Decision Tree

Code:-

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn import tree as sk\_tree

# Step 1: Parse the dataset

data = {

'Age': ['<=30', '<=30', '31-40', '>40', '>40', '>40', '31-40', '<=30', '<=30', '>40', '<=30', '31-40', '31-40', '>40'],

'Income': ['High', 'High', 'High', 'Medium', 'Low', 'Low', 'Low', 'Medium', 'Low', 'Medium', 'Medium', 'Medium', 'High', 'Medium'],

'Student': ['No', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No'],

'Credit Rating': ['Fair', 'Excellent', 'Fair', 'Fair', 'Fair', 'Excellent', 'Excellent', 'Fair', 'Fair', 'Fair', 'Excellent', 'Excellent', 'Fair', 'Excellent'],

'Buys Computer': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

}

df = pd.DataFrame(data)

# Encode the categorical variables

df\_encoded = df.apply(lambda x: pd.factorize(x)[0])

# Fit the decision tree classifier using Gini impurity

clf\_gini = sk\_tree.DecisionTreeClassifier(criterion='gini')

clf\_gini = clf\_gini.fit(df\_encoded.iloc[:, :-1], df\_encoded['Buys Computer'])

# Convert the feature names from Index to list

feature\_names = df.columns[:-1].tolist()

# Convert the class names to a list

class\_names = df['Buys Computer'].unique().tolist()

# Plot the decision tree

plt.figure(figsize=(20,10))

sk\_tree.plot\_tree(clf\_gini, feature\_names=feature\_names, class\_names=class\_names, filled=True)

plt.show()

# Plot the decision tree

plt.figure(figsize=(20,10))

sk\_tree.plot\_tree(clf\_gini, feature\_names=feature\_names, class\_names=class\_names, filled=True)

plt.show()

# Function to print Gini impurity and chosen attribute at each split

def print\_gini\_and\_splits(tree, feature\_names):

tree\_ = tree.tree\_

feature\_name = [

feature\_names[i] if i != sk\_tree.\_tree.TREE\_UNDEFINED else "undefined!"

for i in tree\_.feature

]

print("Decision tree splits and Gini impurities:")

for i in range(tree\_.node\_count):

if tree\_.children\_left[i] != sk\_tree.\_tree.TREE\_LEAF:

print(f"Node {i} (Gini: {tree\_.impurity[i]:.4f}): split on feature '{feature\_name[i]}'")

else:

print(f"Node {i} (Gini: {tree\_.impurity[i]:.4f}): leaf node")

print\_gini\_and\_splits(clf\_gini, feature\_names)

# Example test sample

test\_sample = {

'Age': '<=30',

'Income': 'Medium',

'Student': 'Yes',

'Credit Rating': 'Fair'

}

# Encode the test sample

encoded\_sample = pd.DataFrame([test\_sample]).apply(lambda x: pd.factorize(df[x.name])[0][df[x.name].tolist().index(x[0])])

# Predict using sklearn decision tree

sklearn\_prediction = clf\_gini.predict([encoded\_sample])

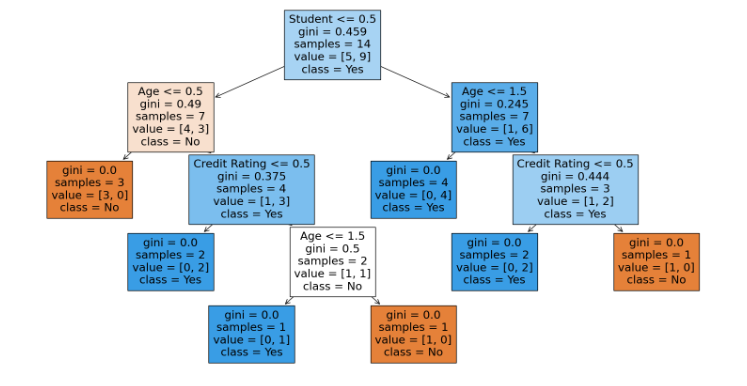
decoded\_prediction = pd.factorize(df['Buys Computer'])[1][sklearn\_prediction[0]]

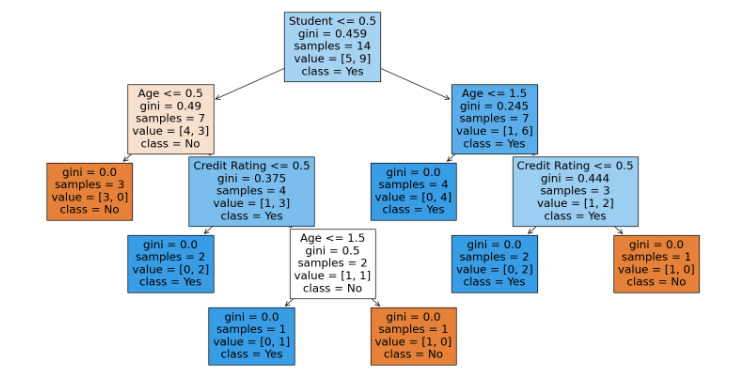
print("Prediction for sklearn decision tree:", decoded\_prediction)

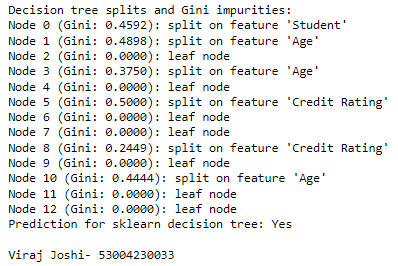
print()

print('Viraj Joshi- 53004230033')

Output:-







KNN

Code:-

# Step 1: Import necessary libraries

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from mpl\_toolkits.mplot3d import Axes3D

# Step 2: Load and display the sample data

data = {

'Age': [19, 21, 20, 23, 31, 22, 35, 25, 23, 64, 30, 67, 35, 58, 24],

'Annual Income (k$)': [15, 15, 16, 16, 17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22],

'Spending Score (1-100)': [39, 81, 6, 77, 40, 76, 6, 94, 3, 72, 79, 65, 76, 76, 94],

'Segment': [0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1] # 0: Low-value, 1: High-value

}

df = pd.DataFrame(data)

print("Sample Data:")

print(df.head())

# Step 3: Data Preprocessing

X = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]

y = df['Segment']

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Step 4: Train-Test Split

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

# Step 5: Apply KNN Algorithm

knn = KNeighborsClassifier(n\_neighbors=3)

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

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

# Step 6: Evaluation

print("\nConfusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

print("\nAccuracy Score:")

print(accuracy\_score(y\_test, y\_pred))

# Step 7: Classify new user input

new\_user\_data = {'Age': [27], 'Annual Income (k$)': [23], 'Spending Score (1-100)': [60]}

new\_user\_df = pd.DataFrame(new\_user\_data)

new\_user\_scaled = scaler.transform(new\_user\_df)

new\_user\_segment = knn.predict(new\_user\_scaled)

new\_user\_df['Segment'] = new\_user\_segment

print("\nNew User Data Prediction:")

print(new\_user\_df)

