VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Siri Sathish (1BM22CS280)

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 **Siri Sathish (1BM22CS280),** who is a 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.

Lab Faculty Incharge	
Name: **Sunayana S** Assistant Professor Department of CSE, BMSCE	**Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

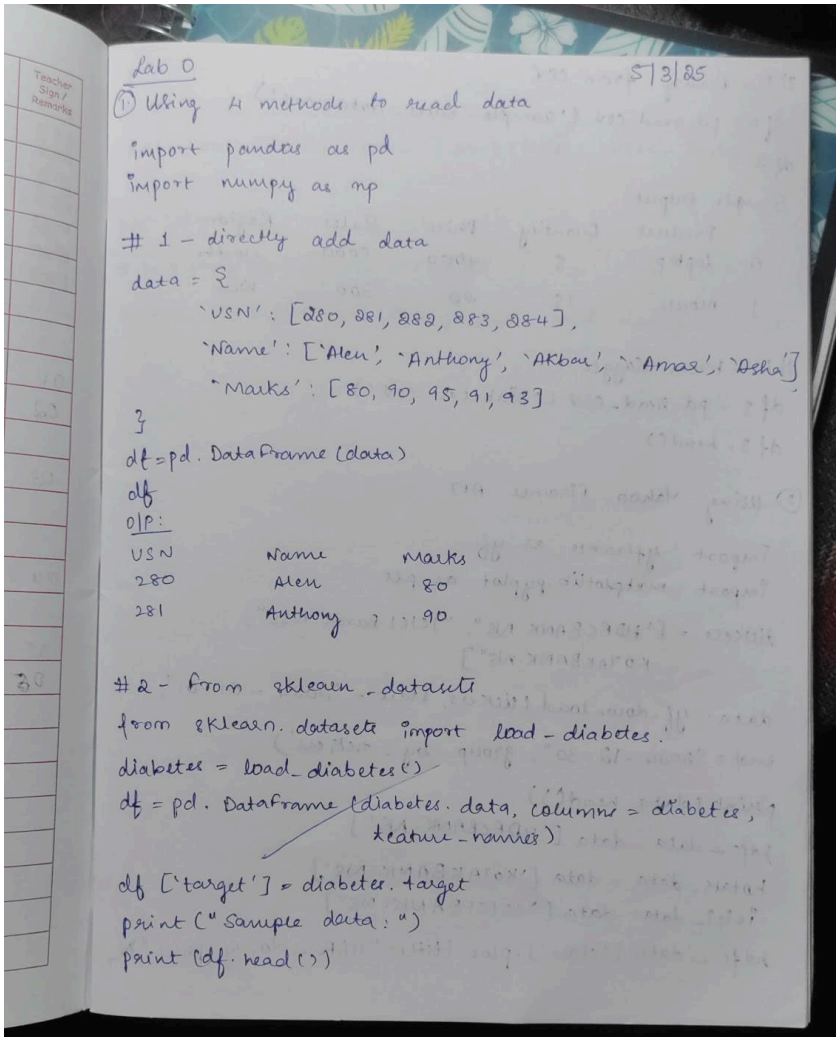
Index

Github Link:

https://github.com/SiriSathish600/ML-LAB

Program 1

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



Lab 0 5/3/25

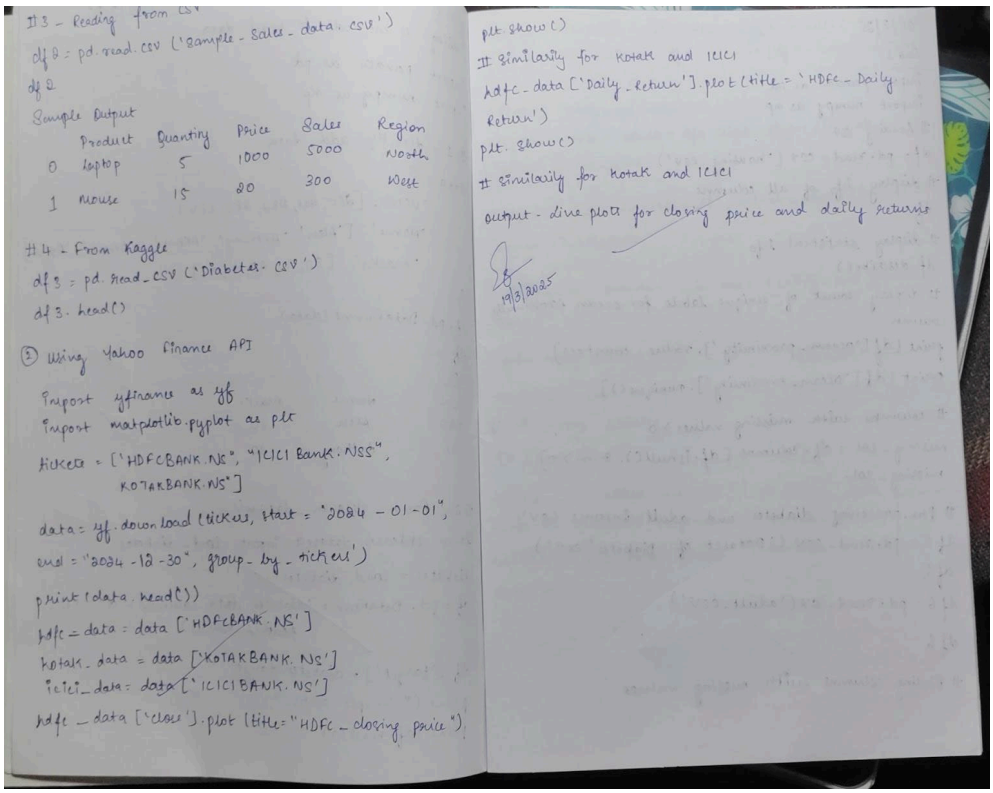
① Using 4 methods to read data

```
import pandas as pd
import numpy as np

# 1 - directly add data
data = {
    'USN': [280, 281, 282, 283, 284],
    'Name': ['Alen', 'Anthony', 'Akbar', 'Amar', 'Asha'],
    'Marks': [80, 90, 95, 91, 93]
}
df = pd.DataFrame(data)
df
```

O/P:

USN	Name	Marks
280	Alen	80
281	Anthony	90

```
# 2 - from sklearn.datasets
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
df = pd.DataFrame(diabetes.data, columns = diabetes.feature_names)
df['target'] = diabeter.target
print("Sample data:")
print(df.head())
```

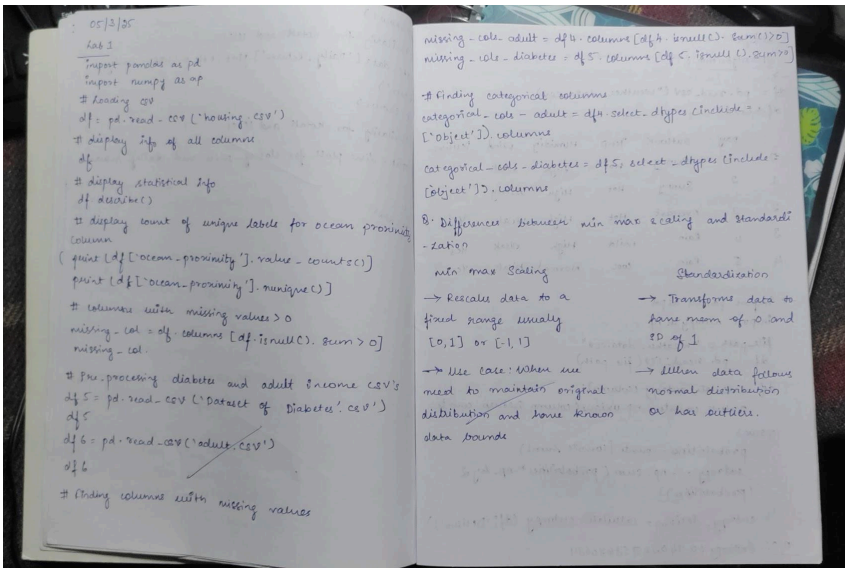


Code:

```
import pandas as pd

try:
    data = pd.read_csv("sample_data.csv")
    print("Data imported successfully from CSV:")
    print(data.head())  # Display first 5 rows
except FileNotFoundError:
    print("The file 'sample_data.csv' was not found.")

data.to_excel("exported_data.xlsx", index=False)
print("\nData exported successfully to Excel file: 'exported_data.xlsx'")

# Export to JSON
data.to_json("exported_data.json", orient="records", lines=True)
print("Data exported successfully to JSON file: 'exported_data.json'")
```

