VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum- 590014, Karnataka.

LAB REPORT

on

Machine Learning (23CS6PCMAL)

Submitted by

Gaurav Ramachandra (1BM22CS100)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU - 560019
February 2025 – July 2025

B.M.S. College of Engineering

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Machine Learning (23CS6PCMAL)" carried out by **Gaurav Ramachandra (1BM22CS100),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Laboratory report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Kayarvizhy N
Professor
Department of CSE, BMSCE

Dr. Kavitha Sooda
Professor & HOD
Department of CSE, BMSCE

INDEX

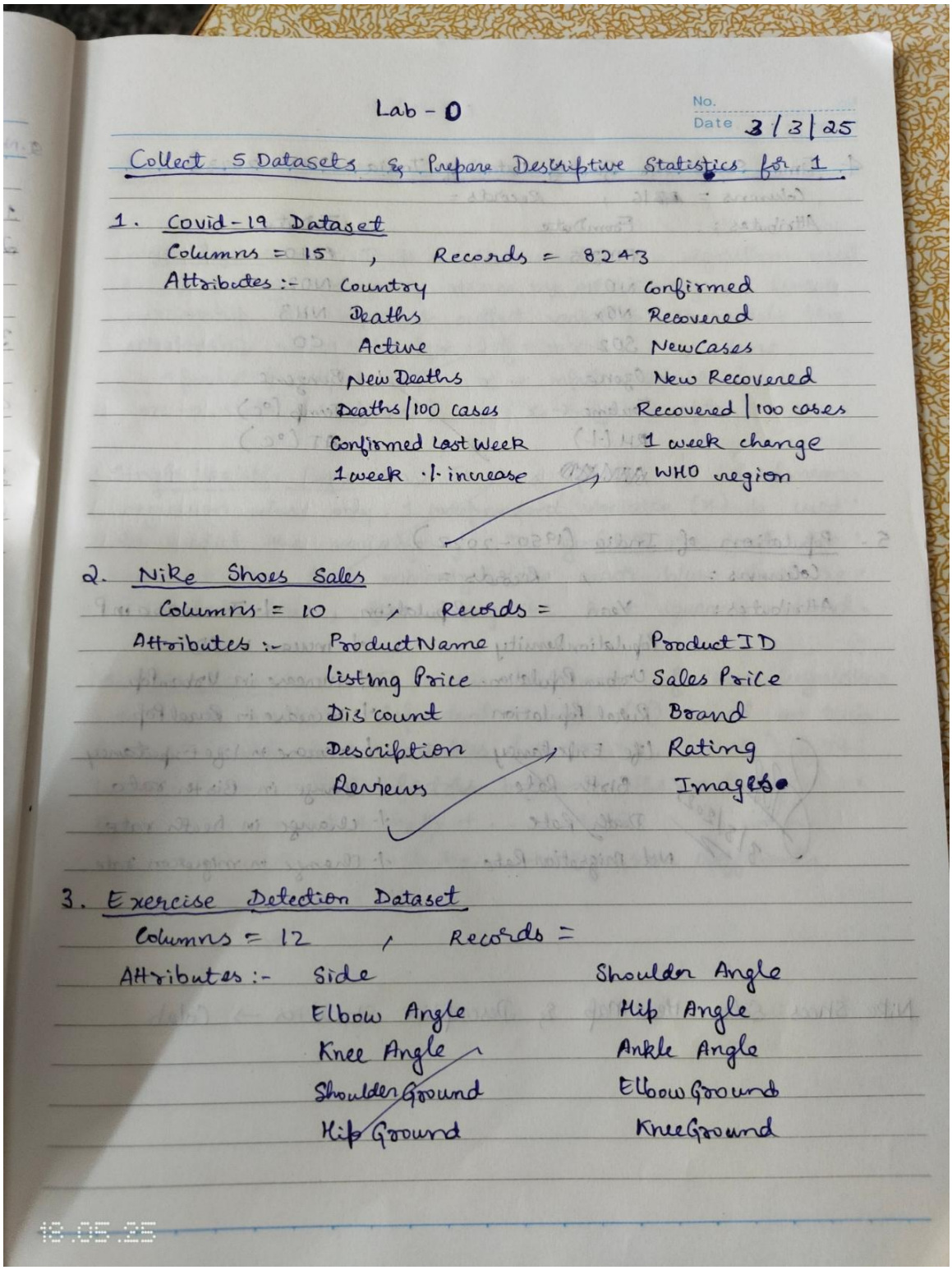
Github Link: https://github.com/Gaurav-Ramachandra/Sem6-ML_Lab

Porgram-1

a. Write a python program to import and export data using pandas library functions.

b. Demonstrate various data pre-processing techniques for a given dataset.

Observation:



Lab - 0

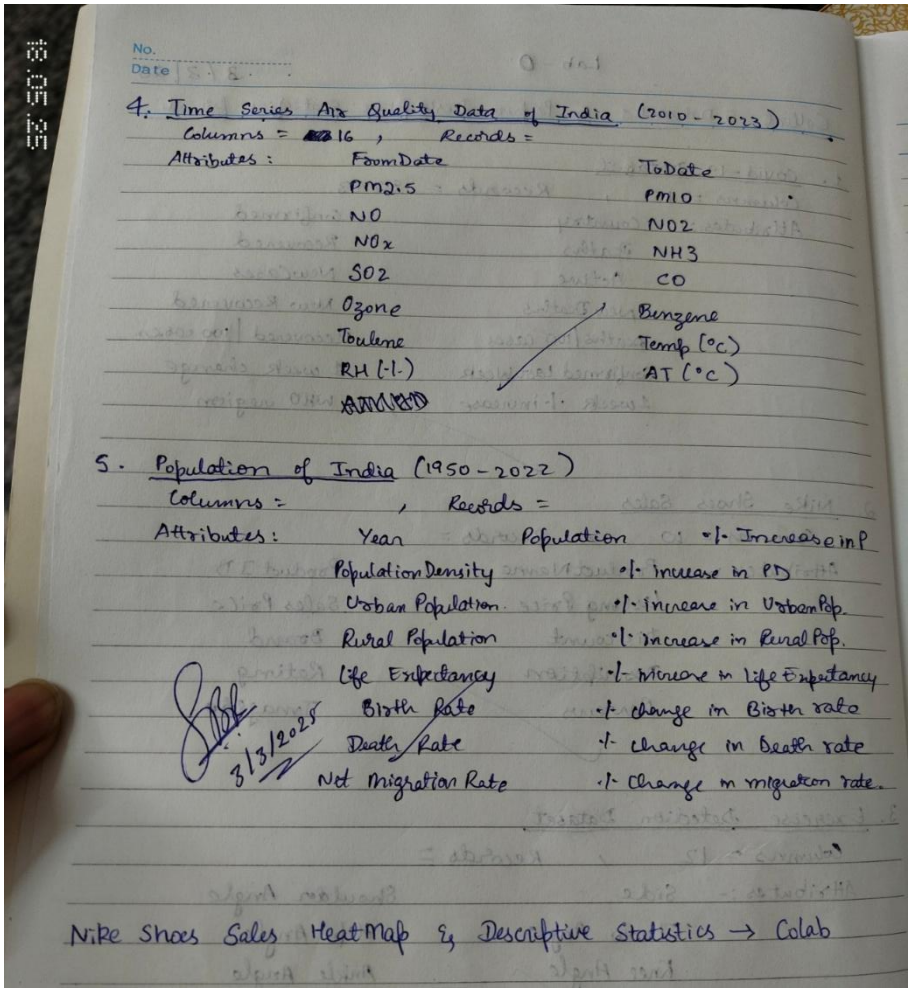
No.
Date 3/3/25

Collect 5 Datasets & Prepare Descriptive Statistics for 1

1. Covid-19 Dataset
   Columns = 15 , Records = 8243
   Attributes :- Country, Confirmed, Deaths, Recovered, Active, New Cases, New Deaths, New Recovered, Deaths/100 cases, Recovered/100 cases, Confirmed last week, 1 week change, 1 week % increase, WHO region

2. Nike Shoes Sales
   Columns = 10 , Records =
   Attributes :- Product Name, Product ID, Listing Price, Sales Price, Discount, Brand, Description, Rating, Reviews, Images

3. Exercise Detection Dataset
   Columns = 12 , Records =
   Attributes :- Side, Shoulder Angle, Elbow Angle, Hip Angle, Knee Angle, Ankle Angle, Shoulder Ground, Elbow Ground, Hip Ground, Knee Ground

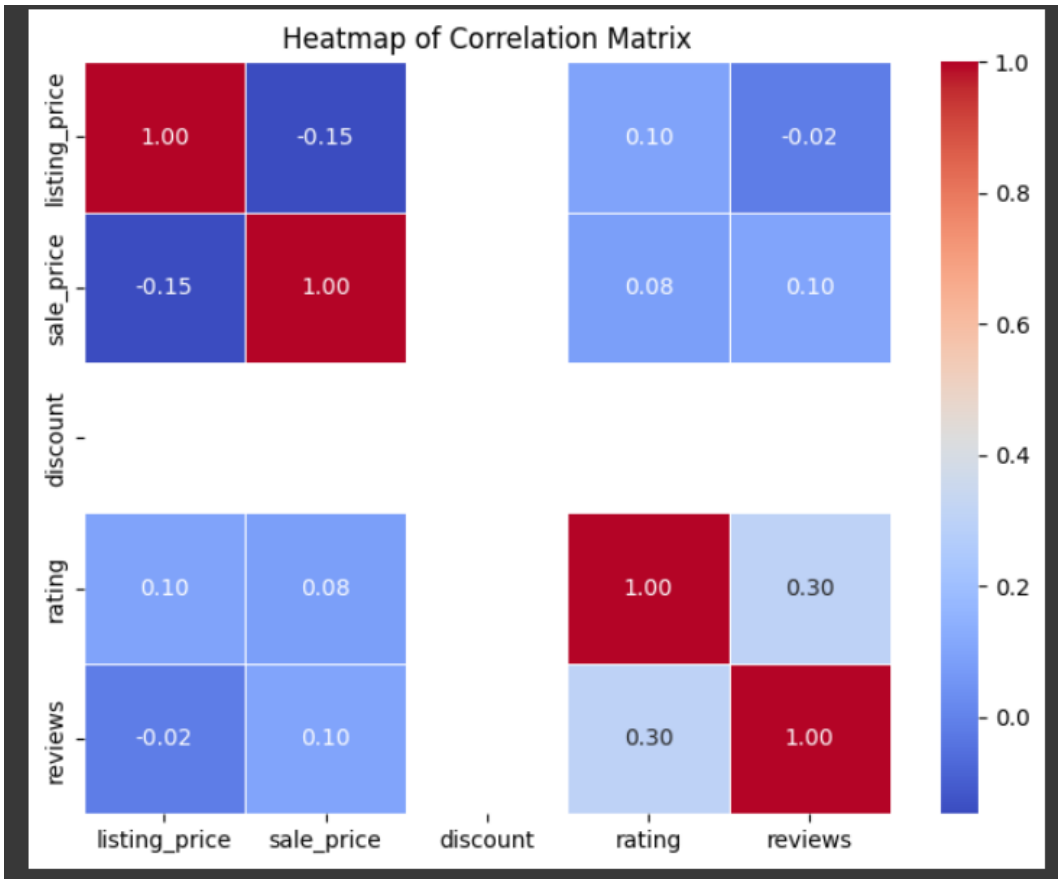


Code:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
file_path = "nike_shoes_sales.csv"  # Update the path if needed
df = pd.read_csv("/content/drive/MyDrive/Sem-6/ML/Lab0/nike_shoes_sales.csv")

