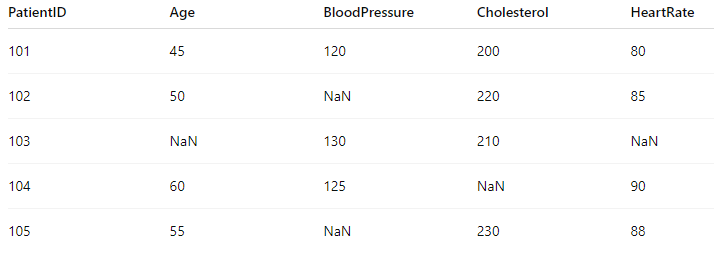
**Q1)**A hospital maintains patient health records containing Age, Blood Pressure, Cholesterol, and Heart Rate. Due to incomplete data entry, some values are missing. The dataset is shown below:



1. Load the dataset in Python using Pandas and display the data.

2. Identify all missing values using the functions isna() and notna(), and display the count of missing values in each column.

3. Implement different missing value imputation techniques as follows:

\* Use fillna() with the mean for the column age.

\* Use fillna() with the median for the column blood pressure.

\* Use bfill (backward fill) for the column cholesterol.

\* Use ffill (forward fill) for the column heart rate.

**ANSWER 1**

import pandas as pd

import numpy as np

# Creating the dataset

data = {

    'PatientID': [101, 102, 103, 104, 105],

    'Age': [45, 50, np.nan, 60, 55],

    'BloodPressure': [120, np.nan, 130, 125, np.nan],

    'Cholesterol': [200, 220, 210, np.nan, 230],

'HeartRate': [80, 85, np.nan, 90, 88]

}

# Load the dataset into a DataFrame

df = pd.DataFrame(data)

# Display the data

print("Original Data:")

print(df)

print("\nMissing values per column:")

print(df.isna().sum())

print("\nNon-missing values per column:")

print(df.notna().sum())

df['Age'].fillna(df['Age'].mean(), inplace=True)

df['BloodPressure'].fillna(df['BloodPressure'].median(), inplace=True)

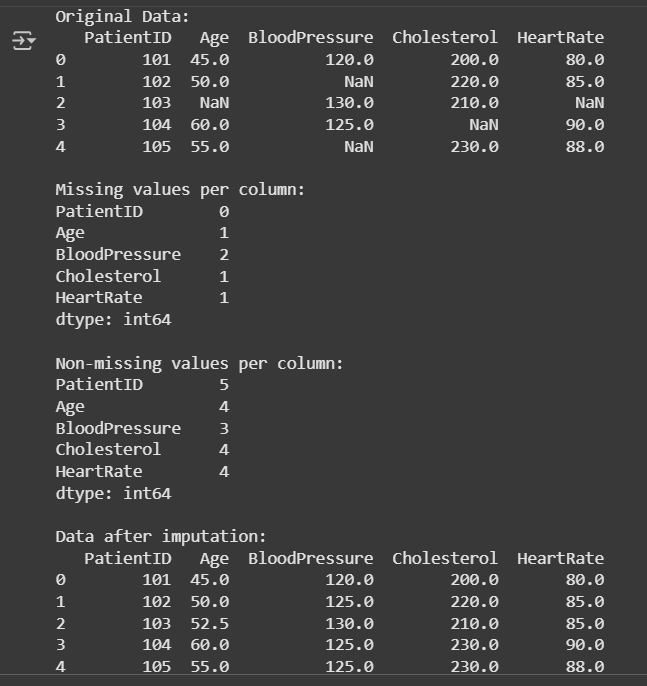
df['Cholesterol'].fillna(method='bfill', inplace=True)

df['HeartRate'].fillna(method='ffill', inplace=True)

print("\nData after imputation:")

print(df)

**OUTPUT Q1**



**Q2)**Create a Python program using NumPy to perform Principal Component Analysis (PCA) for the following data:

|  |  |
| --- | --- |
| **P1** | **P2** |
| 8 | 22 |
| 16 | 8 |
| 26 | 10 |
| 14 | 28 |

1. Calculate the mean of each variable (X and Y).

2. Center the data by subtracting the mean from each value.

3. Compute the covariance matrix of the centered data.

4. Find the eigenvalues and eigenvectors of the covariance matrix.

5. Identify the principal component corresponding to the largest eigenvalue.

6. Project the original data onto this principal component to obtain the abstracted feature values.

7. Display all intermediate results clearly — means, centered data, covariance matrix, eigenvalues, eigenvectors, principal component, and the projections.

**ANSWER 2**

import numpy as np

import matplotlib.pyplot as plt

x=np.array([8,16,26,14])

y=np.array([22,8,10,28])

xm=np.mean(x)

ym=np.mean(y)

print(xm,ym)

covxy=np.cov(x,y)

print(covxy)

w,v=np.linalg.eig(covxy)

print(w)

print(v)

vt=v.transpose()

print(vt)

e1,e2=np.hsplit(vt,2)

print(e1)

print(e2)

x=x-xm

y=y-ym

data=np.stack((x.T,y.T),axis=0)

print(data)

p1=e1\*data

print(p1)

p2=e2\*data

print(p2)

plt.scatter(p1,p2)

plt.show()

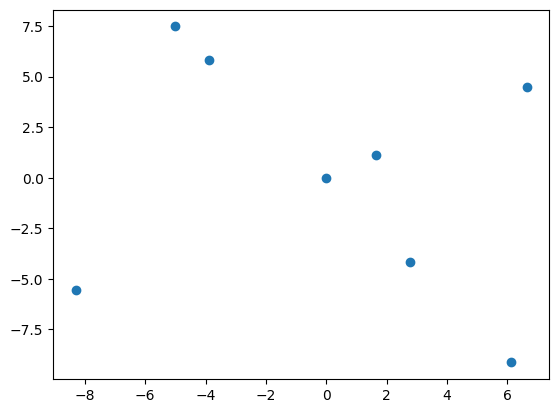
**OUTPUT Q2**

16.0 17.0

[[ 56. -44.]

