**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum- 590014, Karnataka.**

****

**LAB REPORT**

**on**

**Machine Learning (23CS6PCMAL)**

***Submitted by***

**Nischal Kiran (1BM22CS182)**

***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**

**February 2025 – June 25**

**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 **Nischal Kiran (1BM22CS182),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Laboratory report has been approved as it satisfies the academic requirements in respect of a Machine Learning (23CS6PCMAL) work prescribed for the said degree.



|  |  |
| --- | --- |
|  |  |

**INDEX**

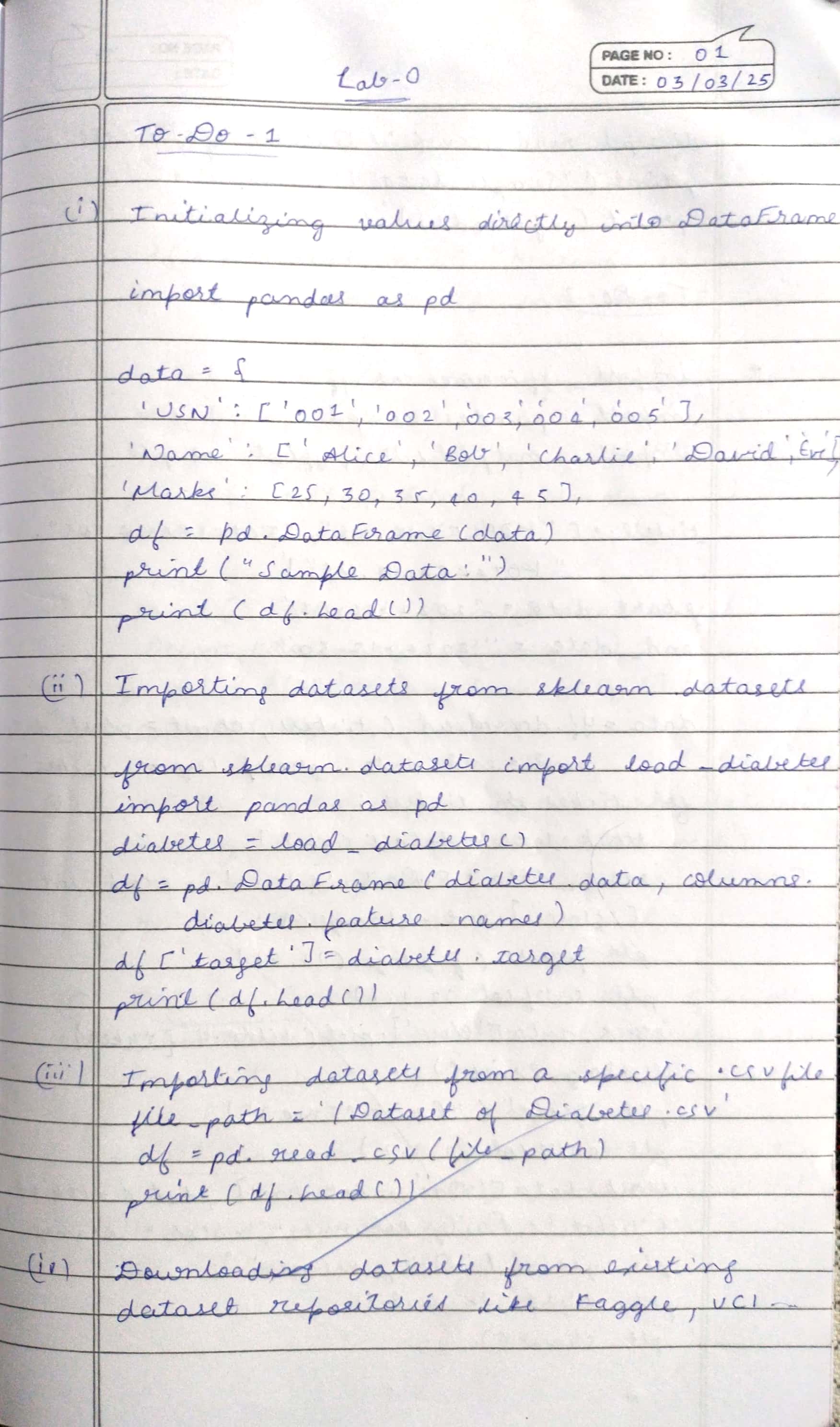
|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.**  **No.** | **Date** | **Experiment Title** | **Page No.** |
| 1 | 21-02-2025 | Write a python program to import and export data using pandas library functions. | 1 |
| 2 | 03-03-2025 | Demonstrate various data pre-processing techniques for a given dataset. | 6 |
| 3 | 10-03-2025 | Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample. | 11 |
| 4 | 17-03-2025 | Implement Linear and Multi-Linear Regression algorithm using appropriate dataset. | 20 |
| 5 | 24-03-2025 | Build Logistic Regression Model for a given dataset. | 24 |
| 6 | 07-04-2025 | Build KNN Classification model for a given dataset. | 26 |
| 7 | 07-04-2025 | Build Support vector machine model for a given dataset. | 30 |
| 8 | 21-04-2025 | Implement Random forest ensemble method on a given dataset. | 34 |
| 9 | 05-05-2025 | Implement Boosting ensemble method on a given dataset. | 37 |
| 10 | 05-05-2025 | Build k-Means algorithm to cluster a set of data stored in a .CSV file. | 40 |
| 11 | 12-05-2025 | Implement Dimensionality reduction using Principle Component Analysis (PCA) method. | 44 |

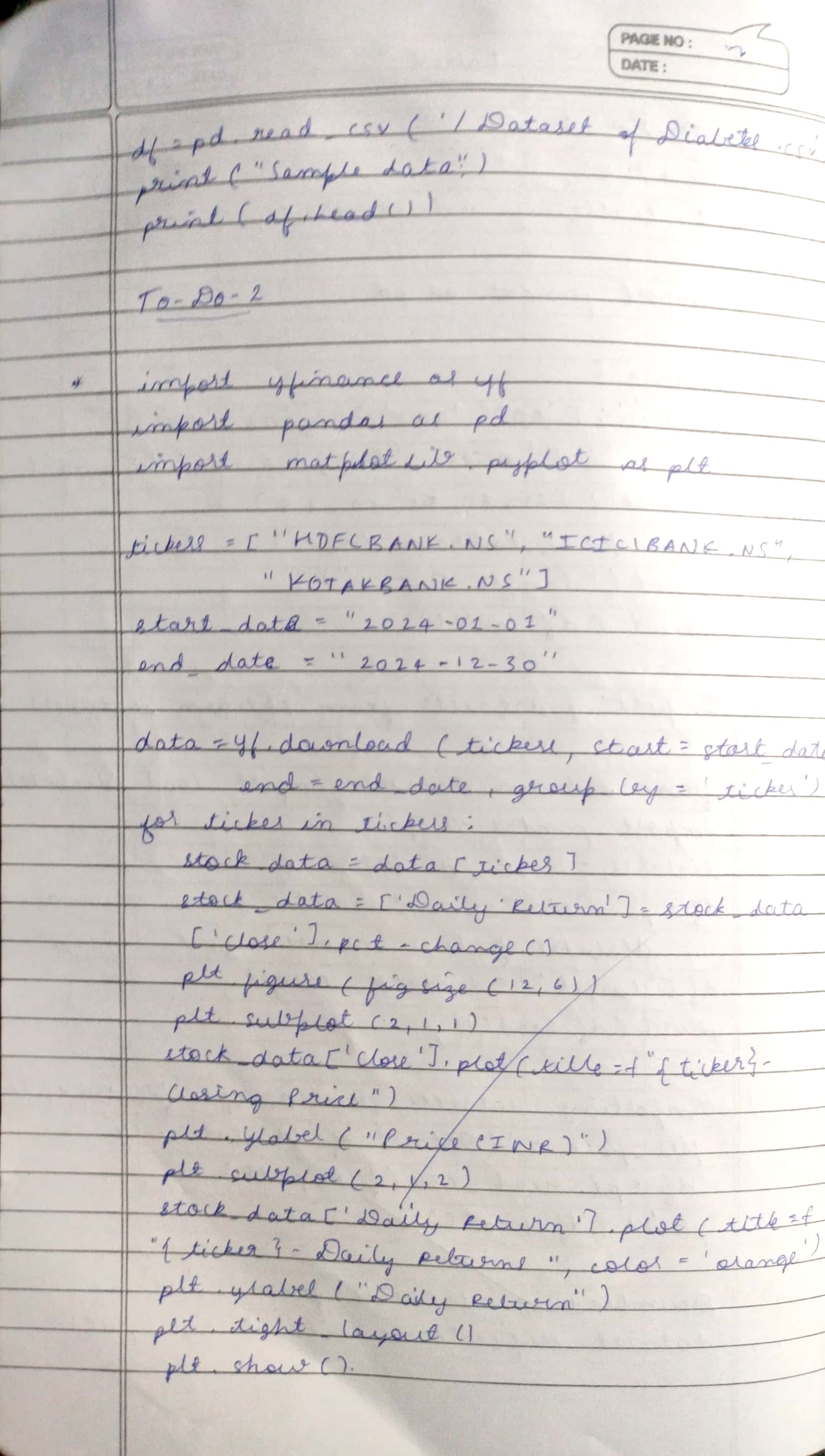
Github Link: https://github.com/nischal-kiran/ML

**Program 1**

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

**Screenshot:**

****



**Code:**

from sklearn.datasets import load\_iris import pandas as pd

iris = load\_iris()

df = pd.DataFrame(iris.data, columns=iris.feature\_names) df.head()

df['target'] = iris.target

df

import kagglehub

# Download latest version

path = kagglehub.dataset\_download("abdulmalik1518/mobiles-dataset-2025")

print("Path to dataset files:", path)

df = pd.read\_csv("/content/Mobiles\_Dataset\_(2025).csv", encoding='latin-1') # or 'ISO-8859-1', or 'cp1252'

df.head() df['Company Name']

data = {"USN" : ['1', "2", "3"], "Name" : ["A", "B", "C"]}

df = pd.DataFrame(data) df

from sklearn.datasets import load\_diabetes diabetes = load\_diabetes()

df = pd.DataFrame(diabetes.data, columns=diabetes.feature\_names) df.head()

df.columns

df = pd.read\_csv("/content/Dataset\_of\_Diabetes .csv") df.head()

import yfinance as yf import pandas as pd

import matplotlib.pyplot as plt

tickers = ["RELIANCE.NS", "TCS.NS", "INFY.NS"]

# Fetch historical data for the last 1 year

data = yf.download(tickers, start="2022-10-01", end="2023-10-01", group\_by='ticker')

# Display the first 5 rows of the dataset

print("First 5 rows of the dataset:")

print(data.head())

print("\nShape of the dataset:")

print(data.shape)

# Summary statistics for a specific stock (e.g., Reliance)

reliance\_data = data['RELIANCE.NS']

print("\nSummary statistics for Reliance Industries:")

print(reliance\_data.describe())

# Calculate daily returns

reliance\_data['Daily Return'] = reliance\_data['Close'].pct\_change()

# Plot the closing price and daily returns

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

plt.subplot(2, 1, 1)

reliance\_data['Close'].plot(title="Reliance Industries - Closing Price")

plt.subplot(2, 1, 2)