# Display basic info
df.info()

# Generate descriptive statistics
desc_stats = df.describe()
print("Descriptive Statistics:\n", desc_stats)

# Plot heatmap of the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Heatmap of Correlation Matrix")
plt.show()
```

Output:

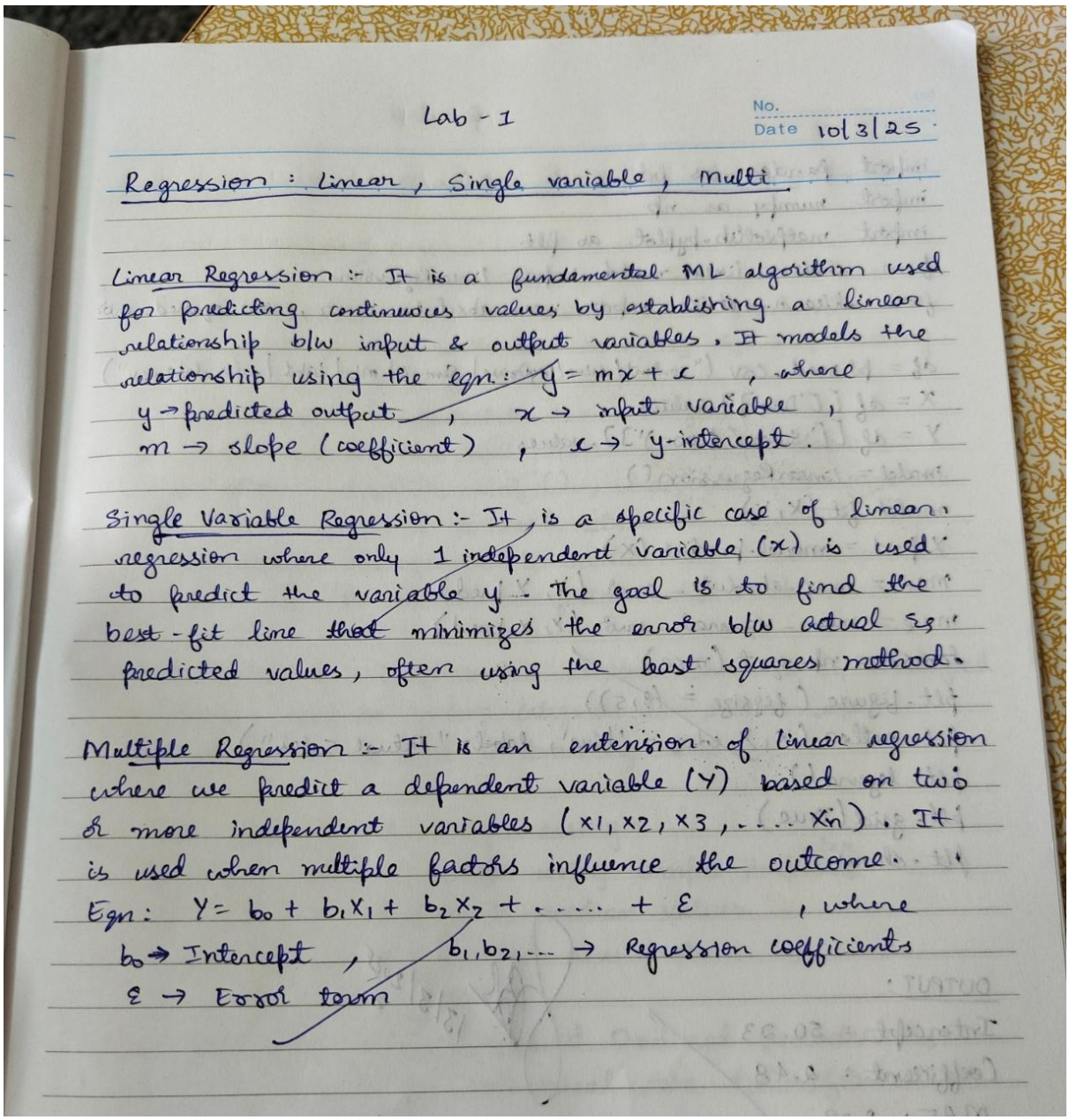
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 643 entries, 0 to 642
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   product_name   643 non-null    object
 1   product_id     643 non-null    object
 2   listing_price  643 non-null    int64
 3   sale_price     643 non-null    int64
 4   discount       643 non-null    int64
 5   brand          643 non-null    object
 6   description    640 non-null    object
 7   rating         643 non-null    float64
 8   reviews        643 non-null    int64
 9   images         572 non-null    object
dtypes: float64(1), int64(4), object(5)
memory usage: 50.4+ KB
Descriptive Statistics:
       listing_price    sale_price  discount      rating     reviews
count     643.000000    643.000000     643.0  643.000000  643.000000
mean     3875.762053  10213.676516       0.0    2.734837    7.181960
std      5889.947172   4513.289512       0.0    2.137756   15.968315
min         0.000000   1595.000000       0.0    0.000000    0.000000
25%         0.000000   6995.000000       0.0    0.000000    0.000000
50%         0.000000   9597.000000       0.0    3.800000    1.000000
75%      8495.000000  12797.000000       0.0    4.600000    6.000000
max     19995.000000  36500.000000       0.0    5.000000  223.000000
```



Program-2:

Implement Linear, Single variable and Multi-Linear Regression algorithm using appropriate dataset

Observation:



Lab - 1

No.
Date 10/3/25

Regression : Linear, Single variable, Multi

Linear Regression :- It is a fundamental ML algorithm used for predicting continuous values by establishing a linear relationship b/w input & output variables. It models the relationship using the eqn: $y = mx + c$, where
$y \rightarrow$ predicted output, $x \rightarrow$ input variable,
$m \rightarrow$ slope (coefficient), $c \rightarrow$ y-intercept.

Single Variable Regression :- It is a specific case of linear regression where only 1 independent variable (x) is used to predict the variable y. The goal is to find the best-fit line that minimizes the error b/w actual & predicted values, often using the least squares method.

Multiple Regression :- It is an extension of linear regression where we predict a dependent variable (Y) based on two or more independent variables ($x_1, x_2, x_3, \ldots x_n$). It is used when multiple factors influence the outcome.
Eqn: $Y = b_0 + b_1x_1 + b_2x_2 + \ldots + \varepsilon$, where
$b_0 \rightarrow$ Intercept, $b_1, b_2, \ldots \rightarrow$ Regression coefficients
$\varepsilon \rightarrow$ Error term

No.
Date.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

df = pd.read_csv("content/drive/MyDrive/Sem-6/ML/Lab1/sales_data.csv")
X = df[['Day']].values
Y = df[['Sales ($1000)']].values
model = LinearRegression()
model.fit(X, Y)
Y_pred = model.predict(X)
mae = mean_absolute_error(Y, Y_pred)
mse = mean_squared_error(Y, Y_pred)
rmse = np.sqrt(mse)
plt.figure(figsize = (8,5))
plt.scatter(X, Y, color = 'blue', label = "Actual Sales")
plt.legend()
plt.grid(True)
plt.show
```

OUTPUT:

Intercept : 50.23
Coefficient : 2.48
MAE : 2.29
MSE : 8.64
RMSE = 2.94

13/3/2025

18.05.25

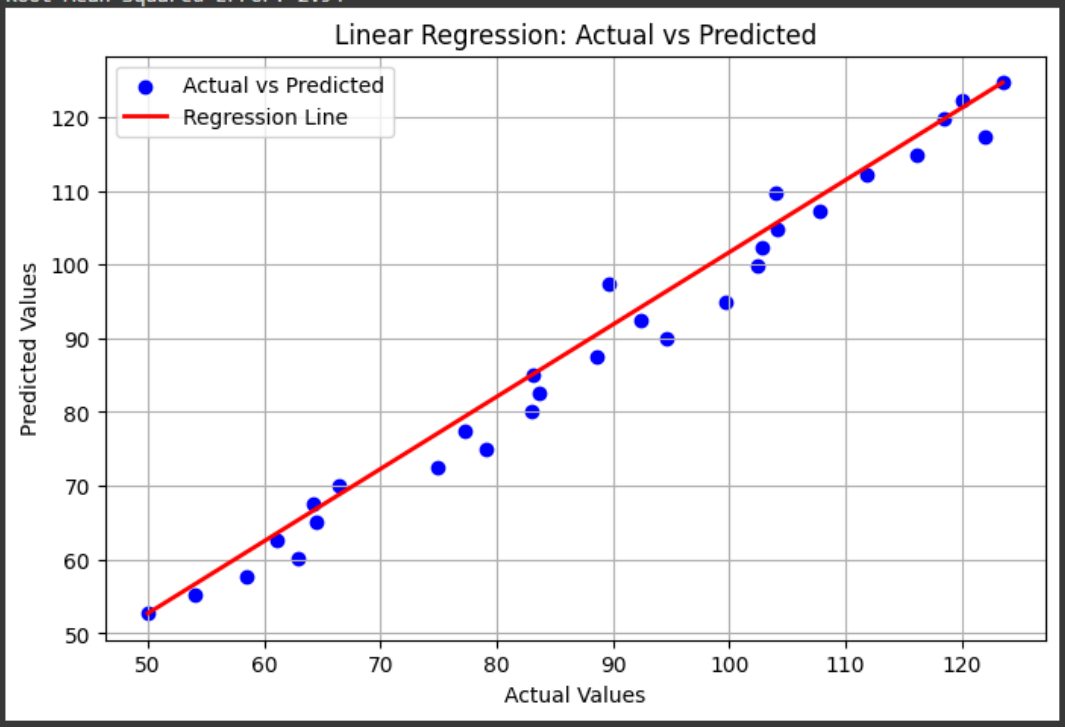
Code:

```
# General Regression- works for both simple and multiple

# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Load the dataset (Modify the path accordingly)
df = pd.read_csv("/content/drive/MyDrive/Sem-6/ML/Lab1/sales_data.csv")

# Display first few rows to check column names
print(df.head())

# Select independent variables (X) and dependent variable (Y)
# Automatically selecting all columns except target column (assumes last column is target)
X = df.iloc[:, :-1].values  # Independent variables (all columns except last)
Y = df.iloc[:, -1].values   # Dependent variable (last column)

# Train the Linear Regression model
model = LinearRegression()
model.fit(X, Y)

# Predict the output
Y_pred = model.predict(X)

# Print Model Parameters
print(f"Intercept: {model.intercept_:.2f}")
```

```
print(f"Coefficients: {model.coef_}")

# Calculate errors
mae = mean_absolute_error(Y, Y_pred)
mse = mean_squared_error(Y, Y_pred)
rmse = np.sqrt(mse)

print(f"Mean Absolute Error: {mae:.2f}")
print(f"Mean Squared Error: {mse:.2f}")
print(f"Root Mean Squared Error: {rmse:.2f}")

