**EXPERIMENT 7**

**Implementation of logistic reggresssion using sklearn**

Import numpy as np

from sklearn.datasets import load\_iris

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

from sklearn.neighbors import KNeighborsClassifier

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

# Step 2: Load the Iris dataset

iris = load\_iris()

X = iris.data # Features

y = iris.target # Labels

# Step 3: 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)

# Step 4: Initialize the KNN classifier with k=3

knn = KNeighborsClassifier(n\_neighbors=3)

# Step 5: Train the model on the training data

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

# Step 6: Make predictions on the test data

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

# Step 7: Evaluate the model’s performance

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

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

# Display results

print("Accuracy:", accuracy)

print("\nClassification Report:\n", classification\_report\_text)

**OUTPUT:**

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 19

1 1.00 1.00 1.00 13

2 1.00 1.00 1.00 13

accuracy 1.00 45

macro avg 1.00 1.00 1.00 45

weighted avg 1.00 1.00 1.00 45

**EXPERIMENT 8**

**Implementation of logistic reggresssion using sklearn**

**# import necessary libraries**

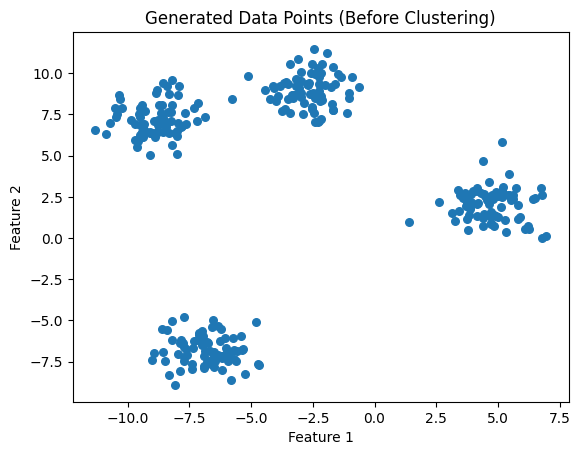
import numpy as np  
from sklearn.datasets import load\_iris  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
from sklearn.linear\_model import LogisticRegression  
from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report  
#Load Iris dataset  
data = load\_iris()  
X = data.data  
y = data.target  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  
  
scaler = StandardScaler()  
X\_train\_scaled = scaler.fit\_transform(X\_train)  
X\_test\_scaled = scaler.transform(X\_test)  
  
log\_reg = LogisticRegression(max\_iter=200)  
log\_reg.fit(X\_train\_scaled, y\_train)  
  
y\_pred = log\_reg.predict(X\_test\_scaled)  
  
accuracy = accuracy\_score(y\_test, y\_pred)  
print(f"Accuracy of the Logistic Regression model: {accuracy:.2f}")  
  
print("\nConfusion Matrix:")  
print(confusion\_matrix(y\_test, y\_pred))  
  
print("\nClassification Report:")  
print(classification\_report(y\_test, y\_pred))

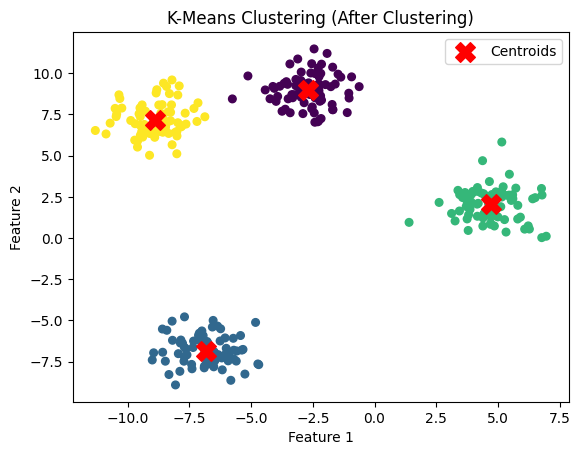
OUTPUT:  
Accuracy of the Logistic Regression model: 1.00  
  
Confusion Matrix:  
[[19 0 0]  
 [ 0 13 0]  
 [ 0 0 13]]  
  
Classification Report:  
 precision recall f1-score support  
  
 0 1.00 1.00 1.00 19  
 1 1.00 1.00 1.00 13  
 2 1.00 1.00 1.00 13  
  
 accuracy 1.00 45  
 macro avg 1.00 1.00 1.00 45  
weighted avg 1.00 1.00 1.00 45

**EXPERIMENT 9**

# Step 1: Import necessary libraries  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.cluster import KMeans  
from sklearn.datasets import make\_blobs  
  