# Visualization: Scatter plot of the customer segments

plt.figure(figsize=(10, 6))

sns.scatterplot(x='Annual Income (k$)', y='Spending Score (1-100)', hue='Segment', data=df, palette='Set1', marker='o', label='Existing Data')

sns.scatterplot(x='Annual Income (k$)', y='Spending Score (1-100)', hue='Segment', data=new\_user\_df, palette='Set2', marker='X', s=200, label='New User Data')

plt.title('Customer Segments with New User Input')

plt.xlabel('Annual Income (k$)')

plt.ylabel('Spending Score (1-100)')

plt.legend()

plt.show()

# Visualization: 3D plot for KNN decision boundaries and customer segments including new user input

fig = plt.figure(figsize=(10, 6))

ax = fig.add\_subplot(111, projection='3d')

# Plot the existing data with original values

ax.scatter(X['Age'], X['Annual Income (k$)'], X['Spending Score (1-100)'], c=y, cmap='Set1', s=50, label='Existing Data')

# Plot the new user input with original values

ax.scatter(new\_user\_df['Age'], new\_user\_df['Annual Income (k$)'], new\_user\_df['Spending Score (1-100)'], c='green', marker='X', s=200, label='New User Data')

ax.set\_xlabel('Age')

ax.set\_ylabel('Annual Income (k$)')

ax.set\_zlabel('Spending Score (1-100)')

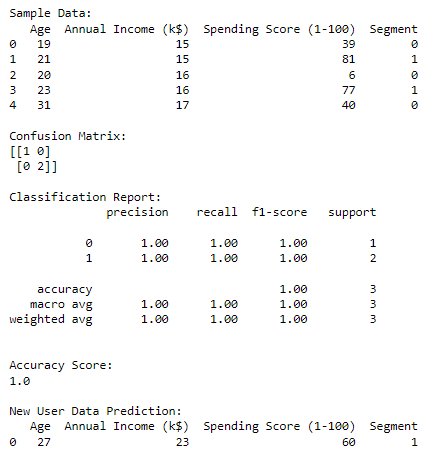
plt.title('3D Plot of Customer Segments with New User Input')

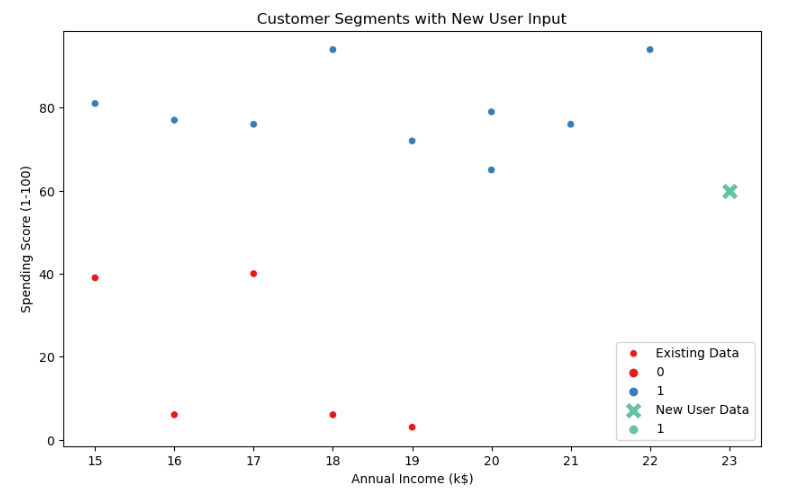
ax.legend()

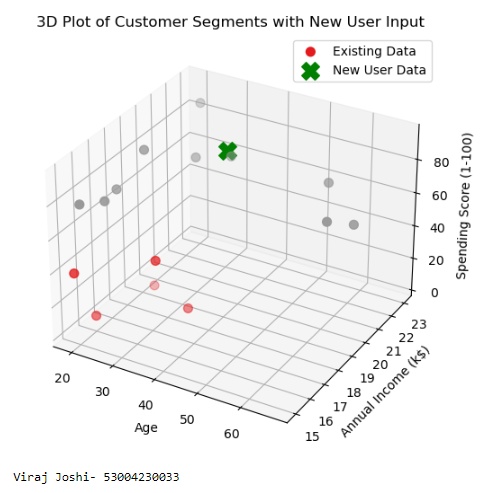
plt.show()

print('Viraj Joshi- 53004230033')

Output:-







Naïve Bayes

Code:-

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values

y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Fitting classifier to the Training set

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

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

# Predicting the Test set results

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

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Training set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Naive Bayes (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

print('Viraj Joshi-53004230033')

plt.show()

# Visualising the Test set results

from matplotlib.colors import ListedColormap

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red', 'green')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label = j)

plt.title('Naive Bayes (Test set)')

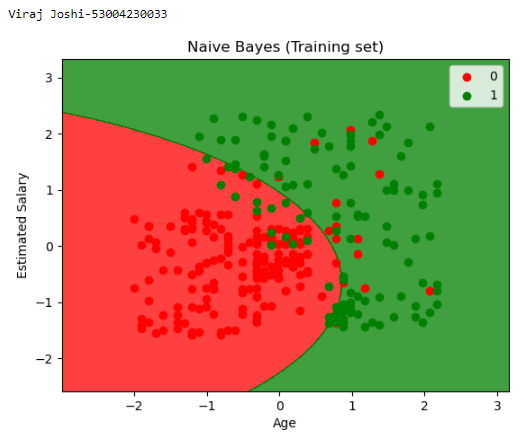
plt.xlabel('Age')

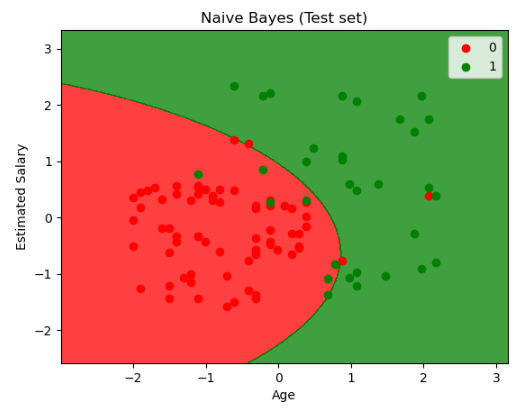
plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

Output:-





Hierarchical Clustering

Code:-

import pandas as pd

import numpy as np

from sklearn.cluster import AgglomerativeClustering

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.datasets import load\_iris

import matplotlib.pyplot as plt

from scipy.cluster.hierarchy import dendrogram, linkage

#Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

#Step 1: Hierarchical Clustering with different Linkage Methods and Draw denograms

n\_clusters = 3 # Number of clusters

linkage\_methods = ['ward', 'single', 'complete'] # Different Linkage methods

cluster\_labels = []