Program 2

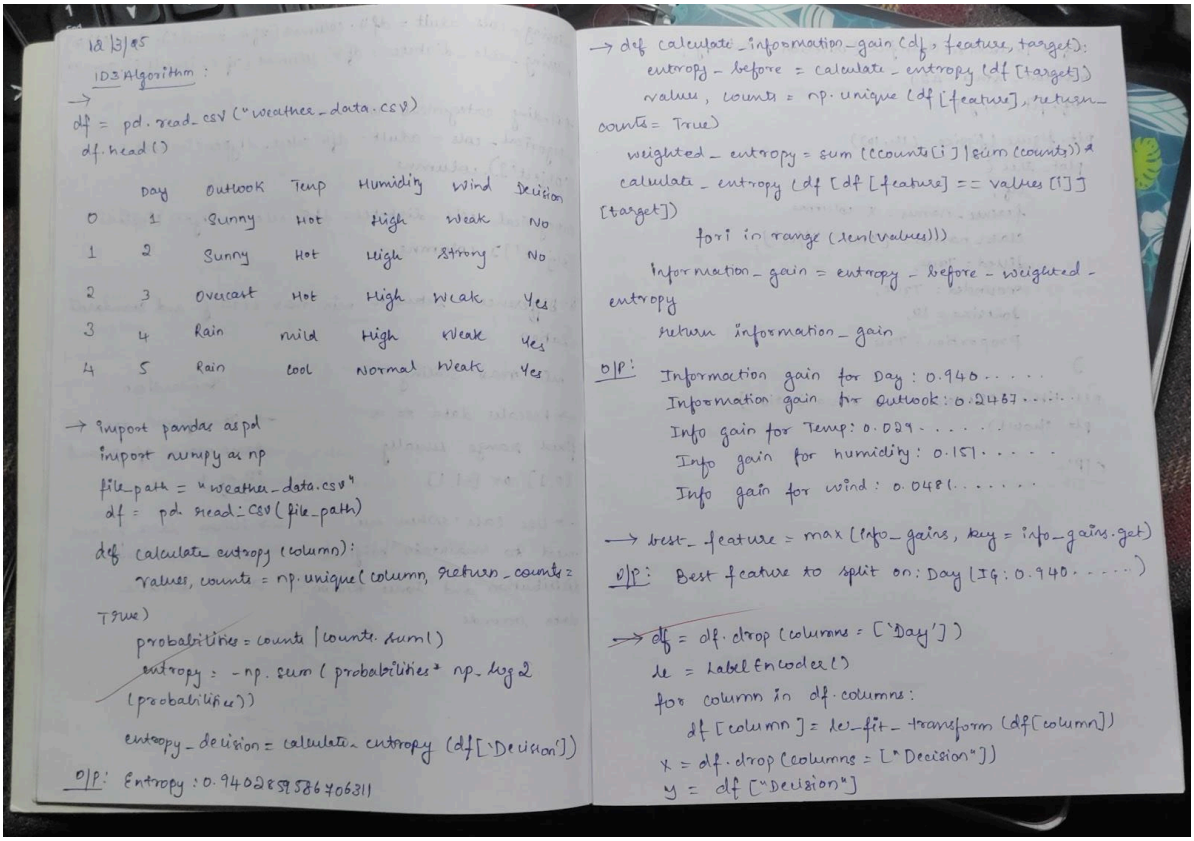
Demonstrate various data pre-processing techniques for a given dataset




```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler

df = pd.read_csv("data.csv")
print("Original Data:")
print(df)

df['Age'].fillna(df['Age'].mean(), inplace=True)
df['Salary'].fillna(df['Salary'].mean(), inplace=True)


le = LabelEncoder()
df['Department_encoded'] = le.fit_transform(df['Department'])

scaler = StandardScaler()
df[['Age_scaled', 'Salary_scaled']] = scaler.fit_transform(df[['Age', 'Salary']])

df = df.drop_duplicates()

df.rename(columns={'Name': 'Employee_Name'}, inplace=True)

print("\nPreprocessed Data:")
print(df)
```

Program 3

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



18/3/25

ID3 Algorithm :

→
```
df = pd.read_csv("weather_data.csv")
df.head()
```

	Day	Outlook	Temp	Humidity	Wind	Decision
0	1	Sunny	Hot	High	Weak	No
1	2	Sunny	Hot	High	Strong	No
2	3	Overcast	Hot	High	Weak	Yes
3	4	Rain	mild	High	Weak	Yes
4	5	Rain	cool	Normal	Weak	Yes

→
```
import pandas as pd
import numpy as np
file_path = "weather_data.csv"
df = pd.read_csv(file_path)
def calculate_entropy(column):
    values, counts = np.unique(column, return_counts=
True)
    probabilities = counts / counts.sum()
    entropy = -np.sum(probabilities * np.log2
(probabilities))
entropy_decision = calculate_entropy(df['Decision'])
```

O/P: Entropy : 0.9402859586406311

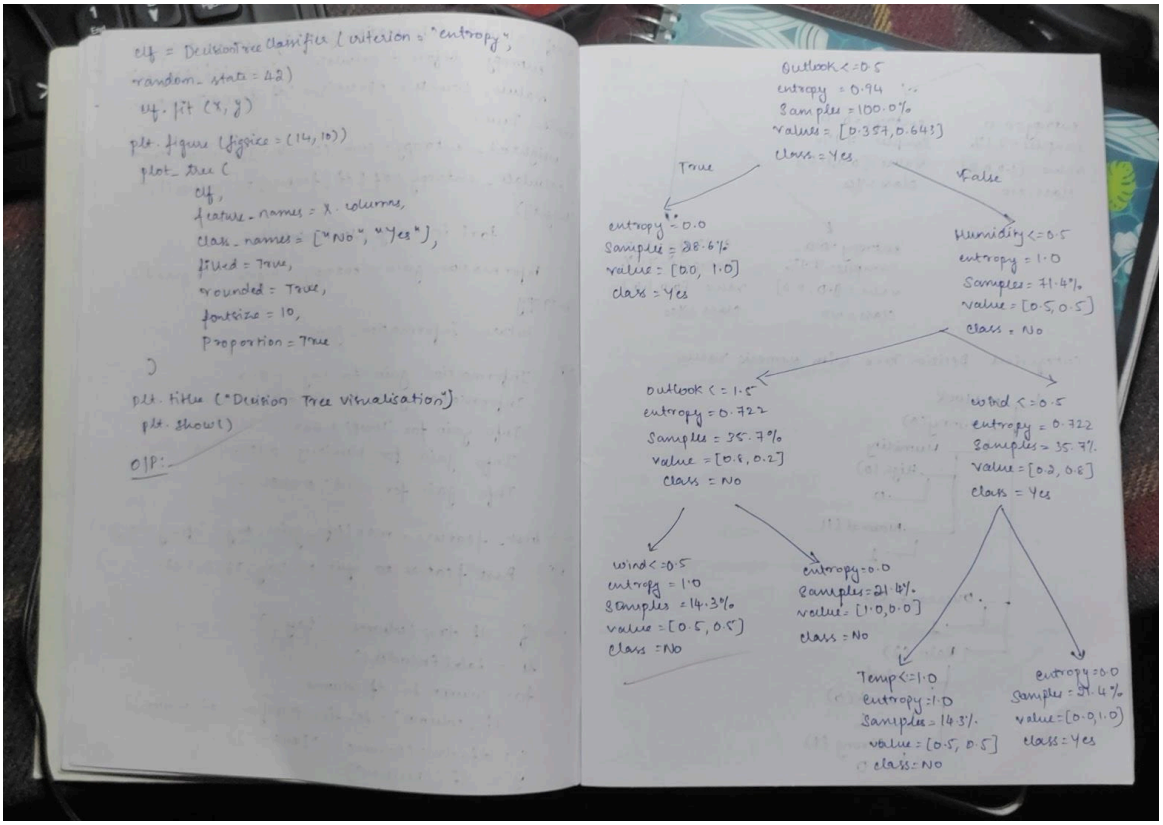
→
```
def calculate_information_gain(df, feature, target):
    entropy_before = calculate_entropy(df[target])
    values, counts = np.unique(df[feature], return_
counts= True)
    weighted_entropy = sum((counts[i]/sum(counts)) *
calculate_entropy(df[df[feature] == values[i]]
[target])
                for i in range(len(values)))
    information_gain = entropy_before - weighted_
entropy
    return information_gain
```