# Step 2: Create a simple synthetic dataset with 2 features  
X, y = make\_blobs(n\_samples=300, centers=4, random\_state=42)  
  
# Step 3: Visualize the data points (before clustering)  
plt.scatter(X[:, 0], X[:, 1], s=30, cmap='viridis')  
plt.title("Generated Data Points (Before Clustering)")  
plt.xlabel("Feature 1")  
plt.ylabel("Feature 2")  
plt.show()  
  
# Step 4: Apply K-Means Clustering with 4 clusters (since we generated 4 centers)  
kmeans = KMeans(n\_clusters=4, random\_state=42)  
kmeans.fit(X)  
  
# Step 5: Get the cluster labels and cluster centers  
y\_kmeans = kmeans.predict(X)  
centroids = kmeans.cluster\_centers\_  
  
# Step 6: Visualize the clusters and the centroids (after clustering)  
plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, s=30, cmap='viridis')  
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200, marker='X', label='Centroids')  
plt.title("K-Means Clustering (After Clustering)")  
plt.xlabel("Feature 1")  
plt.ylabel("Feature 2")  
plt.legend()  
plt.show()

**Output:**

****

****

**EXPERIMENT-10**

**performance analysis of Classification Algorithm on a specific dataset**

# Step 1: Import necessary libraries

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.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.svm import SVC

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

# Step 2: Load the dataset (Iris dataset in this case)

data = load\_iris()

X = data.data # Features (sepal length, sepal width, petal length, petal width)

y = data.target # Target labels (Iris species)

# Step 3: Split the dataset into training and testing sets (70% train, 30% test)

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

# Step 4: Feature scaling (Standardization of features)

scaler = StandardScaler()

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

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

# Step 5: Initialize the classification models

models = {

"Logistic Regression": LogisticRegression(),

"Decision Tree": DecisionTreeClassifier(),

"K-Nearest Neighbors": KNeighborsClassifier(),

"Random Forest": RandomForestClassifier(),

"Support Vector Machine": SVC(probability=True)

}

# Step 6: Train and evaluate each model

results = {}

for model\_name, model in models.items():

# Train the model

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

# Predict on the test set

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

# Calculate accuracy

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

# Confusion matrix

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

# Classification report (precision, recall, F1-score)

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

# ROC-AUC Score

if len(np.unique(y)) == 2: # For binary classification

roc\_auc = roc\_auc\_score(y\_test, model.predict\_proba(X\_test\_scaled)[:, 1])

else:

roc\_auc = roc\_auc\_score(y\_test, model.predict\_proba(X\_test\_scaled), multi\_class='ovr')

results[model\_name] = {

"Accuracy": accuracy,

"Confusion Matrix": conf\_matrix,

"Classification Report": class\_report,

"ROC-AUC": roc\_auc

}

# Step 7: Display results

for model\_name, result in results.items():

print(f"\nModel: {model\_name}")

print(f"Accuracy: {result['Accuracy']:.2f}")

print("Confusion Matrix:")

print(result["Confusion Matrix"])

print("Classification Report:")

print(pd.DataFrame(result["Classification Report"]).T)

print(f"ROC-AUC: {result['ROC-AUC']:.2f}")

# Step 8: Compare performance with a bar plot (Accuracy, ROC-AUC)

accuracy\_scores = [result['Accuracy'] for result in results.values()]

roc\_auc\_scores = [result['ROC-AUC'] for result in results.values()]

# Plotting the results

fig, ax = plt.subplots(1, 2, figsize=(14, 6))

ax[0].barh(list(results.keys()), accuracy\_scores, color='skyblue')

ax[0].set\_xlabel('Accuracy')

ax[0].set\_title('Accuracy Comparison')

ax[1].barh(list(results.keys()), roc\_auc\_scores, color='lightgreen')

ax[1].set\_xlabel('ROC-AUC')

ax[1].set\_title('ROC-AUC Comparison')

plt.tight\_layout()

plt.show()

OUTPUT:

Model: Logistic Regression

Accuracy: 1.00

Confusion Matrix:

[[19 0 0]

[ 0 13 0]

[ 0 0 13]]

Classification Report:

precision recall f1-score support

0 1.0 1.0 1.0 19.0

1 1.0 1.0 1.0 13.0

2 1.0 1.0 1.0 13.0

accuracy 1.0 1.0 1.0 1.0

macro avg 1.0 1.0 1.0 45.0

weighted avg 1.0 1.0 1.0 45.0

ROC-AUC: 1.00

Model: Decision Tree

Accuracy: 1.00

Confusion Matrix:

[[19 0 0]

[ 0 13 0]

[ 0 0 13]]

Classification Report:

precision recall f1-score support

0 1.0 1.0 1.0 19.0

1 1.0 1.0 1.0 13.0

2 1.0 1.0 1.0 13.0

accuracy 1.0 1.0 1.0 1.0

macro avg 1.0 1.0 1.0 45.0

weighted avg 1.0 1.0 1.0 45.0

ROC-AUC: 1.00

Model: K-Nearest Neighbors

Accuracy: 1.00

Confusion Matrix:

[[19 0 0]

[ 0 13 0]

[ 0 0 13]]

Classification Report:

precision recall f1-score support

0 1.0 1.0 1.0 19.0

1 1.0 1.0 1.0 13.0

2 1.0 1.0 1.0 13.0

accuracy 1.0 1.0 1.0 1.0

macro avg 1.0 1.0 1.0 45.0

weighted avg 1.0 1.0 1.0 45.0

ROC-AUC: 1.00

Model: Random Forest

Accuracy: 1.00

Confusion Matrix:

[[19 0 0]

[ 0 13 0]

[ 0 0 13]]

Classification Report:

precision recall f1-score support

0 1.0 1.0 1.0 19.0

1 1.0 1.0 1.0 13.0

2 1.0 1.0 1.0 13.0

accuracy 1.0 1.0 1.0 1.0

macro avg 1.0 1.0 1.0 45.0

weighted avg 1.0 1.0 1.0 45.0

ROC-AUC: 1.00

Model: Support Vector Machine

Accuracy: 1.00

Confusion Matrix:

[[19 0 0]

[ 0 13 0]

[ 0 0 13]]

Classification Report:

precision recall f1-score support

0 1.0 1.0 1.0 19.0

1 1.0 1.0 1.0 13.0

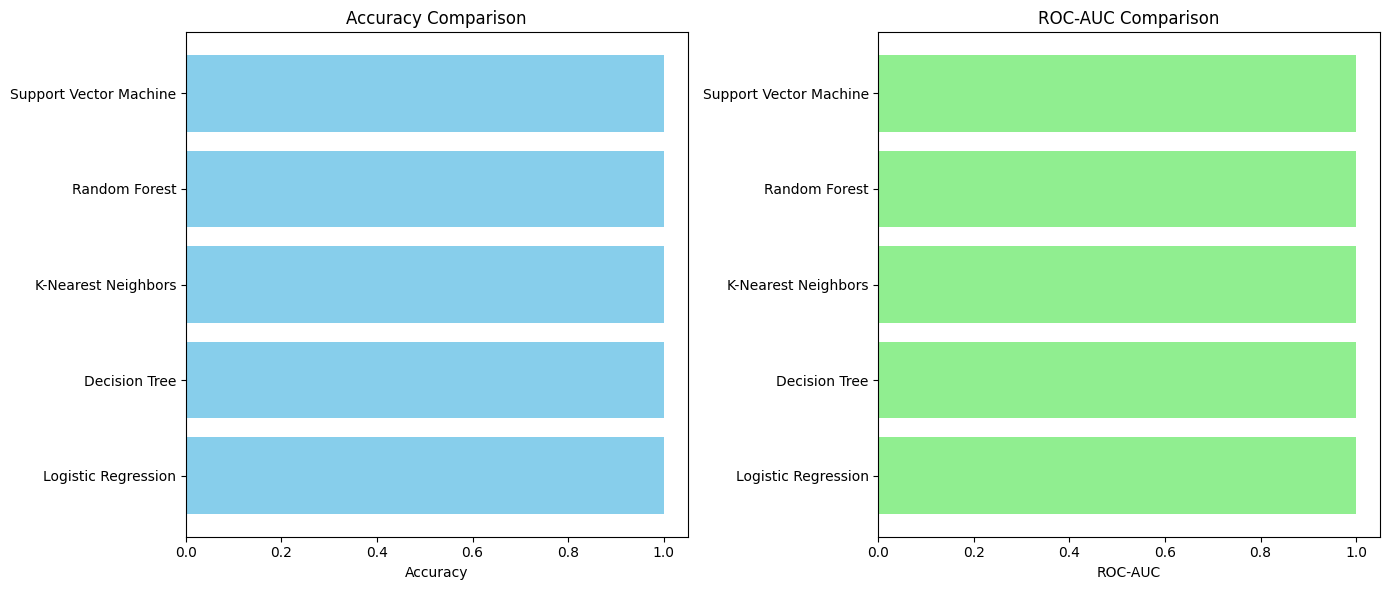
2 1.0 1.0 1.0 13.0

accuracy 1.0 1.0 1.0 1.0

macro avg 1.0 1.0 1.0 45.0

weighted avg 1.0 1.0 1.0 45.0

ROC-AUC: 1.00

****

**EXPERIMENT-4**

**4. Write a python program to implement simple linear regression.**

**Linear regression:** Linear regression is a type of ML algorithm based on supervised learning Simple regression has only one independent variable and one dependent variable.

**Program:**

import numpy as nmp

import matplotlib.pyplot as mtplt

def estimate-coeff(p,q):

#Here we will estimate the total number of points or observation.

n1=nmp.size(P).

#Now we will calculate the mean of a and b vector

m-p=nmp.mean(P)

M-q=nmp.mean(q)

#Here, we will calculate the cross deviation and deviation about 'a'

ss-pq=nmp-Sum(q\*p)-n1 \*m-q\*m-p

ss-pp=nmp.Sum(p\*p)-n1\*m-p\*m-p

#here, we will calculate the regression coefficients

B-1=ss-pq/ss-pp

b-0=m-q-b-1\*m-p

return (b-0, b-1)

def plot-regression line (p,q,b):

#Now, we will plot the actual points or observation as scatter plot

mtplt. scatter(p.q, color= "m",

Marker= "o", s=30)

#Now we will plot the actual points or observation as scatter plot

Mtplt.scatter (p,q,color="m", marker "o", s=30)

#here, we will calculate the predicted response Vector

q-pred= b[0]+b[1]\*p

#here will plot the regression line

mtplt. plot (p,q\_pred, color-"g")

# here, we will put the labels

mtplt. xlabel('p')

mtplt .ylabel('q')

#here, we will define the function to show plot

mipit show()

def main():

#Entering the observation points or data

p=nmp.array([10, 11, 12, 13, 14, 15, 16,17,18,19])

q=nmp.array ([11,13,12,15,17,18,18,19,20,22])

#now, we will estimate the coefficients

b=estimate\_coeff(p,q)

print('Estimated coefficients are: \nb-0={}\nb-1={}”format(b[o], b[1]))

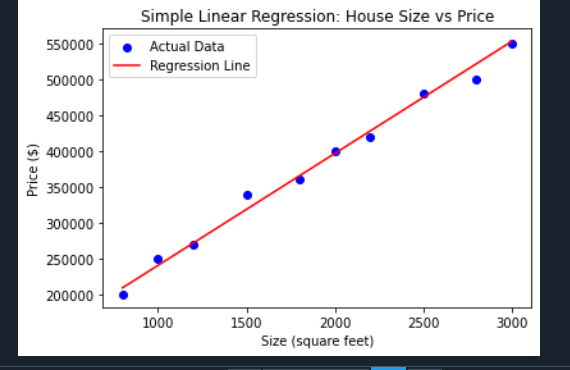
#Now, we will plot the regression line

plot regression-line(p,q,b)

if--name--- "--main--":

Main()

**OUTPUT:**



Estimated coefficient are:

b-0=-0.4606060606060609

b-1=1.169696969696969697

**EXPERIMENT-5**

**5.Implemention of multiple linear regression for house price prediction using sklearn (without graph)**

import numpy as np # For numerical computations

import pandas as pd # For data handling

from sklearn.model\_selection import train\_test\_split # To split the dataset

from sklearn.linear\_model import LinearRegression # For the linear regression model

from sklearn.metrics import mean\_squared\_error, r2\_score # For model evaluation

# Step 1: Create a dataset

# This is an example dataset of houses with multiple features like size, bedrooms, and floors, and the target price

data = {

'size': [2100, 1600, 2400, 1416, 3000, 1985, 1534, 1427, 1380, 1494],

'bedrooms': [3, 2, 4, 3, 4, 3, 3, 3, 2, 3],

'floors': [2, 1, 2, 1, 2, 1, 1, 1, 1, 1],

'price': [400000, 330000, 369000, 232000, 539000, 299000, 314000, 198000, 212000, 242000]

}

# Convert it into a pandas DataFrame

df = pd.DataFrame(data)