# Plot actual vs predicted values (Only applicable for 1D or summary visualization)
plt.figure(figsize=(8, 5))
plt.scatter(Y, Y_pred, color='blue', label="Actual vs Predicted")
plt.plot([min(Y), max(Y)], [min(Y_pred), max(Y_pred)], color='red', linewidth=2,
label="Regression Line")
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Linear Regression: Actual vs Predicted")
plt.legend()
plt.grid(True)
plt.show()
```

Output:

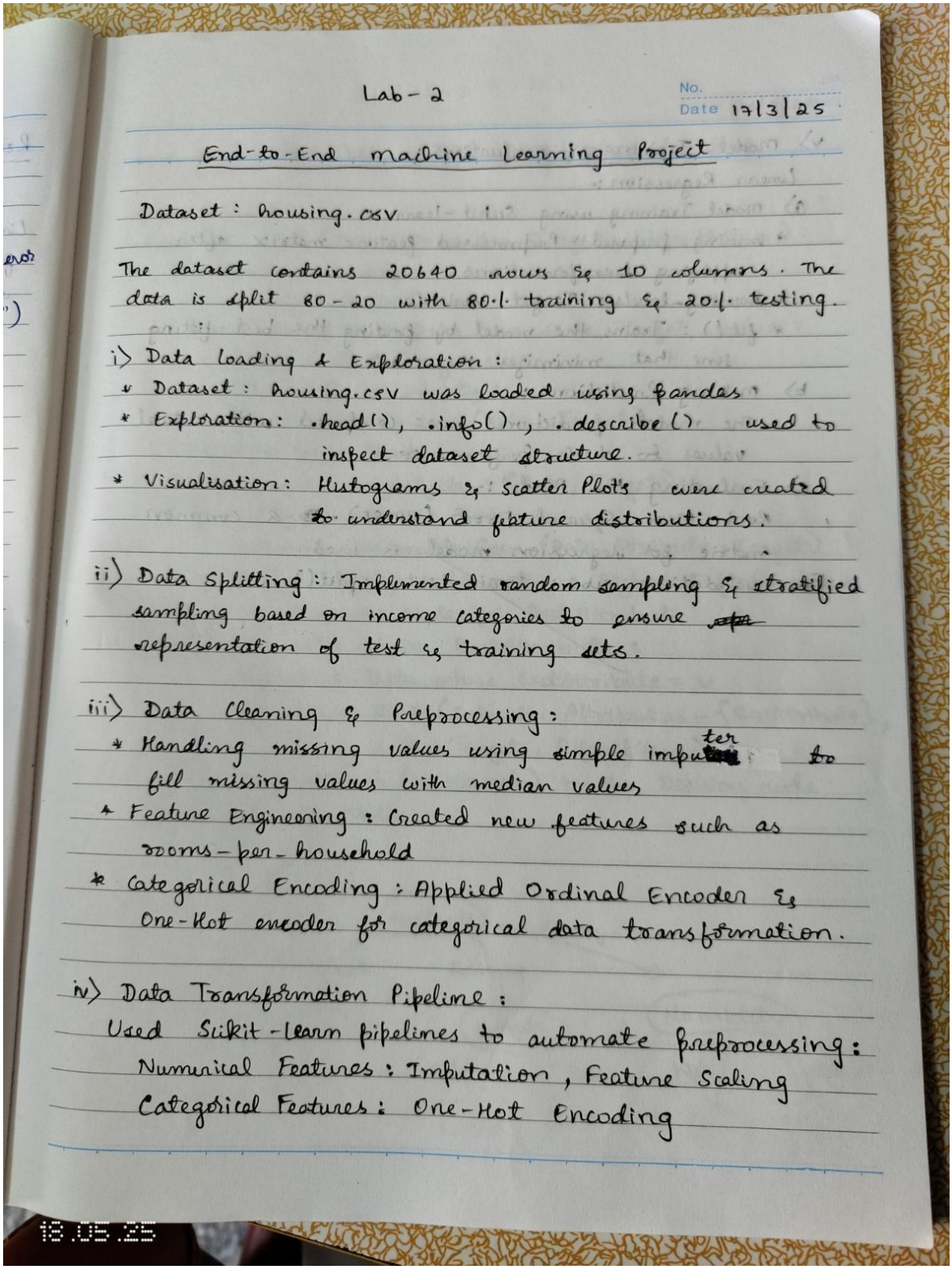
```
   Day  Sales ($1000)
0    1      49.982347
1    2      54.072363
2    3      58.493790
3    4      62.926635
4    5      61.062477
Intercept: 50.23
Coefficients: [2.48075631]
Mean Absolute Error: 2.29
Mean Squared Error: 8.64
Root Mean Squared Error: 2.94
```



Program-3:

End-to-end Machine Learning Project

Observation:



Lab - 2

No.
Date 17/3/25

End-to-End machine Learning Project

Dataset : housing.csv

The dataset contains 20640 rows & 10 columns. The data is split 80-20 with 80% training & 20% testing.

i) Data loading & Exploration :
* Dataset : housing.csv was loaded using pandas
* Exploration : .head(), .info(), .describe() used to inspect dataset structure.
* Visualisation : Histograms & scatter Plot's were created to understand feature distributions.

ii) Data Splitting : Implemented random sampling & stratified sampling based on income categories to ensure representation of test & training sets.

iii) Data Cleaning & Preprocessing :
* Handling missing values using simple imputer to fill missing values with median values
* Feature Engineering : Created new features such as rooms-per-household
* Categorical Encoding : Applied Ordinal Encoder & One-Hot encoder for categorical data transformation.

iv) Data Transformation Pipeline :
Used Scikit-learn pipelines to automate preprocessing:
Numerical Features : Imputation, Feature Scaling
Categorical Features : One-Hot Encoding

No.
Date

v) Model Training & Evaluation :

Linear Regression :-

a) Model Training using Scikit-learn :-

* housing_prepared : Preprocessed feature matrix after applying transformations
* housing_labels : Target variable
* fit() : Trains the model by finding the best-fitting line that minimizes the error

b) Making Predictions :-

The model's predictions were compared with actual values to check performance

c) Evaluating the Model :-

Root Mean Squared Error (RMSE) - a common metric for regression model is used.

For validation use : train_test_split()

18.05.25

Code:

```
import os
import tarfile
import urllib
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/ageron/handson-ml2/master/"
HOUSING_PATH = os.path.join("datasets", "housing")
HOUSING_URL = DOWNLOAD_ROOT + "datasets/housing/housing.tgz"
def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
  os.makedirs(housing_path, exist_ok=True)
  tgz_path = os.path.join(housing_path, "housing.tgz")
  urllib.request.urlretrieve(housing_url, tgz_path)
  housing_tgz = tarfile.open(tgz_path)
  housing_tgz.extractall(path=housing_path)
  housing_tgz.close()

import pandas as pd
def load_housing_data(housing_path=HOUSING_PATH):
  csv_path = os.path.join(housing_path, "housing.csv")
  return pd.read_csv(csv_path)

housing=pd.read_csv("/content/housing.csv")
housing.head()

housing.info()

housing["ocean_proximity"].value_counts()

housing.describe()
```

```
import numpy as np
def split_train_test(data, test_ratio):
  shuffled_indices = np.random.permutation(len(data))
  test_set_size = int(len(data) * test_ratio)
  test_indices = shuffled_indices[:test_set_size]
  train_indices = shuffled_indices[test_set_size:]
  return data.iloc[train_indices], data.iloc[test_indices]
```

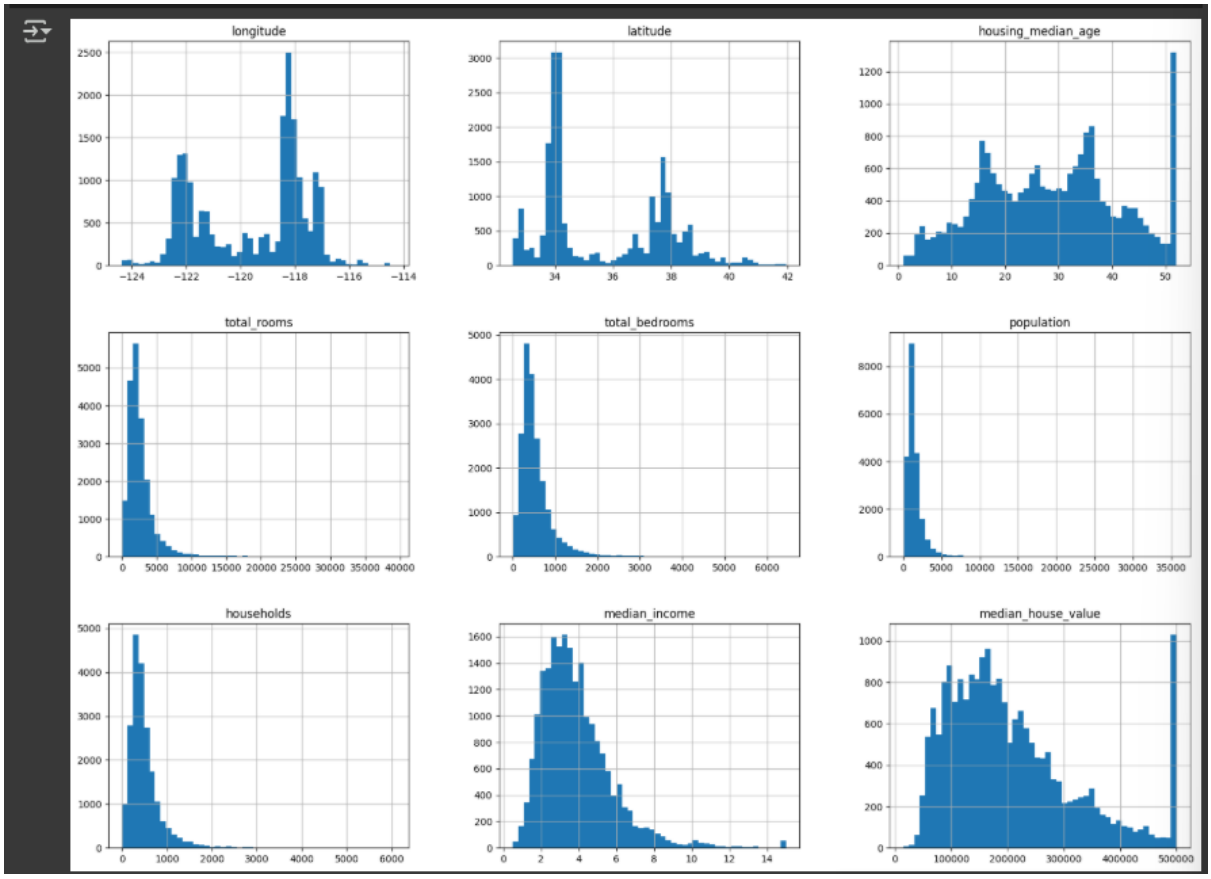
Output:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

ocean_proximity	count
<1H OCEAN	9136
INLAND	6551
NEAR OCEAN	2658
NEAR BAY	2290
ISLAND	5

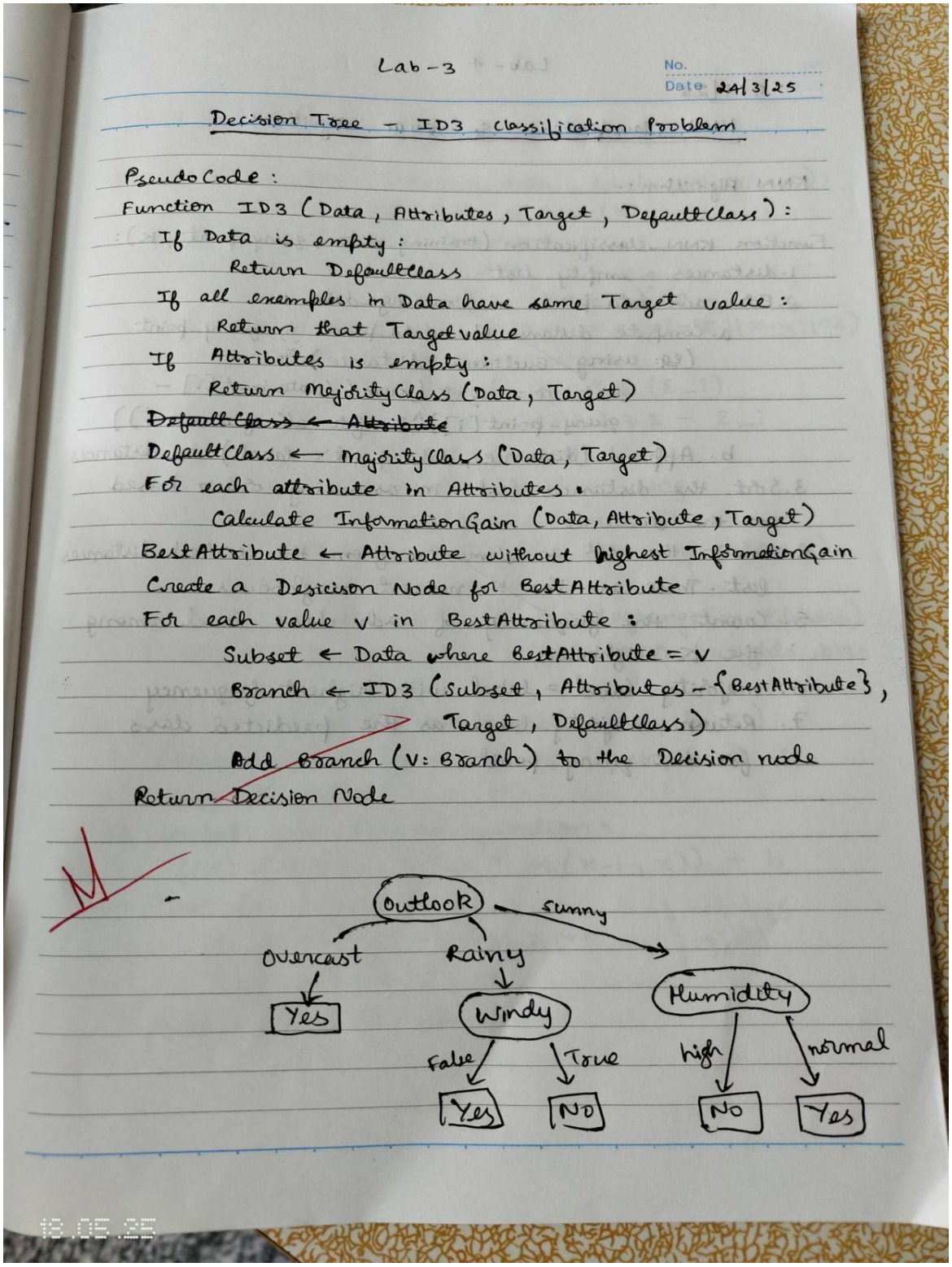
dtype: int64



Program-4:

Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.

Observation:



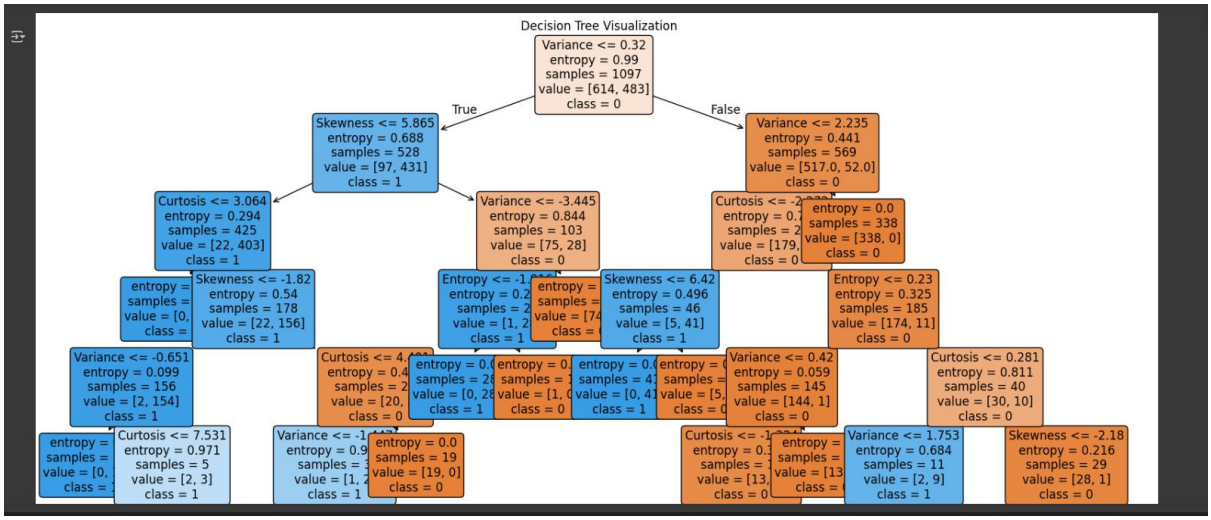
Code:

```
# 2nd dataset - data_authentication code with decision tree

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree, export_text

# Mount Google Drive if needed (for Colab)
# from google.colab import drive
# drive.mount('/content/drive')

# Load the dataset (adjust the file path as necessary)
file_path = "/content/drive/MyDrive/Sem-6/ML/Lab3/bill_authentication.csv"
df = pd.read_csv(file_path)

# Display the dataset's first few rows
print("Dataset Preview:")
print(df.head())

# OPTIONAL: Drop any unnecessary columns (e.g., index column)
df = df.drop(columns=["Unnamed: 0"], errors='ignore')

# Assume the target column is named "class" (adjust if necessary)
target_column = "class" if "class" in df.columns else df.columns[-1]

# Define features (X) and target (y)
X = df.drop(columns=[target_column])
y = df[target_column]
```

```
# 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)

# Build the Decision Tree Classifier using the 'entropy' criterion (ID3)
clf = DecisionTreeClassifier(criterion="entropy", random_state=42)
clf.fit(X_train, y_train)

# Print the decision tree rules in text format
tree_rules = export_text(clf, feature_names=list(X.columns))
print("\nDecision Tree Rules:\n")
print(tree_rules)

# Plot the decision tree pictorially
plt.figure(figsize=(20, 10))
plot_tree(clf,
          feature_names=X.columns,
          class_names=[str(c) for c in np.unique(y)],
          filled=True,
          rounded=True,
          fontsize=12)
plt.title("Decision Tree Visualization")
plt.show()
```

Output:

```
Dataset Preview:
   Variance  Skewness  Curtosis  Entropy  Class
0   3.62160    8.6661   -2.8073 -0.44699      0
1   4.54590    8.1674   -2.4586 -1.46210      0
2   3.86600   -2.6383    1.9242  0.10645      0
3   3.45660    9.5228   -4.0112 -3.59440      0
4   0.32924   -4.4552    4.5718 -0.98880      0
```

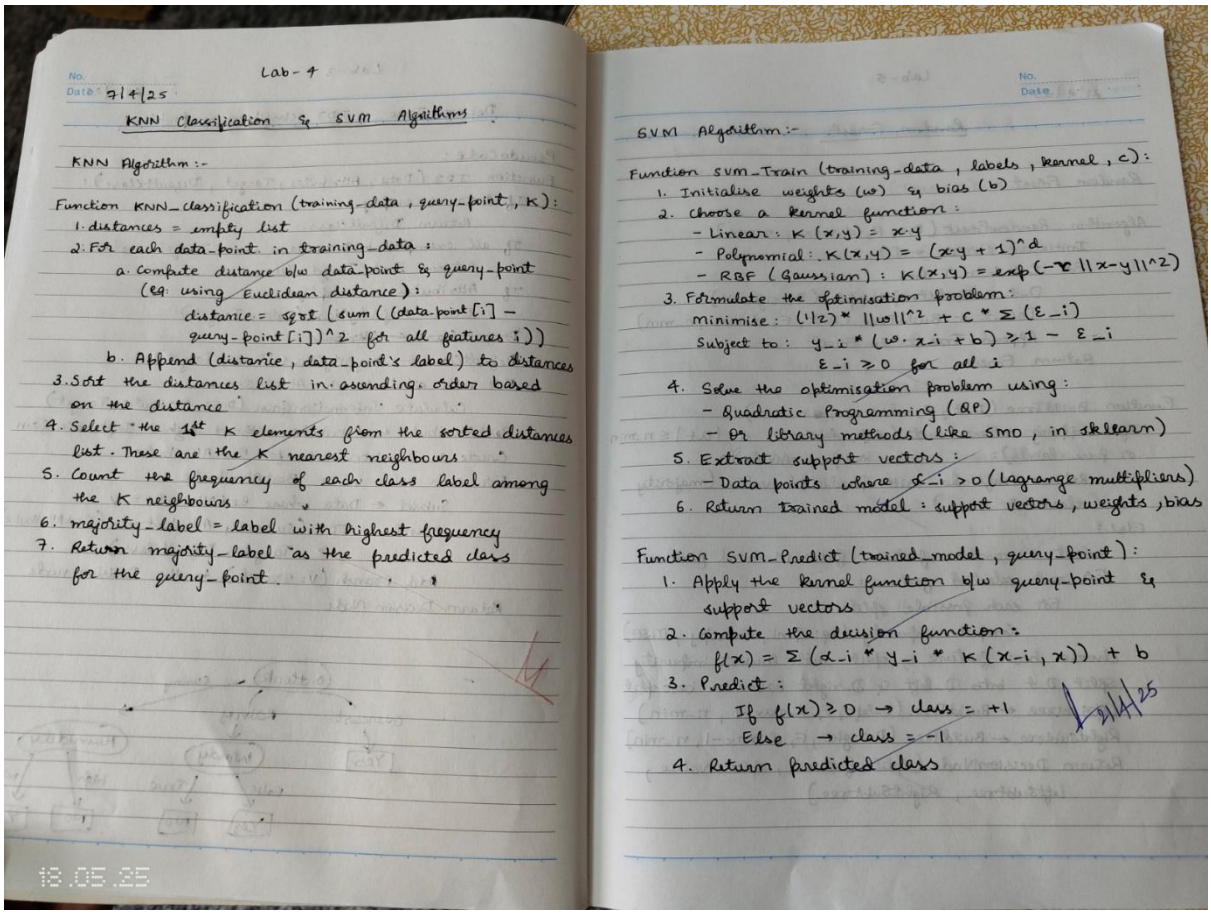


Program-5:

a. Build KNN Classification model for a given dataset.

b. Build Support vector machine model for a given dataset

Observation:



Lab - 4

7/4/25

KNN Classification & SVM Algorithms

KNN Algorithm :-

Function KNN_classification (training_data, query_point, K):
1. distances = empty list
2. For each data_point in training_data :
   a. Compute distance b/w data_point & query_point (eg: using Euclidean distance):
      distance = sqrt (sum ((data_point[i] − query_point[i])^2 for all features i))
   b. Append (distance, data_point's label) to distances
3. Sort the distances list in ascending order based on the distance
4. Select the 1st K elements from the sorted distances list. These are the K nearest neighbours
5. Count the frequency of each class label among the K neighbours
6. majority_label = label with highest frequency
7. Return majority_label as the predicted class for the query_point.

SVM Algorithm :-

Function SVM_Train (training_data, labels, kernel, C):
1. Initialise weights (w) & bias (b)
2. Choose a kernel function :
   - Linear : K (x,y) = x.y
   - Polynomial : K(x,y) = (x.y + 1)^d
   - RBF (Gaussian) : K(x,y) = exp(−γ ||x−y||^2)
3. Formulate the optimisation problem:
   Minimise : (1/2) * ||w||^2 + C * Σ (ε_i)
   Subject to : y_i * (w · x_i + b) ≥ 1 − ε_i
   ε_i ≥ 0 for all i
4. Solve the optimisation problem using :
   - Quadratic Programming (QP)
   - Or library methods (like SMO, in sklearn)
5. Extract support vectors :
   - Data points where α_i > 0 (Lagrange multipliers)
6. Return trained model : support vectors, weights, bias

Function SVM_Predict (trained_model, query_point):
1. Apply the kernel function b/w query_point & support vectors
2. Compute the decision function :
   f(x) = Σ (α_i * y_i * K (x_i, x)) + b
3. Predict :
   If f(x) ≥ 0 → class = +1
   Else → class = −1
4. Return predicted class

7/4/25

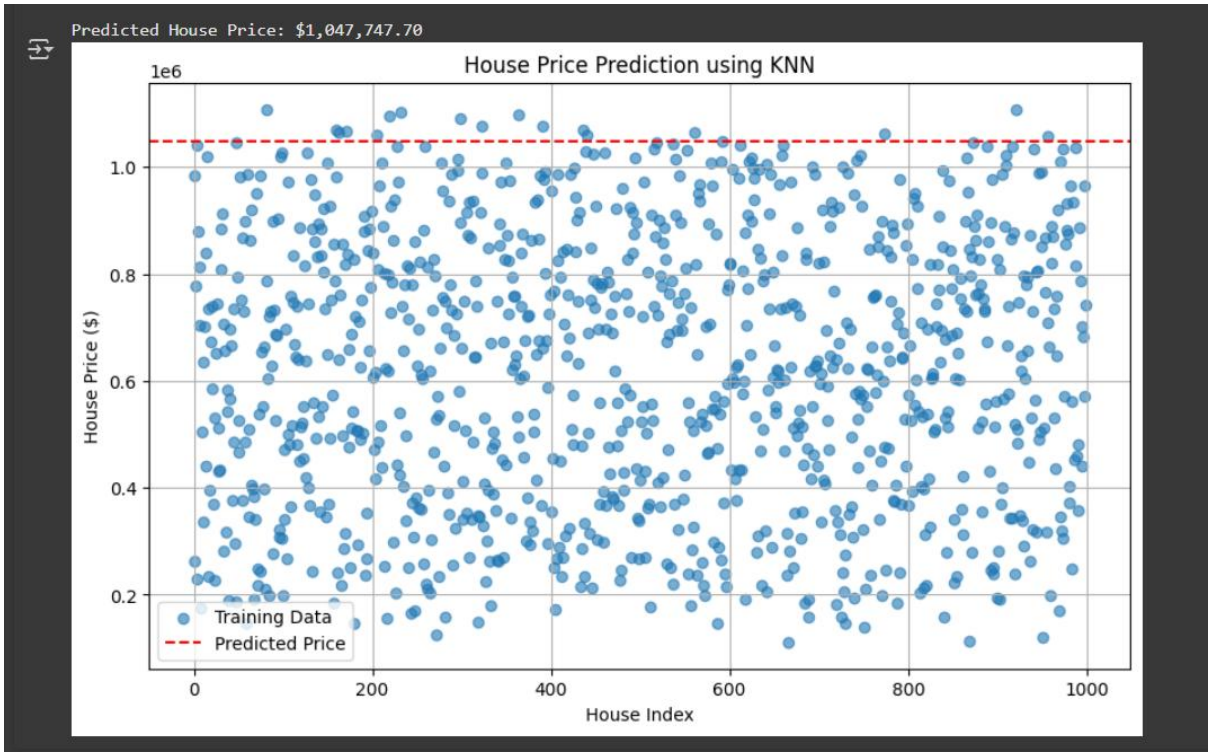
18.05.25

Code:

```
# knn - House Price dataset

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv('/content/drive/MyDrive/Sem-6/ML/Lab4/house_price_regression_dataset.csv')

# Normalize features
def normalize(df):
    return (df - df.min()) / (df.max() - df.min())

features = ['Square_Footage', 'Num_Bedrooms', 'Num_Bathrooms', 'Year_Built',
            'Lot_Size', 'Garage_Size', 'Neighborhood_Quality']
target = 'House_Price'

df_normalized = normalize(df[features])
df_normalized[target] = df[target]  # append target for reference

# KNN Regression with explicit Euclidean distance formula
def euclidean_distance(point1, point2):
    dist = 0
    for i in range(len(point1)):
        dist += (point1[i] - point2[i]) ** 2
    return np.sqrt(dist)

def knn_predict(X_train, y_train, x_input, k=3):
    distances = []
```

```
    for i in range(len(X_train)):
        # Explicit Euclidean distance
        dist = euclidean_distance(list(X_train.iloc[i]), x_input)
        distances.append((dist, y_train.iloc[i]))

    distances.sort(key=lambda x: x[0])
    nearest_neighbors = distances[:k]

    # Mean of K nearest neighbors
    predicted_value = np.mean([neighbor[1] for neighbor in nearest_neighbors])
    return predicted_value

new_input = {
    'Square_Footage': 25000,
    'Num_Bedrooms': 4,
    'Num_Bathrooms': 3,
    'Year_Built': 2015,
    'Lot_Size': 0.5,
    'Garage_Size': 2,
    'Neighborhood_Quality': 8
}

# Normalize new input
new_input_normalized = []
for feature in features:
    norm_value = (new_input[feature] - df[feature].min()) / (df[feature].max() -
df[feature].min())
    new_input_normalized.append(norm_value)

# Predict using KNN
```

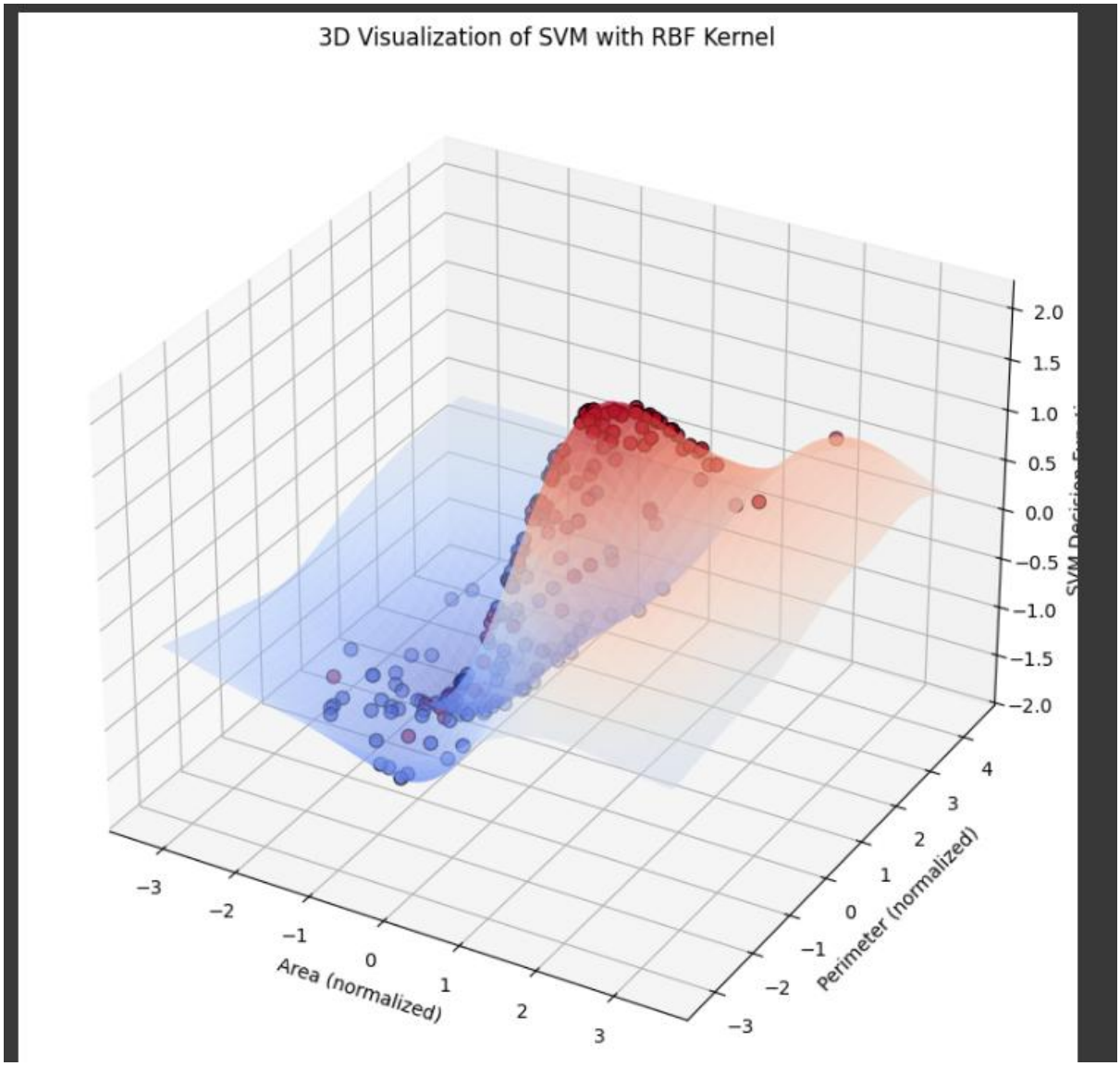
```
predicted_price = knn_predict(df_normalized[features], df[target], new_input_normalized, k=3)

print(f"\nPredicted House Price: ${predicted_price:,.2f}")


# ---------- Visualization ----------

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

plt.scatter(range(len(df)), df['House_Price'], label="Training Data", alpha=0.6)

plt.axhline(y=predicted_price, color='red', linestyle='--', label='Predicted Price')

plt.title('House Price Prediction using KNN')

plt.xlabel('House Index')

plt.ylabel('House Price ($)')

plt.legend()

plt.grid(True)

plt.show()
```



SVM - 1

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler

# Assume X = [[Area, Perimeter], ...] and y = [class labels]
# Load your real data here
# Example data:
from sklearn.datasets import make_classification
X, y = make_classification(n_samples=200, n_features=2, n_redundant=0,
n_clusters_per_class=1, random_state=42)
X = StandardScaler().fit_transform(X)

# Train SVM with RBF kernel
clf = svm.SVC(kernel='rbf', C=1, gamma=0.5)
clf.fit(X, y)
# Create mesh grid
xx, yy = np.meshgrid(np.linspace(X[:,0].min()-1, X[:,0].max()+1, 100),
                     np.linspace(X[:,1].min()-1, X[:,1].max()+1, 100))
# Compute the decision function value for each grid point
zz = np.array([clf.decision_function([[x, y]])[0] for x, y in zip(xx.ravel(), yy.ravel())])
zz = zz.reshape(xx.shape)
# Plot 3D
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Surface
ax.plot_surface(xx, yy, zz, cmap='coolwarm', alpha=0.5, linewidth=0)
```