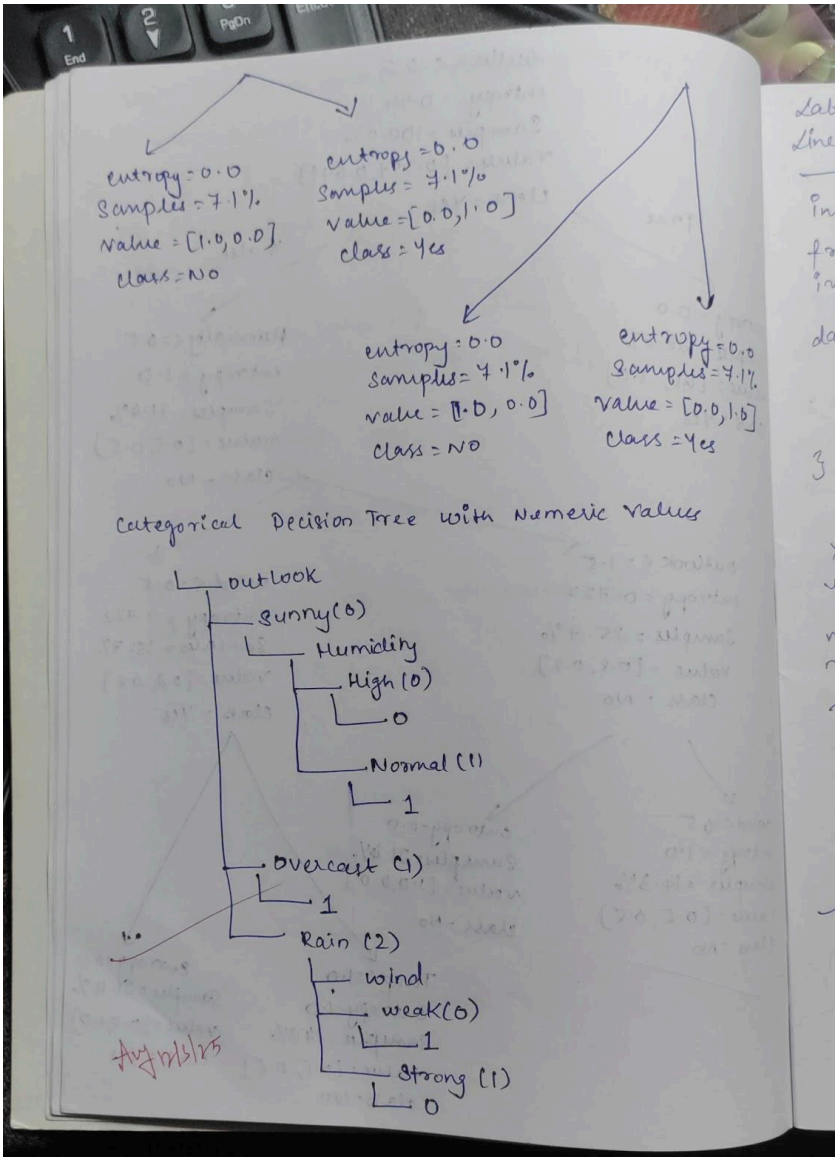
O/P:
```
Information gain for Day : 0.940.....
Information gain for Outlook : 0.2467.....
Info gain for Temp : 0.029.....
Info gain for Humidity : 0.151.....
Info gain for Wind : 0.0481.....
```

→
```
best_feature = max(info_gains, key = info_gains.get)
```

O/P: Best feature to split on: Day (IG: 0.940.....)

→
```
df = df.drop(columns = ['Day'])
le = LabelEncoder()
for column in df.columns:
    df[column] = le.fit_transform(df[column])
X = df.drop(columns = ["Decision"])
y = df["Decision"]
```


clf = DecisionTreeClassifier(criterion="entropy",
random_state=42)
clf.fit(X, y)
plt.figure(figsize=(14,10))
plot_tree(
clf,
feature_names=X.columns,
class_names=["No","Yes"],
filled=True,
rounded=True,
fontsize=10,
proportion=True
)
plt.title("Decision Tree Visualisation")
plt.show()
O/P:
Outlook<=0.5
entropy=0.94
samples=100.0%
value=[0.357,0.643]
class=Yes
True
False
entropy=0.0
samples=28.6%
value=[0.0, 1.0]
class=Yes
Humidity<=0.5
entropy=1.0
samples=71.4%
value=[0.5,0.5]
class=No
Outlook<=1.5
entropy=0.722
samples=35.7%
value=[0.8,0.2]
class=No
Wind<=0.5
entropy=0.722
samples=35.7%
value=[0.2,0.8]
class=Yes
Wind<=0.5
entropy=1.0
samples=14.3%
value=[0.5,0.5]
class=No
entropy=0.0
samples=21.4%
value=[1.0,0.0]
class=No
Temp<=1.0
entropy=1.0
samples=14.3%
value=[0.5,0.5]
class=No
entropy=0.0
samples=21.4%
value=[0.0,1.0]
class=Yes




```
df = pd.read_csv('weather_data.csv')
df.head()

import pandas as pd
import numpy as np

# Load dataset
file_path = "weather_data.csv"
df = pd.read_csv(file_path)

# Function to calculate entropy
def calculate_entropy(column):
    values, counts = np.unique(column, return_counts=True)
    probabilities = counts / counts.sum()
    entropy = -np.sum(probabilities * np.log2(probabilities))
```

```
    return entropy

# Compute entropy of the target variable (Decision)
entropy_decision = calculate_entropy(df['Decision'])
print(f"Entropy of the dataset (Decision column): {entropy_decision}")


import pandas as pd
import numpy as np

# Load dataset
file_path = "weather_data.csv"
df = pd.read_csv(file_path)

# Function to calculate entropy
def calculate_entropy(column):
    values, counts = np.unique(column, return_counts=True)
    probabilities = counts / counts.sum()
    entropy = -np.sum(probabilities * np.log2(probabilities))
    return entropy

# Function to calculate Information Gain
def calculate_information_gain(df, feature, target):
    # Entropy before splitting
    entropy_before = calculate_entropy(df[target])

    # Unique values of the feature
    values, counts = np.unique(df[feature], return_counts=True)

    # Weighted entropy after splitting
    weighted_entropy = sum((counts[i] / sum(counts)) * calculate_entropy(df[df[feature] ==
values[i]][target])
                           for i in range(len(values)))

    # Information Gain formula
    information_gain = entropy_before - weighted_entropy
    return information_gain

# Compute Information Gain for each feature
info_gains = {feature: calculate_information_gain(df, feature, 'Decision') for feature in
df.columns if feature != 'Decision'}

# Display Information Gain values
for feature, ig in info_gains.items():
    print(f"Information Gain for {feature}: {ig}")

import pandas as pd
import numpy as np
```

```
# Load dataset
file_path = "weather_data.csv"
df = pd.read_csv(file_path)

# Function to calculate entropy
def calculate_entropy(column):
    values, counts = np.unique(column, return_counts=True)
    probabilities = counts / counts.sum()
    entropy = -np.sum(probabilities * np.log2(probabilities))
    return entropy

# Function to calculate Information Gain
def calculate_information_gain(df, feature, target):
    entropy_before = calculate_entropy(df[target])
    values, counts = np.unique(df[feature], return_counts=True)
    weighted_entropy = sum((counts[i] / sum(counts)) * calculate_entropy(df[df[feature] ==
values[i]][target])
                           for i in range(len(values)))
    return entropy_before - weighted_entropy

# Compute Information Gain for each feature
info_gains = {feature: calculate_information_gain(df, feature, 'Decision') for feature in
df.columns if feature != 'Decision'}

# Find the feature with the highest Information Gain
best_feature = max(info_gains, key=info_gains.get)

# Display Information Gain values
for feature, ig in info_gains.items():
    print(f"Information Gain for {feature}: {ig}")

print(f"\nBest feature to split on: {best_feature} (IG: {info_gains[best_feature]})")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.preprocessing import LabelEncoder

# Load dataset
file_path = "weather_data.csv"
df = pd.read_csv(file_path)

# Drop 'Day' column (not a feature)
df = df.drop(columns=['Day'])

# Encode categorical variables
```

