VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

Machine Learning (23CS6PCMAL)

Submitted by

Sakshi B R (1BM22CS233)

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 **Sakshi B R (1BM22CS233),** 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.

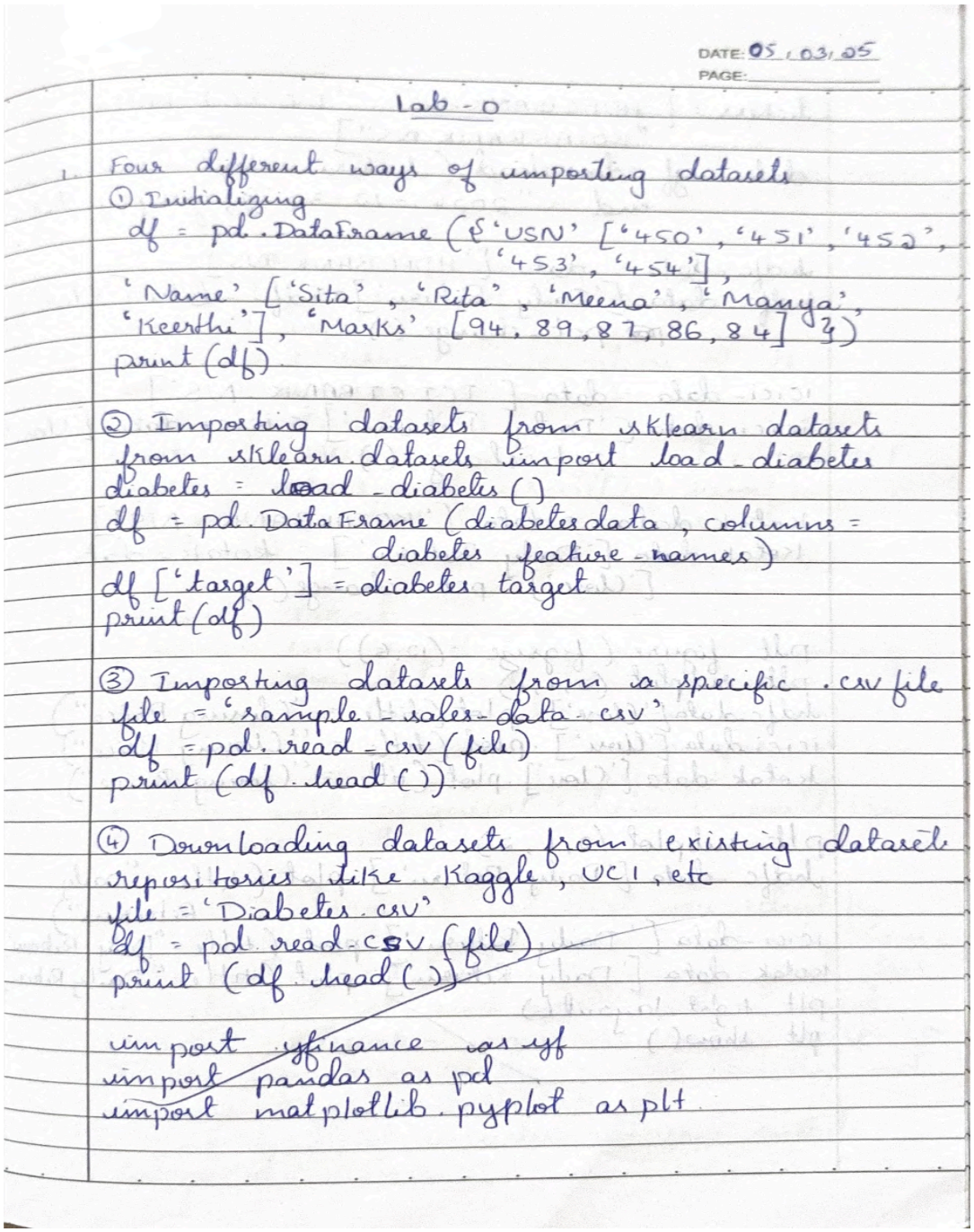
Lab Faculty Incharge Name: **Ms Saritha A N** Assistant Professor Department of CSE, BMSCE	 **Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

Index

Github Link: https://github.com/Sakshiibr/ML

Program 1

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



DATE: 05.03.25
PAGE:

Lab - 0

1. Four different ways of importing datasets

① Initializing

```
df = pd.DataFrame ({'USN' ['450', '451', '452',
                           '453', '454'],
'Name' ['Sita', 'Rita', 'Meena', 'Manya',
'Keerthi'], 'Marks' [94, 89, 87, 86, 84] })
print (df)
```

② Importing datasets from sklearn datasets

```
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
df = pd.DataFrame (diabetes.data, columns =
                   diabetes.feature_names)
df['target'] = diabetes.target
print(df)
```

③ Importing datasets from a specific .csv file

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

④ Downloading datasets from existing datasets repositories like Kaggle, UCI, etc

```
file = 'Diabetes.csv'
df = pd.read_csv(file)
print (df.head())
```

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

DATE: __/__/____
PAGE: _____

```
tickers = ["HDFCBANK.NS", "ICICIBANK",
           "KOTAKBANK.NS"]
data = yf.download(tickers, start = "2024-01-23",
       end = "2024-12-30", groupby='ticker')

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

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

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

plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
hdfc_data['Close'].plot(title="Closing Price")
icici_data['Close'].plot(title="Closing Price")
kotak_data['Close'].plot(title="Closing Price")

plt.subplot(2,1,2)
hdfc_data['Daily Return'].plot(title="Daily
                                Returns")
icici_data['Daily Return'].plot(title="Daily Return")
kotak_data['Daily Return'].plot(title="Daily Return")
plt.tight_layout()
plt.show()
```

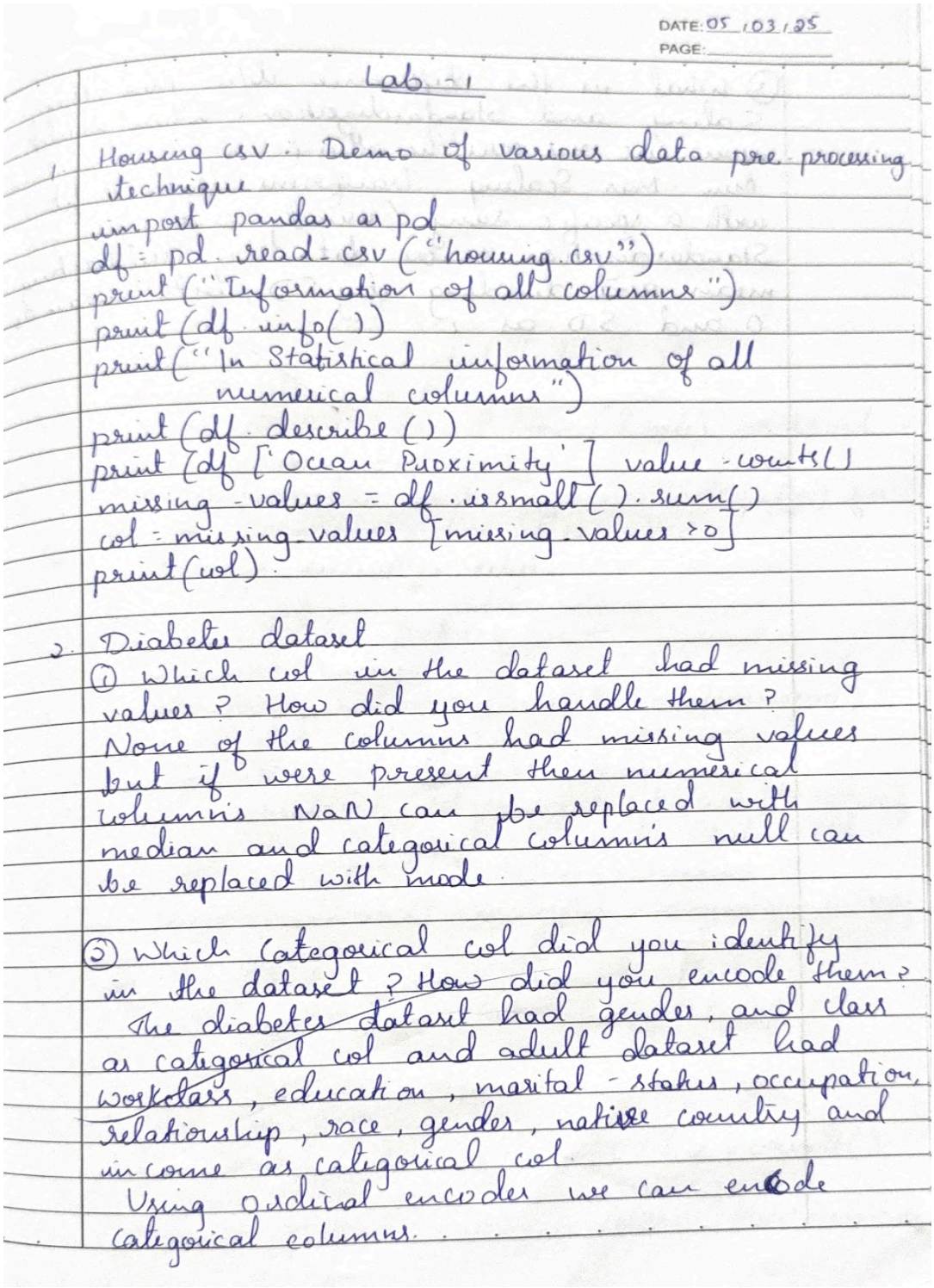
Code:

```
import pandas as pd
data={
   'USN':['1','2','3','4','5'],
   'Name':['Arun', 'Soudarya','neha', 'suraj', 'Prajju'],
   'marks':[50, 80, 45, 99, 87]
}
df=pd.DataFrame(data)
print(df)

from sklearn.datasets import load_diabetes
diabetes =load_diabetes()
df=pd.DataFrame(diabetes.data, columns=diabetes.feature_names )
df['target']=diabetes.target
print(df)


file_path='/sample_sales_data (1).csv'
df=pd.read_csv(file_path)
print("Sample data:")
print(df)


file='Dataset of Diabetes .csv'
df=pd.read_csv(file)
print("Sample data:")
print(df.head())

# Step 1: Import required libraries
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')

# Display the first 5 rows of the dataset
print("First 5 rows of the dataset:")
print(data.head())
```

```
# Calculate daily returns
hdfc=data['HDFCBANK.NS']
hdfc['Daily Return'] = hdfc['Close'].pct_change()
print("\nSummary statistics for HDFC:")
print(hdfc.describe())

icic=data['ICICIBANK.NS']
icic['Daily Return'] = icic['Close'].pct_change()
print("\nSummary statistics for ICICI:")
print(icic.describe())

kotak=data['KOTAKBANK.NS']
kotak['Daily Return'] = kotak['Close'].pct_change()
print("\nSummary statistics for Kotak:")
print(kotak.describe())


# Plot the closing price and daily returns
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
hdfc['Close'].plot(title=" Closing Price")
icic['Close'].plot(title=" Closing Price")
kotak['Close'].plot(title=" Closing Price")
plt.subplot(2, 1, 2)
hdfc['Daily Return'].plot(title="Daily Returns" )
icic['Daily Return'].plot(title=" Daily Returns")
kotak['Daily Return'].plot(title="Daily Returns")
plt.tight_layout()
plt.show()
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset



DATE: 05/03/25
PAGE:

Lab - 1

1. Housing csv - Demo of various data pre-processing technique

