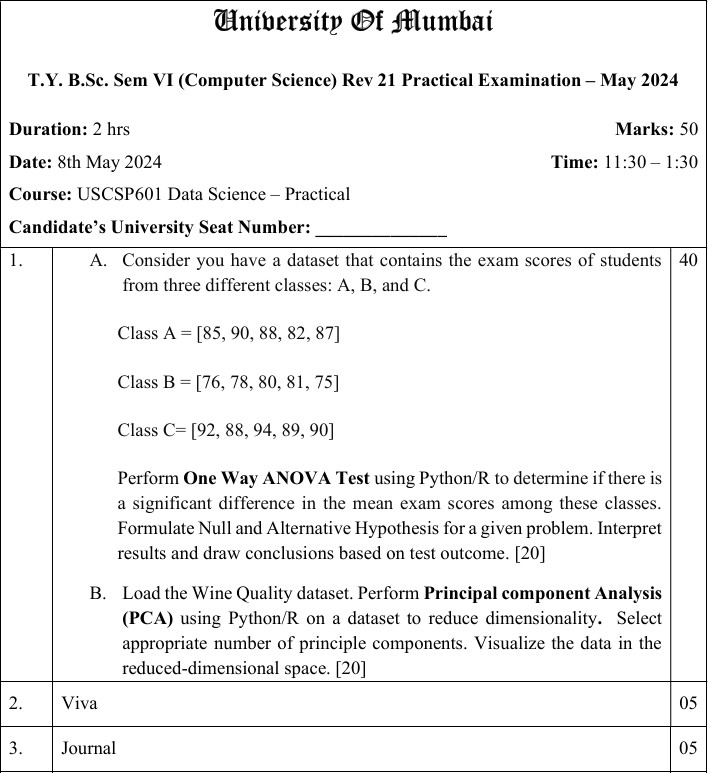
# Data Science University Slip

****

1. Input:

from scipy.stats import f\_oneway class\_A = [85, 90, 88, 82, 87]

class\_B = [76, 78, 80, 81, 75]

class\_C = [92, 88, 94, 89, 90]

f\_statistic, p\_value = f\_oneway(class\_A, class\_B, class\_C) print("F-statistic:", f\_statistic)

print("P-value:", p\_value) alpha = 0.05

if p\_value < alpha:

print("Conclusion: Reject the null hypothesis (H₀).")

print("There is a significant difference in the mean exam scores among the classes.")

else:

print("Conclusion: Fail to reject the null hypothesis (H₀).")

print("There is no significant difference in the mean exam scores among the classes.")

# Output:

F-statistic: 28.583333333333343 P-value: 2.727133500331836e-05

Conclusion: Reject the null hypothesis (H₀).

There is a significant difference in the mean exam scores among the classes.

1. Input:

# Import necessary libraries import pandas as pd

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

# Step 1: Load Wine Quality dataset # Use red wine dataset from UCI

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/wine- quality/winequality-red.csv"

data = pd.read\_csv(url, sep=';')

# Step 2: Separate features and target

X = data.drop('quality', axis=1) # features y = data['quality'] # target

# Step 3: Standardize the data scaler = StandardScaler()

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

# Step 4: Apply PCA pca = PCA()

X\_pca = pca.fit\_transform(X\_scaled)

# Step 5: Explained Variance

explained\_variance = pca.explained\_variance\_ratio\_ cumulative\_variance = explained\_variance.cumsum()

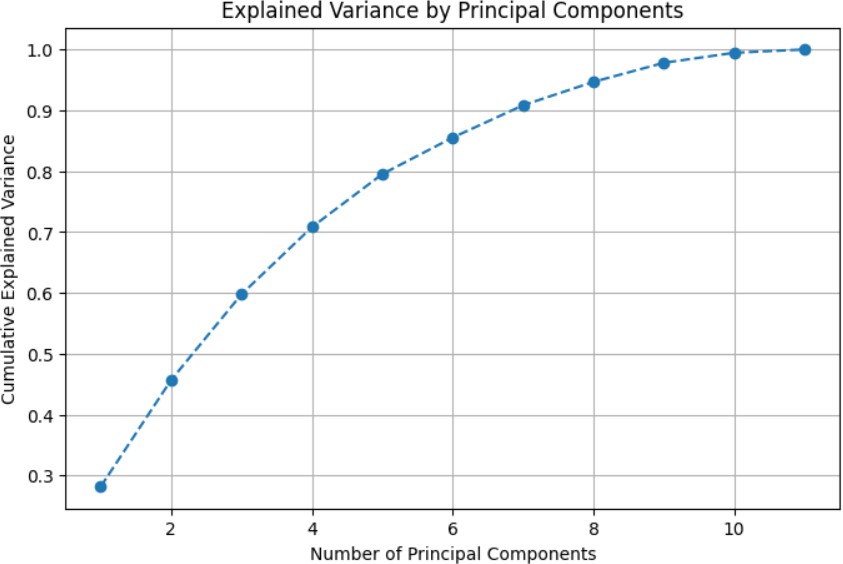
# Plot cumulative explained variance plt.figure(figsize=(8, 5))

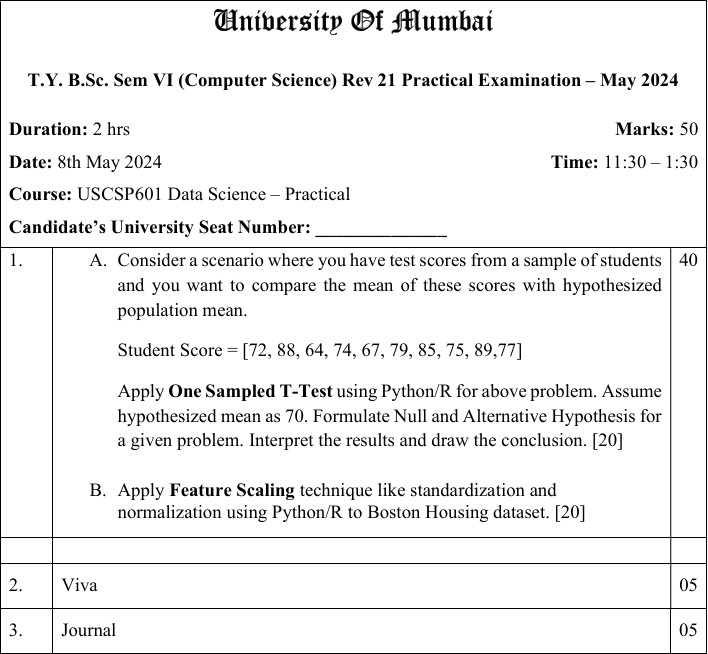
plt.plot(range(1, len(cumulative\_variance)+1), cumulative\_variance, marker='o', linestyle='--')

plt.title('Explained Variance by Principal Components') plt.xlabel('Number of Principal Components') plt.ylabel('Cumulative Explained Variance') plt.grid(True)

plt.show()

# Output:

****



1. Input:

from scipy import stats

# Given data

student\_scores = [72, 88, 64, 74, 67, 79, 85, 75, 89, 77]

# Hypothesized population mean mu\_0 = 70

# One-sample T-test

t\_stat, p\_value = stats.ttest\_1samp(student\_scores, mu\_0)

print("T-statistic:", t\_stat) print("P-value:", p\_value)

# Interpretation alpha = 0.05

if p\_value < alpha:

print("Conclusion: Reject the null hypothesis.")

print("The mean student score is significantly different from 70.") else:

print("Conclusion: Fail to reject the null hypothesis.")

print("There is no significant difference from the hypothesized mean of 70.")

# Output:

T-statistic: 2.6249999999999996

P-value: 0.027583862030934297

Conclusion: Reject the null hypothesis.

The mean student score is significantly different from 70.

1. Input:

import pandas as pd

from sklearn.datasets import fetch\_openml

from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Load Boston Housing dataset

boston = fetch\_openml(name='boston', version=1, as\_frame=True) df = boston.frame

# Separate features and target

X = df.drop(columns='MEDV') # Features

y = df['MEDV'] # Target variable (Median value of owner-occupied homes)

#

# Standardization #

standard\_scaler = StandardScaler()

X\_standardized = standard\_scaler.fit\_transform(X)

df\_standardized = pd.DataFrame(X\_standardized, columns=X.columns)

#

# Normalization (Min-Max Scaling) #

minmax\_scaler = MinMaxScaler()

X\_normalized = minmax\_scaler.fit\_transform(X)

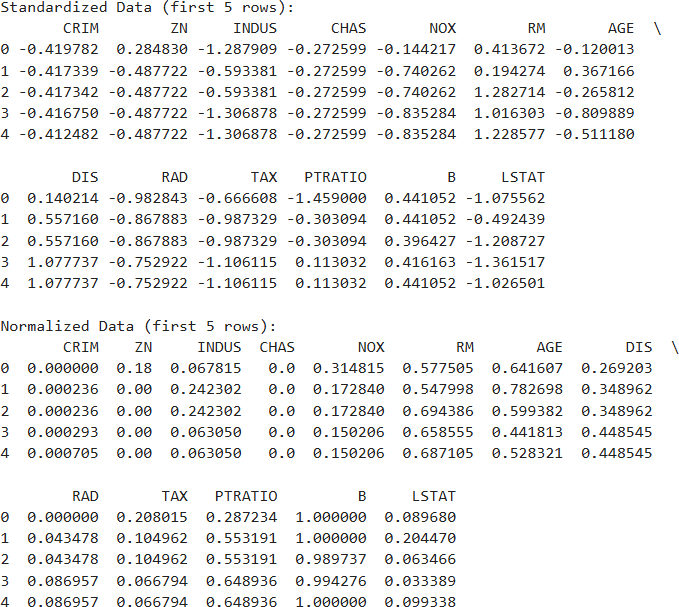
df\_normalized = pd.DataFrame(X\_normalized, columns=X.columns)

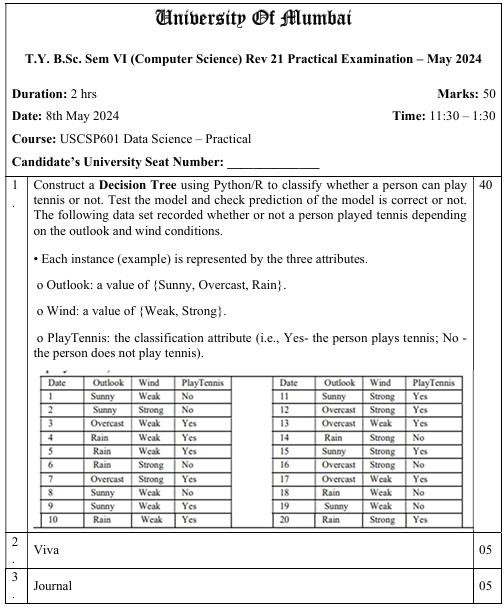
# Display a preview

print("Standardized Data (first 5 rows):") print(df\_standardized.head())

print("\nNormalized Data (first 5 rows):") print(df\_normalized.head())

# Output:

****



**Input:**

import pandas as pd

from sklearn.tree import DecisionTreeClassifier from sklearn.preprocessing import LabelEncoder from sklearn.metrics import accuracy\_score

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

import matplotlib.pyplot as plt

# Data from the image data = {

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Overcast', 'Overcast', 'Sunny', 'Sunny', 'Rain',

'Sunny', 'Overcast', 'Overcast', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain', 'Sunny', 'Rain'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong', 'Weak', 'Weak', 'Strong', 'Weak',

'Strong', 'Strong', 'Weak', 'Strong', 'Strong', 'Strong', 'Weak', 'Strong', 'Weak', 'Strong'],

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

'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes']

}

# Convert to DataFrame df = pd.DataFrame(data)

# Encode categorical variables le\_outlook = LabelEncoder() le\_wind = LabelEncoder() le\_play = LabelEncoder()

df['Outlook'] = le\_outlook.fit\_transform(df['Outlook']) df['Wind'] = le\_wind.fit\_transform(df['Wind'])

df['PlayTennis'] = le\_play.fit\_transform(df['PlayTennis'])

# Features and label

X = df[['Outlook', 'Wind']] y = df['PlayTennis']

# Split dataset into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Train Decision Tree model

model = DecisionTreeClassifier(criterion='entropy', random\_state=0)

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

# Predict and check accuracy y\_pred = model.predict(X\_test)

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

print("Predictions:", y\_pred) print("Actual :", list(y\_test))

print(f"Accuracy: {accuracy \* 100:.2f}%")

# Visualize Decision Tree plt.figure(figsize=(12, 6))

tree.plot\_tree(model, feature\_names=['Outlook', 'Wind'], class\_names=le\_play.classes\_, filled=True)

