VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Sanjeet Prajwal Pandit (1BM22CS241)

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 **Sanjeet Prajwal Pandit (1BM22CS241),** 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

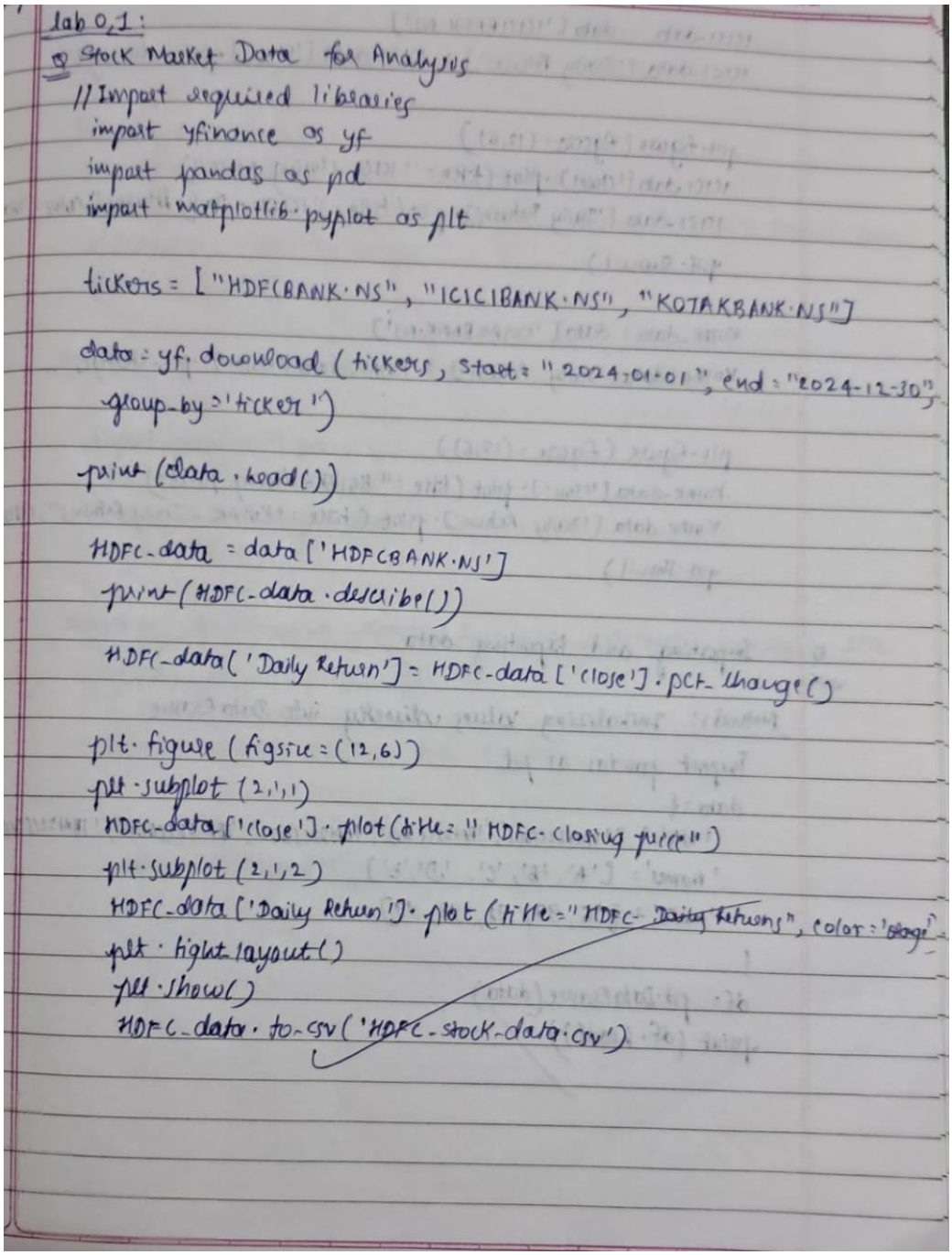
Index

Github Link:
https://github.com/Sanjeet-108/6thSem-ML-Lab

Program 1

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

Screenshot:



DATE:

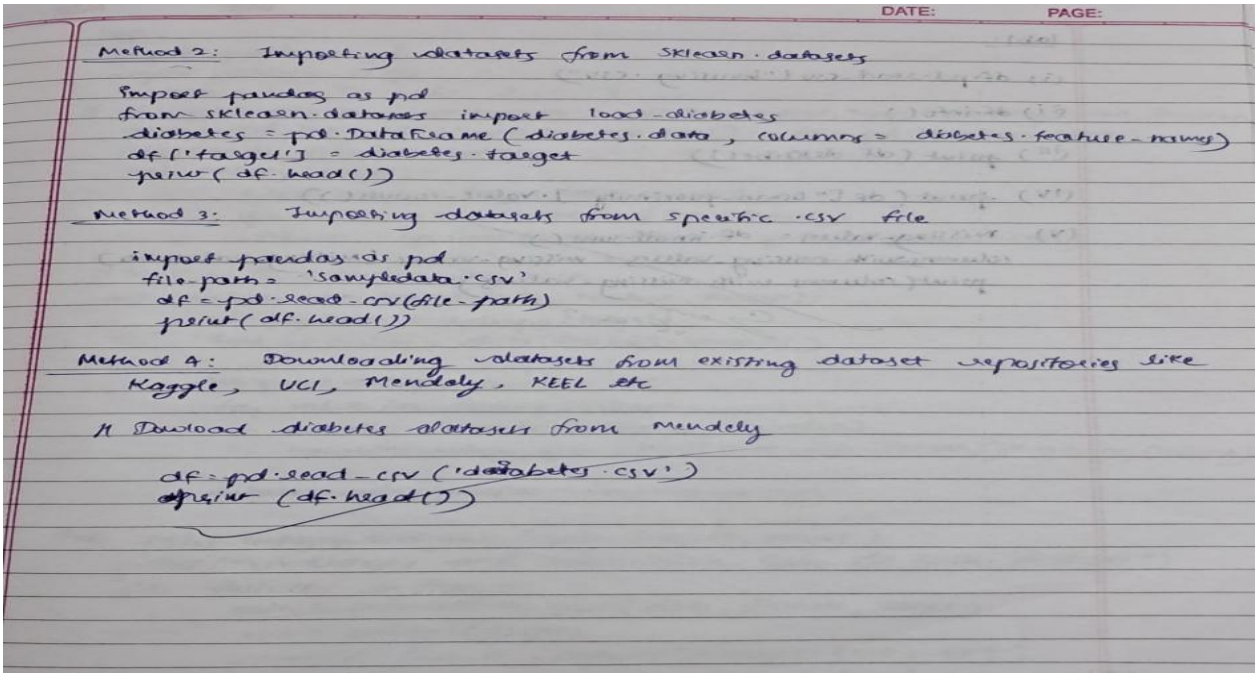
PAGE

```
ICICI_data = data['ICICIBANK.NS']
ICICI_data['Daily Return'] = ICICI_data['Close'].pct_change

plt.figure(figsize=(12,6))
ICICI_data['Close'].plot(title="ICICI - Closing price")
ICICI_data['Daily Return'].plot(title="ICICI - Daily Returns", color='
plt.show()

KOTAK_data = data['KOTAKBANK.NS']
KOTAK_data['Daily Return'] = KOTAK_data['Close'].pct_change

plt.figure(figsize=(12,6))
KOTAK_data['Close'].plot(title="KOTAK - Closing price")
KOTAK_data['Daily Return'].plot(title="KOTAK - Daily Returns", color='
plt.show()
```

Q Importing and Exporting data

Method 1: Initializing values directly into DataFrame

```
import pandas as pd
data = {
  'USN' = ['1BM22CS001', '1BM22CS002', '1BM22CS003', '1BM22CS004', '1BM22CS005'],
  'Names' = ['A', 'B', 'C', 'D', 'E'],
  'Marks' = [96, 97, 98, 98, 99]
}
df = pd.DataFrame(data)
print(df.head())
```



Code:

```
import pandas as pd
df=pd.DataFrame({
   "ÜSN":[21,22,23,24,25],
   "Name":["Sam","Ram","Dam","rahul","rakshith"],
   "Marks":[45,78,24,90,60]
})
Df
```

```
from sklearn.datasets import load_diabetes
dia=load_diabetes()
df=pd.DataFrame(dia.data,columns=dia.feature_names)
df
```

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

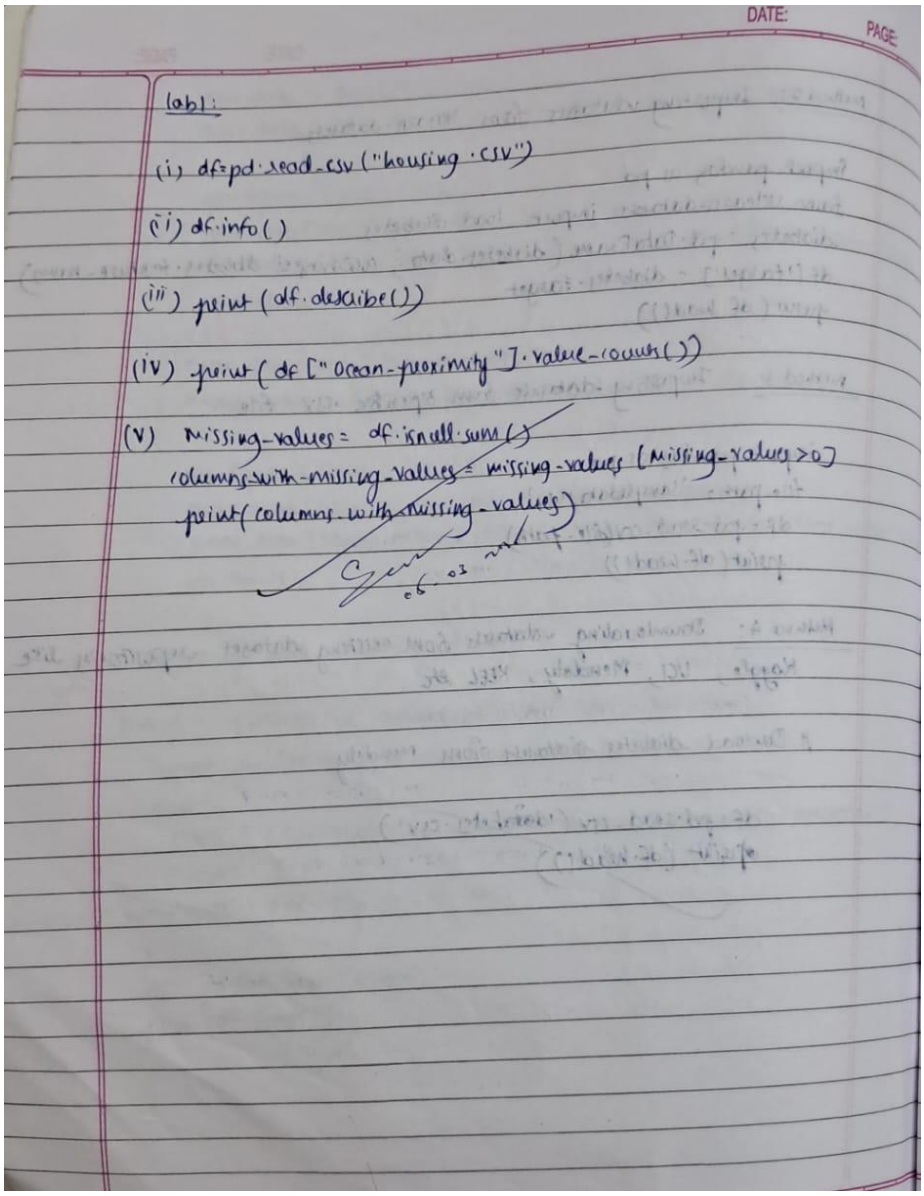
```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

# Define the tickers and date range
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
```

