VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Tanya D Shetty (1BM22CS337)

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
Sep-2024 to Jan-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 **Tanya D Shetty (1BM22CS337),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Radhika A D Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

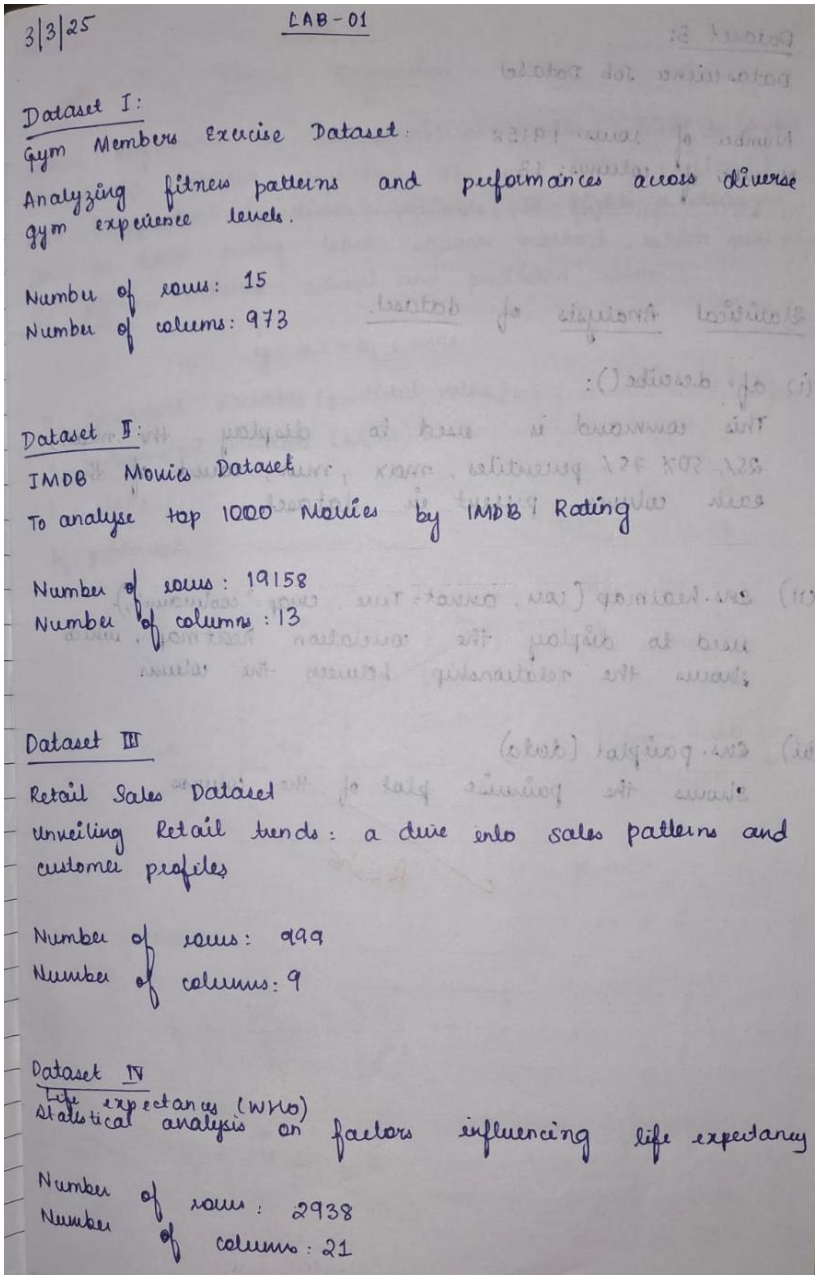
Index

Github Link: https://github.com/ShettyTanya/ML_Lab

Program 1

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

Screenshot



3/3/25 LAB-01

Dataset I:
Gym Members Exercise Dataset
Analyzing fitness patterns and performances across diverse gym experience levels.

Number of rows: 15
Number of columns: 973

Dataset II:
IMDB Movies Dataset
To analyse top 1000 Movies by IMDB Rating

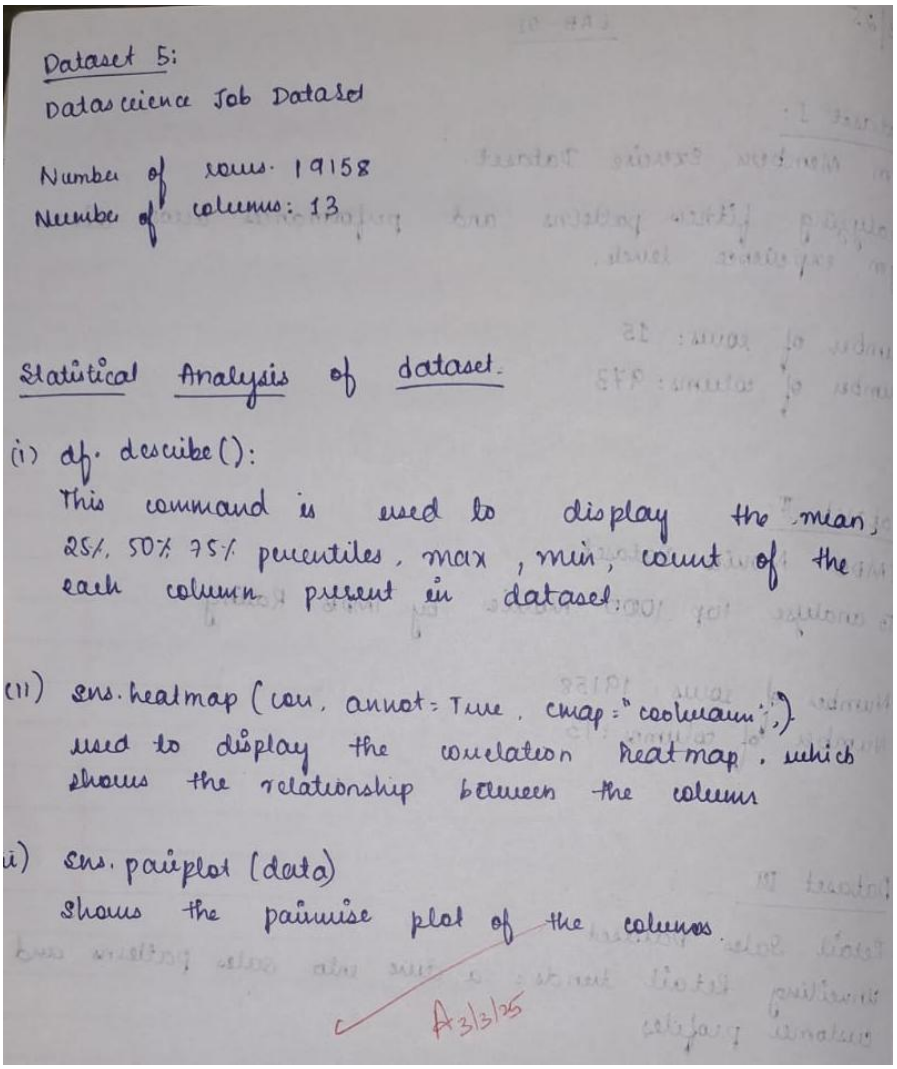
Number of rows: 19158
Number of columns: 13

Dataset III
Retail Sales Dataset
Unveiling Retail trends: a dive into sales patterns and customer profiles

Number of rows: 999
Number of columns: 9

Dataset IV
Life expectancy (WHO)
Statistical analysis on factors influencing life expectancy

Number of rows: 2938
Number of columns: 21



Dataset 5:
Data science Job Dataset

Number of rows: 19158
Number of columns: 13

Statistical Analysis of dataset.

(i) df.describe():
This command is used to display the mean, 25%, 50% 75% percentiles, max, min, count of the each column present in dataset.

(ii) sns.heatmap(cor, annot=True, cmap="coolwarm";)
used to display the correlation heatmap, which shows the relationship between the columns

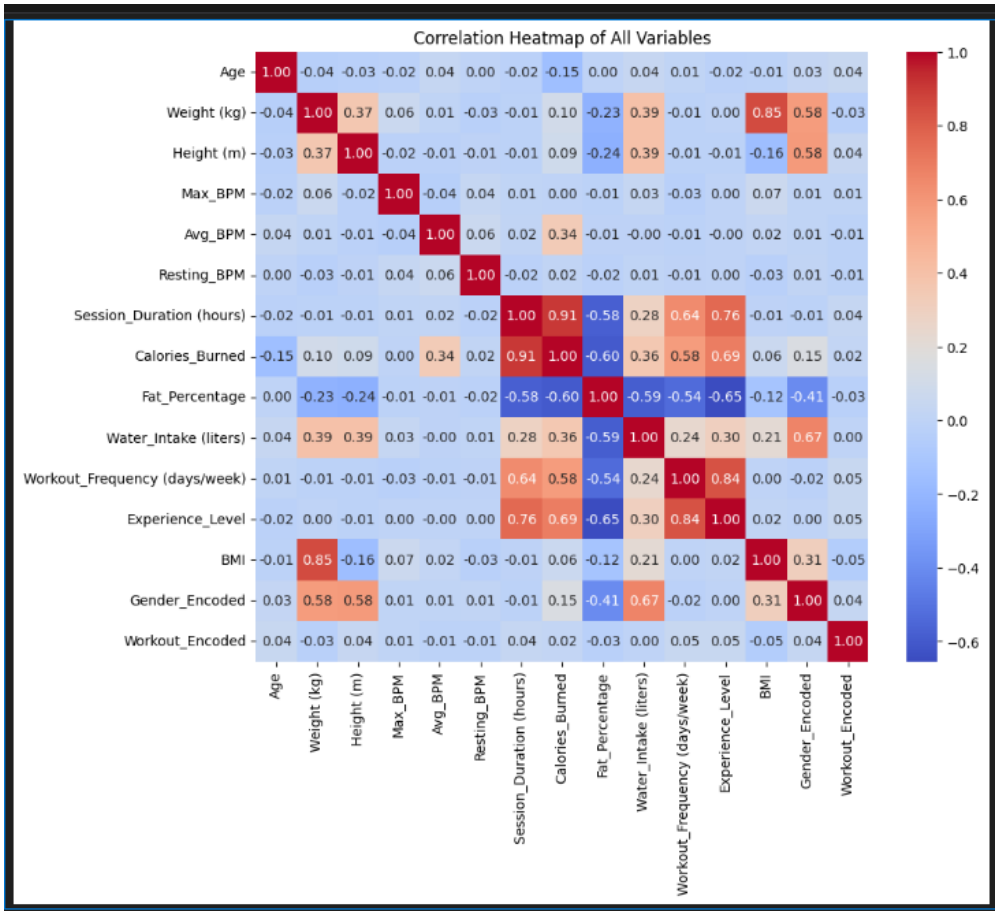
ii) sns.pairplot(data)
shows the pairwise plot of the columns.