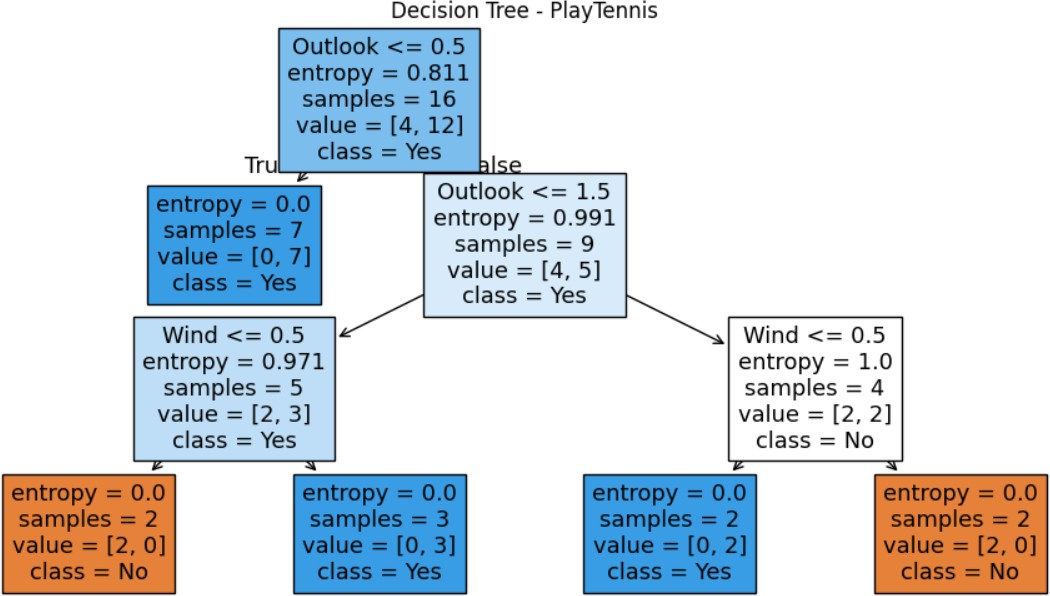
plt.title("Decision Tree - PlayTennis") plt.show()

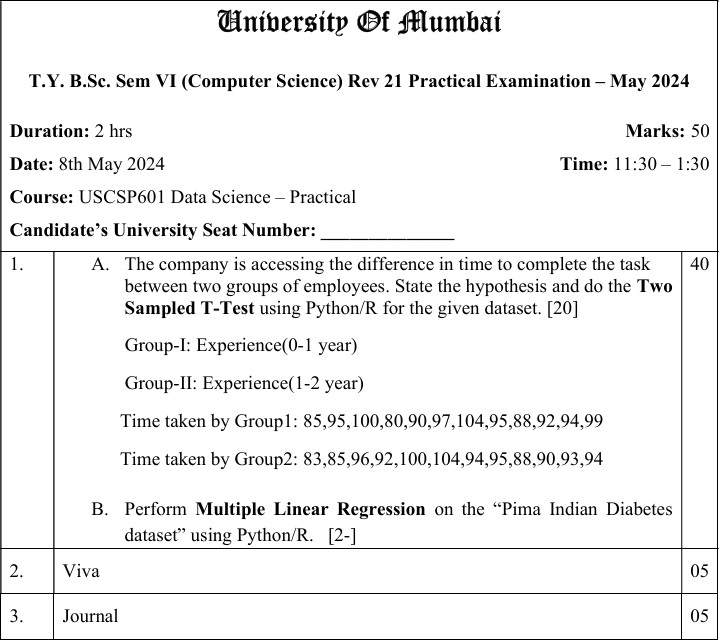
# Output:

Predictions: [0 1 0 1]

Actual : [0, 0, 1, 1]

Accuracy: 50.00%





1. Input:

import scipy.stats as stats

# Group-I: Experience (0-1 year)

group1 = [85, 95, 100, 80, 90, 97, 104, 95, 88, 92, 94, 99]

# Group-II: Experience (1-2 years)

group2 = [83, 85, 96, 92, 100, 104, 94, 95, 88, 90, 93, 94]

# Perform two-sample independent t-test

t\_stat, p\_value = stats.ttest\_ind(group1, group2)

# Print results

print("T-statistic:", t\_stat) print("P-value:", p\_value)

# Significance level alpha = 0.05

# Interpret result if p\_value < alpha:

print("Conclusion: Reject the null hypothesis.")

print("There is a significant difference in task completion time between the groups.")

else:

print("Conclusion: Fail to reject the null hypothesis.")

print("There is no significant difference in task completion time between the groups.")

# Output:

T-statistic: 0.16119885358127883

P-value: 0.8734079490313532

Conclusion: Fail to reject the null hypothesis.

There is no significant difference in task completion time between the groups.

# B)

import pandas as pd

from sklearn.model\_selection import train\_test\_split from sklearn.linear\_model import LinearRegression

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

# Load dataset from online source (or local CSV)

url = "https://raw.githubusercontent.com/plotly/datasets/master/diabetes.csv" df = pd.read\_csv(url)

# Display the first few rows print(df.head())

# Features and target

X = df.drop('Outcome', axis=1) # Independent variables

y = df['Outcome'] # Dependent variable (binary classification)

# Split data (80% training, 20% testing)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Create and train the model model = LinearRegression() model.fit(X\_train, y\_train)

# Predict on test set

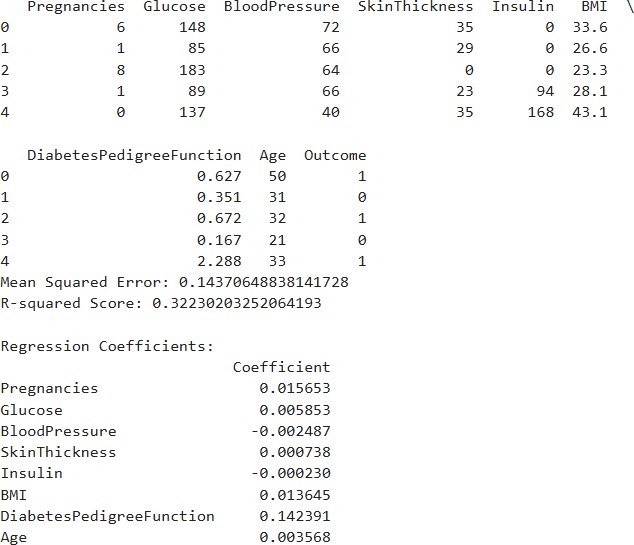
y\_pred = model.predict(X\_test)

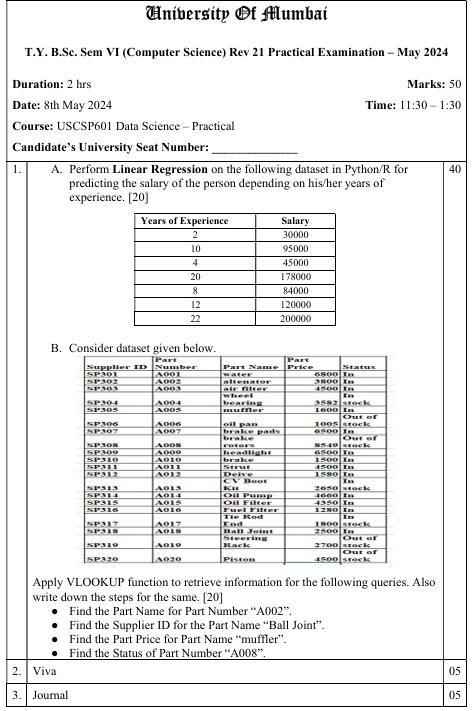
print("Mean Squared Error:", mean\_squared\_error(y\_test, y\_pred)) print("R-squared Score:", r2\_score(y\_test, y\_pred))

coeff\_df = pd.DataFrame(model.coef\_, X.columns, columns=['Coefficient']) print("\nRegression Coefficients:")

print(coeff\_df)

# Output:

****



**A:**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

# Dataset data = {

'YearsExperience': [2, 10, 4, 20, 8, 12, 22],

'Salary': [30000, 95000, 45000, 178000, 84000, 120000, 200000]

}

# Create DataFrame

df = pd.DataFrame(data)

# Features and Target

X = df[['YearsExperience']] # Independent variable y = df['Salary'] # Dependent variable

# Create and train the model model = LinearRegression() model.fit(X, y)

# Predict Salary

df['PredictedSalary'] = model.predict(X)

# Show Coefficients

print("Intercept (b0):", model.intercept\_) print("Coefficient (b1):", model.coef\_[0])

# Plotting the results

plt.scatter(X, y, color='blue', label='Actual Salary')

plt.plot(X, df['PredictedSalary'], color='red', label='Predicted Regression Line') plt.xlabel("Years of Experience")

plt.ylabel("Salary")

plt.title("Linear Regression - Salary vs Experience") plt.legend()

plt.grid(True) plt.show()

# Output:

Intercept (b0): 13745.00000000003

Coefficient (b1): 8407.499999999998



B:

import pandas as pd

# Create the dataset data = {

'Supplier ID':

['SP301','SP302','SP303','SP304','SP305','SP306','SP307','SP308','SP309','SP310',

'SP311','SP312','SP313','SP314','SP315','SP316','SP317','SP318','SP319','SP320'],

'Part Number':

['A001','A002','A003','A004','A005','A006','A007','A008','A009','A010',

'A011','A012','A013','A014','A015','A016','A017','A018','A019','A020'],

'Part Name': ['water','alternator','air filter','wheel bearing','muffler','oil pan','brake pads','brake rotors',

'headlight','brake','Strut','Device','CV Boot Kit','Oil Pump','Oil Filter','Fuel

Filter',

'Tie Rod End','Ball Joint','Steering Rack','Piston'],

'Part Price': [6800, 3800, 4500, 3582, 1600, 10005, 6500, 8549, 6500, 1500,

4500, 1580, 2650, 4660, 4350, 1280, 1800, 2500, 2700, 4500],

'Status': ['In', 'In', 'In', 'stock', 'In', 'Out of stock', 'In', 'Out of stock', 'In', 'In',

'In', 'In', 'In', 'In', 'In', 'In', 'stock', 'In', 'Out of stock', 'stock']

}

# Create DataFrame

df = pd.DataFrame(data)

# Query 1: Part Name for Part Number "A002"

part\_name = df.loc[df['Part Number'] == 'A002', 'Part Name'].values[0]

# Query 2: Supplier ID for Part Name "Ball Joint"

supplier\_id = df.loc[df['Part Name'] == 'Ball Joint', 'Supplier ID'].values[0]

# Query 3: Part Price for Part Name "muffler"

part\_price = df.loc[df['Part Name'] == 'muffler', 'Part Price'].values[0]

# Query 4: Status of Part Number "A008"

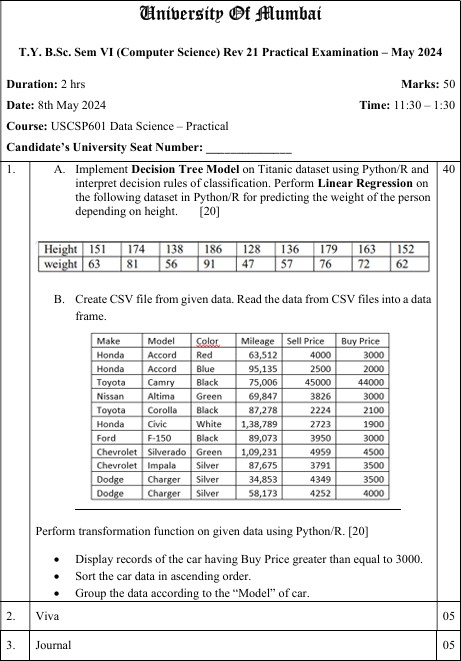
status = df.loc[df['Part Number'] == 'A008', 'Status'].values[0]

# Print results

print("1. Part Name for Part Number 'A002':", part\_name) print("2. Supplier ID for Part Name 'Ball Joint':", supplier\_id) print("3. Part Price for Part Name 'muffler':", part\_price) print("4. Status of Part Number 'A008':", status)

# Output:

1. Part Name for Part Number 'A002': alternator
2. Supplier ID for Part Name 'Ball Joint': SP318
3. Part Price for Part Name 'muffler': 1600
4. Status of Part Number 'A008': Out of stock



* 1. ​
     1. Linear Regression: Predicting Weight from Height import pandas as pd

from sklearn.linear\_model import LinearRegression import matplotlib.pyplot as plt

# Dataset

height = [151, 174, 138, 186, 128, 136, 179, 163, 152]

weight = [63, 81, 56, 91, 47, 57, 76, 72, 62]

# Create DataFrame

df = pd.DataFrame({'Height': height, 'Weight': weight})

# Reshape height for model X = df[['Height']] # Feature y = df['Weight'] # Target

# Linear Regression model model = LinearRegression() model.fit(X, y)

# Predict

df['Predicted\_Weight'] = model.predict(X)

# Print results

print("Intercept:", model.intercept\_) print("Coefficient:", model.coef\_)

# Plot

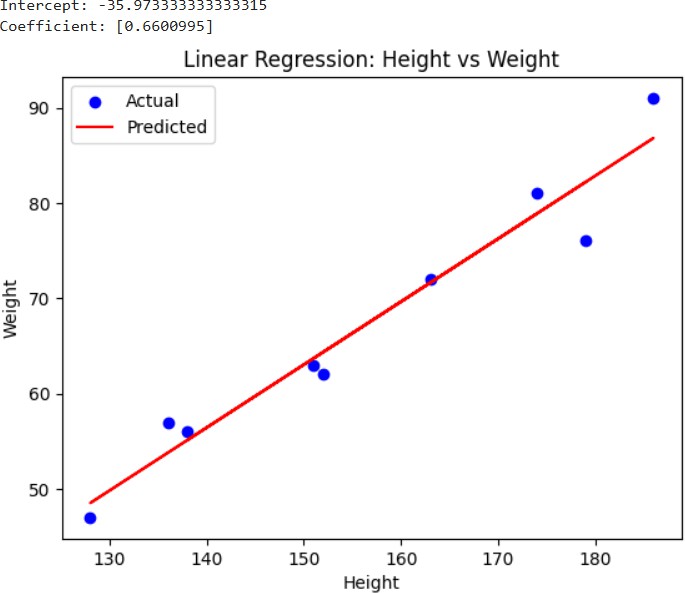
plt.scatter(df['Height'], df['Weight'], color='blue', label='Actual') plt.plot(df['Height'], df['Predicted\_Weight'], color='red', label='Predicted') plt.xlabel("Height")

plt.ylabel("Weight")

plt.title("Linear Regression: Height vs Weight") plt.legend()

plt.show()