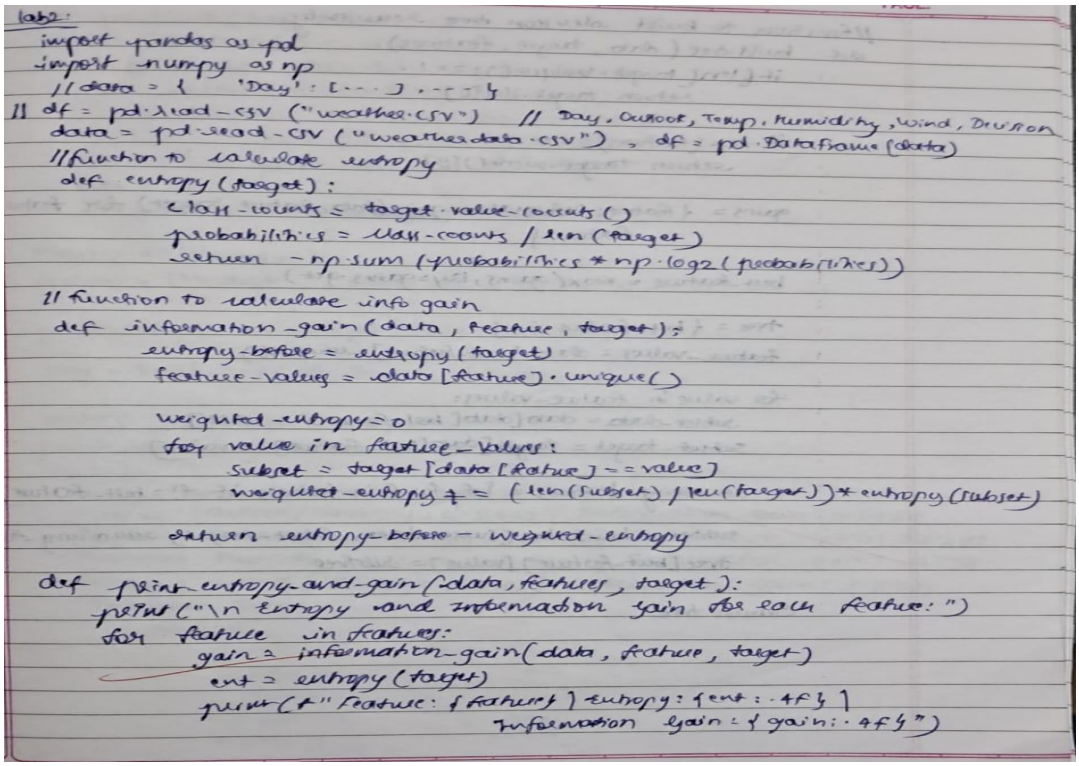
```
start_date = "2024-01-01"
end_date = "2024-12-30"

# Download the data
data = yf.download(tickers, start=start_date, end=end_date)

# Check the structure of the downloaded data
print(data.head())  # Print the first few rows to see the column names

# Extract closing prices (if 'Adj Close' doesn't exist, fall back to 'Close')
closing_prices = data["Close"]

# Handle missing values by forward filling
closing_prices = closing_prices.ffill()

# Calculate daily returns
daily_returns = closing_prices.pct_change()

# Calculate 50-day moving average
moving_average = closing_prices.rolling(window=50).mean()

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

# Closing Prices and 50-day Moving Average
plt.subplot(2, 1, 1)
for ticker in tickers:
    plt.plot(closing_prices.index, closing_prices[ticker], label=f"{ticker} Closing Price")
    plt.plot(moving_average.index, moving_average[ticker], label=f"{ticker} 50-Day MA", linestyle="--")
plt.title("Closing Prices of HDFC Bank, ICICI Bank, and Kotak Mahindra Bank with 50-Day Moving Average")
plt.xlabel("Date")
plt.ylabel("Closing Price")
plt.legend()

# Daily Returns
plt.subplot(2, 1, 2)
for ticker in tickers:
    plt.plot(daily_returns.index, daily_returns[ticker], label=ticker)
plt.title("Daily Returns of HDFC Bank, ICICI Bank, and Kotak Mahindra Bank")
plt.xlabel("Date")
plt.ylabel("Daily Return")
plt.legend()

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

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:





Code:

```
import pandas as pd

import numpy as np

from sklearn.preprocessing import MinMaxScaler, StandardScaler, OneHotEncoder

from sklearn.impute import SimpleImputer

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline


# Load Diabetes dataset (update with your file path in Google Colab)

diabetes_df = pd.read_csv("/content/Dataset of Diabetes .csv")


# Load Adult Income dataset (update with your file path in Google Colab)

adult_income_df = pd.read_csv("/content/adult.csv")
```

```
# Function for data cleaning and transformations

def preprocess_data(df):

    # 1. Data Cleaning: Handle Missing Values


    # Identify columns with missing values

    missing_values = df.isnull().sum()

    print("Columns with missing values:\n", missing_values[missing_values > 0])


    # Handle missing values: Use median for numerical columns and most frequent for categorical columns

    numerical_cols = df.select_dtypes(include=["float64", "int64"]).columns

    categorical_cols = df.select_dtypes(include=["object"]).columns


    # Impute numerical columns with median, and categorical columns with the most frequent value

    imputer = SimpleImputer(strategy='most_frequent')

    df[categorical_cols] = imputer.fit_transform(df[categorical_cols])


    imputer = SimpleImputer(strategy='median')

    df[numerical_cols] = imputer.fit_transform(df[numerical_cols])


    # 2. Handling Categorical Data: One-Hot Encoding

    print("\nCategorical Columns:\n", categorical_cols)


    # Apply OneHotEncoder to categorical columns
```

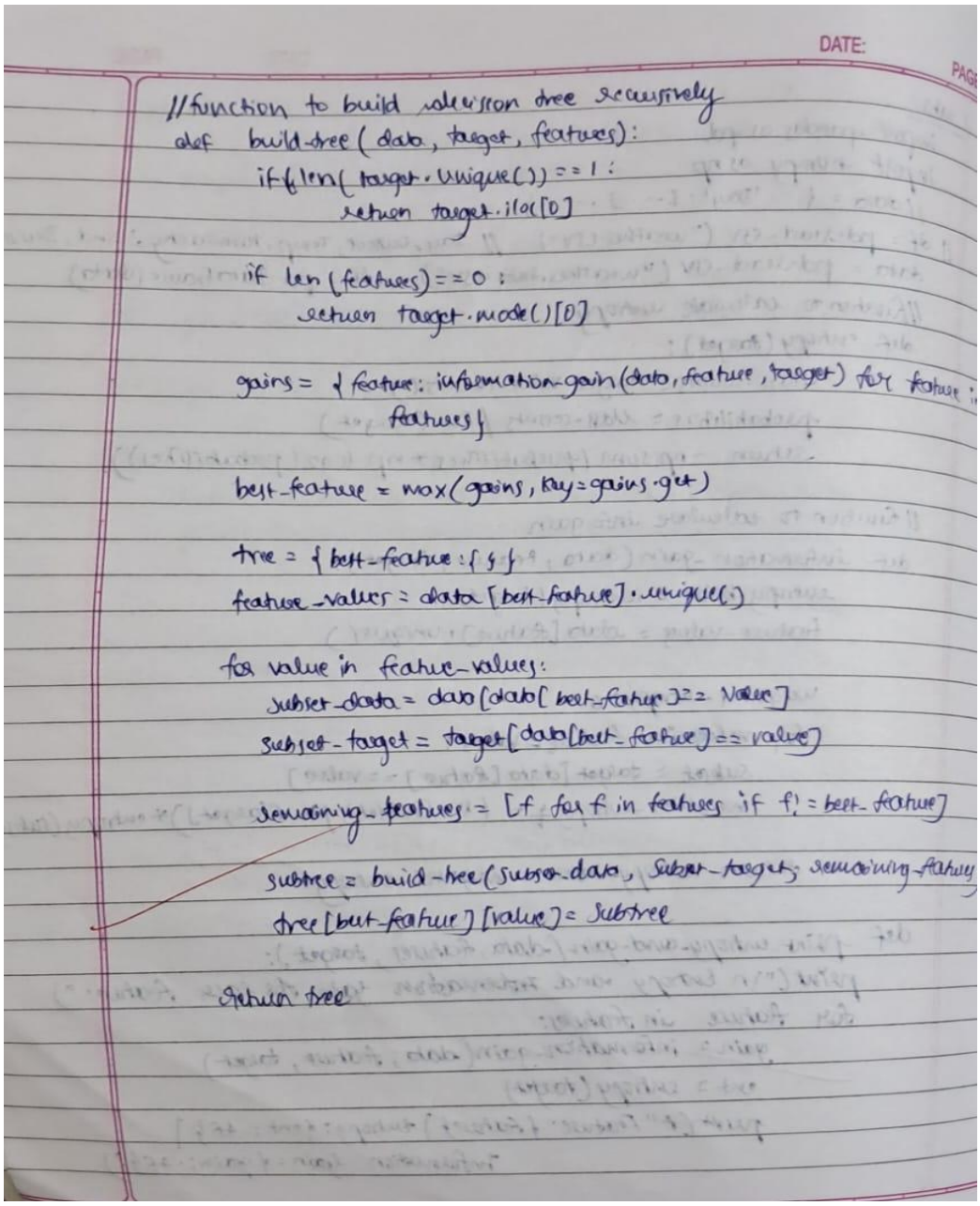
```
    encoder = OneHotEncoder(drop='first', sparse_output=False)
    encoded_categorical_data = encoder.fit_transform(df[categorical_cols])

    # Creating a DataFrame for encoded categorical data
    encoded_categorical_df = pd.DataFrame(encoded_categorical_data,
columns=encoder.get_feature_names_out(categorical_cols))

    # Drop original categorical columns and concatenate the encoded columns
    df = df.drop(categorical_cols, axis=1)
    df = pd.concat([df, encoded_categorical_df], axis=1)

    # 3. Handling Outliers: Using Z-score (Optional - can be added if needed)
    from scipy import stats
    z_scores = np.abs(stats.zscore(df[numerical_cols]))
    df = df[(z_scores < 3).all(axis=1)]  # Remove rows with outliers

    # 4. Data Transformations: Min-Max Scaling
    min_max_scaler = MinMaxScaler()
    df[numerical_cols] = min_max_scaler.fit_transform(df[numerical_cols])

    # 5. Data Transformations: Standardization
    standard_scaler = StandardScaler()
    df[numerical_cols] = standard_scaler.fit_transform(df[numerical_cols])

    return df
```

```
# Apply preprocessing for Diabetes dataset
diabetes_df = preprocess_data(diabetes_df)
print("\nPreprocessed Diabetes Dataset:")
print(diabetes_df.head())

# Apply preprocessing for Adult Income dataset
adult_income_df = preprocess_data(adult_income_df)
print("\nPreprocessed Adult Income Dataset:")
print(adult_income_df.head())
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:



```
DATE:

//function to build decision tree recursively
def build_tree(data, target, features):
    if len(target.unique()) == 1:
        return target.iloc[0]

    if len(features) == 0:
        return target.mode()[0]

    gains = {feature: information_gain(data, feature, target) for feature in
             features}

    best_feature = max(gains, key=gains.get)

    tree = {best_feature: {}}
    feature_values = data[best_feature].unique()

    for value in feature_values:
        subset_data = data[data[best_feature] == value]
        subset_target = target[data[best_feature] == value]

        remaining_features = [f for f in features if f != best_feature]

        subtree = build_tree(subset_data, subset_target, remaining_features)
        tree[best_feature][value] = subtree

    return tree
```

DATE: PAGE:

```
def print_tree(tree, indent=""):
    if isinstance(tree, dict):
        for feature, branches in tree.items():
            print(f"{indent}{feature}:")
            for value, subtree in branches.items():
                print(f"{indent} {value} ->", end=" ")
                print_tree(subtree, indent+"  ")
    else:
        print(f"{indent}{tree}")

target = df['Decision']
features = ['Outlook', 'Temperature', 'Humidity', 'Wind']

print_entropy_and_gain(df, features, target)

tree = build_tree(df, target, features)

print("\n Decision Tree:")
print_tree(tree, indent="  ")
```

Output:

Entropy and Information gain for each feature:
Feature: Outlook | Entropy: 0.9403 | Information gain: 0.2467
Feature: Temperature | Entropy: 0.9403 | Information gain: 0.0292
Feature: Humidity | Entropy: 0.9403 | Information gain: 0.1518
Feature: Wind | Entropy: 0.9403 | Information gain: 0.048

Decision Tree:

```
Outlook:
    Sunny -> Humidity:
        High -> No
        Normal -> Yes
    Overcast -> Yes
    Rainy -> Wind:
        Weak -> Yes
```

Aug/12/3/25

19/3/25

DATE:

PAGE:

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

(a) Write logistic regression equation for this problem
(b) Calculate probability that student who studies for 7 hours will pass
(c) Determine predicted class (pass or fail) for this student using threshold of 0.5.