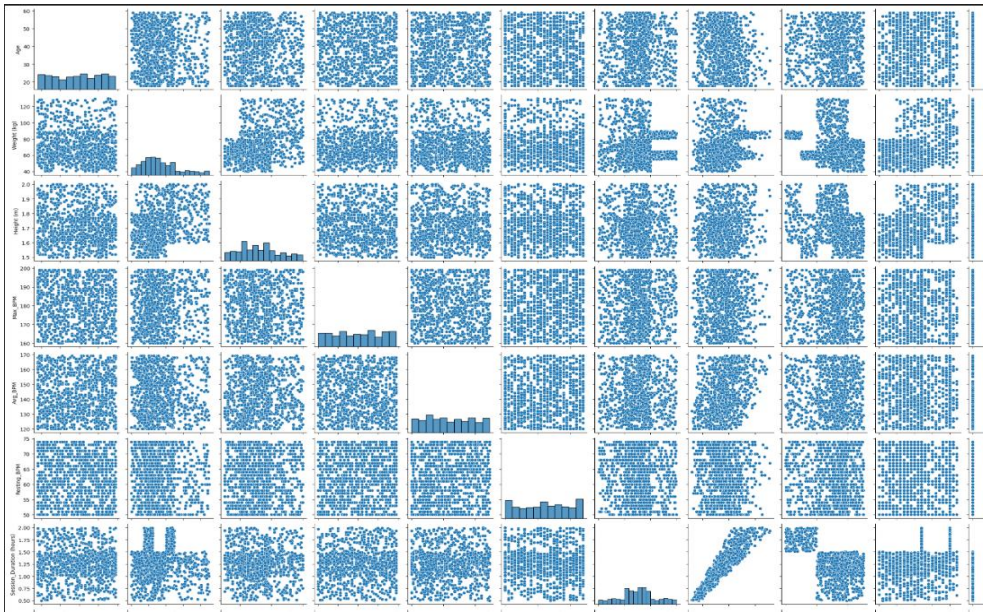
A 3/3/25

Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder

# Load the dataset
file_path = "/gym_members_exercise_tracking.csv"
df = pd.read_csv(file_path)

# Encode categorical variables
encoder = LabelEncoder()
df['Gender_Encoded'] = encoder.fit_transform(df['Gender'])
df['Workout_Encoded'] = encoder.fit_transform(df['Workout_Type'])

# Drop original categorical columns and use encoded ones
encoded_df = df.drop(columns=['Gender', 'Workout_Type'])
```

```
# Compute correlation matrix
corr = encoded_df.corr()

# Plot heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap of All Variables")
plt.show()

# Pair plot
sns.pairplot(encoded_df)
plt.show()
```

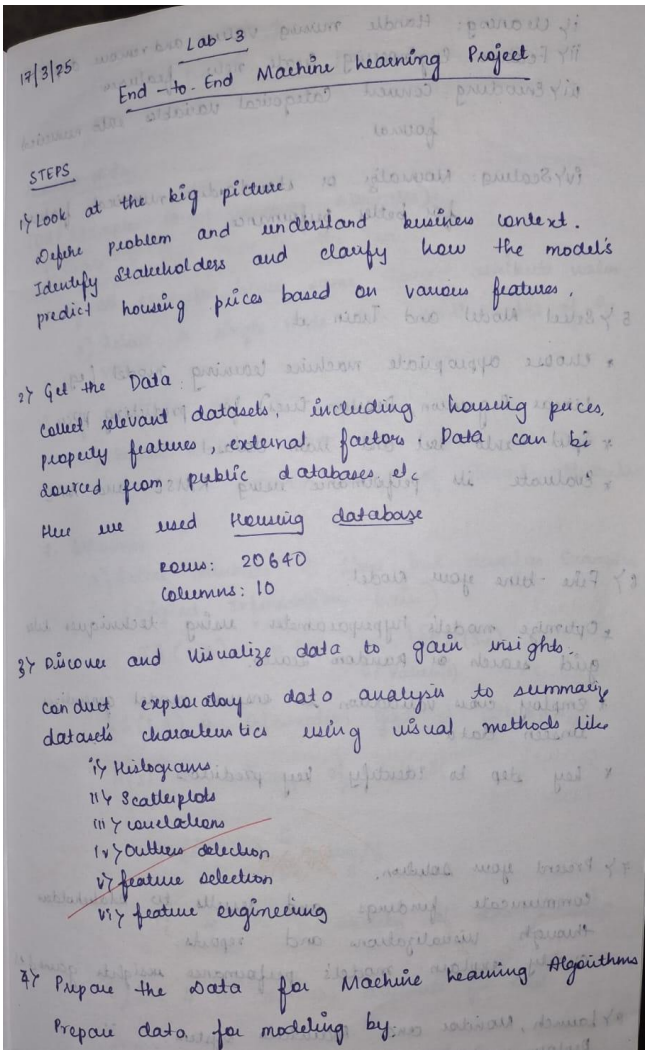



Age
Weight (kg)
Height (m)
Avg_BPM
Avg_BPM
Resting_BPM
Session_Duration (hours)

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot



17/3/25 Lab - 3

End - to - End Machine learning Project

STEPS

1) Look at the big picture

Define problem and understand business context.
Identify Stakeholders and clarify how the model's predict housing prices based on various features.

2) Get the Data

Collect relevant datasets, including housing prices, property features, external factors. Data can be sourced from public databases etc

Here we used Housing database

rows: 20640
columns: 10

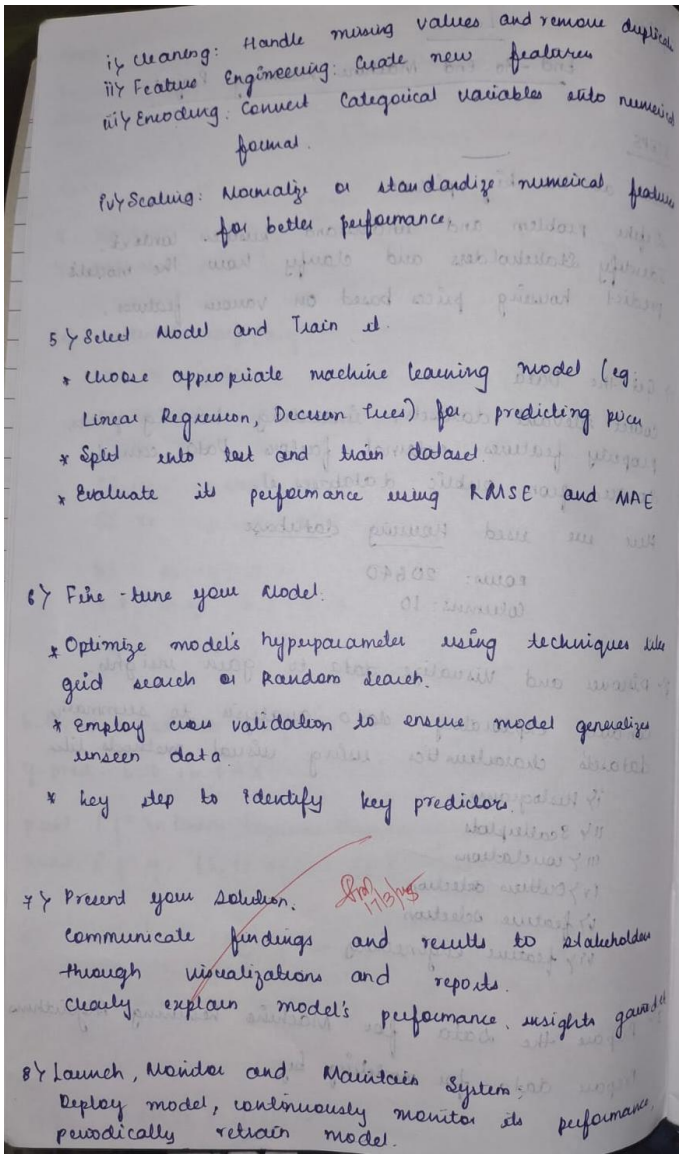
3) Discover and visualize data to gain insights.

Conduct exploratory data analysis to summarize datasets characteristics using visual methods like

i) Histograms
ii) Scatterplots
iii) correlations
iv) Outliers detection
v) feature selection
vi) feature engineering

4) Prepare the data for Machine learning Algorithms

Prepare data for modeling by:


i) Cleaning: Handle missing values and remove duplicat
ii) Feature Engineering: Create new features
iii) Encoding: Convert Categorical variables into numerica
format.
iv) Scaling: Normalize or standardize numerical featur
for better performance.
5) Select Model and Train it
* Choose appropriate machine learning model (eg:
Linear Regression, Decision Trees) for predicting pric
* Split into test and train dataset
* Evaluate its performance using RMSE and MAE
6) Fine-tune your Model
* Optimize model's hyperparameter using techniques like
grid search or Random search.
* Employ cross validation to ensure model generalize
unseen data.
* key step to identify key predictors.
7) Present your Solution.
Communicate findings and results to stakeholder
through visualizations and reports.
Clearly explain model's performance, insights gaine
8) Launch, Monitor and Maintain System:
Deploy model, continuously monitor its performance
periodically retrain model.

Code:

```
import os
import tarfile
import urllib.request

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):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    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()
fetch_housing_data()
```

```
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 = load_housing_data()
housing.head()
```

	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

```
housing.info()
```

```
<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
```

```
housing["ocean_proximity"].value_counts()
```

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

ype: int64

```
housing.describe()
```

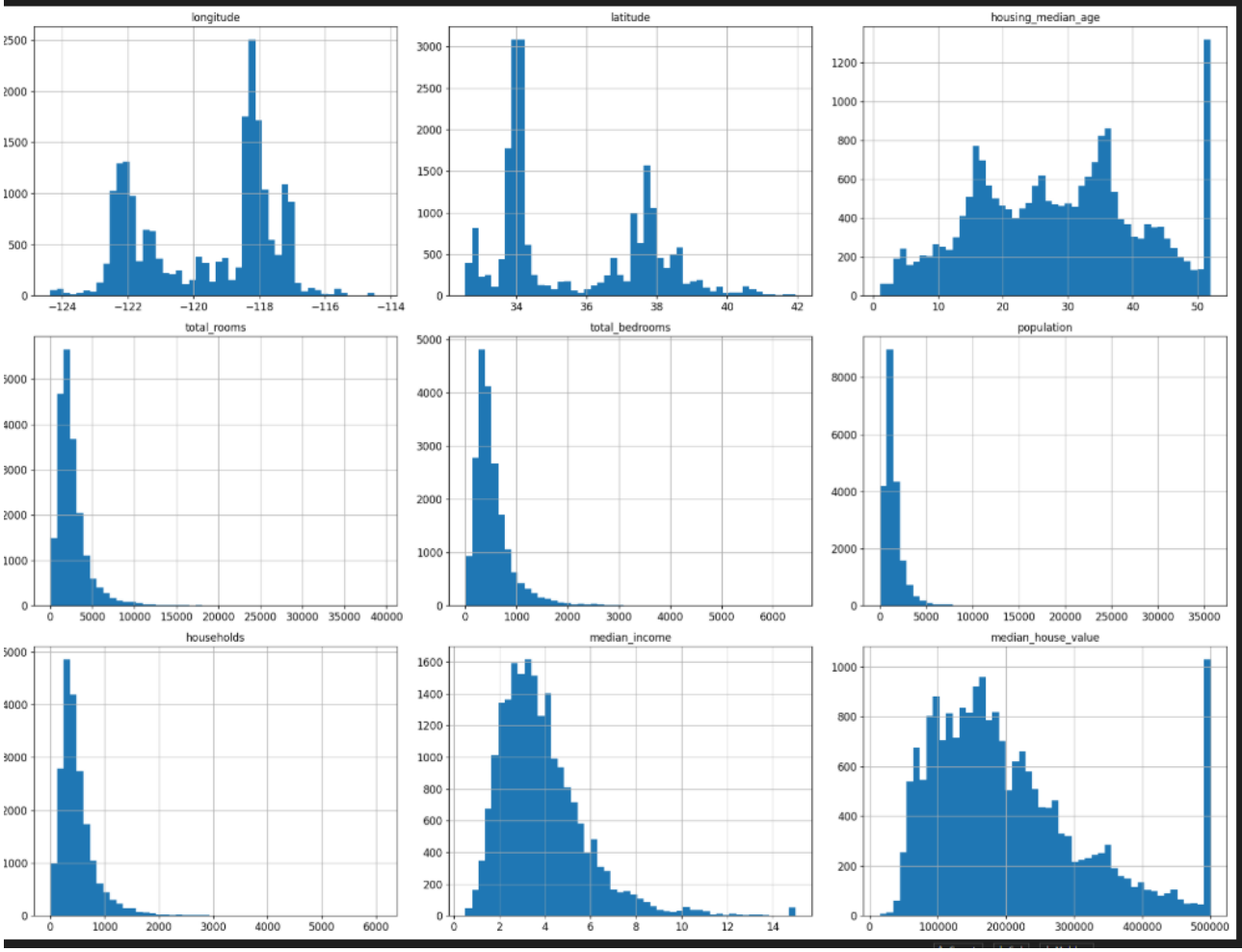
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
unt	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
ean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
save_fig("attribute_histogram_plots")
plt.show()
```

```
ving figure attribute histogram plots
```

```
%matplotlib inline
import matplotlib.pyplot as plt
housing.hist(bins=50, figsize=(20,15))
save_fig("attribute_histogram_plots")
plt.show()
```

```
wing figure attribute_histogram_plots
```



```
np.random.seed(42)
```

```
import numpy as np

# For illustration only. Sklearn has train_test_split()
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]
```

```
train_set, test_set = split_train_test(housing, 0.2)
len(train_set)
```

```
16512
```

```
len(test_set)
```

```
4128
```

```
from zlib import crc32

def test_set_check(identifier, test_ratio):
    return crc32(np.int64(identifier)) & 0xffffffff < test_ratio * 2**32

def split_train_test_by_id(data, test_ratio, id_column):
    ids = data[id_column]
    in_test_set = ids.apply(lambda id_: test_set_check(id_, test_ratio))
    return data.loc[~in_test_set], data.loc[in_test_set]
```

```
import hashlib

def test_set_check(identifier, test_ratio, hash=hashlib.md5):
    return hash(np.int64(identifier)).digest()[-1] < 256 * test_ratio
```

```
def test_set_check(identifier, test_ratio, hash=hashlib.md5):
    return bytearray(hash(np.int64(identifier)).digest())[-1] < 256 * test_ratio
```

```
housing_with_id = housing.reset_index()   # adds an `index` column
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "index")
```

```
housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "id")
```

```
housing_with_id = housing.reset_index()   # adds an `index` column
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "index")
```

```
housing_with_id["id"] = housing["longitude"] * 1000 + housing["latitude"]
train_set, test_set = split_train_test_by_id(housing_with_id, 0.2, "id")
```

```
test_set.head()
```