Output:



* + 1. Decision Tree on Titanic Dataset import seaborn as sns

from sklearn.tree import DecisionTreeClassifier, plot\_tree from sklearn.model\_selection import train\_test\_split

# Load Titanic dataset

titanic = sns.load\_dataset("titanic")

# Select features and target

features = titanic[['pclass', 'sex', 'age', 'fare']].dropna()

features['sex'] = features['sex'].map({'male': 0, 'female': 1}) target = titanic.loc[features.index, 'survived']

# Train/test split

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

# Decision Tree model

dtree = DecisionTreeClassifier(max\_depth=3)

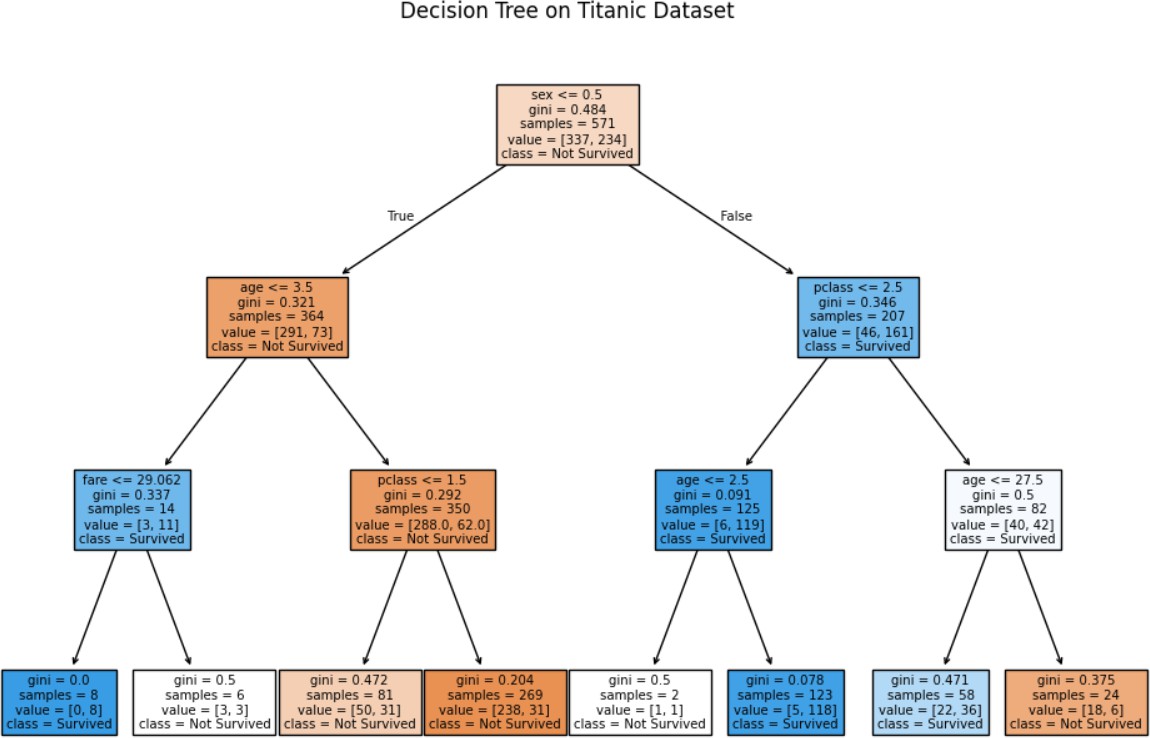
dtree.fit(X\_train, y\_train)

# Plot the tree plt.figure(figsize=(12, 8))

plot\_tree(dtree, feature\_names=['pclass', 'sex', 'age', 'fare'], class\_names=['Not Survived', 'Survived'], filled=True)

plt.title("Decision Tree on Titanic Dataset") plt.show()

# Accuracy

print("Model Accuracy:", dtree.score(X\_test, y\_test)) output:

Model Accuracy: 0.7412587412587412

# B)

import pandas as pd

# Read the CSV file

df = pd.read\_csv('cars.csv')

# 1. Display records where Buy Price >= 3000 filtered\_df = df[df['Buy Price'] >= 3000] print("Records with Buy Price >= 3000:") print(filtered\_df)

# 2. Sort the car data in ascending order (by Buy Price) sorted\_df = df.sort\_values(by='Buy Price')

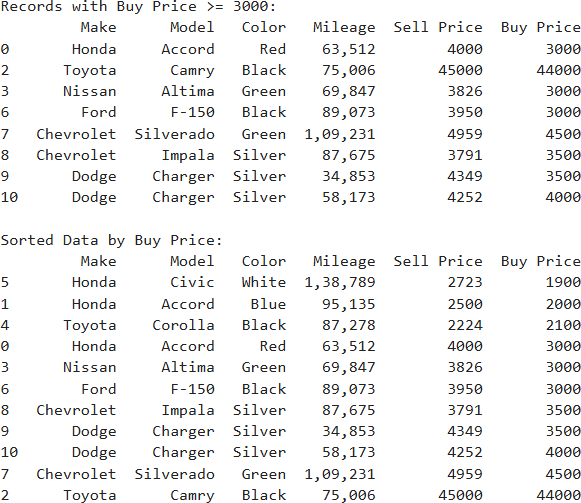
print("\nSorted Data by Buy Price:") print(sorted\_df)

# 3. Group the data by the “Model” of car grouped\_df = df.groupby('Model')

print("\nGrouped Data by Model:") for model, group in grouped\_df:

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

# Output:

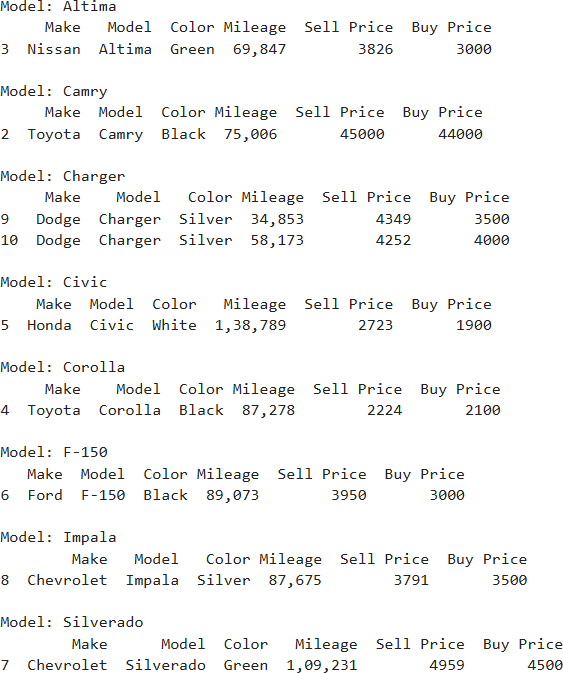


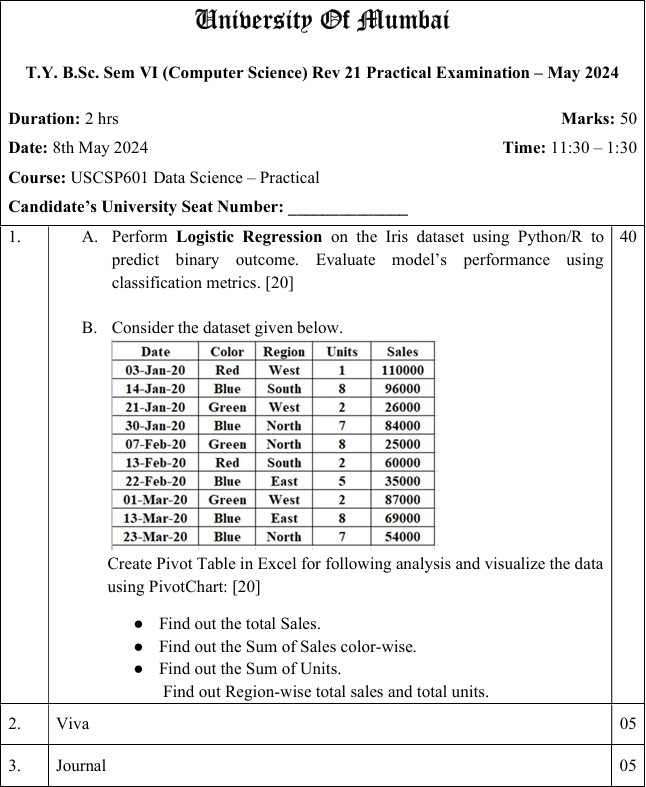
Grouped Data by Model:

Model: Accord

Make Model Color Mileage Sell Price Buy Price

|  |  |  |
| --- | --- | --- |
| 0 Honda Accord Red 63,512 | 4000 | 3000 |
| 1 Honda Accord Blue 95,135 | 2500 | 2000 |





* + - 1. ​

# Import required libraries import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.linear\_model import LogisticRegression from sklearn.model\_selection import train\_test\_split

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

# Load the Iris dataset iris = load\_iris()

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

# Convert to binary classification: Setosa vs Not-Setosa

# Setosa = 0 in original target, so make it binary (1 if Setosa, 0 otherwise) y\_binary = (y == 0).astype(int)

# Split the dataset into training and testing sets

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

# Create and train the Logistic Regression model model = LogisticRegression()

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

# Predict on test data

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

# Evaluate the model print("Confusion Matrix:")

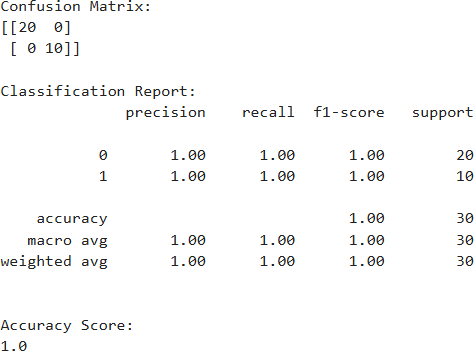
print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

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

print("\nAccuracy Score:")

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



B)

import pandas as pd

# Step 1: Create the dataset as a DataFrame data = {

'Date': ['03-Jan-20', '14-Jan-20', '21-Jan-20', '30-Jan-20', '07-Feb-20',

'13-Feb-20', '22-Feb-20', '01-Mar-20', '13-Mar-20', '23-Mar-20'],

'Color': ['Red', 'Blue', 'Green', 'Blue', 'Green',

'Red', 'Blue', 'Green', 'Blue', 'Blue'],

'Region': ['West', 'South', 'West', 'North', 'North',

'South', 'East', 'West', 'East', 'North'],

'Units': [1, 8, 2, 7, 8, 2, 5, 2, 8, 7],

'Sales': [110000, 96000, 26000, 84000, 25000, 60000, 35000, 87000, 69000,

54000]

}

df = pd.DataFrame(data)

df['Date'] = pd.to\_datetime(df['Date'], format='%d-%b-%y')

# Step 2: Total Sales

total\_sales = df['Sales'].sum() print("Total Sales:", total\_sales)

# Step 3: Sum of Sales Color-wise

sales\_by\_color = df.groupby('Color')['Sales'].sum() print("\nSum of Sales by Color:")

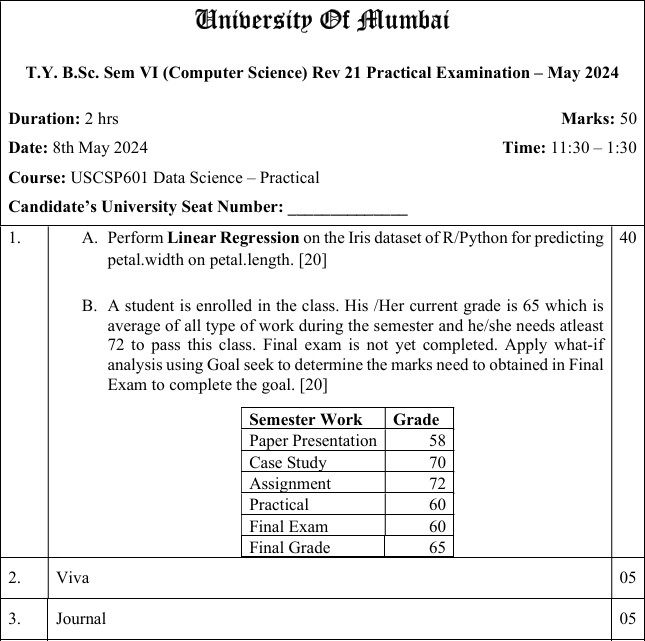
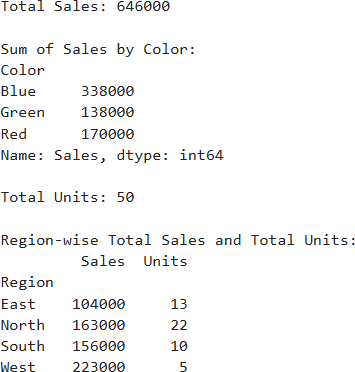
print(sales\_by\_color)

# Step 4: Sum of Units

total\_units = df['Units'].sum() print("\nTotal Units:", total\_units)