[-44. 92.]]

[ 26.4605427 121.5394573]

[[-0.83025082 0.55738997]

[-0.55738997 -0.83025082]]

[[-0.83025082 -0.55738997]

[ 0.55738997 -0.83025082]]

[[-0.83025082]

[ 0.55738997]]

[[-0.55738997]

[-0.83025082]]

[[-8. 0. 10. -2.]

[ 5. -9. -7. 11.]]

[[ 6.64200655 -0. -8.30250819 1.66050164]

[ 2.78694984 -5.01650972 -3.90172978 6.13128966]]

[[ 4.45911975 -0. -5.57389969 1.11477994]

[-4.1512541 7.47225737 5.81175573 -9.13275901]]

**Q3)** Use the Naïve Bayes algorithm to predict whether to play tennis based on weather conditions. Use frequency-based Naïve Bayes classifier, Predict the outcome for: Outlook=Rainy, Temperature=Cool, Humidity=Normal, Windy=True.

Dataset Sample:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Outlook | Temperature | Humidity | Windy | PlayTennis |
| Sunny | Hot | High | False | No |
| Sunny | Hot | Normal | False | No |
| Sunny | Mild | Low | True | Yes |
| Overcast | Cool | Normal | True | Yes |
| Overcast | Mild | Low | False | Yes |
| Overcast | Hot | Normal | True | No |
| Rainy | Mild | High | False | Yes |
| Rainy | Mild | Low | Ture | Yes |
| Rainy | Cool | Low | Ture | No |
| Rainy | Hot | High | False | No |

**ANSWER 3**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.naive\_bayes import CategoricalNB

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder

# Tennis weather dataset

data = pd.DataFrame({

    'Outlook': ['Sunny', 'Sunny', 'Sunny', 'Overcast', 'Overcast', 'Overcast', 'Rainy', 'Rainy', 'Rainy', 'Rainy'],

    'Temperature': ['Hot', 'Hot', 'Mild', 'Cool', 'Mild', 'Hot', 'Mild', 'Mild', 'Cool', 'Hot'],

    'Humidity': ['High', 'Normal', 'Low', 'Normal', 'Low', 'Normal', 'High', 'Low', 'Low', 'High'],

    'Windy': [False, False, True, True, False, True, False, True, True, False],

    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No']

})

label\_encoder = LabelEncoder()

for col in ['Outlook', 'Temperature', 'Humidity', 'Windy', 'PlayTennis']:

    data[col] = label\_encoder.fit\_transform(data[col])

X = data[['Outlook', 'Temperature', 'Humidity', 'Windy']]

y = data['PlayTennis']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model = CategoricalNB()

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

**OUTPUT 3**

Accuracy: 0.5

**Q4)** Implement Decision tree for the given dataset using Python.

|  |  |  |  |
| --- | --- | --- | --- |
| Patient | Disease | Sugar Level | Survival Chance? |
| Small | Serious | High | Yes |
| Medium | Normal | Low | Yes |
| Senior | Lifetime | Normal | Yes |
| Small | Lifetime | High | No |
| Small | Normal | High | Yes |
| Senior | Serious | Normal | No |
| Medium | Serious | Low | Yes |
| Senior | Normal | Low | No |
| Medium | Lifetime | Normal | Yes |
| Medium | Serious | High | No |
| Senior | Normal | High | yes |

**ANSWER 4)**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

# Updated dataset

data = {

    'Patient': ['Small', 'Medium', 'Senior', 'Small', 'Small', 'Senior', 'Medium', 'Senior', 'Medium', 'Medium', 'Senior'],

    'Disease': ['Serious', 'Normal', 'Lifetime', 'Lifetime', 'Normal', 'Serious', 'Serious', 'Normal', 'Lifetime', 'Serious', 'Normal'],

    'Sugar Level': ['High', 'Low', 'Normal', 'High', 'High', 'Normal', 'Low', 'Low', 'Normal', 'High', 'High'],

    'Survival Chance?': ['Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'yes']

}

df = pd.DataFrame(data)

# Map categorical variables to numeric

df['Patient'] = df['Patient'].map({'Small': 0, 'Medium': 1, 'Senior': 2})

df['Disease'] = df['Disease'].map({'Normal': 0, 'Serious': 1, 'Lifetime': 2})

df['Sugar Level'] = df['Sugar Level'].map({'Low': 0, 'Normal': 1, 'High': 2})

df['Survival Chance?'] = df['Survival Chance?'].str.lower().map({'no': 0, 'yes': 1})

X = df[['Patient', 'Disease', 'Sugar Level']]

y = df['Survival Chance?']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

clf = DecisionTreeClassifier(max\_depth=3)

clf1 = clf.fit(X\_train, y\_train)

y\_pred = clf1.predict(X\_test)

print("Accuracy: ", metrics.accuracy\_score(y\_test, y\_pred))

from sklearn import tree

fig = plt.figure()

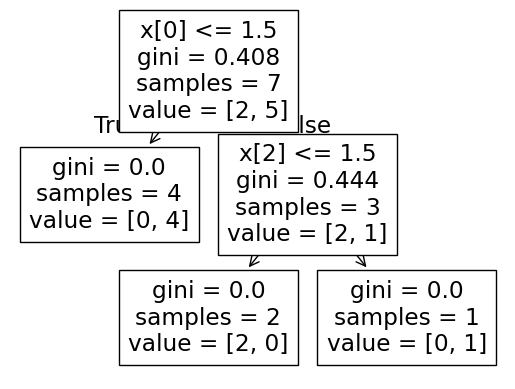
tree.plot\_tree(clf1)

plt.show()