# Step 2: Define independent variables (X) and the dependent variable (y)

# X includes 'size', 'bedrooms', and 'floors' (predictor variables), and y is the 'price' (target variable)

X = df[['size', 'bedrooms', 'floors']] # Feature matrix

y = df['price'] # Target vector

# Step 3: Split the dataset into training and testing sets (80% train, 20% test)

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

# Step 4: Create an instance of LinearRegression and fit the model

model = LinearRegression() # Instantiate the model

model.fit(X\_train, y\_train) # Train the model using the training data

# Step 5: Make predictions on the test set

y\_pred = model.predict(X\_test) # Predict the house prices for the test data

# Step 6: Evaluate the model using Mean Squared Error (MSE) and R-squared (R²) score

mse = mean\_squared\_error(y\_test, y\_pred) # Calculate Mean Squared Error

r2 = r2\_score(y\_test, y\_pred) # Calculate R-squared

print(f"Mean Squared Error: {mse:.2f}") # Print MSE

print(f"R-squared: {r2:.2f}") # Print R² score

# Step 7: Print coefficients of the regression model

# This includes the intercept and coefficients for each feature

print(f"Intercept: {model.intercept\_:.2f}") # Model intercept

print(f"Coefficients: {model.coef\_}") # Model coefficients (for 'size', 'bedrooms', and 'floors')

# Step 8: Display predictions alongside actual values for comparison

results = pd.DataFrame({'Actual': y\_test, 'Predicted': y\_pred})

print("Comparison of actual and predicted prices:")

print(results)

**OUTPUT:**

Mean Squared Error: 2585529550.08

R-squared: 0.26

Intercept: 101856.36

Coefficients: [ 197.80406628 -64682.43976773 38401.20975138]

Comparison of actual and predicted prices:

Actual Predicted

8 212000 283862.302295

1 330000 327379.196877

**5.Implemention of multiple linear regression for house price prediction using sklearn (with graph)**

import numpy as np # For numerical computations

import pandas as pd # For handling data

import matplotlib.pyplot as plt # For plotting graphs

from sklearn.model\_selection import train\_test\_split # To split the dataset into train and test sets

from sklearn.linear\_model import LinearRegression # To implement simple linear regression

from sklearn.metrics import mean\_squared\_error # To evaluate the model

# Step 1: Create a simple dataset

# Let's assume we have a dataset of house size (in square feet) and their corresponding prices

data = {'size': [800, 1000, 1200, 1500, 1800, 2000, 2200, 2500, 2800, 3000],

'price': [200000, 250000, 270000, 340000, 360000, 400000, 420000, 480000, 500000, 550000]}

# Convert it into a pandas DataFrame

df = pd.DataFrame(data)

# Step 2: Define the independent variable (X) and the dependent variable (y)

# X will be the house size, and y will be the price

X = df[['size']] # 2D array required by sklearn (independent variable)

y = df['price'] # 1D array (dependent variable)

# Step 3: Split the dataset into training and testing sets (80% train, 20% test)

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

# Step 4: Create an instance of LinearRegression and fit the model

model = LinearRegression() # Create the linear regression model

model.fit(X\_train, y\_train) # Train the model on the training data

# Step 5: Make predictions on the test set

y\_pred = model.predict(X\_test) # Predict the house prices for the test data

# Step 6: Evaluate the model using Mean Squared Error (MSE)

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

print(f"Mean Squared Error: {mse:.2f}") # Print the MSE

# Step 7: Get the slope (coefficient) and intercept of the linear regression line

slope = model.coef\_ # Slope (Coefficient)

intercept = model.intercept\_ # Intercept

print(f"Slope (Coefficient): {slope[0]:.2f}")

print(f"Intercept: {intercept:.2f}")

# Step 8: Visualize the results by plotting the regression line

plt.scatter(X, y, color='blue', label='Actual Data') # Scatter plot of the actual data

plt.plot(X, model.predict(X), color='red', label='Regression Line') # Plot the regression line