# Step 5: Region-wise total sales and total units

region\_summary = df.groupby('Region')[['Sales', 'Units']].sum() print("\nRegion-wise Total Sales and Total Units:")

print(region\_summary) Output:

A)

import numpy as np import pandas as pd

from sklearn.linear\_model import LinearRegression from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import r2\_score, mean\_squared\_error import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

import statsmodels.api as sm # For detailed summary like R

# Step 1: Load Iris dataset iris = load\_iris()

df = pd.DataFrame(data=np.c\_[iris['data'], iris['target']], columns=iris['feature\_names'] + ['target'])

# Step 2: Prepare features and target

X = df[['petal length (cm)']] # Independent variable y = df['petal width (cm)'] # Dependent variable

# Step 3: Add constant term for statsmodels (intercept) X\_sm = sm.add\_constant(X)

# Step 4: Fit the model using statsmodels for detailed summary model\_sm = sm.OLS(y, X\_sm).fit()

# Step 5: Print model summary (equivalent to R's summary()) print("Model Summary:")

print(model\_sm.summary())

# Step 6: Fit sklearn model for predictions and evaluation

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

model\_sk = LinearRegression() model\_sk.fit(X\_train, y\_train)

# Step 7: Make predictions

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

# Step 8: Evaluate model

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

mse = mean\_squared\_error(y\_test, y\_pred) print(f"\nSklearn Model Evaluation:") print(f"R-squared Score: {r2:.4f}")

print(f"Mean Squared Error: {mse:.4f}")

print(f"Coefficient (Slope): {model\_sk.coef\_[0]:.4f}") print(f"Intercept: {model\_sk.intercept\_:.4f}")

plt.scatter(df['petal length (cm)'], df['petal width (cm)'], color='blue', label='Actual Data', marker='o')

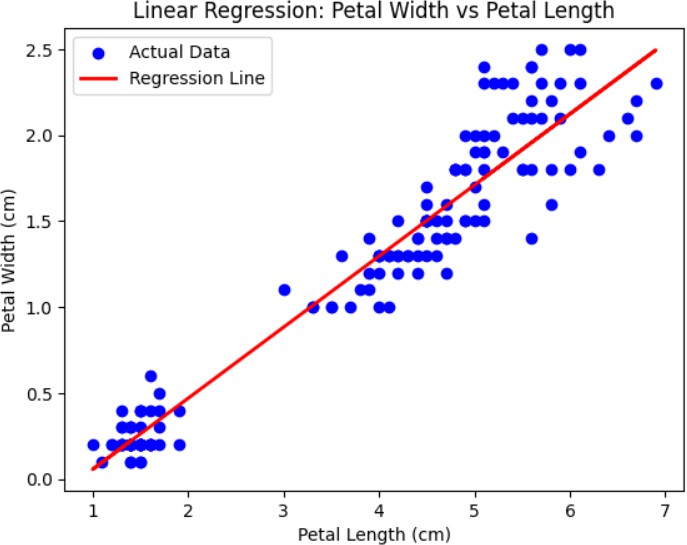
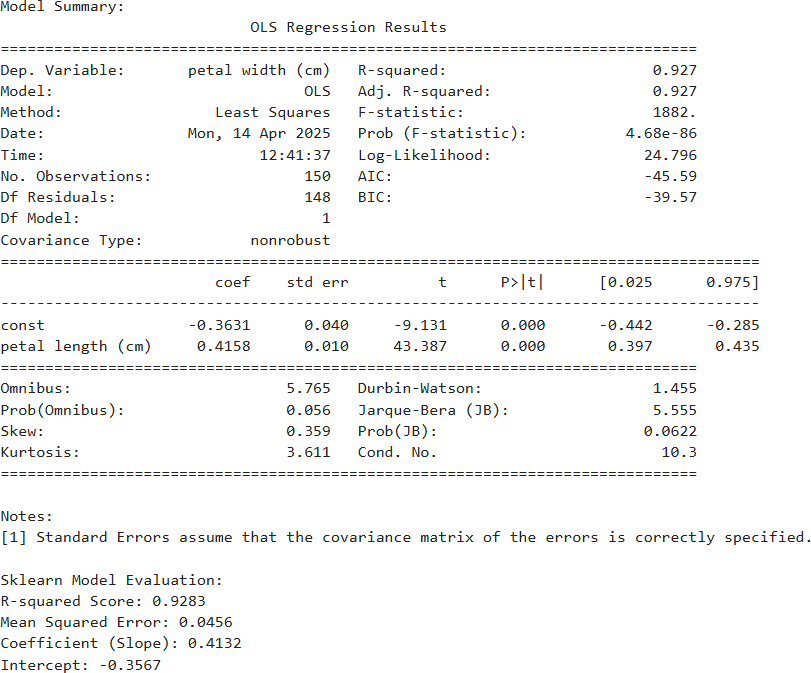
plt.plot(df['petal length (cm)'], model\_sk.predict(X), color='red', label='Regression Line', linewidth=2)

plt.title("Linear Regression: Petal Width vs Petal Length") plt.xlabel("Petal Length (cm)")

plt.ylabel("Petal Width (cm)") plt.legend()

plt.show()

Output:



B)

from scipy.optimize import fsolve

# Given grades for other semester work grades = {

"Paper Presentation": 58,

"Case Study": 70,

"Assignment": 72,

"Practical": 60

}

# Function to find the final exam score that makes the average = 72 def target(final\_exam\_score):

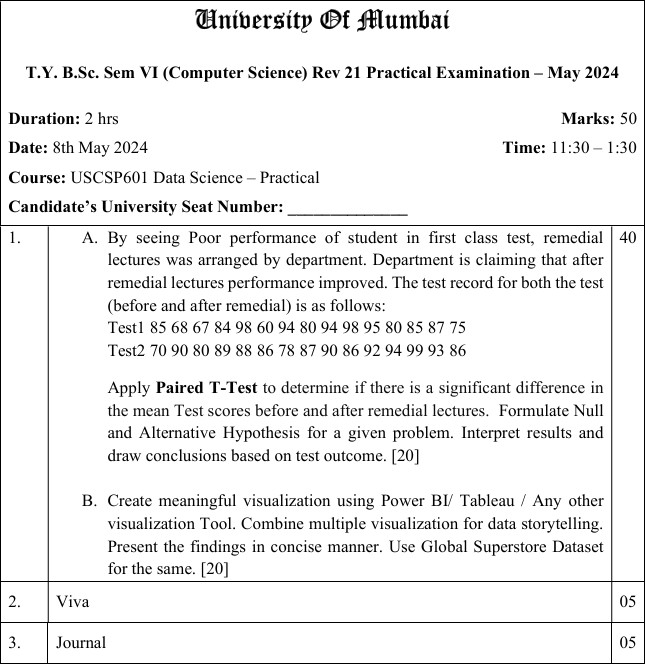
total = sum(grades.values()) + final\_exam\_score avg = total / 5

return avg - 72 # We want this to be zero

# Use fsolve to find the root (final\_exam\_score) from scipy.optimize import fsolve required\_score = fsolve(target, 60)[0]

print(f" Required Final Exam Score to get an average of 72: {required\_score:.2f}") Output:

Required Final Exam Score to get an average of 72: 100.00



A)

from scipy import stats

# Scores before (Test1) and after (Test2) remedial lectures test1 = [85, 68, 67, 84, 98, 60, 94, 80, 94, 98, 95, 80, 85, 87, 75]

test2 = [70, 90, 80, 89, 88, 86, 78, 87, 90, 86, 92, 94, 99, 93, 86]

# Perform paired t-test

t\_statistic, p\_value = stats.ttest\_rel(test2, test1)

print("T-Statistic:", t\_statistic) print("P-Value:", p\_value)

# Interpret the result at alpha = 0.05 alpha = 0.05

if p\_value < alpha:

print("\n Result: Reject the null hypothesis (H₀).")

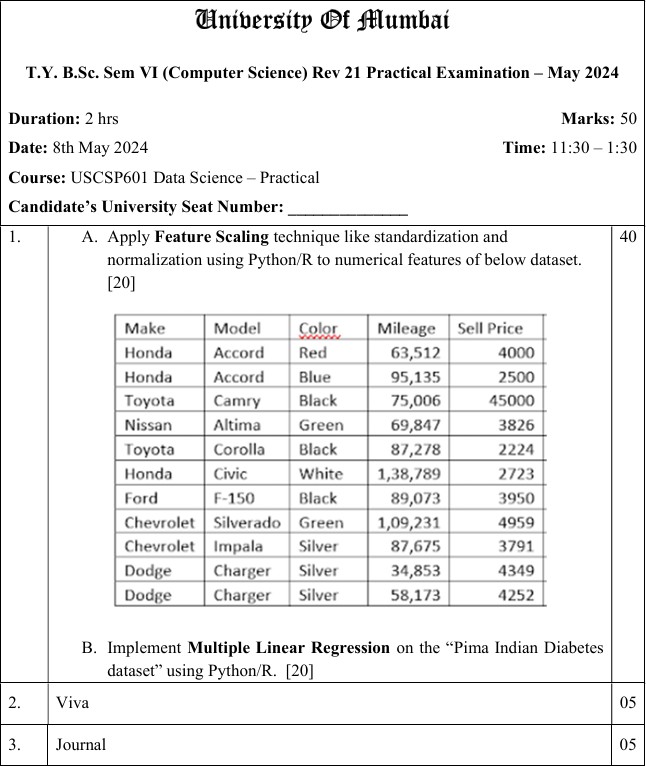
print("Conclusion: There is a significant improvement after remedial lectures.") else:

print("\n Result: Fail to reject the null hypothesis (H₀).") print("Conclusion: No significant improvement observed.")

# Output:

T-Statistic: 1.125690756981246

P-Value: 0.27922443475463854

Result: Fail to reject the null hypothesis (H₀). Conclusion: No significant improvement observed.

A)

import pandas as pd

from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Create DataFrame from the given data data = {

"Make": ["Honda", "Honda", "Toyota", "Nissan", "Toyota", "Honda", "Ford", "Chevrolet", "Chevrolet", "Dodge", "Dodge"],

"Model": ["Accord", "Accord", "Camry", "Altima", "Corolla", "Civic", "F-150", "Silverado", "Impala", "Charger", "Charger"],

"Color": ["Red", "Blue", "Black", "Green", "Black", "White", "Black", "Green", "Silver", "Silver", "Silver"],

"Mileage": [63512, 95135, 75006, 69847, 87278, 138789, 89073, 109231, 87675,

34853, 58173],

"Sell Price": [4000, 2500, 45000, 3826, 2224, 2723, 3950, 4959, 3791, 4349, 4252]

}

df = pd.DataFrame(data)

# Extract only numerical columns

numerical\_features = df[['Mileage', 'Sell Price']]

# 1. Standardization (Z-score) scaler\_std = StandardScaler()

standardized = scaler\_std.fit\_transform(numerical\_features)

df\_standardized = pd.DataFrame(standardized, columns=['Mileage\_Std', 'SellPrice\_Std'])

# 2. Normalization (Min-Max)

scaler\_minmax = MinMaxScaler()

normalized = scaler\_minmax.fit\_transform(numerical\_features)

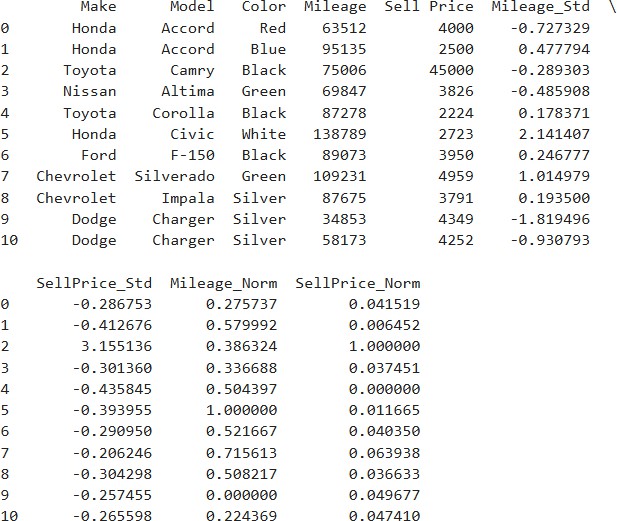
df\_normalized = pd.DataFrame(normalized, columns=['Mileage\_Norm', 'SellPrice\_Norm'])

# Combine all data

df\_combined = pd.concat([df, df\_standardized, df\_normalized], axis=1)

# Display final DataFrame with scaled features print(df\_combined)

Output:



B)

import pandas as pd import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score from sklearn.model\_selection import train\_test\_split

# Load the dataset (you can download it from Kaggle or UCI)

url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians- diabetes.data.csv"

column\_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome']

df = pd.read\_csv(url, names=column\_names)

# Features (excluding Glucose and Outcome) X = df.drop(['Glucose', 'Outcome'], axis=1)

# Target: Predicting Glucose level y = df['Glucose']

# Split dataset into train and test

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

# Fit Multiple Linear Regression model model = LinearRegression()

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

# Predict

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

# Evaluation metrics

mse = mean\_squared\_error(y\_test, y\_pred) r2 = r2\_score(y\_test, y\_pred)

print("Mean Squared Error (MSE):", mse) print("R-squared (R²):", r2)