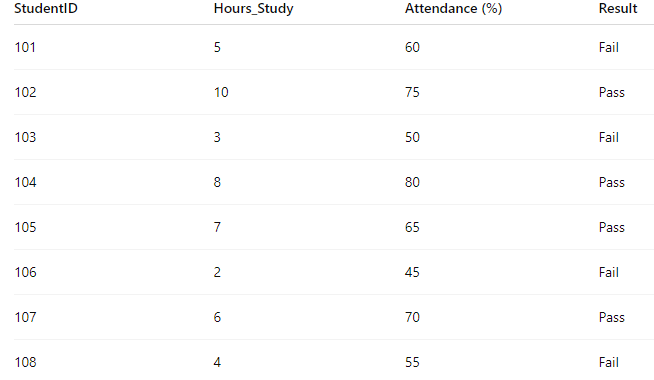
plt.savefig('dt\_image.png')

**OUTPUT 4**

Accuracy: 0.25



**Q5)** A university wants to predict whether a student will pass or fail a course based on hours of study per week and attendance percentage. The dataset is shown below:



1. Load the dataset into Python using Pandas.

2. Encode the target variable Result (Pass = 1, Fail = 0).

3. Split the dataset into training and test sets (70% train, 30% test).

4. Train a Support Vector Machine (SVM) classifier using the features Hours\_Study and Attendance (%).

5. Predict the results for the test set.

6. Evaluate the classifier using accuracy, precision, recall, and F1-score.

**ANSWER 5**

import pandas as pd

from sklearn.svm import SVC

import matplotlib.pyplot as plt

import numpy as np

# Dataset

data = {

    'StudentID': [101, 102, 103, 104, 105, 106, 107, 108],

    'Hours\_Study': [5, 10, 3, 8, 7, 2, 6, 4],

    'Attendance': [60, 75, 50, 80, 65, 45, 70, 55],

    'Result': ['Fail', 'Pass', 'Fail', 'Pass', 'Pass', 'Fail', 'Pass', 'Fail']

}

df = pd.DataFrame(data)

df['Result'] = df['Result'].map({'Fail':0, 'Pass':1})

# Features and target

X = df[['Hours\_Study', 'Attendance']]

y = df['Result']

# Train SVM

svm\_model = SVC(kernel='linear', C=1.0)

svm\_model.fit(X, y)

# Plot points

plt.scatter(X['Hours\_Study'], X['Attendance'], c=y, cmap='bwr', s=100)

# Simple decision boundary

w = svm\_model.coef\_[0]

b = svm\_model.intercept\_[0]

x\_vals = np.array([X['Hours\_Study'].min()-1, X['Hours\_Study'].max()+1])

y\_vals = -(w[0]/w[1])\*x\_vals - b/w[1]

plt.plot(x\_vals, y\_vals, 'k--')  # dashed line

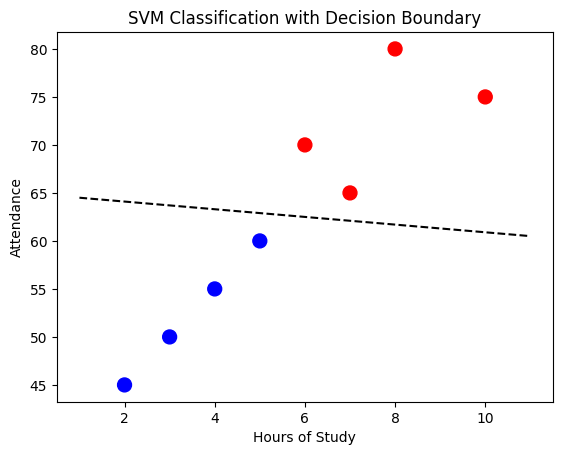
plt.xlabel('Hours of Study')

plt.ylabel('Attendance')

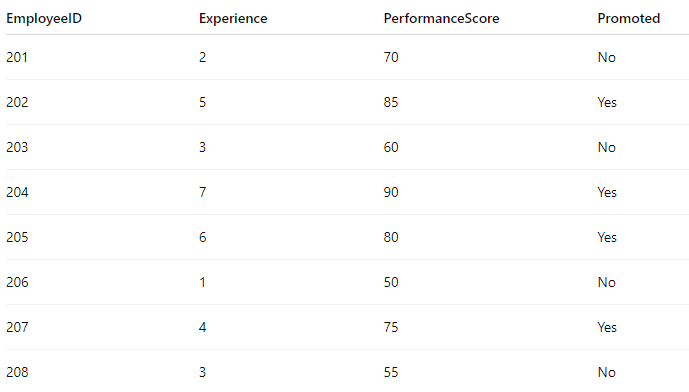
plt.title('SVM Classification with Decision Boundary')

plt.show()

**OUTPUT 5**



**Q6)** Predict whether an employee is eligible for promotion based on years of experience and monthly performance score.