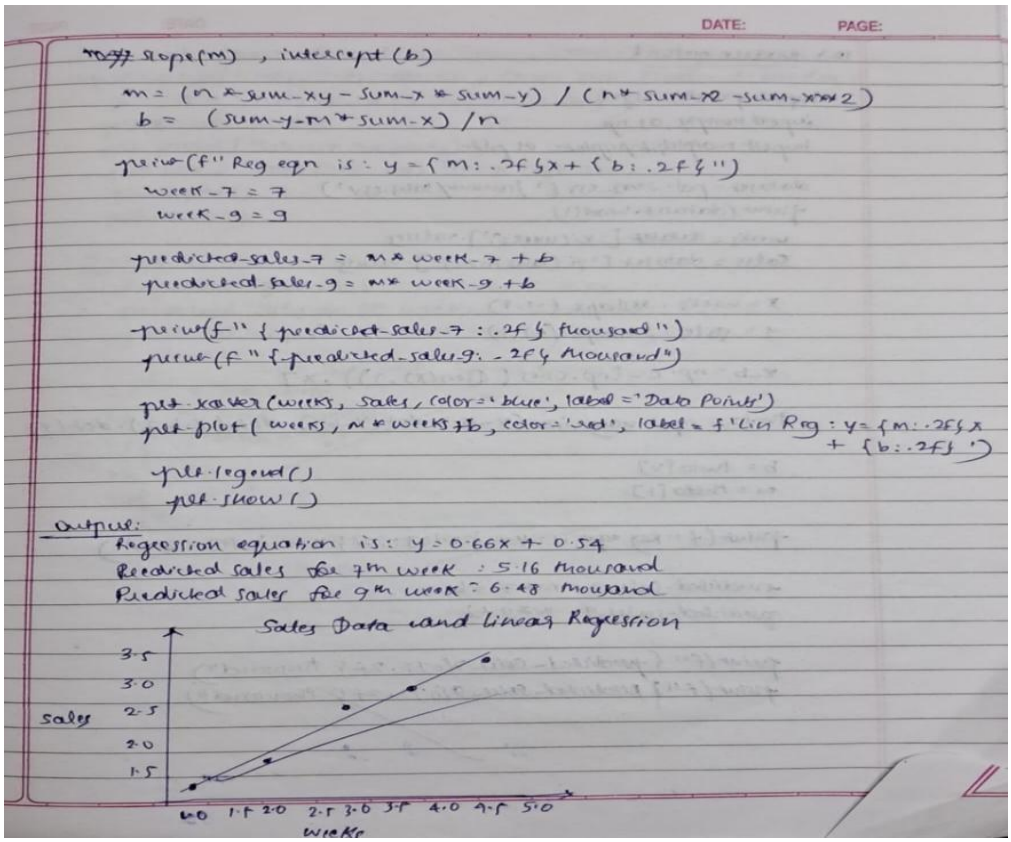
(2) Consider $z = [2, 1, 0]$ for three classes. Apply softmax function to find probability values of three classes.

(*) Linear regression

(1)
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset = pd.read_csv('/content/sales.csv')
print(dataset.head())

weeks = dataset['xi (weeks)'].values
sales = dataset['yi (sales in rs)'].values
n = len(weeks)

sum_x = np.sum(weeks)
sum_y = np.sum(sales)
sum_x2 = np.sum(weeks**2)
sum_xy = np.sum(weeks*sales)
```

sales.csv

Week	Sales (in Rs)
1	1.2
2	1.8
3	2.6
4	3.2
5	3.8



Code:

```
import numpy as np

import matplotlib.pyplot as plt


# Given data

x = np.array([1, 2, 3, 4])  # weeks

y = np.array([1, 3, 4, 8])  # corresponding y values



X = np.vstack([np.ones(len(x)), x]).T
```

```
theta = np.linalg.inv(X.T @ X) @ X.T @ y


print(f"Intercept (theta_0): {theta[0]}")
print(f"Slope (theta_1): {theta[1]}")



week_to_predict = 5
y_pred = theta[0] + theta[1] * week_to_predict
print(f"Predicted value for week {week_to_predict}: {y_pred}")



plt.scatter(x, y, color='blue', label='Data points')  # scatter plot of the data
plt.plot(x, theta[0] + theta[1] * x, color='red', label='Regression line')  # plot the regression line



plt.scatter(week_to_predict, y_pred, color='green', label=f'Prediction for week {week_to_predict}')



plt.xlabel('Weeks')
plt.ylabel('Y values')
plt.title('Linear Regression: Week vs Y values')

# Show legend
plt.legend()
```

```
# Show the plot
plt.show()
```

2. Approach:

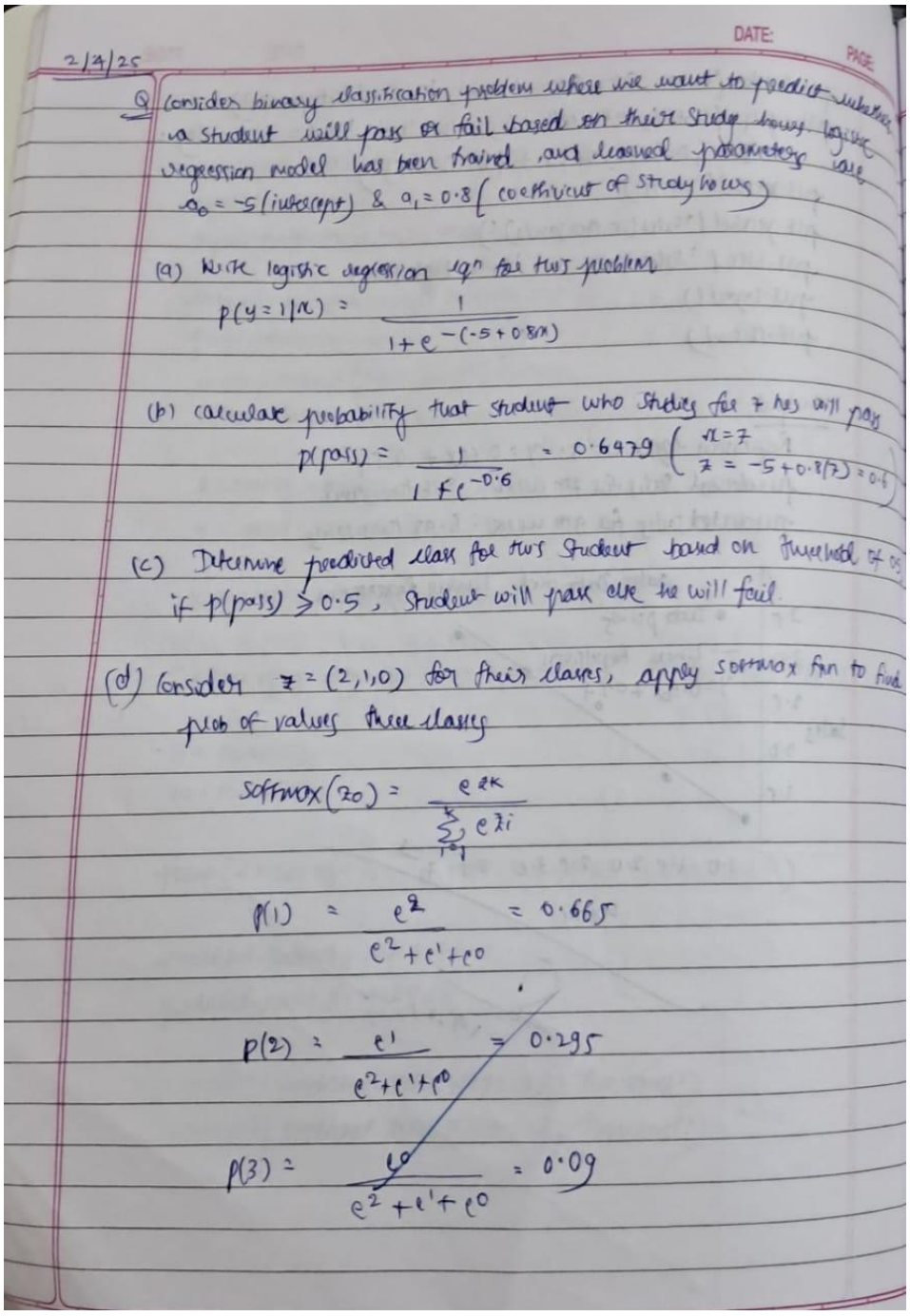
```
import numpy as np
import matplotlib.pyplot as plt

# Given data
x = np.array([1, 2, 3, 4])  # weeks
y = np.array([1, 3, 4, 8])  # corresponding y values

# Calculate means
x_mean = np.mean(x)
y_mean = np.mean(y)

# Calculate slope (a1) using the given formula
a1 = (np.mean(x * y) - x_mean * y_mean) / (np.mean(x**2) - x_mean**2)

# Calculate intercept (a0) using the given formula
a0 = y_mean - a1 * x_mean

# Print the result
print(f"Intercept (a0): {a0}")
```

```
print(f"Slope (a1): {a1}")


# Predict for week 5
week_to_predict = 5
y_pred = a0 + a1 * week_to_predict
print(f"Predicted value for week {week_to_predict}: {y_pred}")


# Plotting the data points and the regression line
plt.scatter(x, y, color='blue', label='Data points')  # scatter plot of the data
plt.plot(x, a0 + a1 * x, color='red', label='Regression line')  # plot the regression line


# Mark the prediction point for week 5
plt.scatter(week_to_predict, y_pred, color='green', label=f'Prediction for week {week_to_predict}')