# Optional: Plot predicted vs actual plt.figure(figsize=(8,5))

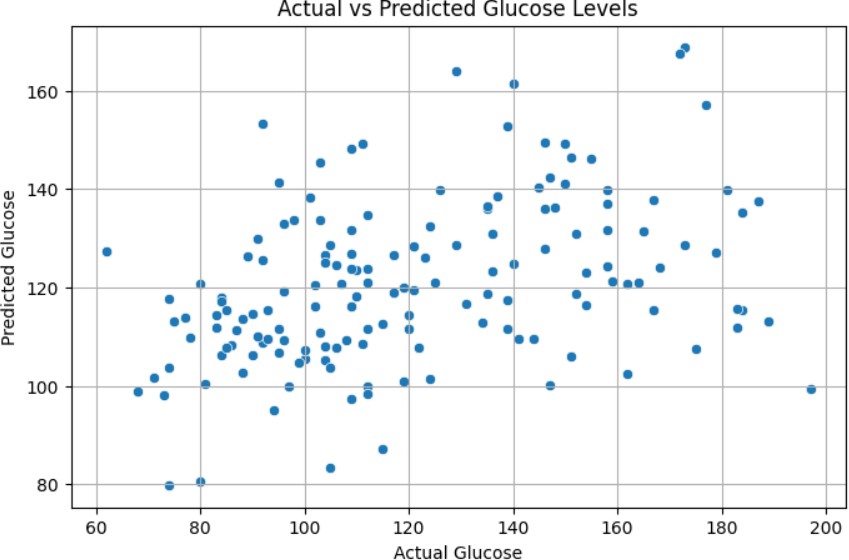
sns.scatterplot(x=y\_test, y=y\_pred) plt.xlabel("Actual Glucose")

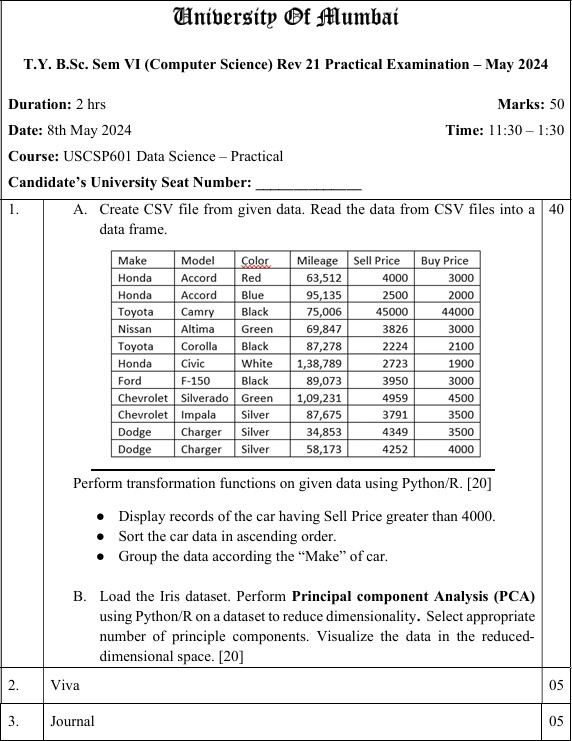
plt.ylabel("Predicted Glucose")

plt.title("Actual vs Predicted Glucose Levels") plt.grid(True)

plt.show() Output:

Mean Squared Error (MSE): 869.4777053181737 R-squared (R²): 0.1359734294261148





A)

import pandas as pd

# Step 1: Create DataFrame data = {

'Make': ['Honda', 'Honda', 'Toyota', 'Nissan', 'Toyota', 'Honda', 'Ford', 'Chevrolet', 'Chevrolet', 'Dodge', 'Dodge'],

'Model': ['Accord', 'Accord', 'Camry', 'Altima', 'Corolla', 'Civic', 'F-150', 'Silverado', 'Impala', 'Charger', 'Charger'],

'Color': ['Red', 'Blue', 'Black', 'Green', 'Black', 'White', 'Black', 'Green', 'Silver', 'Silver', 'Silver'],

'Mileage': [63512, 95135, 75006, 69847, 87278, 138789, 89073, 109231, 87675,

34853, 58173],

'Sell Price': [4000, 2500, 45000, 3826, 2224, 2723, 3950, 4959, 3791, 4349, 4252],

'Buy Price': [3000, 2000, 44000, 3000, 2100, 1900, 3000, 4500, 3500, 3500, 4000]

}

df = pd.DataFrame(data)

# Save to CSV (optional if already saved) df.to\_csv("cars.csv", index=False)

# Step 2: Load from CSV

df = pd.read\_csv("cars.csv")

# Step 3: Display cars with Sell Price > 4000 print("Cars with Sell Price > 4000:") print(df[df['Sell Price'] > 4000])

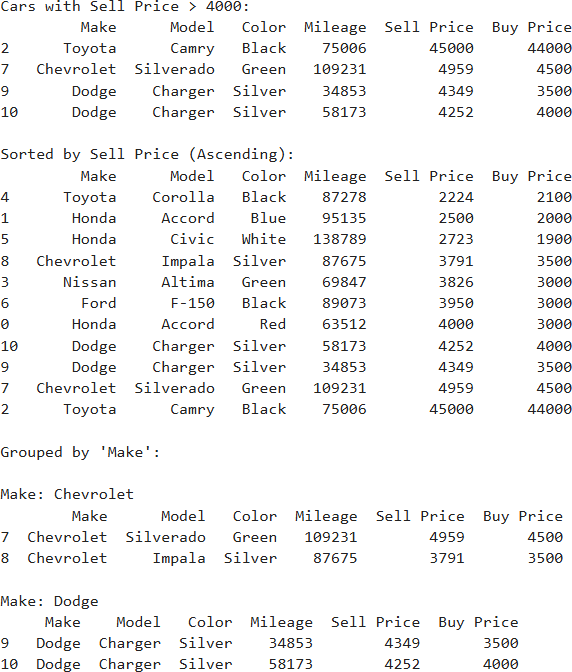
# Step 4: Sort the car data by Sell Price (ascending order) print("\nSorted by Sell Price (Ascending):")

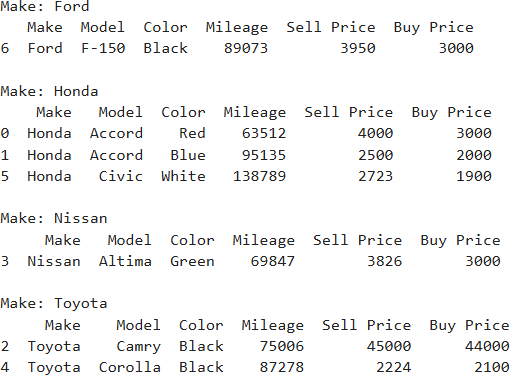
print(df.sort\_values(by='Sell Price'))

# Step 5: Group the data by 'Make' print("\nGrouped by 'Make':") grouped = df.groupby('Make')

for name, group in grouped: print(f"\nMake: {name}") print(group)

Output:





B)

import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler from sklearn.decomposition import PCA

# Step 1: Load the Iris dataset iris = load\_iris()

X = iris.data y = iris.target

feature\_names = iris.feature\_names target\_names = iris.target\_names

# Step 2: Standardize the features scaler = StandardScaler()

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

# Step 3: Apply PCA

pca = PCA()

X\_pca = pca.fit\_transform(X\_scaled)

# Step 4: Explained variance ratio

explained\_variance = pca.explained\_variance\_ratio\_ print("Explained variance ratio:", explained\_variance)

# Step 5: Scree plot to choose number of components plt.figure(figsize=(8, 5))

plt.plot(range(1, len(explained\_variance) + 1), explained\_variance, 'o-', color='green') plt.title('Scree Plot')

plt.xlabel('Principal Component') plt.ylabel('Variance Explained') plt.grid(True)

plt.show()

# Step 6: Use top 2 components for visualization pca\_2 = PCA(n\_components=2)

X\_reduced = pca\_2.fit\_transform(X\_scaled)

# Step 7: Visualize the 2D PCA result

df\_pca = pd.DataFrame(X\_reduced, columns=['PC1', 'PC2']) df\_pca['Target'] = y

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

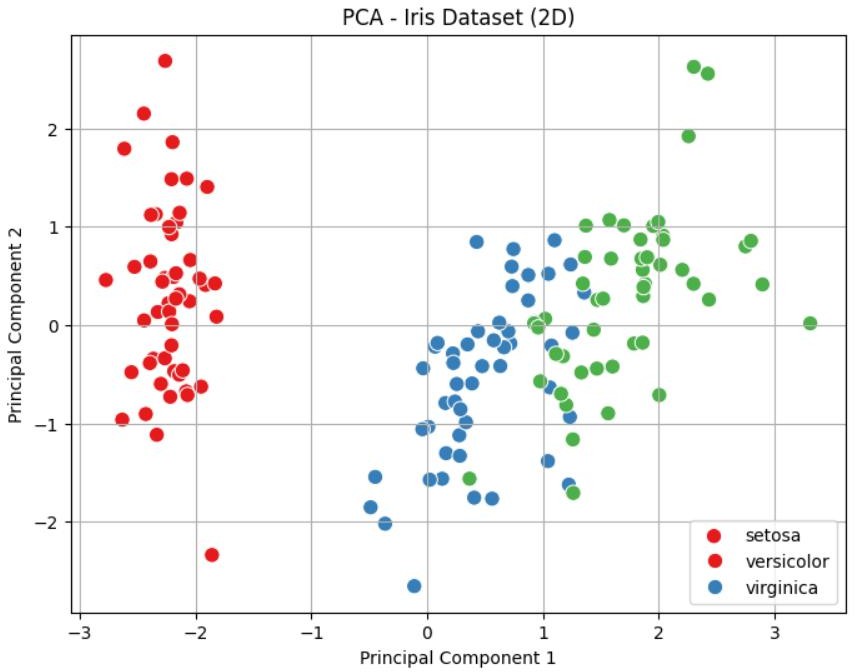
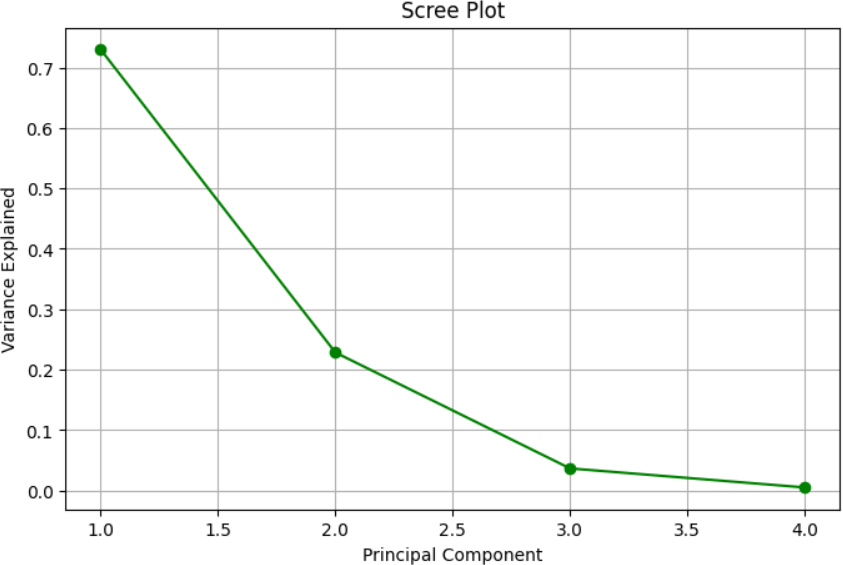
sns.scatterplot(data=df\_pca, x='PC1', y='PC2', hue='Target', palette='Set1', s=80) plt.title('PCA - Iris Dataset (2D)')

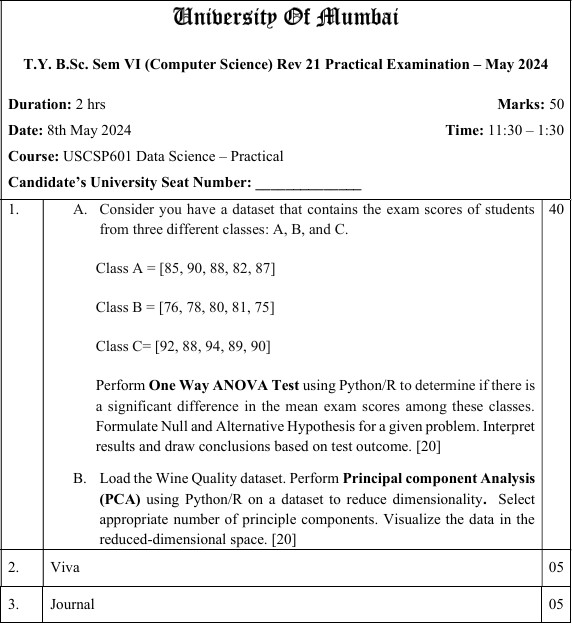
plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2') plt.legend(labels=target\_names) plt.grid(True)

plt.show() Output:

Explained variance ratio: [0.72962445 0.22850762 0.03668922 0.00517871]





A)

import scipy.stats as stats

# Exam scores

class\_A = [85, 90, 88, 82, 87]

class\_B = [76, 78, 80, 81, 75]

class\_C = [92, 88, 94, 89, 90]

# Perform One-Way ANOVA

f\_statistic, p\_value = stats.f\_oneway(class\_A, class\_B, class\_C)

print(f"F-statistic: {f\_statistic:.4f}") print(f"P-value: {p\_value:.4f}")

# Interpret result alpha = 0.05

if p\_value < alpha:

print("Result: Reject the null hypothesis (significant difference exists).") else:

print("Result: Fail to reject the null hypothesis (no significant difference).") Output:

F-statistic: 28.5833

P-value: 0.0000

Result: Reject the null hypothesis (significant difference exists). B)

import pandas as pd import numpy as np

from sklearn.datasets import load\_wine

from sklearn.preprocessing import StandardScaler from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

# Step 1: Load the Wine Quality dataset

# Using the wine dataset from sklearn as a proxy (red wine quality variant) wine = load\_wine()

df = pd.DataFrame(data=wine.data, columns=wine.feature\_names) df['target'] = wine.target

# Step 2: Standardize the data (PCA requires zero mean and unit variance) scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(df.drop('target', axis=1))

# Step 3: Perform PCA pca = PCA()

X\_pca = pca.fit\_transform(X\_scaled)

# Step 4: Determine the appropriate number of principal components explained\_variance\_ratio = pca.explained\_variance\_ratio\_ cumulative\_explained\_variance = np.cumsum(explained\_variance\_ratio)

# Select number of components explaining at least 95% of variance n\_components = np.argmax(cumulative\_explained\_variance >= 0.95) + 1

print(f"Number of components explaining at least 95% variance: {n\_components}") print(f"Cumulative Explained Variance Ratio: {cumulative\_explained\_variance}")

# Step 5: Visualize explained variance ratio plt.figure(figsize=(10, 6))

plt.plot(range(1, len(explained\_variance\_ratio) + 1), cumulative\_explained\_variance, marker='o')

plt.xlabel('Number of Components')

plt.ylabel('Cumulative Explained Variance Ratio')

plt.title('Explained Variance Ratio by Number of Principal Components') plt.axhline(y=0.95, color='r', linestyle='--', label='95% Variance Threshold') plt.legend()

plt.grid() plt.show()

# Step 6: Reduce dimensionality to the selected number of components pca = PCA(n\_components=n\_components)

X\_pca\_reduced = pca.fit\_transform(X\_scaled)

# Step 7: Visualize the data in reduced-dimensional space plt.figure(figsize=(10, 6))

scatter = plt.scatter(X\_pca\_reduced[:, 0], X\_pca\_reduced[:, 1], c=df['target'], cmap='viridis')

plt.xlabel('First Principal Component') plt.ylabel('Second Principal Component')

plt.title('Wine Quality Data in Reduced 2D Space (PCA)') plt.colorbar(scatter, label='Wine Class')

plt.show() Output:

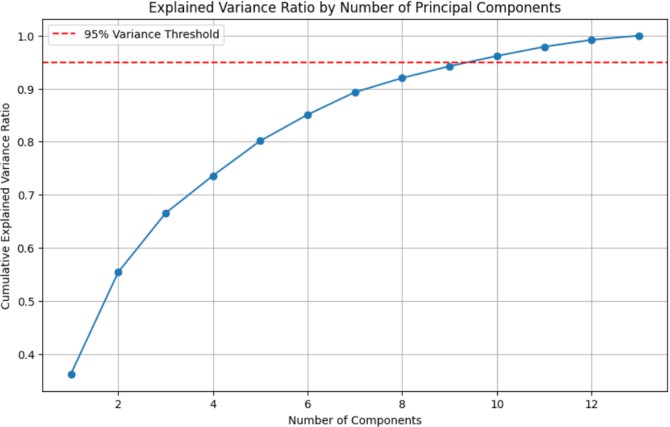
Number of components explaining at least 95% variance: 10

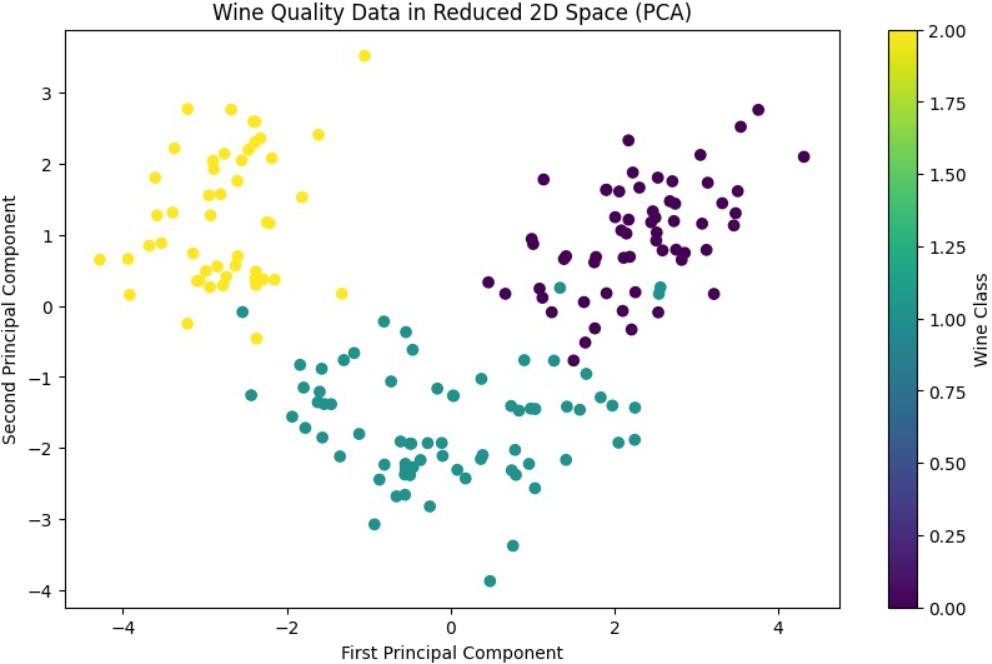
Cumulative Explained Variance Ratio: [0.36198848 0.55406338 0.66529969

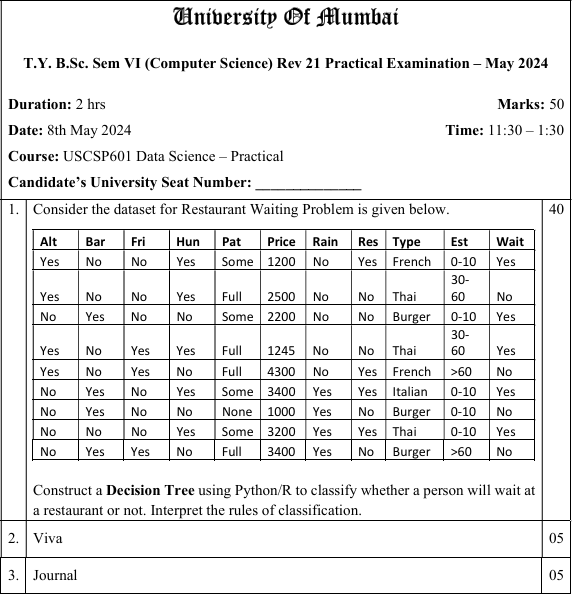
0.73598999 0.80162293 0.85098116

0.89336795 0.92017544 0.94239698 0.96169717 0.97906553 0.99204785

1. ]







Input:

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier, export\_text from sklearn import tree

import matplotlib.pyplot as plt

# Step 1: Create the dataset data = {

'Alt': ['Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'No'],

'Bar': ['No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes'],

'Fri': ['No', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes'],

'Hun': ['Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes'],

'Pat': ['Some', 'Full', 'Some', 'Full', 'Full', 'Some', 'None', 'Some', 'Some', 'Full'], 'Price': [1200, 2500, 2200, 1245, 4300, 3400, 1000, 3200, 1000, 3400],

'Rain': ['No', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes'],

'Res': ['Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'No', 'No'],

'Type': ['French', 'Thai', 'Burger', 'Thai', 'French', 'Italian', 'Burger', 'Thai', 'Burger', 'Burger'],

'Est': ['0-10', '30-60', '0-10', '30-60', '>60', '10-30', '0-10', '0-10', '0-10', '>60'],

'Wait': ['Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No']

}

df = pd.DataFrame(data)

# Step 2: Encode categorical variables le = LabelEncoder()

for col in df.columns:

df[col] = le.fit\_transform(df[col])

# Step 3: Split features and target X = df.drop('Wait', axis=1)

y = df['Wait']

# Step 4: Build the Decision Tree

clf = DecisionTreeClassifier(criterion='entropy', random\_state=0) clf = clf.fit(X, y)

# Step 5: Visualize the tree plt.figure(figsize=(15, 10))

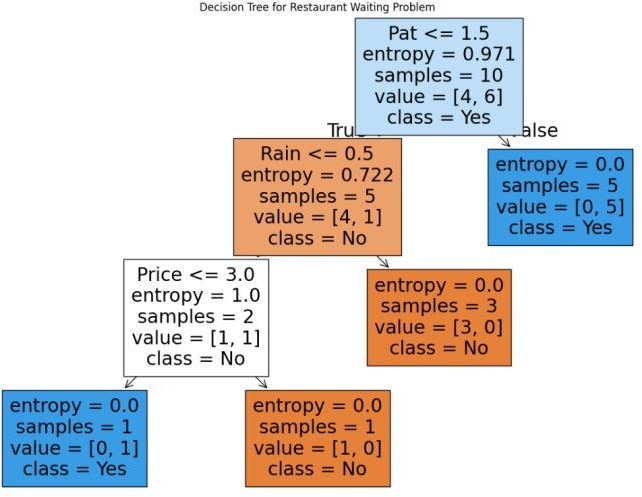
tree.plot\_tree(clf, feature\_names=X.columns, class\_names=['No', 'Yes'], filled=True) plt.title("Decision Tree for Restaurant Waiting Problem")

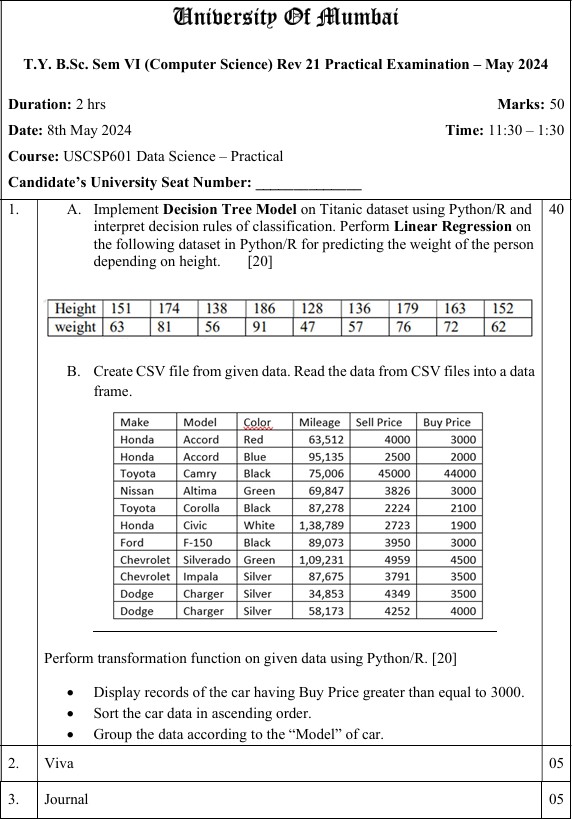
plt.show()

# Step 6: Print human-readable rules

rules = export\_text(clf, feature\_names=list(X.columns)) print("\nClassification Rules:\n")

print(rules) Output:





A)

1: Decision Tree Model on Titanic Dataset

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier, plot\_tree, export\_text import matplotlib.pyplot as plt

# Load Titanic dataset from seaborn (or from CSV if needed) import seaborn as sns

titanic = sns.load\_dataset('titanic')

# Select useful features and drop missing values

df = titanic[['pclass', 'sex', 'age', 'fare', 'embarked', 'survived']].dropna()

# Encode categorical variables

df['sex'] = df['sex'].map({'male': 0, 'female': 1})

df['embarked'] = df['embarked'].map({'S': 0, 'C': 1, 'Q': 2})

# Split data

X = df.drop('survived', axis=1) y = df['survived']

# Train model

clf = DecisionTreeClassifier(max\_depth=4, criterion='entropy', random\_state=0) clf.fit(X, y)

# Plot tree plt.figure(figsize=(15, 10))

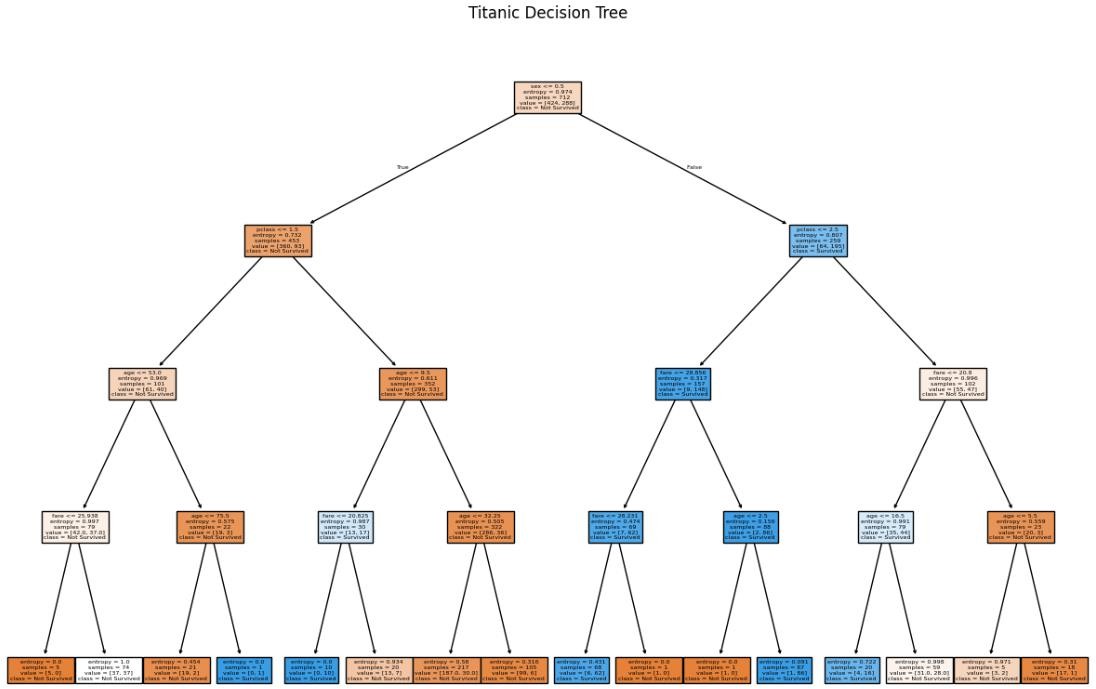
plot\_tree(clf, feature\_names=X.columns, class\_names=['Not Survived', 'Survived'], filled=True)

plt.title("Titanic Decision Tree") plt.show()