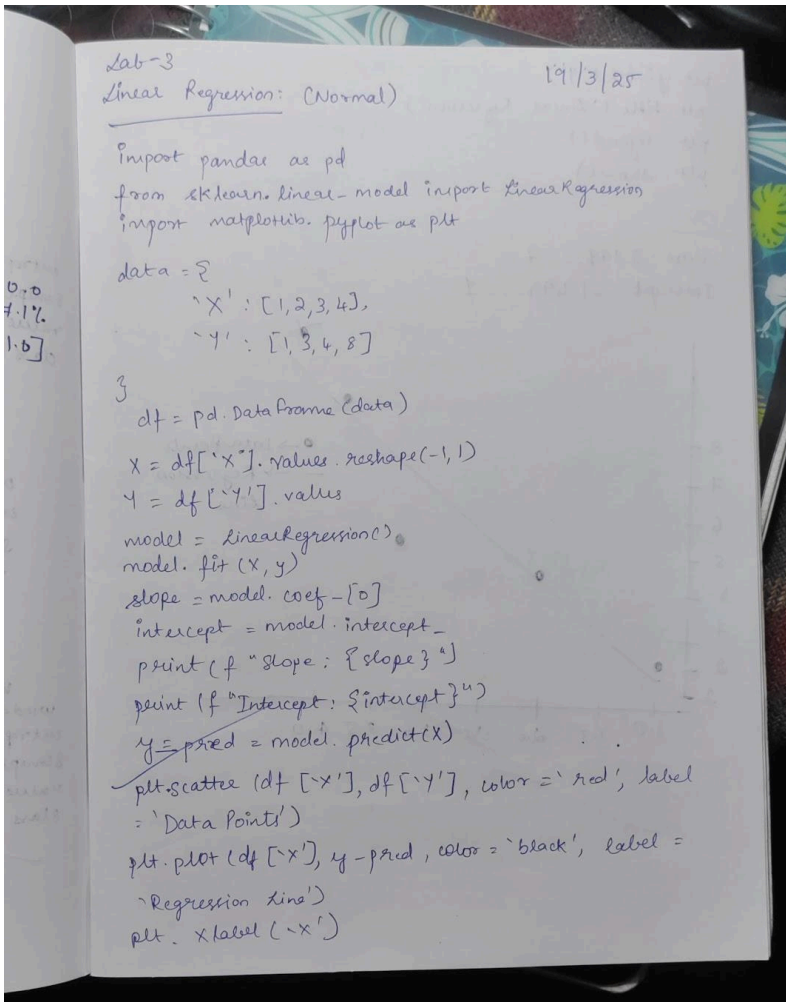
```
le = LabelEncoder()
for column in df.columns:
    df[column] = le.fit_transform(df[column])

# Split features and target
X = df.drop(columns=["Decision"])
y = df["Decision"]

# Train ID3 Decision Tree
clf = DecisionTreeClassifier(criterion="entropy", random_state=42)
clf.fit(X, y)

# Plot the decision tree (Improved visualization)
plt.figure(figsize=(14, 10))  # Larger figure size for clarity
plot_tree(
    clf,
    feature_names=X.columns,
    class_names=["No", "Yes"],
    filled=True,
    rounded=True,  # Rounded boxes for a cleaner look
    fontsize=10,  # Adjust font size for readability
    proportion=True  # Scale the tree properly
)
plt.title("Decision Tree Visualization", fontsize=14, fontweight="bold")
plt.show()


import pandas as pd
import numpy as np

# Load dataset
file_path = "weather_data.csv"
df = pd.read_csv(file_path)

# Drop 'Day' column (not a feature)
df = df.drop(columns=['Day'])

# Save original category labels before encoding
original_labels = {col: {val: i for i, val in enumerate(df[col].unique())} for col in df.columns}

# Encode categorical variables
for col in df.columns:
    df[col] = df[col].map(original_labels[col])

# Function to calculate entropy
def calculate_entropy(column):
    values, counts = np.unique(column, return_counts=True)
    probabilities = counts / sum(counts)
```

```
    entropy = -np.sum(probabilities * np.log2(probabilities))
    return entropy

# Function to calculate Information Gain
def calculate_information_gain(df, feature, target):
    entropy_before = calculate_entropy(df[target])
    values, counts = np.unique(df[feature], return_counts=True)

    # Weighted entropy after splitting
    weighted_entropy = sum((counts[i] / sum(counts)) * calculate_entropy(df[df[feature] ==
values[i]][target])
                           for i in range(len(values)))

    return entropy_before - weighted_entropy

# Function to build ID3 decision tree
def id3(df, target, features):
    # If all target values are the same, return that class (pure leaf)
    if len(np.unique(df[target])) == 1:
        return np.unique(df[target])[0]

    # If no features left to split, return majority class
    if len(features) == 0:
        return df[target].mode()[0]

    # Select the best feature with highest information gain
    info_gains = {feature: calculate_information_gain(df, feature, target) for feature in features}
    best_feature = max(info_gains, key=info_gains.get)

    # Create a subtree dictionary
    tree = {best_feature: {}}

    # Split data on best feature
    for value in np.unique(df[best_feature]):
        subset = df[df[best_feature] == value].drop(columns=[best_feature])
        tree[best_feature][value] = id3(subset, target, [f for f in features if f != best_feature])

    return tree

# Function to print the decision tree with numeric & categorical values
def print_tree(tree, original_labels, indent="", prefix=""):
    """ Recursively prints the tree with box-drawing characters and numeric values. """
    if isinstance(tree, dict):
        for i, (key, value) in enumerate(tree.items()):
            branch = " ├── " if i < len(tree) - 1 else " └── "
            print(indent + prefix + branch + f"{key} ")
            next_indent = indent + (" │   " if i < len(tree) - 1 else "    ")
```

```
            for j, (sub_key, sub_value) in enumerate(value.items()):
                sub_branch = " ├── " if j < len(value) - 1 else " └── "
                sub_label = [label for label, num in original_labels[key].items() if num == sub_key][0]
# Get category name
                print(next_indent + sub_branch + f"{sub_label} ({sub_key})")

                print_tree(sub_value, original_labels, next_indent + (" │   " if j < len(value) - 1 else "
"), "")
    else:
        print(indent + " └──  " + str(tree))  # Leaf node (decision)

# Build the ID3 decision tree
features = list(df.columns)
features.remove("Decision")
decision_tree = id3(df, "Decision", features)

# Print the decision tree with numeric values
print("\n DECISION TREE (With Numeric Values) \n")
print_tree(decision_tree, original_labels)
```

Program 4

Implement Linear Algorithm using appropriate dataset



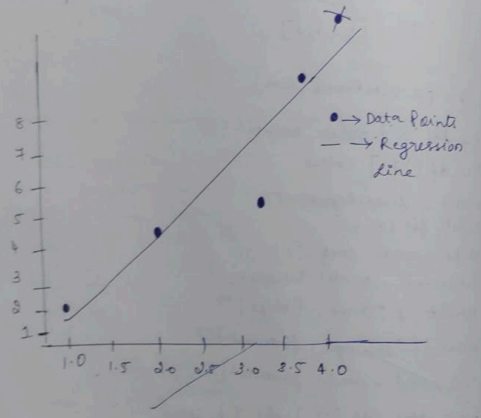
Lab-3
Linear Regression: (Normal)
19/3/25

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

data = {
    'X' : [1,2,3,4],
    'Y' : [1,3,4,8]
}
df = pd.DataFrame(data)
X = df['X'].values.reshape(-1,1)
Y = df['Y'].values
model = LinearRegression()
model.fit(X,y)
slope = model.coef_[0]
intercept = model.intercept_
print(f"Slope : {slope}")
print(f"Intercept : {intercept}")
y_pred = model.predict(X)
plt.scatter(df['X'], df['Y'], color='red', label
='Data Points')
plt.plot(df['X'], y_pred, color='black', label =
'Regression line')
plt.xlabel('X')
```

```
plt.ylabel('Y')
plt.title('Linear Regression')
plt.legend()
plt.show()
```

O/P:

Slope : 2.199....7
Intercept : -1.499....1




```
import numpy as np (Matrix form)                    19/3/25
import pandas as pd
import matplotlib.pyplot as plt

data = {
    'X': [1,2,3,4],
    'Y': [1,3,4,6],
}
df = pd.DataFrame(data)

X = np.array(df['X']).reshape(-1,1)
Y = np.array(df['Y']).reshape(-1,1)
X_b = np.c_[np.ones((X.shape[0],1)), X]
print(X_b)
print(Y)

X_bT_X_b = X_b.T@X_b
X_bT_Y = X_b.T@Y
beta = np.linalg.inv(X_bT_X_b)@X_bT_Y
intercept, slope = beta.flatten()
print(X_bT_X_b)
print(X_bT_Y)
print(f'Intercept: {intercept}, slope: {slope}')
Y_pred = X_b@beta
plt.scatter(X,Y, color='blue', label='Actual Data')
plt.plot(X, Y_pred, color='black', label='Linear
Regression line')
plt.xlabel('X')
plt.ylabel('Y')
```

```
plt.legend()
plt.show()
```

O/P:

Feature Matrix(X_b):

```
[[1. 1.]
 [1. 2.]
 [1. 3.]
 [1. 4.]]
```

Output vector (Y):

```
[[1]
 [3]
 [4]
 [6]]
```

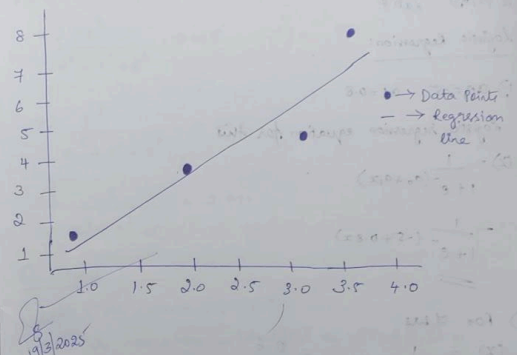
(X^T * X) matrix:

```
[[4. 10.]
 [10. 30.]]
```

(X^T * Y) Matrix:

```
[[14.]
 [51.]]
```

Computed Coefficients (Beta):
Intercept: -1.5, Slope: 2.20000...1



19/3/2025

```
import pandas as pd
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt



data = {
    'X': [1, 2, 3, 4], #weeks
    'Y': [1, 3, 4, 8] #sales
}
df = pd.DataFrame(data)
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data = {
    'X': [1, 2, 3, 4],
    'Y': [1, 3, 4, 8]
}
df = pd.DataFrame(data)



