VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

S Pranav Ranganath (1BM22CS355)

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 **S Pranav Ranganath (1BM22CS355),** 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 a Machine Learning (23CS6PCMAL) work prescribed for the said degree.

Lab Faculty Incharge Name: **Ms. Saritha A N** Assistant Professor Department of CSE, BMSCE	**Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

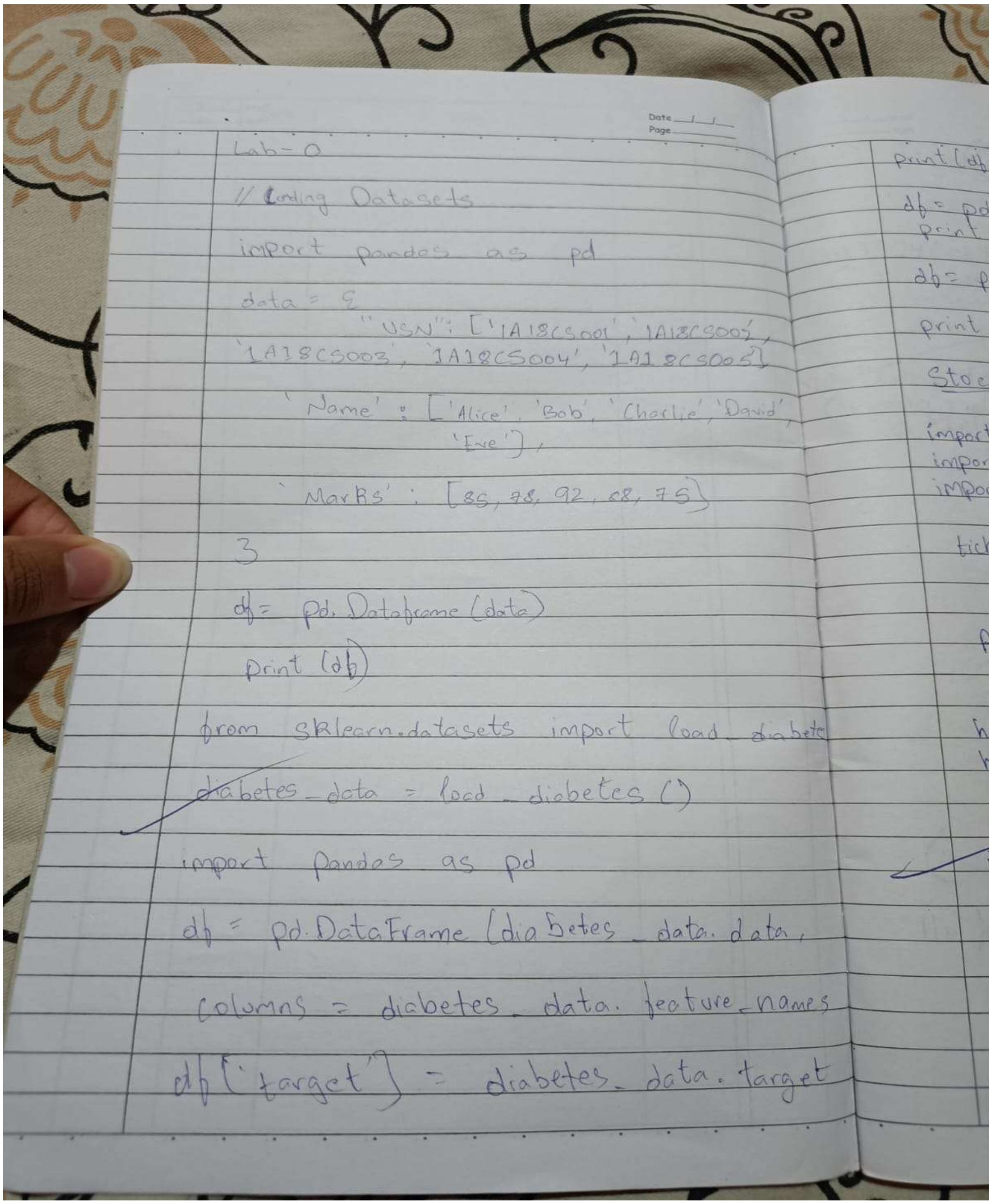
Index

Github Link: https://github.com/pranavsr29-ux/ML-Lab_1BM22CS355/tree/main

Program 1

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

Screenshot



Lab-0

```
// Loading Datasets
import pandas as pd
data = {
        "USN": ['1A18CS001', '1A18CS002',
'1A18CS003', '1A18CS004', '1A18CS005'],
    'Name': ['Alice', 'Bob', 'Charlie', 'David',
             'Eve'],
    'Marks': [85, 78, 92, 68, 75]
}
df = pd.Dataframe(data)
print(df)
from sklearn.datasets import load_diabetes
diabetes_data = load_diabetes()
import pandas as pd
df = pd.DataFrame(diabetes_data.data,
columns = diabetes_data.feature_names
df['target'] = diabetes_data.target
```

Code:

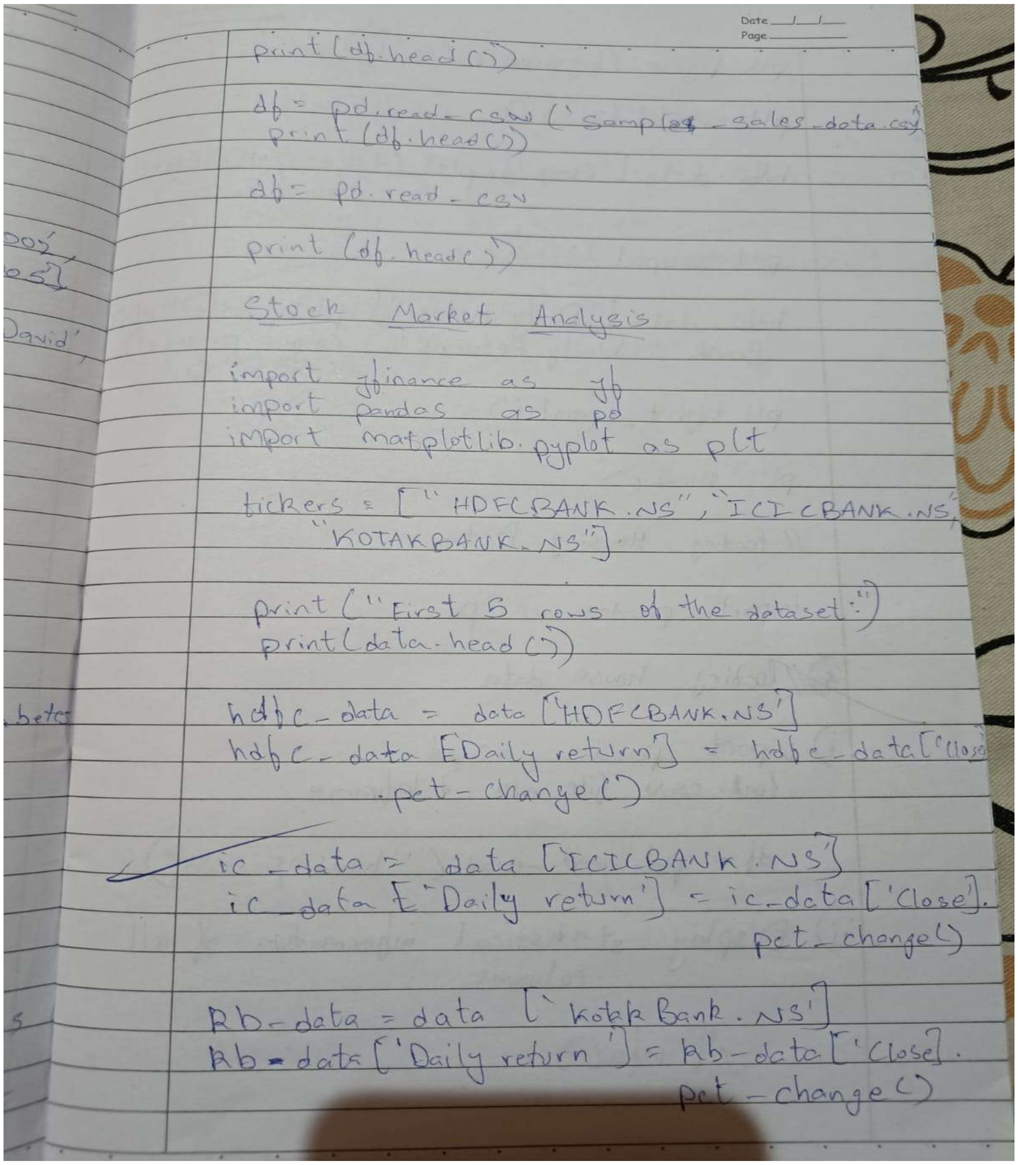
```
import pandas as pd
from sklearn.datasets import load_diabetes

# Part 1: Student data
student_data = {
    'USN': ['1A18CS001', '1A18CS002', '1A18CS003', '1A18CS004', '1A18CS005'],
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Marks': [85, 78, 92, 88, 75]
}
student_df = pd.DataFrame(student_data)
print("Student Data:")
print(student_df)
print("\n" + "-"*50 + "\n")

# Part 2: Load diabetes dataset from sklearn
diabetes_data = load_diabetes()
diabetes_df = pd.DataFrame(diabetes_data.data, columns=diabetes_data.feature_names)
diabetes_df['target'] = diabetes_data.target
print("Scikit-learn Diabetes Dataset:")
print(diabetes_df.head())
print("\n" + "-"*50 + "\n")

# Part 3: Load sample sales data from CSV
```

```
try:
    sales_df = pd.read_csv('sample_sales_data.csv')
    print("Sample Sales Data:")
    print(sales_df.head())
except FileNotFoundError:
    print("sample_sales_data.csv not found.")
print("\n" + "-"*50 + "\n")


# Part 4: Load diabetes dataset from external CSV
try:
    diabetes_csv_df = pd.read_csv('Dataset of Diabetes .csv')
    print("Diabetes Dataset from CSV:")
    print(diabetes_csv_df.head())
except FileNotFoundError:
    print("Dataset of Diabetes .csv not found.")
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:



```
print(df.head())

df = pd.read_csv('sample_sales_data.csv')
print(df.head())

df = pd.read_csv

print(df.head())

Stock Market Analysis

import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

tickers = ["HDFCBANK.NS", "ICICBANK.NS",
          "KOTAKBANK.NS"]

print("First 5 rows of the dataset:")
print(data.head())

hdfc_data = data['HDFCBANK.NS']
hdfc_data['Daily return'] = hdfc_data['Close']
           .pct_change()

ic_data = data['ICICBANK.NS']
ic_data['Daily return'] = ic_data['Close'].
                             pct_change()

kb_data = data['kotak Bank.NS']
kb_data['Daily return'] = kb_data['Close'].
                             pct_change()
```

Date __/__/__
Page ____

```
plt.figure(figsize=(12, 6))
plt.subplot(2,1,1)
hdfc-data['Close'].plot(title="HDFC Bank
        Closing Price")

plt.subplot(2,1,2)
hdfc_data['Daily Return'].plot(title="HDFC
  Bank - Daily Returns", color='orange')

plt.tight_layout()
plt.show()

// Loading Housing Data
import pandas as pd
// Loading house data
```

i) import pandas as pd
load csv file into dataframe

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

ii) Display statistical information of all columns

```
df.info()
```

Date __/__/__
Page ______

iii) Display information

print(df.describe())

iv) print(df ["ocean proximity"] .value_counts())

v) missing. N = df. isnull (). sum()
missing_c = missing_v [missing_v>0]

print (missing_c)

5/3/25

Code:

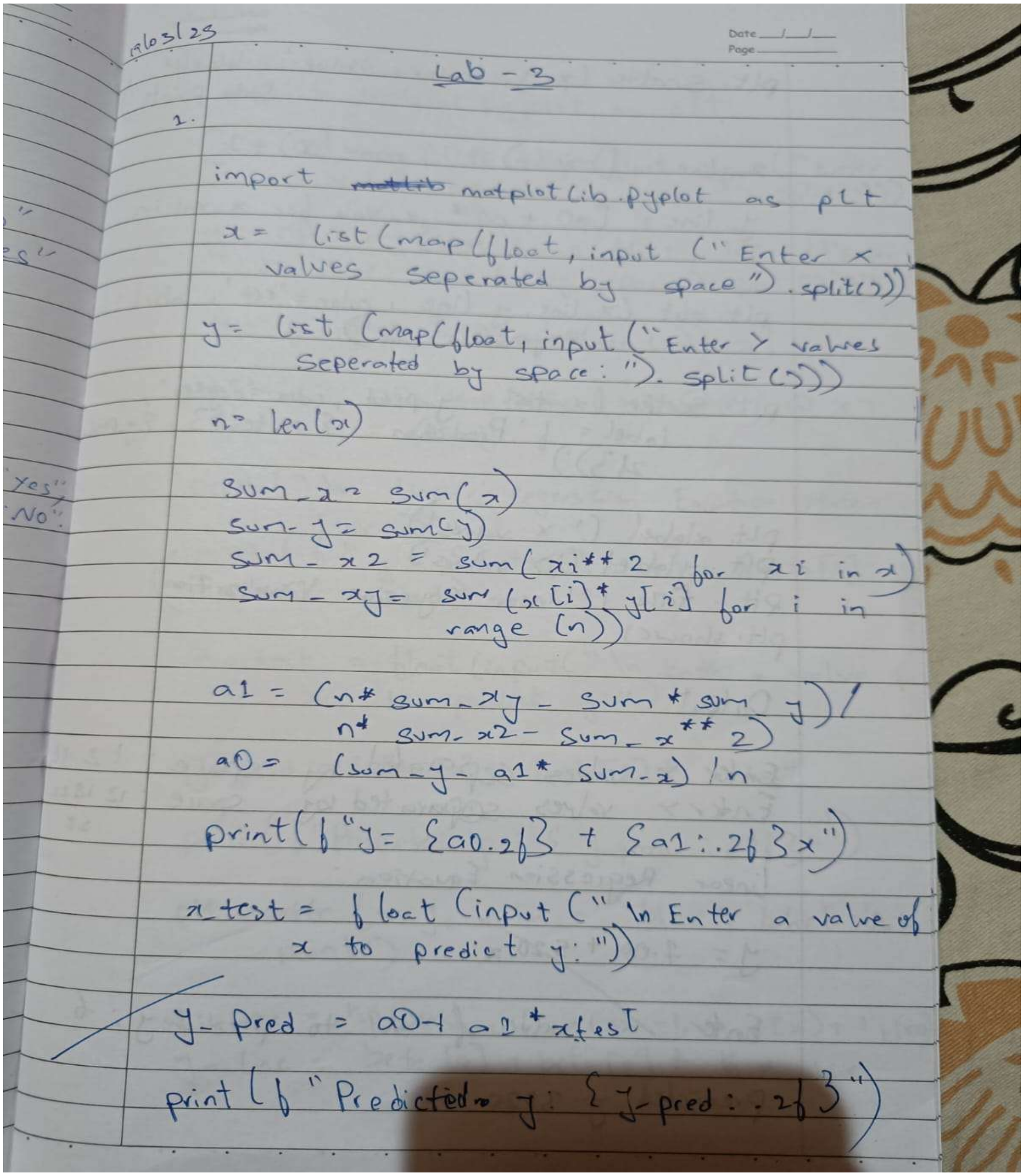
```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

# Fetch historical data for the last 1 year
data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group_by='ticker')

hdfc_data = data['HDFCBANK.NS']
hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()

ic_data = data['ICICIBANK.NS']
ic_data['Daily Return'] = ic_data['Close'].pct_change()

kb_data = data['KOTAKBANK.NS']
kb_data['Daily Return'] = kb_data['Close'].pct_change()

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
hdfc_data['Close'].plot(title="HDFC Bank - Closing Price")
plt.subplot(2, 1, 2)
hdfc_data['Daily Return'].plot(title="HDFC Bank - Daily Returns", color='orange')
```

```
plt.tight_layout()

plt.show()


plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)

ic_data['Close'].plot(title="ICICI Bank - Closing Price")

plt.subplot(2, 1, 2)

ic_data['Daily Return'].plot(title="ICICI Bank - Daily Returns", color='orange')

plt.tight_layout()

plt.show()


plt.figure(figsize=(12, 6))

plt.subplot(2, 1, 1)

kb_data['Close'].plot(title="Kotak Bank - Closing Price")

plt.subplot(2, 1, 2)

kb_data['Daily Return'].plot(title="Kotak Bank - Daily Returns", color='orange')

plt.tight_layout()

plt.show()
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot



19/05/25

Lab - 3

1.

```
import ~~matlib~~ matplotlib.pyplot as plt
x = list(map(float, input("Enter x values seperated by space").split()))
y = list(map(float, input("Enter Y values seperated by space: ").split()))
n = len(x)

sum_x = sum(x)
sum_y = sum(y)
sum_x2 = sum(xi**2 for xi in x)
sum_xy = sum(x[i]*y[i] for i in range(n))

a1 = (n* sum_xy - sum * sum_y)/
      n* sum_x2 - sum_x** 2)
a0 = (sum_y - a1* sum_x) / n
print(f"y= {a0.2f} + {a1:.2f} x")

x_test = float(input("\n Enter a value of x to predict y: "))

y_pred = a0+ a1 * xtest
print(f"Predicted y: {y_pred:.2f}")
```

Date ___/___/___
Page ______

2)

```
plt.scatter(x, y, color='blue', label='Data points')

x_line = [min(x)-1 , max(x)+1]

y_line = [a0 + a1 * x_val for x_val in x_line]

plt.plot(x_line, y_line, color='red', label='Regression Line')

plt.scatter(x_test, y_pred, color='green',
    label=f'Prediction: {x_test}, {y_pred:.2f}')

plt.xlabel('x values')
plt.ylabel('y values')
plt.title('Linear Regression Visualization')
plt.show()
```

Output

Enter x values separated by space: 1 2 3 4
Enter y values separated by space: 12 18 22 28

Linear Regression Equation:

$y = 7.00 + 5.20x$

Enter a value of x to predict y: 6

Predicted y: 38.20

2)

```
import numpy as np
import matplot.pyplot as plt

x = np.array(list(map(float, input("Enter
    x values seperated by space").split())))

y= np.array(list(map(float, input("Enter
    y values seperated by space") split())))

X = np.c_[np.ones(len(x)), x]

beta = np.linalg.inv(X.T @ X) @ X.T
    @ y

print("\n Linear Regression Equation (Matrix
    Form):")
print(f"y = {beta[0]:.2f} + {beta[1]:.2f}
    x")

x_test = float(input("\n Enter a value of
    x to predict y:"))

y_pred = beta[0] + beta[1] * x_test

print(f"Predicted y : {y_pred :.2f}")

plt.scatter(x, y, color = 'blue", label='Data
    point')

x_line = np.linespace(min(x) -1, max(x)+1, 100)
y_line = beta[0] + beta[1] * x_line
```

Date__/__/__
Page____

plt. show ()

Output

Enter x values separated by space:
1 2 3 4

Enter Y values ~~spa~~ seperated by space
12 18 22 28

Linear regression eq:

$y = 7.00 + 5.20x$

Enter a value of x to predict y: 5
predicted y: 33.0

19/3/2025

Code:

```
import matplotlib.pyplot as plt


x = list(map(float, input("Enter X values separated by space: ").split()))
y = list(map(float, input("Enter Y values separated by space: ").split()))
n = len(x)
sum_x = sum(x)
sum_y = sum(y)
sum_x2 = sum(xi**2 for xi in x)
sum_xy = sum(x[i] * y[i] for i in range(n))
a1 = (n * sum_xy - sum_x * sum_y) / (n * sum_x2 - sum_x**2)
a0 = (sum_y - a1 * sum_x) / n
print("\nLinear Regression Equation:")
print(f"y = {a0:.2f} + {a1:.2f}x")
x_test = float(input("\nEnter a value of x to predict y: "))
y_pred = a0 + a1 * x_test
print(f"Predicted y: {y_pred:.2f}")
plt.scatter(x, y, color='blue', label='Data Points')
x_line = [min(x) - 1, max(x) + 1]
y_line = [a0 + a1 * x_val for x_val in x_line]
plt.plot(x_line, y_line, color='red', label='Regression Line')
plt.scatter(x_test, y_pred, color='green', label=f'Prediction: ({x_test}, {y_pred:.2f})')
plt.xlabel('X values')
plt.ylabel('Y values')
```