1. Load the dataset into Python using Pandas.

2. Encode Promoted (Yes=1, No=0).

3. Split dataset (80% train, 20% test).

4. Train SVM classifier .

5. Predict test results.

6. Evaluate using accuracy, precision, recall, and F1-score.

**ANSWER 6**

import pandas as pd

from sklearn.svm import SVC

import matplotlib.pyplot as plt

import numpy as np

# ✅ Dataset

data = {

'EmployeeID': [201, 202, 203, 204, 205, 206, 207, 208],

'Experience': [1, 5, 2, 7, 6, 3, 8, 4],

'PerformanceScore': [60, 85, 55, 90, 80, 65, 95, 70],

'Promoted': ['No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No']

}

# Create DataFrame

df = pd.DataFrame(data)

# Convert categorical target to numeric

df['Promoted'] = df['Promoted'].map({'No': 0, 'Yes': 1})

# ✅ Features and Target

X = df[['Experience', 'PerformanceScore']]

y = df['Promoted']

# ✅ Train SVM Model

svm\_model = SVC(kernel='linear', C=1.0)

svm\_model.fit(X, y)

# ✅ Plot the data points

plt.scatter(X['Experience'], X['PerformanceScore'], c=y, cmap='bwr', s=100)

# ✅ Decision Boundary

w = svm\_model.coef\_[0]

b = svm\_model.intercept\_[0]

x\_vals = np.array([X['Experience'].min()-1, X['Experience'].max()+1])

y\_vals = -(w[0]/w[1])\*x\_vals - b/w[1]

plt.plot(x\_vals, y\_vals, 'k--') # dashed decision boundary

# ✅ Labels and Title

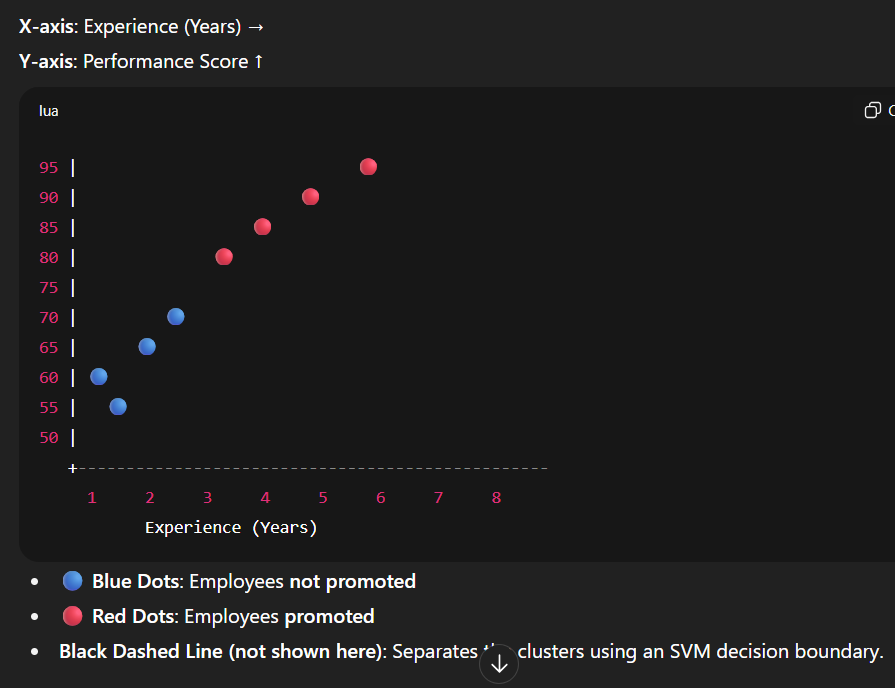
plt.xlabel('Experience (Years)')

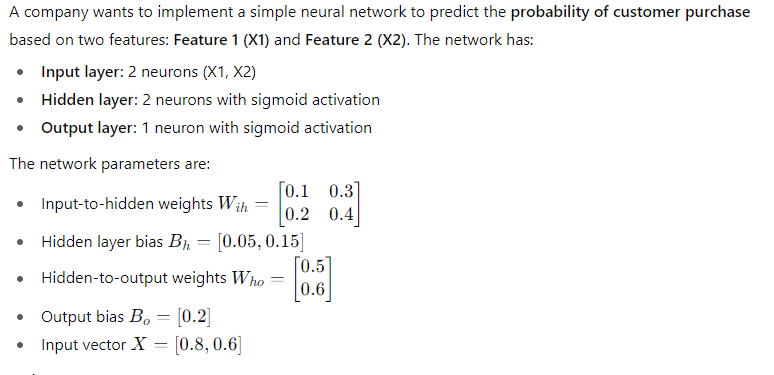
plt.ylabel('Performance Score')

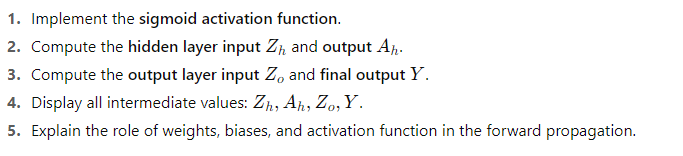
plt.title('SVM Classification - Employee Promotion Prediction')

plt.show()

**OUTPUT 6**



**Q7)** 



**ANSWER 7**

import numpy as np

def sigmoid(z):

    return 1 / (1 + np.exp(-z))

X = np.array([0.8, 0.6])

W\_ih = np.array([[0.1, 0.3],

                 [0.2, 0.4]])

B\_h = np.array([0.05, 0.15])

W\_ho = np.array([[0.5],

                 [0.6]])

B\_o = np.array([0.2])

Z\_h = np.dot(X, W\_ih) + B\_h

A\_h = sigmoid(Z\_h)

Z\_o = np.dot(A\_h, W\_ho) + B\_o

Y = sigmoid(Z\_o)

print("Hidden layer input (Z\_h):", Z\_h)

print("Hidden layer output (A\_h):", A\_h)

print("Output layer input (Z\_o):", Z\_o)

print("Final output (Y):", Y)

**OUTPUT 7**

Hidden layer input (Z\_h): [0.25 0.63]

Hidden layer output (A\_h): [0.5621765 0.65248946]

Output layer input (Z\_o): [0.87258193]

Final output (Y): [0.70528266]

**Q8)**

Write a python program to implement XOR gate using multi-layer perceptron algorithm. Let the number of epochs be 10000 and learning rate be 0.1