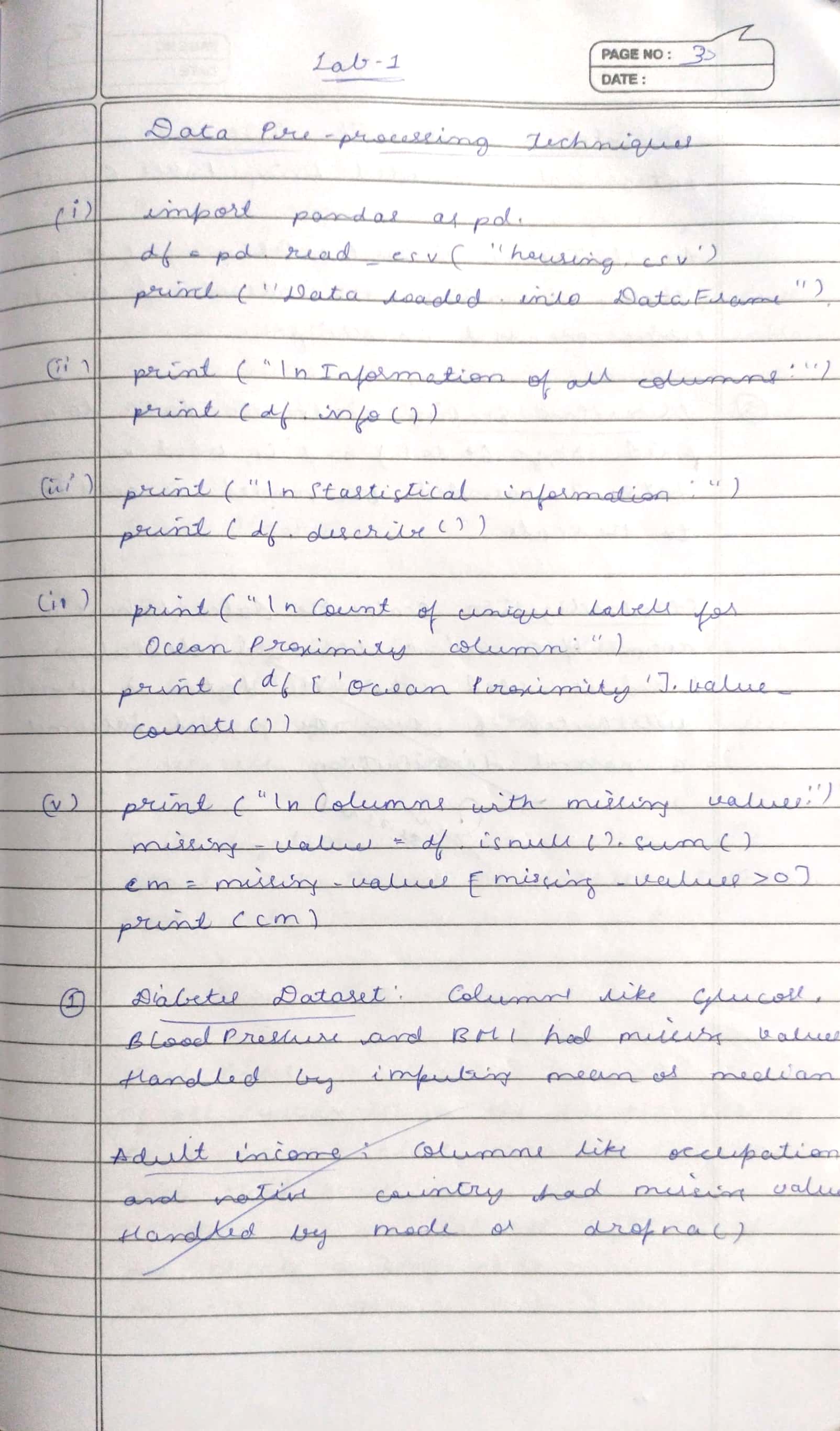
reliance\_data['Daily Return'].plot(title="Reliance Industries - Daily Returns", color='orange')

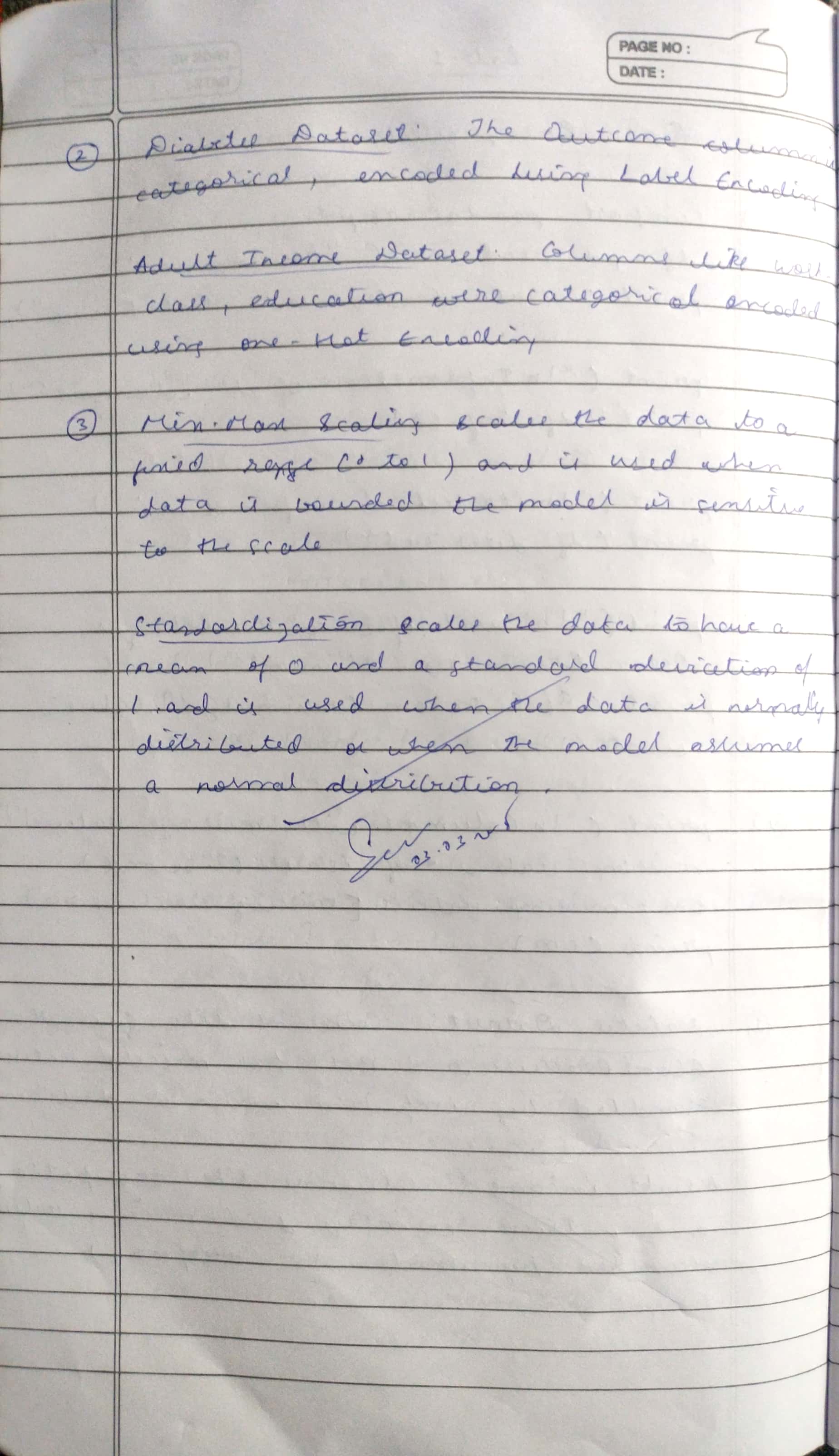
plt.tight\_layout() plt.show()

**Program 2**

Demonstrate various data pre-processing techniques for a given dataset

**Screenshot:**

****



**Code:**

import pandas as pd import numpy as np

# Load dataset

df = pd.read\_csv("data.csv") print(df.head())

# Check missing values print(df.isnull().sum())

# Drop rows with missing values df\_cleaned = df.dropna()

# Or fill missing values with mean/median df['Age'].fillna(df['Age'].mean(), inplace=True) df['Salary'].fillna(df['Salary'].median(), inplace=True)

# For nominal categories

df = pd.get\_dummies(df, columns=['Gender', 'Country'], drop\_first=True)

# For ordinal categories

from sklearn.preprocessing import OrdinalEncoder encoder = OrdinalEncoder()

df[['Education\_Level']] = encoder.fit\_transform(df[['Education\_Level']])

from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Standardization (Z-score) scaler = StandardScaler()

df[['Age', 'Salary']] = scaler.fit\_transform(df[['Age', 'Salary']])

# Min-Max Normalization minmax = MinMaxScaler()

df[['Age', 'Salary']] = minmax.fit\_transform(df[['Age', 'Salary']])

# Using IQR method

Q1 = df['Salary'].quantile(0.25) Q3 = df['Salary'].quantile(0.75)

IQR = Q3 - Q1

df = df[(df['Salary'] >= Q1 - 1.5\*IQR) & (df['Salary'] <= Q3 + 1.5\*IQR)]

df['Age\_Salary\_Ratio'] = df['Age'] / df['Salary']

# Drop irrelevant columns

df.drop(['User\_ID', 'Name'], axis=1, inplace=True)

# Correlation-based filtering correlation\_matrix = df.corr()

print(correlation\_matrix)

from sklearn.model\_selection import train\_test\_split

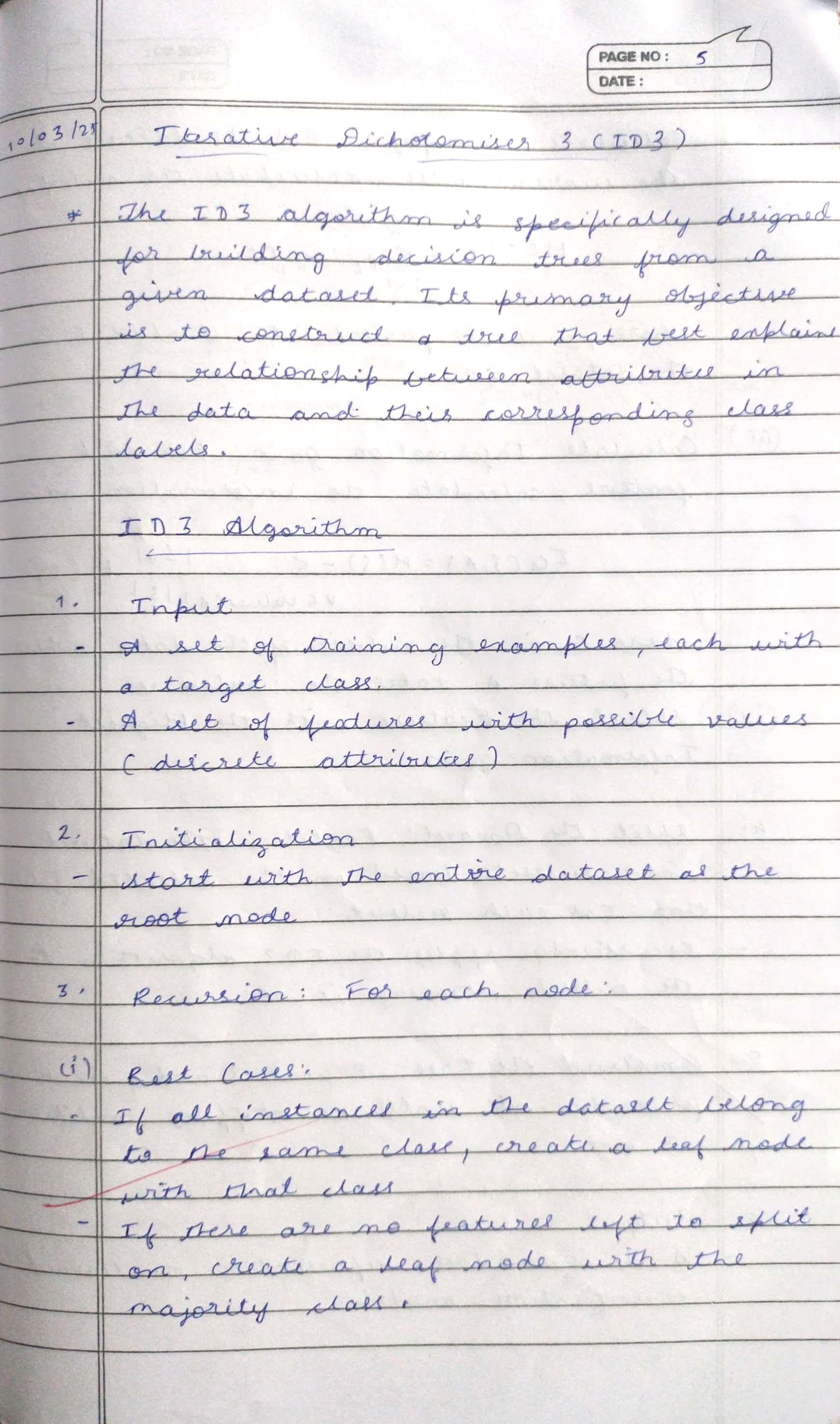
X = df.drop('Purchased', axis=1) y = df['Purchased']