# Print decision rules

print("\nDecision Tree Rules:")

print(export\_text(clf, feature\_names=list(X.columns))) Output:



2: Linear Regression on Height vs Weight import numpy as np

from sklearn.linear\_model import LinearRegression import matplotlib.pyplot as plt

# Given height and weight data

height = np.array([151, 174, 138, 186, 128, 136, 179, 163, 152]).reshape(-1, 1)

weight = np.array([63, 81, 56, 91, 47, 57, 76, 72, 62])

# Train linear regression model

model = LinearRegression() model.fit(height, weight)

# Predict

predicted\_weight = model.predict(height)

# Plotting

plt.scatter(height, weight, color='blue', label='Actual')

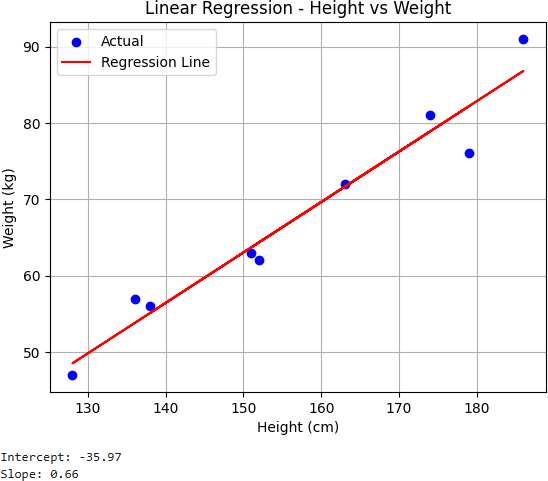
plt.plot(height, predicted\_weight, color='red', label='Regression Line') plt.xlabel("Height (cm)")

plt.ylabel("Weight (kg)")

plt.title("Linear Regression - Height vs Weight") plt.legend()

plt.grid(True) plt.show()

# Print model coefficients

print(f"Intercept: {model.intercept\_:.2f}") print(f"Slope: {model.coef\_[0]:.2f}") Output:

B)

import pandas as pd

# Step 1: Create the DataFrame from the given data data = {

'Make': ['Honda', 'Honda', 'Toyota', 'Nissan', 'Toyota', 'Honda', 'Ford', 'Chevrolet', 'Chevrolet', 'Dodge', 'Dodge'],

'Model': ['Accord', 'Accord', 'Camry', 'Altima', 'Corolla', 'Civic', 'F-150', 'Silverado', 'Impala', 'Charger', 'Charger'],

'Color': ['Red', 'Blue', 'Black', 'Green', 'Black', 'White', 'Black', 'Green', 'Silver', 'Silver', 'Silver'],

'Mileage': [63512, 95135, 75006, 69847, 87278, 138789, 89073, 109231, 87675,

34853, 58173],

'Sell Price': [4000, 2500, 45000, 3826, 2224, 2723, 3950, 4959, 3791, 4349, 4252],

'Buy Price': [3000, 2000, 44000, 3000, 2100, 1900, 3000, 4500, 3500, 3500, 4000]

}

df = pd.DataFrame(data)

# Step 2: Display records with Buy Price >= 3000 filtered\_df = df[df['Buy Price'] >= 3000]

print("Cars with Buy Price >= 3000:") print(filtered\_df)

# Step 3: Sort the data in ascending order by 'Buy Price' sorted\_df = df.sort\_values(by='Buy Price')

print("\nSorted Car Data by Buy Price:") print(sorted\_df)

# Step 4: Group the data according to the 'Model' of the car grouped = df.groupby('Model').agg({

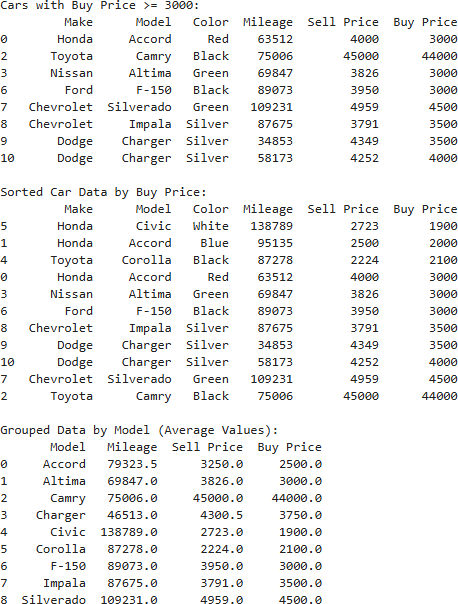
'Mileage': 'mean',

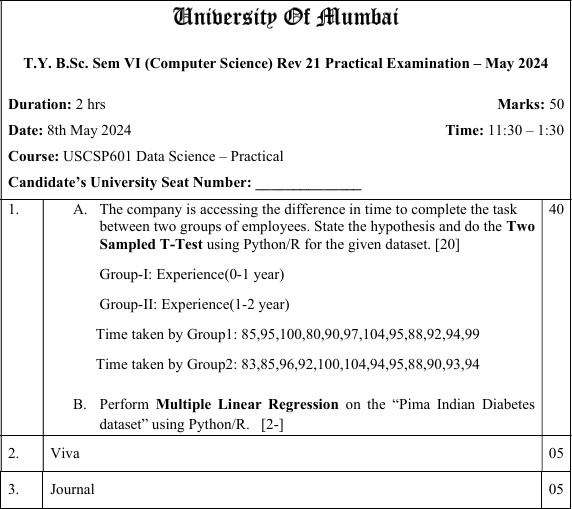
'Sell Price': 'mean', 'Buy Price': 'mean'

}).reset\_index()

print("\nGrouped Data by Model (Average Values):") print(grouped)

Output:





A)

from scipy.stats import ttest\_ind

# Task times for Group 1 (0-1 year experience)

group1 = [85, 95, 100, 80, 90, 97, 104, 95, 88, 92, 94, 99]

# Task times for Group 2 (1-2 years experience)

group2 = [83, 85, 96, 92, 100, 104, 94, 95, 88, 90, 93, 94]

# Perform two-sample independent t-test t\_stat, p\_value = ttest\_ind(group1, group2)

# Print results

print(f"T-statistic: {t\_stat:.4f}") print(f"P-value: {p\_value:.4f}")

# Interpretation alpha = 0.05

if p\_value < alpha:

print("Reject the Null Hypothesis: Significant difference between the groups.") else:

print("Fail to Reject the Null Hypothesis: No significant difference between the groups.")

Output:

T-statistic: 0.1612

P-value: 0.8734

Fail to Reject the Null Hypothesis: No significant difference between the groups. B)

import pandas as pd import numpy as np

import matplotlib.pyplot as plt import seaborn as sns

from sklearn.linear\_model import LinearRegression from sklearn.model\_selection import train\_test\_split

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

# Step 1: Load the dataset

# You can also download from a CSV if needed. Here's how to load it from seaborn url = "https://raw.githubusercontent.com/plotly/datasets/master/diabetes.csv"

df = pd.read\_csv(url)

# Show first few rows print(df.head())

# Step 2: Define features and target

X = df.drop('Outcome', axis=1) # Independent variables

y = df['Outcome'] # Dependent variable (can be considered a regression problem for demonstration)

# Step 3: Train/test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Step 4: Build and train the model model = LinearRegression()

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

# Step 5: Make predictions y\_pred = model.predict(X\_test)

# Step 6: Evaluate the model print("\nModel Coefficients:")

for col, coef in zip(X.columns, model.coef\_): print(f"{col}: {coef:.4f}")

print(f"\nIntercept: {model.intercept\_:.4f}") print(f"R² Score: {r2\_score(y\_test, y\_pred):.4f}")

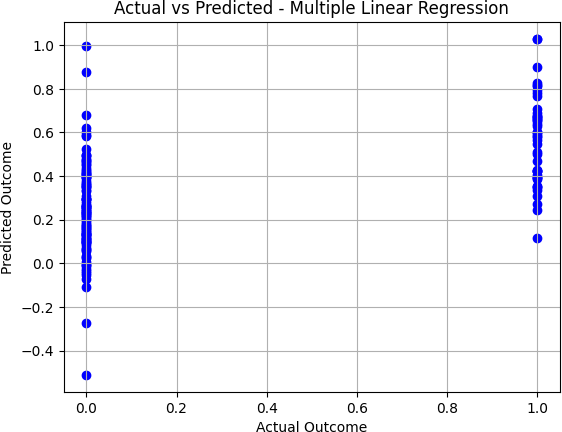
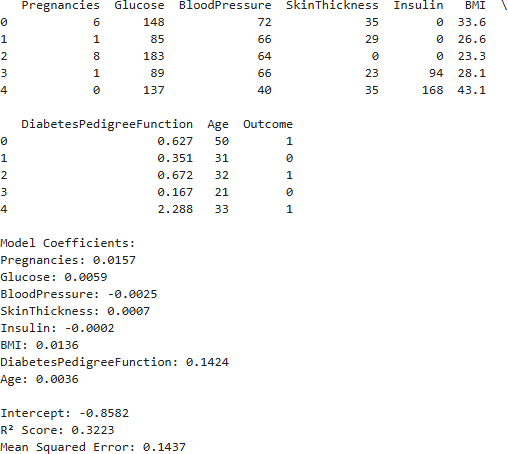
print(f"Mean Squared Error: {mean\_squared\_error(y\_test, y\_pred):.4f}") plt.scatter(y\_test, y\_pred, color='blue')

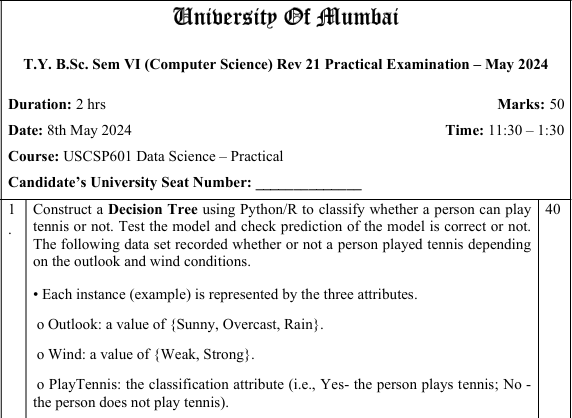
plt.xlabel("Actual Outcome") plt.ylabel("Predicted Outcome")

plt.title("Actual vs Predicted - Multiple Linear Regression") plt.grid(True)

plt.show()

Output:





Input:

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier, plot\_tree from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score

# Step 1: Create the dataset data = {

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

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Strong', 'Weak', 'Strong', 'Weak', 'Strong', 'Strong'],

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

}

df = pd.DataFrame(data)

# Step 2: Encode categorical features le\_outlook = LabelEncoder()

le\_wind = LabelEncoder() le\_play = LabelEncoder()

df['Outlook\_encoded'] = le\_outlook.fit\_transform(df['Outlook']) df['Wind\_encoded'] = le\_wind.fit\_transform(df['Wind'])

df['PlayTennis\_encoded'] = le\_play.fit\_transform(df['PlayTennis'])

# Features and Target

X = df[['Outlook\_encoded', 'Wind\_encoded']] y = df['PlayTennis\_encoded']

# Step 3: Train/Test split

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

# Step 4: Train Decision Tree model

clf = DecisionTreeClassifier(criterion='entropy', random\_state=0) clf.fit(X\_train, y\_train)

# Step 5: Predict and evaluate y\_pred = clf.predict(X\_test)

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

print(f"Predictions: {le\_play.inverse\_transform(y\_pred)}") print(f"Actual: {le\_play.inverse\_transform(y\_test)}") print(f"Accuracy: {accuracy:.2f}")

# Step 6: Visualize the tree import matplotlib.pyplot as plt

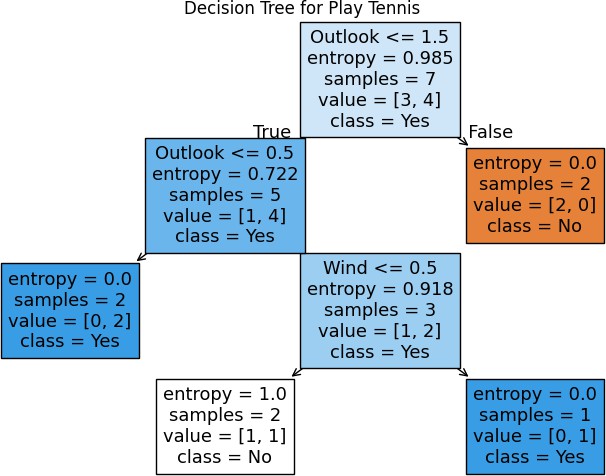
plt.figure(figsize=(10,6))