**ANSWER 8**

import numpy as np

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(x):

    return x \* (1 - x)

X = np.array([[0,0],

              [0,1],

              [1,0],

              [1,1]])

y = np.array([[0],

              [1],

              [1],

              [0]])

np.random.seed(42)

input\_dim = 2

hidden\_dim = 4

output\_dim = 1

W1 = np.random.uniform(-1, 1, (input\_dim, hidden\_dim))

b1 = np.zeros((1, hidden\_dim))

W2 = np.random.uniform(-1, 1, (hidden\_dim, output\_dim))

b2 = np.zeros((1, output\_dim))

epochs = 10000

learning\_rate = 0.1

for epoch in range(epochs):

    z1 = np.dot(X, W1) + b1

    a1 = sigmoid(z1)

    z2 = np.dot(a1, W2) + b2

    a2 = sigmoid(z2)

    error = y - a2

    d2 = error \* sigmoid\_derivative(a2)

    d1 = np.dot(d2, W2.T) \* sigmoid\_derivative(a1)

    W2 += np.dot(a1.T, d2) \* learning\_rate

    b2 += np.sum(d2, axis=0, keepdims=True) \* learning\_rate

    W1 += np.dot(X.T, d1) \* learning\_rate

    b1 += np.sum(d1, axis=0, keepdims=True) \* learning\_rate

    if epoch % 2000 == 0:

        loss = np.mean(np.square(error))

        print(f"Epoch {epoch}, Loss: {loss:.4f}")

print("\nPredictions after training:")

print(a2.round(3))

**OUTPUT 8**

Epoch 0, Loss: 0.2629

Epoch 2000, Loss: 0.1132

Epoch 4000, Loss: 0.0097

Epoch 6000, Loss: 0.0041

Epoch 8000, Loss: 0.0025

Predictions after training:

[[0.028]

[0.956]

[0.958]

[0.052]]

**Q9)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Consider the data sets**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **ID** | **SepalLength** | **SepalWidth** | **PetalLength** | **PetalWidth** | | **1** | **5.1** | **3.5** | **1.4** | **0.2** | | **2** | **4.9** | **3.0** | **1.4** | **0.2** | | **3** | **4.7** | **3.2** | **1.3** | **0.2** | | **4** | **5.0** | **3.6** | **1.4** | **0.2** | | **5** | **7.0** | **3.2** | **4.7** | **1.4** | | **6** | **6.4** | **3.2** | **4.5** | **1.5** | | **7** | **6.9** | **3.1** | **4.9** | **1.5** | | **8** | **6.3** | **3.3** | **6.0** | **2.5** | | **9** | **5.8** | **2.7** | **5.1** | **1.9** | | **10** | **7.1** | **3.0** | **5.9** | **2.1** | |

**ANSWER 9**

import pandas as pd

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import numpy as np

# Dataset from the image

data = {

    'ID': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

    'SepalLength': [5.1, 4.9, 4.7, 5.0, 7.0, 6.4, 6.9, 6.3, 5.8, 7.1],

    'SepalWidth': [3.5, 3.0, 3.2, 3.6, 3.2, 3.2, 3.1, 3.3, 2.7, 3.0],

    'PetalLength': [1.4, 1.4, 1.3, 1.4, 4.7, 4.5, 4.9, 6.0, 5.1, 5.9],

    'PetalWidth': [0.2, 0.2, 0.2, 0.2, 1.4, 1.5, 1.5, 2.5, 1.9, 2.1]

}

df = pd.DataFrame(data)

X = df[['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth']]

kmeans = KMeans(n\_clusters=3, random\_state=42, n\_init=10) # Added random\_state and n\_init for reproducibility

df['Cluster'] = kmeans.fit\_predict(X)

print("Flower ID and Assigned Cluster:")

print(df[['ID', 'Cluster']])

print("-" \* 30)

print("\nCluster Centroids (Mean of Features for Each Cluster):\n", kmeans.cluster\_centers\_)

print("-" \* 30)

plt.figure(figsize=(8, 6))

scatter = plt.scatter(df['PetalLength'], df['PetalWidth'], c=df['Cluster'], cmap='viridis', s=100)

plt.xlabel('PetalLength')

plt.ylabel('PetalWidth')

plt.title('K-Means Clustering of Flowers (Petal Features)')

#plt.colorbar(scatter, label='Cluster Label')

plt.grid(True)

plt.show()

**OUTPUT 9**

Flower ID and Assigned Cluster:

ID Cluster

0 1 0

1 2 0

2 3 0

3 4 0

4 5 1

5 6 1

6 7 1

7 8 2

8 9 1

9 10 2

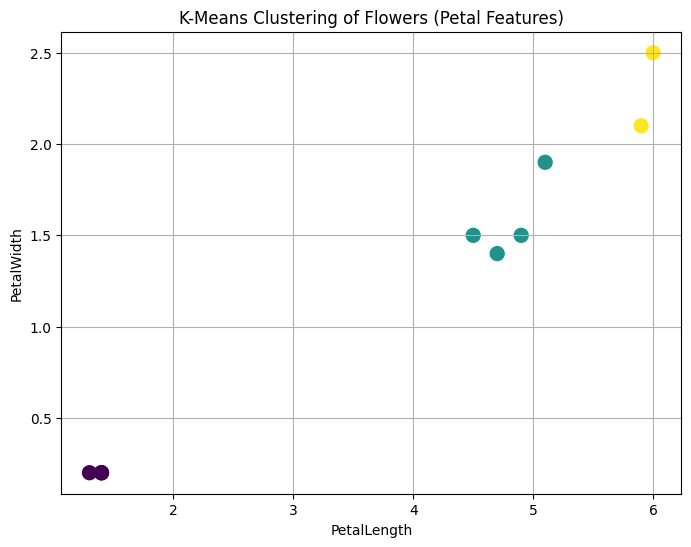
------------------------------

Cluster Centroids (Mean of Features for Each Cluster):