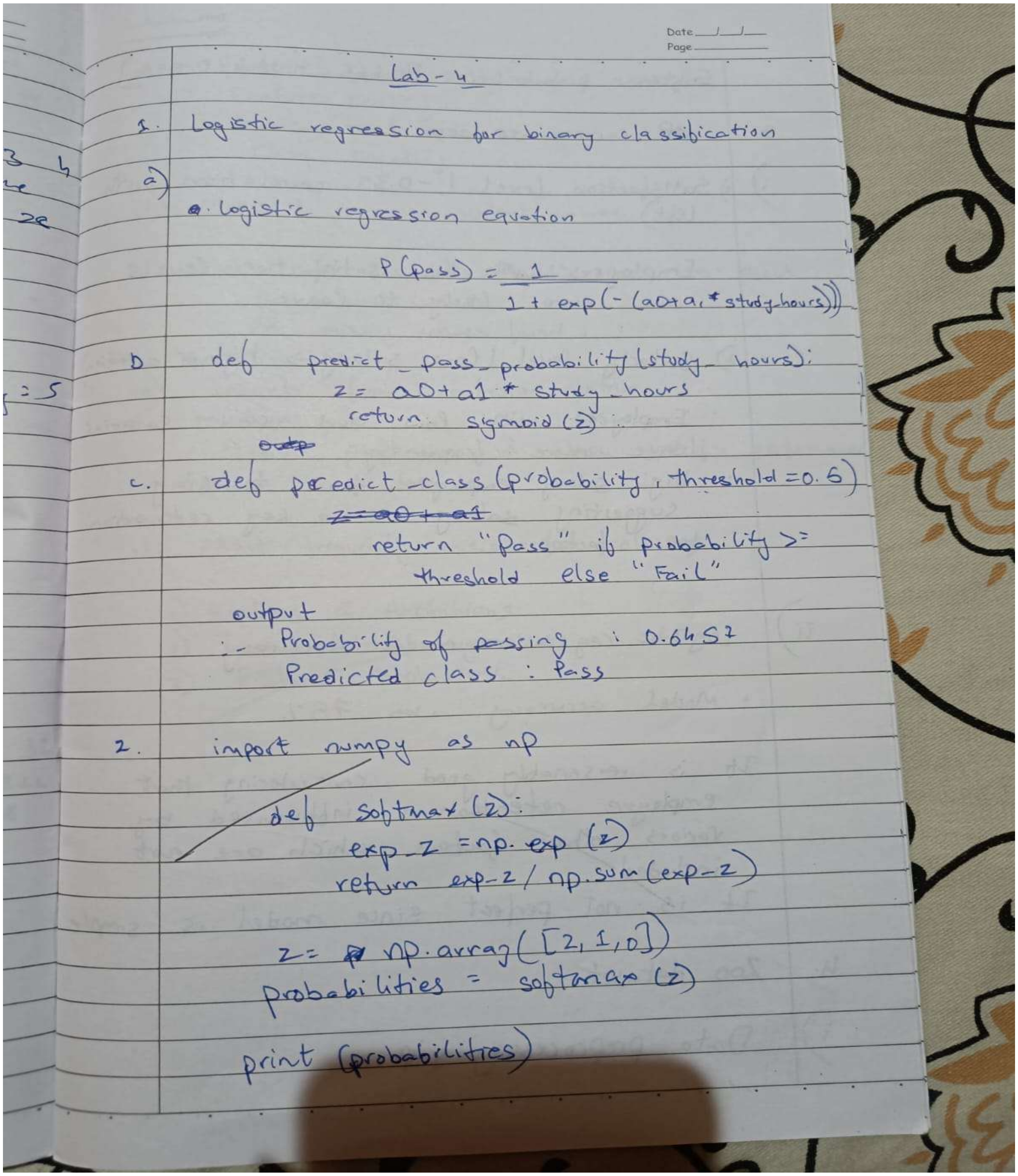
```
plt.title('Linear Regression Visualization')
plt.legend()
plt.grid(True)
plt.show()


import numpy as np
import matplotlib.pyplot as plt


x = np.array(list(map(float, input("Enter X values separated by space: ").split())))
y = np.array(list(map(float, input("Enter Y values separated by space: ").split())))
X = np.c_[np.ones(len(x)), x]
beta = np.linalg.inv(X.T @ X) @ X.T @ y
print("\nLinear Regression Equation (Matrix Form):")
print(f"y = {beta[0]:.2f} + {beta[1]:.2f}x")
x_test = float(input("\nEnter a value of x to predict y: "))
y_pred = beta[0] + beta[1] * x_test
print(f"Predicted y: {y_pred:.2f}")
plt.scatter(x, y, color='blue', label='Data Points')
x_line = np.linspace(min(x) - 1, max(x) + 1, 100)
y_line = beta[0] + beta[1] * x_line
plt.plot(x_line, y_line, color='red', label='Regression Line')
plt.scatter(x_test, y_pred, color='green', label=f'Prediction: ({x_test}, {y_pred:.2f})')
plt.xlabel('X values')
plt.ylabel('Y values')
```

```
plt.title('Linear Regression (Matrix Form)')
plt.legend()
plt.grid(True)
plt.show()
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot



Lab - 4

1. Logistic regression for binary classification

a) ~~a~~. logistic regression equation

$$P(\text{pass}) = \frac{1}{1 + \exp(-(a0 + a_1 * \text{study\_hours}))}$$

b
```
def predict_pass_probability(study_hours):
    z = a0 + a1 * study_hours
    return sigmoid(z)
```

~~outp~~

c.
```
def predict_class(probability, threshold=0.5)
    ~~z = a0 + a1~~
    return "Pass" if probability >=
        threshold else "Fail"
```

output
Probability of passing : 0.6457
Predicted class : Pass

2.
```
import numpy as np

def softmax(z):
    exp_z = np.exp(z)
    return exp_z / np.sum(exp_z)

z = np.array([2, 1, 0])
probabilities = softmax(z)

print(probabilities)
```

Date / /
Page

Softmax probabilities [0.665, 0.244, 0.0900]

3. HR_comma_sep.csv.

i) 1) Satisfaction level (-0.39 correlation with left)

· Employees with low satisfaction levels are more likely to leave

2) Salarly level (low salary -> higher attrition)

· Employees with low and medium salaries leave more frequently
· High salary employees tend to stay, suggesting salary is a key retenention factor.

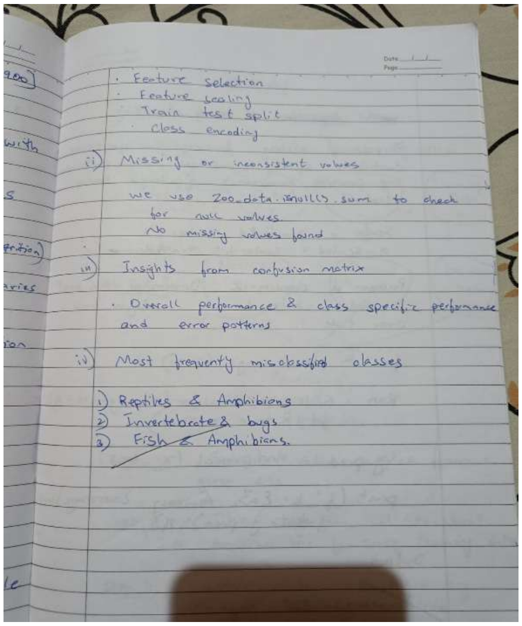
ii) Logistic regression model Accuracy

· Model accuracy was 76%.

It is reasonably good, considering that employee retention is influenced by various other factors which are not included.
It is not perfect since model is simple

4. Zoo dataset

i) Data preprocessing steps



- Feature selection
- Feature scaling
- Train test split
- Class encoding

ii) Missing or inconsistent values

we use zoo_data.isnull().sum to check for null values.
No missing values found

iii) Insights from confusion matrix

- Overall performance & class specific performance and error patterns

iv) Most frequenty misclassified classes

1) Reptiles & Amphibians
2) Invertebrate & bugs
3) Fish & Amphibians.

Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# Load dataset

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


# Convert categorical variables to numerical

label_enc = LabelEncoder()
```