# Adding labels and title
plt.xlabel('Weeks')
plt.ylabel('Y values')
plt.title('Linear Regression: Week vs Y values')


# Show legend
plt.legend()


# Show the plot
plt.show()
```

Program 4:

Build Logistic Regression Model for a given dataset

Screenshot:



DATE: PAGE:

Q After building logistic regression models, answer following questions:

(1) For dataset file "HR_comma_sep.csv"

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

(ii) What was accuracy of your logistic regression model? Do you think this is good accuracy? Why or why not?

(2) For 200 dataset

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

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

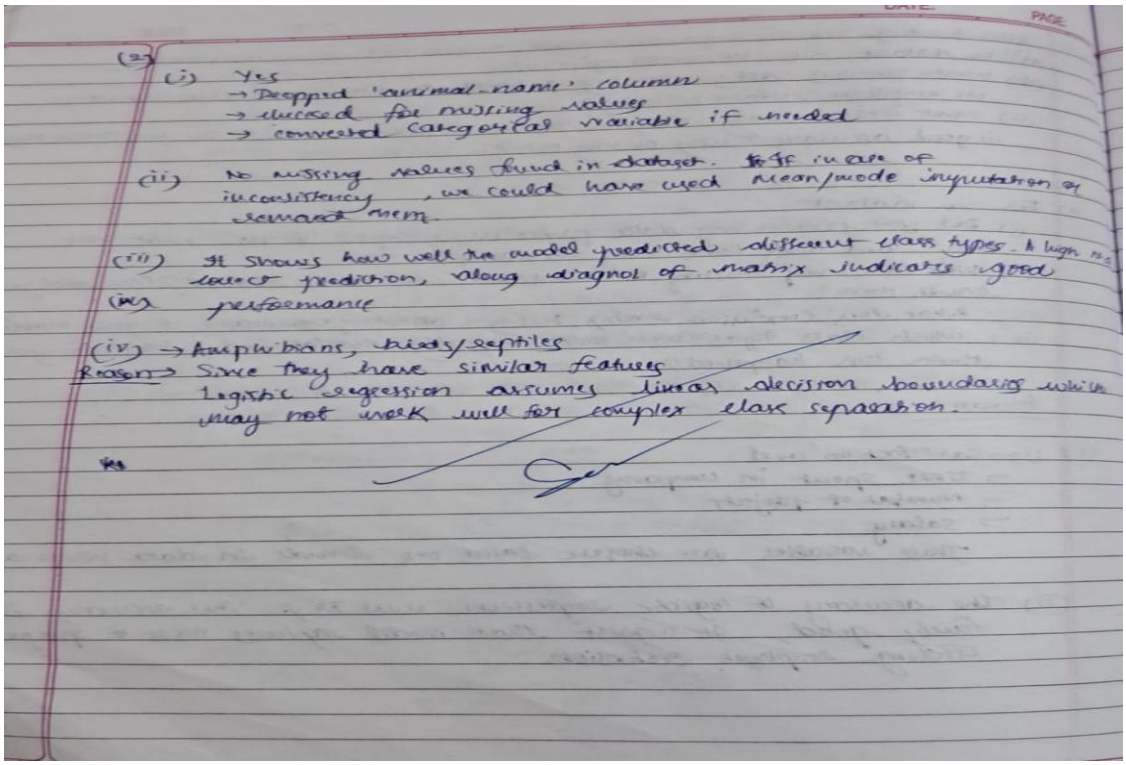
(iii) What does confusion matrix tell you about performance of your model

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

Answer:

(1) (i) → Satisfaction Level
→ Time Spent in company
→ Number of projects
→ Salary
These variables are chosen based on trends in data visualization

(ii) The accuracy of logistic regression was 78%. This accuracy is fairly good, it suggest that model captures most of properties affecting employee retension.



Code:

#LogisticRegression_Multiclass.ipynb

 # Import necessary libraries

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

from sklearn import metrics

import matplotlib.pyplot as plt

```
# Load the Iris dataset
iris =  pd.read_csv("/content/iris (2).csv")
iris.head()

X=iris.drop('species',axis='columns')# Features (sepal length, sepal width, petal length, petal width)
y = iris.species # Target labels (0: Setosa, 1: Versicolor, 2: Virginica)

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

# Initialize the Multinomial Logistic Regression model
# Use 'multinomial' for multi-class classification and 'lbfgs' solver
model = LogisticRegression(multi_class='multinomial')

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model on the test data
accuracy = accuracy_score(y_test, y_pred)

# Display the accuracy
```

```
print(f"Accuracy of the Multinomial Logistic Regression model on the test set: {accuracy:.2f}")


confusion_matrix = metrics.confusion_matrix(y_test, y_pred)


cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix, display_labels =
["Setosa", "Versicolor", "Virginica"])


cm_display.plot()

plt.show()
```

Binary Classification:

```
#LogisticRegression_Binary.ipynb


# Commented out IPython magic to ensure Python compatibility.

import pandas as pd

from matplotlib import pyplot as plt

# %matplotlib inline

#"%matplotlib inline" will make your plot outputs appear and be stored within the notebook.


df = pd.read_csv("/content/insurance_data (1).csv")

df.head()


plt.scatter(df.age,df.bought_insurance,marker='+',color='red')


from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
```

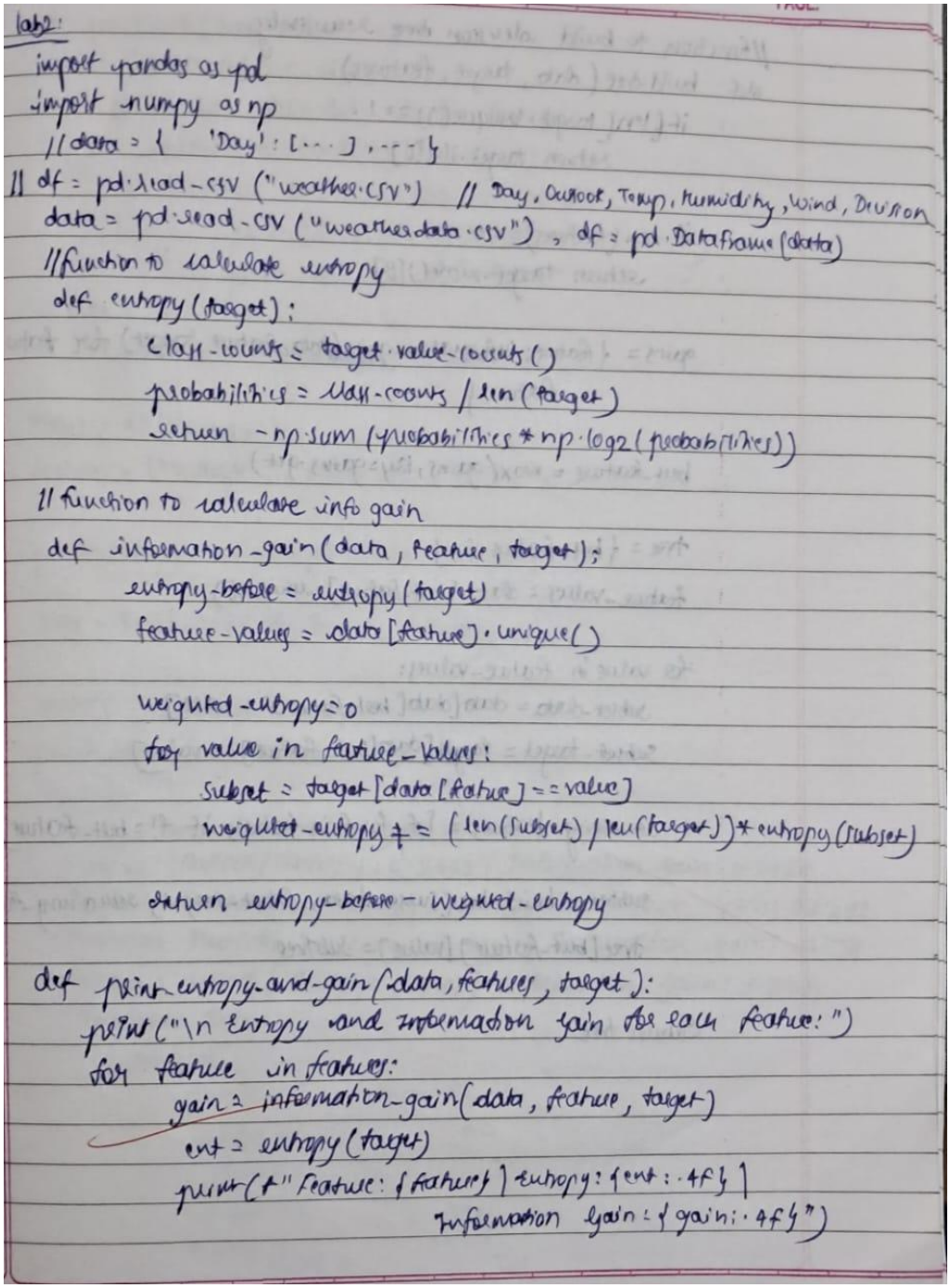
```
train_test_split(df[['age']],df.bought_insurance,train_size=0.9,random_state=10)

X_train.shape


X_test


from sklearn.linear_model import LogisticRegression

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([[60]])

y_predicted
```

```
#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_

#Lets defined sigmoid function now and do the math with hand
import math
def sigmoid(x):
  return 1 / (1 + math.exp(-x))

def prediction_function(age):
   z = 0.127 * age - 4.973 # 0.12740563 ~ 0.0127 and -4.97335111 ~ -4.97
   y = sigmoid(z)
   return y

age = 35
prediction_function(age)

"""0.37 is less than 0.5 which means person with 35 will not buy the insurance"""
```

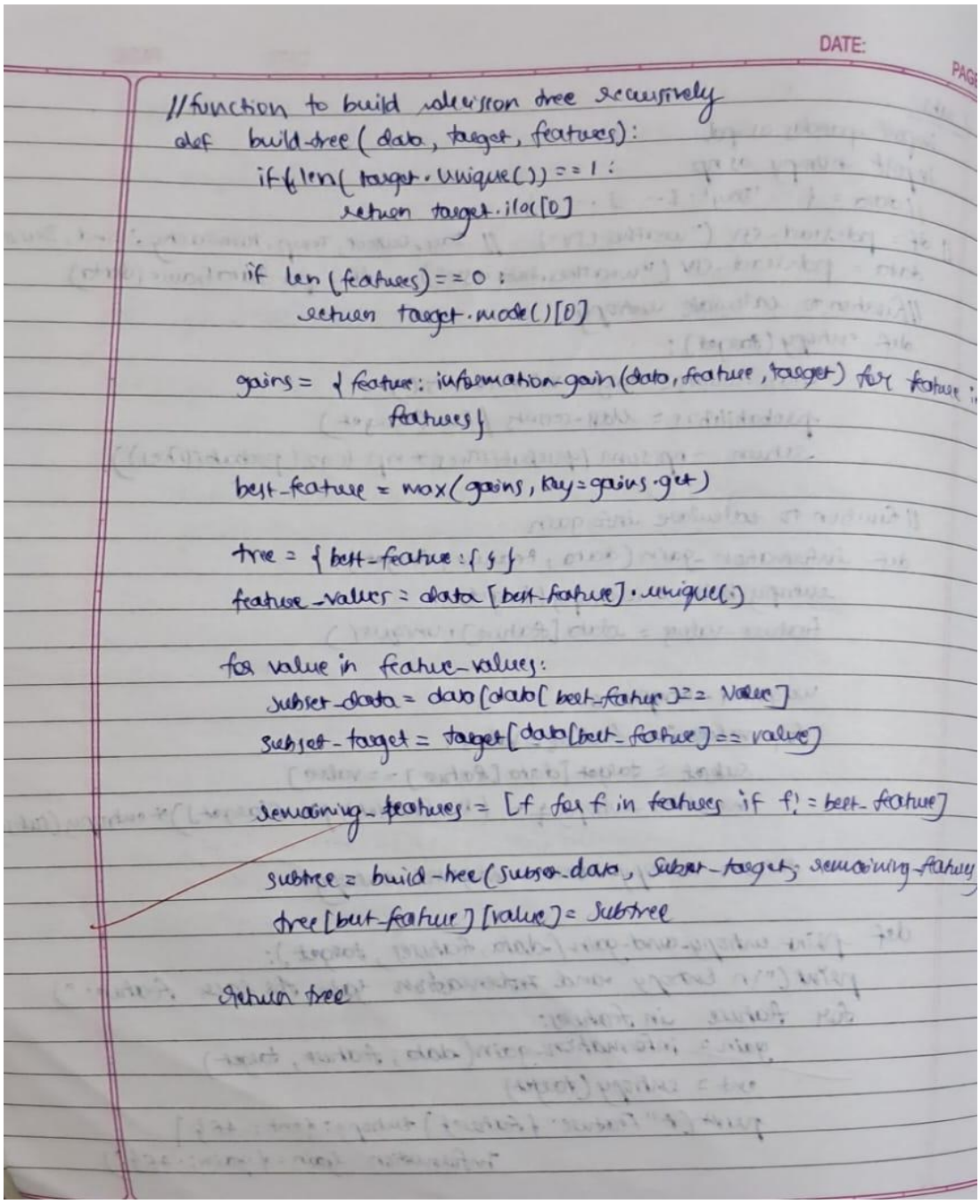
Program 5

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

Screenshot:



```
lab2:
import pandas as pd
import numpy as np
// data = {  'Day': [...] , ... }
// df = pd.read_csv("weather.csv")  // Day, Outlook, Temp, Humidity, Wind, Decision
data = pd.read_csv("weather_data.csv") , df = pd.DataFrame(data)
//function to calculate entropy
def entropy(target):
    class_counts = target.value_counts()
    probabilities = class_counts / len(target)
    return -np.sum(probabilities * np.log2(probabilities))