	index	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity	id
8	8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0	NEAR BAY	-122222.16
10	10	-122.26	37.85	52.0	2202.0	434.0	910.0	402.0	3.2031	281500.0	NEAR BAY	-122222.15
11	11	-122.26	37.85	52.0	3503.0	752.0	1504.0	734.0	3.2705	241800.0	NEAR BAY	-122222.15
12	12	-122.26	37.85	52.0	2491.0	474.0	1098.0	468.0	3.0750	213500.0	NEAR BAY	-122222.15
13	13	-122.26	37.84	52.0	696.0	191.0	345.0	174.0	2.6736	191300.0	NEAR BAY	-122222.16

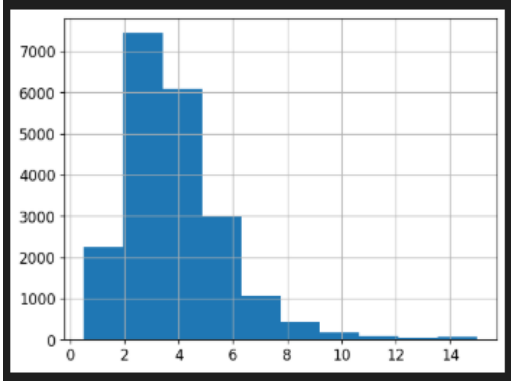
```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
```

```
test_set.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
20046	-119.01	36.06	25.0	1505.0	NaN	1392.0	359.0	1.6812	47700.0	INLAND
3024	-119.46	35.14	30.0	2943.0	NaN	1565.0	584.0	2.5313	45800.0	INLAND
15663	-122.44	37.80	52.0	3830.0	NaN	1310.0	963.0	3.4801	500001.0	NEAR BAY
20484	-118.72	34.28	17.0	3051.0	NaN	1705.0	495.0	5.7376	218600.0	<1H OCEAN
9814	-121.93	36.62	34.0	2351.0	NaN	1063.0	428.0	3.7250	278000.0	NEAR OCEAN

```
housing["median_income"].hist()
```

```
<Axes: >
```



```
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
```

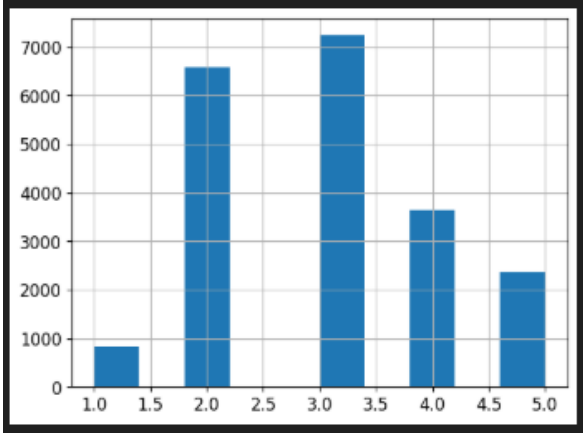
```
housing["income_cat"].value_counts()
```

income_cat	count
3	7236
2	6581
4	3639
5	2362
1	822

dtype: int64

```
housing["income_cat"].hist()
```

<Axes: >




```
from sklearn.model_selection import StratifiedShuffleSplit

split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=42)
for train_index, test_index in split.split(housing, housing["income_cat"]):
    strat_train_set = housing.loc[train_index]
    strat_test_set = housing.loc[test_index]
```

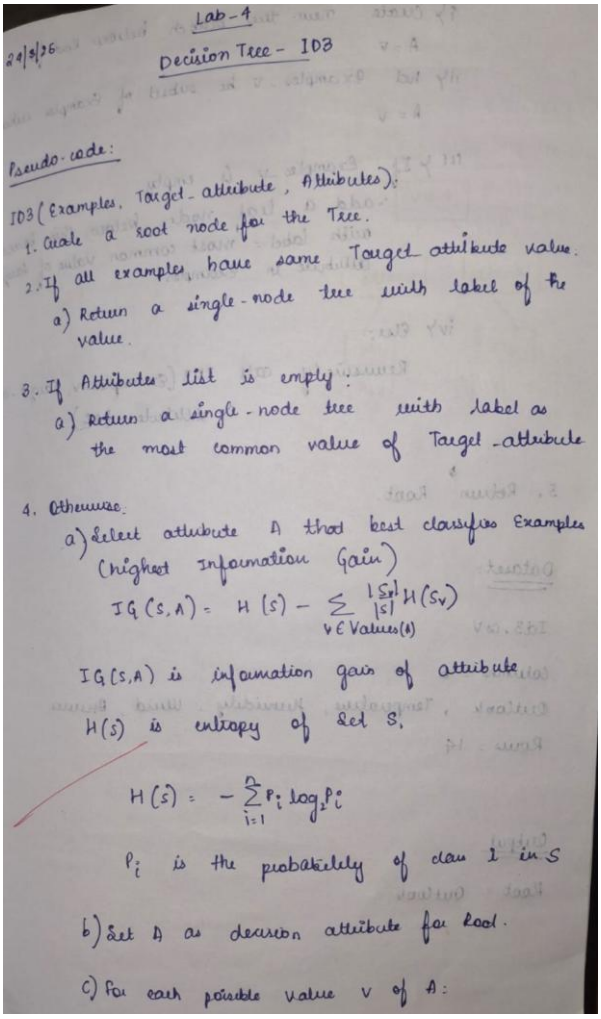
```
strat_test_set["income_cat"].value_counts() / len(strat_test_set)
```

income_cat	count
3	0.350533
2	0.318798
4	0.176357
5	0.114341

Program 3

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

Screenshot



Lab-4

24/9/26

Decision Tree - ID3

Pseudo-code:

ID3 (Examples, Target_attribute, Attributes):

1. Create a root node for the Tree.
2. If all examples have same Target attribute value.
   a) Return a single-node tree with label of the value.
3. If Attributes list is empty:
   a) Return a single-node tree with label as the most common value of Target_attribute
4. Otherwise:
   a) Select attribute A that best classifies Examples (highest Information Gain)

$$IG(S,A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v)$$

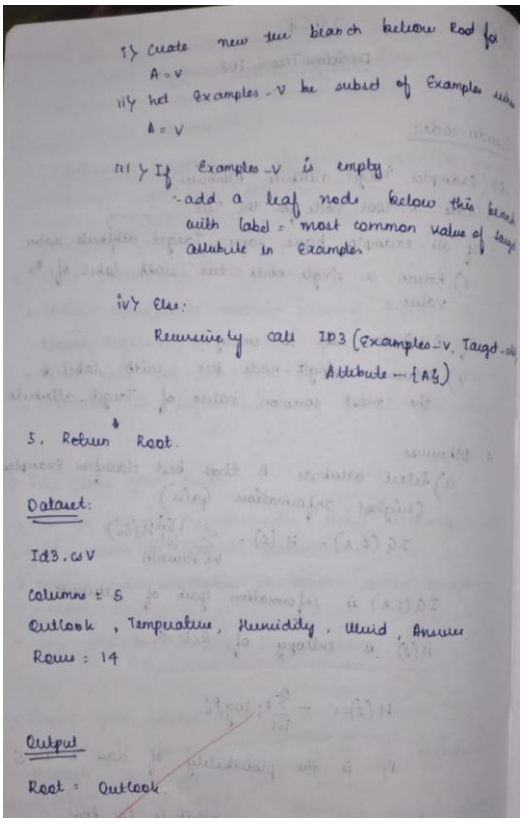
IG(S,A) is information gain of attribute

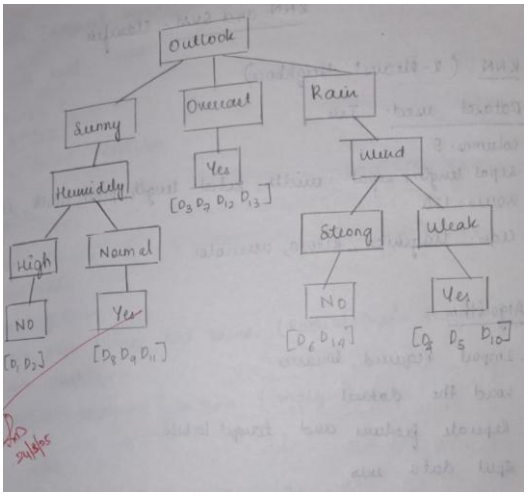
H(S) is entropy of set S.

$$H(S) = -\sum_{i=1}^{n} P_i \log_2 P_i$$

P_i is the probability of class i in S

   b) Set A as decision attribute for root.
   c) For each possible value v of A:





Code:

```
import pandas as pd
import math
from anytree import Node, RenderTree

# Function to calculate entropy
def entropy(target_col):
    counts = target_col.value_counts()
    total = counts.sum()
    return -sum((count/total) * math.log2(count/total) for count in counts)
```

```
# Function to calculate information gain
def information_gain(df, attribute, target="Answer"):
    total_entropy = entropy(df[target])
    values = df[attribute].unique()
    weighted_entropy = sum(
        (len(df[df[attribute] == value]) / len(df)) * entropy(df[df[attribute] ==
value][target])
        for value in values
    )
    return total_entropy - weighted_entropy

# Recursive function to build the decision tree
def id3(df, attributes, target="Answer", parent_node=None, branch_value=None):
    # If all values in the target column are the same, return that value as a leaf node
    if len(df[target].unique()) == 1:
        leaf = Node(f"{branch_value} -> {df[target].iloc[0]}", parent=parent_node)
        return leaf