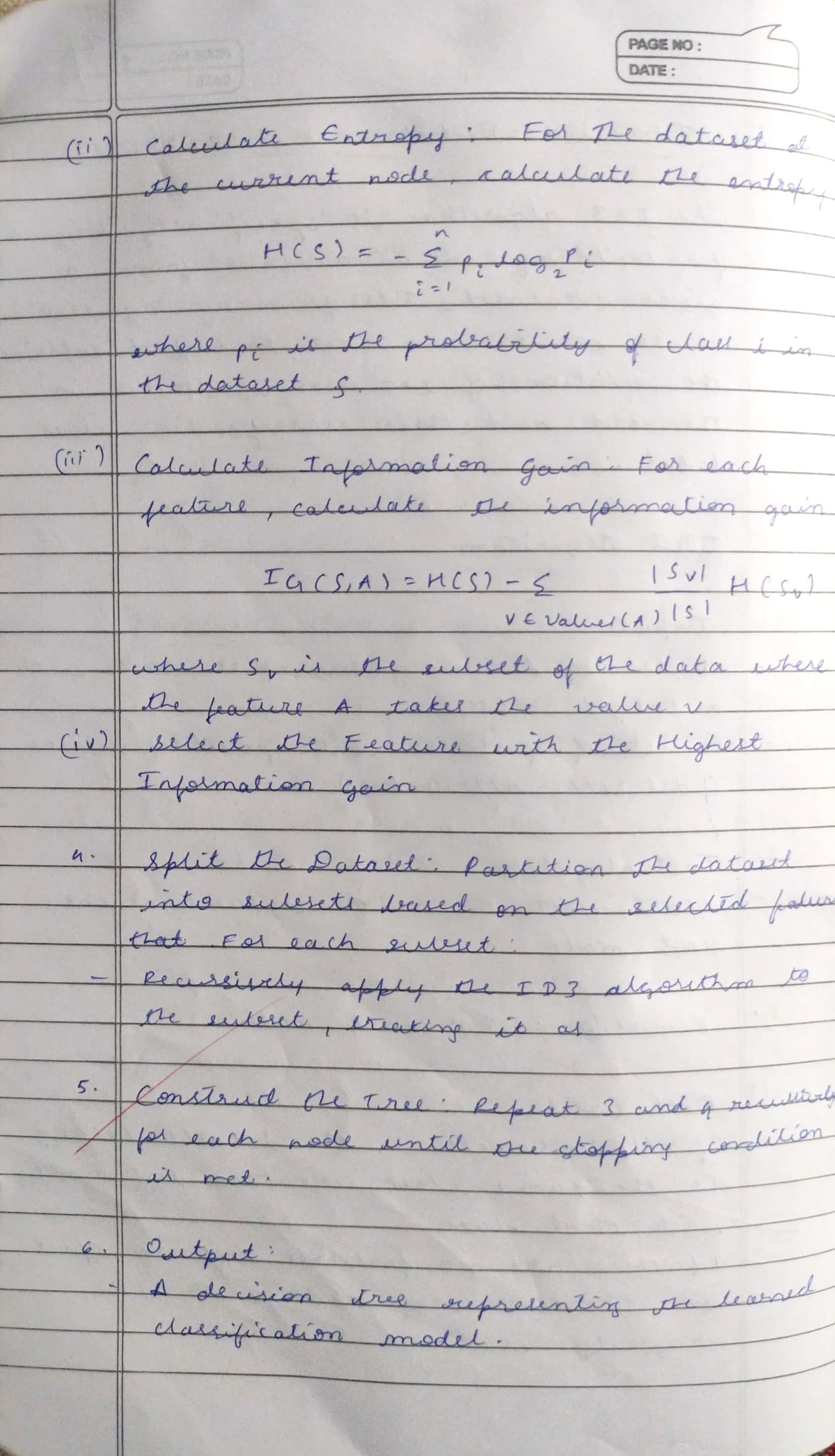
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

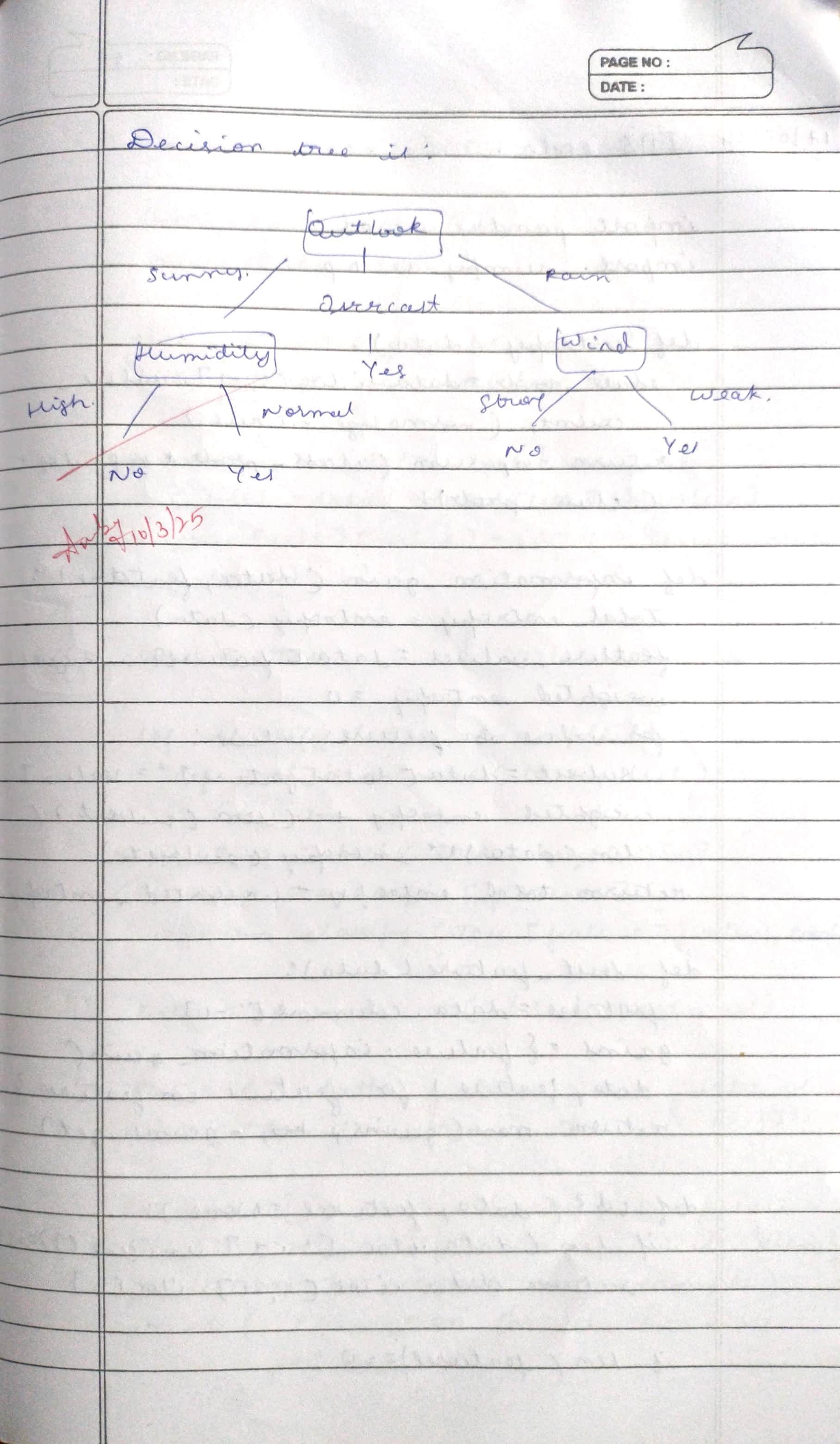
**Program 3**

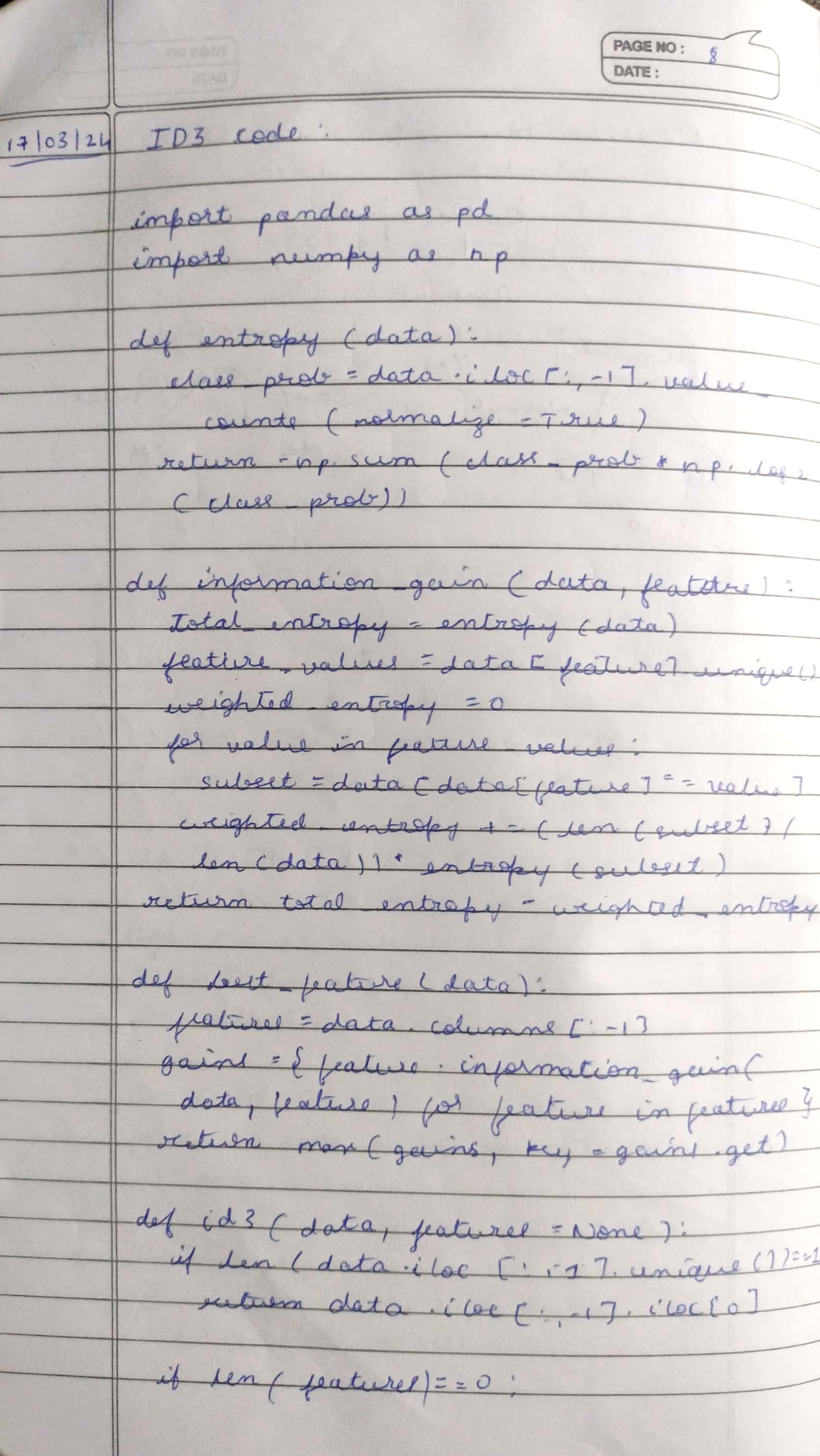
Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample

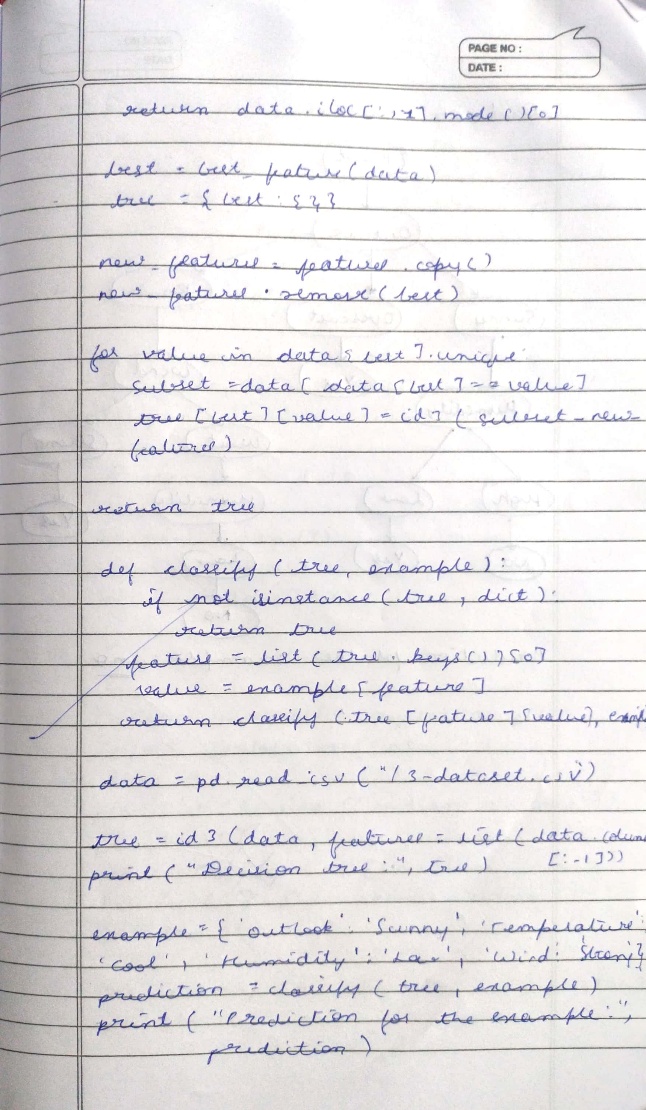
**Screenshot:**

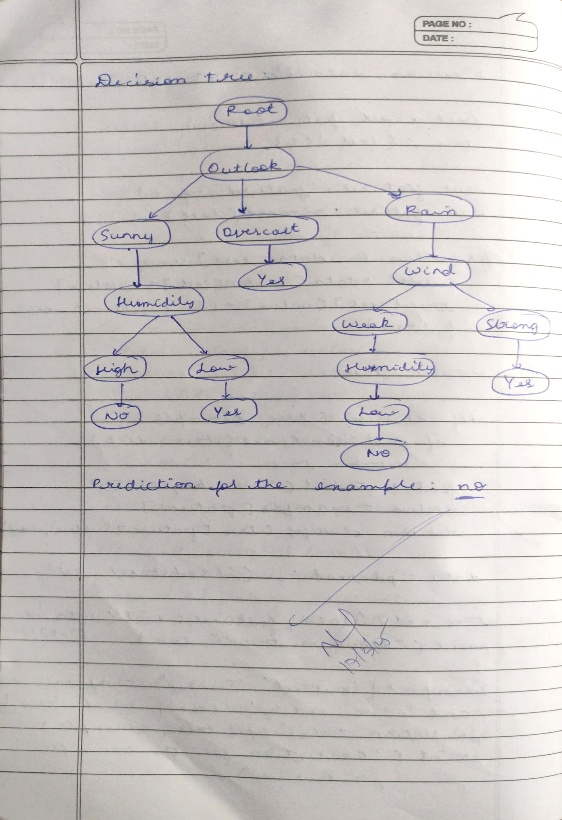
****









****

**Code:**

import pandas as pd import numpy as np

from graphviz import Digraph

# Calculate Entropy def entropy(data):

class\_probabilities = data.iloc[:, -1].value\_counts(normalize=True) return -np.sum(class\_probabilities \* np.log2(class\_probabilities))