// function to calculate info gain
def information_gain(data, feature, target):
    entropy_before = entropy(target)
    feature_values = data[feature].unique()

    weighted_entropy = 0
    for value in feature_values:
        subset = target[data[feature] == value]
        weighted_entropy += (len(subset) / len(target)) * entropy(subset)

    return entropy_before - weighted_entropy

def print_entropy_and_gain(data, features, target):
    print("\n Entropy and information gain for each feature:")
    for feature in features:
        gain = information_gain(data, feature, target)
        ent = entropy(target)
        print(f"Feature: {feature} | Entropy: {ent:.4f} |
                    Information Gain: {gain:.4f}")
```



```
//function to build decision tree recursively
def build_tree(data, target, features):
    if len(target.unique()) == 1:
        return target.iloc[0]

    if len(features) == 0:
        return target.mode()[0]

    gains = {feature: information_gain(data, feature, target) for feature in
             features}

    best_feature = max(gains, key=gains.get)

    tree = {best_feature: {}}
    feature_values = data[best_feature].unique()

    for value in feature_values:
        subset_data = data[data[best_feature] == value]
        subset_target = target[data[best_feature] == value]

        remaining_features = [f for f in features if f != best_feature]

        subtree = build_tree(subset_data, subset_target, remaining_features)
        tree[best_feature][value] = subtree

    return tree
```

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from sklearn.tree import DecisionTreeClassifier
```

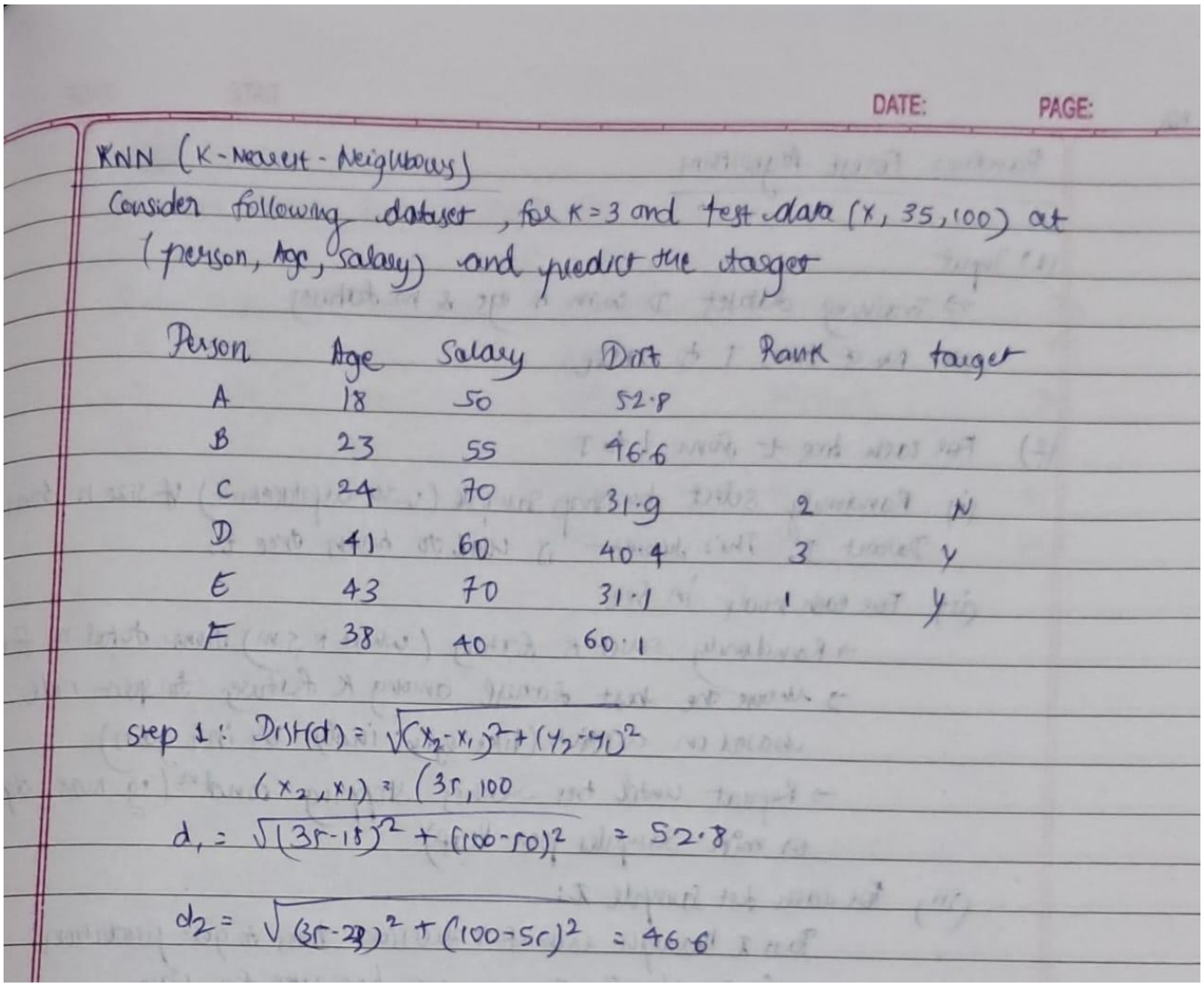
```
from sklearn import tree
import matplotlib.pyplot as plt

# Load the newly uploaded dataset
data = pd.read_csv('/content/seattle-weather.csv')

# Data Preparation
data_cleaned = data.drop('date', axis=1)
label_encoder = LabelEncoder()
data_cleaned['weather'] = label_encoder.fit_transform(data_cleaned['weather'])

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

# Split the 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)

# Create and train the Decision Tree Classifier with improved clarity
id3_classifier = DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=42)
id3_classifier.fit(X_train, y_train)

# Visualize the Decision Tree with larger font size for clarity
plt.figure(figsize=(20, 12))
```

```
tree.plot_tree(id3_classifier, feature_names=X.columns, class_names=list(label_encoder.classes_),
filled=True, fontsize=10)

plt.title("ID3 Decision Tree Classifier (Enhanced Clarity)")

plt.show()
```

Program 6

Build KNN Classification model for a given dataset

Screenshot:



DATE: PAGE:

KNN (K-Nearest-Neighbours)

Consider following dataset, for K=3 and test data (x, 35, 100) at (person, Age, Salary) and predict the target

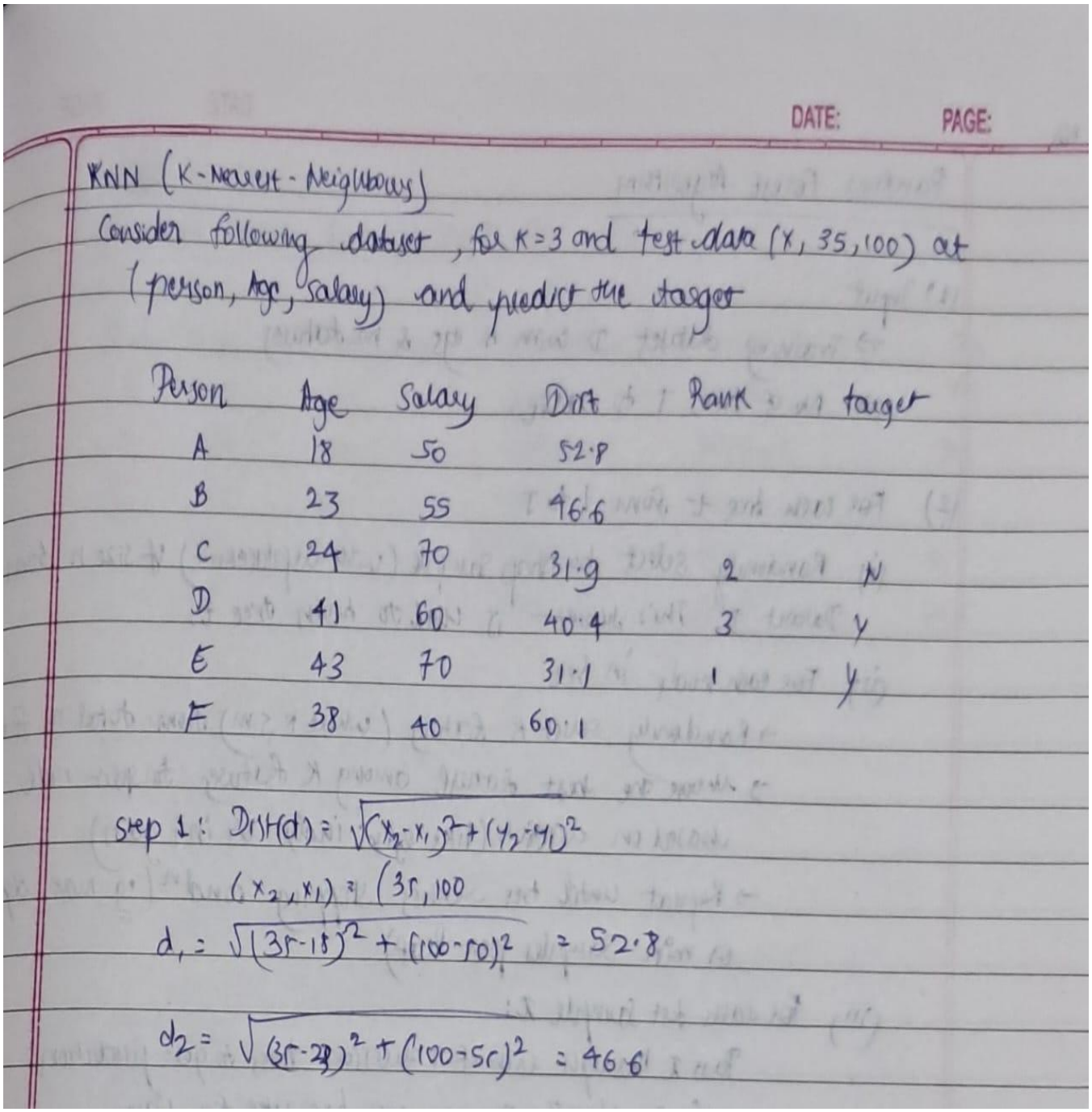
Person	Age	Salary	Dist	Rank	target
A	18	50	52.8		
B	23	55	46.6		
C	24	70	31.9	2	N
D	41	60	40.4	3	Y
E	43	70	31.1	1	Y
F	38	40	60.1		

step 1: $Dist(d) = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

$(x_2, x_1) = (35, 100$

$d_1 = \sqrt{(35-18)^2+(100-50)^2} = 52.8$

$d_2 = \sqrt{(35-23)^2+(100-55)^2} = 46.6$



DATE: PAGE:

KNN (K-Nearest-Neighbours)

Consider following dataset, for K=3 and test data (x, 35, 100) at (person, Age, Salary) and predict the target