[[4.925 3.325 1.375 0.2 ]

[6.525 3.05 4.8 1.575]

[6.7 3.15 5.95 2.3 ]]



**Q10)**

Train SVM with linear and polynomial kernels.Classify fruits into Apple, Orange, or Banana based on Weight and Color\_Score.

|  |  |  |
| --- | --- | --- |
| Weight | Color\_Score | Fruit |
| 150 | 0.80 | Apple |
| 170 | 0.65 | Orange |
| 120 | 0.90 | Banana |
| 180 | 0.60 | Orange |
| 140 | 0.85 | Apple |
| 130 | 0.88 | Banana |
| 200 | 0.55 | Orange |

**ANSWER 10**

import pandas as pd

from sklearn.svm import SVC

import matplotlib.pyplot as plt

import numpy as np

# Dataset

data = {

    'Weight': [150, 170, 120, 180, 140, 130, 200],

    'Color\_Score': [0.80, 0.65, 0.90, 0.60, 0.85, 0.88, 0.55],

    'Fruit': ['Apple', 'Orange', 'Banana', 'Orange', 'Apple', 'Banana', 'Orange']

}

df = pd.DataFrame(data)

# Mapping the 'Fruit' column to numerical labels for the existing structure to run

fruit\_mapping = {'Apple': 0, 'Orange': 1, 'Banana': 2}

df['Fruit\_Encoded'] = df['Fruit'].map(fruit\_mapping)

df['Result'] = df['Fruit'].map({'Apple':0, 'Orange':1, 'Banana':1}) # Treat Banana as 1 for binary plot compatibility

X = df[['Weight', 'Color\_Score']] # Use the new feature names

y = df['Result'] # Use the new encoded target

for kernelName in ['linear','poly']:

    svm\_model = SVC(kernel=kernelName, C=1.0)

    svm\_model.fit(X, y)

    plt.figure() # Added to create a new figure for each kernel

    plt.scatter(X['Weight'], X['Color\_Score'], c=y, cmap='bwr', s=100) # Use new features for scatter plot

    if kernelName == 'linear': # ONLY calculate and plot the linear boundary for the 'linear' kernel

        w = svm\_model.coef\_[0]

        b = svm\_model.intercept\_[0]

        x\_vals = np.array([X['Weight'].min()-10, X['Weight'].max()+10]) # Adjusted range for Weight

        y\_vals = -(w[0]/w[1])\*x\_vals - b/w[1]

        plt.plot(x\_vals, y\_vals, 'k--')

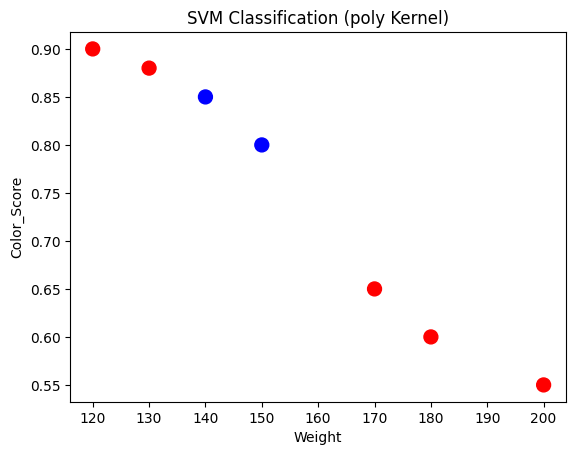
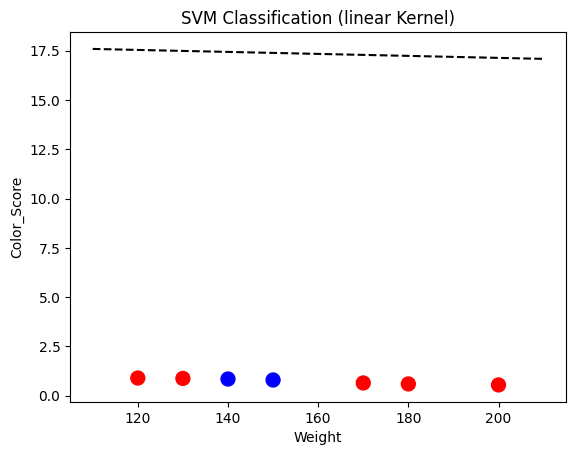
    plt.xlabel('Weight') # Change x-label

    plt.ylabel('Color\_Score') # Change y-label

    plt.title(f'SVM Classification ({kernelName} Kernel)') # Modified title to include kernel name

    plt.show()

**OUTPUT 10**



**Q11)**

Implement K-Means clustering on the customer dataset with features: Age, Annual Income, Spending Score, and Membership Level. Visualize the clusters and interpret each group’s characteristics.

|  |  |  |  |
| --- | --- | --- | --- |
| Age | Income (₹k) | Spending Score | Membership (1–5) |
| 25 | 50 | 60 | 3 |
| 34 | 70 | 50 | 2 |
| 22 | 45 | 80 | 4 |
| 28 | 55 | 65 | 3 |
| 40 | 85 | 30 | 1 |
| 35 | 75 | 35 | 2 |
| 30 | 60 | 60 | 3 |
| 26 | 48 | 75 | 4 |
| 45 | 95 | 20 | 1 |
| 23 | 42 | 82 | 4 |