    # If there are no attributes left, return the most common target value
    if not attributes:
        leaf = Node(f"{branch_value} -> {df[target].mode()[0]}", parent=parent_node)
        return leaf

    # Choose the best attribute
    gains = {attr: information_gain(df, attr, target) for attr in attributes}
    best_attribute = max(gains, key=gains.get)

    # Create root node if it's the first call
    if parent_node is None:
        root = Node(best_attribute)
    else:
        root = Node(f"{branch_value} -> {best_attribute}", parent=parent_node)

    # Split data based on best attribute and recurse
    for value in df[best_attribute].unique():
        subset = df[df[best_attribute] == value]
        new_attributes = [attr for attr in attributes if attr != best_attribute]
        id3(subset, new_attributes, target, root, value)

    return root

# Load dataset from CSV
file_path = "/content/id3.csv"  # Ensure you have the correct file path
df = pd.read_csv(file_path)
```

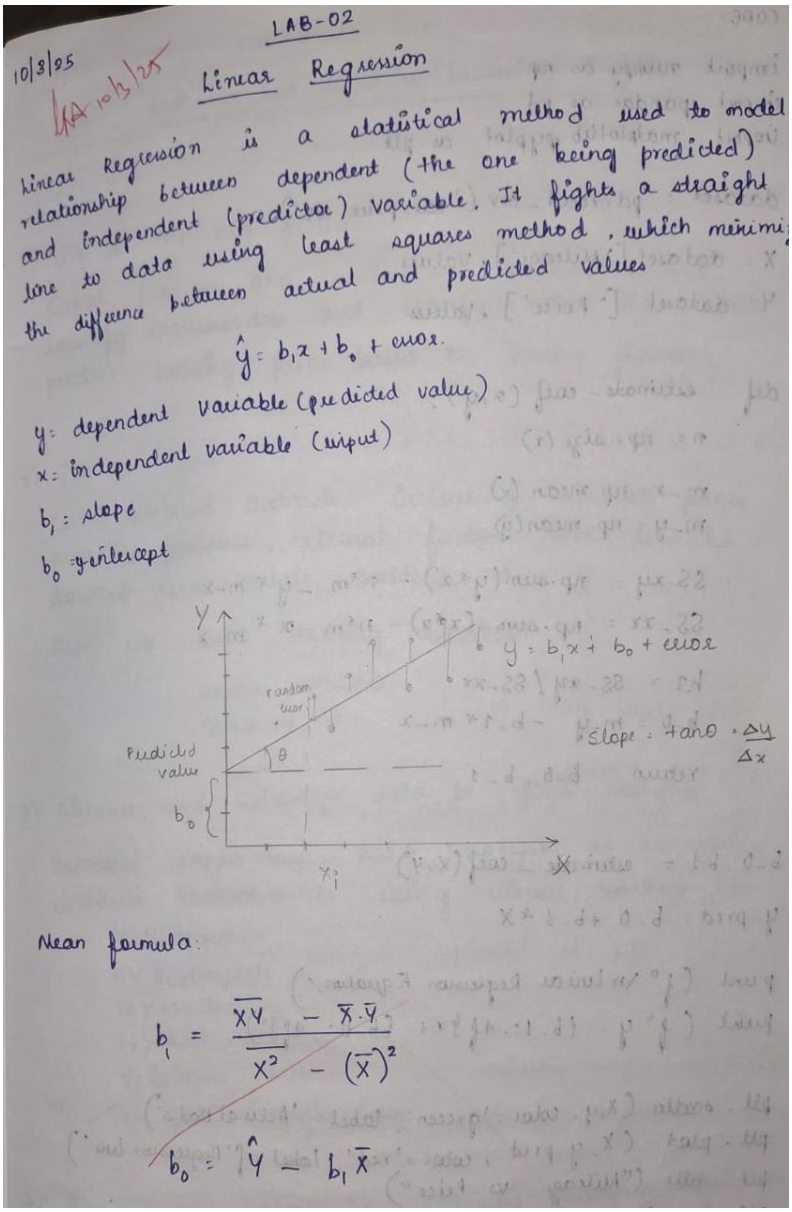
```python
# Define attributes (excluding target column)
target = "Answer"  # Ensure the target column matches your dataset
attributes = [col for col in df.columns if col != target]

# Build decision tree and visualize
decision_tree_root = id3(df, attributes, target)

# Display decision tree in a hierarchical format
print("\nDecision Tree Structure:")
for pre, _, node in RenderTree(decision_tree_root):
    print(f"{pre}{node.name}")
```

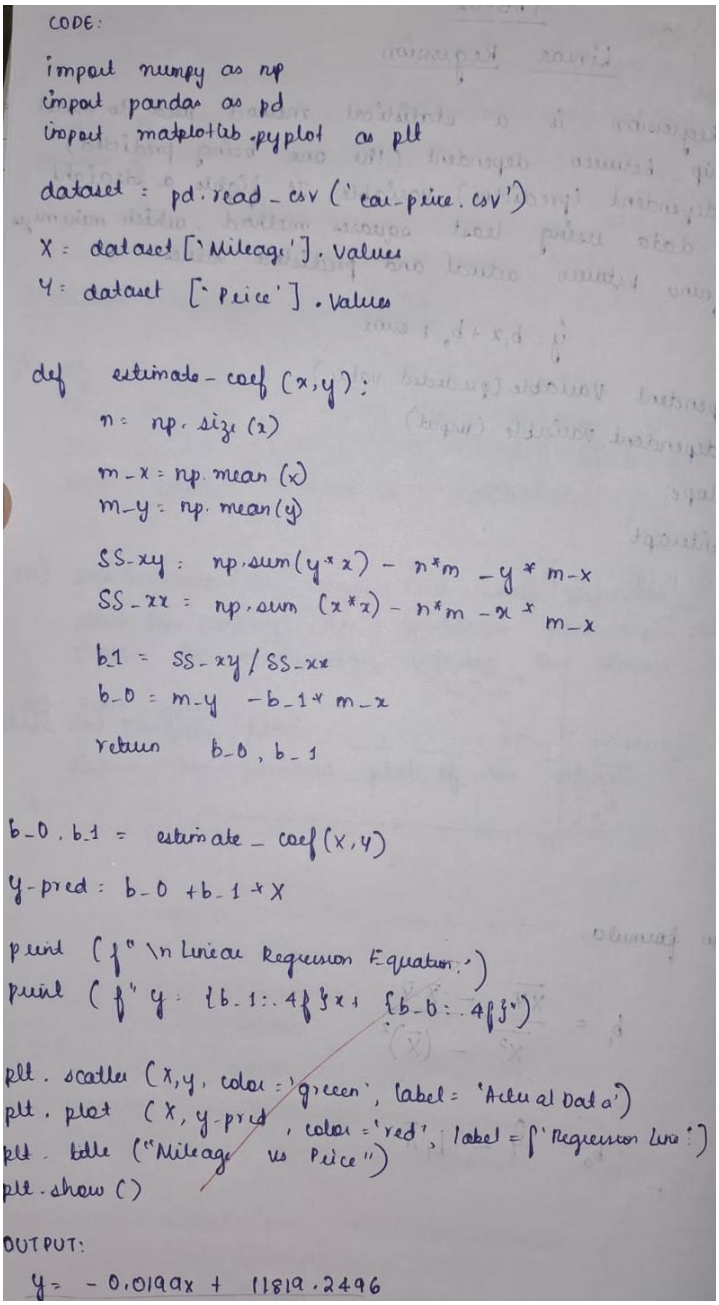
```
Decision Tree Structure:
Outlook
├── sunny -> Humidity
│   ├── high -> no
│   └── normal -> yes
├── overcast -> yes
└── rain -> Wind
    ├── weak -> yes
    └── strong -> no
```

Program 4

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot





Code:

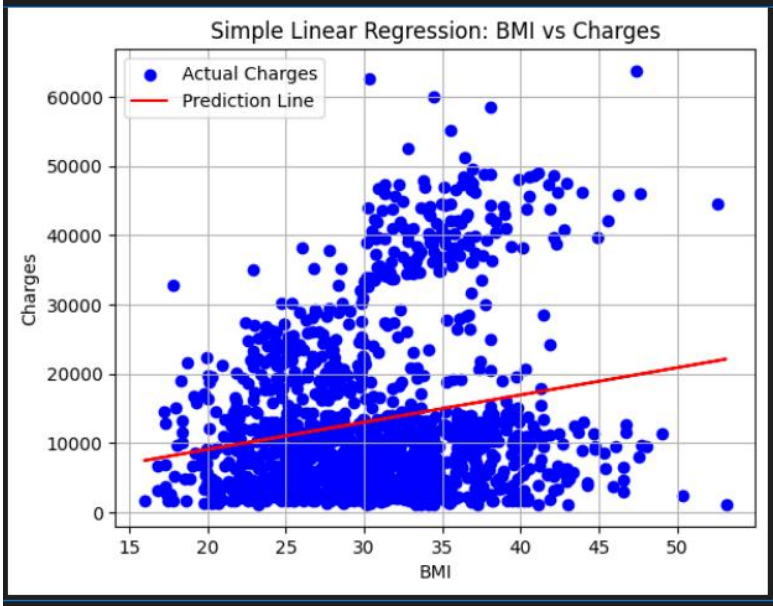
Linear:

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

# Load your dataset
df = pd.read_csv('/content/sample_data/insurance.csv')  # Change path as needed

# Selecting one independent variable (e.g., bmi) and target (charges)
```