```
import pandas as pd
df = pd.read_csv("housing.csv")
print("Information of all columns")
print(df.info())
print("In Statistical information of all
      numerical columns")
print(df.describe())
print(df['Ocean Proximity'].value_counts())
missing_values = df.isnull().sum()
col = missing_values[missing_values>0]
print(col)
```

2. Diabetes dataset

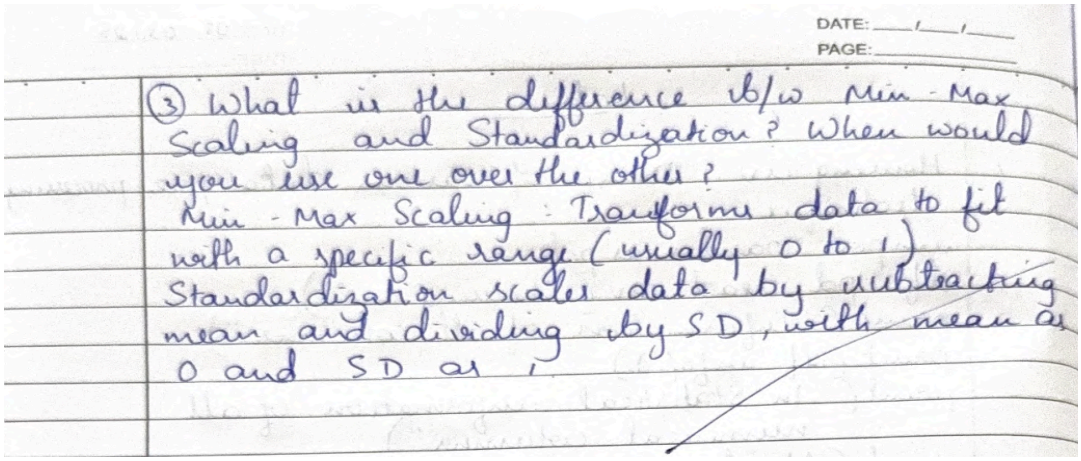
(1) Which col in the dataset had missing values? How did you handle them?

None of the columns had missing values but if were present then numerical columns NaN can be replaced with median and categorical columns null can be replaced with mode.

(2) Which categorical col did you identify in the dataset? How did you encode them?

The diabetes dataset had gender, and class as categorical col and adult dataset had workclass, education, marital-status, occupation, relationship, race, gender, native country and income as categorical col.

Using ordinal encoder we can encode categorical columns.



Code:

#Handling Missing Values, Handling categorical data, Handling Outliers

```
import pandas as pd
df=pd.read_csv('/content/Dataset of Diabetes .csv')
missing_values=df.isnull().sum()
print(missing_values[missing_values > 0])
```

 # Data Transformations: Min-max Scaler/Normalization , Standard Scaler

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder, StandardScaler
file_path = "Dataset of Diabetes .csv"
df = pd.read_csv(file_path)
df["Gender"] = df["Gender"].str.upper()
ordinal_encoder = OrdinalEncoder(categories=[["M", "F"]])
df["Gender_Encoded"] = ordinal_encoder.fit_transform(df[["Gender"]])
onehot_encoder = OneHotEncoder(sparse_output=False, drop="first")
encoded_class = onehot_encoder.fit_transform(df[["CLASS"]])
encoded_class_df = pd.DataFrame(encoded_class,
columns=onehot_encoder.get_feature_names_out(["CLASS"]))
df_encoded = pd.concat([df, encoded_class_df], axis=1)
df_encoded.drop(["Gender", "CLASS"], axis=1, inplace=True)
num_cols = ["AGE", "Urea", "Cr", "HbA1c", "Chol", "TG", "HDL", "LDL", "VLDL", "BMI"]
scaler = StandardScaler()
df_encoded[num_cols] = scaler.fit_transform(df_encoded[num_cols])
print(df_encoded.head())
df_encoded.to_csv("cleaned_diabetes_dataset.csv", index=False)
df_encoded_copy1=df_encoded
df_encoded_copy2=df_encoded
df_encoded_copy3=df_encoded
```

```
Q1 = df_encoded_copy1['BMI'].quantile(0.25)
Q3 = df_encoded_copy1['BMI'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
df_encoded_copy1['BMI'] = np.where(df_encoded_copy1['BMI'] > upper_bound, upper_bound,
                    np.where(df_encoded_copy1['BMI'] < lower_bound, lower_bound,
df_encoded_copy1['BMI']))

print(df_encoded_copy1.head())
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

DATE: 19/3/25
PAGE:

Lab - 3

Linear Regression

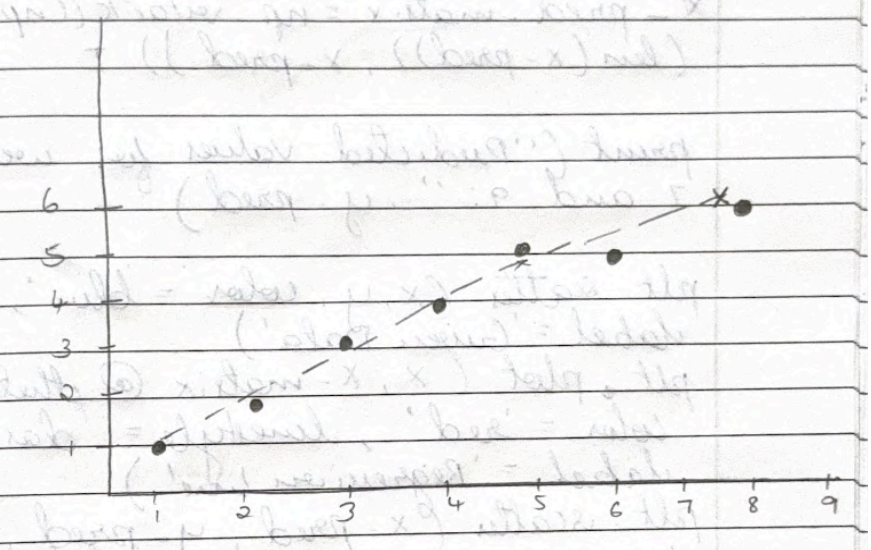
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Given dataset
x = np.array([[1,2,3,4,5,6,7,8,9]]).reshape(-1,1)
y = np.array([1.2, 1.8, 2.6, 3.2, 3.8, 4.4, 5.0, 5.4, 6.0])

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

# Predict for 8 th and 9 th week

x_pred = np.array([7, 9]).reshape(-1,1)
y_pred = model.predict(x_pred)

# Print predictions
print("Predicted values for week 7 and 9:",
y_pred)

# Visualization

plt.scatter(x, y, color = 'blue', label = 'Given
plt.plot(x, model.predict(x), color = 'red',
linestyle = 'dashed', label = 'Regression
line')
plt.scatter(x_pred, y_pred, color =
```

DATE: ___/___/____
PAGE: _____

```
'green', marker = 'o', label = 'Predicted Births')
plt.xlabel('Weeks')
plt.ylabel('Values')
plt.legend()
plt.show
```

Output

Predicted values for weeks 7 and 9:
[4.911111 6.111111]



→ Linear Regression in Matrix Form

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

# Given dataset
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
y = np.array([1.2, 1.8, 2.6, 3.2, 3.8, 4.4,
    5.0, 5.4, 6.0])
```

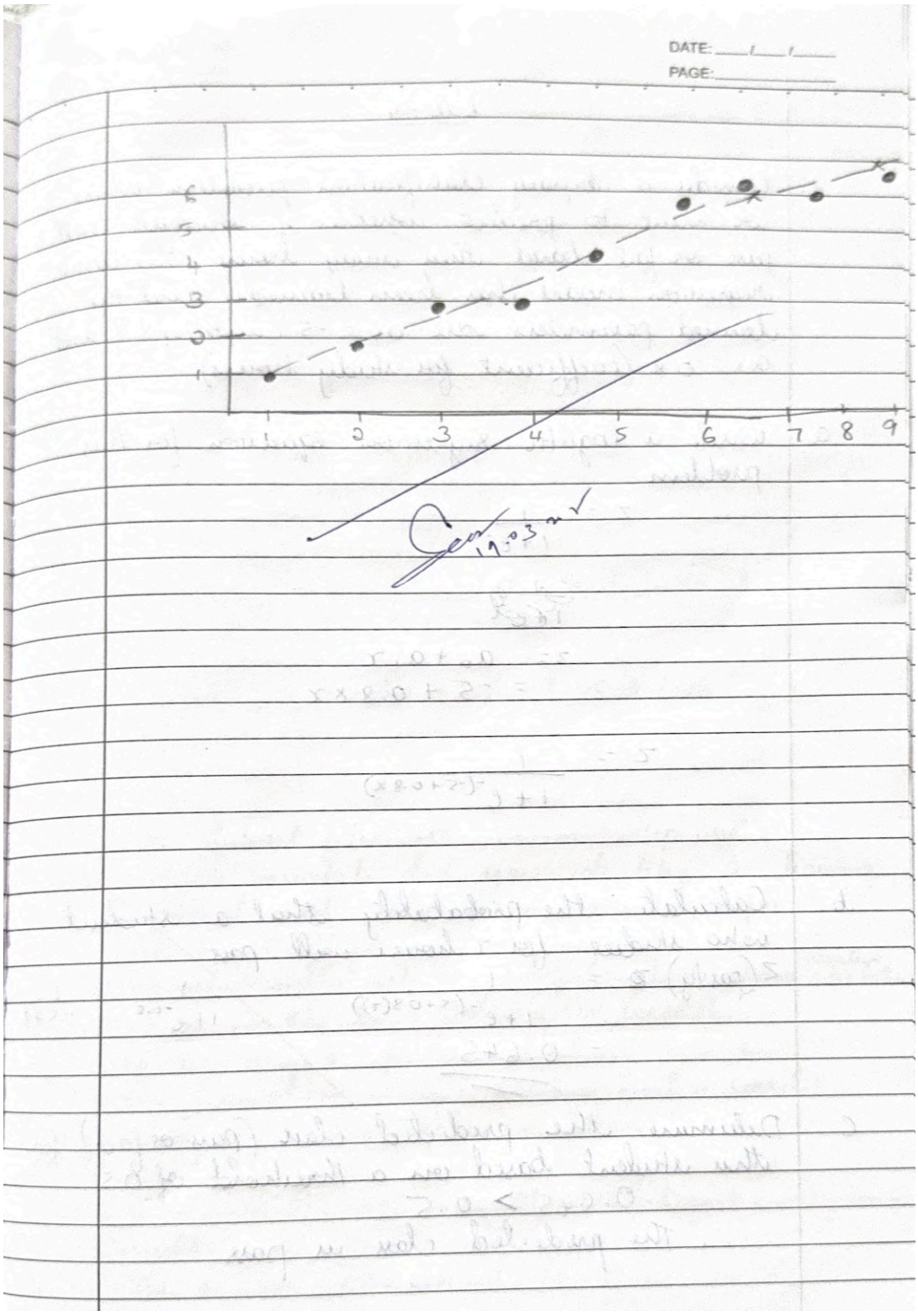
DATE: __/__/____
PAGE: _____

matrix approach for Linear Regression