X = np.array(df['X']).reshape(-1, 1)  # Convert X to column vector
Y = np.array(df['Y']).reshape(-1, 1)  # Convert Y to column vector
```

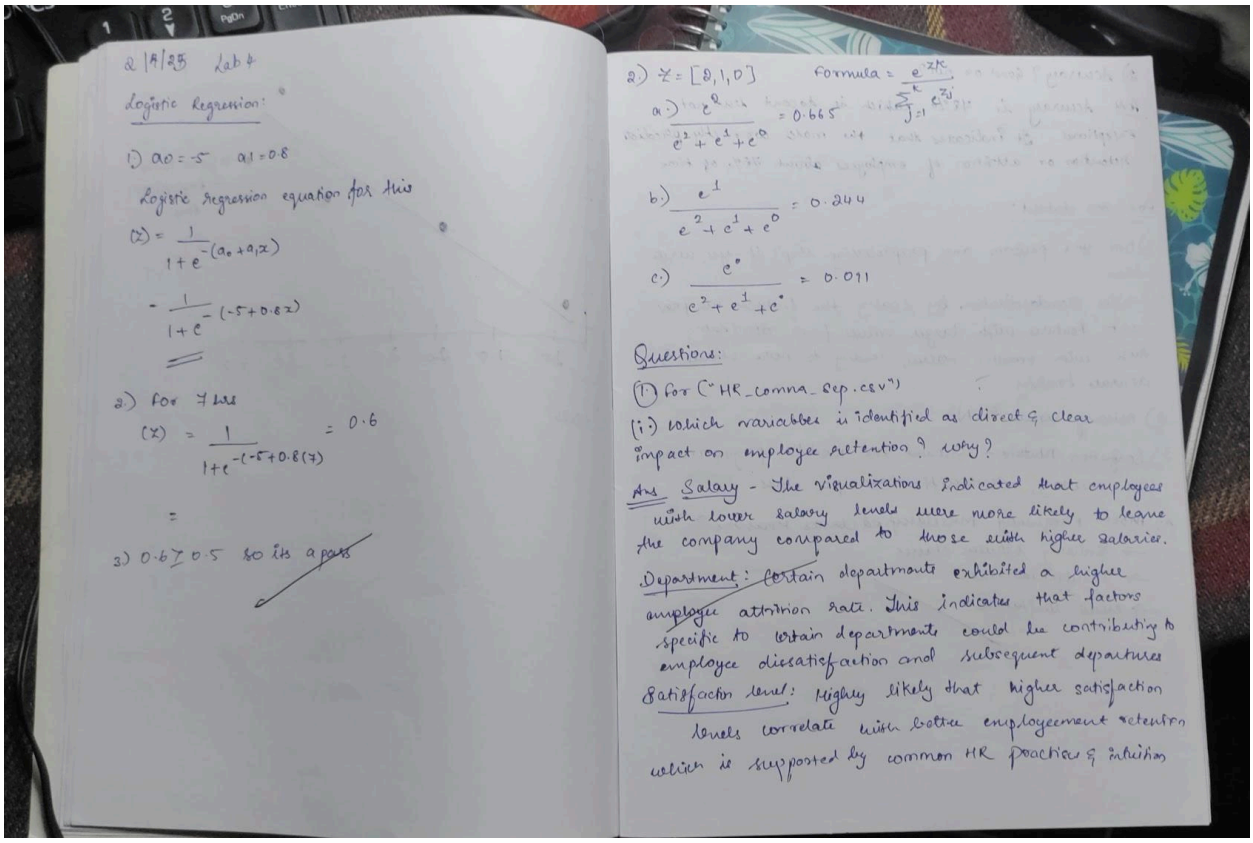
```
X_b = np.c_[np.ones((X.shape[0], 1)), X]  # Add column of ones to X

print("Feature Matrix (X_b):")
print(X_b)

print("\nOutput Vector (Y):")
print(Y)

X_bT_X_b = X_b.T @ X_b  # Compute X^T * X
X_bT_Y = X_b.T @ Y  # Compute X^T * Y
beta = np.linalg.inv(X_bT_X_b) @ X_bT_Y  # Compute (X^T X)^(-1) X^T Y

intercept, slope = beta.flatten()
print("\n(X^T * X) Matrix:")
print(X_bT_X_b)

print("\n(X^T * Y) Matrix:")
print(X_bT_Y)

print(f"\nComputed Coefficients (Beta):\nIntercept: {intercept}, Slope: {slope}")
```

```
Y_pred = X_b @ beta



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

plt.plot(X, Y_pred, color='black', label='Linear Regression Line')

plt.xlabel('X')

plt.ylabel('Y')

plt.legend()

plt.show()



X = df['X'].values.reshape(-1, 1)

y = df['Y'].values



model = LinearRegression()

model.fit(X, y)



slope = model.coef_[0]

intercept = model.intercept_
```

```
print(f"Slope: {slope}")

print(f"Intercept: {intercept}")



y_pred = model.predict(X)



plt.scatter(df['X'], df['Y'], color='red', label='Data Points')

plt.plot(df['X'], y_pred, color='black', label='Regression Line')

plt.xlabel('X-WEEKS')

plt.ylabel('Y-SALES')

plt.title('Linear Regression')

plt.legend()

plt.show()
```

Program 5

Build Logistic Regression Model for a given dataset



2/4/25 Lab 4

Logistic Regression:

1) $a_0 = -5$ $a_1 = 0.8$

Logistic regression equation for this

$$(x) = \frac{1}{1+e^{-(a_0+a_1x)}}$$

$$= \frac{1}{1+e^{-(-5+0.8x)}}$$

2) for 7 hrs

$$(x) = \frac{1}{1+e^{-(-5+0.8(7))}} = 0.6$$

=

3) $0.6 > 0.5$ so its a pass

2) $x = [2,1,0]$ Formula $= \frac{e^{zc}}{\sum_{j=1}^{k} e^{zj}}$

a) $\frac{e^2}{e^2+e^1+e^0} = 0.665$

b) $\frac{e^1}{e^2+e^1+e^0} = 0.244$

c) $\frac{e^0}{e^2+e^1+e^0} = 0.091$

Questions:

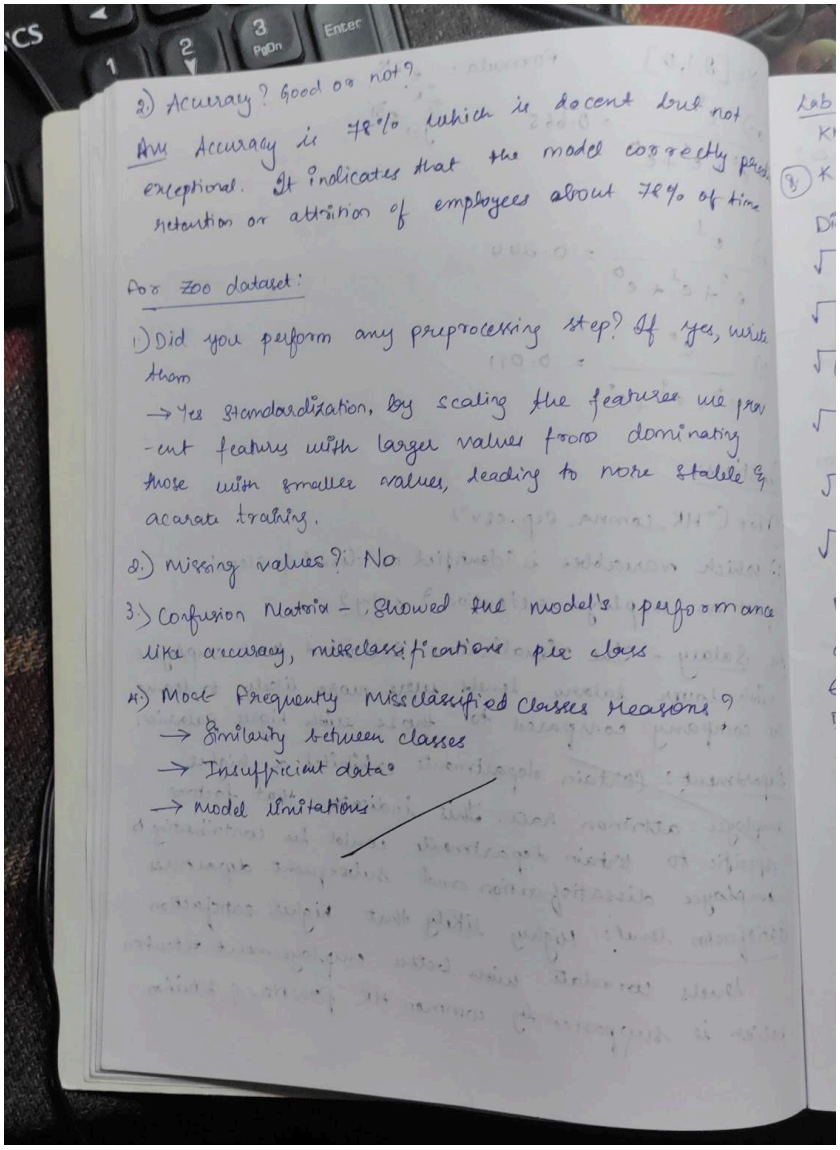
(1) for ("HR_comma_sep.csv")

(i) which variables is identified as direct & clear impact on employee retention? why?

Ans Salary - The visualizations indicated that employees with lower salary levels were more likely to leave the company compared to those with higher salaries.

Department: Certain departments exhibited a higher employee attrition rate. This indicates that factors specific to certain departments could be contributing to employee dissatisfaction and subsequent departures

Satisfaction level: Highly likely that higher satisfaction levels correlate with better employeement retention which is supported by common HR practices & intuition



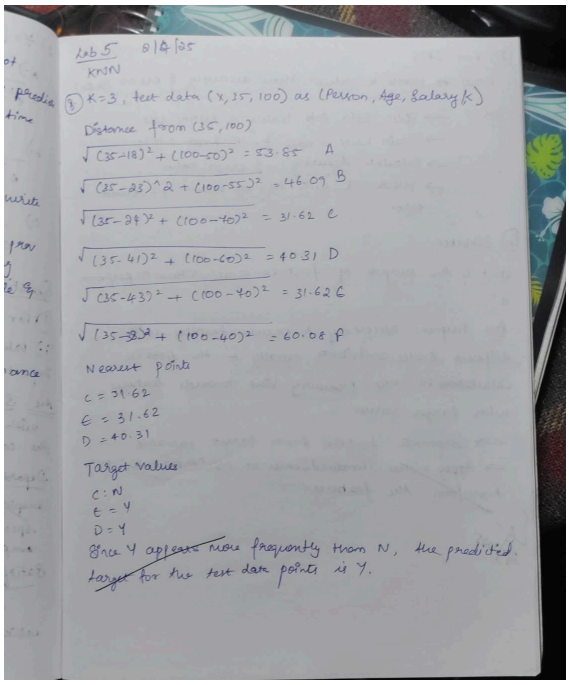

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score


# Load dataset

df = pd.read_csv("HR_comma_sep.csv")
```