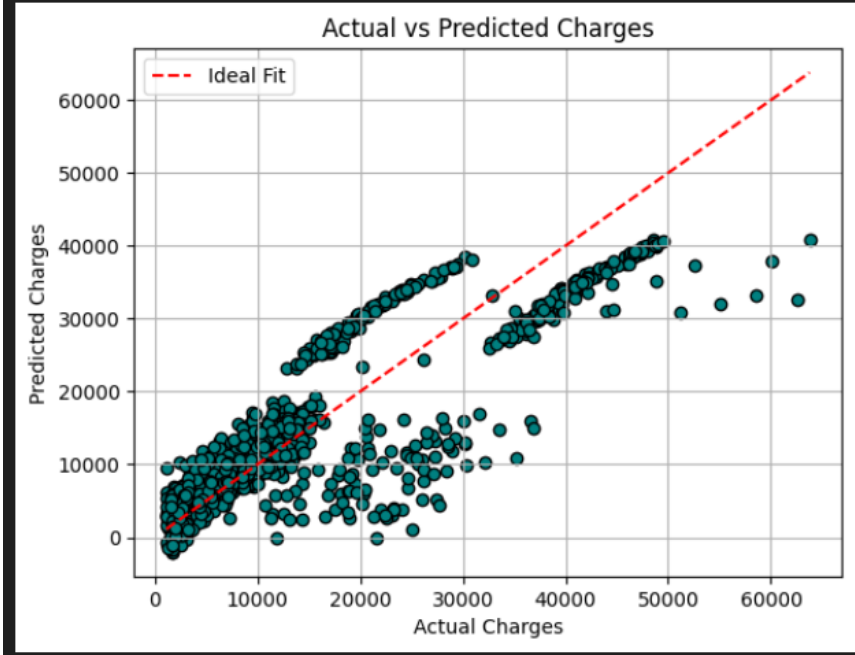
```
x = df['bmi'].values
y = df['charges'].values

# Convert to numpy arrays and reshape if needed
x = np.array(x)
y = np.array(y)

# Number of data points
n = len(x)

# Calculate means
mean_x = np.mean(x)
mean_y = np.mean(y)

# Calculate the coefficients
SS_xy = np.sum((x - mean_x) * (y - mean_y))
SS_xx = np.sum((x - mean_x) ** 2)

b1 = SS_xy / SS_xx
b0 = mean_y - b1 * mean_x

# Print regression equation
print(f"\n ◇  Linear Regression Equation:")
print(f"charges = {b1:.4f} * bmi + {b0:.4f}")

# Predict y values using regression line
y_pred = b0 + b1 * x

# Plot actual data points
plt.scatter(x, y, color='blue', label='Actual Charges')
# Plot regression line
plt.plot(x, y_pred, color='red', label=f'Prediction Line')
plt.xlabel('BMI')
plt.ylabel('Charges')
plt.title('Simple Linear Regression: BMI vs Charges')
plt.legend()
plt.grid(True)
plt.show()

multi-linear:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('/content/sample_data/insurance.csv')
```

```
# Encode categorical variables (One-hot encoding)
df_encoded = pd.get_dummies(df, drop_first=True)

# Convert all values to float (important!)
df_encoded = df_encoded.astype(float)

# Separate independent (X) and dependent (y) variables
X = df_encoded.drop(columns=['charges']).values
y = df_encoded['charges'].values.reshape(-1, 1)

# Add bias term (intercept column)
ones = np.ones((X.shape[0], 1))
X_b = np.hstack((ones, X))

# Apply the Normal Equation using pseudo-inverse to avoid singular matrix error
theta = np.linalg.pinv(X_b.T @ X_b) @ X_b.T @ y

# Print the learned coefficients
print("\n ◇  Multiple Linear Regression Coefficients:")
feature_names = ['Intercept'] + list(df_encoded.drop(columns=['charges']).columns)
for name, coef in zip(feature_names, theta.flatten()):
    print(f"{name}: {coef:.4f}")

# Predict charges using the model
y_pred = X_b @ theta

# Plot actual vs predicted charges
plt.scatter(y, y_pred, color='teal', edgecolors='k')
plt.plot([y.min(), y.max()], [y.min(), y.max()], color='red', linestyle='--',
label='Ideal Fit')
plt.xlabel('Actual Charges')
plt.ylabel('Predicted Charges')
plt.title('Actual vs Predicted Charges')
plt.legend()
plt.grid(True)
plt.show()
```

🔹 Linear Regression Equation:
charges = 393.8730 * bmi + 1192.9372



🔹 Multiple Linear Regression Coefficients:
Intercept: -11938.5386
age: 256.8564
bmi: 339.1935
children: 475.5005
sex_male: -131.3144
smoker_yes: 23848.5345
region_northwest: -352.9639
region_southeast: -1035.0220
region_southwest: -960.0510



Program 5

Build Logistic Regression Model for a given dataset

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Sample Dataset
data = {
    'Feature1': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    'Feature2': [1, 1, 2, 3, 3, 4, 5, 6, 7, 8],
    'Target':   [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]
}

df = pd.DataFrame(data)

X = df[['Feature1', 'Feature2']]
y = df['Target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)

model = LogisticRegression()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

print("Accuracy: {:.2f}%".format(accuracy_score(y_test, y_pred) * 100))
print(classification_report(y_test, y_pred, target_names=["Class 0", "Class 1"]))

sample = pd.DataFrame([[5, 2]], columns=['Feature1', 'Feature2'])
prediction = model.predict(sample)
print(f"Prediction for Feature1=5, Feature2=2: {'Class 1' if prediction[0] == 1 else
'Class 0'}")

# Plot decision boundary
x_min, x_max = X['Feature1'].min() - 1, X['Feature1'].max() + 1
y_min, y_max = X['Feature2'].min() - 1, X['Feature2'].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))
```

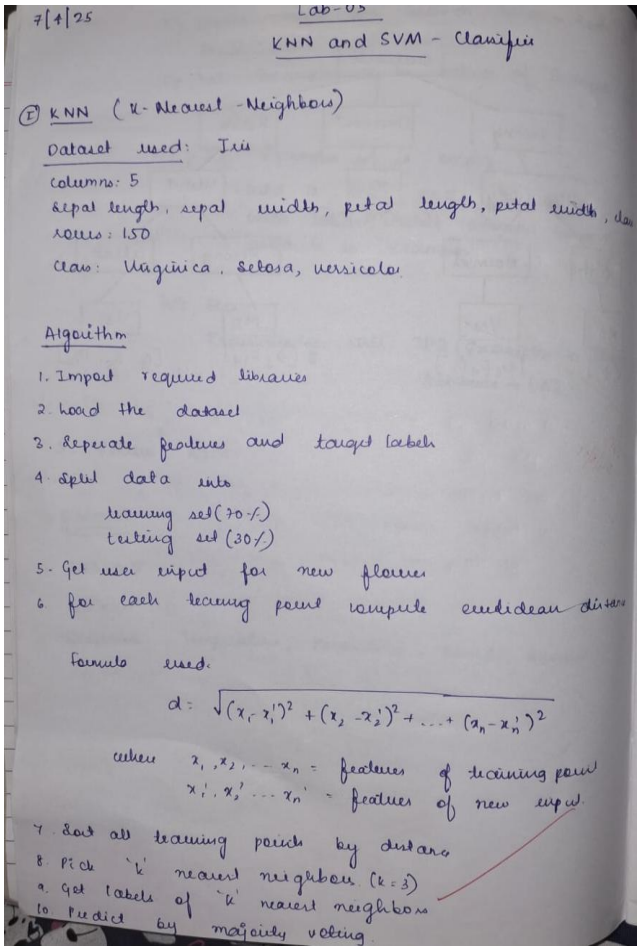
```python
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.3, cmap=plt.cm.Paired)
plt.scatter(X['Feature1'], X['Feature2'], c=y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.title('Logistic Regression Decision Boundary')
plt.show()
```

```
Accuracy: 66.67%
              precision    recall  f1-score   support

     Class 0       0.50      1.00      0.67         1
     Class 1       1.00      0.50      0.67         2

    accuracy                           0.67         3
   macro avg       0.75      0.75      0.67         3
weighted avg       0.83      0.67      0.67         3

Prediction for Feature1=5, Feature2=2: Class 0
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: Use
  warnings.warn(
```

Program 6

Build KNN Classification model for a given dataset.

Screenshot



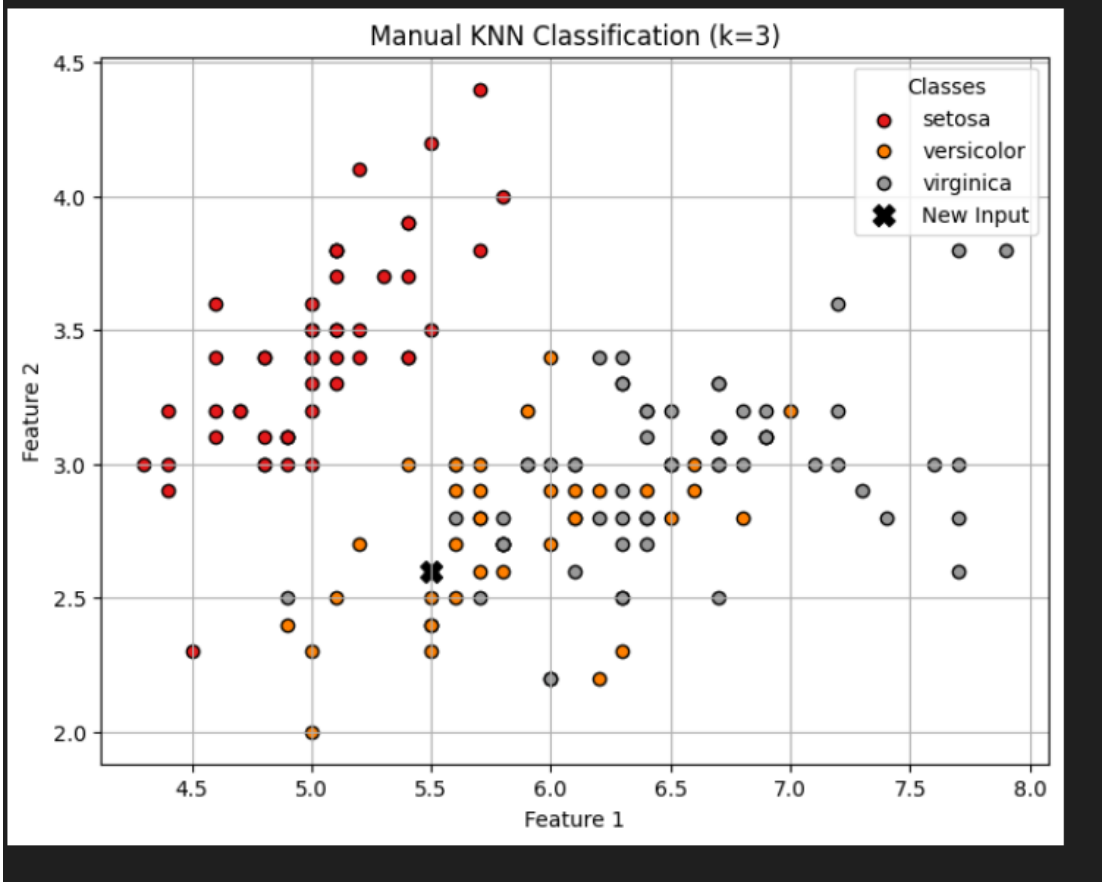
Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from collections import Counter

# Load dataset
df = pd.read_csv('/content/iris.csv')  # Replace with your dataset

# Separate features and labels
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values

# For plotting (first 2 features only)
X_plot = X[:, :2]
```

```
# Train/test split (70% train, 30% test)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# ---- USER INPUT SECTION ----
print("\nEnter values for a new data point:")
feature1 = float(input("Feature 1: "))
feature2 = float(input("Feature 2: "))
feature3 = float(input("Feature 3: "))
feature4 = float(input("Feature 4: "))

new_input = np.array([feature1, feature2, feature3, feature4])

# ---- MANUAL KNN IMPLEMENTATION ----
k = 3  # Number of neighbors

# Calculate Euclidean distance from new input to all training points
distances = []
for i in range(len(X_train)):
    dist = np.sqrt(np.sum((X_train[i] - new_input) ** 2))
    distances.append((dist, y_train[i]))

# Sort by distance
sorted_neighbors = sorted(distances, key=lambda x: x[0])

# Pick k nearest neighbors
k_neighbors = sorted_neighbors[:k]

# Perform majority vote
k_labels = [label for _, label in k_neighbors]
predicted_label = Counter(k_labels).most_common(1)[0][0]

print(f"\nPredicted Class: {predicted_label}")

# ---- PLOT DATA + NEW POINT (Only First Two Features) ----
plt.figure(figsize=(8, 6))
unique_classes = np.unique(y)
colors = plt.cm.get_cmap('Set1', len(unique_classes))

# Plot existing data
for idx, cls in enumerate(unique_classes):
    plt.scatter(
        X_plot[y == cls, 0], X_plot[y == cls, 1],
        color=colors(idx),
```

```
        label=str(cls),
        edgecolors='k'
    )

# Plot new input point (first two features only)
plt.scatter(feature1, feature2, color='black', marker='X', s=100, label='New Input')

plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Manual KNN Classification (k=3)')
plt.legend(title="Classes")
plt.grid(True)
plt.show()
```

