(n_neighbors=3)

Step 7: model.fit(X_train, y_train)

Step 8: y_pred = model.predict(X_test)

Step 9: print("Actual Flower\tPredicted Flower\tResult")

for actual, predicted in zip(y_test, y_pred):

actual_name = iris.target_names[actual]

predicted_name = iris.target_names[predicted]

if actual == predicted:

```
        result = "Correct"
    else:
        result = "Wrong"


    print(actual_name, "\t", predicted_name, "\t", result)
```

OUTPUT:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

Actual Flower	Predicted Flower	Result
versicolor	versicolor	Correct
setosa	setosa	Correct
virginica	virginica	Correct
versicolor	versicolor	Correct
versicolor	versicolor	Correct
setosa	setosa	Correct
versicolor	versicolor	Correct
virginica	virginica	Correct
versicolor	versicolor	Correct
versicolor	versicolor	Correct
virginica	virginica	Correct
setosa	setosa	Correct
setosa	setosa	Correct
setosa	setosa	Correct
setosa	setosa	Correct
versicolor	versicolor	Correct
virginica	virginica	Correct
versicolor	versicolor	Correct
versicolor	versicolor	Correct
virginica	virginica	Correct
setosa	setosa	Correct
virginica	virginica	Correct
setosa	setosa	Correct
virginica	virginica	Correct
virginica	virginica	Correct
virginica	virginica	Correct
virginica	virginica	Correct
virginica	virginica	Correct
setosa	setosa	Correct
setosa	setosa	Correct

PRACTICAL 8

8. Write a program to demonstrate the working of the K-means clustering algorithm.

Step 1: from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import numpy as np

Step 2: X = np.array([

[1,2],

[1.5,1.8],

[5,8],

[8,8],

[1,0.6],

[9,11],

[8,2],

[10,2],

[9,3]

])

Step 3: kmeans = KMeans(n_clusters=2, random_state=0)

Step 4: kmeans.fit(X)

Step 5: labels = kmeans.labels_

Step 6: centroids = kmeans.cluster_centers_

Step 7: print("Cluster Labels:")

print(labels)

Step 8: print("\nCentroids:")

print(centroids)

Step 9: plt.scatter(X[:, 0], X[:, 1], c=labels)

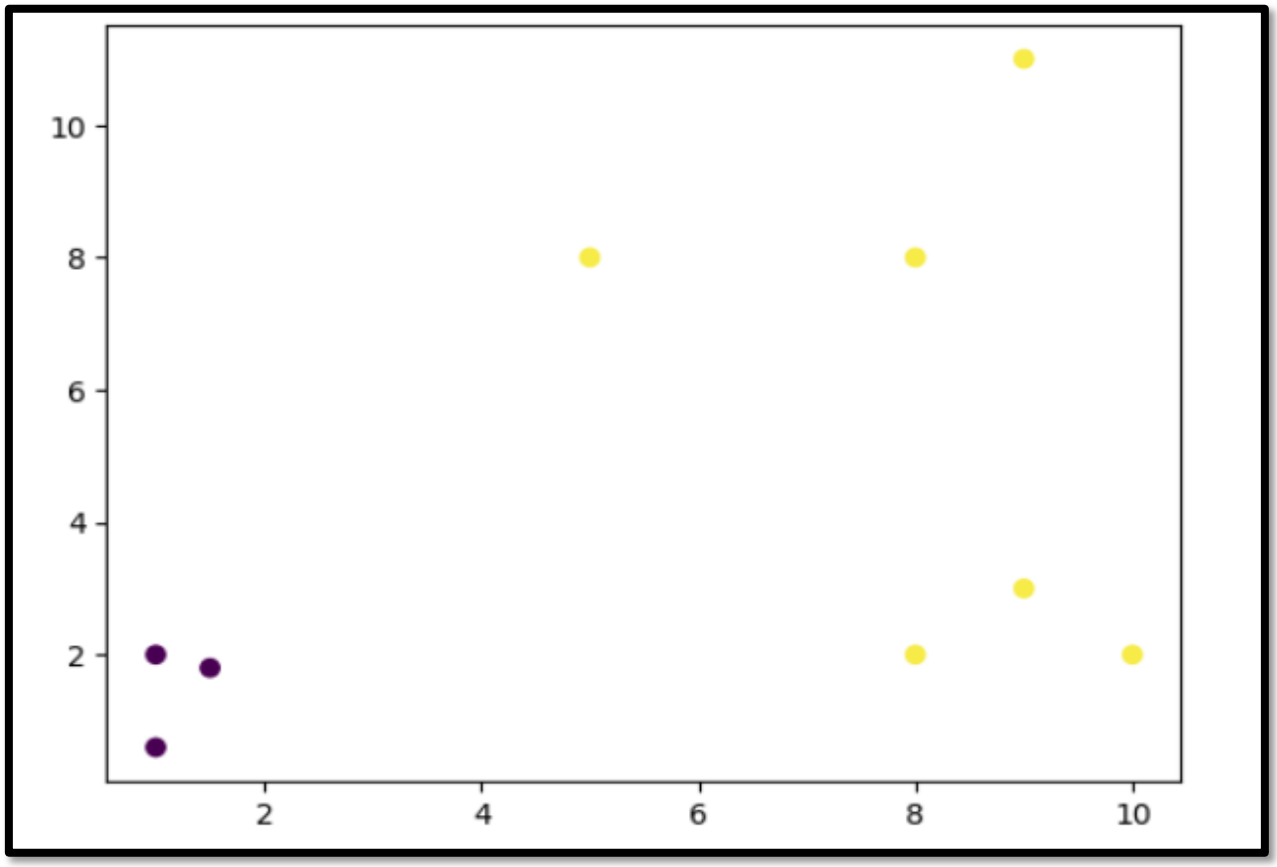
plt.scatter(centroids[:, 0], centroids[:, 1], marker='+', s=300)