```
X_matrix = np.vstack((np.ones(len(X)),
    X )).T

theta = np.linalg.inv (X_matrix.T @ X_matrix
@ y

# Predict for 7th and 9th week

X_pred = np.array ([7,9])
X_pred_matrix = np.vstack ((np.ones
(len(X_pred)), X_pred )).T

print ("Predicted values for weeks
7 and 9:", y_pred)

plt.scatter (X, y, color = 'blue',
label = 'Given Data')
plt.plot ( X, X_matrix @ theta,
color = 'red', linestyle = 'dashed',
label = 'Regression Line')
plt.scatter (X_pred, y_pred, color =
'green' marker = 'o', label =
'Predicted Points')
plt.xlabel ('Week')
plt.ylabel ('Values')
plt.legend()
plt.show
```

Output:

Predicted values for weeks 7 and 9: [4.91111111 6.11111111]


DATE: / /
PAGE:
6
5
4
3
2
1
1
2
3
4
5
6
7
8
9
19.03 m ✓

Code:

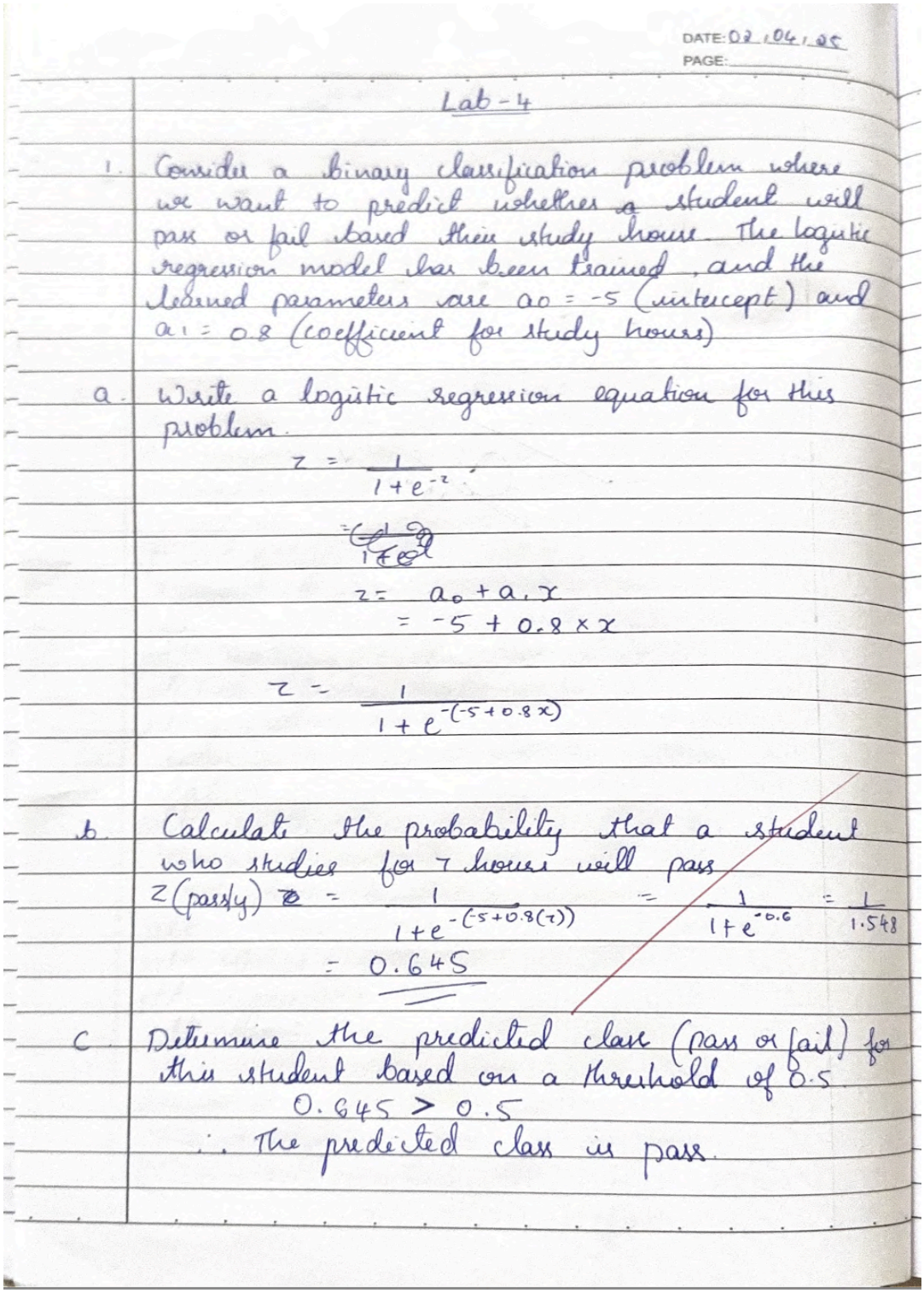
```
import pandas as pd
import matplotlib.pyplot as plt

# Data
x = [1, 2, 3, 4, 5]
y = [12, 18, 22, 28, 35]
df = pd.DataFrame({'x': x, 'y': y})

# Calculate required values
xy = []
x2 = []
for i, j in zip(x, y):
    xy.append(i * j)
for i in x:
    x2.append(i ** 2)

x_mean = sum(x) / len(x)
y_mean = sum(y) / len(y)
xy_mean = sum(xy) / len(xy)
x2_mean = sum(x2) / len(x2)

# Calculate coefficients a0 and a1
a1 = (xy_mean - x_mean * y_mean) / (x2_mean - x_mean ** 2)
a0 = y_mean - a1 * x_mean

# Output the coefficients
print(f"a0 = {a0}, a1 = {a1}")

# Predict y for a given x value
x_input = int(input("Enter the value of x: "))
y_pred = a0 + a1 * x_input
print(f"Predicted y for x = {x_input} is: {y_pred}")

# Plotting
plt.scatter(x, y, color='blue', label='Data Points') # Scatter plot for the data points plt.plot(x,
[a0 + a1 * xi for xi in x], color='red', label='Regression Line') # Regression line plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear Regression')
plt.legend()
plt.grid(True)
plt.show()
```

Code:

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

def matrix_operations(x, y, n):
    ax = np.ones((n, 2))
    ax[:, 1] = x
    ay = np.array(y).reshape(-1, 1)

    xt = ax.T
    A = np.dot(xt, ax)
    det_A = np.linalg.det(A)

    if det_A != 0:
        A_inv = np.linalg.inv(A)
        a = np.dot(A_inv, np.dot(xt, ay))
        return a.flatten() # Returns [a0, a1]
    else:
        return "Inverse doesn't exist"

# Input
n = int(input("Enter the number of elements: "))
x = [int(input(f"Enter the value of x[{i+1}]: ")) for i in range(n)]
y = [int(input(f"Enter the value of y[{i+1}]: ")) for i in range(n)]

# Perform matrix operation
result = matrix_operations(x, y, n)
if isinstance(result, str):
    print(result)
else:
    a0, a1 = result
    print(f"Slope (a1) = {a1}, Intercept (a0) = {a0}")

    # Plotting
    plt.scatter(x, y, color='blue', label='Data Points')
    plt.plot(x, a0 + a1 * np.array(x), color='red', label='Regression Line')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Linear Regression')
    plt.legend()
    plt.grid(True
    ) plt.show()
```

Program 4

Build Logistic Regression Model for a given dataset



DATE: 02/04/25
PAGE:

Lab - 4

1. Consider a binary classification problem where we want to predict whether a student will pass or fail based their study hours. The logistic regression model has been trained, and the learned parameters are $a_0 = -5$ (intercept) and $a_1 = 0.8$ (coefficient for study hours)

a. Write a logistic regression equation for this problem.

$$z = \frac{1}{1+e^{-z}}$$

$$z = a_0 + a_1 x$$
$$= -5 + 0.8 \times x$$

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

b. Calculate the probability that a student who studies for 7 hours will pass

$$z(\text{passly}) = \frac{1}{1+e^{-(-5+0.8(7))}} = \frac{1}{1+e^{-0.6}} = \frac{1}{1.548}$$
$$= 0.645$$

c. Determine the predicted class (pass or fail) for this student based on a threshold of 0.5.

$0.645 > 0.5$

∴ The predicted class is pass.

DATE: __/__/____
PAGE: _____

2 Consider $z = [2, 1, 0]$ for three classes Apply SoftMax function to find the probability values of three classes.

$$\text{Softmax}(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{k} e^{z_j}} \qquad \text{11.1073}$$

$$z_2 = \frac{e^2}{e^2+e^1+e^0} = 0.665$$

$$z = \frac{e^1}{e^2+e^1+e^0} = 0.2447$$

$$z = \frac{e^0}{e^2+e^1+e^0} = 0.09003$$

66.5%, 24.47%, 9.003%.

→ For dataset file "HR - comma - sep.csv"

(i) Which variable did you identify as having a direct and clear impact on employee retention? Why?

Variables such as: satisfaction_level: strong negative correlation
average_monthly_hours: strong +ve correlation
number_project: positive correlation
time_spend_company: strong positive correlation
Salary: strong negative correlation

(ii) What was the accuracy of your logistic regression model?

The accuracy was around (0.79) = 79%

DATE: __/__/____
PAGE: ____

2. For Zoo Dataset

(i) Did you perform any data processing steps? If yes, what were they, and why were they necessary?

Yes, following steps are performed.