#Define figure and axes for dendrograms

plt.figure(figsize=(15, 5))

dendrogram\_axes = []

for i, linkage\_method in enumerate(linkage\_methods):

labels = AgglomerativeClustering(n\_clusters=n\_clusters, linkage=linkage\_method).fit\_predict(X)

cluster\_labels.append(labels)

#Create a dendrgram for the current linkage method

dendrogram\_data = linkage(X, method=linkage\_method)

dendrogram\_axes.append(plt.subplot(1, len(linkage\_methods), i+1))

dendrogram(dendrogram\_data, orientation='top', labels=labels)

plt.title(f"{linkage\_method.capitalize()} Linkage Dendrogram")

plt.xlabel('Samples')

plt.ylabel('Distance')

#Plot clustering results for different linkage methods

plt.figure(figsize=(15, 5))

for i, linkage\_method in enumerate(linkage\_methods):

plt.subplot(1, len(linkage\_methods), i + 1)

scatter = plt.scatter(X[:, 0], X[:, 1], c=cluster\_labels[i], cmap='viridis',

label=f'Clusters ({linkage\_method.capitalize()} Linkage)')

plt.title(f"{linkage\_method.capitalize()} Linkage")

#Add legend to scatter plots

plt.legend(handles=scatter.legend\_elements()[0], labels=[f'Cluster {i}' for i in range(n\_clusters)])

#sTEP 2 :fEATURE ENGINEERING (uSING CLUSTER ASSIGNMENT AS A feature)

X\_with\_cluster = np.column\_stack((X, cluster\_labels[-1])) # using complete linkage

#Step 3: Classification

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

classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

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

#Step 4: Prediction

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

#Step 5 : Test Score and Confusion Matrix

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

#Genrate classification report with zero\_division parametrs

classification\_rep = classification\_report(y\_test, y\_pred, zero\_division=0)

#Print cluster description

cluster\_descriptions = {

'ward': 'Clusters based on Ward linkage interpretation.',

'single': 'Cluster based on Single linkage interpretation.',

'complete': 'Clusters based on Complete linkage interpretation.'

}

for method in linkage\_methods:

print(f"Cluster Descriptions ({method.capitalize()} Linkage):")

print(cluster\_descriptions[method.lower()]) # Convert to lowercase for dictionary access

# Print accuracy, confusion matrix, and classification report

print("Accuracy:", accuracy)

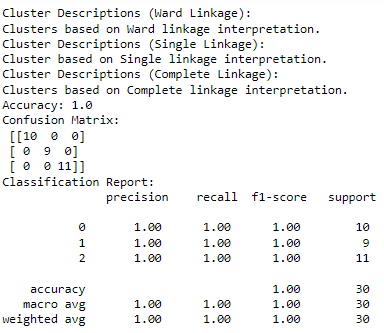
print("Confusion Matrix:\n", conf\_matrix)

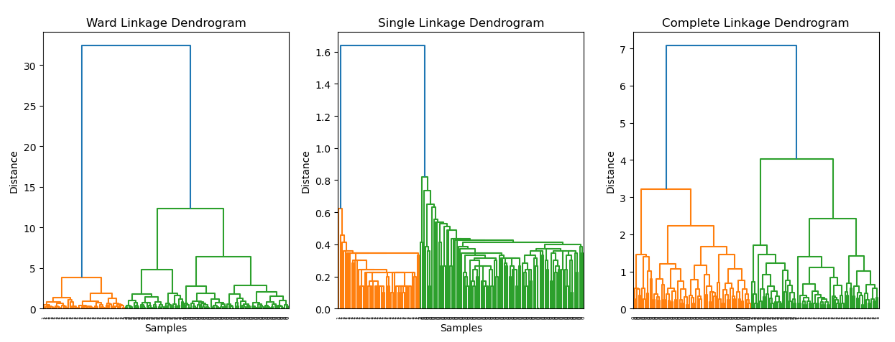
print("Classification Report:\n", classification\_rep)

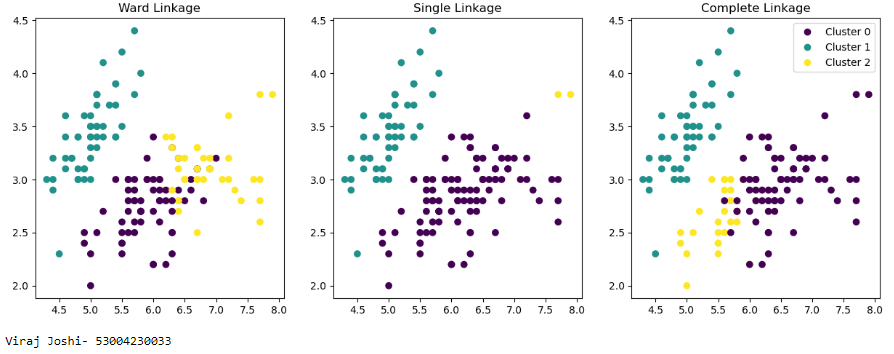
plt.show()

print("Viraj Joshi- 53004230033")

Output:-







KMeans Clustering

Code:-

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

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

from sklearn.cluster import KMeans

from sklearn.metrics import classification\_report, confusion\_matrix

#Load the Iris dataset

iris = load\_iris()

X = iris.data[:, :2] #Select only the features (sepal lengthy and sepal width)

y = iris.target

#Split database into traini9ng and testing

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

#Initalize K-Means clustering with the number of clusters equal to the number of classes

n\_clusters = len(np.unique(y))

kmeans = KMeans(n\_clusters=n\_clusters, random\_state=42)

#Fit K-Means clustering to the training data

kmeans.fit(X\_train)

#Assign cluster labels to data points in test set

cluster\_labels = kmeans.predict(X\_test)

#Assign class labels to clusters based on thge most frequent class label in each cluster

cluster\_class\_labels = []

for i in range(n\_clusters):

cluster\_indices = np.where(cluster\_labels ==i)[0]

cluster\_class\_labels.append(np.bincount(y\_test[cluster\_indices]).argmax())

#Assign cluster class labels to data points in the test set

y\_pred = np.array([cluster\_class\_labels[cluster\_labels[i]] for i in range(len(X\_test))])

#Evaluate the classifier's performance

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

#Visualize the dataset and cluster cemters

plt.figure(figsize=(10, 6))

#Plot the training data points

plt.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap='viridis', label='Training Data')

#Plot testing data

plt.scatter(X\_test[:, 0], X\_test[:, 1], c=y\_test, cmap='viridis', marker='x', s=100, label='Testing Data')

#plt cluster centers

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], c='red', marker='o', s=100, label='Cluster Centers')

plt.xlabel('Sepal Length (cm)')

plt.ylabel('Sepal Width (cm)')