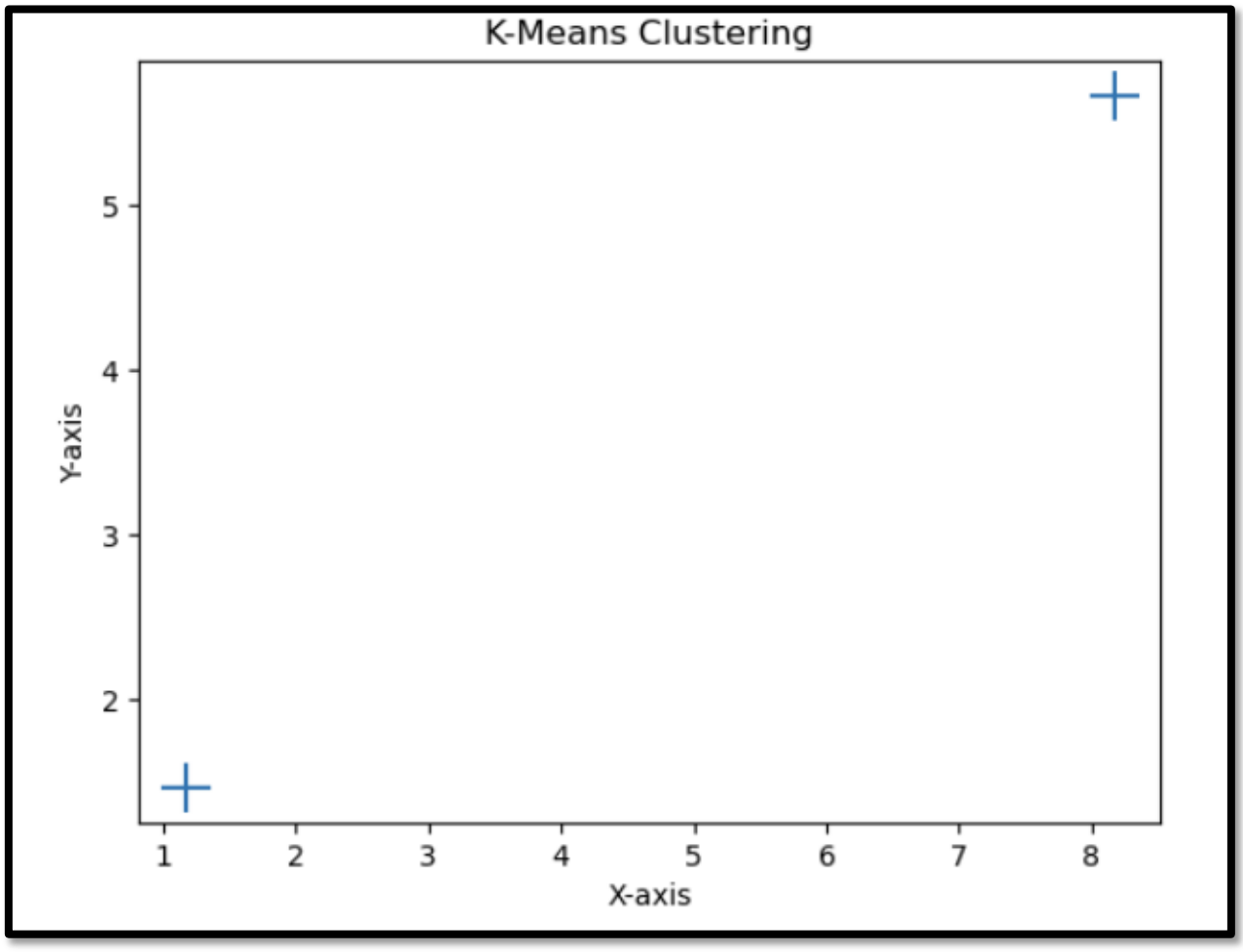
plt.title("K-Means Clustering")

```
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

OUTPUT:

```
print("Cluster Labels:")
print(labels)

Cluster Labels:
[0 0 1 1 0 1 1 1 1]

print("\nCentroids:")
print(centroids)

Centroids:
[[1.16666667 1.46666667]
 [8.16666667 5.66666667]]
```




K-Means Clustering
Y-axis
X-axis
1
2
3
4
5
6
7
8
2
3
4
5

PRACTICAL 9

9. Write a program to demonstrate the working of the Support Vector Machine for Classification Algorithm.

Step 1: import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score

Step 2: X, y = datasets.make_classification(

n_samples=100,

n_features=2,

n_redundant=0,

n_informative=2,

n_clusters_per_class=1,

random_state=42

)

Step 3: X_train, X_test, y_train, y_test = train_test_split(

X, y, test_size=0.2, random_state=42

)

Step 4: svm_model = SVC(kernel='linear')

Step 5: svm_model.fit(X_train, y_train)

Step 6: y_pred = svm_model.predict(X_test)

Step 7: accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)

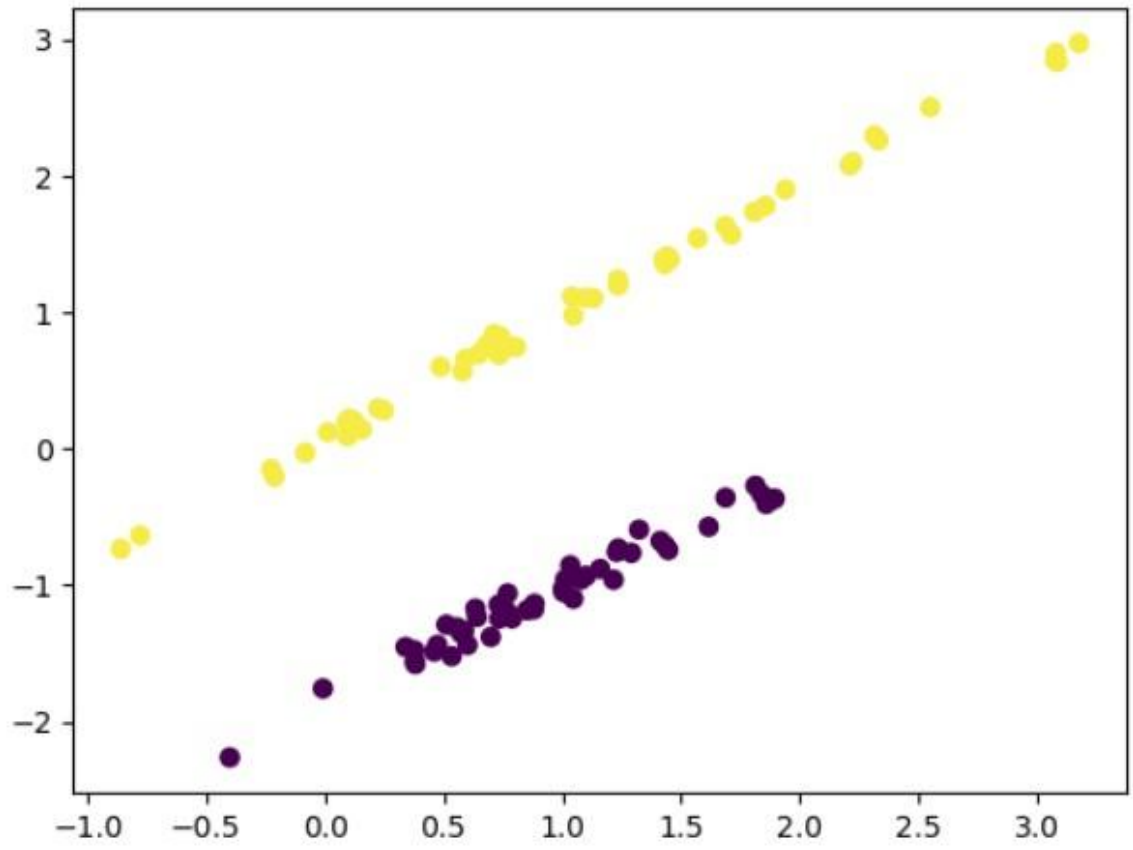
Step 8: plt.scatter(X[:, 0], X[:, 1], c=y)

OUTPUT:

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