# Calculate Information Gain

def information\_gain(data, feature): total\_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 total\_entropy - weighted\_entropy

# Find the best feature to split the data def best\_feature(data):

features = data.columns[:-1] # Exclude the target column

gains = {feature: information\_gain(data, feature) for feature in features}

return max(gains, key=gains.get)

# Create the decision tree

def id3(data, features=None):

if len(data.iloc[:, -1].unique()) == 1: # All data points belong to the same class return data.iloc[:, -1].iloc[0]

if len(features) == 0: # No more features to split on return data.iloc[:, -1].mode()[0]

best = best\_feature(data) tree = {best: {}}

new\_features = features.copy() new\_features.remove(best)

for value in data[best].unique(): subset = data[data[best] == value]

tree[best][value] = id3(subset, new\_features)

return tree

# Function to classify new examples based on the decision tree def classify(tree, example):

if not isinstance(tree, dict): return tree

feature = list(tree.keys())[0] value = example[feature]

return classify(tree[feature][value], example)

# Function to visualize the decision tree using Graphviz

def create\_tree\_diagram(tree, dot=None, parent\_name="Root", parent\_value=""): if dot is None:

dot = Digraph(format="png", engine="dot")

if isinstance(tree, dict): # Tree node for feature, branches in tree.items():

feature\_name = f"{parent\_name}\_{feature}" dot.node(feature\_name, feature)

dot.edge(parent\_name, feature\_name, label=parent\_value)

for value, subtree in branches.items(): value\_name = f"{feature\_name}\_{value}" dot.node(value\_name, f"{feature}: {value}")

dot.edge(feature\_name, value\_name, label=str(value)) # Recurse for each subtree

create\_tree\_diagram(subtree, dot, value\_name, str(value)) else: # Leaf node

dot.node(parent\_name + "\_class", f"Class: {tree}") dot.ede(parent\_name, parent\_name + "\_class", label="Leaf")

return dot

# Example usage

data = pd.DataFrame({

'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain', 'Overcast', 'Sunny', 'Sunny', 'Rain', 'Sunny', 'Overcast', 'Overcast', 'Rain'],

'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool', 'Cool', 'Mild', 'Cool', 'Mild', 'Mild', 'Mild', 'Hot', 'Mild'],

'Humidity': ['High', 'High', 'High', 'High', 'High', 'Low', 'Low', 'High', 'Low', 'Low', 'Low', 'High', 'Low', 'High'],

'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Weak', 'Strong', 'Weak', 'Weak', 'Strong', 'Strong', 'Weak', 'Strong', 'Weak'],

'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']

})

# Train the decision tree

tree = id3(data, features=list(data.columns[:-1])) print("Decision Tree:", tree)

# Classify a new example

example = {'Outlook': 'Sunny', 'Temperature': 'Cool', 'Humidity': 'Low', 'Wind': 'Strong'} prediction = classify(tree, example)

print("Prediction for the example:", prediction)

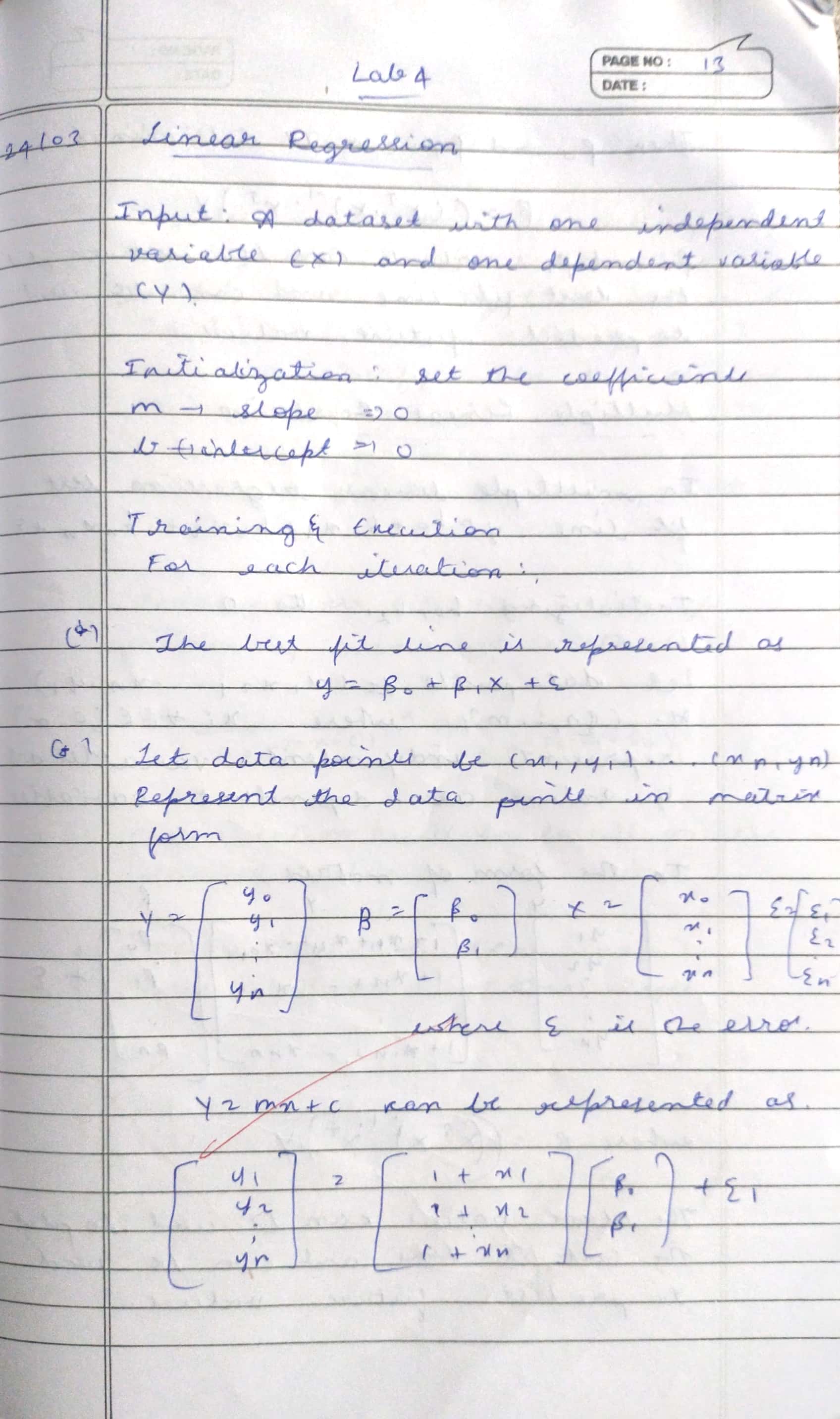
# Visualize the decision tree dot = create\_tree\_diagram(tree)

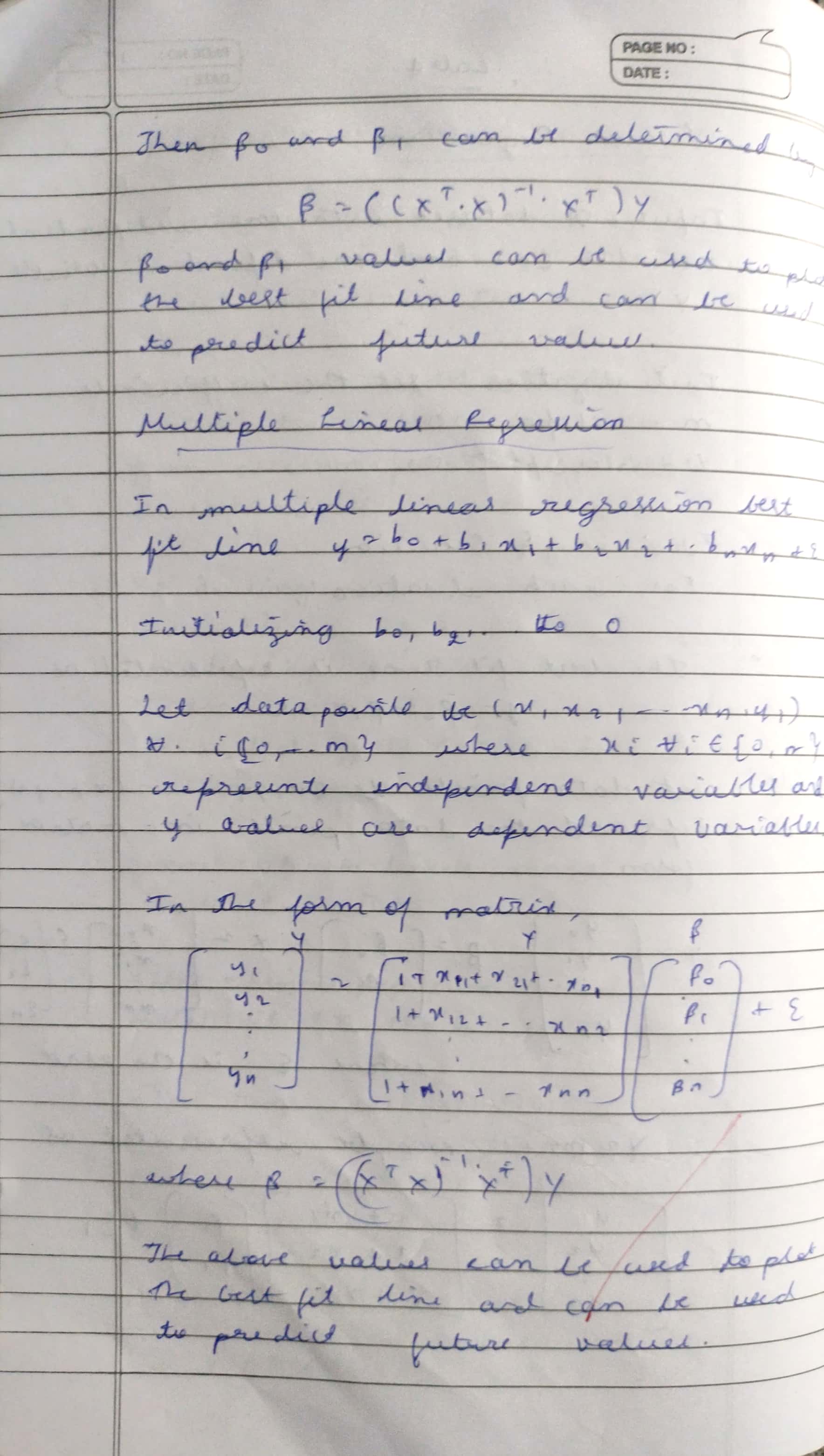
dot.render("decision\_tree", view=True) # This will generate and open the tree diagram

**Program 4**

Implement Linear and Multi-Linear Regression algorithm for appropriate dataset

**Screenshot:**

****



**Code:**

**Linear Regression**

import pandas as pd

df = pd.read\_csv("/content/tvmarketing.csv") df

# Visualise the relationship between the features and the response using scatterplots df.plot(x='TV',y='Sales',kind='scatter')

from sklearn.model\_selection import train\_test\_split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(df['TV'], df['Sales'], test\_size=0.2, random\_state=42)

from sklearn.linear\_model import LinearRegression model = LinearRegression() model.fit(x\_train.values.reshape(-1, 1), y\_train) y\_train

model.coef\_ model.intercept\_

**MultiLinearRegression**

import pandas as pd # Step 2 : import data

house = pd.read\_csv('https://github.com/YBIFoundation/Dataset/raw/main/Boston.csv')

# display first 5 rows house.head()

y = house['MEDV']