Person	Age	Salary	Dist	Rank	target
A	18	50	52.8		
B	23	55	46.6		
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		

step 1: $Dist(d) = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

$(x_2, x_1) = (35, 100$

$d_1 = \sqrt{(35-18)^2+(100-50)^2} = 52.8$

$d_2 = \sqrt{(35-23)^2+(100-55)^2} = 46.6$

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

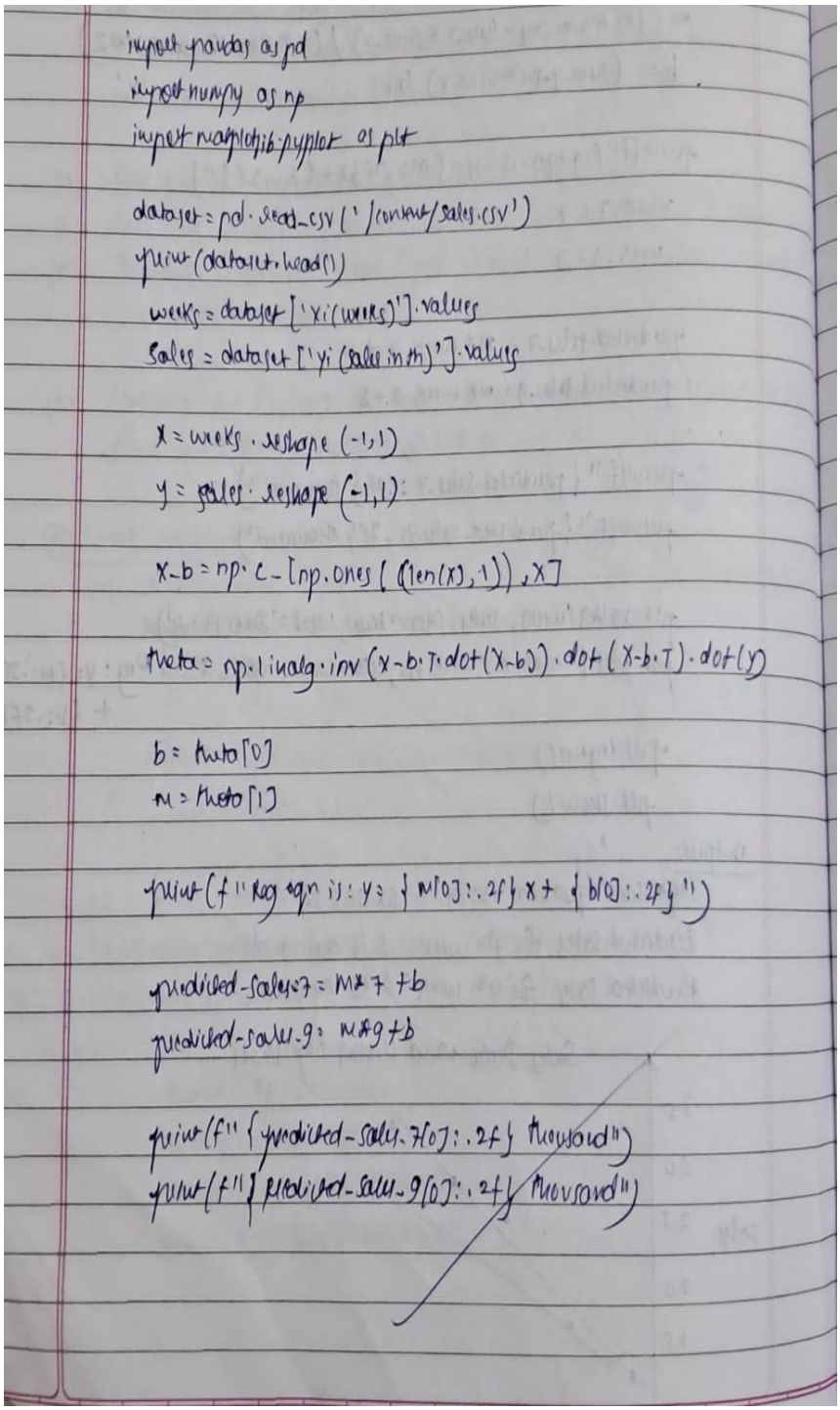
```
import matplotlib.pyplot as plt

import seaborn as sns


# Load the Iris dataset

iris = pd.read_csv("/content/iris (2).csv")


# Prepare the data

X = iris.drop('species', axis='columns')

y = iris.species


# Split the 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)


# Initialize the KNN classifier with k=3

knn = KNeighborsClassifier(n_neighbors=3)  # You can experiment with different k values


# Train the classifier

knn.fit(X_train, y_train)


# Make predictions

y_pred = knn.predict(X_test)


# Evaluate the model

accuracy = accuracy_score(y_test, y_pred)
```

```
print(f"Accuracy: {accuracy}")


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


cm = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", cm)


# Visualize the confusion matrix (optional)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
        xticklabels=["Setosa", "Versicolor", "Virginica"],
        yticklabels=["Setosa", "Versicolor", "Virginica"])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

Program 7

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:



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

dataset = pd.read_csv('/content/sales.csv')
print(dataset.head())
weeks = dataset['x:(weeks)'].values
Sales = dataset['y: (Sales in th)'].values

X = weeks.reshape(-1,1)
y = Sales.reshape(-1,1)

X_b = np.c_[np.ones((len(X),1)),X]

theta = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)

b = theta[0]
m = theta[1]

print(f"Reg eqn is: y= {m[0]:.2f} x + {b[0]:.2f}")

predicted_sales_7 = m*7 +b
predicted_sales_9 = m*9+b

print(f"{predicted_sales_7[0]:.2f} thousand")
print(f"{predicted_sales_9[0]:.2f} thousand")
```

Code:

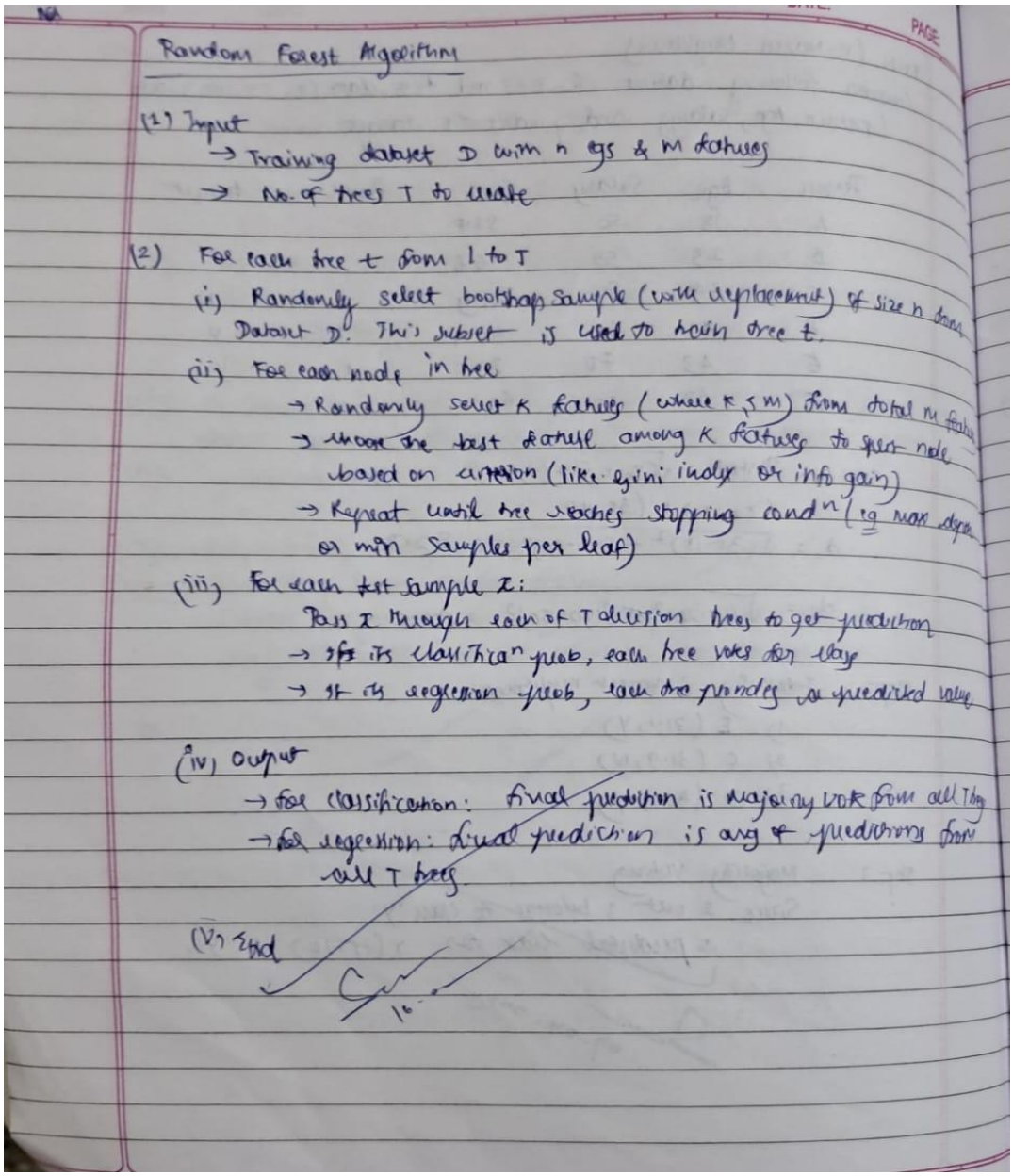
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Load the dataset from CSV
df = pd.read_csv("/content/iris (1).csv")

# Features and target
X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]
y = df['species']

# Encode species labels (setosa → 0, versicolor → 1, virginica → 2)
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

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

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

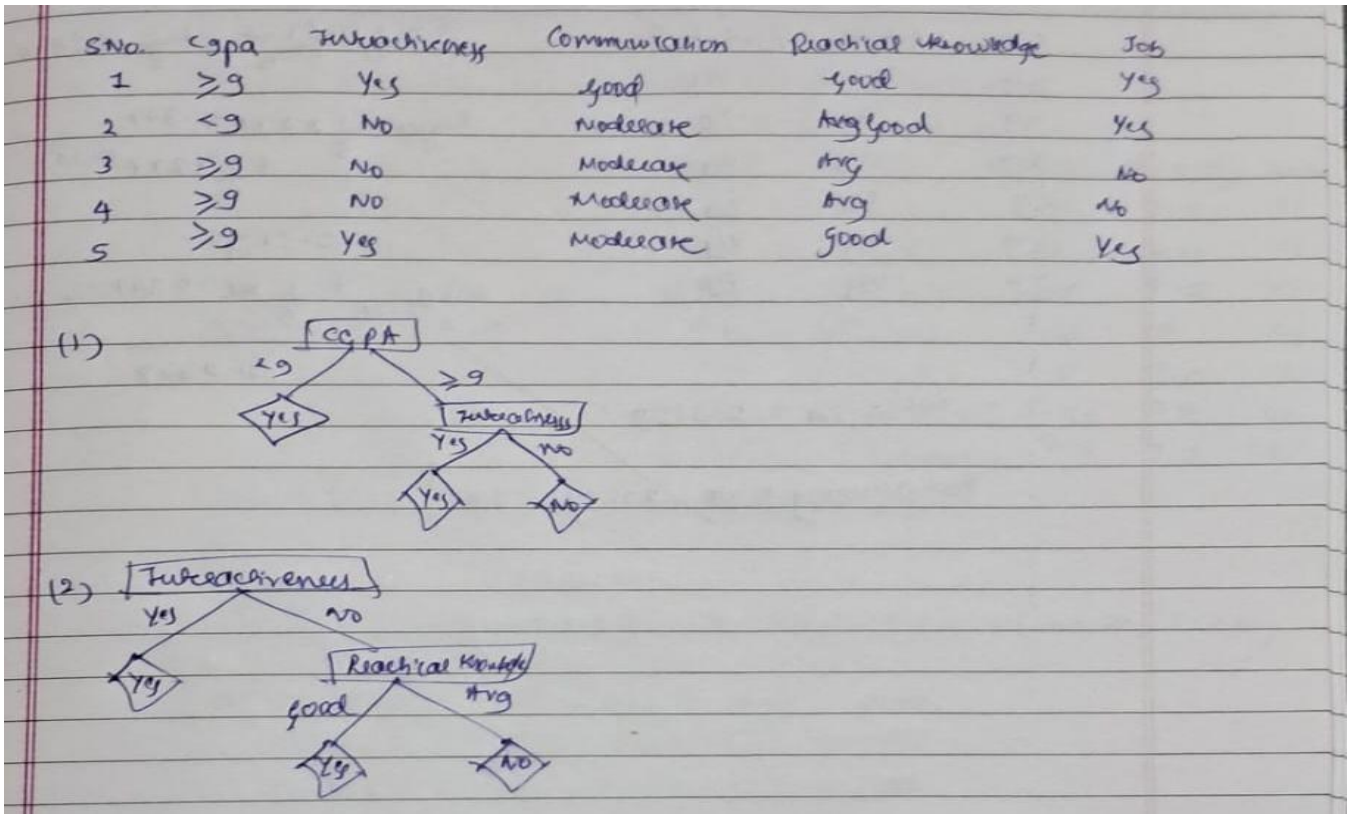
```
X_test_scaled = scaler.transform(X_test)


# Train the SVM model

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

svm_model.fit(X_train_scaled, y_train)


# Predictions

y_pred = svm_model.predict(X_test_scaled)


# Evaluation

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

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("Classification Report:\n", classification_report(y_test, y_pred,
target_names=label_encoder.classes_))
```