```
<matplotlib.collections.PathCollection at 0x1c11e81efd0>
```



PRACTICAL 10

10. Write a program to demonstrate the working of the Hierarchical Clustering.

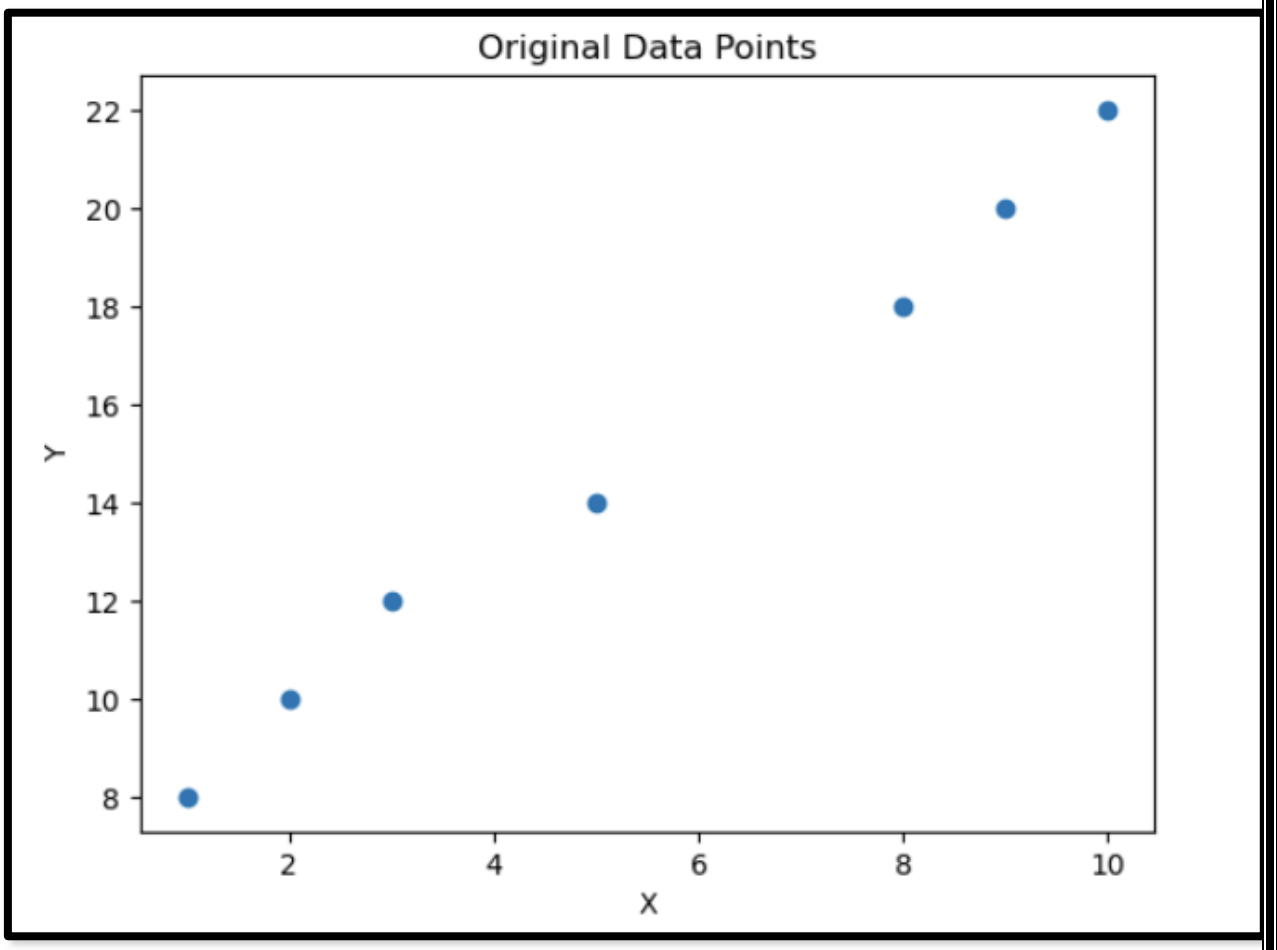
```
Step 1: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
Step 2: from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram,linkage
Step 3: data={
   'X':[2,3,5,8,1,9,10],
   'Y':[10,12,14,18,8,20,22]
}
df=pd.DataFrame(data)
print(df)