X = house.drop(['MEDV'],axis=1)

# Step 4 : train test split

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, train\_size=0.7, random\_state=2529)

# Step 5 : select model

from sklearn.linear\_model import LinearRegression model = LinearRegression()

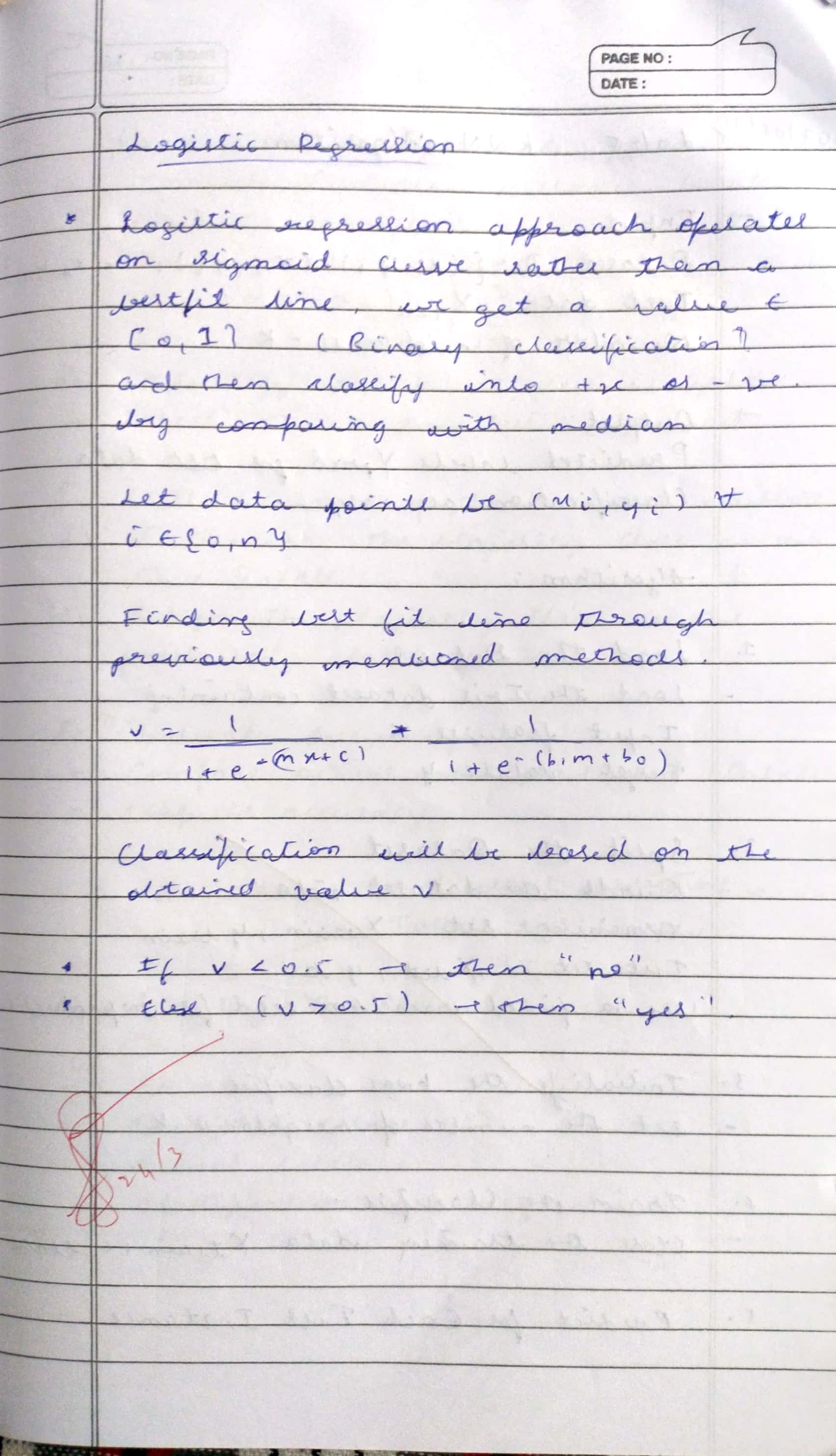
# Step 6 : train or fit model model.fit(X\_train,y\_train) model.intercept\_

model.coef\_

**Program 5**

Build Logistic Regression Model for a given dataset

**Screenshot:**

****

**Code:**

from sklearn.linear\_model import LogisticRegression from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score

# Load sample dataset (binary classification - Iris with only 2 classes) iris = load\_iris()

X = iris.data[iris.target != 2] y = iris.target[iris.target != 2]

# Train/Test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

# Logistic Regression model model = LogisticRegression() model.fit(X\_train, y\_train)

# Predict and evaluate

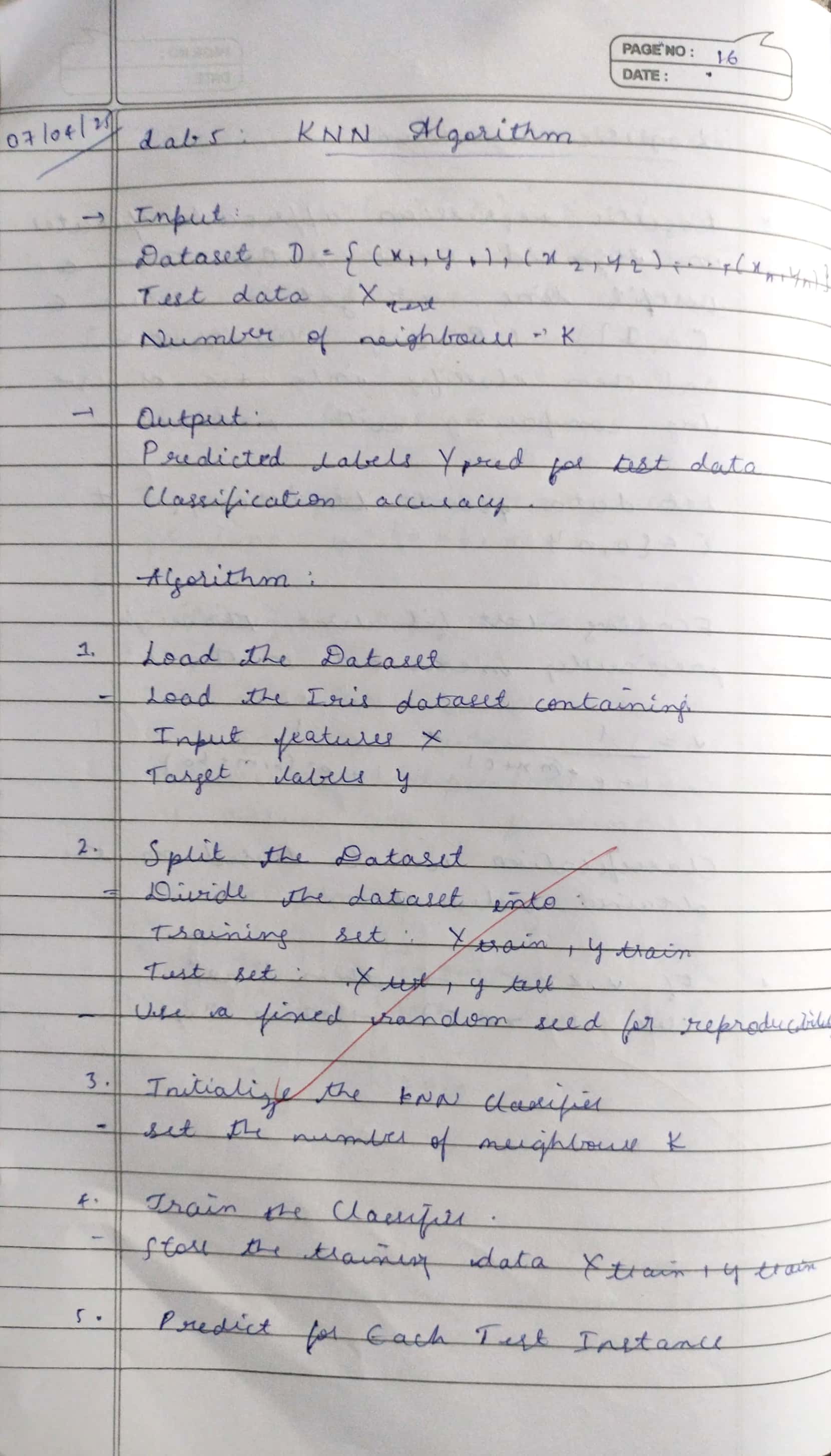
y\_pred = model.predict(X\_test)

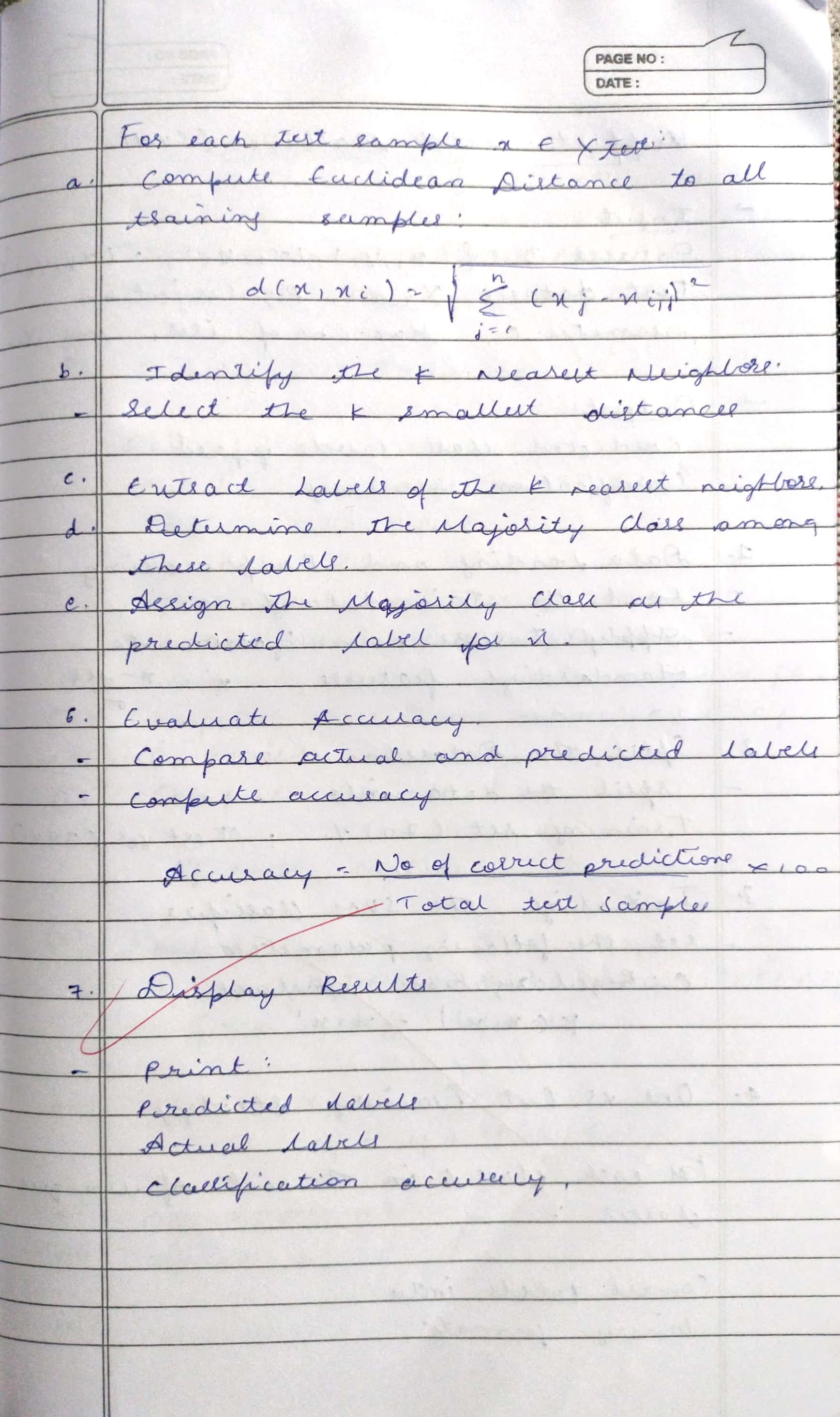
print("Accuracy:", accuracy\_score(y\_test, y\_pred))

**Program 6**

Build KNN Classification model for a given dataset

**Screenshot:**

****



**Code:**

**KNN**

import numpy as np

from collections import Counter

class KNN:

def init (self, k=3): self.k = k

def fit(self, X, y): self.X\_train = np.array(X) self.y\_train = np.array(y)

def euclidean\_distance(self, x1, x2): return np.sqrt(np.sum((x1 - x2) \*\* 2))

def predict(self, X):

predictions = [self.\_predict(x) for x in X] return np.array(predictions)

def \_predict(self, x):

# Compute distances to all training points

distances = [self.euclidean\_distance(x, x\_train) for x\_train in self.X\_train]