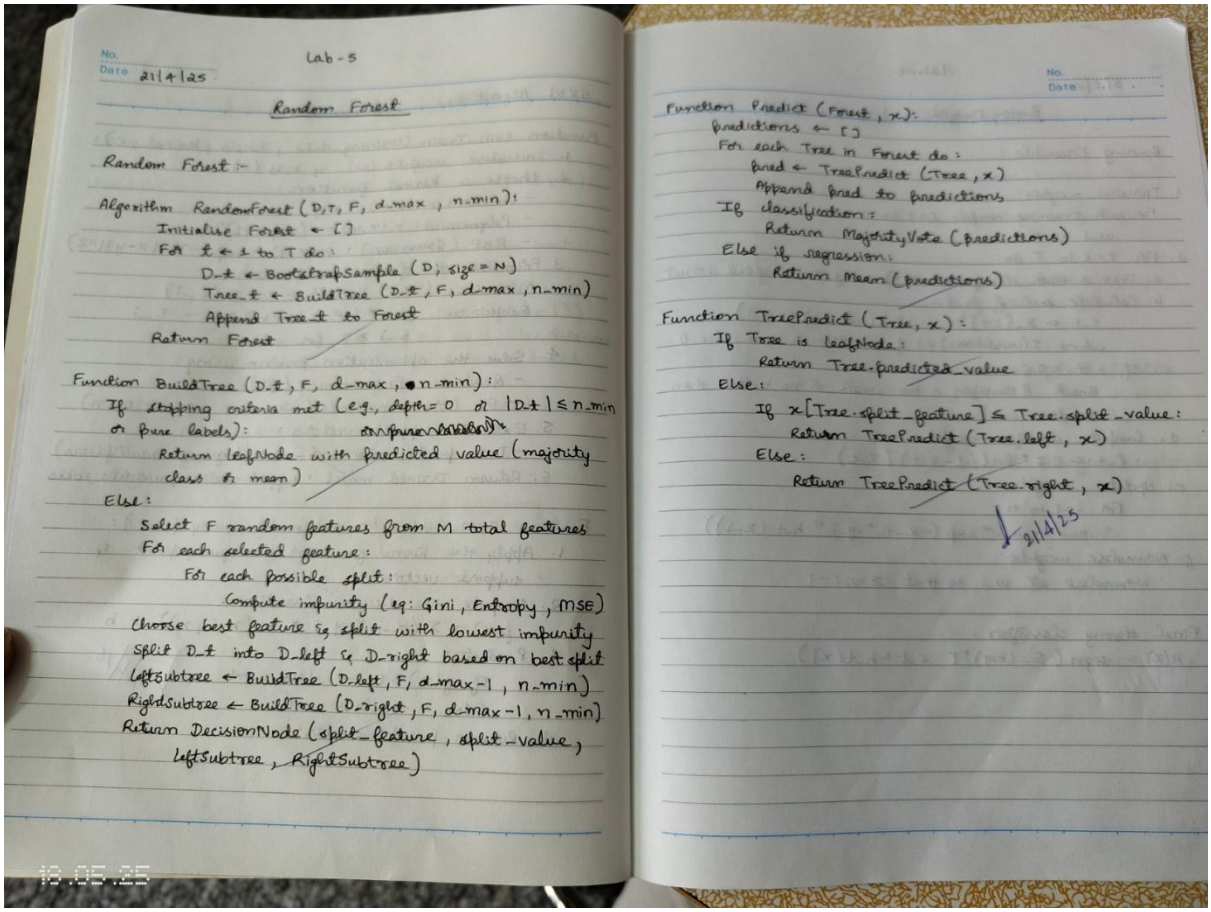
```
# Scatter the actual data points
z_vals = clf.decision_function(X)
ax.scatter(X[:, 0], X[:, 1], z_vals, c=y, cmap='coolwarm', edgecolors='k', s=50)
# Labels
ax.set_xlabel('Area (normalized)')
ax.set_ylabel('Perimeter (normalized)')
ax.set_zlabel('SVM Decision Function')
ax.set_title('3D Visualization of SVM with RBF Kernel')
plt.tight_layout()
plt.show()
```



Program-6:

Implement Random Forest ensemble method on a given dataset

Observation:



Code:

Classification

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Load the dataset
file_path = "/content/drive/MyDrive/Sem-6/ML/Lab5/Pumpkin_Seeds_Dataset.xlsx"
df = pd.read_excel(file_path, sheet_name='Pumpkin_Seeds_Dataset')

# Separate features and target
X = df.drop(columns=['Class'])
y = df['Class']

# Encode the target labels
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
)

# Train the Random Forest classifier
rf_model = RandomForestClassifier(n_estimators=100, max_depth=None, random_state=42)
rf_model.fit(X_train, y_train)

# Predict on test data
y_pred = rf_model.predict(X_test)

# Evaluate the model
print("✅ Accuracy:", accuracy_score(y_test, y_pred))
print("\n📊 Classification Report:\n")
print(classification_report(y_test, y_pred, target_names=le.classes_))
```

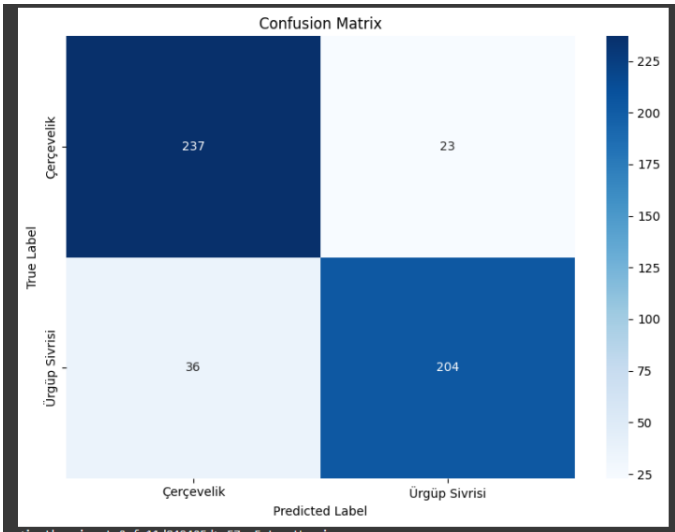
```
# Plot the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.tight_layout()
plt.show()

# Plot top 10 feature importances
importances = rf_model.feature_importances_
features = X.columns
indices = importances.argsort()[::-1][:10]  # Top 10

plt.figure(figsize=(10, 6))
sns.barplot(x=importances[indices], y=features[indices], palette="viridis")
plt.title("Top 10 Feature Importances")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```

Output:

Accuracy: 0.882

Classification Report:

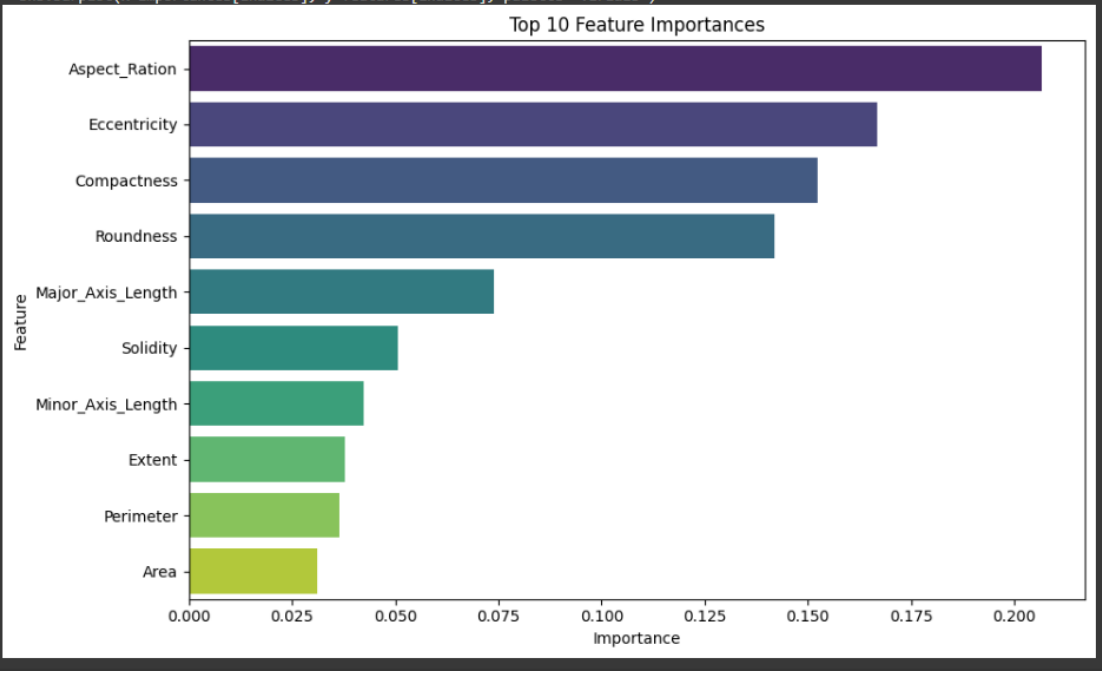
	precision	recall	f1-score	support
Çerçevelik	0.87	0.91	0.89	260
Ürgüp Sivrisi	0.90	0.85	0.87	240
accuracy			0.88	500
macro avg	0.88	0.88	0.88	500
weighted avg	0.88	0.88	0.88	500



<ipython-input-2-fc11d849425d>:57: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

```
sns.barplot(x=importances[indices], y=features[indices], palette="viridis")
```

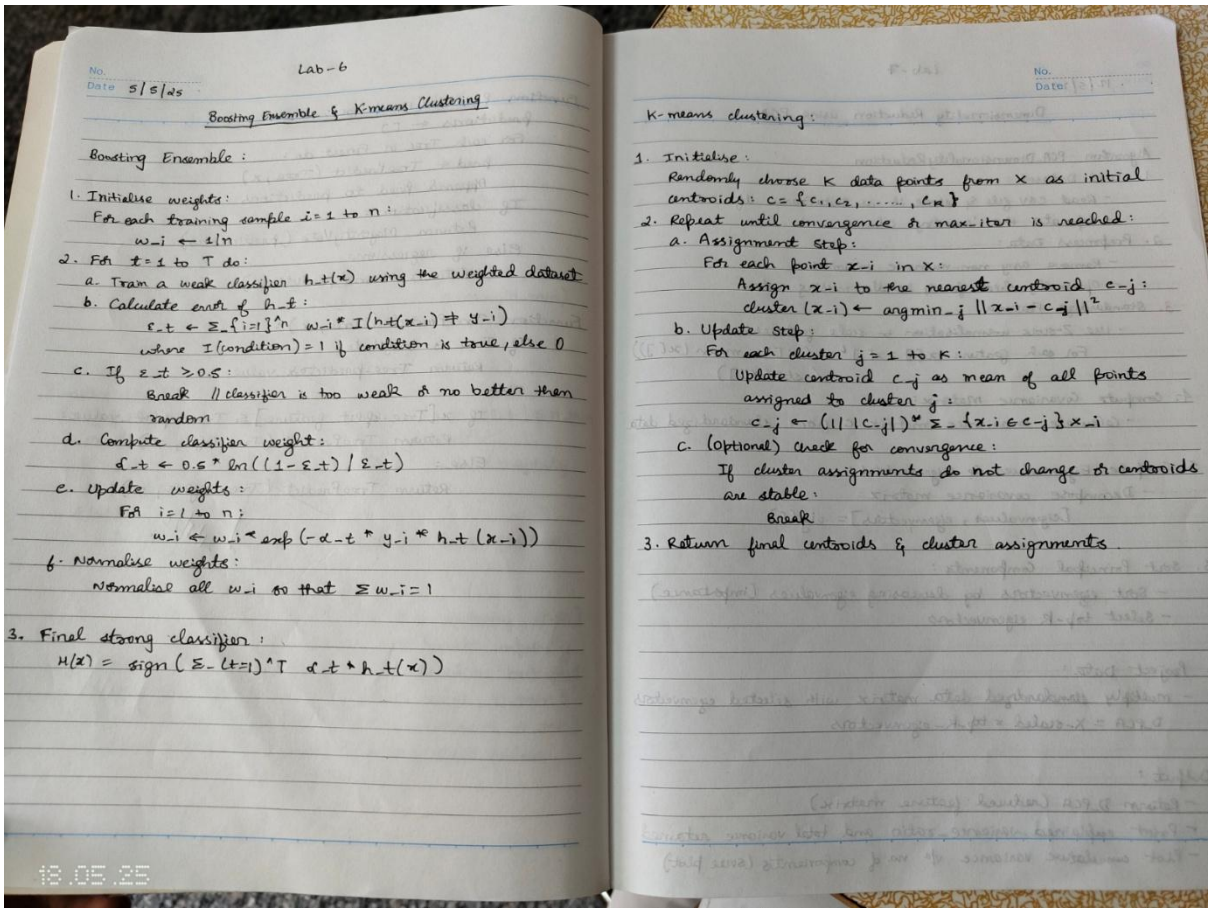


Program-7:

a. Implement Boosting ensemble method on a given dataset.

b. Build k-Means algorithm to cluster a set of data stored in a .CSV file.