Step 4: plt.scatter(df['X'],df['Y'])
plt.title("Original Data Points")
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
Step 5: linked = linkage(df, method='ward')
dendrogram(linked)
plt.title("Dendrogram")
plt.xlabel("Data Points")
plt.ylabel("Distance")
plt.show()
Step 6: model = AgglomerativeClustering(n_clusters=2)
clusters = model.fit_predict(df)
Step 7: df['Cluster'] = clusters
```

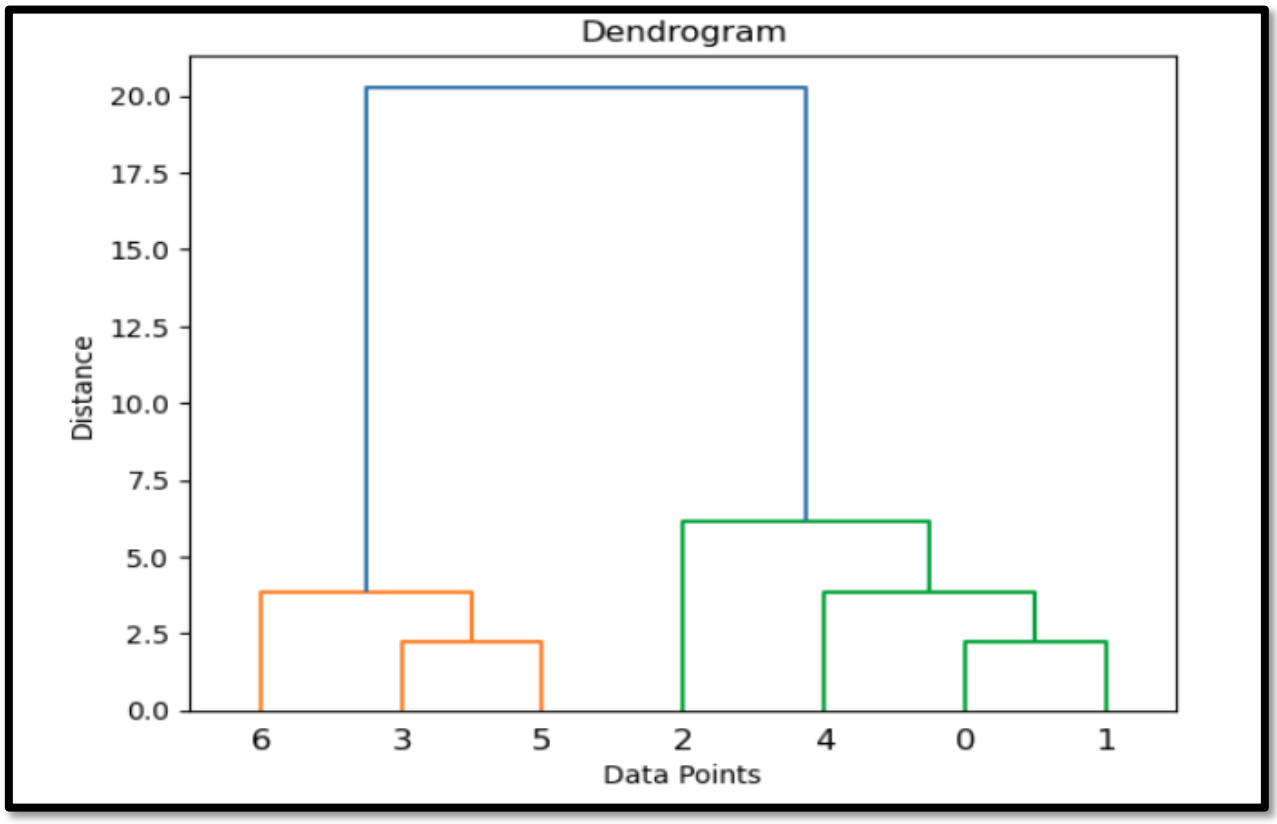
```
print(df)