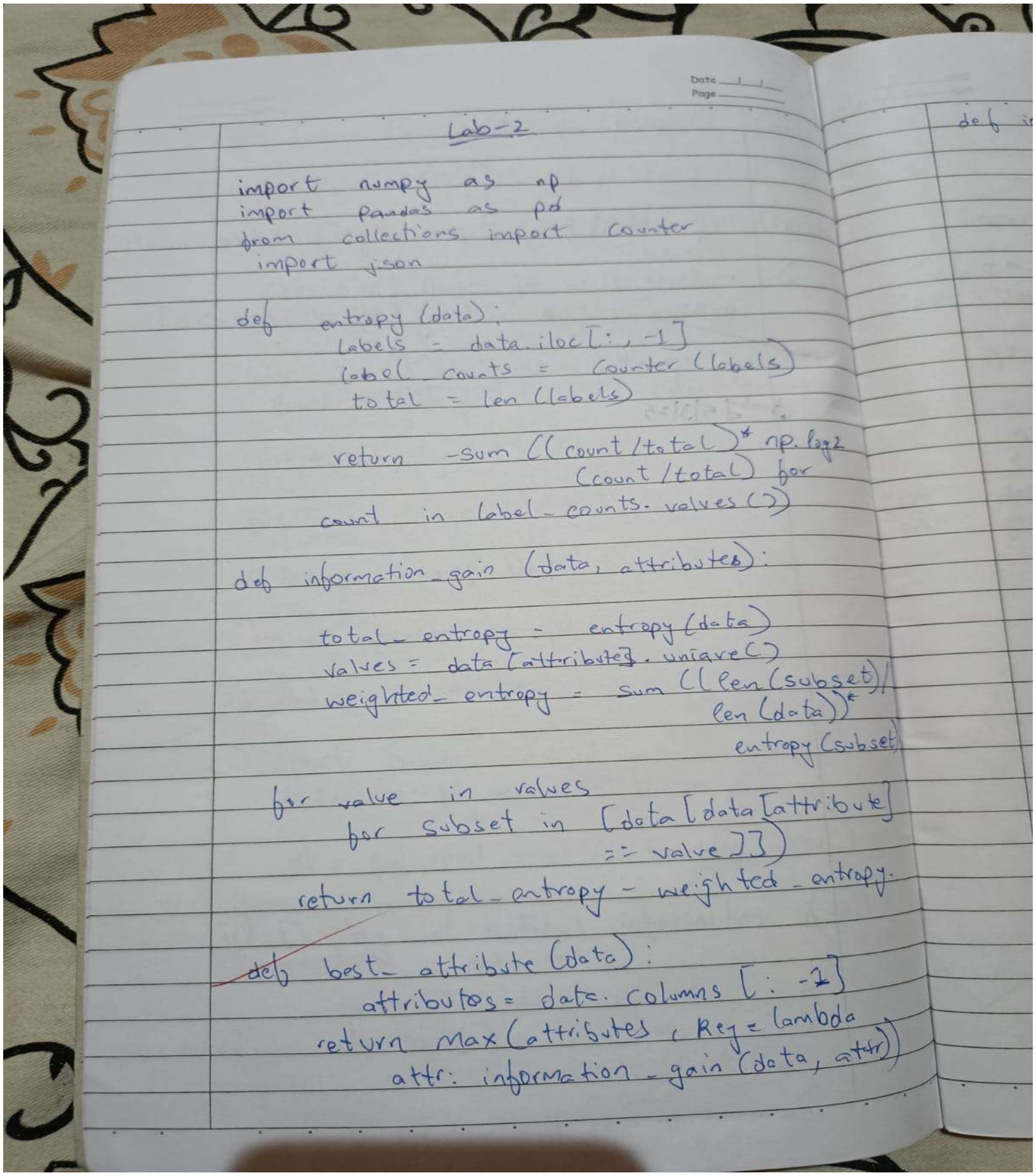
```
df["salary"] = label_enc.fit_transform(df["salary"])
df["Department"] = label_enc.fit_transform(df["Department"])


# Step 1: Exploratory Data Analysis (EDA)
correlation = df.corr()["left"].sort_values(ascending=False)
print("\nFeature Correlation with Employee Retention:\n", correlation)


# Step 2: Impact of Salary on Retention
plt.figure(figsize=(6,4))
sns.barplot(x="salary", y="left", data=df, ci=None)
plt.xlabel("Salary Level (Encoded)")
plt.ylabel("Retention Rate")
plt.title("Impact of Salary on Employee Retention")
plt.show()


# Step 3: Correlation between Department and Retention
plt.figure(figsize=(8,4))
sns.barplot(x="Department", y="left", data=df, ci=None)
plt.xlabel("Department (Encoded)")
plt.ylabel("Retention Rate")
plt.title("Department vs Employee Retention")
plt.show()


# Step 4: Logistic Regression Model
```

```
features = ["satisfaction_level", "last_evaluation", "number_project", "average_montly_hours",
"time_spend_company", "salary"]

X = df[features]

y = df["left"]


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


model = LogisticRegression()

model.fit(X_train, y_train)


# Step 5: Model Accuracy

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

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


# Display Confusion Matrix

conf_matrix = confusion_matrix(y_test, y_pred)

print("\nConfusion Matrix:\n", conf_matrix)


# Display Classification Report

print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Program 5

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

Screenshot



Lab-2

```
import numpy as np
import pandas as pd
from collections import counter
import json

def entropy (data):
    Labels = data.iloc[:, -1]
    label_counts = Counter (labels)
    total = len (labels)

    return -sum ((count/total)* np.log2
                  (count/total) for
    count in label_counts.values ())

def information_gain (data, attributes):

    total_entropy = entropy(data)
    values = data[attributes].uniaue()
    weighted_entropy = sum ((len(subset)/
                          len (data))*
                              entropy (subset)

  for value in values
      for subset in [data [data [attribute]
                       == value]])
    return total_entropy - weighted_entropy.

def best_attribute (data):
      attributes = data.columns [: -1]
    return max(attributes, key = lambda
       attr: information_gain (data, attr))
```

```
def id3 (data, features):
    labels = data.iloc[:, -1]
    if len(set(labels)) == 1:
        return labels.iloc[0]

    if len(features) == 0:

        return labels.mode()[0]

    best_attr = best_attribute(data)

    tree = {best_attr: {}}

    for value in data[best_attr].unique():

        subset = data[data[best_attr] == value].
                 drop(columns = [best_attr])

        tree[best_attr][value] = id3(subset,
               subset.columns[:-1])

    return tree

data = pd.read_csv("weather_data.csv")

decision_tree = id3(data, list(data.columns[:-1]))

print(json.dumps(decision_tree, indent=4))
```

P.T.O

Output

```
{
    "Outlook": {
        "Sunny": {
            "Humidity": {
                "High": "No"
                "Normal" = "Yes"
            }
        },
        "Overcast" = "Yes"

        "Rain": {
            "Wind": {
                "Weak": "Yes",
                "Strong": "No"
            }
        }
    }
}
```

A 12/3/25

Code:

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

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

# Convert categorical variables to numerical
label_enc = LabelEncoder()
df["salary"] = label_enc.fit_transform(df["salary"])
df["Department"] = label_enc.fit_transform(df["Department"])

# Step 1: Exploratory Data Analysis (EDA)
correlation = df.corr()["left"].sort_values(ascending=False)
print("\nFeature Correlation with Employee Retention:\n", correlation)

# Step 2: Impact of Salary on Retention
plt.figure(figsize=(6,4))
```