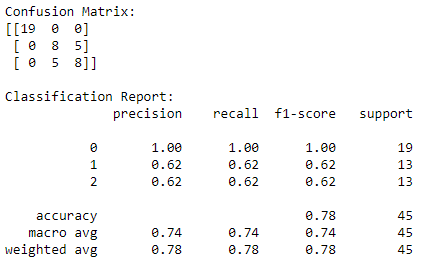
plt.title('K-Means Clustering with Class Labels on Iris Dataset')

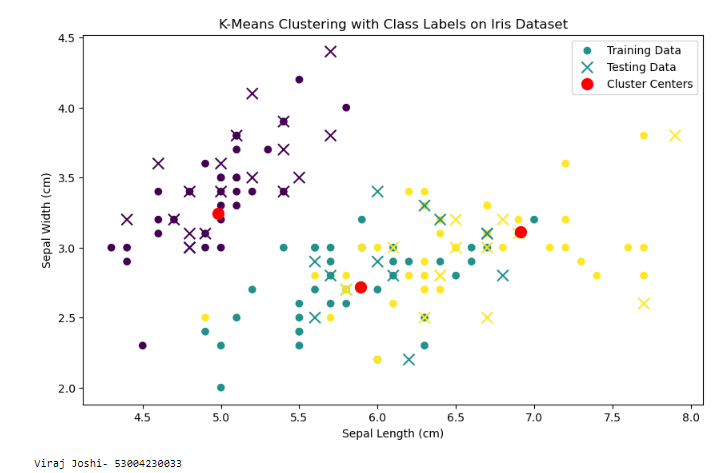
plt.legend()

plt.show()

print("Viraj Joshi- 53004230033")

Output:-





Linear Regression

Code:-

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.datasets import fetch\_california\_housing

housing = fetch\_california\_housing()

X = pd.DataFrame(housing.data, columns = housing.feature\_names)

y = pd.DataFrame(housing.target, columns = ['MEDV'])

plt.figure(figsize=(10,8))

sns.heatmap(X.corr(), annot=True, cmap='coolwarm')

plt.title("Correlation Heatmap of California Housing Features")

plt.show()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

reg\_model = LinearRegression()

reg\_model.fit(X\_train, y\_train)

y\_train\_pred = reg\_model.predict(X\_train)

y\_test\_pred = reg\_model.predict(X\_test)

train\_mse = mean\_squared\_error(y\_train, y\_train\_pred)

test\_mse = mean\_squared\_error(y\_test, y\_test\_pred)

train\_r2 = r2\_score(y\_train, y\_train\_pred)

test\_r2 = r2\_score(y\_test, y\_test\_pred)

print(f'Training Mean Squared Error: {train\_mse}')

print(f'Test Mean Squared Error: {test\_mse}')

print(f'Training R^2 Score: {train\_r2}')

print(f'Test R^2 Score: {test\_r2}')

coefficients = pd.DataFrame(reg\_model.coef\_.T, X.columns, columns=['Coefficients'])

print(coefficients)

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

plt.scatter(y\_test, y\_test\_pred, c='blue')

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], '--r', lw =3)

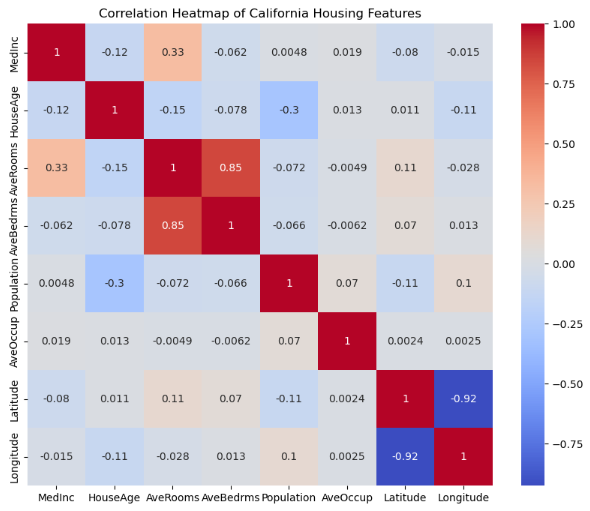
plt.xlabel('Actual Value')

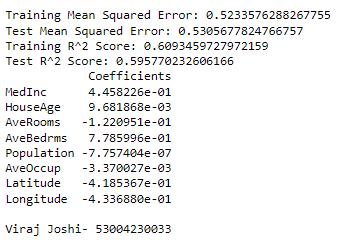
plt.ylabel('Predicted Value')

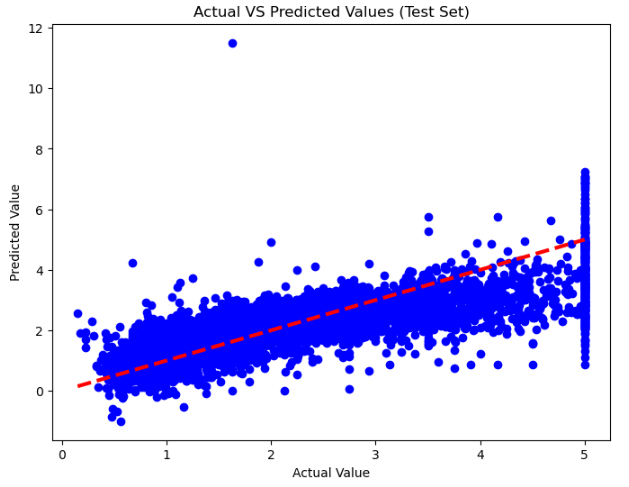
plt.title('Actual VS Predicted Values (Test Set)')

print('Viraj Joshi- 53004230033')

Output:-







ANN-Backpropogation

Code:-

import numpy as np

import matplotlib.pyplot as plt

def sigmoid(x):

return 1/(1+np.exp(-x))

def sigmoid\_derivative(x):

return x\*(1-x)

class NeuralNetwork:

def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):

self.weights\_input\_hidden = np.random.uniform(size=(input\_size, hidden\_size))

self.weights\_hidden\_output = np.random.uniform(size=(hidden\_size, output\_size))

def forward(self, X):

self.hidden\_input = np.dot(X, self.weights\_input\_hidden)

self.hidden\_output = sigmoid(self.hidden\_input)

self.output = sigmoid(np.dot(self.hidden\_output, self.weights\_hidden\_output))

return self.output

def backward(self, X, y, learning\_rate):

error\_output = y-self.output

delta\_output = error\_output\*sigmoid\_derivative(self.output)

error\_hidden = delta\_output.dot(self.weights\_hidden\_output.T)

delta\_hidden = error\_hidden\*sigmoid\_derivative(self.hidden\_output)

self.weights\_hidden\_output += self.hidden\_output.T.dot(delta\_output)\*learning\_rate

self.weights\_input\_hidden += X.T.dot(delta\_hidden)\*learning\_rate

def train(self, X, y, learning\_rate, epochs):

self.loss\_history = []

for \_ in range(epochs):

output = self.forward(X)

error = y-output

self.loss\_history.append(np.mean(error\*\*2))

self.backward(X,y,learning\_rate)

def predict(self, X):

return self.forward(X)