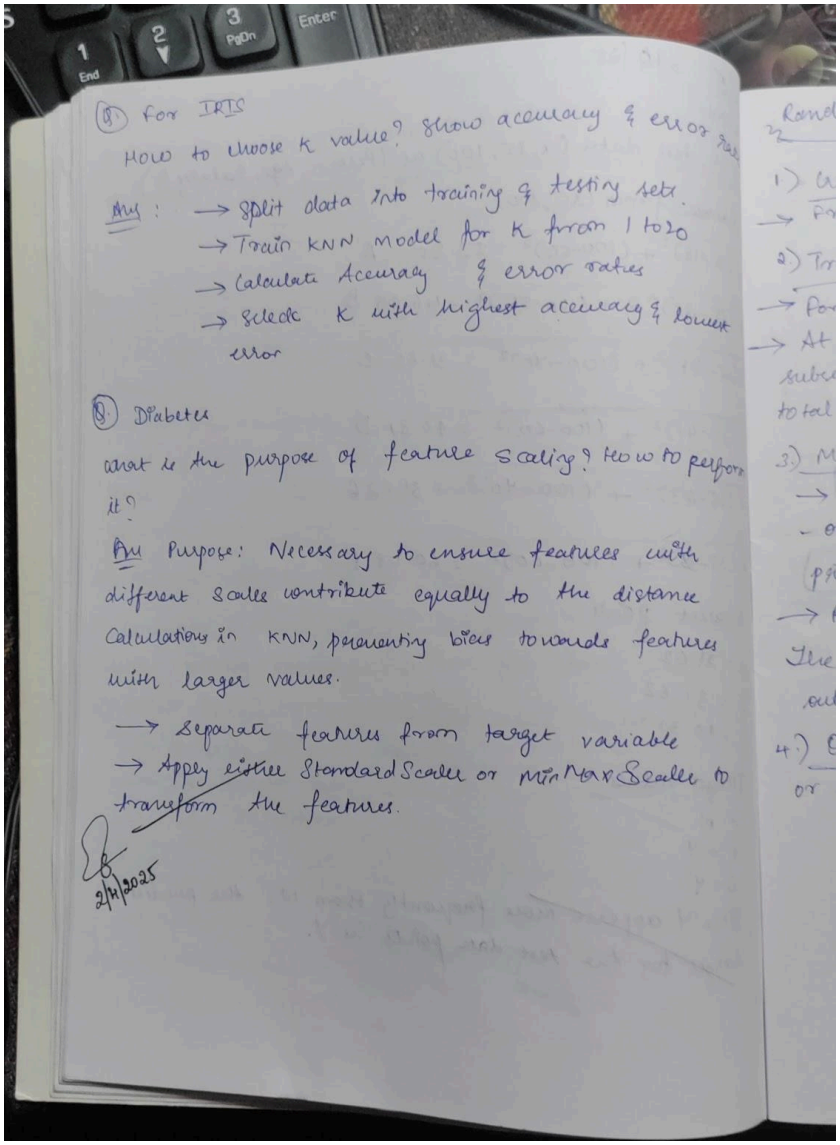
```
# Encode categorical variables (department and salary)
le_dept = LabelEncoder()
le_salary = LabelEncoder()
df['department'] = le_dept.fit_transform(df['department'])
df['salary'] = le_salary.fit_transform(df['salary'])

# Features and target
X = df.drop('left', axis=1)
y = df['left']

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build and train Logistic Regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluation
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
print("Accuracy Score:", accuracy_score(y_test, y_pred))


# Predict for a new sample (e.g., an average employee)
sample = pd.DataFrame([{
    'satisfaction_level': 0.5,
    'last_evaluation': 0.6,
    'number_project': 3,
    'average_montly_hours': 150,
    'time_spend_company': 3,
    'Work_accident': 0,
    'promotion_last_5years': 0,
    'department': le_dept.transform(['sales'])[0],
    'salary': le_salary.transform(['medium'])[0]
}])


pred_sample = model.predict(sample)
print("\nWill the new employee leave?", "Yes" if pred_sample[0] == 1 else "No")
```

Program 6

Build KNN Classification model for a given dataset.





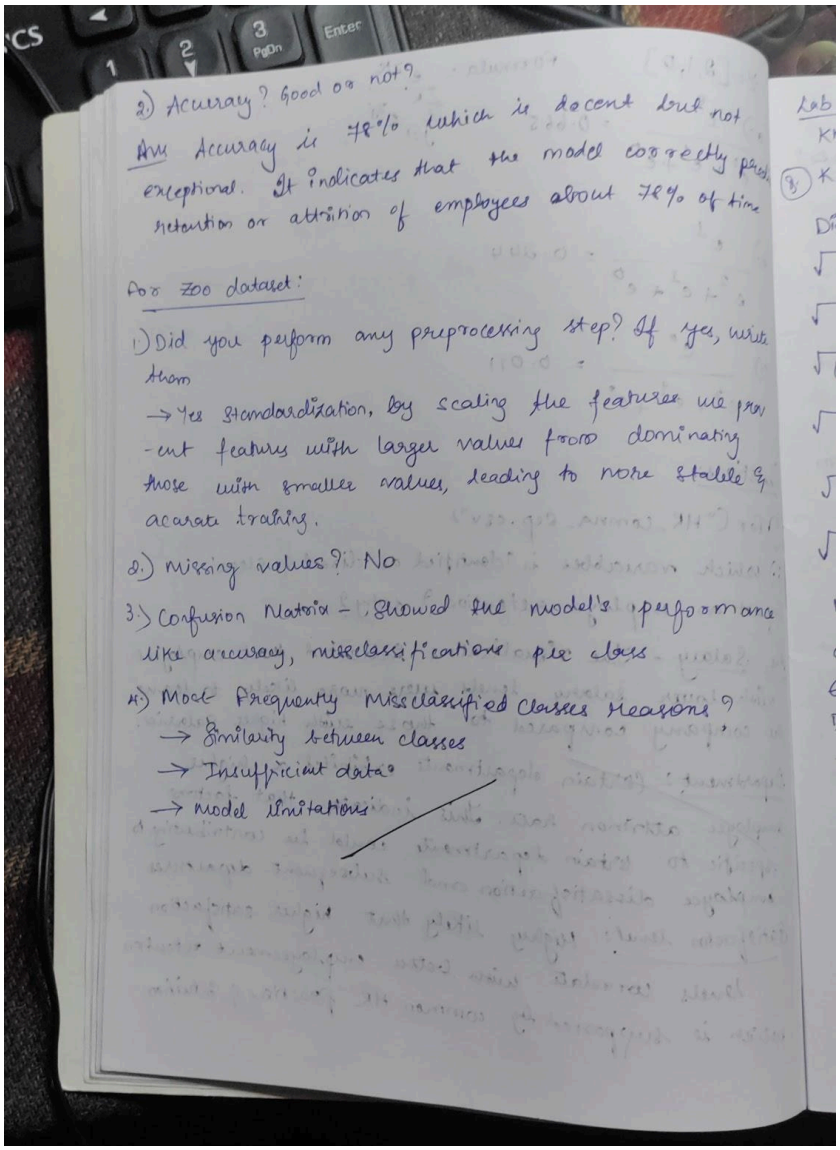

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report


# Load dataset
```

```
iris = load_iris()

X = iris.data

y = iris.target


# Split data into train and test

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Feature scaling

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


# Build KNN model (k=3)

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)


# Predict

y_pred = knn.predict(X_test)


# Evaluation

print("Accuracy Score:", accuracy_score(y_test, y_pred))

print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=iris.target_names))