# Add labels and title

plt.xlabel('Size (square feet)')

plt.ylabel('Price ($)')

plt.title('Simple Linear Regression: House Size vs Price')

plt.legend() # Show the legend

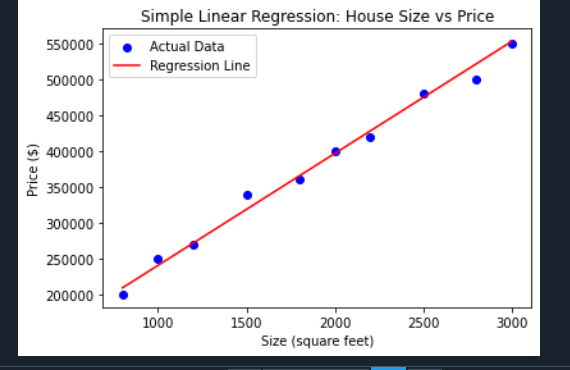
plt.show() # Display the graph

**OUTPUT:**

Mean Squared Error: 294571460.58

Slope (Coefficient): 156.58

Intercept: 83917.96



**EXPERIMENT-6**

**6.a. Implementation of Decision tree using sklearn and its parameter tuning.**

import pandas as pd

from sklearn.datasets import load\_iris

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

from sklearn.tree import DecisionTreeClassifier

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

from sklearn.model\_selection import GridSearchCV

# Load the Iris dataset

iris = load\_iris()

X = iris.data # Features

y = iris.target # Target variable

# Split 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)

# Create a Decision Tree classifier

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

# Train the model

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

# Make predictions

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

# Evaluate the model

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

print(f"Accuracy: {accuracy:.2f}")

print("Classification Report:")

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

print("Confusion Matrix:")

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

# Define the parameter grid for tuning

param\_grid = {

'max\_depth': [None, 2, 4, 6, 8, 10],

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

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

}

# Create a GridSearchCV object

grid\_search = GridSearchCV(estimator=dt\_model, param\_grid=param\_grid,

cv=5, n\_jobs=-1, verbose=1)

# Fit the grid search

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

# Print best parameters and best score

print("Best Parameters:", grid\_search.best\_params\_)

print("Best Cross-Validation Score:", grid\_search.best\_score\_)

**OUTPUT:**

Accuracy: 1.00

Classification Report:

precision recall f1-score support

0 1.00 1.00 1.00 10

1 1.00 1.00 1.00 9

2 1.00 1.00 1.00 11

accuracy 1.00 30

macro avg 1.00 1.00 1.00 30

weighted avg 1.00 1.00 1.00 30

Confusion Matrix:

[[10 0 0]

[ 0 9 0]

[ 0 0 11]]

Fitting 5 folds for each of 54 candidates, totalling 270 fits

Best Parameters: {'max\_depth': None, 'min\_samples\_leaf': 4, 'min\_samples\_split': 2}

Best Cross-Validation Score: 0.95

**6.b.Implementation of Decision tree using sklearn and its parameter tuning.**

# Importing the required libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn import metrics

import seaborn as sns

from sklearn.datasets import load\_iris

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

from sklearn import tree

# Loading the dataset

iris = load\_iris()

#converting the data to a pandas dataframe

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

#creating a separate column for the target variable of iris dataset

data['Species'] = iris.target

#replacing the categories of target variable with the actual names of the species

target = np.unique(iris.target)

target\_n = np.unique(iris.target\_names)

target\_dict = dict(zip(target, target\_n))

data['Species'] = data['Species'].replace(target\_dict)

# Separating the independent dependent variables of the dataset

x = data.drop(columns = "Species")

y = data["Species"]

names\_features = x.columns

target\_labels = y.unique()

# Splitting the dataset into training and testing datasets

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.3, random\_state = 93)

# Importing the Decision Tree classifier class from sklearn

from sklearn.tree import DecisionTreeClassifier

# Creating an instance of the classifier class

dtc = DecisionTreeClassifier(max\_depth = 3, random\_state = 93)

# Fitting the training dataset to the model

dtc.fit(x\_train, y\_train)

# Plotting the Decision Tree

plt.figure(figsize = (30, 10), facecolor = 'b')

Tree = tree.plot\_tree(dtc, feature\_names = names\_features, class\_names = target\_labels, rounded = True, filled = True, fontsize = 14)

plt.show()

y\_pred = dtc.predict(x\_test)

# Finding the confusion matrix

confusion\_matrix = metrics.confusion\_matrix(y\_test, y\_pred)

matrix = pd.DataFrame(confusion\_matrix)

axis = plt.axes()

sns.set(font\_scale = 1.3)

plt.figure(figsize = (10,7))

# Plotting heatmap

sns.heatmap(matrix, annot = True, fmt = "g", ax = axis, cmap = "magma")

axis.set\_title('Confusion Matrix')

axis.set\_xlabel("Predicted Values", fontsize = 10)

axis.set\_xticklabels([''] + target\_labels)

axis.set\_ylabel( "True Labels", fontsize = 10)

axis.set\_yticklabels(list(target\_labels), rotation = 0)

plt.show()

**OUTPUT:**