① Dropped the animal_name column - it was a non-numeric identifier that didn't contribute to classification.

② Checked for missing values - to ensure data integrity - No missing values were found.

③ Standardized the dataset - using StandardScaler() to normalize the numerical features, improving the logistic regression model's performance.

(ii) Were there any missing or inconsistent values in the dataset? How did you handle them?

No missing / inconsistent value were found. If there had been missing values strategies like mean/median replacement or removing rows with excessive missing values.

(iii) What does the confusion matrix tell you about the performance of your model.

The confusion matrix showed perfect classification (100% accuracy), meaning that all test samples were classified correctly into their respective classes.

(iv) Which class types were most frequently misclassified? Why do you think this happened?

No misclassifications.

If misclassifications had occured reasons could be : ① Similar feature values across diff.

DATE: __/__/____
PAGE: ____

classes ② Limited training samples for certain class type.

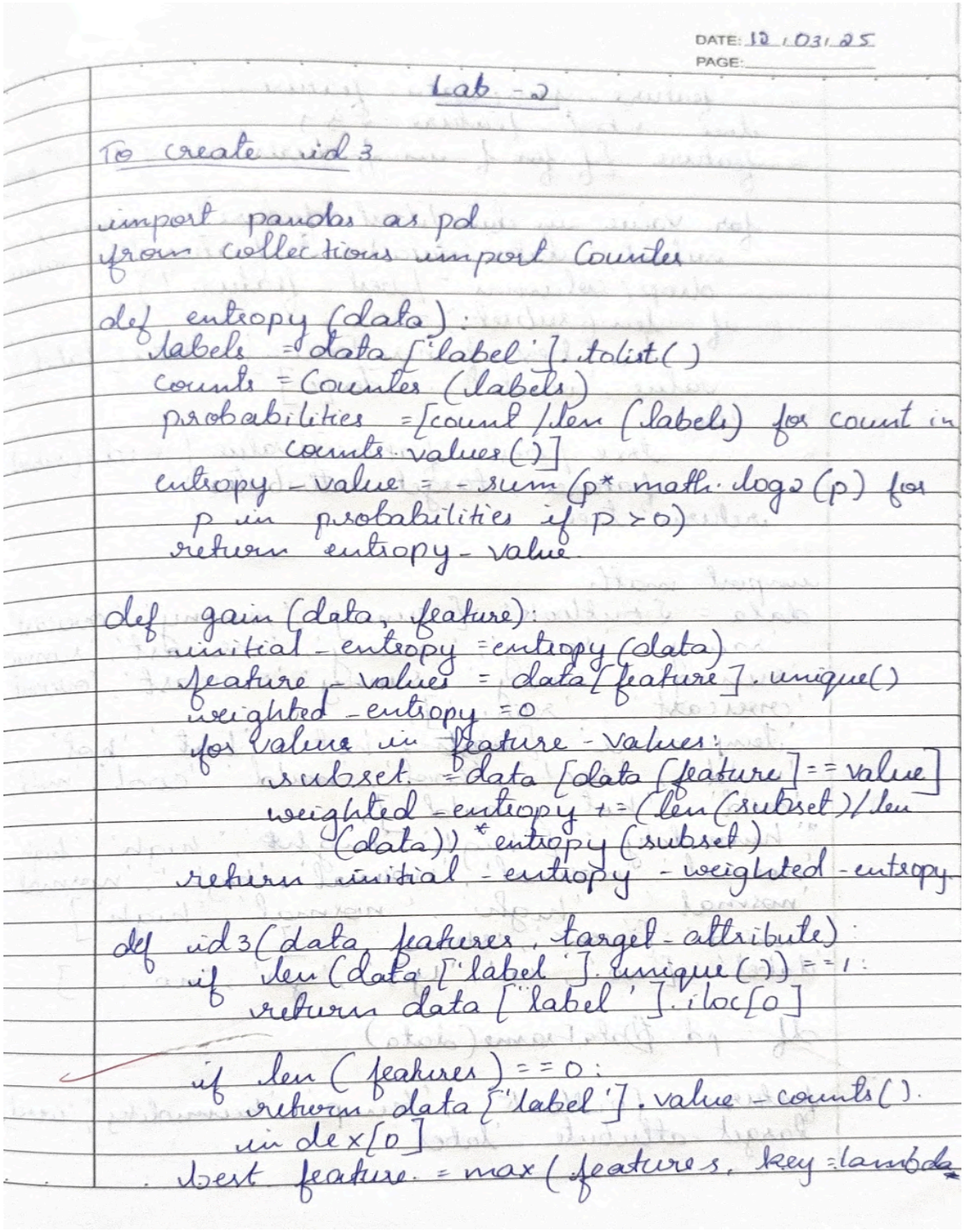
Code:

```
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.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Load dataset
file_path = "/content/zoo-data.csv"
df = pd.read_csv(file_path)
# Drop the animal_name column as it is not a feature
df = df.drop(columns=["animal_name"])
# Define features and target
X = df.drop(columns=["class_type"])
y = df["class_type"]
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train multinomial logistic regression model
model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=500)
model.fit(X_train, y_train)
# Make predictions
y_pred = model.predict(X_test)
# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred))
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
labels = np.unique(y)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
import pandas as pd
from matplotlib import pyplot as plt
import math
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Load HR dataset
```

```
df = pd.read_csv("/content/HR_comma_sep.csv")
df.head()

plt.scatter(df.satisfaction_level, df.left, marker='+', color='red')

# Splitting dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df[['satisfaction_level']], df.left, train_size=0.9,
random_state=10)
X_train.shape
X_test

# Training logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)
X_test
y_test
y_predicted = model.predict(X_test)
y_predicted
model.score(X_test, y_test)
model.predict_proba(X_test)
y_predicted = model.predict([[0.5]])

# model.coef_ indicates value of m in y=m*x + b equation
model.coef_

# model.intercept_ indicates value of b in y=m*x + b equation
model.intercept_

# Define sigmoid function and do the math manually
def sigmoid(x):
   return 1 / (1 + math.exp(-x))

def prediction_function(satisfaction_level):
   z = model.coef_[0][0] * satisfaction_level + model.intercept_[0]
   y = sigmoid(z)
   return y
satisfaction_level = 0.35
probability =
prediction_function(satisfaction_level)
print(f"Probability of leaving: {probability:.2f}")
if probability >= 0.5:
   print("The employee will leave.")
else:
   print("The employee will stay.")
```

Program 5

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



DATE: 12/03/25
PAGE:

Lab - 2

To create id3

```
import pandas as pd
from collections import Counter

def entropy(data):
    labels = data['label'].tolist()
    counts = Counter(labels)
    probabilities = [count/len(labels) for count in
        counts.values()]
    entropy_value = -sum(p* math.log2(p) for
        p in probabilities if p>0)
    return entropy_value

def gain(data, feature):
    initial_entropy = entropy(data)
    feature_values = data[feature].unique()
    weighted_entropy = 0
    for value in feature_values:
        subset = data[data[feature] == value]
        weighted_entropy += (len(subset)/len
            (data)) * entropy(subset)
    return initial_entropy - weighted_entropy

def id3(data, features, target_attribute)
    if len(data['label'].unique()) == 1:
        return data['label'].iloc[0]

    if len(features) == 0:
        return data['label'].value_counts().
            index[0]
    best_feature = max(features, key=lambda
```

DATE: __/__/____
PAGE: _____

```
    feature : gain(data, feature))
tree = {best_feature : {}}
features = [f for f in features if f != best_feature]

for value in data[best_feature].unique():
    subset = data[data[best_feature] == value]
    .drop(columns = [best_feature])
    if len(subset) == 0:
        tree[best_feature][value] = data['label'].
    value_counts().index[0]
    else:
        tree[best_feature][value] = id3(subset,
        features, target_attribute)
return tree
```

```
import math
data = {'outlook': ['sunny', 'sunny', 'overcast',
'rainy', 'rainy', 'rainy', 'overcast', 'sunny',
'sunny', 'rainy', 'sunny', 'overcast', 'overcast',
'overcast', 'rainy'],
'temperature': ['hot', 'hot', 'hot', 'hot',
'mild', 'cool', 'cool', 'mild', 'cool', 'mild',
'mild', 'hot', 'mild'],
'humidity': ['high', 'high', 'high', 'high',
'normal', 'normal', 'normal', 'high', 'normal',
'normal', 'high', 'normal', 'high']
'wind': ['weak', 'strong', ...]
'label': ['no', 'no', 'yes', 'yes', 'no', ...]

df = pd.DataFrame(data)

features = ['outlook', 'temp', 'humidity', 'wind']
target_attribute = 'label'
```

DATE: / /
PAGE:

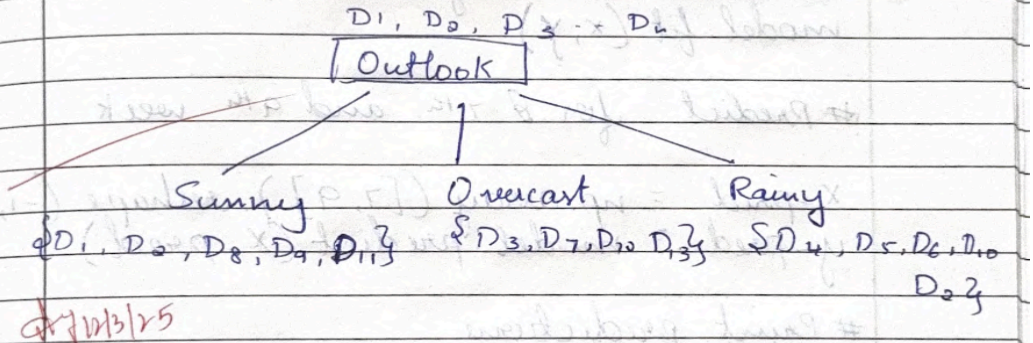
decision_tree_id3(df, features, target_attribute)

decision_tree

Output:

{'outlook': {'sunny': {'humidity': {'high': 'no', 'normal': 'yes'}},
'overcast': 'yes'
'rainy': {'wind': {'weak': 'yes', 'strong': 'no'}}}}