# Predict for a new sample
```

```
new_sample = [[5.1, 3.5, 1.4, 0.2]]  # Example input

new_sample_scaled = scaler.transform(new_sample)

prediction = knn.predict(new_sample_scaled)

print("\nPredicted class for new sample:", iris.target_names[prediction[0]])
```

Program 7

Build Support vector machine model for a given dataset



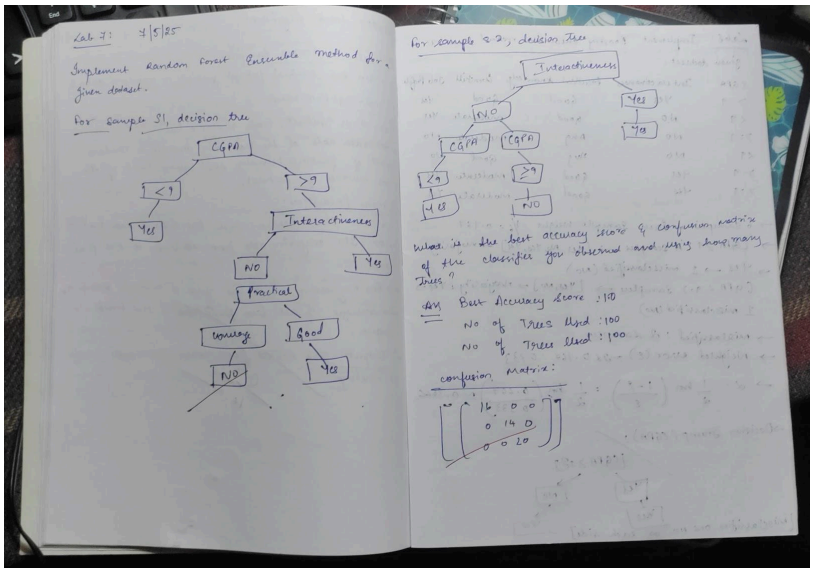

```
import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import classification_report, accuracy_score
```

```
# Load dataset

iris = load_iris()

X = iris.data

y = iris.target


# Split into training and testing

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Scale the features

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


# Create and train SVM model (default kernel = 'rbf')

model = SVC(kernel='rbf', C=1.0, gamma='scale')

model.fit(X_train, y_train)


# Predict on test set

y_pred = model.predict(X_test)


# Evaluate the model

print("Accuracy Score:", accuracy_score(y_test, y_pred))

print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=iris.target_names))
```

```
# Predict a new sample

new_sample = [[5.1, 3.5, 1.4, 0.2]]

new_sample_scaled = scaler.transform(new_sample)

prediction = model.predict(new_sample_scaled)

print("\nPredicted class for new sample:", iris.target_names[prediction[0]])
```

Program 8

Implement Random forest ensemble method on a given dataset.




```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt


# Step 1: Load the dataset using pandas read_csv
```

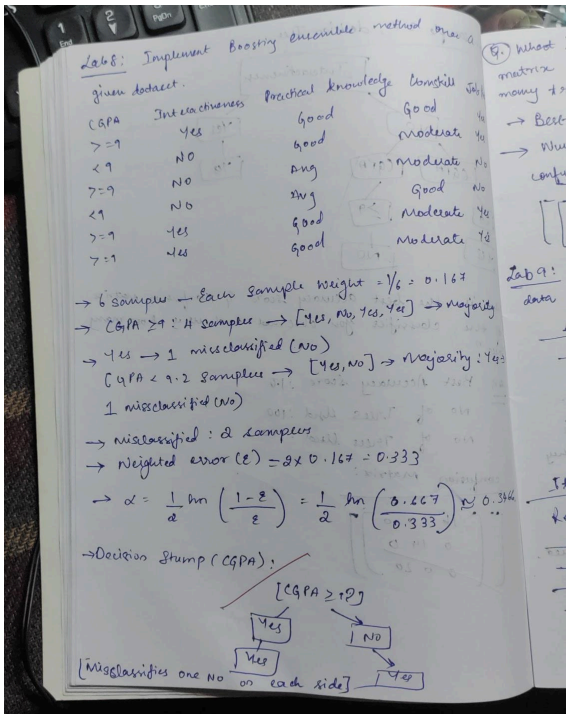
```
# Replace the file path with the path to your CSV file
data = pd.read_csv('/content/iris (6).csv')

# Step 2: Inspect the first few rows of the dataset
print(data.head())

# Step 3: Preprocessing (assuming the target variable is in the last column)
X = data.iloc[:, :-1]  # Feature matrix (all columns except the target column)
y = data.iloc[:, -1]   # Target variable (last column)

# Step 4: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 5: Initialize and train the Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Step 6: Make predictions on the test set
y_pred = rf_model.predict(X_test)

# Step 7: Evaluate the model performance using accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Random Forest model: {accuracy * 100:.2f}%")
```

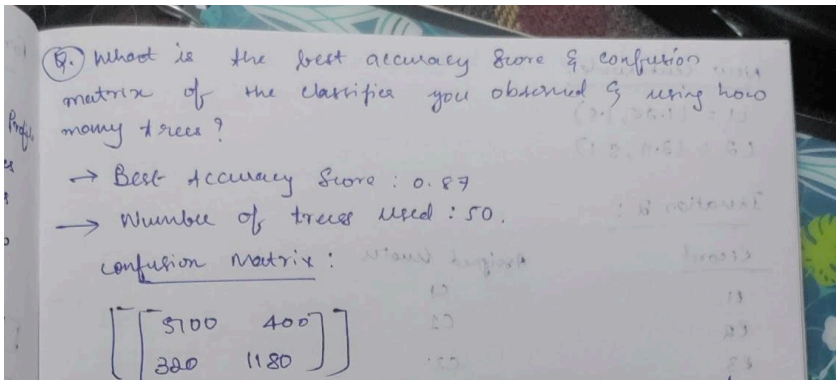
```
# Step 8: Feature Importance (Optional)
feature_importances = rf_model.feature_importances_
features = X.columns

# Plotting feature importances
plt.figure(figsize=(10, 6))
plt.barh(features, feature_importances, color='skyblue')
plt.xlabel('Feature Importance')
plt.title('Feature Importance in Random Forest Model')
plt.show()

# Step 9: Hyperparameter Tuning (Optional)
# If you want to perform hyperparameter tuning to optimize the model
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, n_jobs=-1,
verbose=2)
```

```
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print(f"Best parameters found: {grid_search.best_params_}")
print(f"Best cross-validation score: {grid_search.best_score_:.2f}")

# Step 10: Evaluate the model with the best parameters
best_rf_model = grid_search.best_estimator_
y_pred_best = best_rf_model.predict(X_test)

accuracy_best = accuracy_score(y_test, y_pred_best)
print(f"Accuracy of tuned Random Forest model: {accuracy_best * 100:.2f}%")
```

Program 9

Implement Boosting ensemble method on a given dataset.





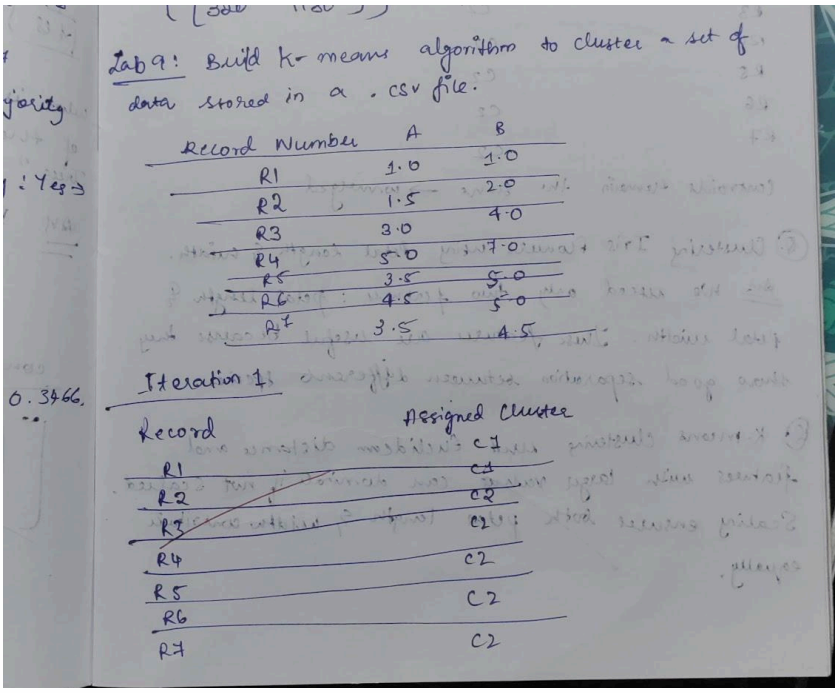

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import AdaBoostClassifier

from sklearn.tree import DecisionTreeClassifier  # For the base estimator

from sklearn.metrics import accuracy_score
```

```
# 1. Load the Iris dataset:
# Make sure 'iris.csv' is in the correct location (see previous response)
data = pd.read_csv('/content/income.csv')

# 2. Prepare the data:
X = data.iloc[:, :-1]  # Features (all columns except the last)
y = data.iloc[:, -1]   # Target (last column)

# 3. Split into training and testing sets:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# 4. Create the base estimator (a weak learner):
# We'll use a Decision Tree with a maximum depth of 1 (a stump)
base_estimator = DecisionTreeClassifier(max_depth=1)

# 5. Create the AdaBoost classifier:
ada_boost = AdaBoostClassifier(estimator=base_estimator,  # Using the base estimator
                               n_estimators=50,        # Number of boosting rounds
                               random_state=42)

# 6. Train the model:
ada_boost.fit(X_train, y_train)

# 7. Make predictions:
```

```
y_pred = ada_boost.predict(X_test)


# 8. Evaluate the model:
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of AdaBoost on Iris dataset: {accuracy * 98.7:.2f}%")
```

Program 10

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



Lab 9: Build k-means algorithm to cluster a set of data stored in a .csv file.