X = np.array([[0,0],[0,1],[1,0],[1,1]])

y = np.array([[0], [1], [1], [0]])

input\_size = 2

hidden\_size = 4

output\_size = 1

learning\_rate = 0.1

epochs = 10000

nn = NeuralNetwork(input\_size, hidden\_size, output\_size)

nn.train(X, y, learning\_rate, epochs)

predictions = nn.predict(X)

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

plt.scatter(X[:,0], X[:,1], c=y, cmap='viridis', label='XOR Data')

plt.scatter(X[:,0], X[:,1], c=np.round(predictions), cmap='plasma', marker='x', s=200, label='Predictions')

plt.title('XOR Dataset and Predictions')

plt.xlabel('Input 1')

plt.ylabel('Input 2')

plt.legend()

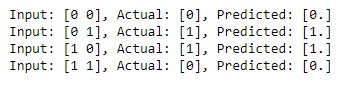
for i in range(len(X)):

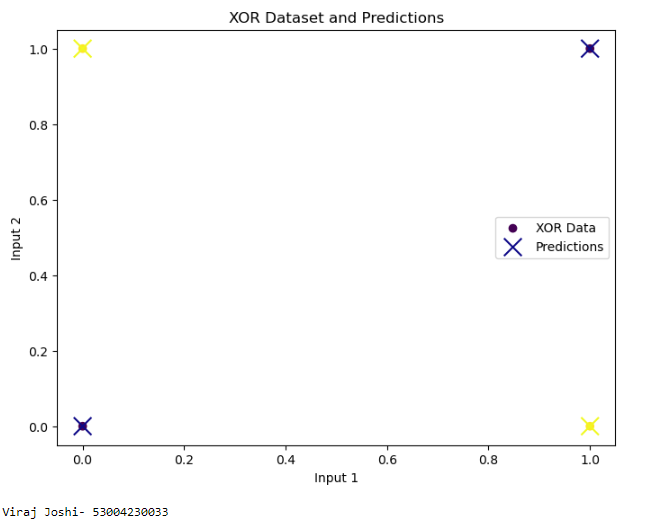
print(f"Input: {X[i]}, Actual: {y[i]}, Predicted: {np.round(predictions[i])}")

plt.show()

print('Viraj Joshi- 53004230033')

Output:-





Logistic Regression

Code:-

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.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.datasets import load\_breast\_cancer

cancer\_data = load\_breast\_cancer()

X = pd.DataFrame(cancer\_data.data, columns= cancer\_data.feature\_names)

y = pd.DataFrame(cancer\_data.target, columns= ['target'])

print('Dataset Head:')

print(X.head())

print('Target Distribution:')

print(y['target'].value\_counts())

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state = 42)

logreg = LogisticRegression(max\_iter=10000, random\_state=42)

logreg.fit(X\_train, y\_train.values.ravel())

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

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

print(f'Accuracy: {accuracy}')

print()

print('Confusion matrix:')

print(conf\_matrix)

print()

print('Classification report:')

print(class\_report)

plt.figure(figsize=(6,4))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix')

plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.show()

new\_input = np.array([X.mean().values])

print(f'New input for prediction: {new\_input}')

new\_prediction = logreg.predict(new\_input)

predicted\_class = 'benign' if new\_prediction == 1 else 'maligant'

print(f'Predicted class for the new input: {predicted\_class}')

plt.figure(figsize=(6,4))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion matrix - Test set')

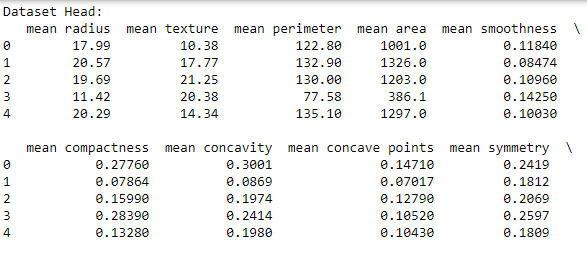
plt.xlabel('Predicted Label')

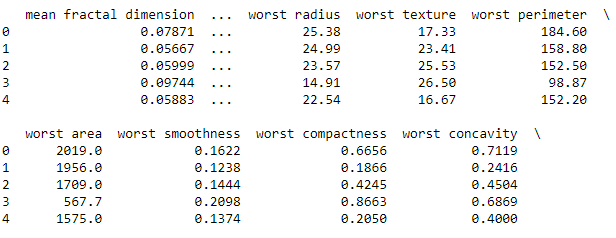
plt.ylabel('True Label')

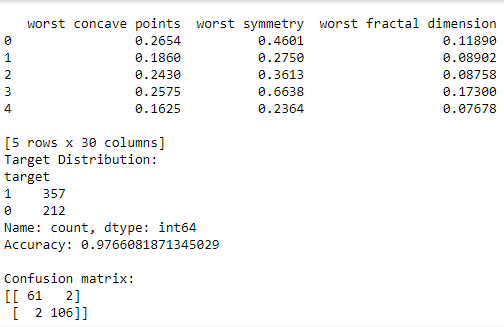
plt.show()

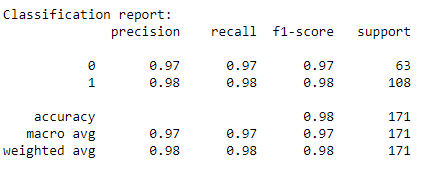
print('Viraj Joshi- 53004230033')

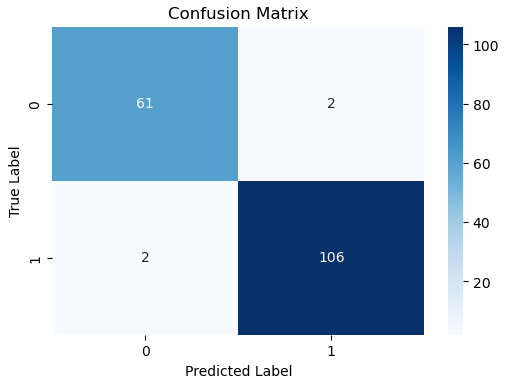
Output:-

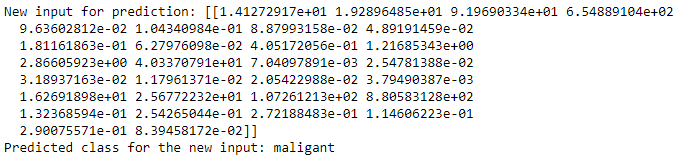


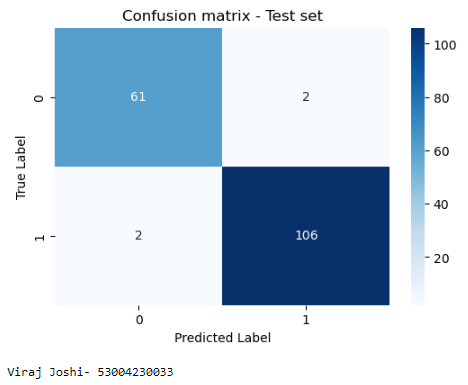












Random Forest

Code:-

import numpy as np

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

iris = load\_iris()