# Get indices of k nearest neighbors k\_indices = np.argsort(distances)[:self.k]

# Get the labels of those neighbors

k\_nearest\_labels = [self.y\_train[i] for i in k\_indices]

# Return the most common label

most\_common = Counter(k\_nearest\_labels).most\_common(1) return most\_common[0][0]

# Sample dataset (like a mini version of Iris) X\_train = [[1, 2], [2, 3], [3, 1], [6, 5], [7, 7], [8, 6]]

y\_train = [0, 0, 0, 1, 1, 1]

# Test data

X\_test = [[5, 5], [1, 1]]

# Using the KNN modelh knn = KNN(k=3) knn.fit(X\_train, y\_train)

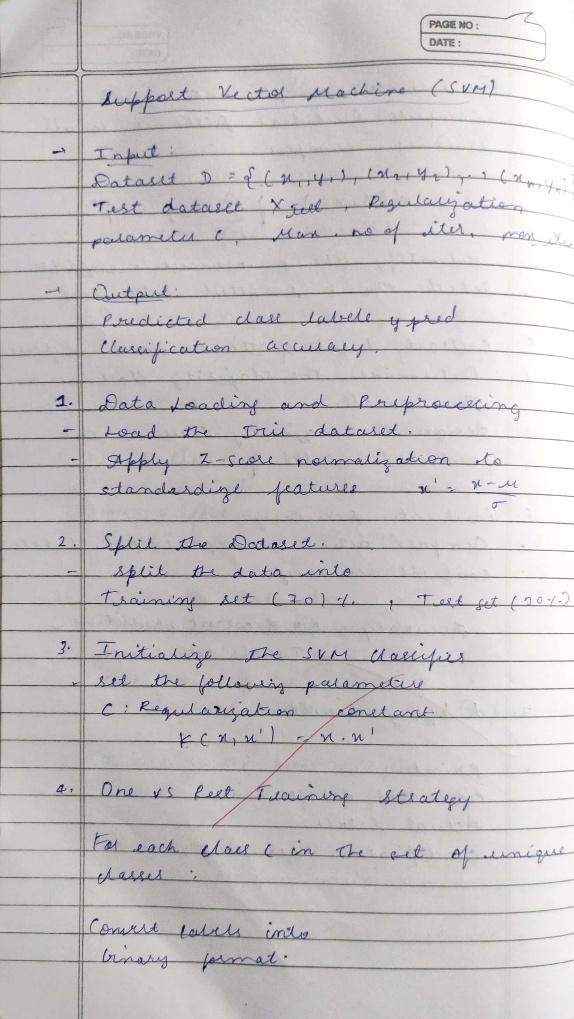
predictions = knn.predict(X\_test)

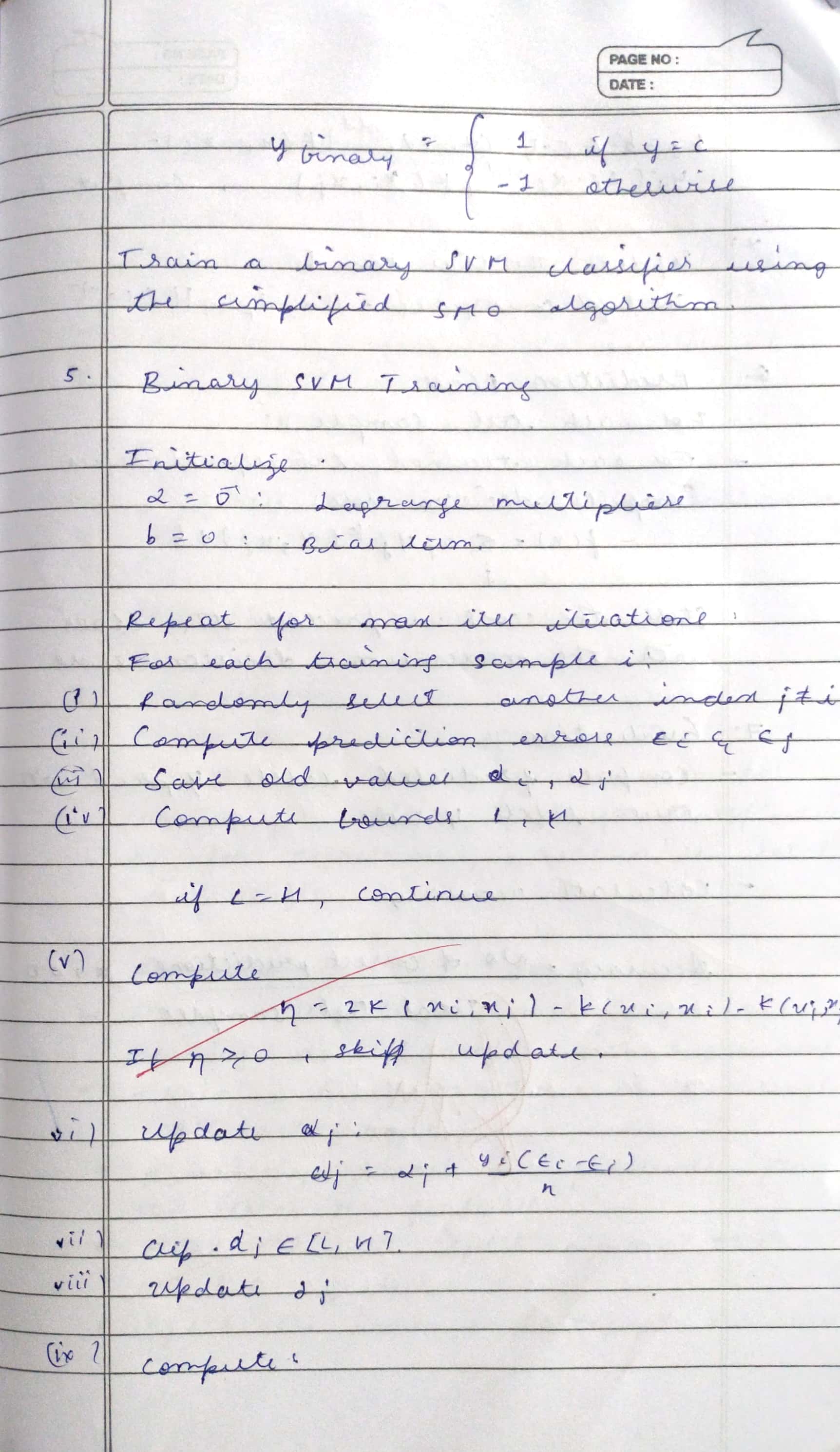
print("Predictions:", predictions)

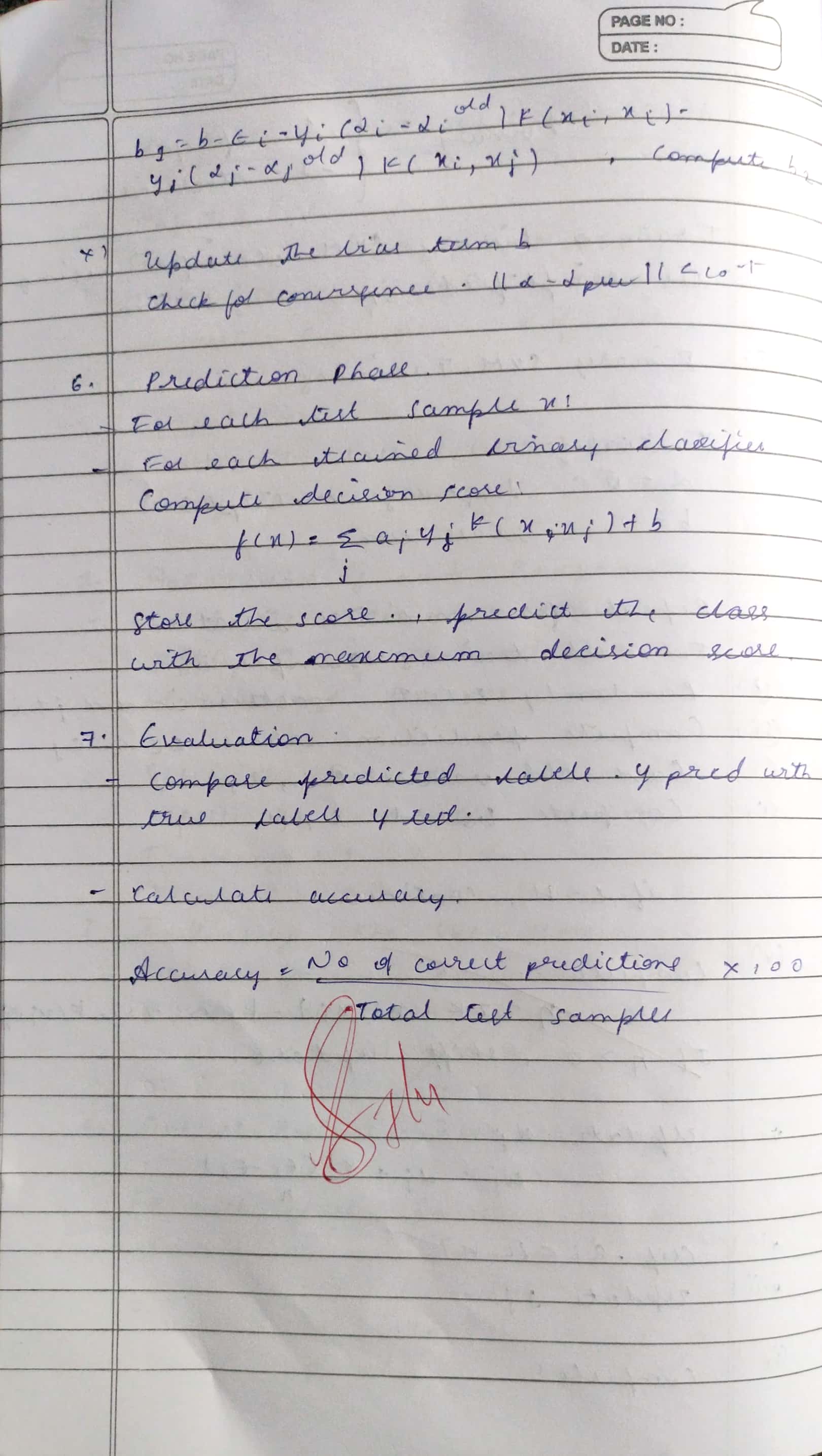
**Program 7**

Build Support vector machine model for a given dataset

**Screenshot:**

****





**Code:**

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split from sklearn.svm import SVC

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

# Load dataset

iris = datasets.load\_iris()

X = iris.data y = iris.target

# For binary classification (class 0 vs 1) X = X[y != 2]

y = y[y != 2]

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

# Train SVM

clf = SVC(kernel='linear') # Try 'rbf', 'poly', etc. clf.fit(X\_train, y\_train)

# Accuracy

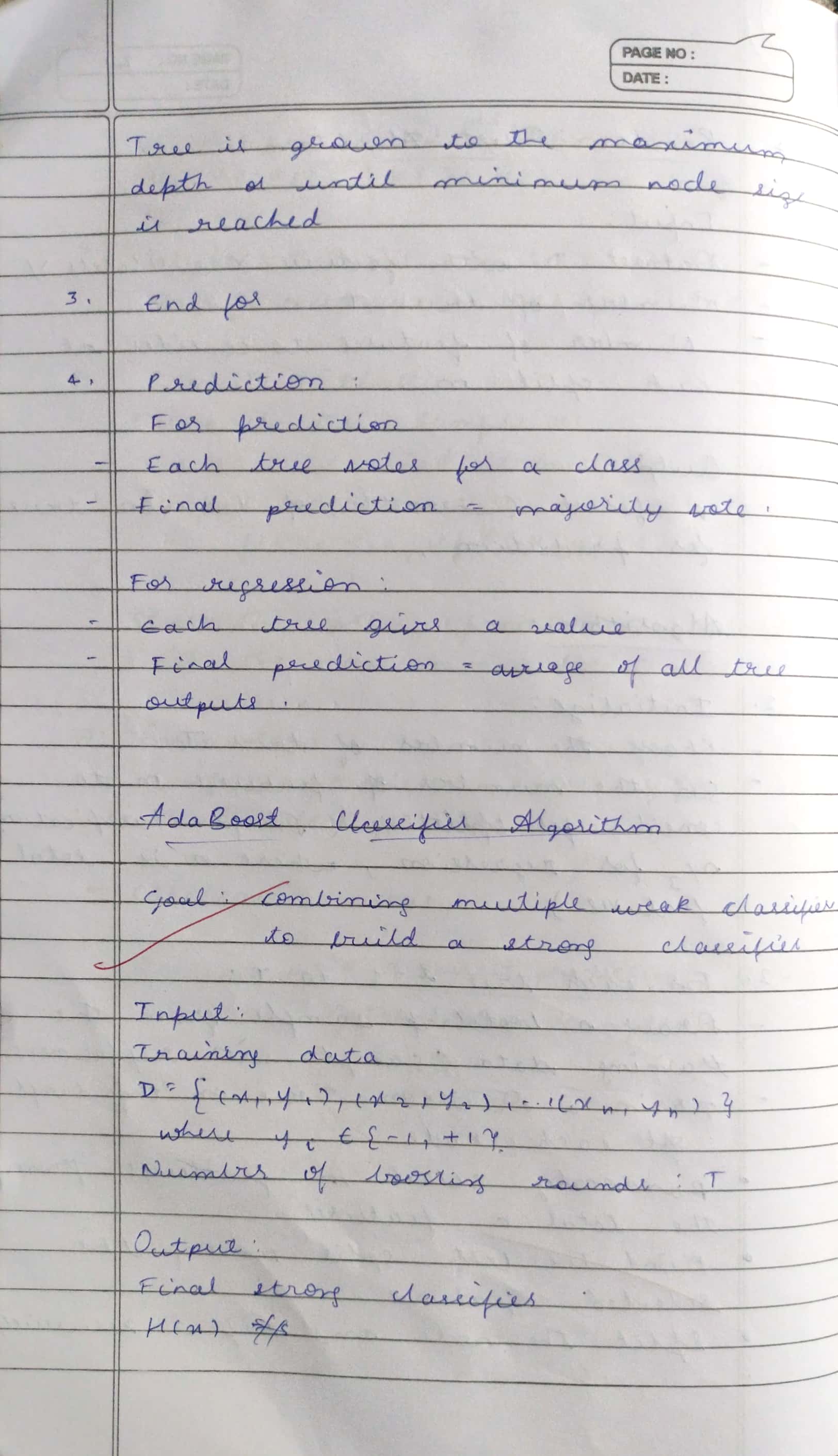
print("Test Accuracy:", clf.score(X\_test, y\_test))