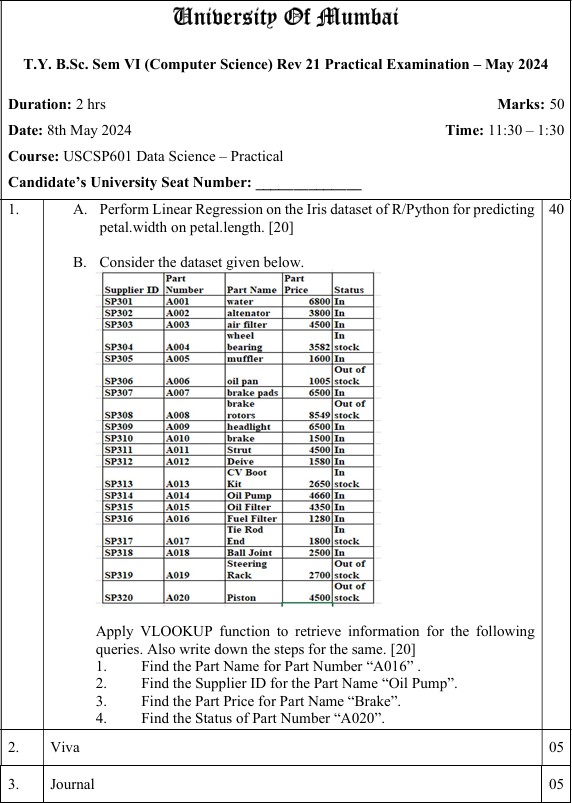
plot\_tree(clf, feature\_names=['Outlook', 'Wind'], class\_names=le\_play.classes\_, filled=True)

plt.title("Decision Tree for Play Tennis") plt.show()

Output:

Predictions: ['No' 'No' 'Yes']

Actual: ['Yes' 'No' 'Yes'] Accuracy: 0.67



A)

import numpy as np import pandas as pd

from sklearn.linear\_model import LinearRegression from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import r2\_score, mean\_squared\_error import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

# Step 1: Load the Iris dataset iris = load\_iris()

df = pd.DataFrame(data=np.c\_[iris['data'], iris['target']], columns=iris['feature\_names'] + ['target'])

# Step 2: Prepare features and target

X = df[['petal length (cm)']] # Independent variable y = df['petal width (cm)'] # Dependent variable

# Step 3: Split 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)

# Step 4: Train Linear Regression model model = LinearRegression()

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

# Step 5: Make predictions y\_pred = model.predict(X\_test)

# Step 6: Evaluate model

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

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

print(f"R-squared Score: {r2:.4f}") print(f"Mean Squared Error: {mse:.4f}")

print(f"Coefficient (Slope): {model.coef\_[0]:.4f}") print(f"Intercept: {model.intercept\_:.4f}")

# Step 7: Predict petal width for a new petal length (e.g., 5 cm) new\_length = np.array([[5.0]])

predicted\_width = model.predict(new\_length)

print(f"Predicted Petal Width for Petal Length 5.0 cm: {predicted\_width[0]:.2f} cm")

# Step 8: Visualize the regression line

plt.scatter(X, y, color='blue', label='Actual Data')

plt.plot(X, model.predict(X), color='red', label='Regression Line') plt.xlabel('Petal Length (cm)')

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

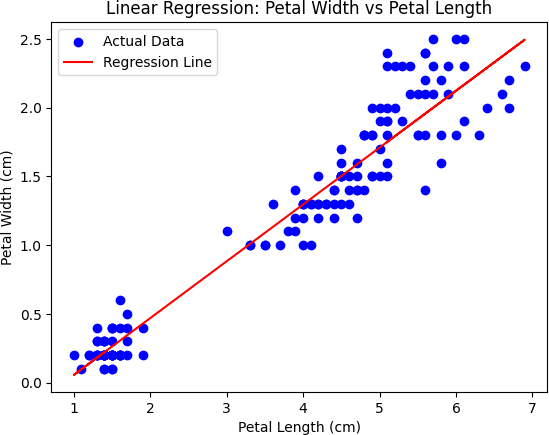
plt.title('Linear Regression: Petal Width vs Petal Length') plt.legend()

plt.show() Output:

R-squared Score: 0.9283 Mean Squared Error: 0.0456 Coefficient (Slope): 0.4132

Intercept: -0.3567

Predicted Petal Width for Petal Length 5.0 cm: 1.71 cm



B)

import pandas as pd

# Step 1: Create DataFrame from the given dataset data = {

'Supplier ID': ['SP301', 'SP302', 'SP304', 'SP305', 'SP306', 'SP307', 'SP308', 'SP309', 'SP310', 'SP311',

'SP312', 'SP313', 'SP314', 'SP315', 'SP316', 'SP317', 'SP318', 'SP319',

'SP320'],

'Part Number': ['A001', 'A002', 'A004', 'A005', 'A006', 'A007', 'A008', 'A009', 'A010', 'A011',

'A012', 'A013', 'A014', 'A015', 'A016', 'A017', 'A018', 'A019', 'A020'],

'Part Name': ['water alternator', 'air filter', 'wheel bearing', 'muffler', 'oil pan', 'brake pads',

'headlight', 'brake rotor', 'strut', 'Drive belt', 'CV Boot', 'Oil Pump', 'Fuel

Filter',

'Tie Rod', 'End Joint', 'Ball Joint', 'Steering Rack', 'Rack and Pinion', 'Piston'], 'Price': [6800, 4500, 3582, 1600, 1005, 6500, 8500, 6500, 1500, 4500, 1580, 2650,

4350, 1280,

1800, 2500, 2700, 4500, 4500],

'Status': ['In', 'In', 'In stock', 'In', 'Out of stock', 'Out of stock', 'In', 'In', 'In', 'In',

'In', 'In stock', 'In', 'In', 'In stock', 'In', 'Out of stock', 'Out of stock', 'Out of

stock']

}

df = pd.DataFrame(data)

# Step 2: Define a VLOOKUP-like function

def vlookup(key\_column, lookup\_value, return\_column): if lookup\_value in df[key\_column].values:

result = df[df[key\_column] == lookup\_value][return\_column].iloc[0] return result

else:

return "Not found"

# Step 3: Answer the queries print("Query Results:")

print("1. Find the Part Name for Part Number 'A016':")

part\_name\_a016 = vlookup('Part Number', 'A016', 'Part Name') print(f"Part Name: {part\_name\_a016}")

print("\n2. Find the Supplier ID for the Part Number 'A016':") supplier\_id\_a016 = vlookup('Part Number', 'A016', 'Supplier ID') print(f"Supplier ID: {supplier\_id\_a016}")

print("\n3. Find the Part Price for Part Name 'Oil Pump':") price\_oil\_pump = vlookup('Part Name', 'Oil Pump', 'Price') print(f"Part Price: ${price\_oil\_pump}")

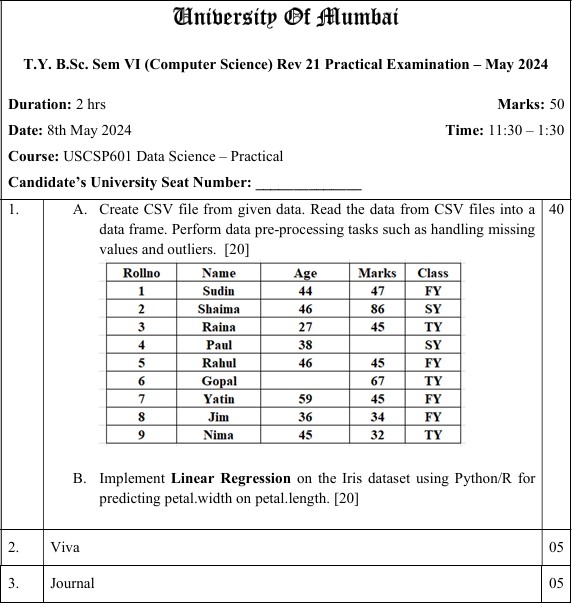
print("\n4. Find the Status of Part Number 'A020':") status\_a020 = vlookup('Part Number', 'A020', 'Status') print(f"Status: {status\_a020}")

Output:

Query Results:

1. Find the Part Name for Part Number 'A016': Part Name: End Joint
2. Find the Supplier ID for the Part Number 'A016': Supplier ID: SP316
3. Find the Part Price for Part Name 'Oil Pump': Part Price: $2650
4. Find the Status of Part Number 'A020':

Status: Out of stock



* 1. ​

import pandas as pd import numpy as np

# Step 1: Create DataFrame from given data with missing values data = {

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

'Name': ['Sudin', 'Shaima', 'Raina', 'Paul', 'Rahul', 'Gopal', 'Yatin', 'Jim', 'Nima'], 'Age': [44, np.nan, 27, 38, 46, np.nan, 59, 36, 45],

'Marks': [47, 86, 45, np.nan, 45, 67, 45, 34, 32], 'Class': ['FY', 'SY', 'TY', 'FY', 'FY', 'TY', 'FY', 'FY', 'TY']

}

df = pd.DataFrame(data)

# Step 2: Save to CSV file

df.to\_csv('student\_data.csv', index=False)

print("CSV file 'student\_data.csv' created successfully.")

# Step 3: Read data from CSV file into a DataFrame df\_read = pd.read\_csv('student\_data.csv')

print("\nData read from CSV:") print(df\_read)

# Step 4: Handle missing values # Fill missing Age with median

df\_read['Age'] = df\_read['Age'].fillna(df\_read['Age'].median()) # Fill missing Marks with mean

df\_read['Marks'] = df\_read['Marks'].fillna(df\_read['Marks'].mean().round(2))

print("\nData after handling missing values:") print(df\_read)

# Step 5: Detect and handle outliers using IQR method def detect\_outliers(df, column):

Q1 = df[column].quantile(0.25) Q3 = df[column].quantile(0.75)

IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR upper\_bound = Q3 + 1.5 \* IQR

outliers = df[(df[column] < lower\_bound) | (df[column] > upper\_bound)][column] return outliers

# Check outliers in Age

age\_outliers = detect\_outliers(df\_read, 'Age')

print(f"\nOutliers in Age: {age\_outliers.tolist() if not age\_outliers.empty else 'None'}")

# Check outliers in Marks

marks\_outliers = detect\_outliers(df\_read, 'Marks')

print(f"Outliers in Marks: {marks\_outliers.tolist() if not marks\_outliers.empty else 'None'}")

# Handle outliers by capping them at bounds for column in ['Age', 'Marks']:

Q1 = df\_read[column].quantile(0.25) Q3 = df\_read[column].quantile(0.75)

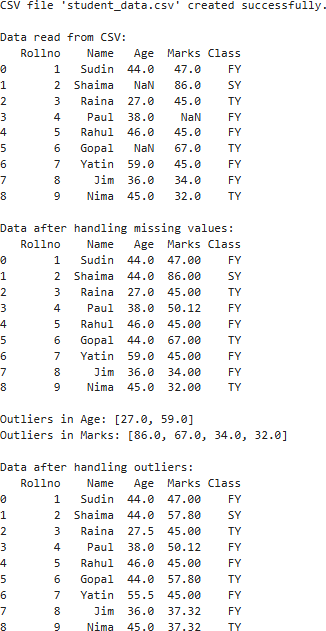
IQR = Q3 - Q1

lower\_bound = Q1 - 1.5 \* IQR upper\_bound = Q3 + 1.5 \* IQR

df\_read[column] = df\_read[column].clip(lower\_bound, upper\_bound)

print("\nData after handling outliers:") print(df\_read)

Output:



B)

import numpy as np import pandas as pd

from sklearn.linear\_model import LinearRegression from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import r2\_score, mean\_squared\_error import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

# Step 1: Load the Iris dataset iris = load\_iris()

df = pd.DataFrame(data=np.c\_[iris['data'], iris['target']], columns=iris['feature\_names'] + ['target'])

# Step 2: Prepare features and target

X = df[['petal length (cm)']] # Independent variable y = df['petal width (cm)'] # Dependent variable

# Step 3: Split 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)

# Step 4: Train Linear Regression model model = LinearRegression()

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

# Step 5: Make predictions y\_pred = model.predict(X\_test)

# Step 6: Evaluate model

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

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

print(f"R-squared Score: {r2:.4f}") print(f"Mean Squared Error: {mse:.4f}")

print(f"Coefficient (Slope): {model.coef\_[0]:.4f}") print(f"Intercept: {model.intercept\_:.4f}")

# Step 7: Predict petal width for a new petal length (e.g., 5 cm) new\_length = np.array([[5.0]])

predicted\_width = model.predict(new\_length)

print(f"Predicted Petal Width for Petal Length 5.0 cm: {predicted\_width[0]:.2f} cm")

# Step 8: Visualize the regression line

plt.scatter(X, y, color='blue', label='Actual Data')

plt.plot(X, model.predict(X), color='red', label='Regression Line') plt.xlabel('Petal Length (cm)')

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

plt.title('Linear Regression: Petal Width vs Petal Length') plt.legend()

plt.show() Output:

R-squared Score: 0.9283 Mean Squared Error: 0.0456 Coefficient (Slope): 0.4132

Intercept: -0.3567

Predicted Petal Width for Petal Length 5.0 cm: 1.71 cm