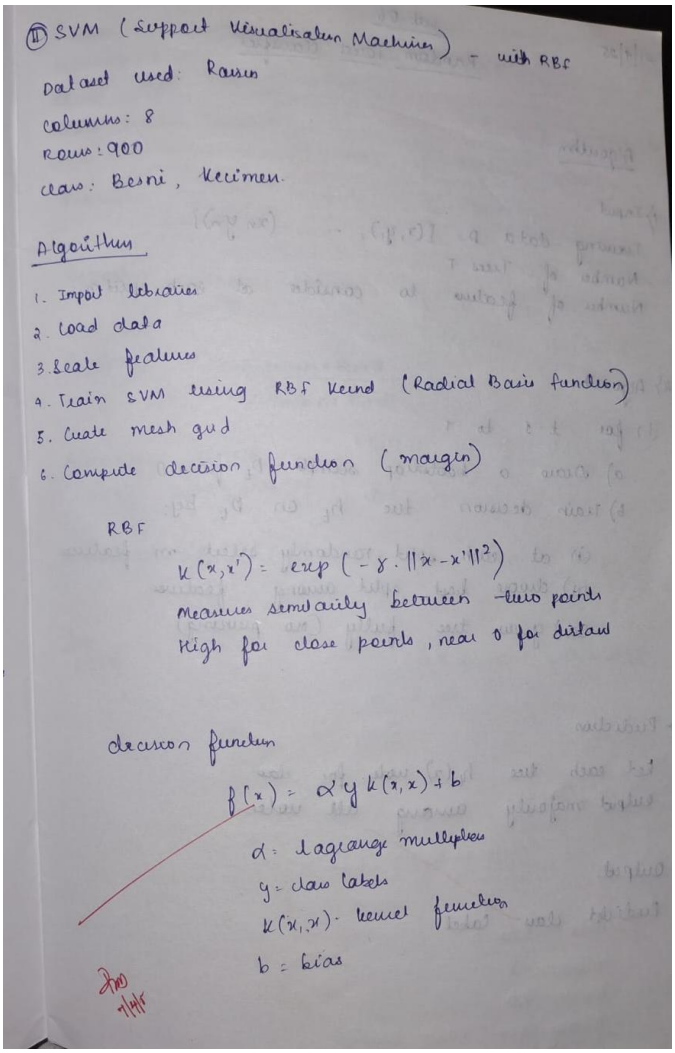
```
Enter values for a new data point:
Feature 1: 5.5
Feature 2: 2.6
Feature 3: 4.2
Feature 4: 1.1

Predicted Class: versicolor
<ipython-input-6-0f8f08f98b6e>:53: MatplotlibDeprecationWarning: The get_cmap function was de
  colors = plt.cm.get_cmap('Set1', len(unique_classes))
```



Program 7

Build Support vector machine model for a given dataset

Screenshot

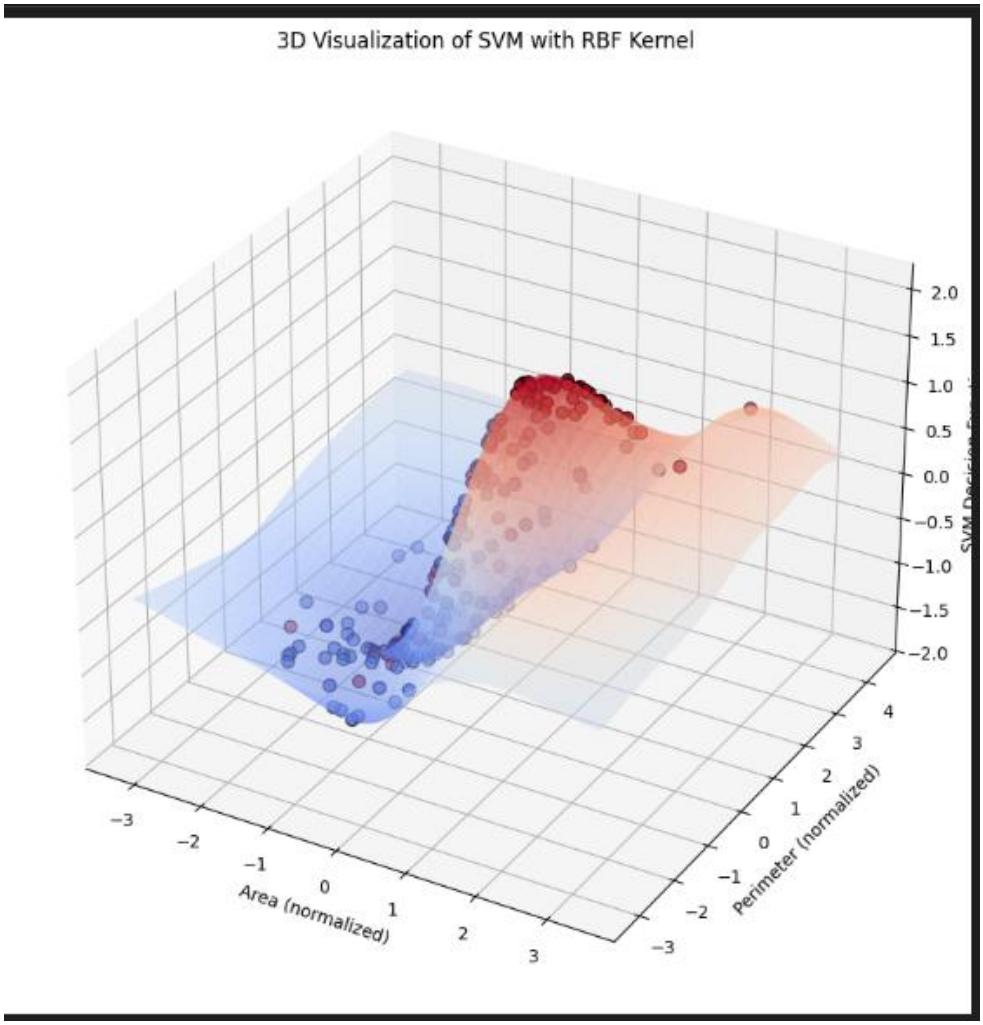


Code:

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

# Generate and normalize 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)
```

```python
# Train SVM with RBF kernel
gamma = 0.5
clf = svm.SVC(kernel='rbf', C=1, gamma=gamma)
clf.fit(X, y)

# Extract model parameters
support_vectors = clf.support_vectors_
dual_coefs = clf.dual_coef_[0]  # Shape: (n_support_vectors,)
intercept = clf.intercept_[0]

# Print out the actual decision function equation
print("\n--- Decision Function (f(x)) Equation ---")
print("f(x) = Σ [αᵢ * exp(-γ * ||x - xᵢ||²)] + b")
print(f"γ = {gamma}")
print(f"b = {intercept:.4f}")
print("\nSupport Vectors and αᵢ coefficients:")

for i, (alpha, vec) in enumerate(zip(dual_coefs, support_vectors)):
    print(f"α[{i}] = {alpha:.4f}, x[{i}] = {vec}")

# 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 decision function for each point
def decision_function_rbf(x1, x2):
    z = np.zeros_like(x1)
    for alpha, sv in zip(dual_coefs, support_vectors):
        z += alpha * np.exp(-gamma * ((x1 - sv[0]) ** 2 + (x2 - sv[1]) ** 2))
    return z + intercept

zz = decision_function_rbf(xx, yy)

# Plot 3D surface
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(xx, yy, zz, cmap='coolwarm', alpha=0.5, linewidth=0)

# Actual 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('Feature 1 (normalized)')
ax.set_ylabel('Feature 2 (normalized)')
```

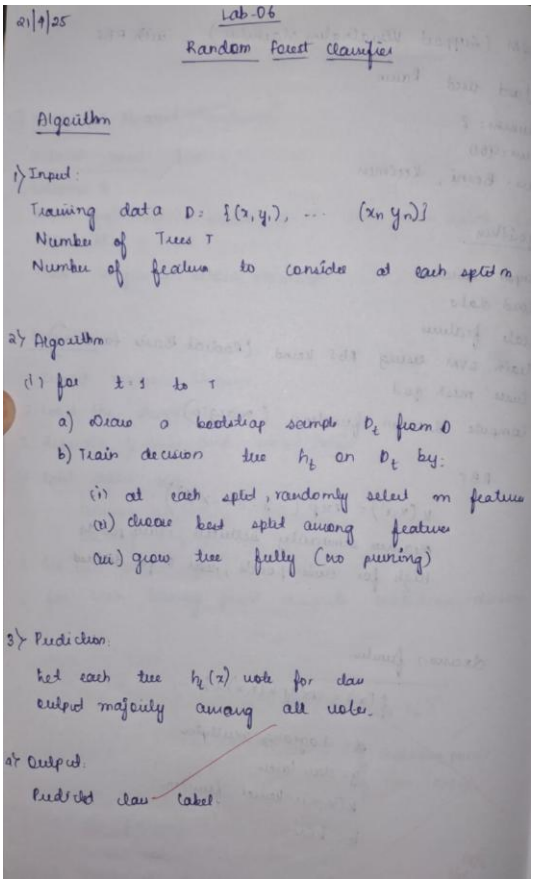
```
ax.set_zlabel('f(x): Decision Function Value')
ax.set_title('SVM Decision Surface with RBF Kernel')

plt.tight_layout()
plt.show()
```



Program 8

Implement Random forest ensemble method on a given dataset.

Screenshot



21/4/25 Lab-06

Random Forest Classifier

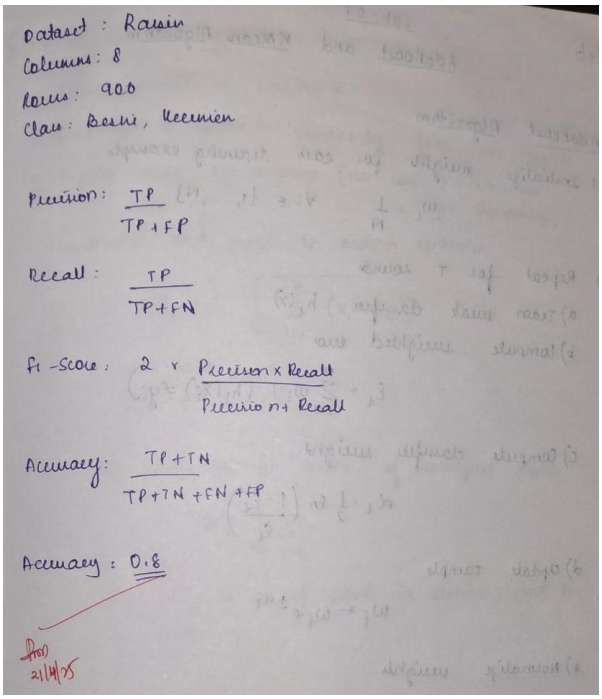
Algorithm

1) Input:
Training data $D = \{(x_1, y_1), \ldots (x_n, y_n)\}$
Number of Trees T
Number of feature to consider at each split m

2) Algorithm
(1) for t = 1 to T
a) Draw a bootstrap sample D_t from D
b) Train decision tree h_t on D_t by:
(i) at each split, randomly select m features
(ii) choose best split among features
(iii) grow tree fully (no pruning)

3) Prediction:
Get each tree $h_t(x)$ vote for class
Output majority among all votes.

4) Output:
Predicted class label



Code:

```
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Load dataset
iris = load_iris()
X = iris.data
y = iris.target
target_names = iris.target_names

# Split (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Random Forest with optimized parameters
clf = RandomForestClassifier(
    n_estimators=200,        # More trees
    max_depth=5,             # Limit tree depth
    min_samples_split=2,
    random_state=42
)
clf.fit(X_train, y_train)