X = pd.DataFrame(iris.data, columns=iris.feature\_names).iloc[:,:2]

y = pd.DataFrame(iris.target, columns=['species'])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

def plot\_decision\_boundary(clf, X, y, title):

x\_min, x\_max = X.iloc[:,0].min()-1, X.iloc[:,0].max()+1

y\_min, y\_max = X.iloc[:,1].min()-1, X.iloc[:,1].max()+1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.01),

np.arange(y\_min, y\_max, 0.01))

Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contour(xx, yy, Z, alpha=0.4, cmap=plt.cm.RdYlBu)

plt.scatter(X.iloc[:,0], X.iloc[:,1], c=y.values.ravel(), s=40, edgecolor='k', cmap=plt.cm.RdYlBu)

plt.title(title)

plt.xlabel(iris.feature\_names[0])

plt.ylabel(iris.feature\_names[1])

plt.show()

dt\_model = DecisionTreeClassifier(random\_state=42)

dt\_model.fit(X\_train, y\_train)

dt\_predictions = dt\_model.predict(X\_test)

dt\_accuracy = accuracy\_score(y\_test, dt\_predictions)

dt\_confusion\_matrix = confusion\_matrix(y\_test, dt\_predictions)

print(f'Decision Tree Accuracy: {dt\_accuracy}')

print('Decision Tree Classification Report:')

print(classification\_report(y\_test, dt\_predictions))

sns.heatmap(dt\_confusion\_matrix, annot=True, fmt='d', cmap='Blues')

plt.title('DecisionTree Confusion Matrix')

plt.ylabel('True Label')

plt.xlabel('Predicted Label')

plt.show()

plot\_decision\_boundary(dt\_model, X\_test, y\_test, 'Decision Tree Decision Boundary')

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train.values.ravel())

rf\_predictions = rf\_model.predict(X\_test)

rf\_accuracy = accuracy\_score(y\_test, rf\_predictions)

rf\_confusion\_matrix = confusion\_matrix(y\_test, rf\_predictions)

print(f'Random Forest Accuracy: {rf\_accuracy}')

print('Random Forest Classification Report:')

print(classification\_report(y\_test, rf\_predictions))

sns.heatmap(rf\_confusion\_matrix, annot=True, fmt='d', cmap='Greens')

plt.title('Random Forest Confusion Matrix')

plt.ylabel('True Label')

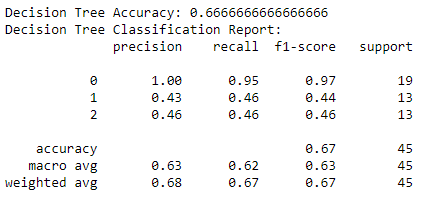
plt.xlabel('Predicted Label')

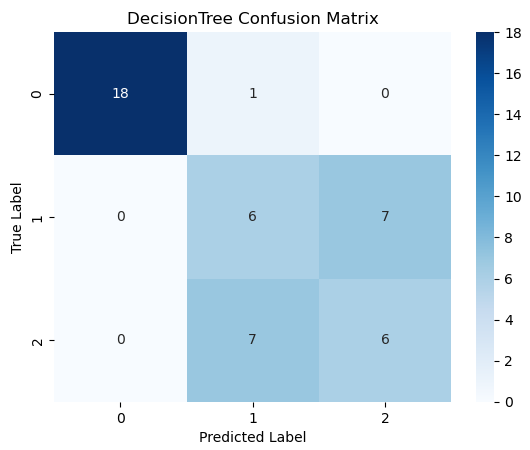
plt.show()

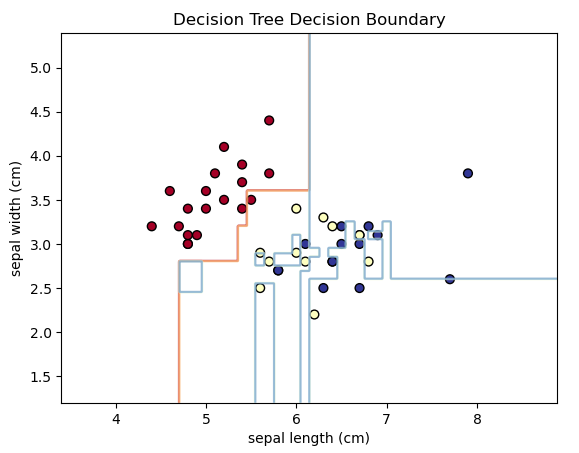
plot\_decision\_boundary(rf\_model, X\_test, y\_test, 'Random Forest Decision Boundary')

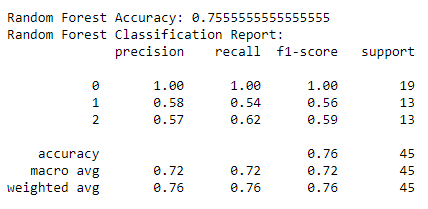
print('Viraj Joshi- 53004230033')

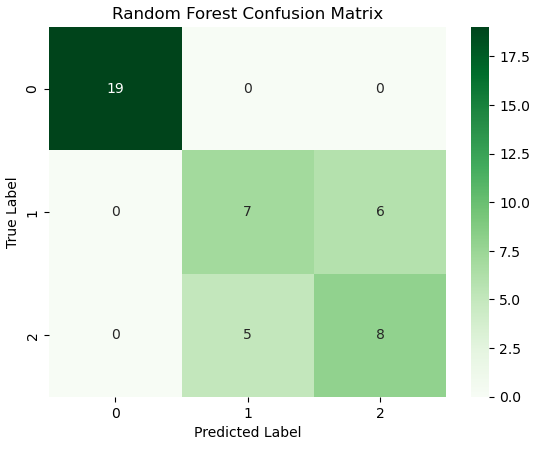
Output:-

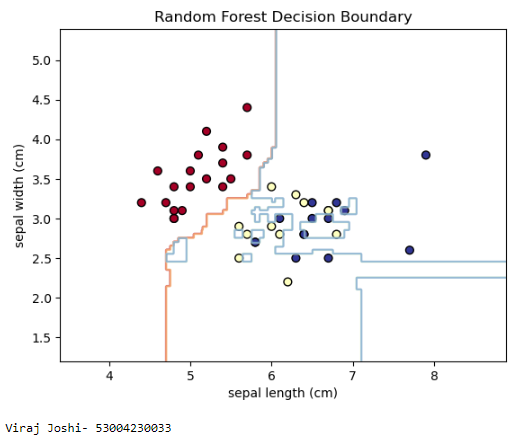












Locally Weighted Regression

Code:-

import numpy as np

import matplotlib.pyplot as plt

# Seed for reproducibility

np.random.seed(0)