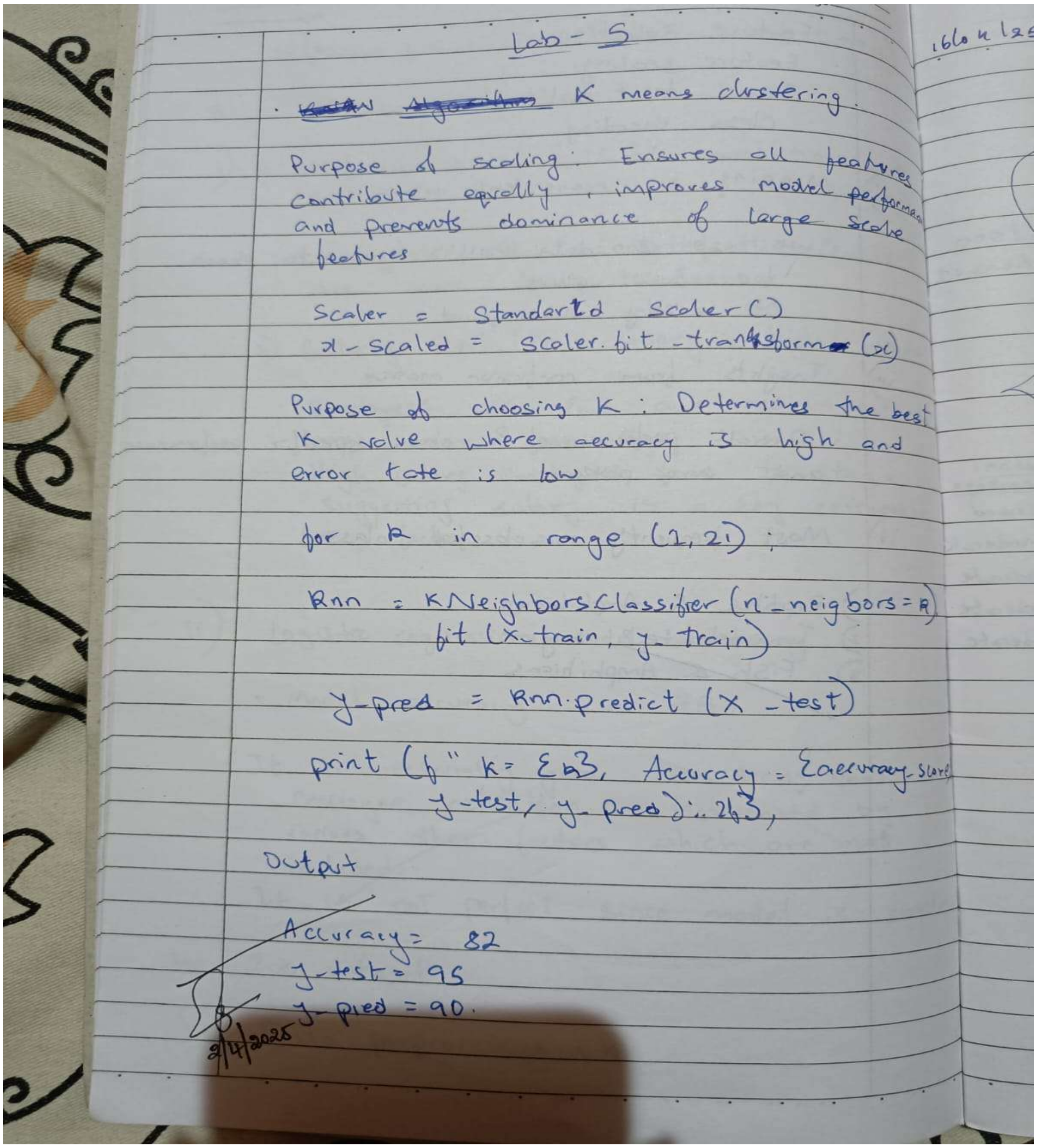
```
sns.barplot(x="salary", y="left", data=df, ci=None)

plt.xlabel("Salary Level (Encoded)")

plt.ylabel("Retention Rate")

plt.title("Impact of Salary on Employee Retention")

plt.show()


# Step 3: Correlation between Department and Retention

plt.figure(figsize=(8,4))

sns.barplot(x="Department", y="left", data=df, ci=None)

plt.xlabel("Department (Encoded)")

plt.ylabel("Retention Rate")

plt.title("Department vs Employee Retention")

plt.show()


# Step 4: Logistic Regression Model

features = ["satisfaction_level", "last_evaluation", "number_project", "average_montly_hours",
"time_spend_company", "salary"]

X = df[features]

y = df["left"]


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


model = LogisticRegression()

model.fit(X_train, y_train)
```

```
# Step 5: Model Accuracy

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

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


# Display Confusion Matrix

conf_matrix = confusion_matrix(y_test, y_pred)

print("\nConfusion Matrix:\n", conf_matrix)


# Display Classification Report

print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

Program 6

Build KNN Classification model for a given dataset

Screenshot



Lab - 5

~~KNN Algorithm~~ K means clustering.

Purpose of scaling: Ensures all features contribute equally, improves model performance and prevents dominance of large scale features

Scaler = Standard Scaler()
x - scaled = scaler.fit_transform(x)

Purpose of choosing K: Determines the best K value where accuracy is high and error rate is low

for k in range (1, 21):

knn = KNeighborsClassifier (n_neigbors = k)
fit (x_train, y_train)

y-pred = knn.predict (x_test)

print (f" k= {k}, Accuracy = {accuracy_score(y_test, y_pred):.2f}",

Output

Accuracy = 82
y-test = 95
y-pred = 90.

2/4/2025

Code:

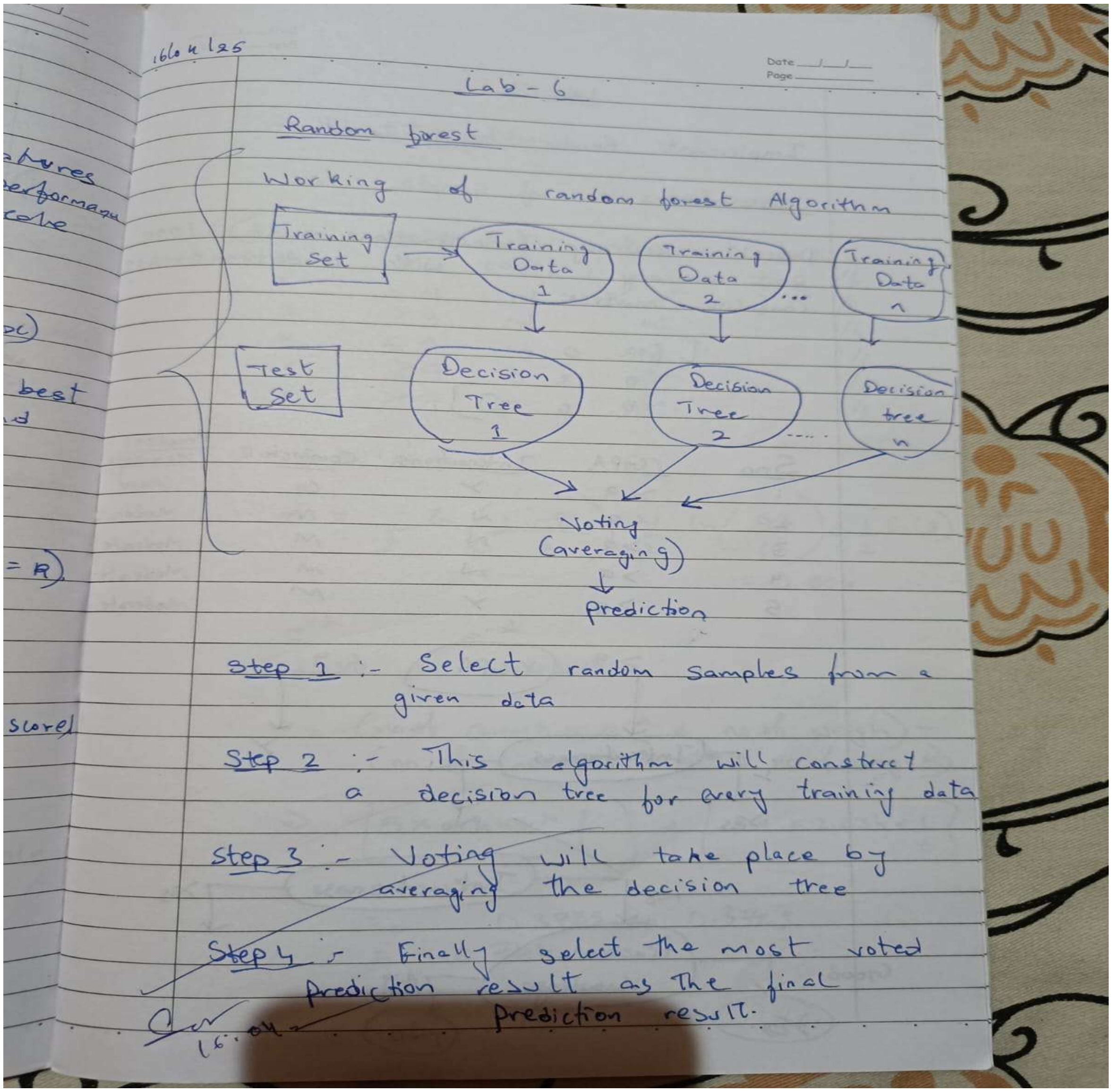
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

file_path = input("Enter the path to your CSV file: ")
df = pd.read_csv(file_path)

# Automatically select the target column (assuming it's the last column)
target_column = df.columns[-1]

# Splitting features (X) and target variable (y)
X = df.iloc[:, :-1]  # All columns except last (features)
y = df.iloc[:, -1]   # Last column (target)

# Encode categorical target variable if necessary
if y.dtype == 'object':
    label_encoder = LabelEncoder()
    y = label_encoder.fit_transform(y)

# Convert categorical features to numerical if any
X = pd.get_dummies(X, drop_first=True)

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

# Standardize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Determine optimal k (square root heuristic)
k = int(np.sqrt(len(y_train)))
if k % 2 == 0:
    k += 1

# Train the KNN classifier
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)

# Make predictions
y_pred = knn.predict(X_test)
```

```
# Display accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"\nAccuracy Score: {accuracy:.2f}")

# Display confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", conf_matrix)

# Display classification report
class_report = classification_report(y_test, y_pred)
print("\nClassification Report:\n", class_report)

# Plot Confusion Matrix
plt.figure(figsize=(6,5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y),
yticklabels=np.unique(y))
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.title("Confusion Matrix")
plt.show()
```

Program 7

Implement Random Forest ensemble method on a given dataset

Screenshot


16/04/25

Lab - 6

Random forest

Working of random forest Algorithm

Training Set

Training Data 1

Training Data 2

Training Data n

Test Set

Decision Tree 1

Decision Tree 2

Decision tree n

Voting (averaging)

prediction

Step 1 :- Select random samples from a given data

Step 2 :- This algorithm will construct a decision tree for every training data

Step 3 :- Voting will take place by averaging the decision tree

Step 4 :- Finally select the most voted prediction result as the final prediction result.

Lab - 7

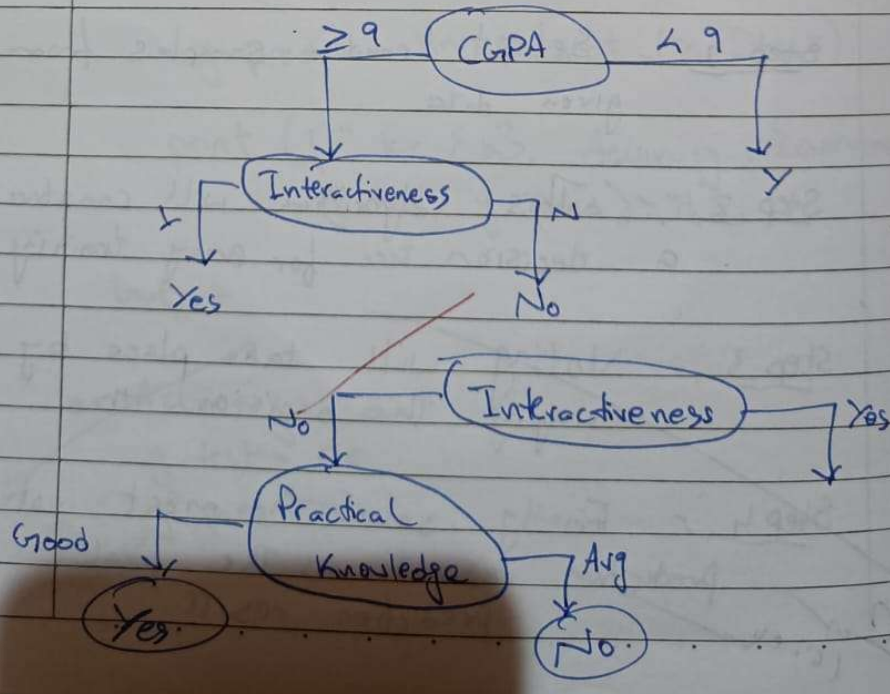
Implement Random Forest

Observation

Default accuracy (n estimations = 16) : 1.000
Best accuracy : 1.000 with n estimations :
Confusion matrix

$$\begin{bmatrix} 19 & 0 & 0 \\ 0 & 13 & 0 \\ 0 & 0 & 13 \end{bmatrix}$$

Sno.	CGPA	Interactions	Commskill	Practical Knowledge
1	≥ 9	Y	G	Good
2	< 9	N	M	Moderate
3	≥ 9	N	M	Moderate
4	> 9	N	M	Moderate
5	≥ 9	Y	M	Moderate