# Predict and evaluate
```

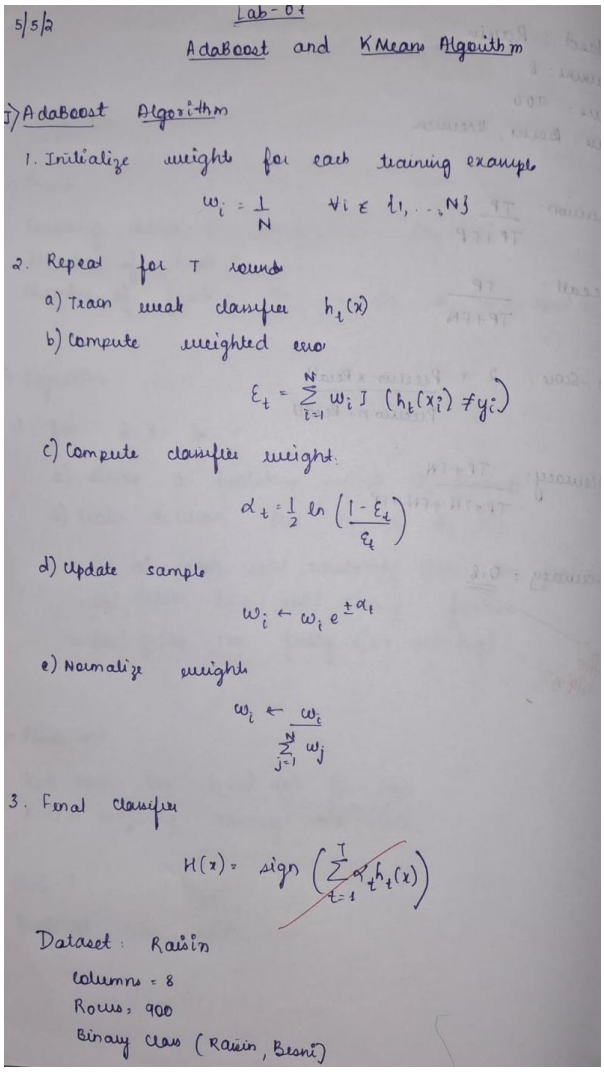
```python
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", round(accuracy * 100, 2), "%")
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Sample prediction
sample_input = X_test[0].reshape(1, -1)
sample_true_label = y_test[0]
sample_pred_label = clf.predict(sample_input)[0]
sample_pred_proba = clf.predict_proba(sample_input)[0]

print("\n Sample Prediction")
print("Input Features:", sample_input[0])
print("True Class:", target_names[sample_true_label])
print("Predicted Class:", target_names[sample_pred_label])
print("Prediction Probabilities:")
for i, prob in enumerate(sample_pred_proba):
    print(f"  {target_names[i]}: {prob:.2f}")
```

```
✅ Accuracy: 100.0 %

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30


🔍 Sample Prediction
Input Features: [6.1 2.8 4.7 1.2]
True Class: versicolor
Predicted Class: versicolor
Prediction Probabilities:
  setosa: 0.00
  versicolor: 0.98
  virginica: 0.02
```

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot



5/5/2 Lab - 01

AdaBoost and KMean Algorithm

1) AdaBoost Algorithm

1. Initialize weight for each training example

$$w_i = \frac{1}{N} \quad \forall i \in \{1, \ldots, N\}$$

2. Repeat for T rounds

a) Train weak classifier $h_t(x)$

b) Compute weighted error

$$\varepsilon_t = \sum_{i=1}^{N} w_i \, I\,(h_t(x_i) \neq y_i)$$

c) Compute classifier weight

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

d) Update sample

$$w_i \leftarrow w_i \, e^{\pm \alpha_t}$$

e) Normalize weights

$$w_i \leftarrow \frac{w_i}{\sum_{j=1}^{N} w_j}$$

3. Final classifier

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Dataset: Raisin

Columns = 8

Rows: 900

Binary class (Raisin, Besni)

Code:

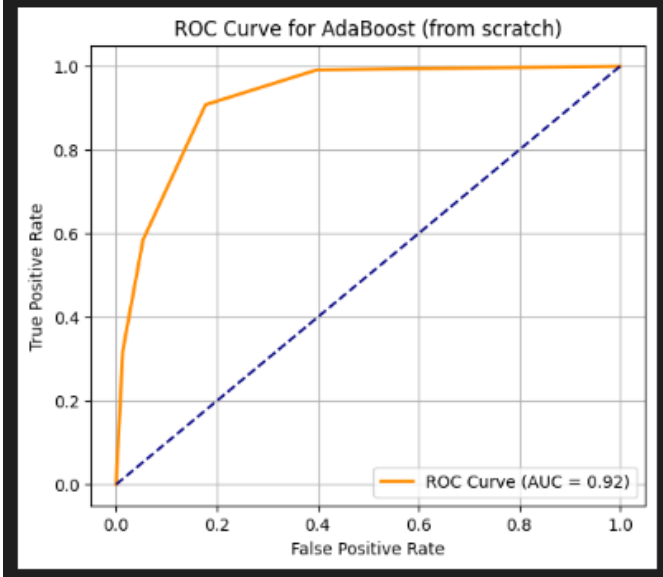
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score,
    f1_score, roc_auc_score, roc_curve
)

# 1. Load the dataset
df = pd.read_excel("/content/Raisin_Dataset.xlsx")
```

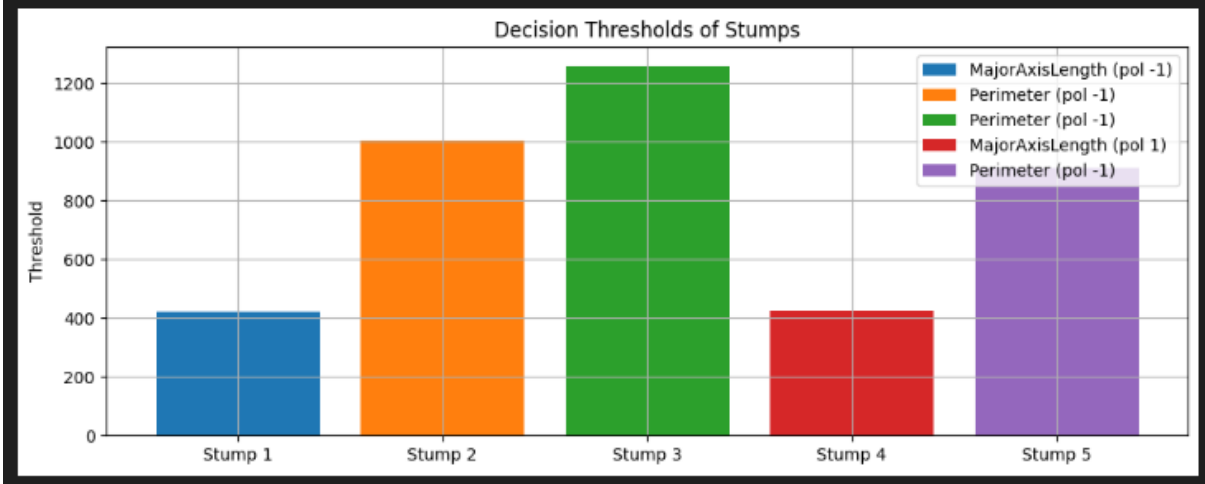
```
# 2. Preprocess: Map class to binary labels
df['Label'] = df['Class'].map({'Kecimen': 1, 'Besni': -1})

# 3. Select features and labels
features = ['Area', 'MajorAxisLength', 'MinorAxisLength', 'Eccentricity',
            'ConvexArea', 'Extent', 'Perimeter']
X = df[features].values
y = df['Label'].values
n_samples, n_features = X.shape

# 4. AdaBoost from scratch
T = 5
weights = np.ones(n_samples) / n_samples
alphas = []
stumps = []

def decision_stump(X, y, weights):
    n_samples, n_features = X.shape
    best_feature, best_threshold, best_polarity, min_error = None, None, None,
float('inf')

    for feature_i in range(n_features):
        feature_values = X[:, feature_i]
        thresholds = np.unique(feature_values)

        for threshold in thresholds:
            for polarity in [1, -1]:
                predictions = np.ones(n_samples)
                predictions[polarity * feature_values < polarity * threshold] = -1
                misclassified = predictions != y
                error = np.sum(weights * misclassified)

                if error < min_error:
                    min_error = error
                    best_feature = feature_i
                    best_threshold = threshold
                    best_polarity = polarity

    return best_feature, best_threshold, best_polarity, min_error

# Training
for t in range(T):
    feature, threshold, polarity, error = decision_stump(X, y, weights)
    alpha = 0.5 * np.log((1 - error + 1e-10) / (error + 1e-10))
```

```
    preds = np.ones(n_samples)
    preds[polarity * X[:, feature] < polarity * threshold] = -1

    weights *= np.exp(-alpha * y * preds)
    weights /= np.sum(weights)

    alphas.append(alpha)
    stumps.append((feature, threshold, polarity))

    print(f"Round {t+1} | Feature: {features[feature]}, Threshold: {threshold:.4f},
Polarity: {polarity}, Error: {error:.4f}, Alpha: {alpha:.4f}")

# Prediction function
def strong_classifier(X):
    final_pred = np.zeros(X.shape[0])
    for alpha, (feature, threshold, polarity) in zip(alphas, stumps):
        pred = np.ones(X.shape[0])
        pred[polarity * X[:, feature] < polarity * threshold] = -1
        final_pred += alpha * pred
    return np.sign(final_pred), final_pred  # Return raw scores for ROC

# Predict
y_pred, y_scores = strong_classifier(X)

# Metrics
accuracy = accuracy_score(y, y_pred)
precision = precision_score(y, y_pred)
recall = recall_score(y, y_pred)
f1 = f1_score(y, y_pred)
roc_auc = roc_auc_score(y, y_scores)

# Print metrics
print(f"\nAccuracy: {accuracy * 100:.2f}%")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")

# Plot ROC Curve
fpr, tpr, _ = roc_curve(y, y_scores)
plt.figure(figsize=(6, 5))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for AdaBoost (from scratch)")
```

```
plt.legend(loc="lower right")
plt.grid(True)
plt.show()

# Optional: Plot Decision Thresholds per stump
plt.figure(figsize=(10, 4))
for i, (feature, threshold, polarity) in enumerate(stumps):
    plt.bar(i, threshold, label=f"{features[feature]} (pol {polarity})")
plt.ylabel("Threshold")
plt.xticks(range(T), [f"Stump {i+1}" for i in range(T)])
plt.title("Decision Thresholds of Stumps")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Round 1 | Feature: MajorAxisLength, Threshold: 422.2791, Polarity: -1, Error: 0.1356, Alpha: 0.9264
Round 2 | Feature: Perimeter, Threshold: 1006.3750, Polarity: -1, Error: 0.3602, Alpha: 0.2872
Round 3 | Feature: Perimeter, Threshold: 1259.4510, Polarity: -1, Error: 0.3894, Alpha: 0.2250
Round 4 | Feature: MajorAxisLength, Threshold: 423.8444, Polarity: 1, Error: 0.3882, Alpha: 0.2274
Round 5 | Feature: Perimeter, Threshold: 912.2590, Polarity: -1, Error: 0.4009, Alpha: 0.2008

Accuracy: 86.56%
Precision: 0.8364
Recall: 0.9089
F1 Score: 0.8711
ROC-AUC: 0.9216
```