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot:





Code:

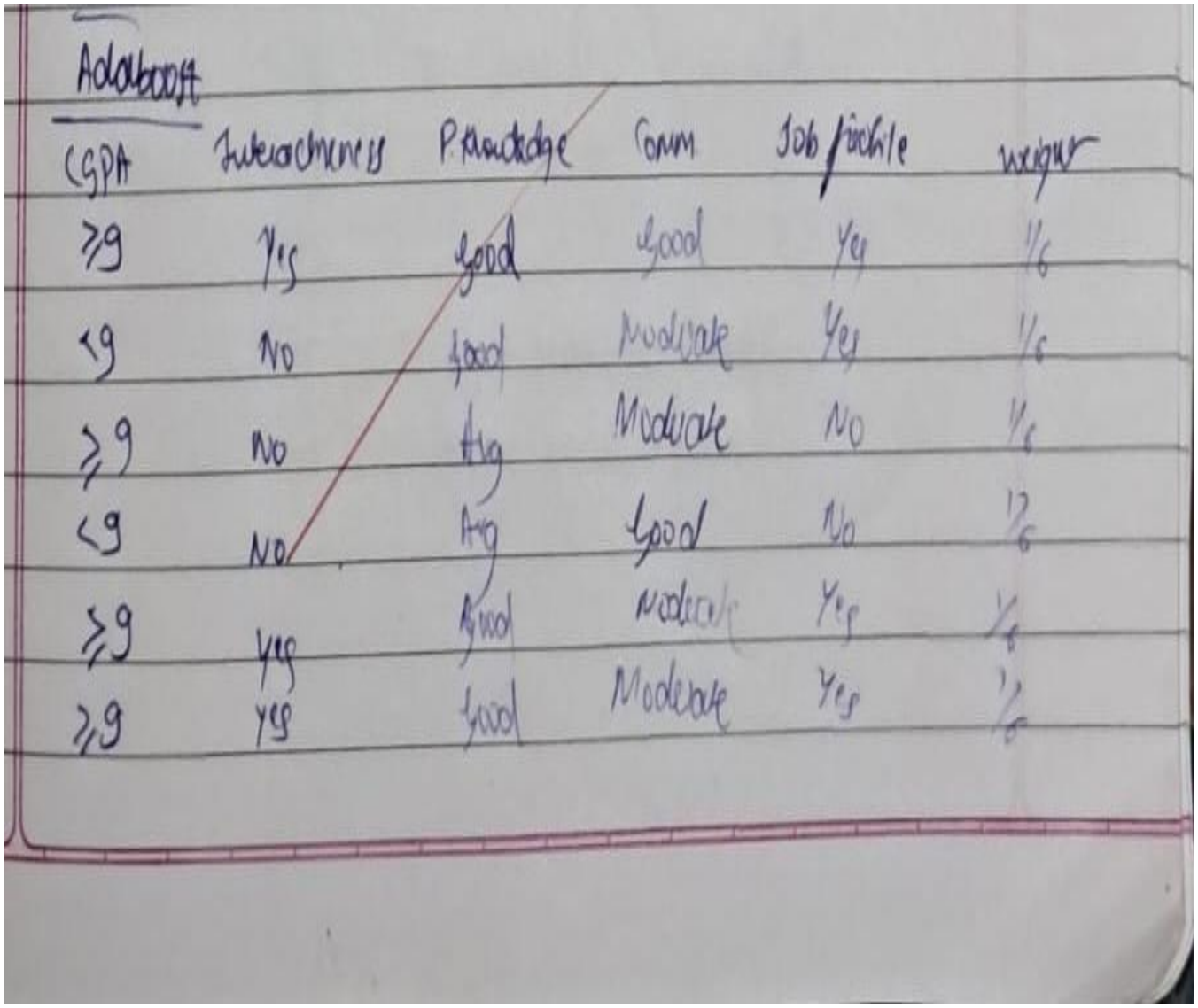
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score



df = pd.read_csv('/content/iris (3).csv')


# Prepare the data
X = df.drop('species', axis=1)
y = df['species']
```

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

# Build and evaluate the Random Forest classifier with default n_estimators
rf_classifier_default = RandomForestClassifier(random_state=42)
rf_classifier_default.fit(X_train, y_train)
y_pred_default = rf_classifier_default.predict(X_test)
default_accuracy = accuracy_score(y_test, y_pred_default)
print(f"Accuracy with default n_estimators (10): {default_accuracy}")

# Fine-tune the model by varying the number of trees
best_accuracy = 0
best_n_estimators = 0

for n_estimators in range(1, 101):  # Try different number of trees
    rf_classifier = RandomForestClassifier(n_estimators=n_estimators, random_state=42)
    rf_classifier.fit(X_train, y_train)
    y_pred = rf_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_n_estimators = n_estimators

print(f"\nBest Accuracy: {best_accuracy} achieved with n_estimators = {best_n_estimators}".
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot:



Initial wt = 1/6

if CGPA ≥ 9 ⇒ Job profile = Yes

CGPA < 9 ⇒ Job profile = No

CGPA	Actual	Predicted
≥9	Yes	Yes
<9	Yes	No
≥9	No	Yes
<9	No	No
≥9	Yes	Yes
≥9	Yes	Yes

$\varepsilon = \sum w_i = \frac{2}{6} = 0.333$

$\alpha = \frac{1}{2} \ln\left(\frac{1-\varepsilon}{\varepsilon}\right) = \frac{1}{2} 0.347$

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

$= 0.9438$

$Wt(d_i)_{iH} = \frac{\frac{1}{6} * e^{-0.347}}{0.9438} = 0.125$

$Wt(d_i)_{iH} = 0.2508$

But accuracy score = 83.3%

Code:

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

# Load the dataset
df = pd.read_csv("/content/income.csv")

# Encode the target column (income_level)
le = LabelEncoder()
df['income_level'] = le.fit_transform(df['income_level'])  # e.g., <=50K → 0, >50K → 1

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

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

# 1 □□Train AdaBoostClassifier with n_estimators = 10
model = AdaBoostClassifier(n_estimators=10, random_state=42)
```

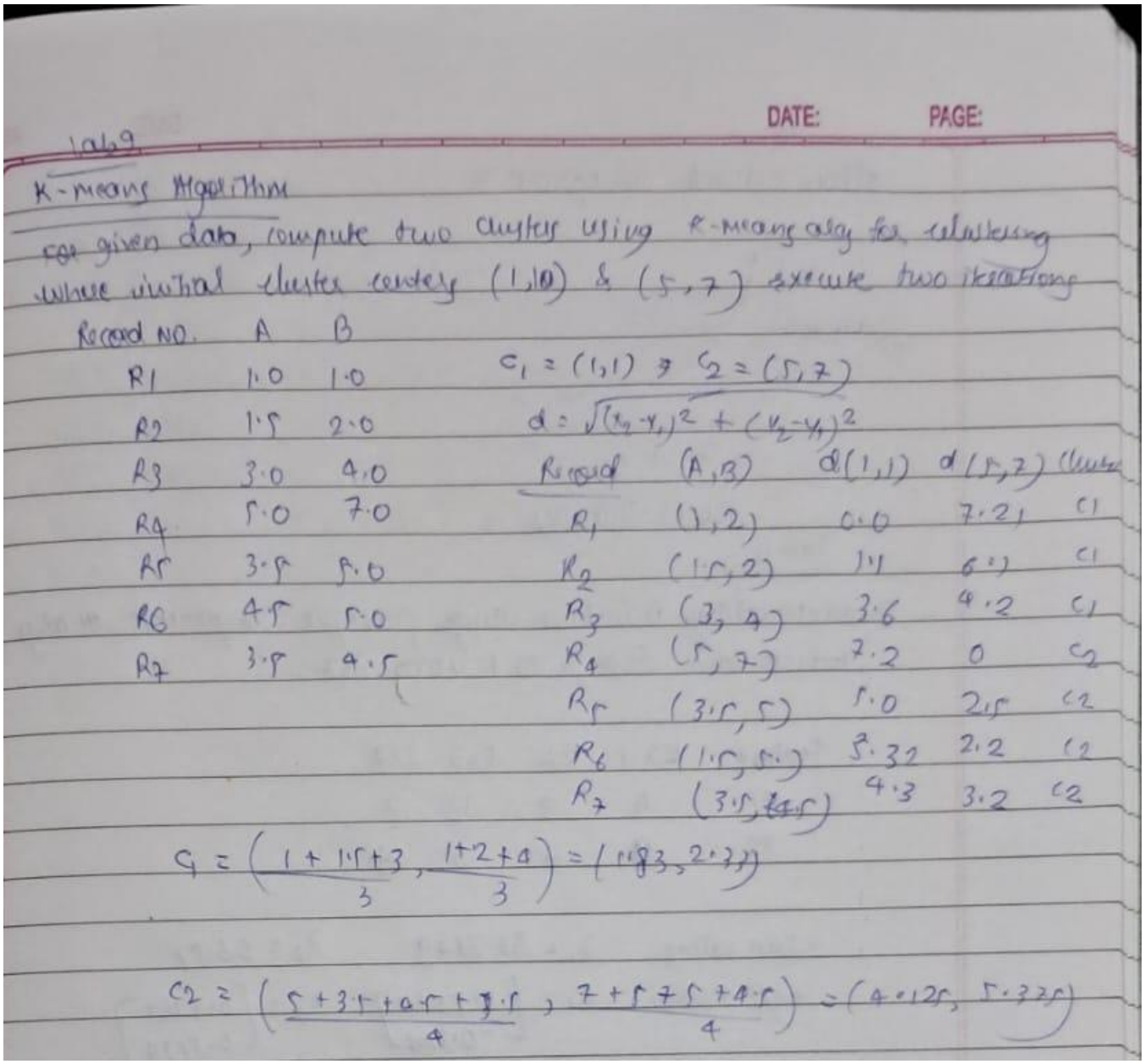
```
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy with 10 estimators: {accuracy:.4f}")


# 2 Fine-tune the model by changing n_estimators

scores = []

n_values = range(10, 101, 10)


for n in n_values:

    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}")


# 3 Find the best result

best_n = n_values[scores.index(max(scores))]

best_score = max(scores)

print(f"\n Best accuracy: {best_score:.4f} with {best_n} estimators")


# Optional: Plot the result

plt.plot(n_values, scores, marker='o')
```

```
plt.title("Accuracy vs Number of Estimators")

plt.xlabel("n_estimators (number of trees)")

plt.ylabel("Accuracy")

plt.grid(True)

plt.show()
```

Program 10

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

Screenshot:



lab9

DATE: PAGE:

K-means Algorithm

For given data, compute two clusters using K-means algo for clustering where initial cluster centers (1,1) & (5,7) execute two iterations