# Generate random dataset

X = np.sort(5 \* np.random.rand(80, 1), axis=0)

y = np.sin(X).ravel()

y[::5] += 3 \* (0.5 - np.random.rand(16))

# Locally Weighted Regression function

def locally\_weighted\_regression(query\_point, X, y, tau=0.1):

m = X.shape[0]

# Calculate weights

weights = np.exp(-((X - query\_point) \* 2).sum(axis=1) / (2 \* tau \* 2))

W = np.diag(weights)

# Add bias term to X

X\_bias = np.c\_[np.ones((m, 1)), X]

# Calculate theta using weighted least squares

theta = np.linalg.inv(X\_bias.T.dot(W).dot(X\_bias)).dot(X\_bias.T).dot(W).dot(y)

# Predict for query\_point

x\_query = np.array([1, query\_point])

prediction = x\_query.dot(theta)

return prediction

# Generate test points

X\_test = np.linspace(0, 5, 100)

# Predict using locally weighted regression

predictions = [locally\_weighted\_regression(query\_point, X, y, tau=0.1) for query\_point in X\_test]

# Plot results

plt.scatter(X, y, color='black', s=30, marker='o', label='Data Points')

plt.plot(X\_test, predictions, color='blue', linewidth=2, label='LWR Fit')

plt.xlabel('X')

plt.ylabel('y')

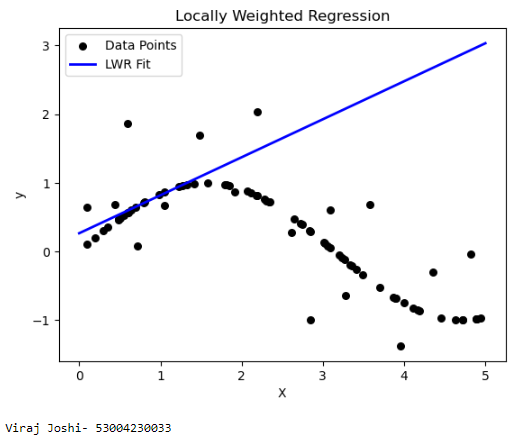
plt.title('Locally Weighted Regression')

plt.legend()

plt.show()

print('Viraj Joshi- 53004230033')

Output:-



Bayesian Network

Code:-

import numpy as np

import pandas as pd

from pgmpy.models import BayesianNetwork

from pgmpy.estimators import ParameterEstimator, MaximumLikelihoodEstimator

from pgmpy.inference import VariableElimination

import networkx as nx

import matplotlib.pyplot as plt

data = pd.DataFrame (data={'Age': [30, 40, 50, 60, 70],

'Gender': ['Male', 'Female', 'Male', 'Female', 'Male'],

'ChestPain': ['Typical', 'Atypical', 'Typical', 'Atypical', 'Typical'],

'HeartDisease': ['Yes', 'No', 'Yes', 'No', 'Yes']})

model = BayesianNetwork([('Age', 'HeartDisease'),

('Gender', 'HeartDisease'),

('ChestPain', 'HeartDisease')])

model.fit(data, estimator=MaximumLikelihoodEstimator)

pos = nx.circular\_layout(model)

nx.draw(model, pos, with\_labels=True, node\_size=5000, node\_color="skyblue", font\_size=12, font\_color="black")

print('Viraj Joshi- 53004230033')

plt.title("Bayesian Network Structure")

plt.show()

for cpd in model.get\_cpds():

print("CPD of", cpd.variable)

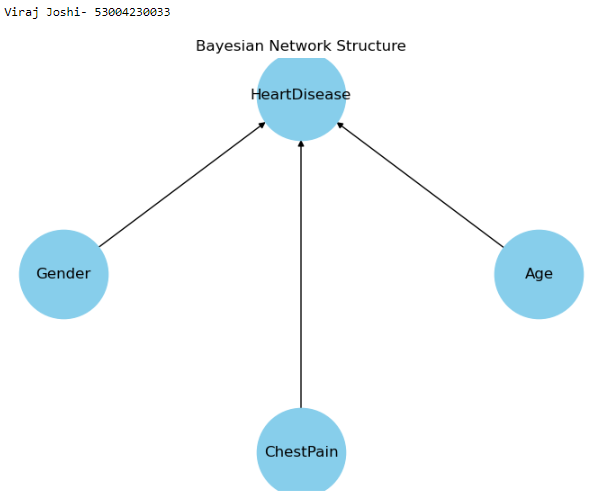
print(cpd)

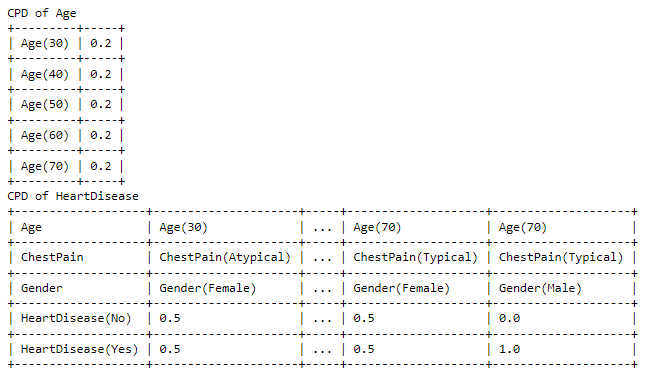
inference = VariableElimination(model)

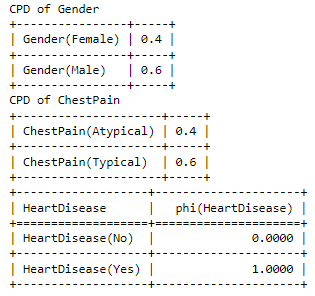
query = inference.query(variables=['HeartDisease'], evidence={'Age':50, 'Gender': 'Male', 'ChestPain': 'Typical'})

print(query)

Output:-







Perform Data Loading, Feature selection (Principal Component analysis) and Feature Scoring and Ranking

Code:-

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

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

from sklearn.metrics import accuracy\_score, classification\_report

sms\_data = [

"Free entry in 2 a weekly competition to win FA Cup final tickets",

"Hey, I will call you later. Don't forget to bring the document.",

"Congratulations! You've won a free cruise to the Bahamas",

"Hi there, can we meet tomorrow for lunch?",

"URGENT! Your mobile number has won a $2000 prize!",

"Reminder: Your appointment with the dentist is at 3 PM today.",

"You have won a lottery! Claim your prize now by calling us.",

"Are we still meeting at the coffee shop today?",

"Exclusive deal just for you! Buy now and get 50% off!",

"Can you send me the report by end of the day?"