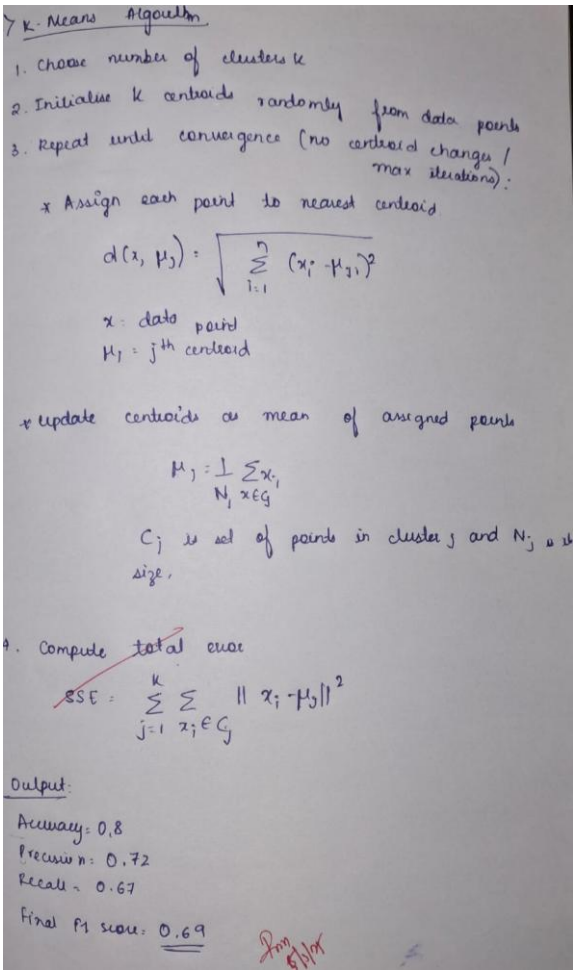




Program 10

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

Screenshot



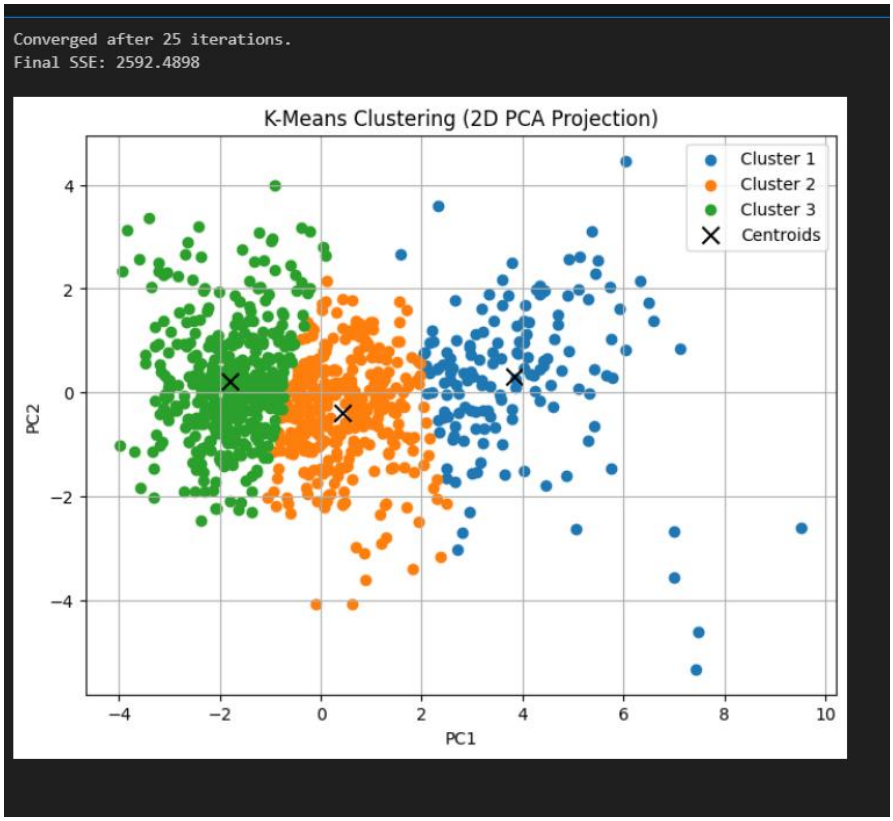
Code:

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

# Load dataset
df = pd.read_excel("/content/sample_data/Raisin_Dataset.xlsx")
features = ['Area', 'MajorAxisLength', 'MinorAxisLength', 'Eccentricity',
            'ConvexArea', 'Extent', 'Perimeter']
X = df[features].values

# Standardize features (optional but recommended)
X = (X - X.mean(axis=0)) / X.std(axis=0)
```

```python
# K-Means Parameters
k = 3
n_samples, n_features = X.shape

# Step 1: Randomly initialize k centroids
np.random.seed(42)
centroids = X[np.random.choice(n_samples, k, replace=False)]

def euclidean(a, b):
    return np.sqrt(np.sum((a - b)**2))

def assign_clusters(X, centroids):
    clusters = []
    for x in X:
        distances = [euclidean(x, centroid) for centroid in centroids]
        cluster = np.argmin(distances)
        clusters.append(cluster)
    return np.array(clusters)

def update_centroids(X, clusters, k):
    new_centroids = []
    for i in range(k):
        cluster_points = X[clusters == i]
        new_centroids.append(np.mean(cluster_points, axis=0))
    return np.array(new_centroids)

# K-Means Loop
max_iters = 100
for iteration in range(max_iters):
    clusters = assign_clusters(X, centroids)
    new_centroids = update_centroids(X, clusters, k)

    # Convergence check
    if np.allclose(centroids, new_centroids):
        print(f"Converged after {iteration+1} iterations.")
        break
    centroids = new_centroids

# SSE (Sum of Squared Errors)
sse = 0
for i in range(k):
    points = X[clusters == i]
    sse += np.sum((points - centroids[i])**2)

print(f"Final SSE: {sse:.4f}")
```

```python
# Visualize in 2D using PCA
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_2d = pca.fit_transform(X)
centroids_2d = pca.transform(centroids)

plt.figure(figsize=(8, 6))
for i in range(k):
    plt.scatter(X_2d[clusters == i, 0], X_2d[clusters == i, 1], label=f"Cluster {i+1}")
plt.scatter(centroids_2d[:, 0], centroids_2d[:, 1], color='black', marker='x', s=100,
label="Centroids")
plt.title("K-Means Clustering (2D PCA Projection)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend()
plt.grid(True)
plt.show()
```

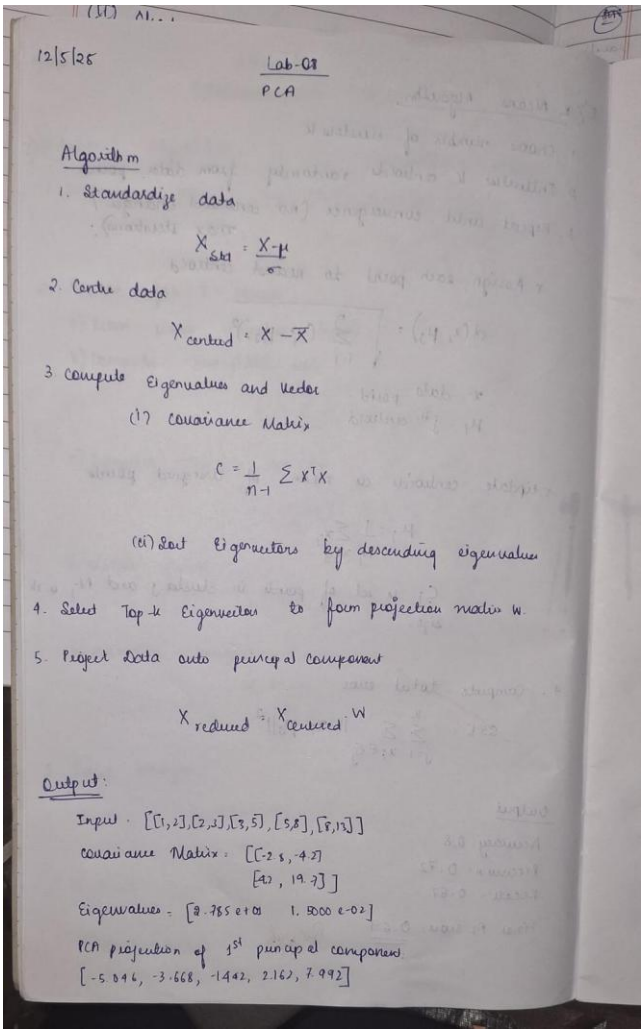
```
Converged after 25 iterations.
Final SSE: 2592.4898
```



Program 11

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

Screenshot



12/5/25

Lab-09

PCA

Algorithm

1. Standardize data

$$X_{std} = \frac{X-\mu}{\sigma}$$

2. Centre data

$$X_{centred} = X - \bar{X}$$

3. Compute Eigenvalues and Vector

(i) covariance Matrix

$$C = \frac{1}{n-1} \sum X^T X$$

(ii) Sort Eigenvectors by descending eigenvalues

4. Select Top-k Eigenvectors to form projection matrix W.

5. Project Data onto principal component

$$X_{reduced} = X_{centered} \cdot W$$

Output:

Input : [[1,2],[2,3],[3,5],[5,8],[8,13]]

covariance Matrix : [[7.5, 4.7]
[4.7, 19.7]]

Eigenvalues : [2.755e+01 1.5000e-02]

PCA projection of 1st principal component

[-5.046, -3.668, -1.442, 2.162, 7.992]

Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# New sample data
X = np.array([
    [1, 2],
    [2, 3],
    [3, 5],
    [5, 8],
    [8, 13]
])
```

```
print("=== Original Data (X) ===")
print(X)

# Step 1: Center the data
mean = np.mean(X, axis=0)
X_meaned = X - mean
print("\nMean of each feature:")
print(f"x1 mean = {mean[0]:.3f}, x2 mean = {mean[1]:.3f}")

print("\nMean-centered Data:")
print(np.round(X_meaned, 3))

# Step 2: Covariance matrix
cov_matrix = np.cov(X_meaned, rowvar=False)
print("\nCovariance Matrix:")
print(np.round(cov_matrix, 3))

# Step 3: Eigenvalues and Eigenvectors
eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)

# Step 4: Sort eigenvalues and eigenvectors descending
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]

print("\nEigenvalues (sorted):")
print(np.round(eigenvalues, 3))

print("\nEigenvectors (columns correspond to eigenvalues):")
print(np.round(eigenvectors, 3))

# Step 5: Select top k eigenvectors (k=1)
k = 1
W = eigenvectors[:, :k]

# Step 6: Project data
X_reduced_manual = X_meaned @ W

print("\nManual PCA Projection onto first principal component:")
print(np.round(X_reduced_manual.flatten(), 3))

# Using sklearn PCA for comparison
pca = PCA(n_components=1)
X_reduced_sklearn = pca.fit_transform(X)
```

```
print("\nScikit-learn PCA projection:")
print(np.round(X_reduced_sklearn.flatten(), 3))

print("\nExplained Variance Ratio (sklearn PCA):")
print(np.round(pca.explained_variance_ratio_, 3))

# Plotting
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.scatter(X[:, 0], X[:, 1], color='navy', s=60, label='Original Data')
plt.title("Original 2D Data")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid(True)
plt.legend()

plt.subplot(1, 2, 2)
plt.scatter(X_reduced_manual, np.zeros_like(X_reduced_manual), color='crimson', s=60,
label='Manual PCA Projection')
plt.title("1D PCA Projection (Manual)")
plt.xlabel("Principal Component 1")
plt.yticks([])
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
```

```
=== Original Data (X) ===
[[ 1  2]
 [ 2  3]
 [ 3  5]
 [ 5  8]
 [ 8 13]]

Mean of each feature:
x1 mean = 3.800, x2 mean = 6.200

Mean-centered Data:
[[-2.8 -4.2]
 [-1.8 -3.2]
 [-0.8 -1.2]
 [ 1.2  1.8]
 [ 4.2  6.8]]

Covariance Matrix:
[[ 7.7 12.3]
 [12.3 19.7]]

Eigenvalues (sorted):
[2.7385e+01 1.5000e-02]

Eigenvectors (columns correspond to eigenvalues):
...
[-5.046 -3.668 -1.442  2.162  7.992]

Explained Variance Ratio (sklearn PCA):
[0.999]
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...