Date __/__/__
Page ____

Lab - 7

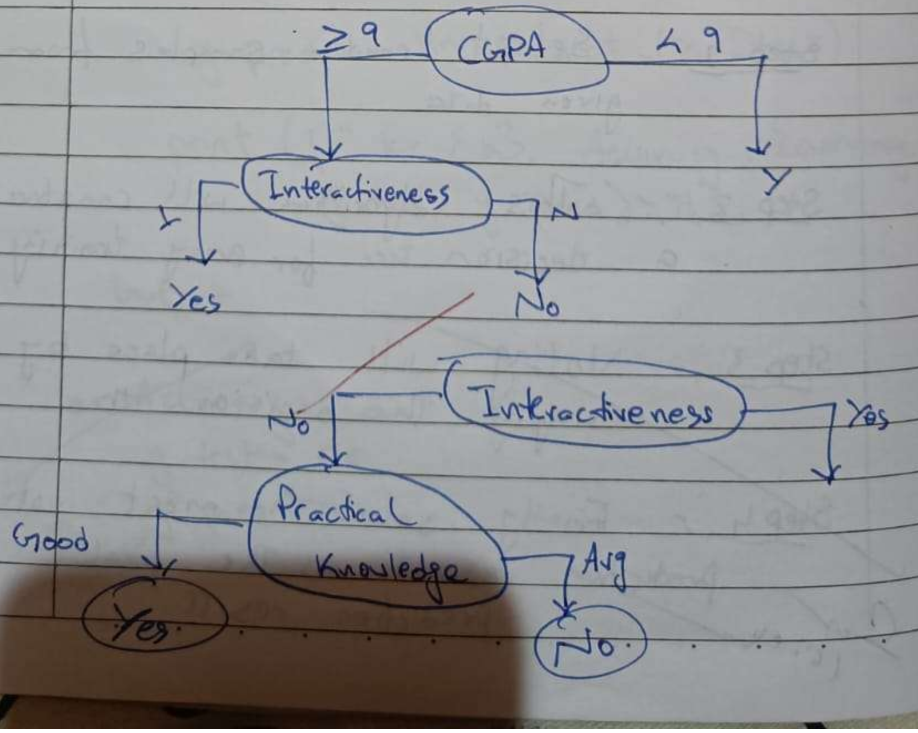
Implement Random Forest

Observation

Default accuracy (n estimations = 16) : 1.000
Best accuracy : 1.000 with n estimations :
Confusion matrix

[[19 0 0
0 13 0
0 0 13]]

Sno.	CGPA	Interactions	Commskill	Practical Knowledge
1	≥9	Y	G	Good
2	<9	N	M	Moderate
3	≥9	N	M	Moderate
4	>9	N	M	Moderate
5	≥9	Y	M	Moderate



Code:

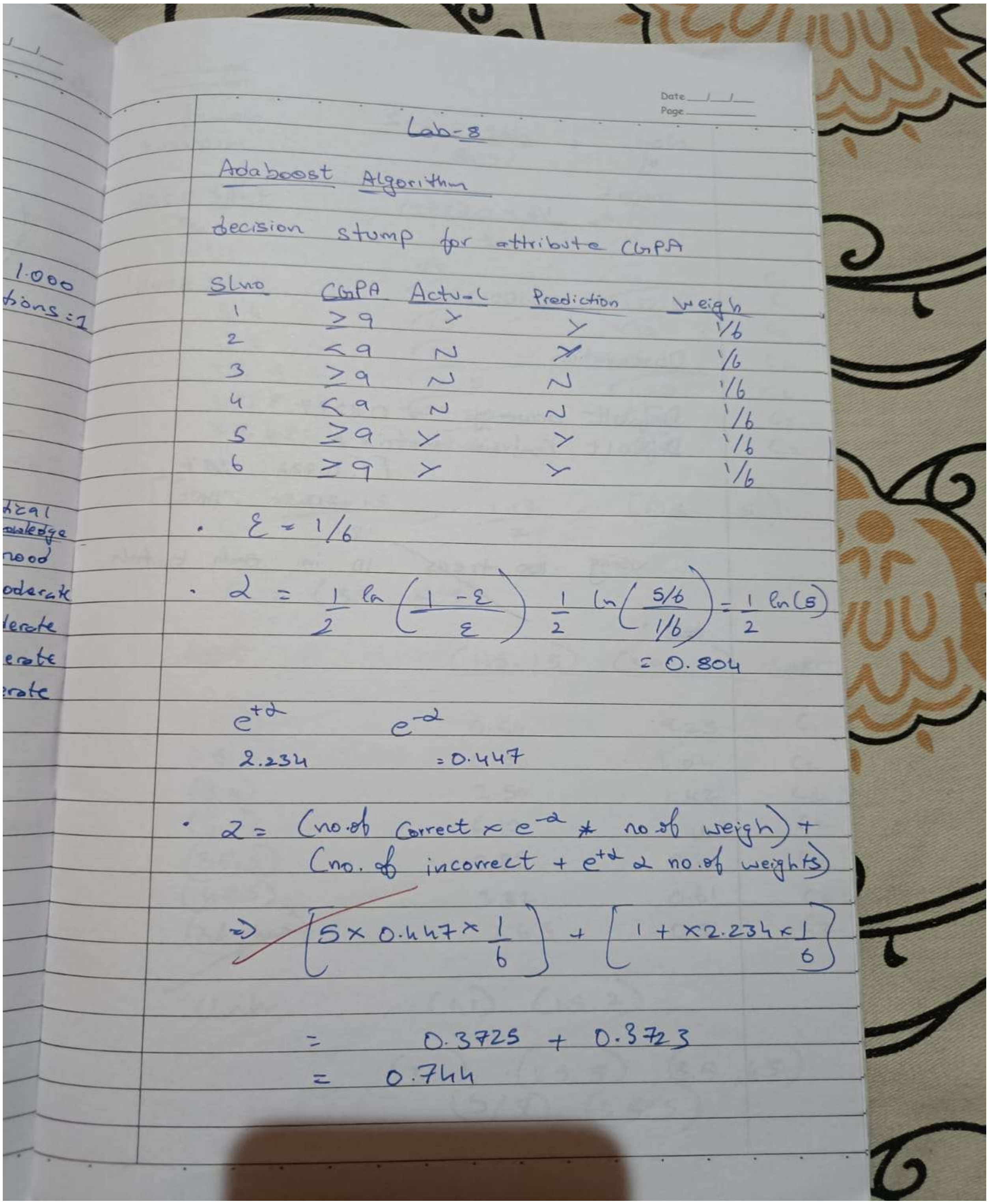
```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

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

# Features and target
X = df.drop(columns=["species"])
y = df["species"]

# Split into train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 1. Train with default n_estimators = 10
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
default_score = accuracy_score(y_test, y_pred_default)

print(f"Accuracy with default (10 trees): {default_score:.4f}")
```

```
# 2. Fine-tune n_estimators
scores = []
tree_range = range(1, 51)

for n in tree_range:
    model = RandomForestClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    scores.append(acc)
# Plotting results
plt.figure(figsize=(10,5))
plt.plot(tree_range, scores, marker='o')
plt.title("Random Forest Accuracy vs Number of Trees")
plt.xlabel("Number of Trees (n_estimators)")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
# Best result
best_score = max(scores)
best_n = tree_range[scores.index(best_score)]

print(f"Best Accuracy = {best_score:.4f} with n_estimators = {best_n}")
```

Program 8

Implement Boosting ensemble method on a given dataset

Screenshot



Lab-8

Adaboost Algorithm

decision stump for attribute CGPA

Sl.no	CGPA	Actual	Prediction	weigh
1	≥ 9	Y	Y	1/6
2	< 9	N	~~Y~~	1/6
3	≥ 9	N	N	1/6
4	< 9	N	N	1/6
5	≥ 9	Y	Y	1/6
6	≥ 9	Y	Y	1/6

- $\varepsilon = 1/6$

- $\alpha = \frac{1}{2} \ln\left(\frac{1-\varepsilon}{\varepsilon}\right) \; \frac{1}{2} \ln\left(\frac{5/6}{1/6}\right) = \frac{1}{2} \ln(5)$

$= 0.804$

$e^{+\alpha}$ $\quad$ $e^{-\alpha}$

2.234 $\quad$ $= 0.447$

- $\alpha =$ (no. of correct $\times e^{-\alpha} *$ no. of weigh) +
(no. of incorrect + $e^{+\alpha}$ α no. of weights)

$\Rightarrow \left[5 \times 0.447 \times \frac{1}{6}\right] + \left[1 + \times 2.234 \times \frac{1}{6}\right]$

$= 0.3725 + 0.3723$

$= 0.744$

Date __/__/__
Page ____

New of correct $= \frac{old\,w \times e^{-2}}{2}$ New of incorrect

$= \frac{1/6 \times 0.447}{2}$ $\frac{1/6 \times 0.234}{2}$

$= 0.10013$ 0.5004

Observation

Default accuracy = 0.8277
Default Confusion matrix ~~=~~ :
[[10722 387
2138 1406]]

using ~~100 trees~~ 10 in each batch

~~tate~~ ,

Code:

```
import pandas as pd
import numpy as np
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# Load the dataset
data = pd.read_csv('income.csv')

# Explore the dataset
print(data.head())
print("\nDataset info:")
print(data.info())
print("\nClass distribution:")
print(data['income_level'].value_counts())

# Split into features and target
X = data.drop('income_level', axis=1)
y = data['income_level']

# Encode categorical features (one-hot encoding)
```

```
X = pd.get_dummies(X)

# Check for missing values
if X.isnull().sum().sum() > 0:
    print("Missing values found. Filling missing values with column mean.")
    X = X.fillna(X.mean())

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y)

# Define base estimator
base_estimator = DecisionTreeClassifier(max_depth=1)  # Stump tree for AdaBoost

# Initial AdaBoost model with 10 estimators
ada_model = AdaBoostClassifier(
    estimator=base_estimator,
    n_estimators=10,
    random_state=42
)

# Train the model
ada_model.fit(X_train, y_train)
```

```
# Make predictions
y_pred = ada_model.predict(X_test)

# Evaluate initial model
print("\nInitial Model with 10 estimators:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Fine-tune the number of trees
n_estimators_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200]
train_scores = []
test_scores = []

for n in n_estimators_range:
    model = AdaBoostClassifier(
        estimator=base_estimator,
        n_estimators=n,
        random_state=42
    )
    model.fit(X_train, y_train)

    # Training accuracy
    train_pred = model.predict(X_train)
```

