4.

import numpy as np

def simple\_linear\_regression(x, y):

n = len(x)

x\_mean = np.mean(x)

y\_mean np.mean(y)

# Calculate slope (m) and y-intercept (b) using Least squares method

numerator = sum((x[i]x\_mean) (y[i]y\_mean) for i in range(n))

denominator = sum((x[i]x\_mean) \*\*2 for i in range(n))

slope numerator / denominator

y\_intercept = y\_mean slope x\_mean

return slope, y\_intercept

def main():

# Example data

x = np.array([1, 2, 3, 4, 5])

y= np.array([2, 3, 4, 5, 6])

#Perform Linear regression

slope, y\_intercept simple\_linear\_regression(x, y)

#Predict y for a new x value

new\_x = 6

predicted\_y = slope new\_x + y\_intercept

print("Predicted y for x =", new\_x, ":", predicted\_y)

5.

import numpy as np

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

from sklearn.linear\_model import LinearRegression

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

def multiple\_linear\_regression(X, y):

# Split the data into training and testing sets

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

# Fit the Linear regression model

model LinearRegression()

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

# Predict on the testing set

y\_pred model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 r2\_score(y\_test, y\_pred)

return model.coef, model.intercept, mse, r2

def main():

# Example data

X = np.array([[1, 2, 3],

[4, 5, 6],

[7, 8, 9],

[10, 11, 12]])

y = np.array([4, 7, 10, 13])

# Perform multiple linear regression

coefficients, intercept, mse, r2 multiple\_linear\_regression(X, y)

# Print results

print("Coefficients:", coefficients)

print("Intercept:",

intercept)

print("Mean Squared Error:", mse)

print("R^2 Score:", r2)

6.

# Importing necessary libraries

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

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

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Splitting the dataset into training and testing sets

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

# Defining the Decision Tree Classifier

clf = DecisionTreeClassifier()

# Define the grid of parameters to search

param\_grid = {

'criterion': ['entropy','gini'],

'max\_depth': [None, 5, 10, 15, 20],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4],

'max\_features': ['auto', 'sqrt', 'log2']

}

# Create the grid search cross-validation object

grid\_search = GridSearchCV(estimator=clf, param\_grid=param\_grid, cv=5, scoring='accuracy', n\_jobs=-1) # Fit the grid search to the data

grid\_search.fit(X\_train, y\_train)

# Get the best parameters and the best score

best\_params = grid\_search.best\_params

\_ best\_score = grid\_search.best\_score\_

# Print the best parameters and the best score

print("Best Parameters:", best\_params)

print("Best Score:", best\_score)

# Using the best model to make predictions

best\_clf = grid\_search.best\_estimator\_

y\_pred = best\_clf.predict(X\_test)

# Evaluate the model

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

print("Accuracy:", accuracy)

# Plotting the decision tree

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

plot\_tree(best\_clf, filled=True, feature\_names=iris.feature\_names,

class\_names=iris.target\_names)

plt.show()

7.

import pandas as pd

from sklearn.datasets import load\_iris

iris = load iris()

df = pd.concat([

(pd.DataFrame(data=iris['data'], columns=iris['feature names'])),

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

axis=1)

#Split the dataset into features (X) and target (y)

X= df.drop('target', axis=1)

y = df['target']

from sklearn.model\_selection import train test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaler.fit(X\_train)

X train = scaler.transform(X train)

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

from sklearn.neighbors import KNeighbors Classifier

classifier = KNeighbors Classifier(n neighbors=5)

classifier.fit(X train, y train)

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

from sklearn.metrics import classification report, confusion matrix

print(confusion matrix(y test, y pred))

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