**Program 8**

Implement Random forest ensemble method on a given dataset

**Screenshot:**

****



**Code:**

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

# Load sample dataset iris = load\_iris()

X, y = iris.data, iris.target

# Train/test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize Random Forest

rf = RandomForestClassifier(n\_estimators=100, random\_state=42) rf.fit(X\_train, y\_train)

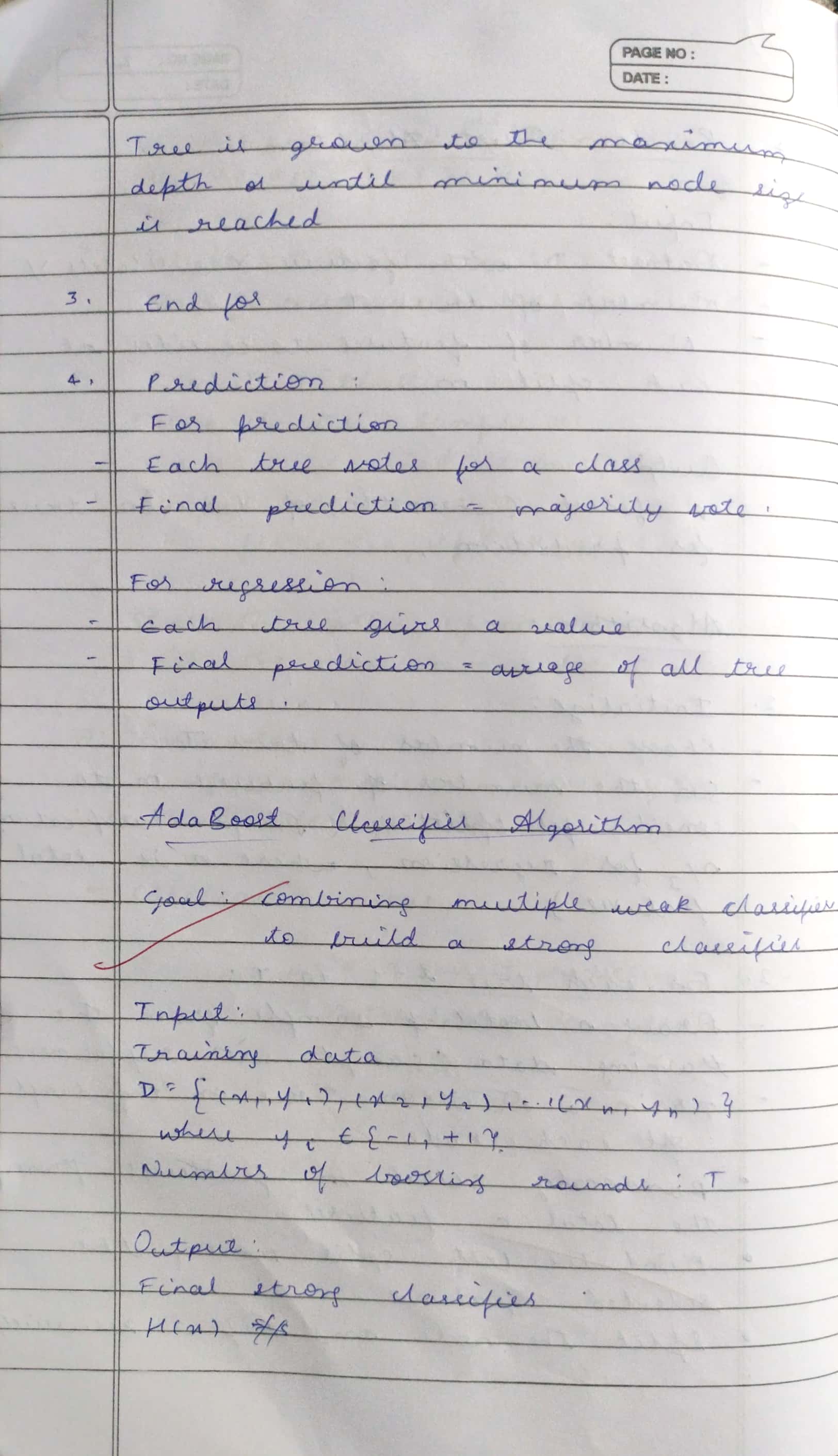
# Predict and evaluate y\_pred = rf.predict(X\_test)

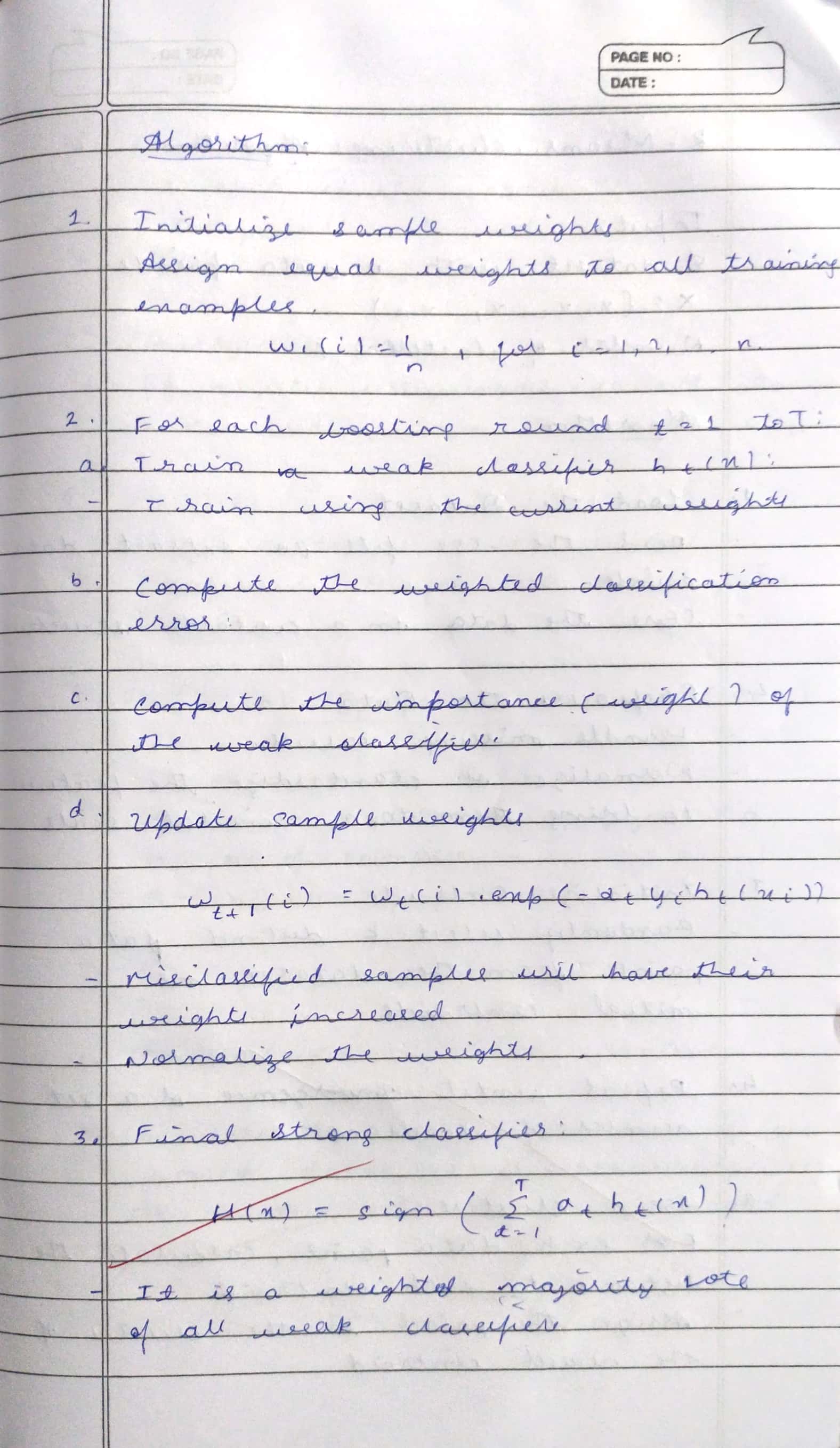
print("Accuracy:", accuracy\_score(y\_test, y\_pred))

**Program 9**

Implement Boosting ensemble method on a given dataset

**Screenshot:**

****



**Code:**

from sklearn.ensemble import AdaBoostClassifier from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split from sklearn.metrics import accuracy\_score

# Load Iris dataset iris = load\_iris()

X, y = iris.data, iris.target

# For AdaBoost, we'll use binary classification # Convert to binary (setosa vs. not-setosa)

y = (y == 0).astype(int)

# Split data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train AdaBoost

model = AdaBoostClassifier(n\_estimators=50, learning\_rate=1.0, random\_state=42) model.fit(X\_train, y\_train)

# Predict and evaluate

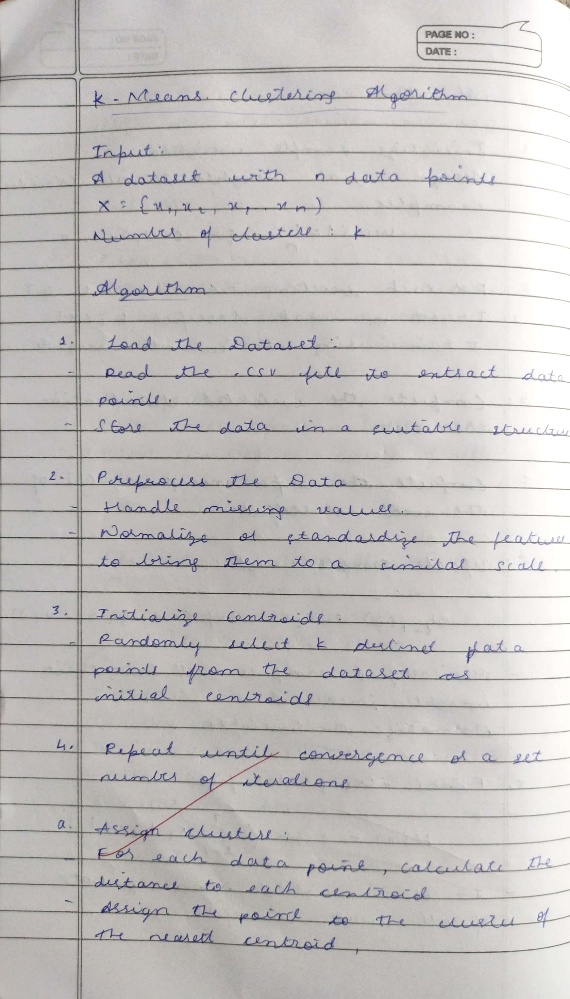
y\_pred = model.predict(X\_test)