print(clusters)
```

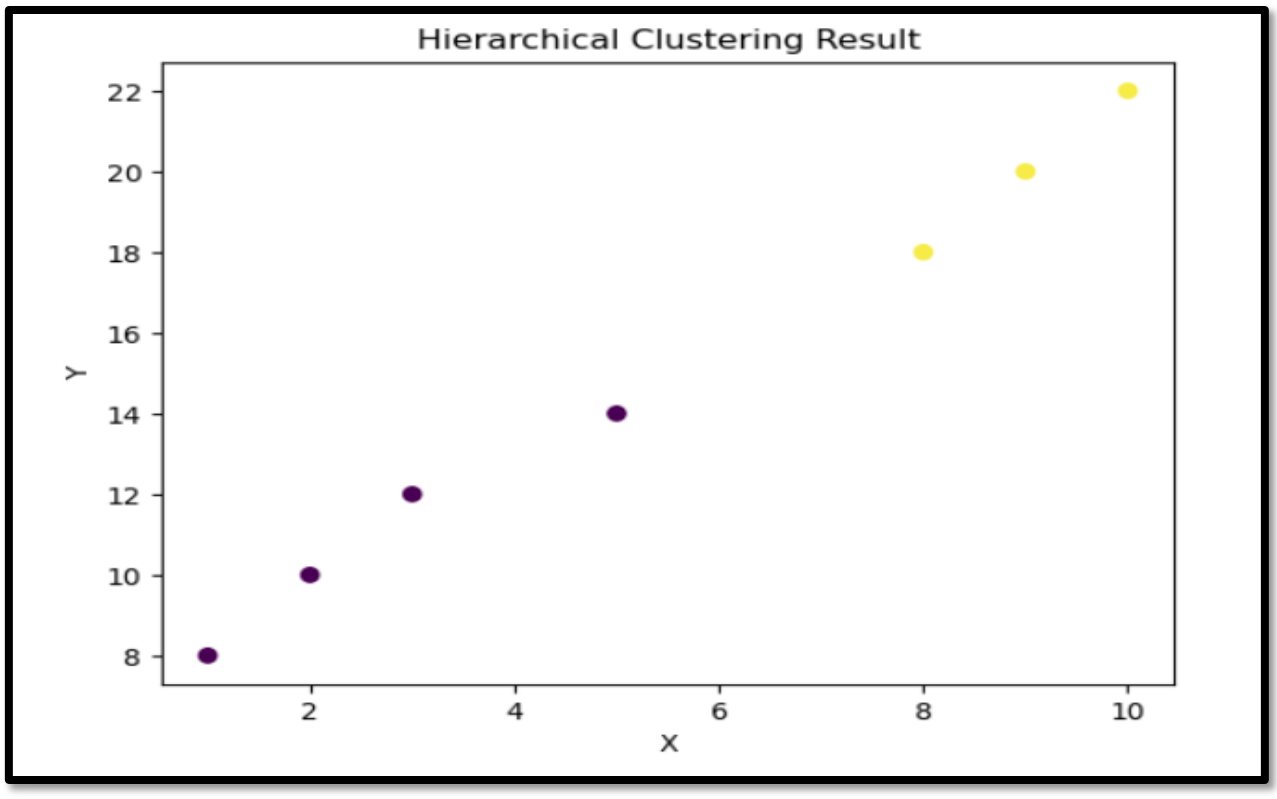
Step 8:

```
plt.scatter(df['X'], df['Y'], c=df['Cluster'])

plt.title("Hierarchical Clustering Result")

plt.xlabel("X")

plt.ylabel("Y")

plt.show()
```

OUTPUT:




Dendrogram
Distance
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
6
3
5
2
4
0
1
Data Points


Hierarchical Clustering Result
Y
22
20
18
16
14
12
10
8
2
4
6
8
10
X