Decision Tree



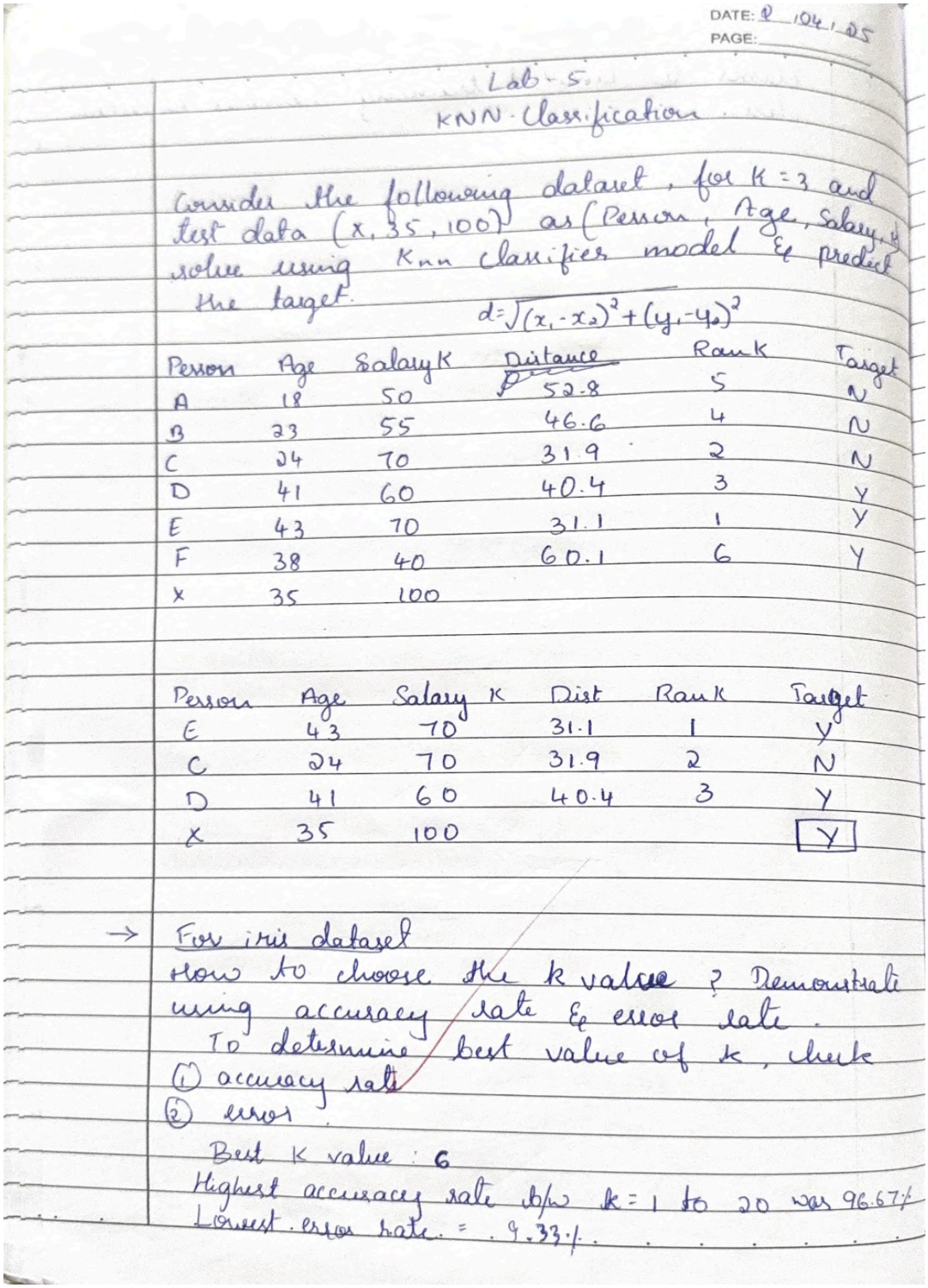
7/12/23/25

Code:

```
import pandas as pd
data=pd.read_csv('tennis.csv')
data
from sklearn.preprocessing import LabelEncoder
outlook=LabelEncoder()
temp=LabelEncoder()
humidity=LabelEncoder()
wind=LabelEncoder()
play=LabelEncoder()
data['outlook']=outlook.fit_transform(data['outlook'])
data['temp']=outlook.fit_transform(data['temp'])
data['humidity']=outlook.fit_transform(data['humidity'])
data['windy']=outlook.fit_transform(data['windy'])
data['play']=outlook.fit_transform(data['play'])
data
features_cols=['outlook','temp','humidity','windy']
x=data[features_cols]
y=data.play
print(x)
print()
print(y)
from sklearn.model_selection import train_test_split
x_train, x_test, y_train,y_test=train_test_split(x,y,test_size=0.2)
from sklearn.tree import DecisionTreeClassifier
classifier=DecisionTreeClassifier(criterion='entropy'
) classifier.fit(x_train,y_train)
classifier.predict(x_test)
x_test
y_test
classifier.score(x_test,y_test)
from sklearn import tree
tree.plot_tree(classifier)
clf = DecisionTreeClassifier(criterion='entropy') # Using entropy to calculate information gain
clf.fit(x, y)
importances = clf.feature_importances_
# Create a DataFrame to see the feature importances
feature_importance_df = pd.DataFrame({
    'Feature': x.columns,
    'Information Gain': importances
})
# Sort by importance
feature_importance_df = feature_importance_df.sort_values(by='Information Gain', ascending=False)
print(feature_importance_df)
```

Program 6

Build KNN Classification model for a given dataset



DATE: 2/04/25
PAGE:

Lab - 5

KNN - Classification

Consider the following dataset, for K = 3 and test data (x, 35, 100) as (Person, Age, Salary) solve using Knn classifier model & predict the target.

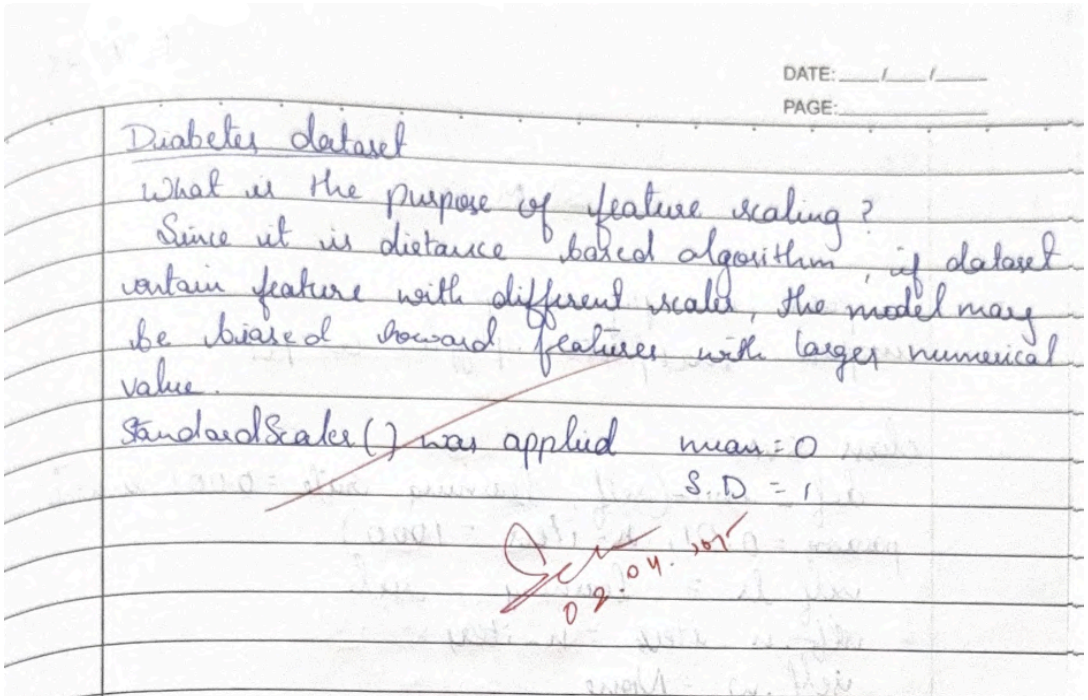
$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Person	Age	SalaryK	Distance	Rank	Target
A	18	50	52.8	5	N
B	23	55	46.6	4	N
C	24	70	31.9	2	N
D	41	60	40.4	3	Y
E	43	70	31.1	1	Y
F	38	40	60.1	6	Y
X	35	100			

Person	Age	Salary K	Dist	Rank	Target
E	43	70	31.1	1	Y
C	24	70	31.9	2	N
D	41	60	40.4	3	Y
X	35	100			Y

→ For iris dataset
How to choose the k value ? Demonstrate using accuracy rate & error rate.
To determine best value of k, check
① accuracy rate
② error

Best K value : 6
Highest accuracy rate b/w k = 1 to 20 was 96.67%
Lowest error rate = 3.33%



Code:

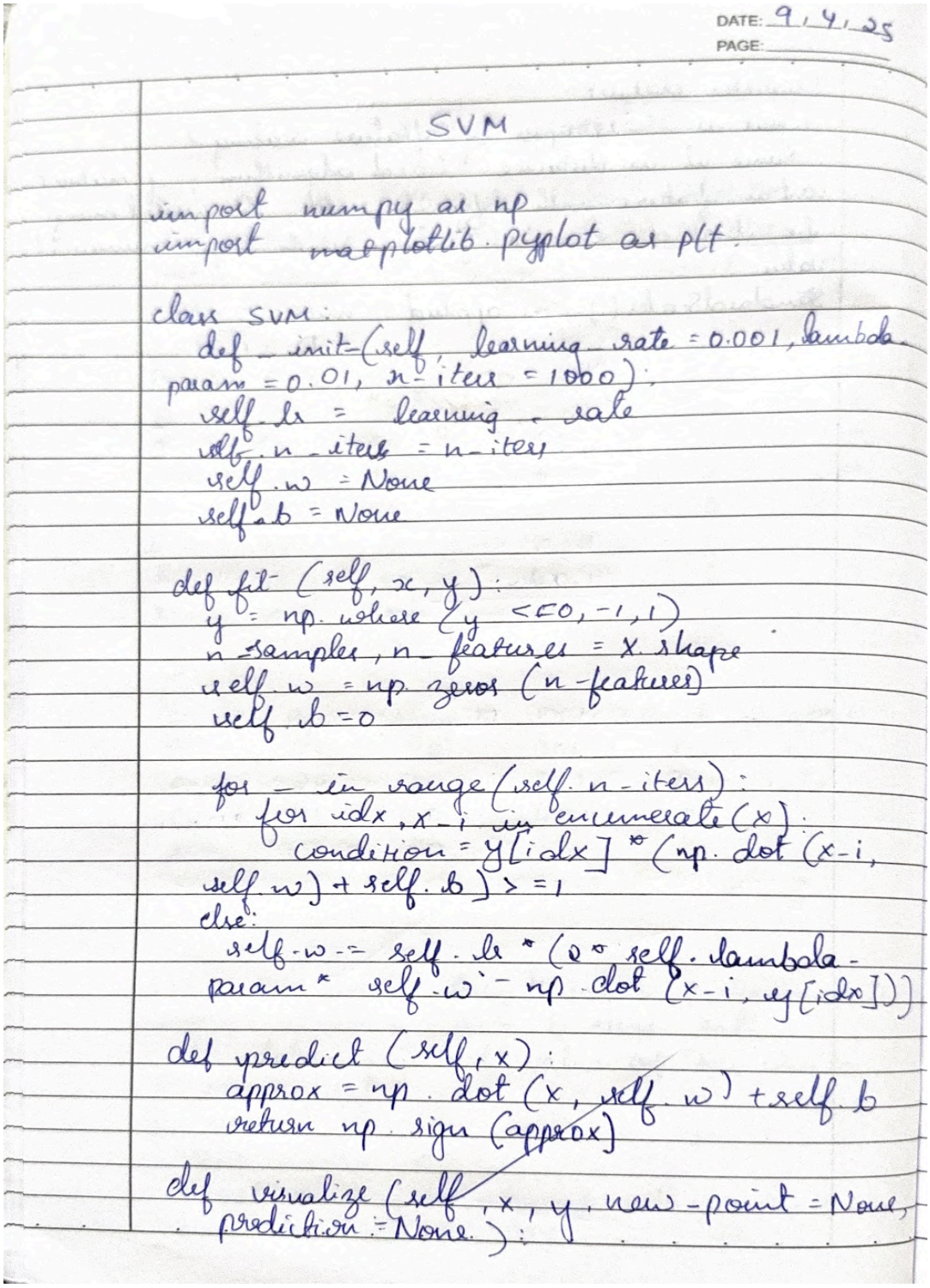
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matri
# Load dataset
file_path = "/content/diabetes.csv"
df = pd.read_csv(file_path)
# Define features and target
X = df.drop(columns=["Outcome"])
y = df["Outcome"]
# Split data into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Feature scaling (Standardization)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Initialize and train KNN classifier with k=5
k = 5
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
```