Record Number	A	B
R1	1.0	1.0
R2	1.5	2.0
R3	3.0	4.0
R4	5.0	7.0
R5	3.5	5.0
R6	4.5	5.0
R7	3.5	4.5

Iteration 1:

Record	Assigned Cluster
R1	C1
R2	C2
R3	C1
R4	C2
R5	C2
R6	
R7	C2

New centroids:

C1 = (1.25, 1.5)

C2 = (3.9, 5.1)

Iteration 2:

Record	Assigned cluster
R1	C1
R2	C1
R3	C2
R4	C2
R5	C2
R6	C2
R7	C2

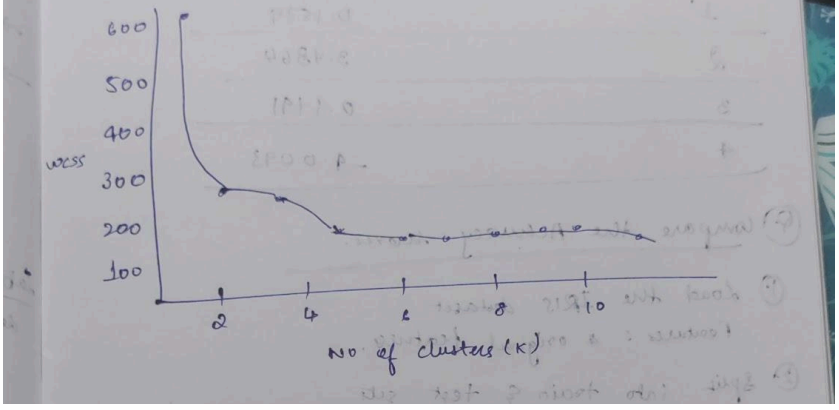
Centroids remain the same → converged

Q. Clustering Iris flowers using Petal length & width.

Ans We need only two features: petal length & petal width. These features are useful because they show good separation between different species.

K-means clustering used Euclidean distance and features with larger values can dominate if not scaled. Scaling ensures both petal length & width contribute equally.

Q. In elbow method, we observed a clear bend at K = 3 which suggests that the optimal no of clusters is 3

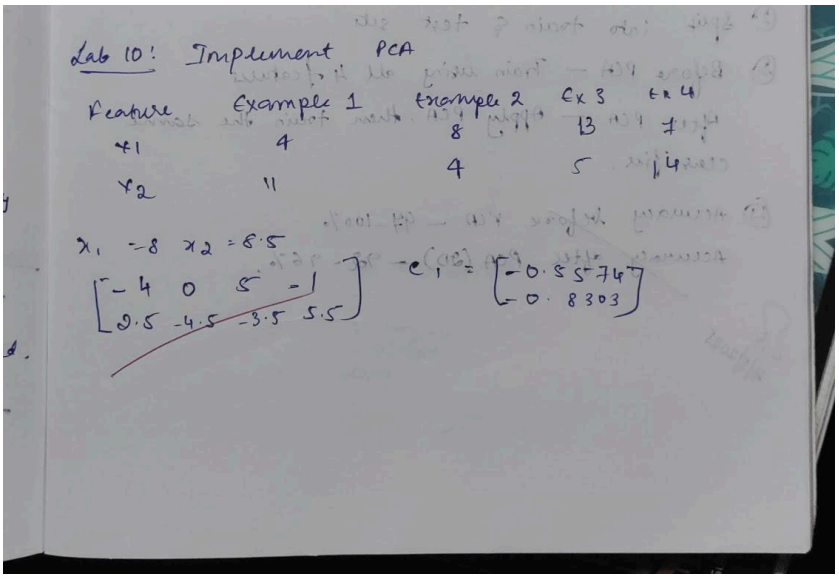


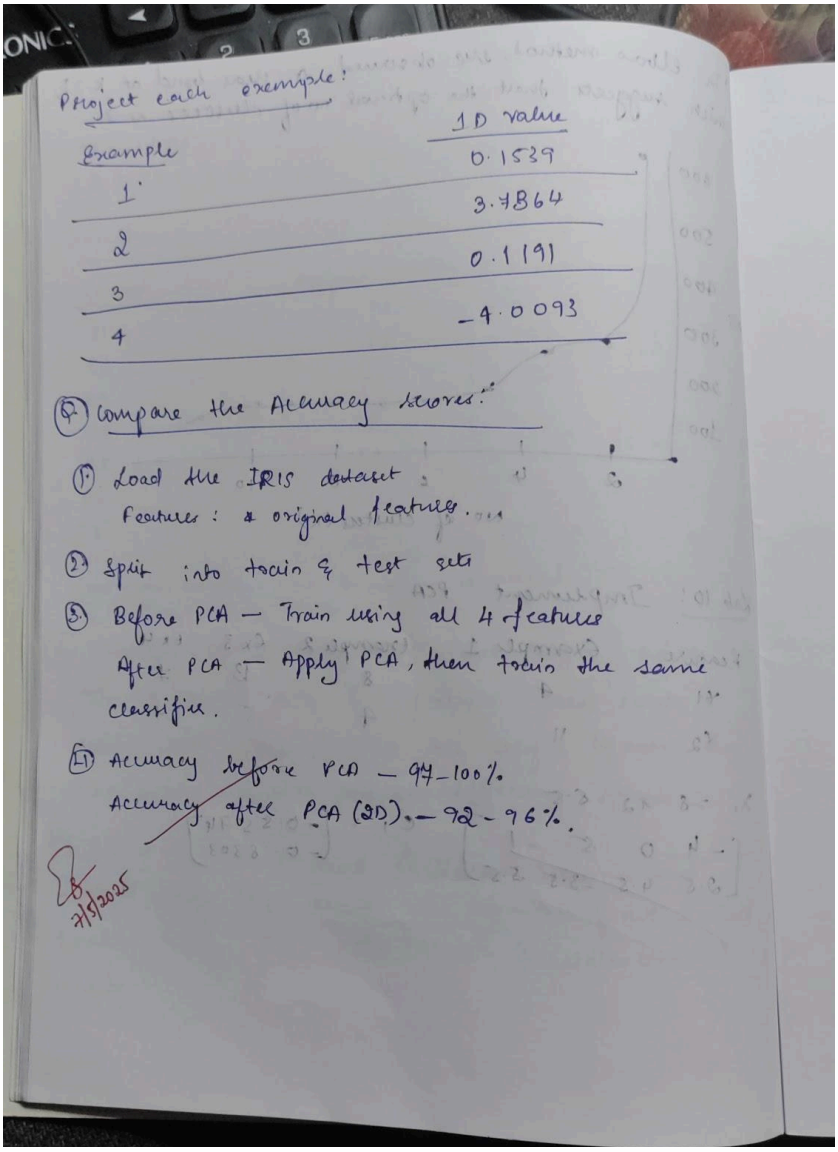
```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# 1. Load the Iris dataset:
data = pd.read_csv('/content/iris (6).csv')

# 2. Prepare the data:
X = data.iloc[:, :-1]  # Features (all columns except the last)

# 3. Standardize the features (important for k-Means):
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Determine the optimal number of clusters (using the Elbow method):
wcss = []  # Within-cluster sum of squares
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

# Plot the Elbow method graph
```

```
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('WCSS')
plt.show()

# Based on the Elbow method, choose the optimal k (e.g., k=3)

# 5. Apply k-Means clustering with the chosen k:
kmeans = KMeans(n_clusters=3, random_state=42)
clusters = kmeans.fit_predict(X_scaled)

# 6. Add the cluster labels to the DataFrame:
data['Cluster'] = clusters

# 7. Visualize the clusters (if possible - 2D or 3D):
# Example for 2 features: Sepal Length and Sepal Width
plt.scatter(data['SepalLengthCm'], data['SepalWidthCm'], c=data['Cluster'], cmap='viridis')
plt.title('k-Means Clustering')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.show()
```

Program 11

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






```
import pandas as pd

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt


# 1. Load the Iris dataset:

data = pd.read_csv('/content/heart.csv')
```

```
# 2. Prepare the data:
X = data.iloc[:, :-1]  # Features (all columns except the last)
y = data.iloc[:, -1]   # Target variable (last column)

# 3. Standardize the features (important for PCA):
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Apply PCA:
# Choose the number of components (n_components) you want to keep
# For example, to reduce to 2 dimensions:
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# 5. Create a new DataFrame with the reduced dimensions:
pca_df = pd.DataFrame(data=X_pca, columns=['PC1', 'PC2'])
pca_df['Species'] = y  # Add the target variable back for visualization

# 6. Visualize the reduced data:
plt.figure(figsize=(8, 6))
plt.scatter(pca_df['PC1'], pca_df['PC2'], c=pca_df['Species'].astype('category').cat.codes, cmap='viridis')
plt.title('PCA of HeartDataset')
plt.xlabel('Principal Component 1 (PC1)')
plt.ylabel('Principal Component 2 (PC2)')
```

```
plt.show()


# 7. Explained variance ratio:
# This tells you how much variance is explained by each principal component
explained_variance_ratio = pca.explained_variance_ratio_
print(f"Explained Variance Ratio: {explained_variance_ratio}")
```