Observation:



Code:

ADAboost boosting ensemble

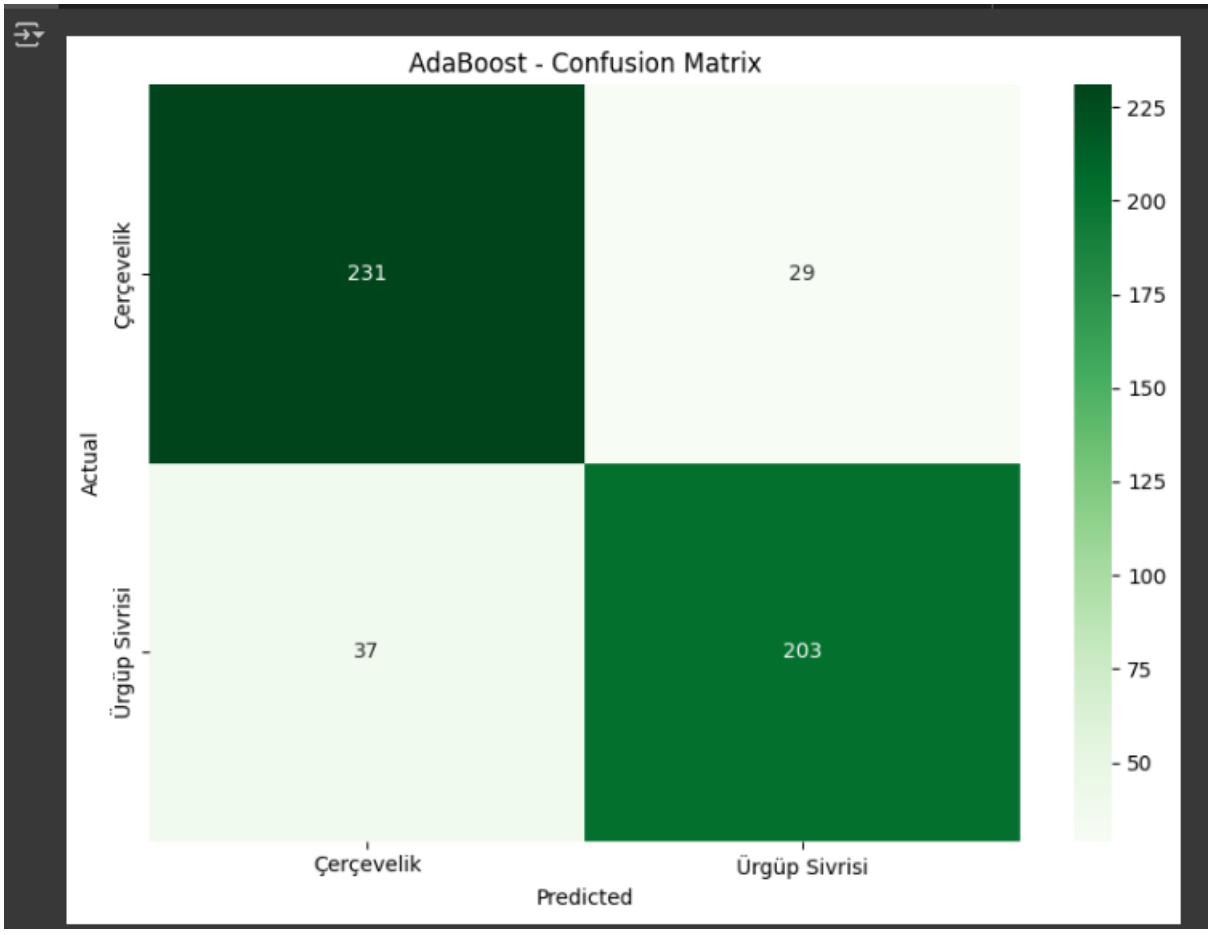
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset
file_path = "/content/drive/MyDrive/Sem-6/ML/Lab6/Pumpkin_Seeds_Dataset.xlsx"
df = pd.read_excel(file_path, sheet_name='Pumpkin_Seeds_Dataset')

# Features and target
X = df.drop(columns=['Class'])
y = df['Class']

# Encode target labels
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded, test_size=0.2, stratify=y_encoded, random_state=42
)

# Initialize AdaBoost with DecisionTree as base estimator
base_tree = DecisionTreeClassifier(max_depth=1)
adaboost_model = AdaBoostClassifier(
    estimator=base_tree,
    n_estimators=100,
    learning_rate=1.0,
    random_state=42
)

# Train the model
```

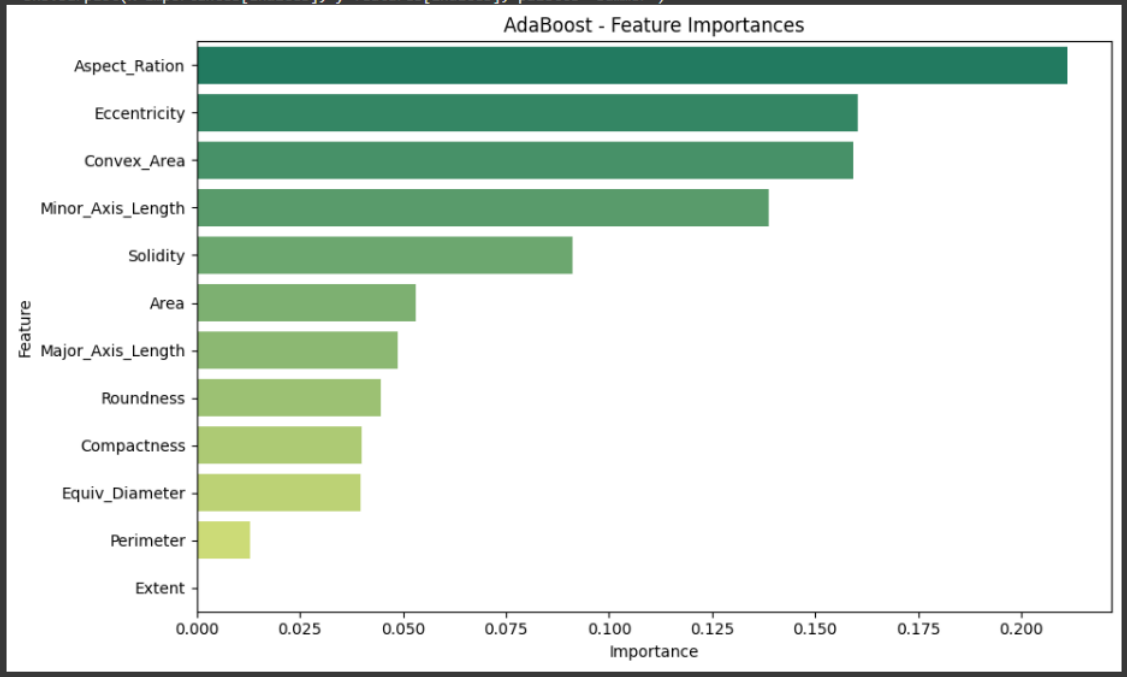
```
adaboost_model.fit(X_train, y_train)

# Predict on test data
y_pred = adaboost_model.predict(X_test)

# Evaluation
print(" ✅ Accuracy:", accuracy_score(y_test, y_pred))
print("\n 📊 Classification Report:\n")
print(classification_report(y_test, y_pred, target_names=le.classes_))
# Confusion Matrix Plot
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Greens',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title("AdaBoost - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
# Feature Importance Plot
importances = adaboost_model.feature_importances_
features = X.columns
indices = importances.argsort()[::-1]
plt.figure(figsize=(10, 6))
sns.barplot(x=importances[indices], y=features[indices], palette="summer")
plt.title("AdaBoost - Feature Importances")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```

Output:

```
Accuracy: 0.868

Classification Report:

              precision    recall  f1-score   support

   Çerçevelik       0.86      0.89      0.88       260
Ürgüp Sivrisi       0.88      0.85      0.86       240

    accuracy                           0.87       500
   macro avg       0.87      0.87      0.87       500
weighted avg       0.87      0.87      0.87       500
```



```
sns.barplot(x=importances[indices], y=features[indices], palette="summer")
```