**ANSWER 11**

import pandas as pd

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

# ---------------------------------------------

# Dataset

# ---------------------------------------------

data = {

    'Age': [25, 34, 22, 28, 40, 35, 30, 26, 45, 23],

    'Income(₹k)': [50, 70, 45, 55, 85, 75, 60, 48, 95, 42],

    'SpendingScore': [60, 50, 80, 65, 30, 35, 60, 75, 20, 82],

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

}

df = pd.DataFrame(data)

# ---------------------------------------------

# K-Means Clustering (3 Clusters)

# ---------------------------------------------

kmeans = KMeans(n\_clusters=3, random\_state=0)

df['Cluster'] = kmeans.fit\_predict(df)

# ---------------------------------------------

# Display Cluster Results and Centroids

# ---------------------------------------------

print("===== K-Means Clustering Results =====")

print(df)

print("\nCentroids:\n", kmeans.cluster\_centers\_)

print("======================================\n")

# ---------------------------------------------

# Visualization (Income vs Spending Score)

# ---------------------------------------------

plt.scatter(df['Income(₹k)'], df['SpendingScore'],

            c=df['Cluster'], cmap='viridis', s=100)

plt.xlabel('Income (₹k)')

plt.ylabel('Spending Score')

plt.title('K-Means Clustering of Customers')

plt.show()

**OUTPUT 11**

===== K-Means Clustering Results =====

Age Income(₹k) SpendingScore Membership Cluster

0 25 50 60 3 1

1 34 70 50 2 0

2 22 45 80 4 1

3 28 55 65 3 1

4 40 85 30 1 0

5 35 75 35 2 0

6 30 60 60 3 1

7 26 48 75 4 1

8 45 95 20 1 2

9 23 42 82 4 1

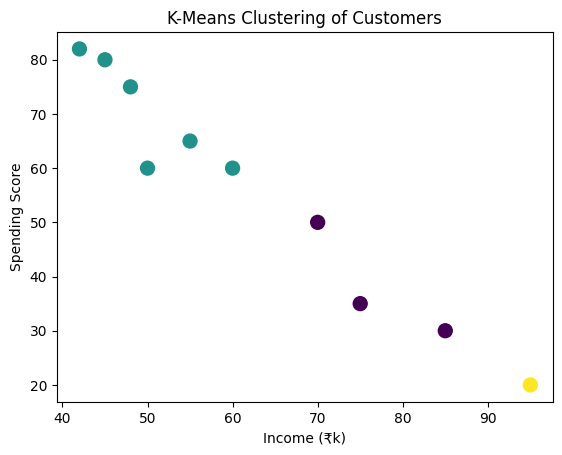
Centroids:

[[36.33333333 76.66666667 38.33333333 1.66666667]

[25.66666667 50. 70.33333333 3.5 ]

[45. 95. 20. 1. ]]

======================================



**Q12)**



1)Choose logistic regression for the above data set and apply 5 fold cross validation to evaluate models performance

1)Choose decision tree for the above data set and apply 3 fold cross validation to evaluate models performance

**ANSWER 12**

**1)**

import pandas as pd

import numpy as np

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import cross\_val\_score

from sklearn.linear\_model import LogisticRegression

# Dataset

data = {

    'CustomerID': [301, 302, 303, 304, 305, 306, 307, 308, 309, 310],

    'Age': [25, 40, 30, 50, 35, 28, 45, 32, 38, 27],

    'Balance(₹K)': [50, 200, 150, 300, 120, 80, 250, 100, 180, 60],

    'Loan': ['No', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'No'],

    'CreditCard': ['Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No']

}

df = pd.DataFrame(data)

# Encode categorical data

label\_enc = LabelEncoder()

df['Loan'] = label\_enc.fit\_transform(df['Loan'])           # Yes=1, No=0

df['CreditCard'] = label\_enc.fit\_transform(df['CreditCard'])  # Yes=1, No=0

# Define features and target

X = df[['Age', 'Balance(₹K)', 'Loan']]

y = df['CreditCard']

# Logistic Regression with 5-fold CV

log\_reg = LogisticRegression()

log\_reg\_scores = cross\_val\_score(log\_reg, X, y, cv=5, scoring='accuracy')

print("===== Logistic Regression (5-Fold Cross Validation) =====")

print("Fold Accuracies:", log\_reg\_scores)

print("Average Accuracy: {:.2f}".format(np.mean(log\_reg\_scores)))

**OUTPUT 12-1)**

/usr/local/lib/python3.12/dist-packages/sklearn/model\_selection/\_split.py:805: UserWarning: The least populated class in y has only 4 members, which is less than n\_splits=5.

warnings.warn(

===== Logistic Regression (5-Fold Cross Validation) =====

Fold Accuracies: [0.5 0.5 0. 0.5 1. ]

Average Accuracy: 0.50

**2)**

from sklearn.model\_selection import KFold

from sklearn.tree import DecisionTreeClassifier

from sklearn.datasets import load\_iris

import numpy as np

from matplotlib import pyplot as plt

from sklearn import tree

data = load\_iris()

X, y = data.data, data.target

max\_depth = 4

dtree = DecisionTreeClassifier(max\_depth=max\_depth, random\_state=42)

kf = KFold(n\_splits=5, shuffle=False)

kf\_scores = []

for train\_index, test\_index in kf.split(X):

    X\_train, X\_test = X[train\_index], X[test\_index]

    y\_train, y\_test = y[train\_index], y[test\_index]

    dtree.fit(X\_train, y\_train)

    accuracy = dtree.score(X\_test, y\_test)

    kf\_scores.append(accuracy)

    fig = plt.figure()