]

sms\_labels = [

"spam", "ham", "spam", "ham", "spam",

"ham", "spam", "ham", "spam", "ham"

]

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(sms\_data)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, sms\_labels, test\_size=0.3, random\_state=42)

classifier = MultinomialNB()

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

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

accuracy = accuracy\_score(y\_test,y\_pred)

report = classification\_report(y\_test, y\_pred)

print("Test Data:")

for doc, actual, predicted in zip(sms\_data[len(sms\_data) - len(y\_test):], y\_test, y\_pred):

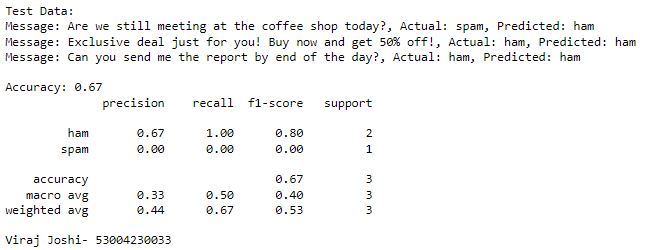
print(f"Message: {doc}, Actual: {actual}, Predicted: {predicted}")

print(f"\nAccuracy: {accuracy:.2f}")

print(report)

print('Viraj Joshi- 53004230033')

Output:-



Distance Metrics

Code:-

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification\_report,confusion\_matrix

# Load the Iris dataset

iris = load\_iris()

X = iris.data[:, :2] # Select only the first two features (sepal length and sepal width)

y = iris.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Initialize k-NN classifier with different distance metrics

k = 3

# List of distance metrics to test

distance\_metrics = ['euclidean', 'manhattan', 'chebyshev']

# Create subplots for each distance metric

fig, axes = plt.subplots(1, len(distance\_metrics), figsize=(15, 5))

for i, metric in enumerate(distance\_metrics):

knn\_classifier = KNeighborsClassifier(n\_neighbors=k, metric=metric)

# Fit the classifier to the training data

knn\_classifier.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = knn\_classifier.predict(X\_test)

# Evaluate the classifier's performance

print(f"Distance Metric: {metric}")

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

print("\n")

# Visualize the dataset and decision boundaries for the current metric

ax = axes[i]

# Plot the training data points

ax.scatter(X\_train[:, 0], X\_train[:, 1], c=y\_train, cmap='viridis', label='Training Data')

# Plot the testing data points

ax.scatter(X\_test[:, 0], X\_test[:, 1], c=y\_test, cmap='viridis', marker='x', s=100, label='Testing Data')

# Plot decision boundaries using the current metric

knn\_classifier = KNeighborsClassifier(n\_neighbors=k, metric=metric)

knn\_classifier.fit(X, y)

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.01), np.arange(y\_min, y\_max, 0.01))

Z = knn\_classifier.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

ax.contourf(xx, yy, Z, cmap='viridis', alpha=0.5, levels=range(4))

ax.set\_title(f'K-NN ({metric.capitalize()} Metric)')

ax.set\_xlabel('Sepal Length (cm)')

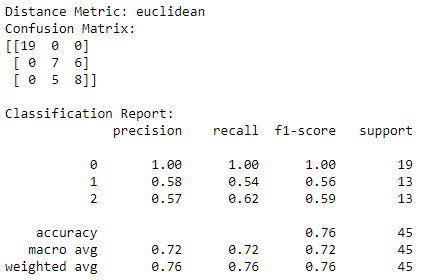
ax.set\_ylabel('Sepal Width (cm)')

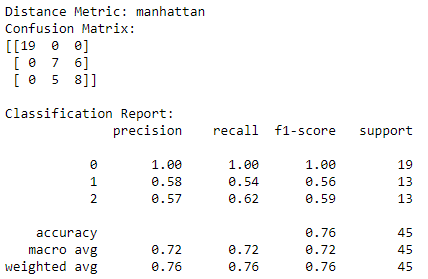
ax.legend()

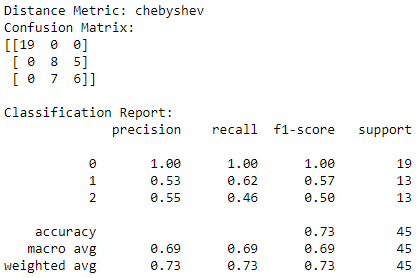
plt.show()

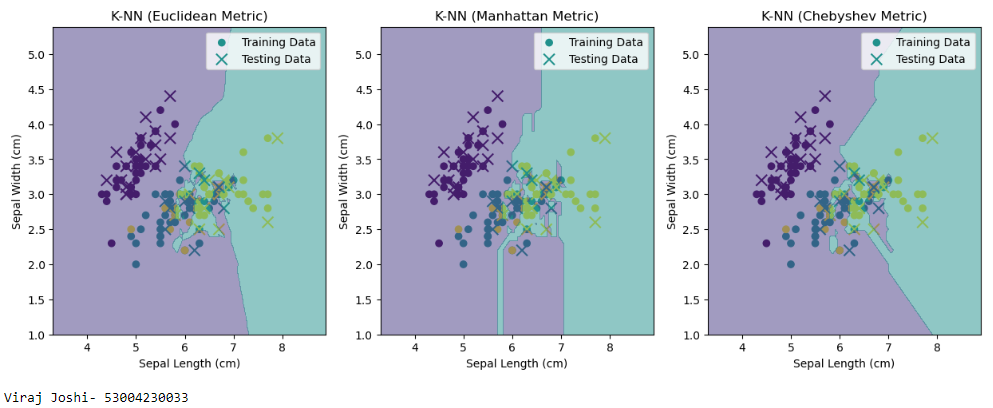
print('Viraj Joshi- 53004230033')

Output:-









Rule-Based Methods

Code:-

import numpy as np

from sklearn.datasets import load\_iris

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

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

#Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

#Split the data for testing

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

#Define a simple rule-based classifier function

def rule\_based\_classifier(x):

if x[2] < 2.0:

rule = "If feature 2 < 2.0, assign to Classd 0"

return 0 # Class 0

elif x[3] > 1.5:

rule = "If feature 2 >= 2.0 and feature 3 > 1.5, assign to Class 2"

return 2 # Class 2

else:

rule = "If feature 2 >= 2.0 and feature 3 <=1.5, assign to Class 1"

return 1 # Class 1

print("Rule:", rule)

# Apply the rule-based classifier to make predictions on the test set

y\_pred = [rule\_based\_classifier(x) for x in X\_test]

# Calculate accuracy, confusion matrix, and classification report

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

classification\_rep = classification\_report(y\_test, y\_pred, target\_names=iris.target\_names)

# Print the results

print("Accuracy:", accuracy)

print("Confusion Matrix:\n", conf\_matrix)

print("Classification Report:\n", classification\_rep)

print('Viraj Joshi- 53004230033')

Output:-