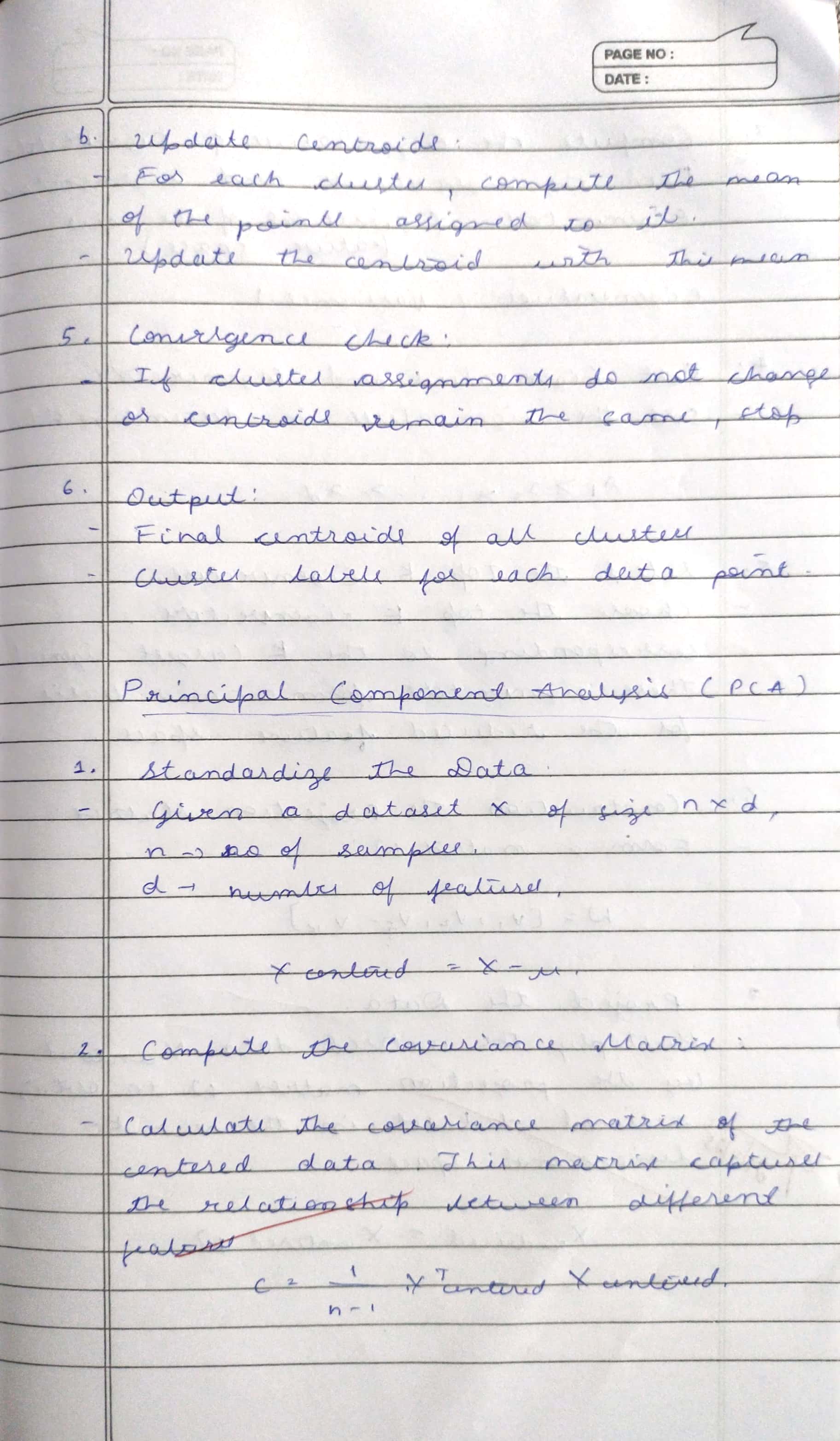
print("AdaBoost Accuracy (sklearn):", accuracy\_score(y\_test, y\_pred))

**Program 10**

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

**Screenshot:**

****



**Code:**

import pandas as pd

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris # Import load\_iris

# Step 1: Load the Iris dataset directly iris = load\_iris()

# Create a DataFrame from the data and target

data = pd.DataFrame(data=iris.data, columns=iris.feature\_names)

# Add the target column for potential reference, though not used for clustering data['target'] = iris.target

# Step 2: Extract only numeric columns (or select required features) # All features in the Iris dataset are numeric

X = data[iris.feature\_names].values # Use the feature names to select columns

# Step 3: Apply KMeans

# Adjust n\_clusters based on the expected number of clusters in your data (3 for Iris)

kmeans = KMeans(n\_clusters=3, random\_state=42, n\_init=10) # Added n\_init to suppress future warnings

data['Cluster'] = kmeans.fit\_predict(X)

# Step 4: Plot clusters (for 2D data)

# Iris data has 4 features. We will plot the first two features for visualization. if X.shape[1] >= 2:

plt.scatter(X[:, 0], X[:, 1], c=data['Cluster'], cmap='viridis')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], color='red', marker='x', s=200)

plt.title("K-Means Clustering of Iris Dataset") plt.xlabel(iris.feature\_names[0]) # Label with actual feature name plt.ylabel(iris.feature\_names[1]) # Label with actual feature name plt.show()

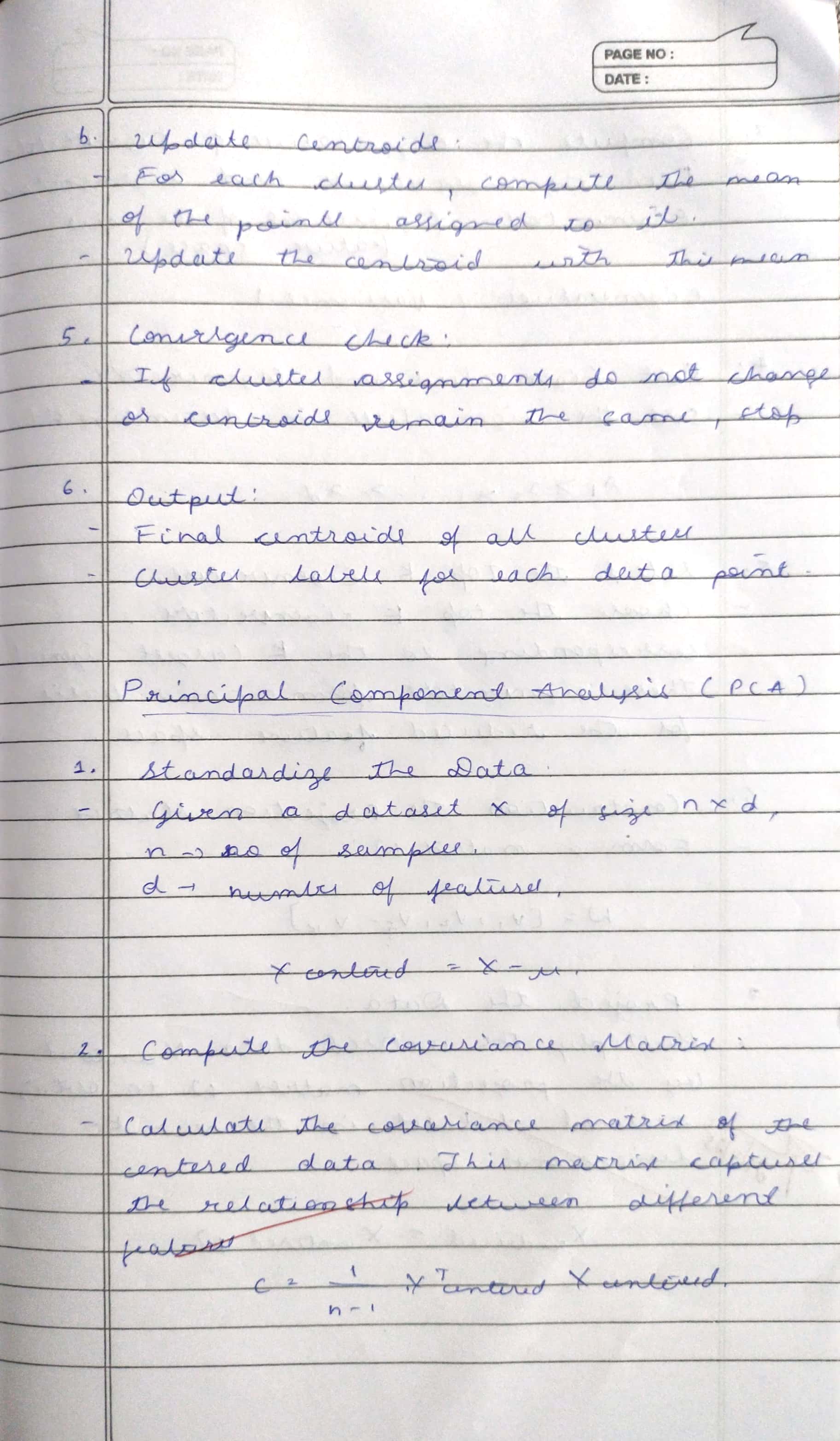
else:

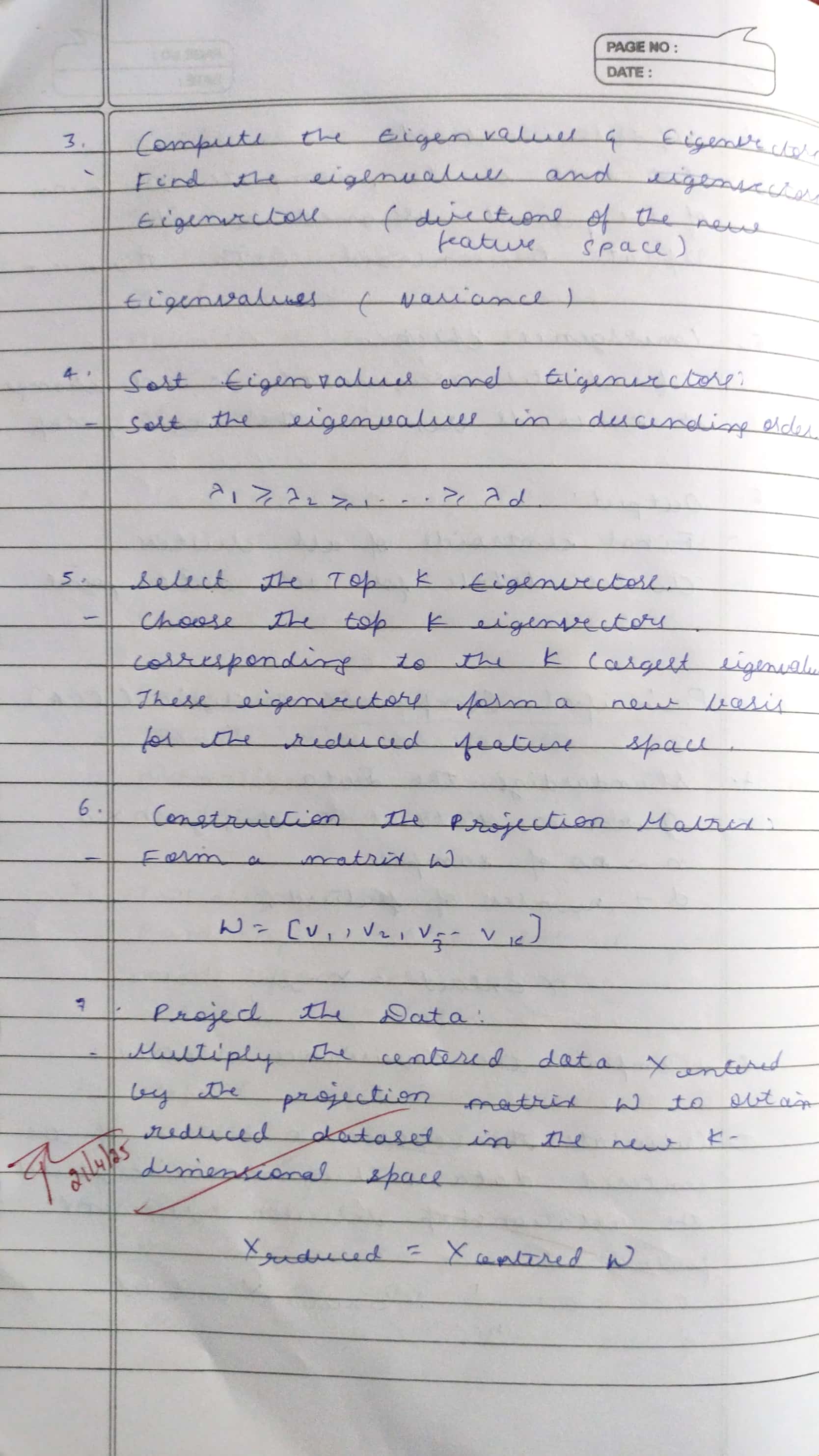
print("Cannot plot clustering results directly for data with less than 2 features.")

**Program 11**

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

**Screenshot:**

****



**Code:**

import pandas as pd

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

# Load dataset

data = pd.read\_csv("your\_data.csv") # Replace with your file X = data.select\_dtypes(include=['float64', 'int64'])

# Step 1: Standardize scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Step 2: Apply PCA

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

# Print explained variance ratio

print("Explained variance ratio:", pca.explained\_variance\_ratio\_)

# Visualize

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c='blue', alpha=0.5) plt.title("PCA - 2D Projection")

plt.xlabel("Principal Component 1")

plt.ylabel("Principal Component 2") plt.show()