Record NO.	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

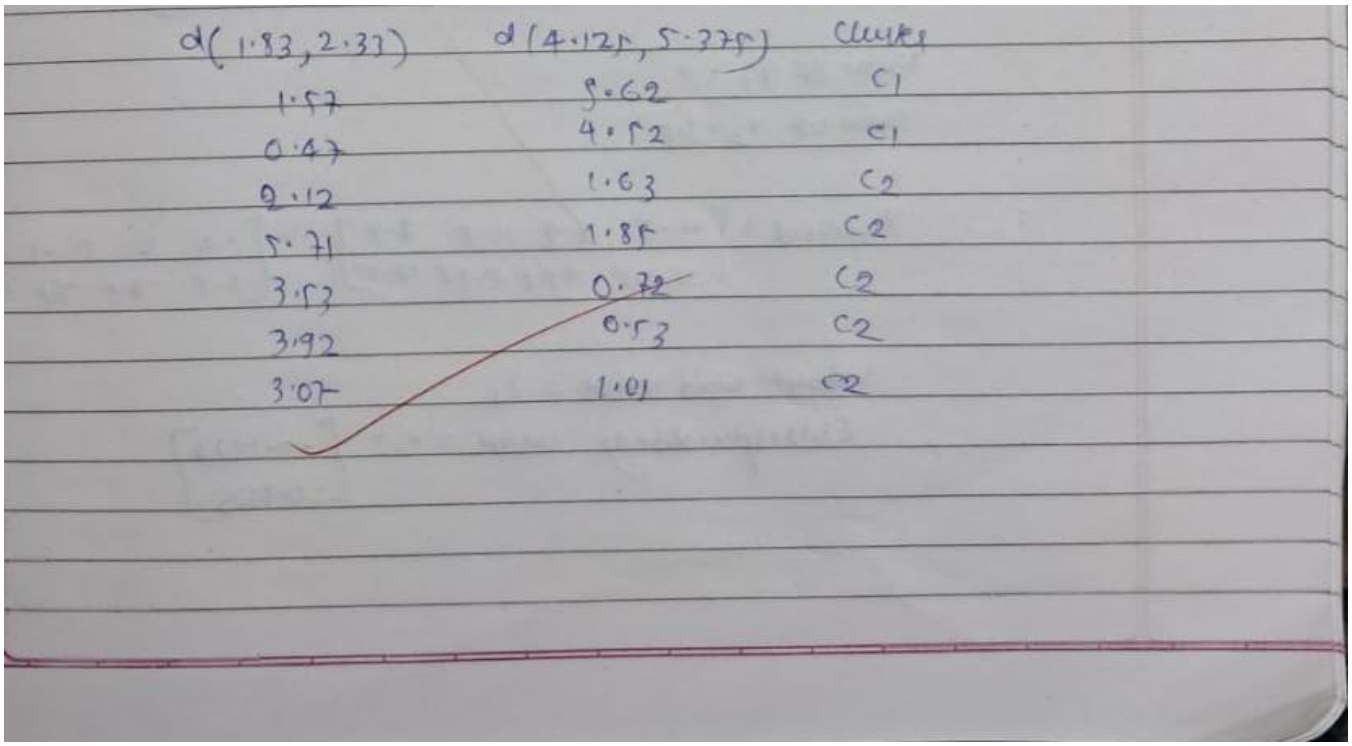
$C_1 = (1,1)$ & $C_2 = (5,7)$

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

Record	(A,B)	d(1,1)	d(5,7)	Cluster
R_1	(1,2)	0.0	7.21	C1
R_2	(1.5,2)	1.1	6.1	C1
R_3	(3,4)	3.6	4.2	C1
R_4	(5,7)	7.2	0	C2
R_5	(3.5,5)	5.0	2.5	C2
R_6	(4.5,5.)	5.32	2.2	C2
R_7	(3.5,4.5)	4.3	3.2	C2

$C_1 = \left(\frac{1+1.5+3}{3}, \frac{1+2+4}{3}\right) = (1.83, 2.33)$

$C_2 = \left(\frac{5+3.5+4.5+3.5}{4}, \frac{7+5+5+4.5}{4}\right) = (4.125, 5.375)$



Code:

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

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

# 2. Drop Sepal features, keep only Petal length and width
X = df[['petal_length', 'petal_width']]

# 3. Scaling (K-Means is distance-based, scaling helps!)
```

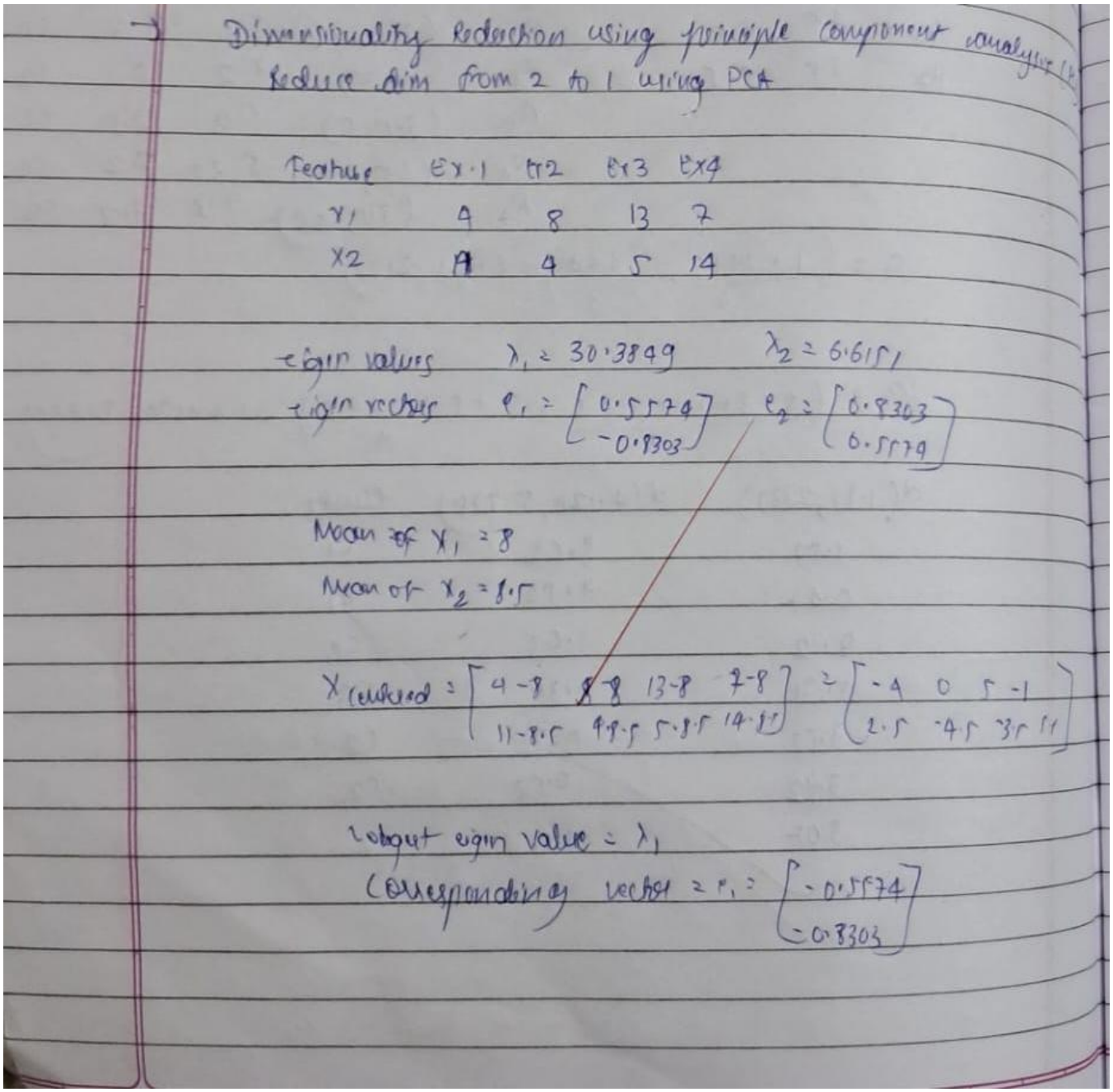
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 4. Elbow Method to find 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, 'bo-')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (Within-cluster sum of squares)')
plt.title('Elbow Method for Optimal k')
plt.grid(True)
plt.show()

# 5. Choose optimal k (usually at 'elbow' point, say k=3)
optimal_k = 3
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
```

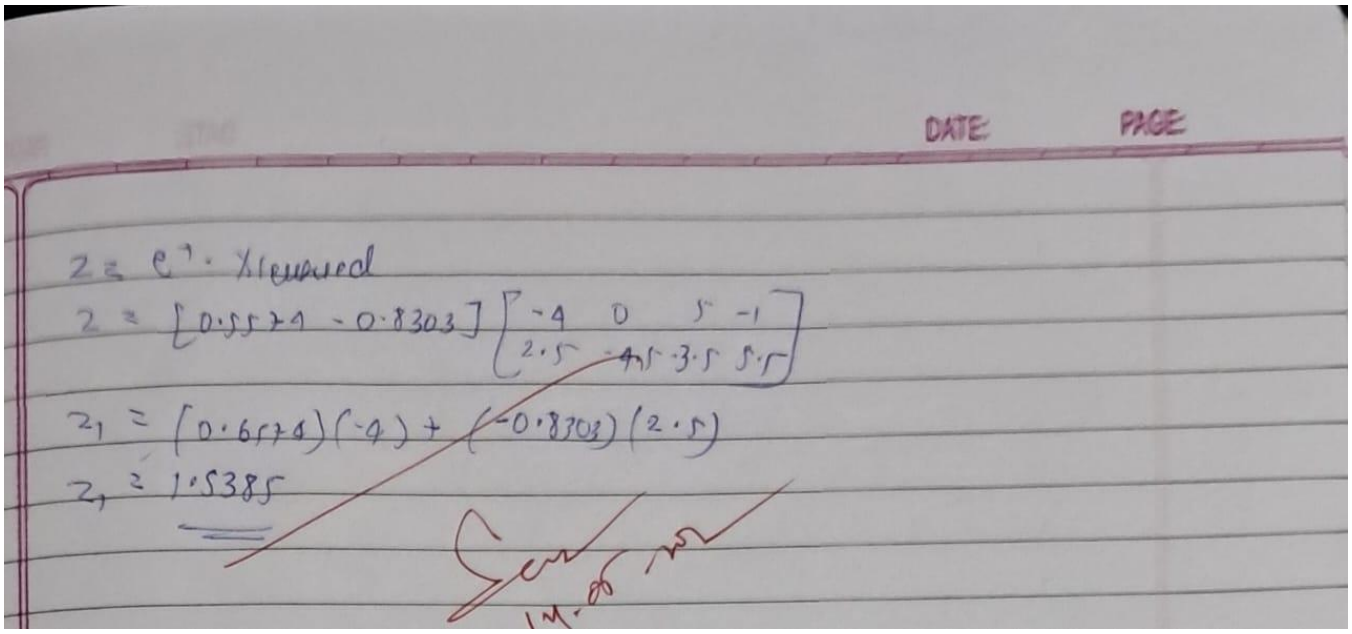
```
clusters = kmeans.fit_predict(X_scaled)


# 6. Add clusters to original dataframe for visualization
df['Cluster'] = clusters


# Plot clusters
plt.figure(figsize=(8,5))
for i in range(optimal_k):
    cluster_data = df[df['Cluster'] == i]
    plt.scatter(cluster_data['petal_length'], cluster_data['petal_width'], label=f'Cluster {i}')


plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.title('K-Means Clustering of Iris (Petal Features)')
plt.legend()
plt.grid(True)
plt.show()
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot:





Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score

from sklearn.decomposition import PCA


# Load the dataset

df = pd.read_csv('/content/heart (2).csv')


# Detect and encode categorical columns

text_cols = df.select_dtypes(include=['object']).columns.tolist()
```

```
for col in text_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col].astype(str))

# Use correct target column
target_col = 'HeartDisease'
X = df.drop(columns=[target_col])
y = df[target_col]

# Scale the 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)

# Models
models = {
    'SVM': SVC(),
    'Logistic Regression': LogisticRegression(),
    'Random Forest': RandomForestClassifier()
}

# Evaluate without PCA
```

```
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    results[name] = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy: {results[name]:.4f}")

# Apply PCA
pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)

# Evaluate after PCA
pca_results = {}
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    y_pred = model.predict(X_test_pca)
    pca_results[name] = accuracy_score(y_test, y_pred)
    print(f"{name} Accuracy after PCA: {pca_results[name]:.4f}")

# Final comparison
print("\nComparison (Original vs PCA):")
for name in models:
    print(f"{name}: Original = {results[name]:.4f}, PCA = {pca_results[name]:.4f}")
```