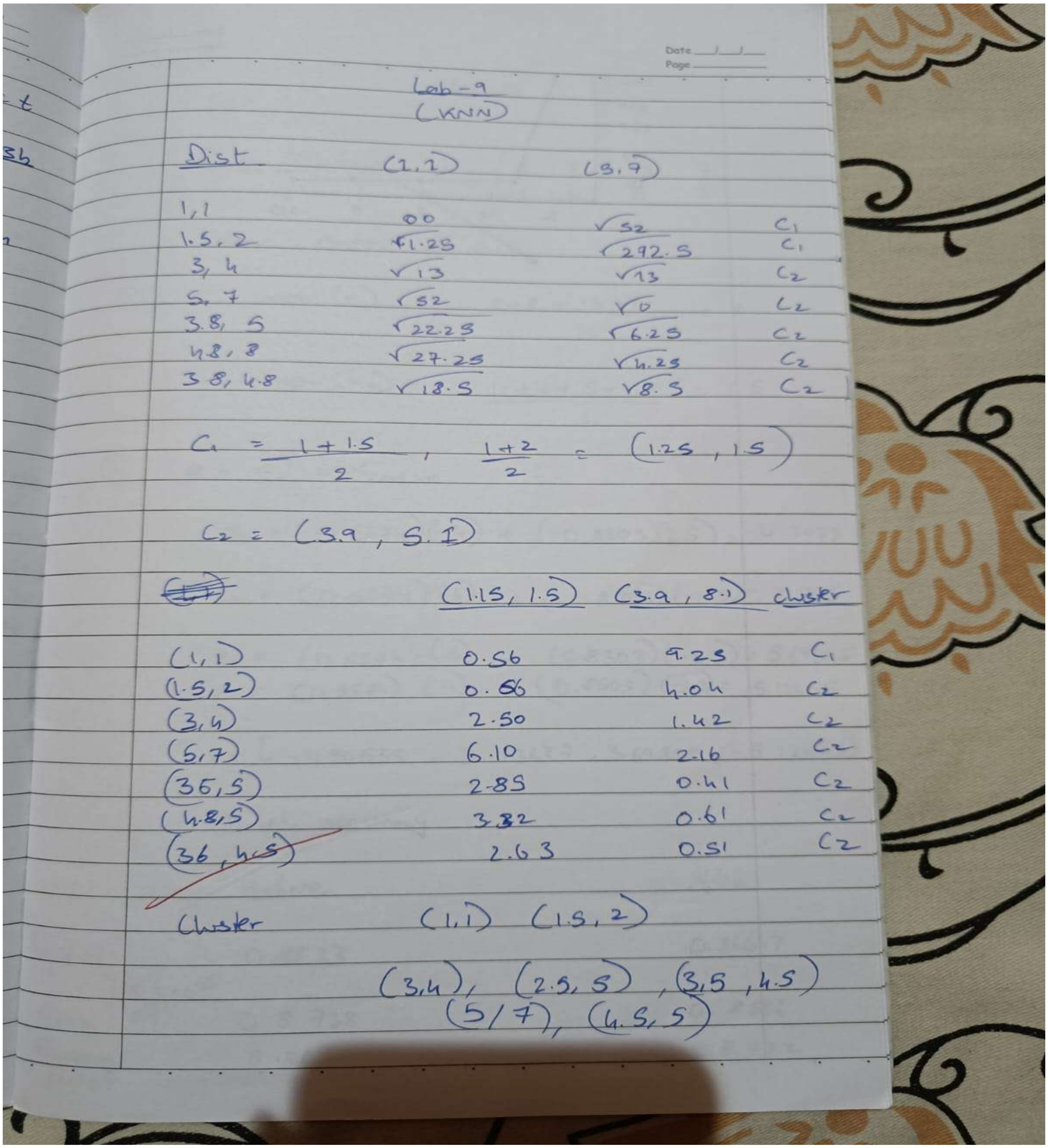
```
    train_acc = accuracy_score(y_train, train_pred)
    train_scores.append(train_acc)

    # Test accuracy
    test_pred = model.predict(X_test)
    test_acc = accuracy_score(y_test, test_pred)
    test_scores.append(test_acc)

    print(f"n_estimators: {n}, Train Accuracy: {train_acc:.4f}, Test Accuracy: {test_acc:.4f}")

# Find the best number of estimators
best_n = n_estimators_range[np.argmax(test_scores)]
best_score = max(test_scores)

print(f"\nBest performance: n_estimators={best_n} with test accuracy of {best_score:.4f}")

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(n_estimators_range, train_scores, label='Train Accuracy', marker='o')
plt.plot(n_estimators_range, test_scores, label='Test Accuracy', marker='o')
plt.xlabel('Number of Estimators')
plt.ylabel('Accuracy')
plt.title('AdaBoost Performance vs Number of Estimators')
plt.axvline(x=best_n, color='r', linestyle='--', label=f'Best n_estimators={best_n}')
```

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

# Train final model with best number of estimators
final_model = AdaBoostClassifier(
    estimator=base_estimator,
    n_estimators=best_n,
    random_state=42
)
final_model.fit(X_train, y_train)

# Evaluate final model
final_pred = final_model.predict(X_test)
print("\nFinal Model Performance:")
print("Accuracy:", accuracy_score(y_test, final_pred))
print("\nClassification Report:")
print(classification_report(y_test, final_pred))