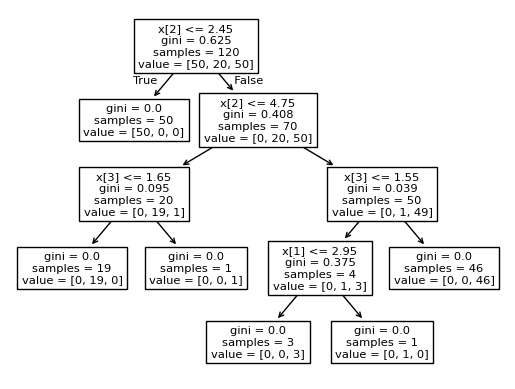
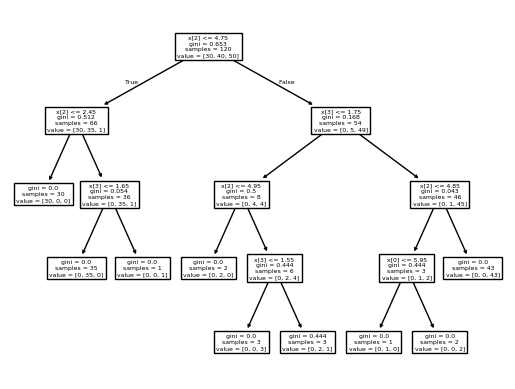
    tree.plot\_tree(dtree)

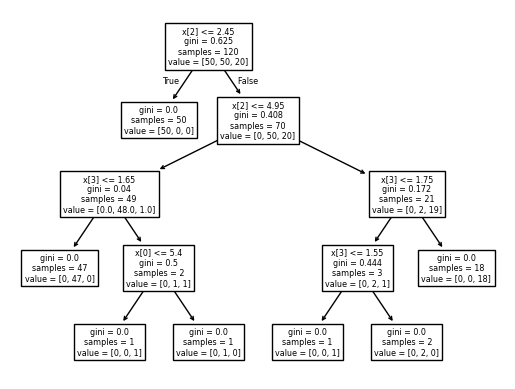
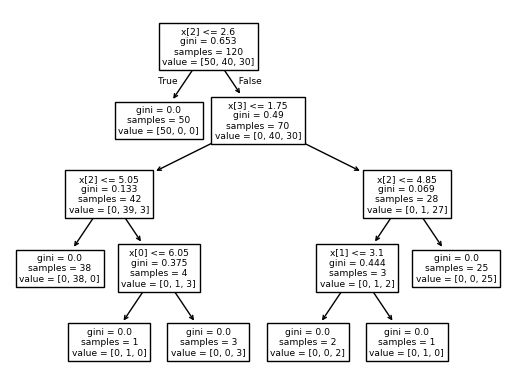
    plt.show()

print("K-Fold Accuracy Scores:", kf\_scores)

print("Mean K-Fold Accuracy:", np.mean(kf\_scores))

**OUTPUT 12-2)**

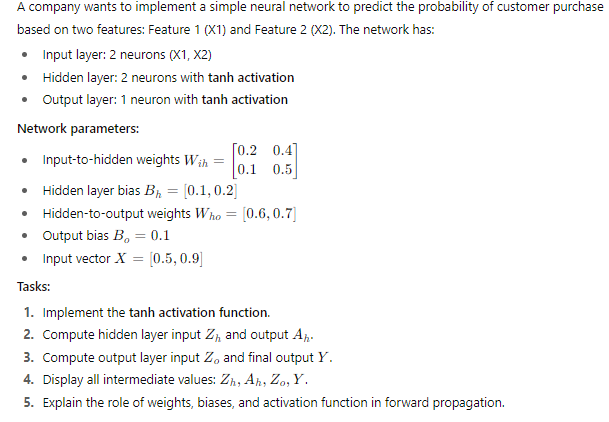




**K-Fold Accuracy Scores: [1.0, 1.0, 0.8333333333333334, 0.9333333333333333, 0.8]**

**Mean K-Fold Accuracy: 0.9133333333333333**

**Q14)**



**ANSWER 14**

import numpy as np

# ---------------------------------------------

# 1. Define tanh activation function

# ---------------------------------------------

def tanh(z):

    return np.tanh(z)

# ---------------------------------------------

# 2. Given Parameters

# ---------------------------------------------

X = np.array([0.8, 0.6])

# Input-to-hidden weights and bias

W\_ih = np.array([[0.1, 0.3],

                 [0.2, 0.4]])

B\_h = np.array([0.05, 0.15])

# Hidden-to-output weights and bias

W\_ho = np.array([[0.5],

                 [0.6]])

B\_o = np.array([0.2])

# ---------------------------------------------

# 3. Hidden layer computation

# ---------------------------------------------

Z\_h = np.dot(X, W\_ih) + B\_h     # Weighted sum for hidden layer

A\_h = tanh(Z\_h)                 # Apply tanh activation

# ---------------------------------------------

# 4. Output layer computation

# ---------------------------------------------

Z\_o = np.dot(A\_h, W\_ho) + B\_o   # Weighted sum for output layer

Y = tanh(Z\_o)                   # Final output with tanh activation

# ---------------------------------------------

# 5. Display all intermediate values

# ---------------------------------------------

print("===== Neural Network with tanh Activation =====")

print(f"Input Vector (X): {X}")

print(f"Hidden Layer Weighted Sum (Z\_h): {Z\_h}")

print(f"Hidden Layer Output (A\_h): {A\_h}")

print(f"Output Layer Weighted Sum (Z\_o): {Z\_o}")

print(f"Final Output (Y): {Y}")

print("================================================")

**OUTPUT 14**

===== Neural Network with tanh Activation =====

Input Vector (X): [0.8 0.6]

Hidden Layer Weighted Sum (Z\_h): [0.25 0.63]

Hidden Layer Output (A\_h): [0.24491866 0.55805222]

Output Layer Weighted Sum (Z\_o): [0.65729066]

Final Output (Y): [0.57655753]