```
# Make predictions
y_pred = knn.predict(X_test)
# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred))
conf_matrix = confusion_matrix(y_test, y_pred)
labels = ["Non-Diabetic", "Diabetic"]
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
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 StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_repor
df = pd.read_csv('/content/heart.csv')
# Define features and target
X = df.drop(columns=['target']) # Assuming 'target' is the classification column
y = df['target']
# Split data
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)
# Find the best K value
k_values = range(1, 21)
accuracy_scores = []
for k in k_values:
    model =
    KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)accuracy_scores.append(accuracy_score(y_test, y_pred))
best_k = k_values[np.argmax(accuracy_scores)]
print(f'Best K value: {best_k}')
# Train model with best K
best_model = KNeighborsClassifier(n_neighbors=best_k)
```

```
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)
# Evaluate model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy with best K ({best_k}): {accuracy:.4f}')
print("Classification Report:")
print(classification_report(y_test, y_pred)
# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title(f'Confusion Matrix - KNN (K={best_k})')
plt.show()
# Plot K values vs. Accuracy
plt.plot(k_values, accuracy_scores, marker='o')
plt.xlabel('K Value')
plt.ylabel('Accuracy')
plt.title('K Value vs Accuracy')
plt.show()
```

Program 7

Build Support vector machine model for a given dataset



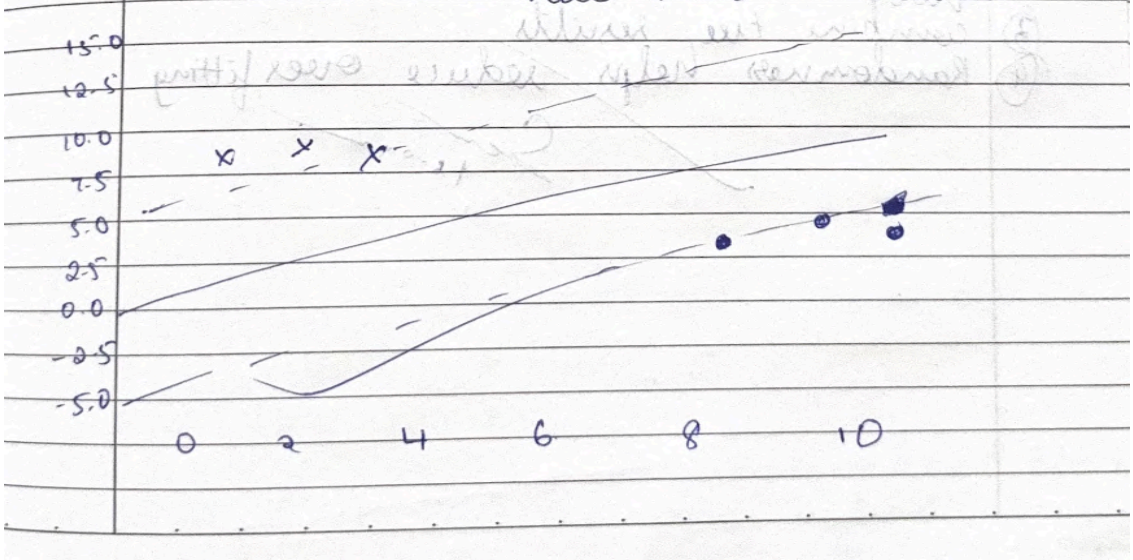
DATE: 9, 4, 25
PAGE:

SVM

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

class SVM:
    def __init__(self, learning_rate = 0.001, lambda_
param = 0.01, n_iters = 1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.w = None
        self.b = None

    def fit (self, x, y):
        y = np.where (y <=0, -1, 1)
        n_samples, n_features = x.shape
        self.w = np.zeros (n_features)
        self.b = 0

        for _ in range (self.n_iters):
            for idx, x_i in enumerate (x):
                condition = y[idx] * (np.dot (x_i,
self.w) + self.b ) >= 1
            else:
                self.w -= self.lr * (2 * self.lambda_
param * self.w - np.dot (x_i, y[idx]))

    def predict (self, x):
        approx = np.dot (x, self.w) + self.b
        return np.sign (approx)

    def visualize (self, x, y, new_point = None,
prediction = None ):
```

```
def get_hyperplane(x, w, b, offset):
    return (-w[0] * x + b + offset) / w[1]

fig = plt.figure()
ax = fig.add_subplot(1,1,1)

ax.legend()
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("SVM with New Point Prediction")
plt.grid(True)
plt.show

sv.visualize(X, y, new_point = new_point
  [0], prediction = prediction)

print(f"New point {new_point[0]} is
classified as: {'Class 1' if prediction == 1
else 'Class 0'}")
```

O/P



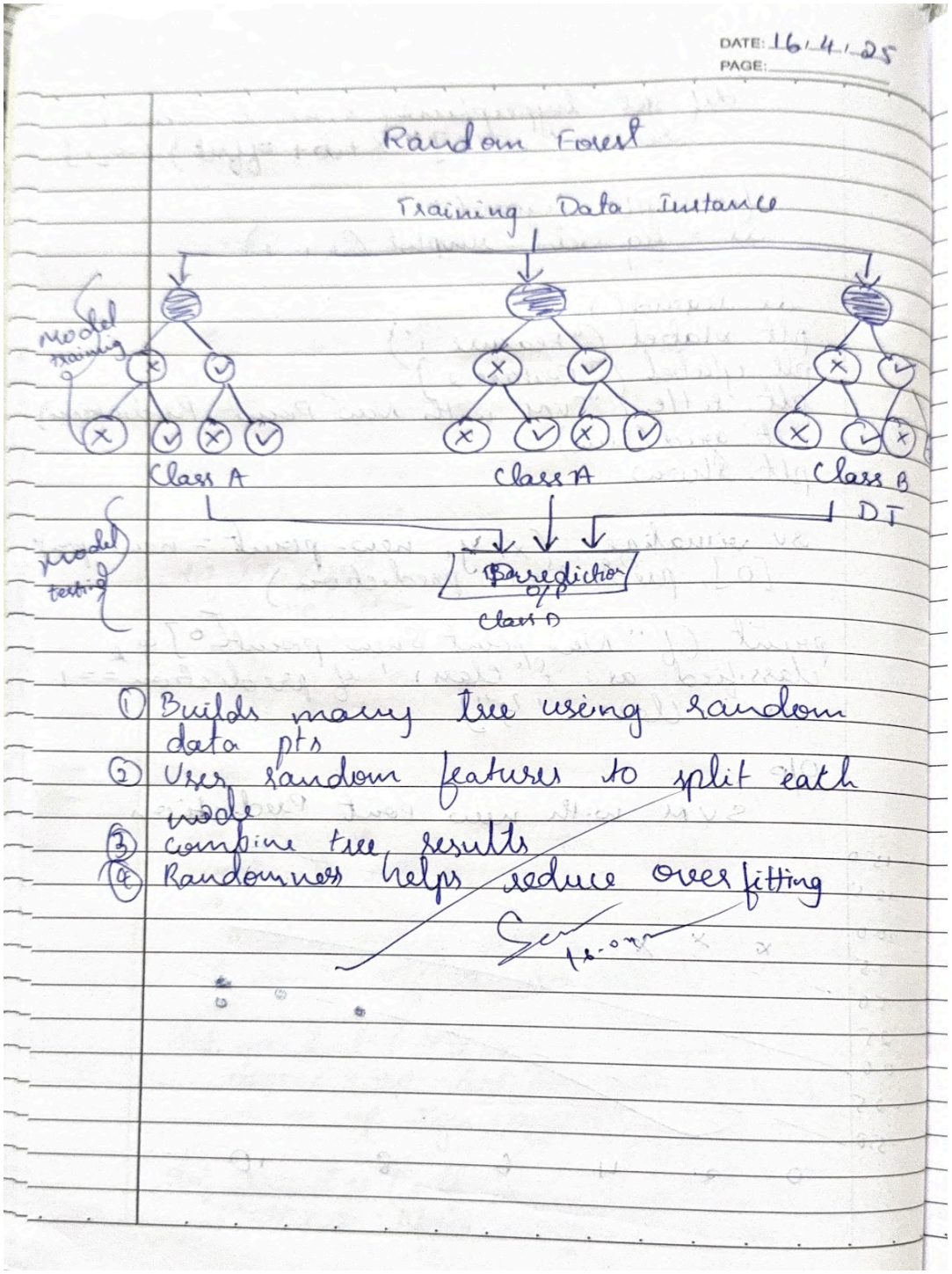
Code:

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

class SVM:
    def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000): self.lr =
        learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.w = None
        self.b = None

    def fit(self, X, y):
        y = np.where(y <= 0, -1, 1) # Convert labels to -1 and 1
        n_samples, n_features = X.shape
        self.w = np.zeros(n_features)
        self.b = 0
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                condition = y[idx] * (np.dot(x_i, self.w) + self.b) >= 1
                if condition:
                    self.w -= self.lr * (2 * self.lambda_param * self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y[idx]))
                    self.b += self.lr * y[idx]
    def predict(self, X):
        approx = np.dot(X, self.w) + self.b
        return np.sign(approx)

def visualize(self, X, y, new_point=None, prediction=None):
        def get_hyperplane(x, w, b, offset):
            return (-w[0] * x + b + offset) / w[1]
        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)
        for i, sample in enumerate(X):
            if y[i] == 1:
                plt.scatter(sample[0], sample[1], marker='o', color='blue', label='Class +1' if i == 0 else "")
            else:
                plt.scatter(sample[0], sample[1], marker='x', color='red', label='Class -1' if i == 0 else "")

        # Plot decision boundary
        x0 = np.linspace(np.min(X[:, 0])-1, np.max(X[:, 0])+1, 100)
        x1 = get_hyperplane(x0, self.w, self.b, 0)
        x1_m = get_hyperplane(x0, self.w, self.b, -1)
```

```
        x1_p = get_hyperplane(x0, self.w, self.b, 1)
        ax.plot(x0, x1, 'k-', label='Decision Boundary')
        ax.plot(x0, x1_m, 'k--', label='Margins')
        ax.plot(x0, x1_p, 'k--')

        if new_point is not None:
            color = 'green' if prediction == 1 else 'orange'
            label = f'New Point: Class {"1" if prediction == 1 else "0"}'
            plt.scatter(new_point[0], new_point[1], c=color, s=100, edgecolors='black', label=label, marker='*')
        ax.legend()
        plt.xlabel("Feature 1")
        plt.ylabel("Feature 2")
        plt.title("SVM with New Point Prediction")
        plt.grid(True)
        plt.show()

if _name____== "_main_":
    # Training data
    X = np.array([
        [1, 7],
        [2, 8],
        [3, 8],
        [8, 1],
        [9, 2],
        [10, 2]
    ])
    y = np.array([0, 0, 0, 1, 1, 1]) # 0 -> -1, 1 -> +1

    # New point to classify
    new_point = np.array([[5, 5]])

    # Train and predict
    svm = SVM()
    svm.fit(X, y)
    prediction = svm.predict(new_point)[0]

    # Visualize
    svm.visualize(X, y, new_point=new_point[0], prediction=prediction)

    # Print prediction
    print(f"New point {new_point[0]} classified as: {'Class 1' if prediction == 1 else 'Class 0'}")
```

Program 8

Implement Random forest ensemble method on a given dataset

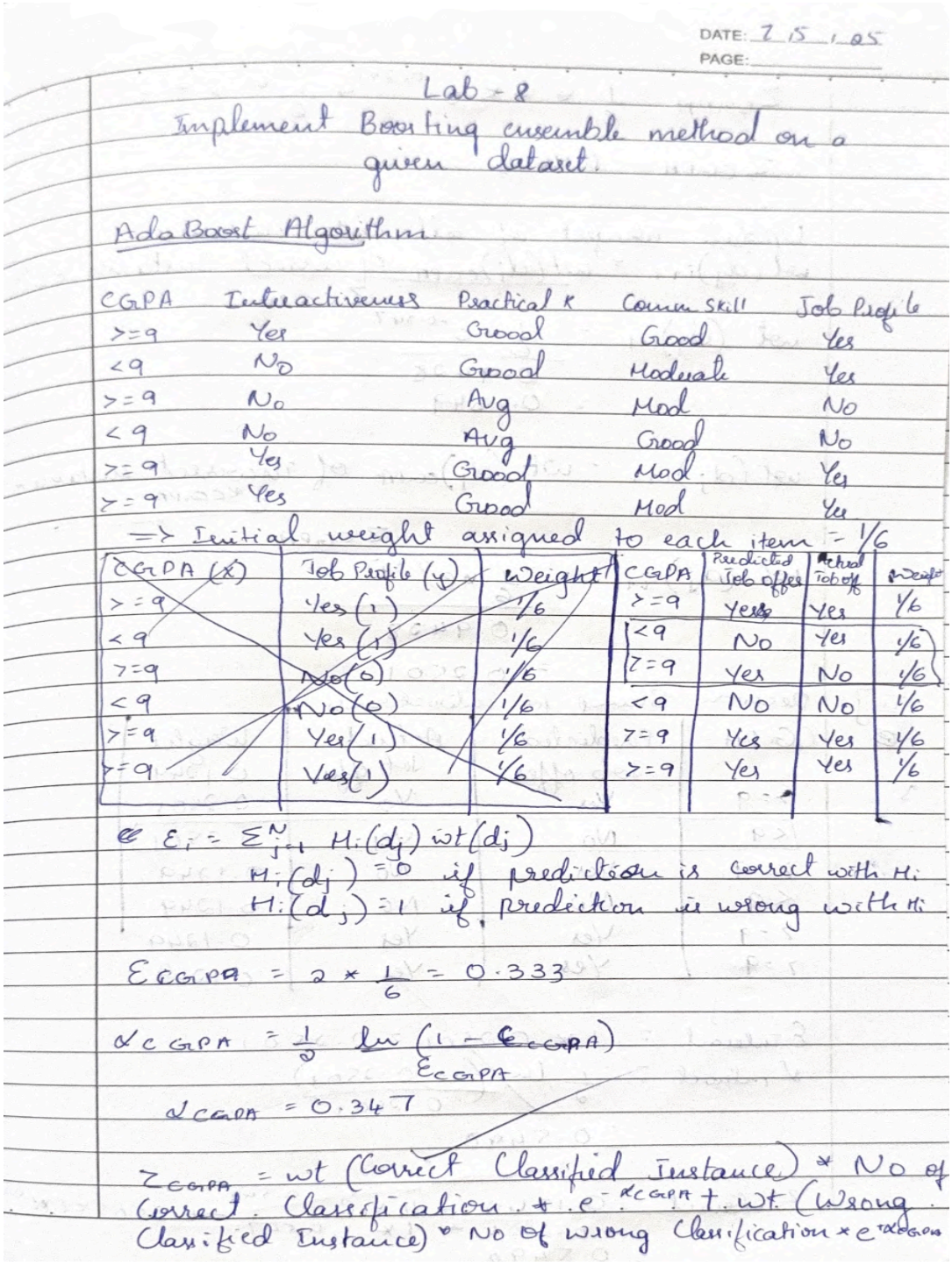


Code:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the iris dataset
iris = load_iris()

# Convert to DataFrame
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target

# Split features and target
X = df.drop('species', axis=1)
y = df['species']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

best_score = 0
best_n = 0
best_cm = None

# Try different numbers of trees
for n in range(1, 101):
    clf = RandomForestClassifier(n_estimators=n, random_state=42)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    if acc > best_score:
        best_score = acc
        best_n = n
        best_cm = confusion_matrix(y_test, y_pred)

print(f"Best Accuracy: {best_score}")
print(f"Number of Trees: {best_n}")
print("Confusion Matrix:")
print(best_cm)
```

Program 9

Implement Boosting ensemble method on a given dataset



DATE: 7/5/25
PAGE:

Lab - 8

Implement Boosting ensemble method on a given dataset.

Ada Boost Algorithm

CGPA	Interactiveness	Practical K	Comm Skill	Job Profile
>=9	Yes	Good	Good	Yes
<9	No	Good	Moderate	Yes
>=9	No	Avg	Mod	No
<9	No	Avg	Good	No
>=9	Yes	Good	Mod	Yes
>=9	Yes	Good	Mod	Yes

=> Initial weight assigned to each item = 1/6

CGPA (x)	Job Profile (y)	Weight
>=9	Yes (1)	1/6
<9	Yes (1)	1/6
>=9	No (0)	1/6
<9	No (0)	1/6
>=9	Yes (1)	1/6
>=9	Yes (1)	1/6

CGPA	Predicted Job offer	Actual Job offer	Weight
>=9	Yes	Yes	1/6
<9	No	Yes	1/6
>=9	Yes	No	1/6
<9	No	No	1/6
>=9	Yes	Yes	1/6
>=9	Yes	Yes	1/6

$\varepsilon_i = \sum_{j=1}^{N} H_i(d_j)\, wt(d_j)$

$H_i(d_j) = 0$ if prediction is correct with H_i

$H_i(d_j) = 1$ if prediction is wrong with H_i

$\varepsilon_{CGPA} = 2 * \frac{1}{6} = 0.333$

$\alpha_{CGPA} = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_{CGPA}}{\varepsilon_{CGPA}}\right)$

$\alpha_{CGPA} = 0.347$

Z_{CGPA} = wt (Correct Classified Instance) * No of Correct Classification * $e^{-\alpha_{CGPA}}$ + wt (Wrong Classified Instance) * No of wrong Classification * $e^{\alpha_{CGPA}}$

PAGE:

$$Z_{CGPA} = \frac{1}{6} * 4 * e^{-0.347} + \frac{1}{6} * 2 * e^{0.347}$$

$$Z_{CGPA} = 0.9428$$

Update weight of all data instances

$$wt(d_j)_{i+1} = \frac{wt(d_j)_{CGPA} \text{ of correct Instance} * e^{\alpha_{CGPA}}}{Z_{CGPA}}$$

$$wt(d_j)_{i+1} = \frac{\frac{1}{6} * e^{-0.347}}{0.9428}$$

$$= 0.1249$$

$$wt(d_j)_{i+1} = \frac{wt(d_j)_{CGPA} \text{ of Incorrect Instance} * e^{\alpha_{CGPA}}}{Z_{CGPA}}$$

$$wt(d_j)_{i+1} = \frac{\frac{1}{6} * e^{0.347}}{0.9428}$$

$$= 0.2501$$

II] Decision Stump for Entrepreneurs

CGPA	Predicted Job offers	Actual Job offer	Weight
			0.1249
>=9	Yes	Yes	0.2501
<9	No	Yes	0.2501
>=9	Yes	No	0.1249
<9	No	No	0.1249
>=9	Yes	Yes	0.1249
>=9	Yes	Yes	0.1249

$$\varepsilon_{interest} = 1 * 0.2501 = 0.2501$$

$$\alpha_{interest} = \frac{1}{2} \ln\left(\frac{1 - 0.2501}{0.2501}\right)$$

$$= 0.5490$$

$$Z_{interest} = 0.1249 * 4 * e^{-0.549} + 0.2501 * 1 * e^{0.549}$$

$$= 0.5490$$

DATE: / /
PAGE:

$wt(d_j)_{i+1} = \frac{0.1249 \times e^{-0.549}}{0.866} = 0.0832$

$wt(d_j)_{i+1} = \frac{0.2501 \times e^{-0.549}}{0.866} = 0.1667$

$wt(d_j)_{i+1} = \frac{0.2501 \times e^{0.549}}{0.866} = 0.509$

III DS for practical knowledge
No ~~exc~~ classification

IV DS for communication skill

V

$\alpha_{CGPA} = 0.347$	$\alpha_{Interact} = 0.5490$	$\alpha_{com} = -0.5485$	Final pred
Yes	Yes	Yes	Yes
No	No	No	No
Yes	No	No	Yes
No	No	Yes	No
Yes	Yes	No	Yes
Yes	Yes	No	Yes

For income.csv

Best accuracy score = 83.81 %
Confusion matrix =
No of trees = 100

Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
# Load dataset
df = pd.read_csv("/content/income.csv")
# Define features and target
X = df.drop("income_level", axis=1)
y = df["income_level"]
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Step 1: AdaBoost with 10 estimators
model = AdaBoostClassifier(n_estimators=10, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
base_accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
print(f"Accuracy with 10 estimators: {base_accuracy:.4f}")
print("Confusion Matrix:\n", conf_matrix)
# Step 2: Fine-tuning n_estimators
estimator_range = range(10, 201, 10)
scores = []
for n in estimator_range:
    model = AdaBoostClassifier(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)
    print(f"n_estimators={n},
Accuracy={acc:.4f}") # Plot accuracy vs. number
of estimators plt.plot(estimator_range, scores,
marker='o')
plt.title("Accuracy vs. Number of Estimators (AdaBoost)")
plt.xlabel("Number of Estimators")
plt.ylabel("Accuracy")
plt.grid(True)
plt.show()
# Best score and configuration
best_n = estimator_range[np.argmax(scores)]
best_score = max(scores)
print(f"Best accuracy: {best_score:.4f} achieved with {best_n} estimators.")
```

Program 10

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

DATE: 7.5.25
PAGE:

Lab - 9

For the given data
(1.0, 1.0) & (5.0, 7.0)

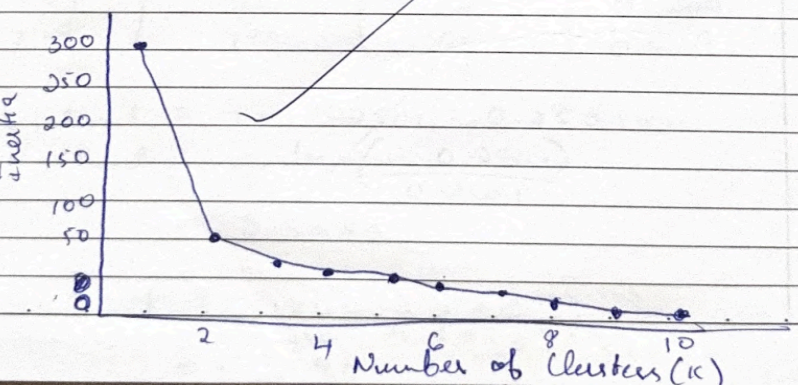
Record No	Close to C1	Close to C2	Assign to Cluster
R1(1.0,1.0)	dist(R1,C1)=0.0	7.21	Cluster 1
R2(1.5,2.0)	dist(R2,C1)=1.12	6.10	Cluster 1
R3(3.0,4.0)	dist(R3,C1)=3.61	3.61	Cluster 1
R4(5.0,7.0)	dist(R4,C1)=7.21	0.0	Cluster 2
R5(3.5,5.0)	dist(R5,C1)=4.12	2.5	Cluster 2
R6(4.5,5.0)	dist(R6,C1)=5.31	2.06	Cluster 2
R7(3.5,4.5)	dist(R7,C1)=4.30	2.92	Cluster 2

Cluster 1 {R1, R2, R3} & Cluster 2 {R4, R5, R6, R7}

C1 = (1.0+1.5+3.0)/3, (1.0+2.0+4.0)/3
= 5.5/3, 7.0/3
= 1.83, 2.33

C2 = (5.0+3.5+4.5+3.5)/4, (7+5+5+4.5)/4
= 16.5/4, 21.5/4
= 4.12, 5.37

For iris.csv dataset
Optimal value of K = 3

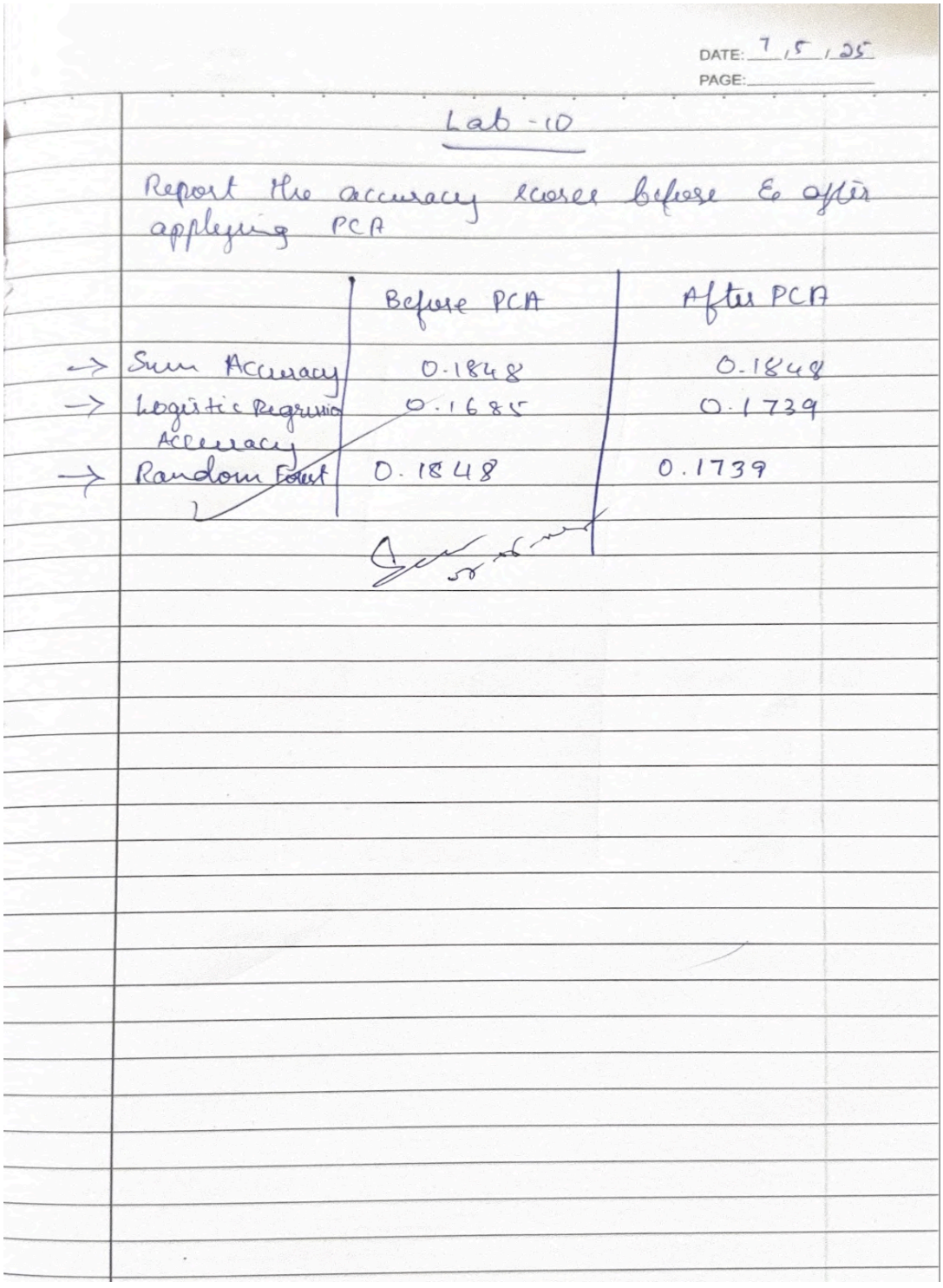


Code:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
df = pd.read_csv("/content/iris (6).csv") # Replace with actual file path if needed
# Use only petal_length and petal_width
X = df[['petal_length', 'petal_width']]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Elbow method to find optimal k
wcss = []
k_values = range(1, 11)
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
plt.figure(figsize=(8, 5))
plt.plot(k_values, wcss, marker='o')
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('WCSS')
plt.grid(True)
plt.tight_layout()
plt.show()
# Apply K-Means with optimal k = 3
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
clusters = kmeans.fit_predict(X_scaled)
# Add cluster labels to original dataframe
df['cluster'] = clusters
plt.figure(figsize=(8, 5))
for cluster in range(3):
    cluster_points = X_scaled[clusters == cluster]
    plt.scatter(cluster_points[:, 0], cluster_points[:, 1], label=f'Cluster {cluster}')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=200, c='black', marker='X', label='Centroids')
plt.title('K-Means Clustering (k=3)')
plt.xlabel('Petal Length (scaled)')
plt.ylabel('Petal Width (scaled)')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method



DATE: 7/5/25
PAGE:

Lab - 10

Report the accuracy scores before & after applying PCA

	Before PCA	After PCA
→ Svm Accuracy	0.1848	0.1848
→ Logistic Regression Accuracy	0.1685	0.1739
→ Random Forest	0.1848	0.1739

Code:

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

# Load dataset
df = pd.read_csv("/content/heart (1).csv") # Use actual path if needed

# Label encode binary categorical columns
le = LabelEncoder()
df["Sex"] = le.fit_transform(df["Sex"]) # M/F -> 1/0
df["ExerciseAngina"] = le.fit_transform(df["ExerciseAngina"]) # Y/N -> 1/0

# One-hot encode categorical columns with >2 categories
df = pd.get_dummies(df, columns=["ChestPainType", "RestingECG", "ST_Slope"],
drop_first=True)

# Split features and target
X = df.drop("HeartDisease", axis=1)
y = df["HeartDisease"]

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

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Define models
models = {
   "SVM": SVC(),
   "Logistic Regression": LogisticRegression(max_iter=1000),
   "Random Forest": RandomForestClassifier()
}

# Train and evaluate models (before PCA)
print("=== Accuracy Before PCA ===")
for name, model in models.items():
```

```
    model.fit(X_train, y_train)
    acc = accuracy_score(y_test, model.predict(X_test))
    print(f"{name}: {acc:.4f}")

# Apply PCA to reduce dimensions to 2
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Split again for PCA-transformed data
X_train_pca, X_test_pca, y_train, y_test = train_test_split(X_pca, y, test_size=0.2,
random_state=42)

# Train and evaluate models (after PCA)
print("\n=== Accuracy After PCA (2 components) ===")
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    acc = accuracy_score(y_test, model.predict(X_test_pca))
    print(f"{name}: {acc:.4f}")
```