print("\nFeature Importances:")
feature_importances = pd.Series(final_model.feature_importances_, index=X.columns)
print(feature_importances.sort_values(ascending=False))
```

Program 9

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

Screenshot



Lab - 9
(KNN)

Dist	(1,1)	(3,7)	
1,1	0 0	√52	C1
1.5, 2	√1.25	√292.5	C1
3, 4	√13	√13	C2
5, 7	√52	√0	C2
3.8, 5	√22.25	√6.25	C2
4.8, 8	√27.25	√4.25	C2
3.8, 4.8	√18.5	√8.5	C2

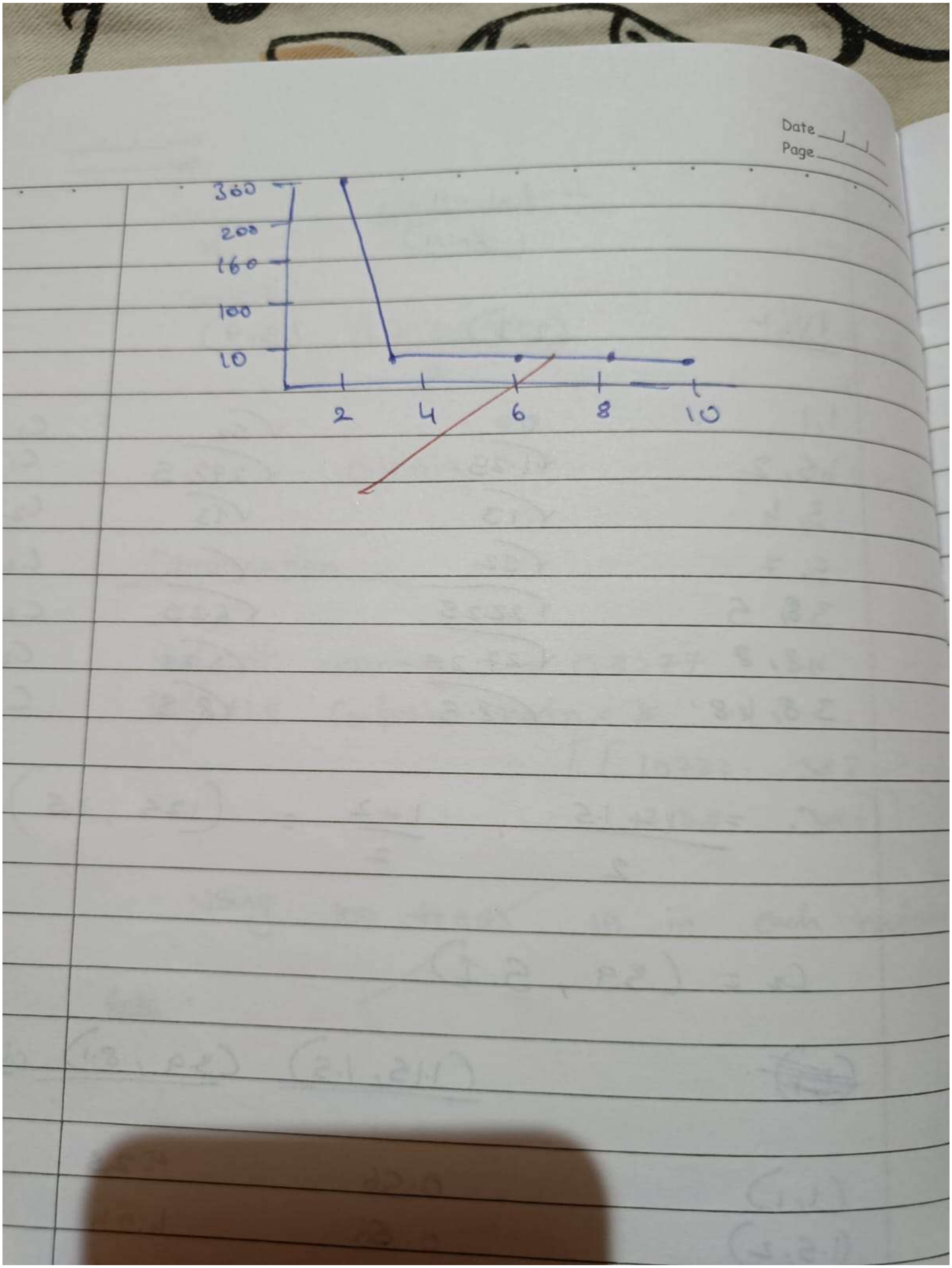
$C_1 = \frac{1+1.5}{2}, \frac{1+2}{2} = (1.25, 1.5)$

$C_2 = (3.9, 5.1)$

	(1.15, 1.5)	(3.9, 8.1)	cluster
(1,1)	0.56	9.25	C1
(1.5,2)	0.56	4.04	C2
(3,4)	2.50	1.42	C2
(5,7)	6.10	2.16	C2
(35,5)	2.85	0.41	C2
(4.8,5)	3.82	0.61	C2
(36, 4.5)	2.63	0.51	C2

Cluster (1,1) (1.5, 2)

(3,4), (2.5, 5), (3.5, 4.5)
(5/7), (4.5, 5)


Date / /
Page
300
200
160
100
10
2
4
6
8
10

Code:

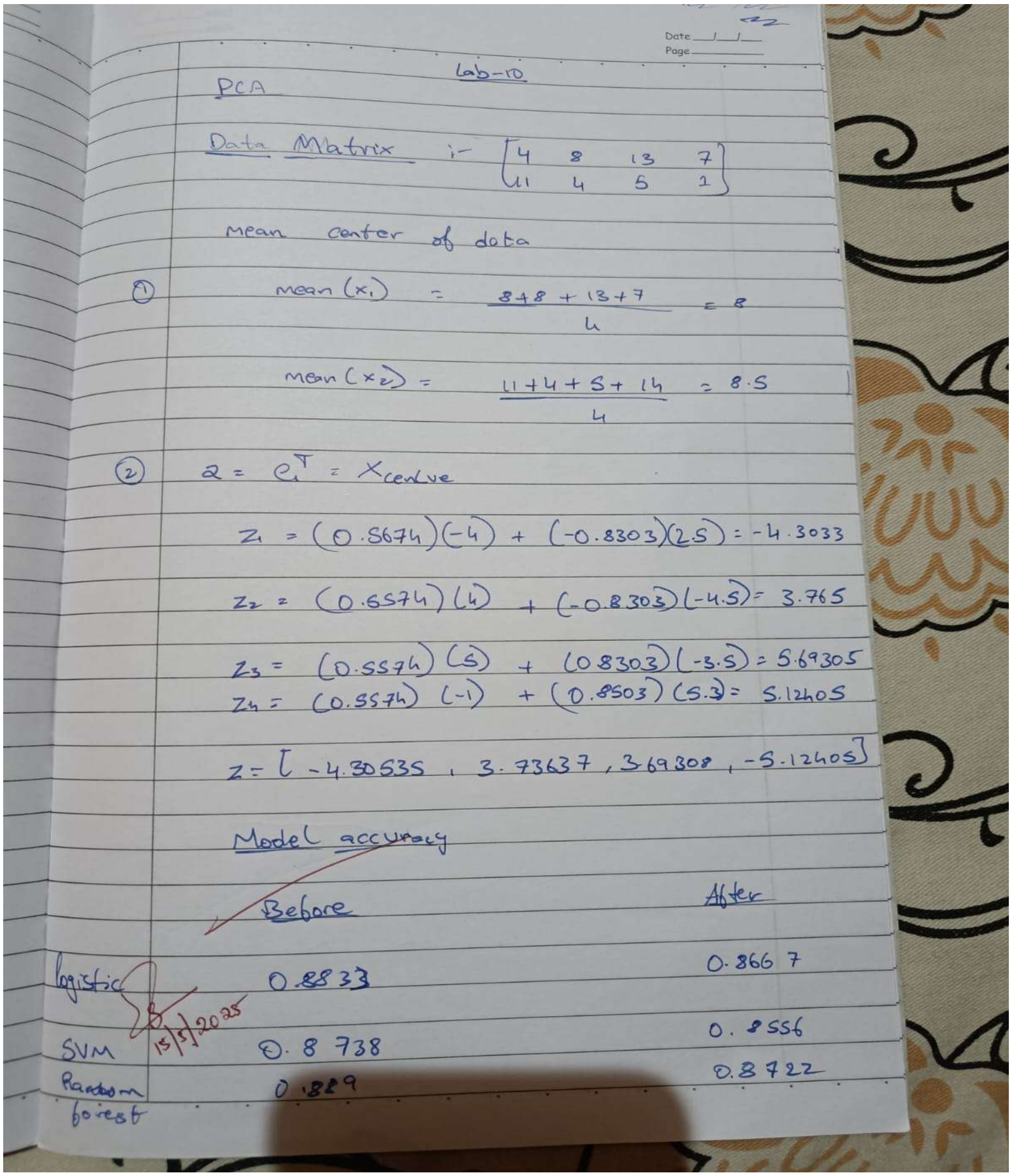
```
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans


# Load your iris dataset

df = pd.read_csv("iris.csv")


# Use only Petal Length and Petal Width

X = df[["petal_length", "petal_width"]]


# Scale the features

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

# Elbow Method to determine optimal k

inertia = []

k_range = range(1, 11)

for k in k_range:

    kmeans = KMeans(n_clusters=k, random_state=42)

    kmeans.fit(X_scaled)

    inertia.append(kmeans.inertia_)


# Plot Elbow Curve
```

```
plt.figure(figsize=(8, 5))
plt.plot(k_range, inertia, marker='o')
plt.title("Elbow Method for Optimal k")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia")
plt.grid(True)
plt.show()

# Final KMeans with optimal k (e.g., 3 from elbow)
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)

# Visualize Clusters
plt.figure(figsize=(8, 5))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=df['Cluster'], cmap='viridis', s=50)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            color='red', marker='X', label='Centroids')
plt.xlabel("Petal Length (scaled)")
plt.ylabel("Petal Width (scaled)")
plt.title(f"K-Means Clustering (k={optimal_k}) on Iris Petal Features")
plt.legend()
plt.grid(True)
plt.show()
```

Program 10

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot



Date __/__/__
Page ____

Lab-10

PCA

Data Matrix :- $\begin{bmatrix} 4 & 8 & 13 & 7 \\ 11 & 4 & 5 & 1 \end{bmatrix}$

mean center of data

① mean $(x_1) = \frac{8+8+13+7}{4} = 8$

mean $(x_2) = \frac{11+4+5+14}{4} = 8.5$

② $z = e_1^T = X_{centre}$

$z_1 = (0.5674)(-4) + (-0.8303)(2.5) = -4.3033$

$z_2 = (0.5574)(4) + (-0.8303)(-4.5) = 3.765$

$z_3 = (0.5574)(5) + (0.8303)(-3.5) = 5.69305$

$z_4 = (0.5574)(-1) + (0.8503)(5.3) = 5.12405$

$z = [-4.30535, 3.73637, 3.69308, -5.12405]$

Model accuracy

	Before	After
logistic	0.8833	0.866 7
SVM	0.8 738	0.8556
Random forest	0.889	0.8722

15/5/2025

Code:

```
import pandas as pd

from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

from sklearn.metrics import accuracy_score

from sklearn.pipeline import Pipeline


# Load dataset

data = pd.read_csv('heart.csv')


# Separate features and target

X = data.drop('HeartDisease', axis=1)

y = data['HeartDisease']


# Identify categorical and numerical columns

cat_cols = ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope']

num_cols = [col for col in X.columns if col not in cat_cols]


# Label encode binary categorical columns with two categories (e.g. Sex, ExerciseAngina)
```

```
label_enc_cols = ['Sex', 'ExerciseAngina']
le = LabelEncoder()
for col in label_enc_cols:
    X[col] = le.fit_transform(X[col])

# For other categorical columns with more than two categories, apply OneHotEncoding
onehot_cols = list(set(cat_cols) - set(label_enc_cols))

# Preprocessing pipeline: OneHotEncoding + scaling numerical features
preprocessor = ColumnTransformer(
    transformers=[
        ('onehot', OneHotEncoder(drop='first'), onehot_cols),
        ('scaler', StandardScaler(), num_cols)
    ],
    remainder='passthrough'  # To keep label encoded columns as is
)

# Split data into train/test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y)

# Helper function to train and evaluate a model
def train_evaluate_model(model, X_train, X_test, y_train, y_test):
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
```

```
                              ('classifier', model)])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    return acc, pipeline


# Train and evaluate SVM
svm = SVC(random_state=42)
svm_acc, svm_pipeline = train_evaluate_model(svm, X_train, X_test, y_train, y_test)


# Train and evaluate Logistic Regression
logreg = LogisticRegression(max_iter=1000, random_state=42)
logreg_acc, logreg_pipeline = train_evaluate_model(logreg, X_train, X_test, y_train, y_test)


# Train and evaluate Random Forest
rf = RandomForestClassifier(random_state=42)
rf_acc, rf_pipeline = train_evaluate_model(rf, X_train, X_test, y_train, y_test)


print(f"Accuracy Scores without PCA:\nSVM: {svm_acc:.4f}\nLogistic Regression:
{logreg_acc:.4f}\nRandom Forest: {rf_acc:.4f}")


# Now apply PCA for dimensionality reduction after preprocessing (scaling + encoding)
# We modify the pipeline to include PCA before classification


def train_evaluate_model_pca(model, X_train, X_test, y_train, y_test, n_components):
```

```
    pca = PCA(n_components=n_components, random_state=42)
    pipeline = Pipeline(steps=[
        ('preprocessor', preprocessor),
        ('pca', pca),
        ('classifier', model)
    ])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    return acc, pipeline


# Choose number of components to keep 95% variance or fixed number (e.g. 5)
# Here let's pick 5 components arbitrarily
n_components = 5


svm_pca_acc, _ = train_evaluate_model_pca(svm, X_train, X_test, y_train, y_test, n_components)
logreg_pca_acc, _ = train_evaluate_model_pca(logreg, X_train, X_test, y_train, y_test,
n_components)
rf_pca_acc, _ = train_evaluate_model_pca(rf, X_train, X_test, y_train, y_test, n_components)


print(f"\nAccuracy Scores with PCA (n_components={n_components}):")
print(f"SVM: {svm_pca_acc:.4f}")
print(f"Logistic Regression: {logreg_pca_acc:.4f}")
print(f"Random Forest: {rf_pca_acc:.4f}")
```