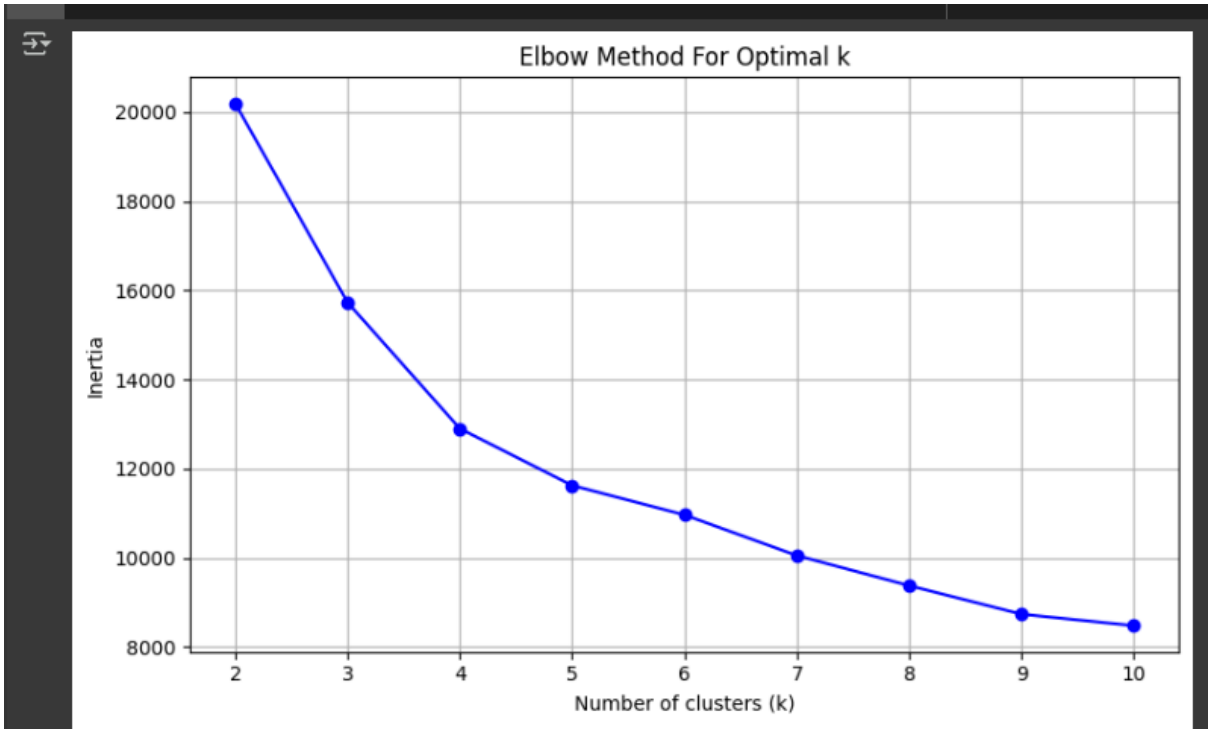


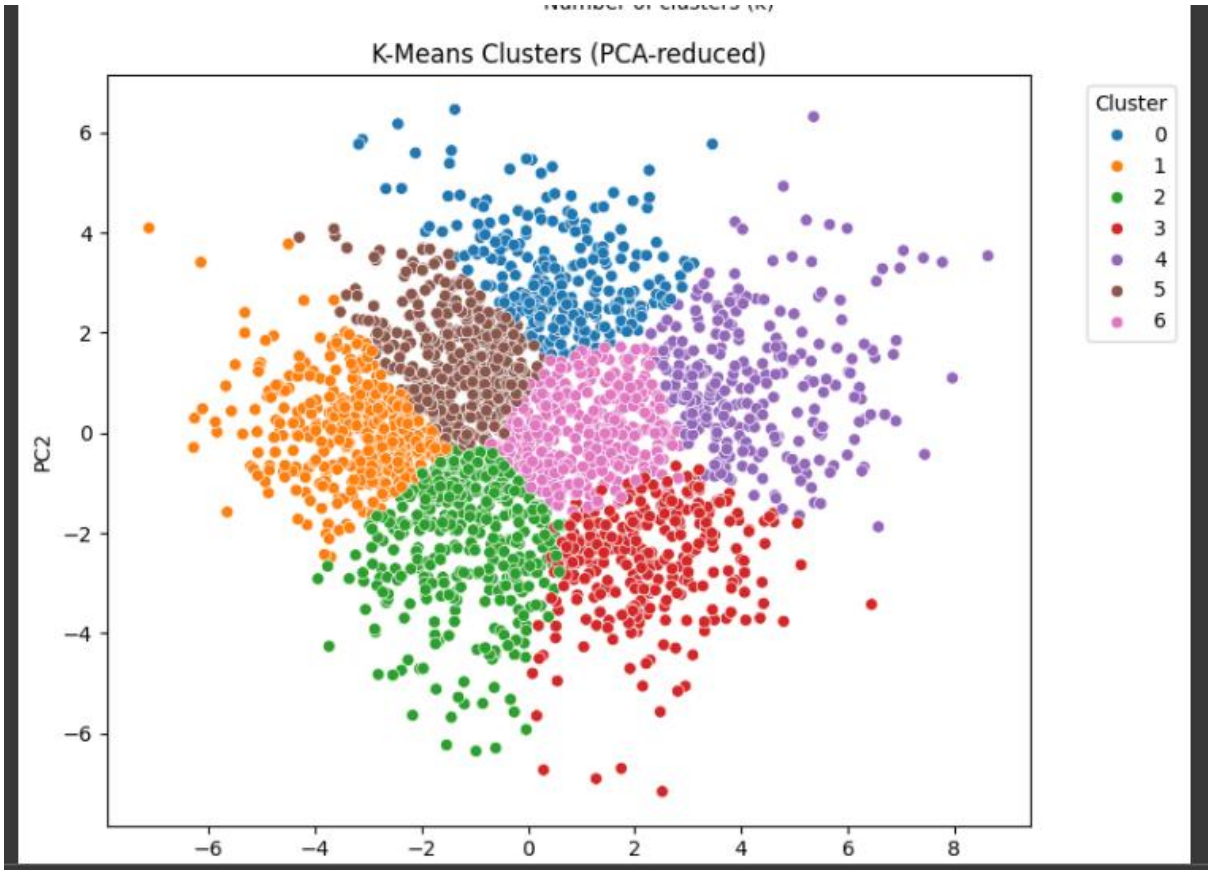
```
# K-means Clustering

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score

# Load dataset
file_path = "/content/drive/MyDrive/Sem-6/ML/Lab6/Pumpkin_Seeds_Dataset.xlsx"
df = pd.read_excel(file_path, sheet_name='Pumpkin_Seeds_Dataset')

# Drop label column for unsupervised clustering
X = df.drop(columns=['Class'])

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Optional: Reduce dimensions for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Determine optimal number of clusters (Elbow Method)
inertia = []
K = range(2, 11)
for k in K:
    km = KMeans(n_clusters=k, random_state=42, n_init='auto')
```

```
    km.fit(X_scaled)
    inertia.append(km.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(K, inertia, 'bo-')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method For Optimal k')
plt.grid(True)
plt.tight_layout()
plt.show()

# Fit final K-Means model (e.g., k=7 for 7 seed types)
k = 7
kmeans = KMeans(n_clusters=k, random_state=42, n_init='auto')
clusters = kmeans.fit_predict(X_scaled)

# Add cluster labels to DataFrame
df['Cluster'] = clusters

# Plot PCA visualization
plt.figure(figsize=(8, 6))
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=clusters, palette='tab10')
plt.title("K-Means Clusters (PCA-reduced)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend(title="Cluster", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

```
# Optional: Compare clusters with actual classes
df['True_Class'] = df['Class']
ct = pd.crosstab(df['Cluster'], df['True_Class'])
print("\n📊 Cluster vs Actual Class Comparison:\n")
print(ct)
# Silhouette Score
score = silhouette_score(X_scaled, clusters)
print(f"\n✅ Silhouette Score: {score:.3f}")
```

Output:

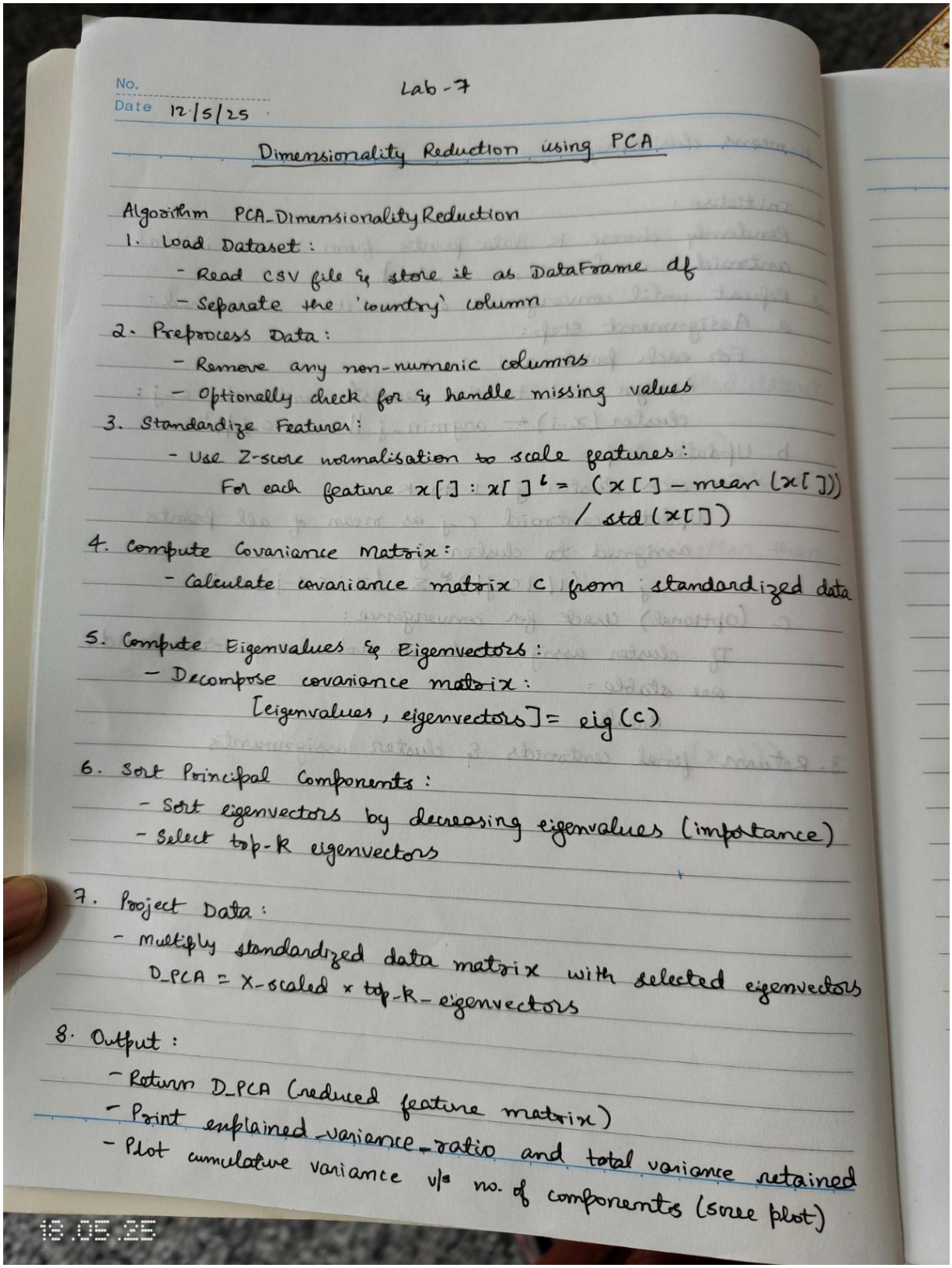

Elbow Method For Optimal k
Inertia
Number of clusters (k)
20000
18000
16000
14000
12000
10000
8000
2
3
4
5
6
7
8
9
10


K-Means Clusters (PCA-reduced)
PC2
Cluster
0
1
2
3
4
5
6
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4
6
8

Program-8:

Implement Dimensionality reduction using Principal Component Analysis (PCA) method.

Observation:



No.
Date 12/5/25

Lab-7

Dimensionality Reduction using PCA

Algorithm PCA_DimensionalityReduction
1. Load Dataset:
   - Read CSV file & store it as DataFrame df
   - Separate the 'country' column
2. Preprocess Data:
   - Remove any non-numeric columns
   - Optionally check for & handle missing values
3. Standardize Features:
   - Use Z-score normalisation to scale features:
     For each feature x[]: x[]' = (x[] - mean(x[]))
     / std(x[])
4. Compute Covariance Matrix:
   - Calculate covariance matrix C from standardized data
5. Compute Eigenvalues & Eigenvectors:
   - Decompose covariance matrix:
     [eigenvalues, eigenvectors] = eig(C)
6. Sort Principal Components:
   - Sort eigenvectors by decreasing eigenvalues (importance)
   - Select top-K eigenvectors
7. Project Data:
   - Multiply standardized data matrix with selected eigenvectors
     D_PCA = X_scaled x top-K-eigenvectors
8. Output:
   - Return D_PCA (reduced feature matrix)
   - Print explained_variance_ratio and total variance retained
   - Plot cumulative variance v/s no. of components (scree plot)

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# === STEP 1: Load the dataset ===
df = pd.read_csv("/content/drive/MyDrive/Sem-6/ML/Lab7/Country-data.csv")

# === STEP 2: Separate and remove the non-numeric 'country' column ===
countries = df['country']  # Save for optional labeling
X = df.drop(columns=['country'])  # Only keep numerical features

# === STEP 3: Standardize the features ===
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# === STEP 4: Apply PCA (2 components) ===
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# === STEP 5: Print explained variance ===
print("Explained Variance Ratio (per PC):", pca.explained_variance_ratio_)
print("Total Variance Explained:", np.sum(pca.explained_variance_ratio_))

# === STEP 6: Plot PCA with country labels ===
plt.figure(figsize=(12, 8))
plt.scatter(X_pca[:, 0], X_pca[:, 1], alpha=0.7, color='royalblue')
```

```
# Annotate each point with the corresponding country name
for i, country in enumerate(countries):
    plt.text(X_pca[i, 0]+0.05, X_pca[i, 1]+0.05, country, fontsize=8, alpha=0.7)

plt.title("PCA Projection of Country Dataset", fontsize=14)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Output: