VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum- 590014, Karnataka.

LAB REPORT on

Machine Learning (23CS6PCMAL)

Submitted by

Pranav R Hegde (1BM22CS202)

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

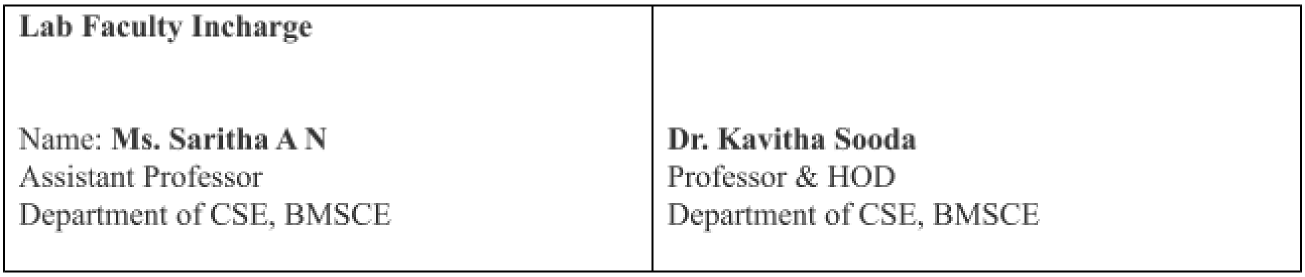
B.M.S. College of Engineering

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 **Pranav Hegde (1BM22CS202),** 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.



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

INDEX

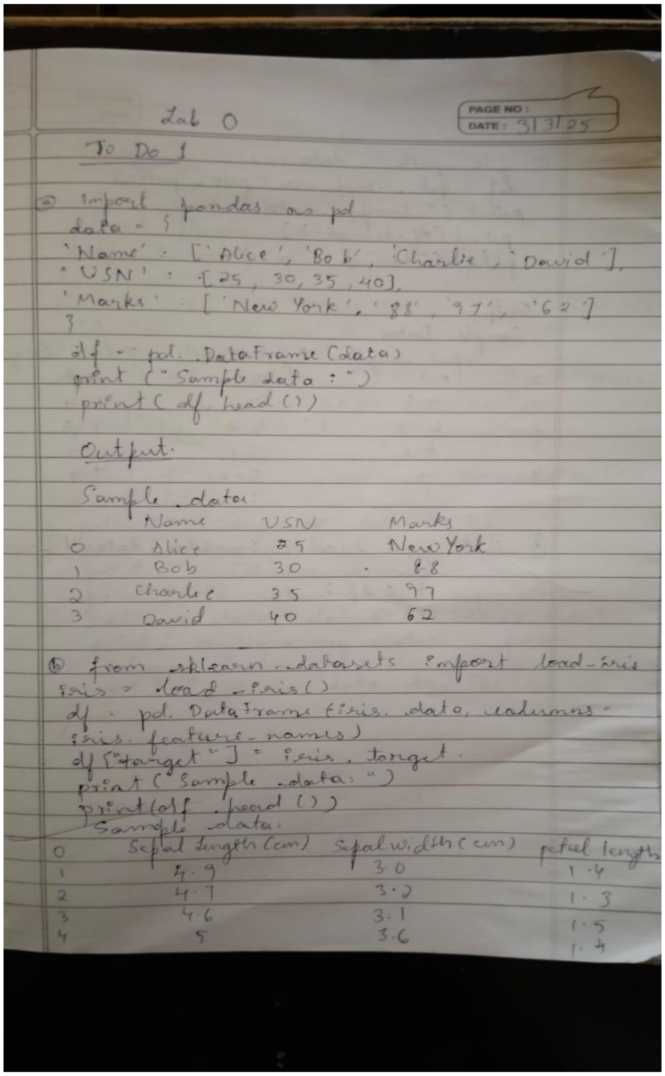
11	12-05-2025	Implement Dimensionality reduction using Principle Component Analysis (PCA) method.	44

Github Link: https://github.com/pranav0hegde/ML

Program 1

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

Screenshot:



Lab 0

PAGE NO :
DATE : 3/3/25

To Do 1

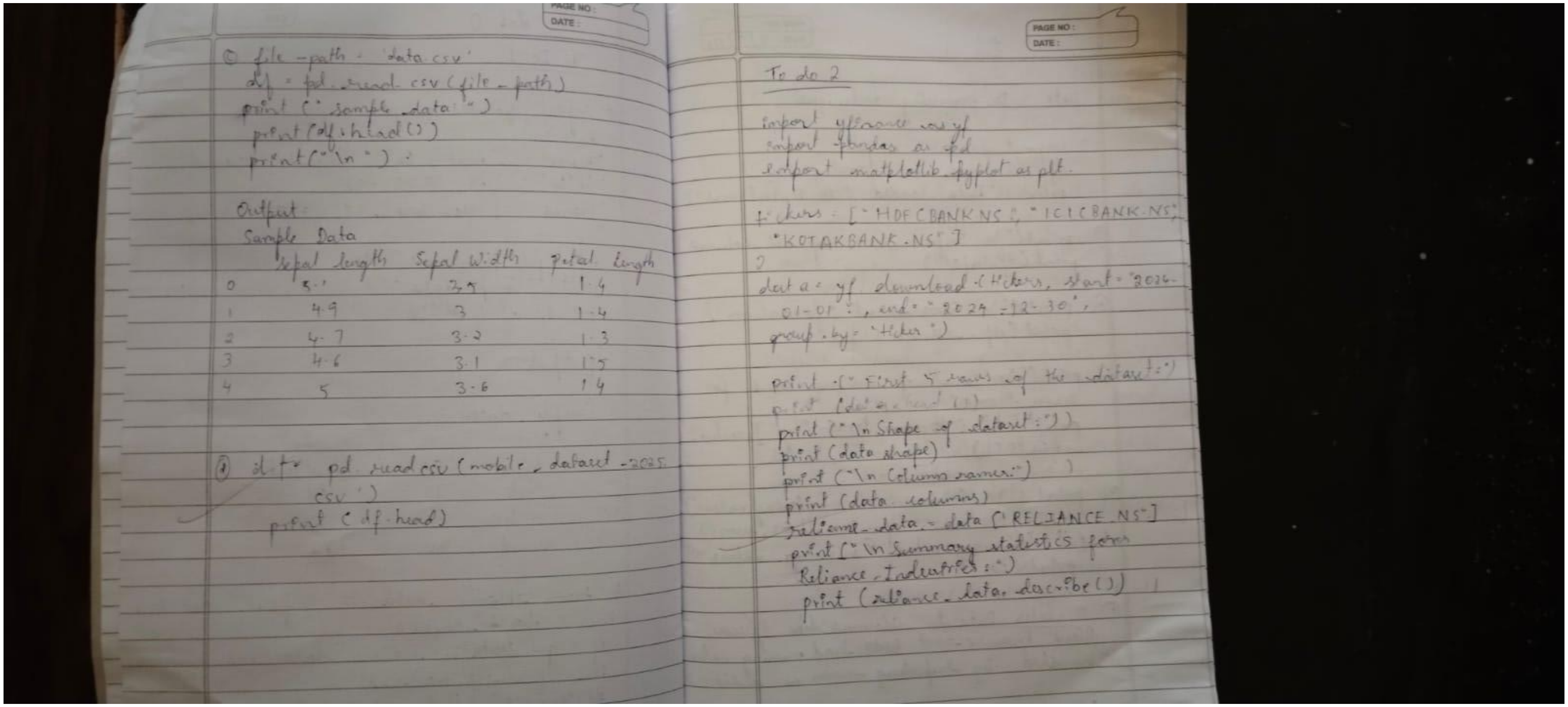
```
(a) import pandas as pd
data = {
'Name' : ['Alice', 'Bob', 'Charlie', 'David'],
'USN' : [25, 30, 35, 40],
'Marks' : ['New York', '88', '97', '62']
}
df = pd.DataFrame(data)
print("Sample data :")
print(df.head())
```

Output.

```
Sample data
      Name      USN      Marks
0     Alice     25       New York
1     Bob       30       88
2     Charlie   35       97
3     David     40       62
```

```
(b) from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns=
iris.feature_names)
df["target"] = iris.target
print("Sample data:")
print(df.head())
```

```
Sample data:
0     Sepal length (cm)   Sepal width (cm)   petal length
1          4.9                 3.0                1.4
2          4.7                 3.2                1.3
3          4.6                 3.1                1.5
4          5                   3.6                1.4
```



PAGE NO :
DATE :

```
© file -path = 'data.csv'
  df = pd.read csv (file - path)
  print (" sample data ")
   print (df.head())
   print("\n")
```

Output

Sample Data

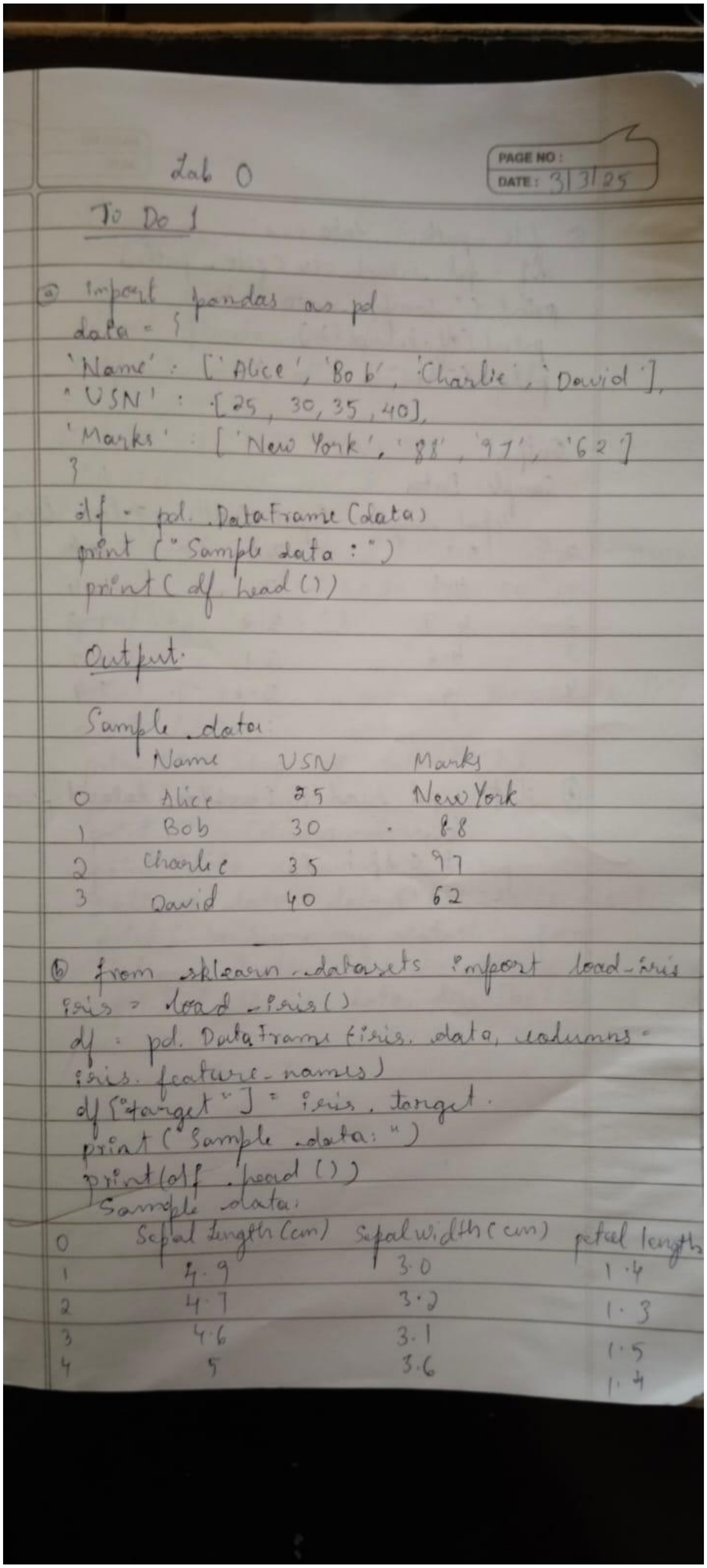
```
     sepal length    sepal width    petal length
0        5.1             3.5            1.4
1        4.9             3              1.4
2        4.7             3.2            1.3
3        4.6             3.1            1.5
4        5               3.6            1.4
```

```
ⓓ df = pd.read csv (mobile, dataset -2025.
          csv ')
    print (df.head)
```

PAGE NO :
DATE :

To do 2

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

tickers = [" HDFCBANK NS ", " ICICBANK.NS",
 "KOTAKBANK.NS" ]
?
data = yf.download.(tickers, start = "2024-
  01-01 ", end = " 2024 -12-30 ",
group.by = 'ticker ')

 print .(" First 5 rows of the dataset:")
 print (data.head ())
 print (" \n Shape of dataset:"))
 print (data shape)
 print ("\n Column names:")
 print (data columns)
 reliance_data = data [' RELIANCE NS"]
 print (" \n Summary statistics for
  Reliance Industries : ")
  print (reliance_data. describe())
```



Lab 0

PAGE NO :
DATE : 3|3|25

To Do 1

```
ⓐ import pandas as pd
data = {
'Name' : ['Alice', 'Bob', 'Charlie', 'David'],
'USN' : [25, 30, 35, 40],
'Marks' : ['New York', '88', '97', '62']
}
df = pd.DataFrame(data)
print ("Sample data :")
print(df.head())
```

Output.

```
Sample data
      Name     USN     Marks
0     Alice    25      New York
1     Bob      30      88
2     Charlie  35      97
3     David    40      62
```

```
ⓑ from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(iris.data, columns=
iris.feature_names)
df["target"] = iris.target.
print("Sample data:")
print(df.head())
```

```
Sample data:
0    Sepal length(cm)   Sepal width(cm)   petal length
1         4.9               3.0               1.4
2         4.7               3.2               1.3
3         4.6               3.1               1.5
4         5                 3.6               1.4
```

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
```

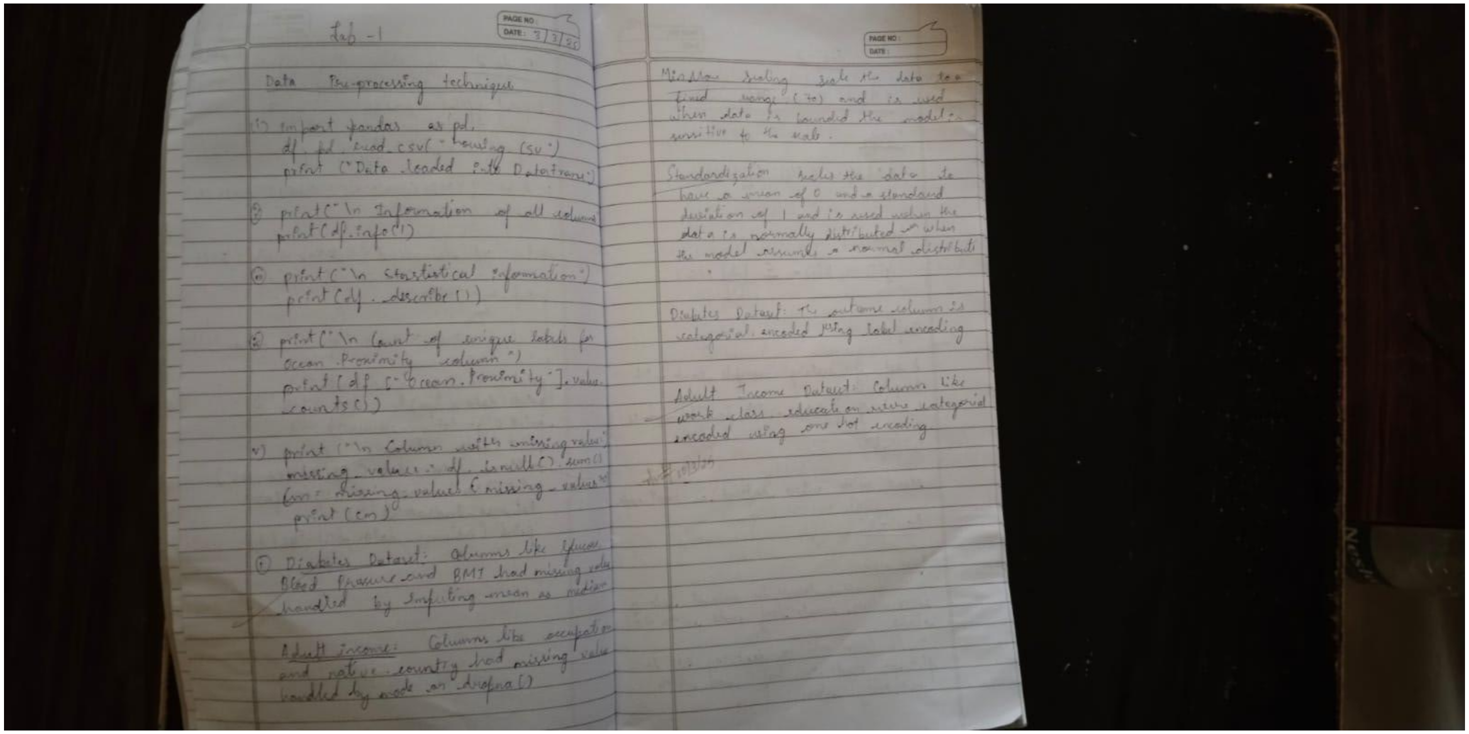
```
from sklearn.datasets import load_diabetes diabetes =

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:



Lab -1

PAGE NO:
DATE: 3/3/25

Data Pre-processing technique

```
1) import pandas as pd.
   df = pd.read_csv("housing.csv")
   print("Data loaded into Dataframe")

2) print("\n Information of all columns
   print(df.info())

iii) print("\n statistical information")
   print(df.describe())

4) print("\n Count of unique labels for
   ocean.Proximity column")
   print(df["ocean.Proximity"].value.
   counts())

v) print("\n Column with missing values")
   missing_values = df.isnull().sum()
   cm = missing_values[missing_values>0]
   print(cm)
```

1 Diabetes Dataset: Columns like glucose Blood Pressure and BMI had missing value handled by imputing mean or median

Adult income: Columns like occupation and native.country had missing value handled by mode or dropna()

PAGE NO:
DATE:

MinMax Scaling scale the data to a fixed range (0-1) and is used when data is bounded the model is sensitive to the scale.

Standardization scales the data to have a mean of 0 and a standard deviation of 1 and is used when the data is normally distributed or when the model assumes a normal distributi

Diabetes Dataset: The outcome column is categorical encoded using label encoding

Adult Income Dataset: Column like work class education were categorical encoded using one hot encoding

10/3/25

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']])
```

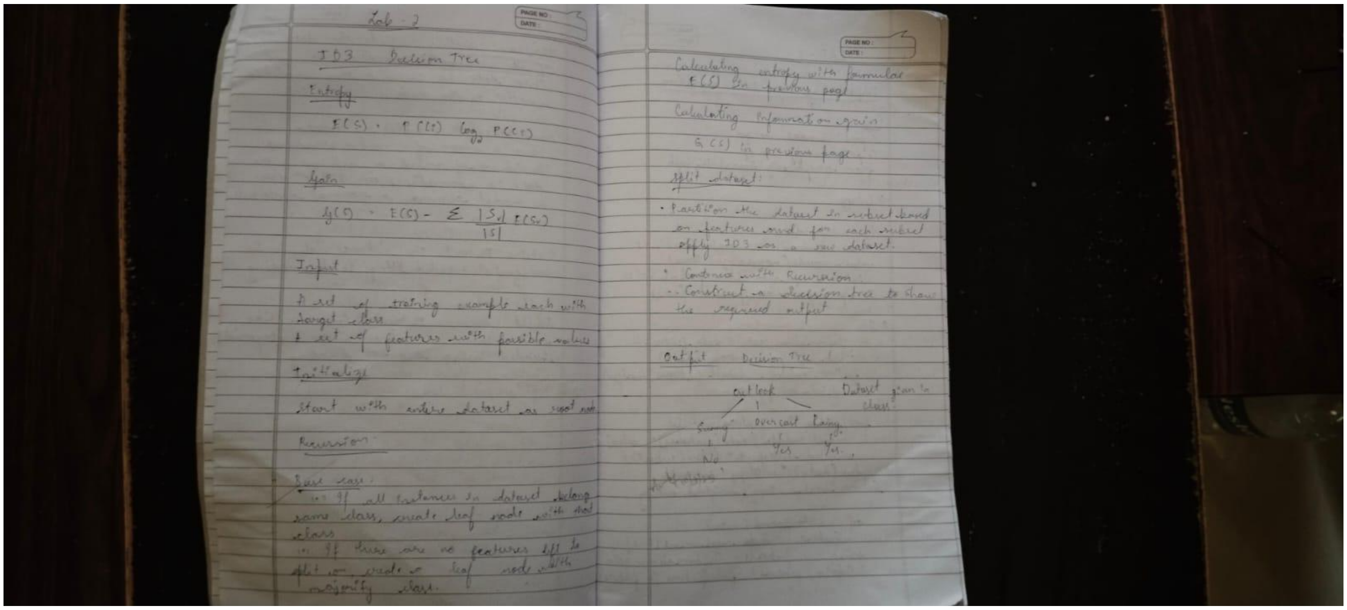
```
# Using IQR method
Q1 = df['Salary'].quantile(0.25)
Q3 = df['Salary'].quantile(0.75)
IQR = Q3 - Q1
```

```
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:



Lab - 2

ID3 Decision Tree

Entropy

E(S) = P(Ci) $\log_2$ P(Ci)

Gain

G(S) = E(S) - Σ $\frac{|S_v|}{|S|}$ E(Sv)

Input

A set of training example each with target class
A set of features with possible values

Initialize

Start with entire dataset as root node

Recursion

Base case:
i. If all instances in dataset belong same class, create leaf node with that class
ii. If there are no features left to split on, create a leaf node with majority class.

Calculating entropy with formular E(S) in previous page

Calculating Information gain

G(S) in previous page

Split dataset:

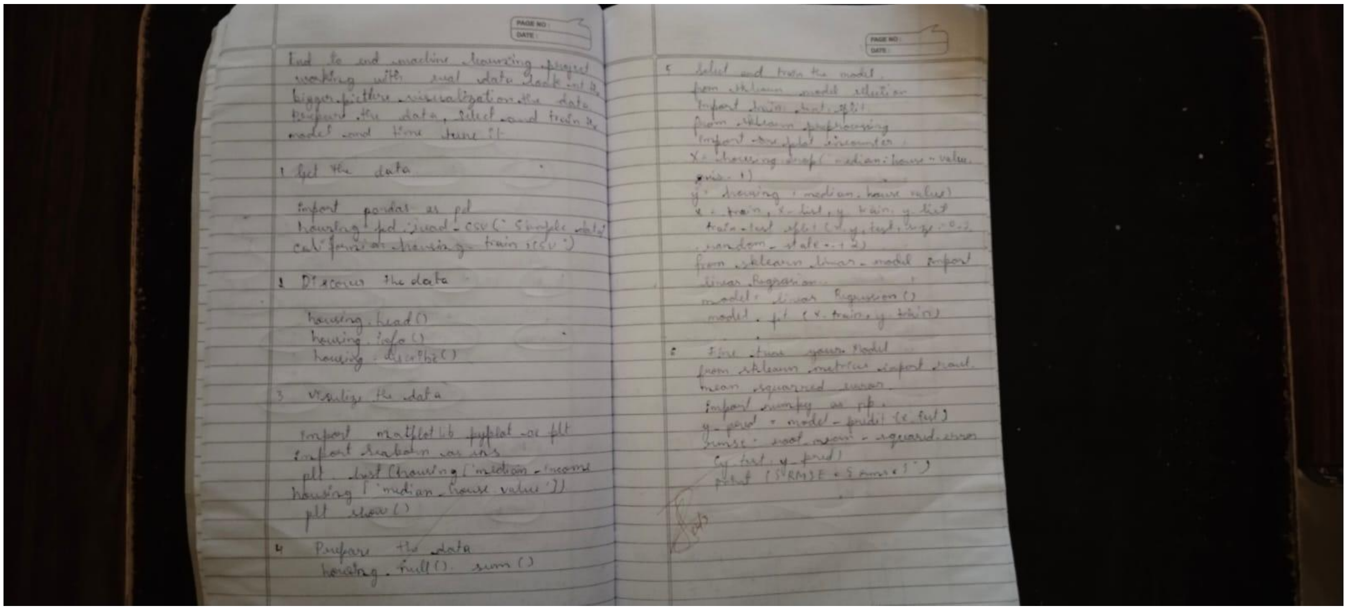
- Partition the dataset in subset based on features and for each subset apply ID3 on a new dataset.

- Continue with Recursion
- Construct a decision tree to show the required output

Output Decision Tree

Outlook
Sunny Overcast Rainy
No Yes Yes

Dataset given in class



End to end machine learning project working with real data look at the bigger picture visualization the data prepare the data, select and train the model and fine tune it

1 Get the data

```
import pandas as pd
housing = pd.read_csv('Sample_data/california_housing_train.csv')
```

2 Discover the data

```
housing.head()
housing.info()
housing.describe()
```

3 Visualize the data

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.hist(housing['median_income'],
housing['median_house_value'])
plt.show()
```

4 Prepare the data

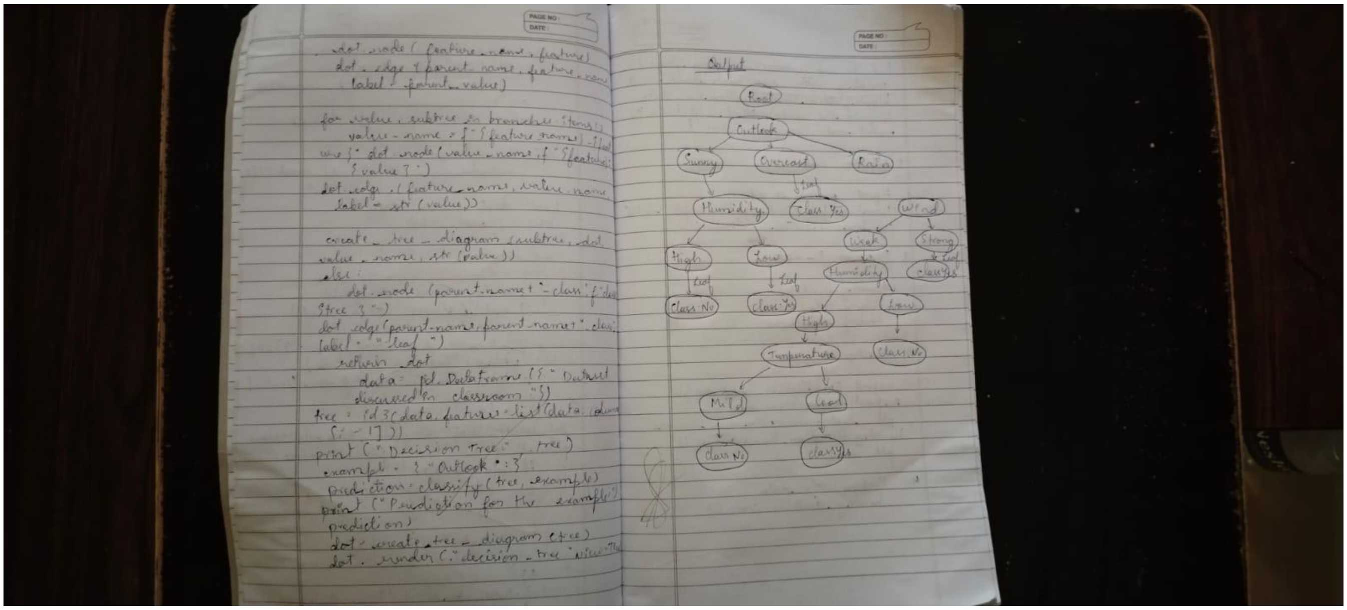
```
housing.isnull().sum()
```

5 Select and train the model

```
from sklearn.model_selection
import train_test_split
from sklearn.preprocessing
import OneHotEncoder
X = housing.drop('median_house_value',
axis=1)
y = housing['median_house_value']
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)
from sklearn.linear_model import
LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
```

6 Fine tune your Model

```
from sklearn.metrics import root_
mean_squared_error
import numpy as np
y_pred = model.predict(X_test)
rmse = root_mean_squared_error
(y_test, y_pred)
print(f"RMSE = {rmse}")
```



PAGE NO:
DATE:

```
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}_{feat
ure}" dot.node(value_name, f"{feature}
    {value}")
dot.edge.(feature_name, value_name
  label = str(value))

create_tree_diagram (subtree, dot
value_name, str (value))
else:
    dot.node (parent_name + "_class", f"cla
{tree }")
dot.edge(parent_name, parent_name + "_class
label = "leaf ")
  return dot
      data = pd.Dataframe ({ " Dataset
      discussed in classroom "})
tree = id3(data, features = list(data. colum
 [: -1]))
print (" Decision Tree:" , tree)
 example = { "Outlook ": }
 prediction = classify ( tree, example)
print ("Prediction for the example",
 prediction)
 dot = create_tree_diagram (tree)
 dot . render (" decision_tree " view=Tr
```

PAGE NO:
DATE:

Output

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):

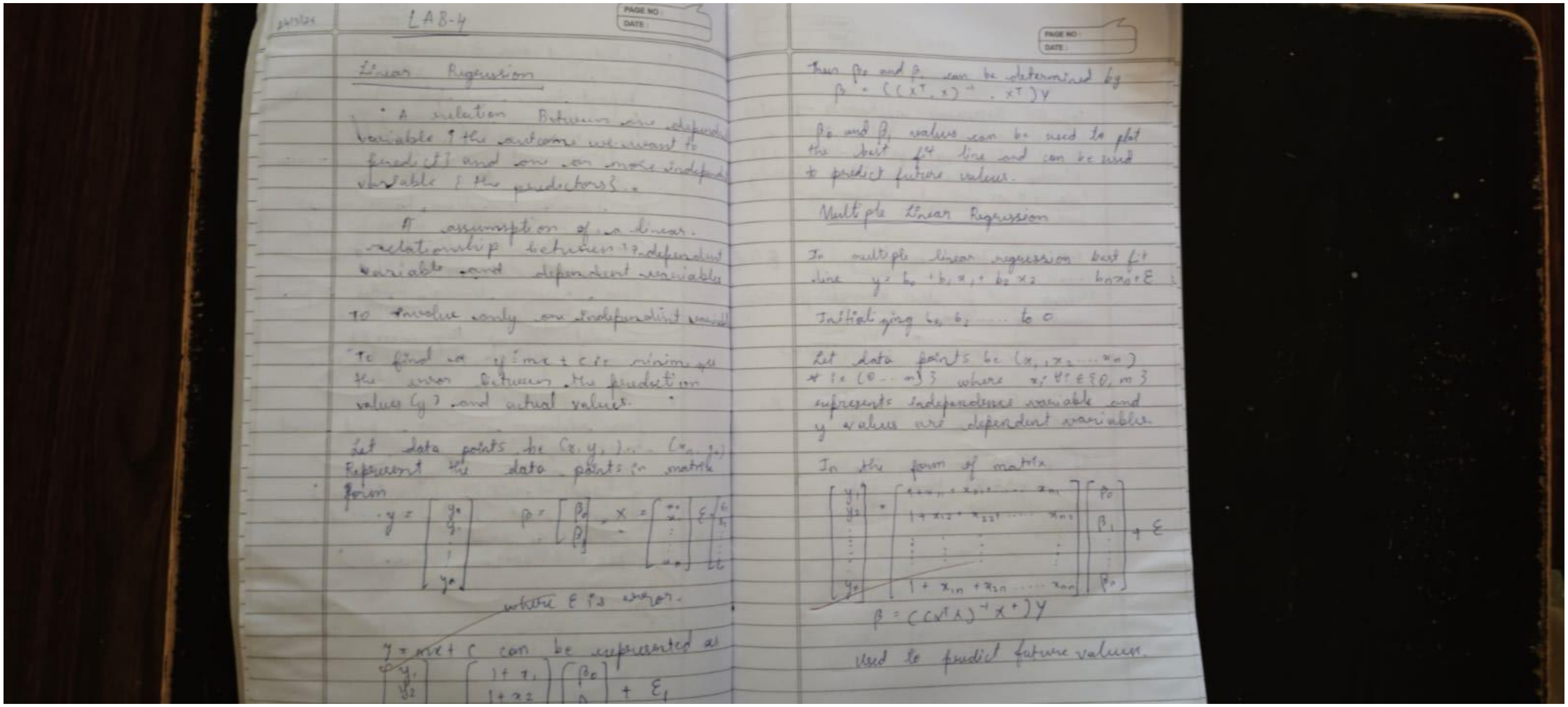
total_entropy = entropy(data)

```
    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:



24/3/24

LAB-4

PAGE NO :
DATE :

Linear Regression

- A relation Between one dependent variable {the outcome we want to predict} and one or more independent variable {the predictors}.

A assumption of a linear relationship between independent variable and dependent variables

To involve only one independent variable

To find a $y = mx + c$ is minimizes the error between the prediction values (y) and actual values.

Let data points be $(x_1, y_1) \ldots (x_n, y_n)$
Represent the data points in matrix form

$$y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}, \quad X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \varepsilon = \begin{bmatrix} \varepsilon_0 \\ \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

where ε is error.

$y = mx + c$ can be represented as

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 + x_1 \\ 1 + x_2 \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \varepsilon_1$$

PAGE NO :
DATE :

Then β_0 and β_1 can be determined by
$\beta = ((X^T . X)^{-1} . X^T)Y$

β_0 and β_1 values can be used to plot the best fit line and can be used to predict future values.

Multiple Linear Regression

In multiple linear regression best fit line $y = b_0 + b_1 x_1 + b_2 x_2 \ \ldots \ b_n x_n + \varepsilon$

Initializing $b_0, b_1 \ldots$ to 0

Let data points be $(x_1, x_2 \ldots x_n)$ $\forall i \in \{0 \ldots n\}$ where $x_i \ \forall i \in \{0, m\}$ represents independence variable and y values are dependent variable.

In the form of matrix

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 + x_{11} + x_{21} + \ldots & x_{m1} \\ 1 + x_{12} + x_{22} + \ldots & x_{m2} \\ \vdots & \vdots \\ 1 + x_{1n} + x_{2n} \ldots & x_{mn} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} + \varepsilon$$

$\beta = ((X^T X)^{-1} X^T) Y$

Used to predict future values.

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

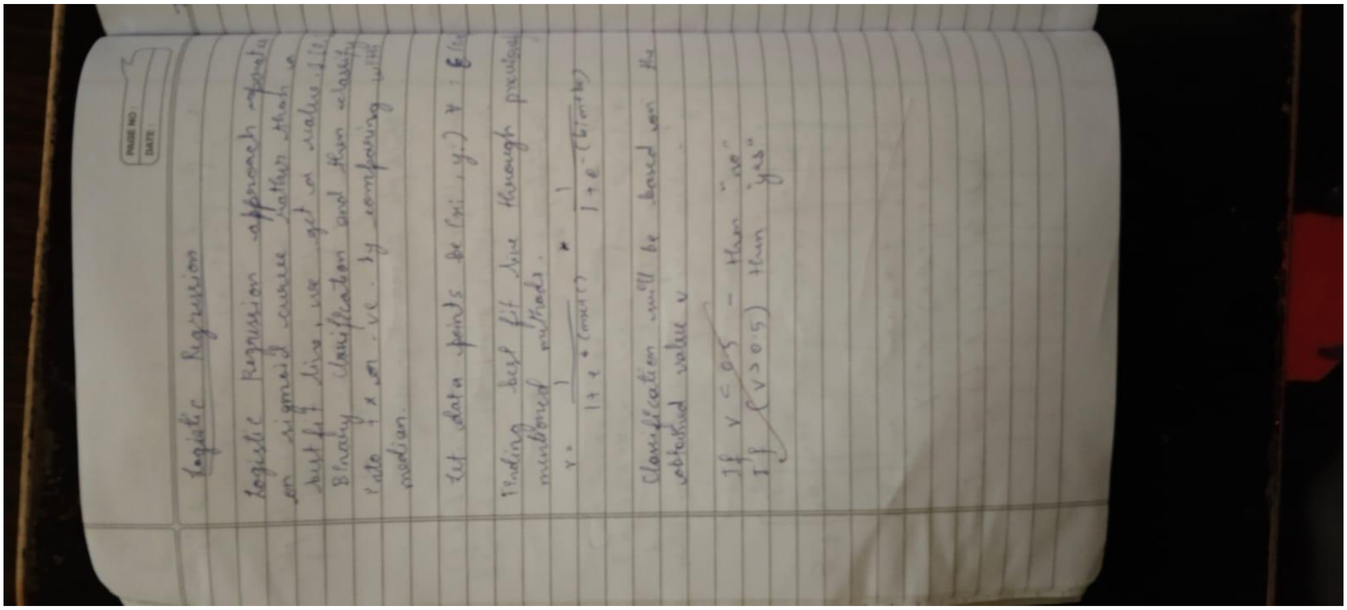
```
import pandas as pd #

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:



PAGE NO :
DATE :

Logistic Regression

Logistic Regression approach operates on sigmoid curve rather than a best fit line, we get a value i.e. Binary classification and then classify into +ve or -ve by comparing with median.

Let data points be $(x_i, y_i) \forall i \in$ (1

Finding best fit line through previously mentioned methods.

$$v = \frac{1}{1+e^{+(mx+c)}} * \frac{1}{1+e^{-(b_1 m+b_0)}}$$

Classification will be based on the obtained value v

If v < 0.5 - then "no"
If (v > 0.5) then "yes"

Code:

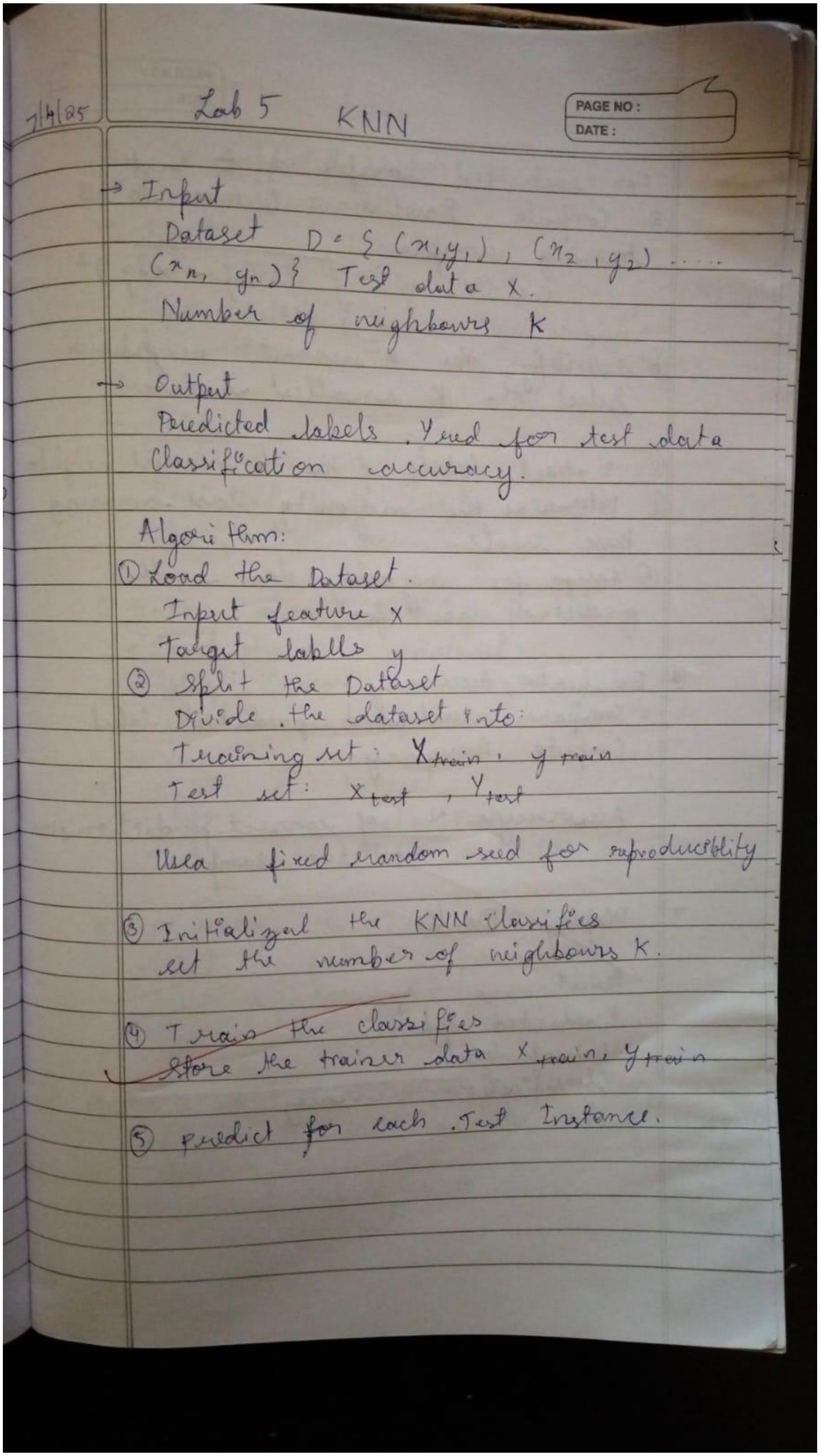
```
from sklearn.linear_model import LogisticRegression

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:



7/4/25 Lab 5 KNN

PAGE NO :
DATE :

→ Input

Dataset $D = \{(x_1, y_1), (x_2, y_2) \ldots\ldots (x_n, y_n)\}$ Test data X.

Number of neighbours K

→ Output

Predicted labels Y_{pred} for test data

Classification accuracy.

Algorithm:

① Load the Dataset.

Input feature X

Target lablls y

② Split the Dataset

Divide the dataset into:

Training set: X_{train}, y_{train}

Test set: X_{test}, Y_{test}

Usea fixed random seed for reproduciblity

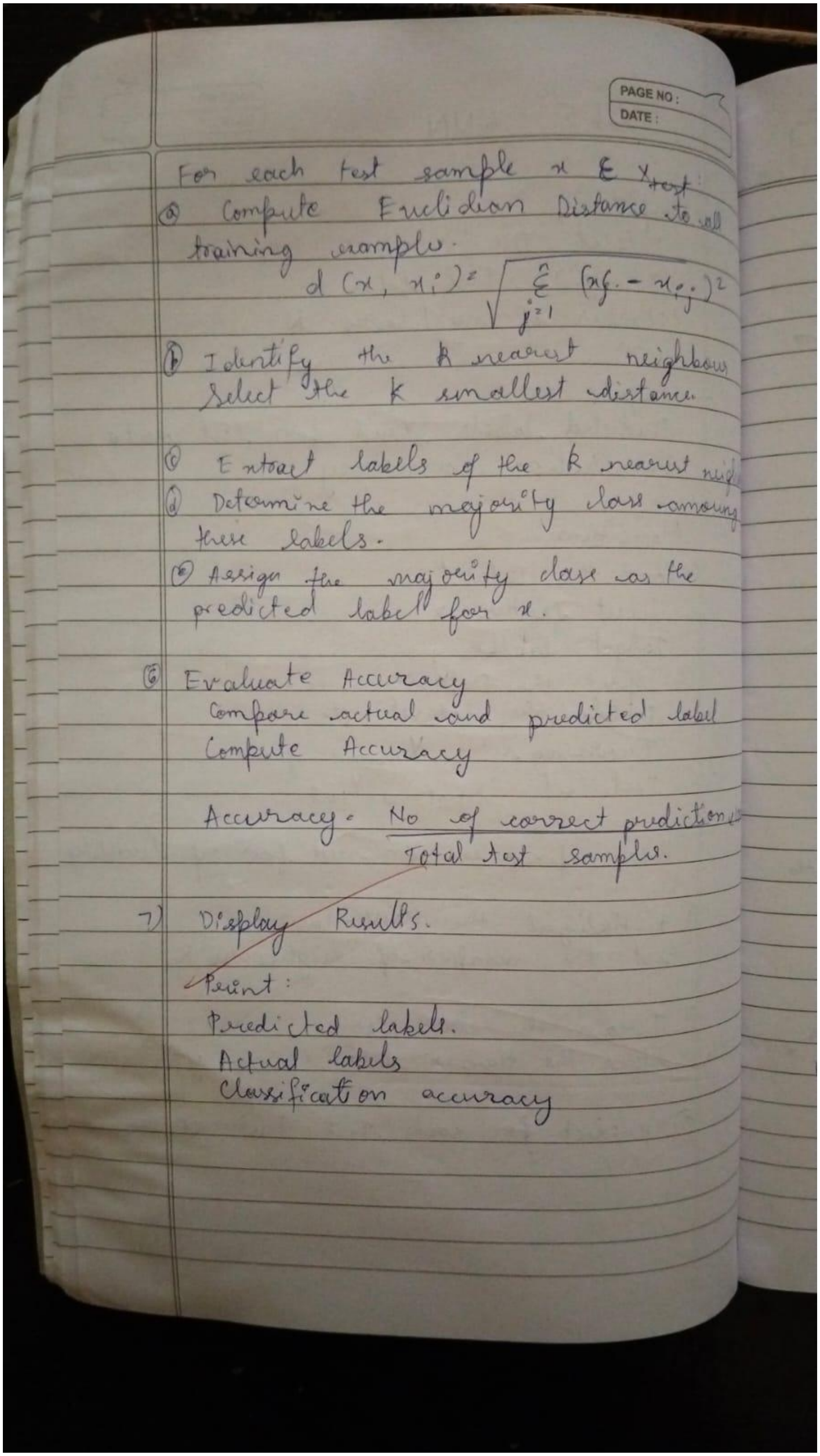
③ Initialized the KNN classifies

set the number of neighbours K.

④ Train the classifies

Store the trainer data X_{train}, y_{train}

⑤ Predict for each Test Instance.



PAGE NO :
DATE :

For each test sample $x \in X_{test}$:

(a) Compute Euclidian Distance to all training example.

$$d(x, x_i) = \sqrt{\sum_{j=1}^{n} (x_j - x_{ij})^2}$$

(b) Identify the k nearest neighbours
Select the k smallest distance.

(c) Extract labels of the k nearest neig

(d) Determine the majority class among these labels.

(e) Assign the majority class as the predicted label for x.

(6) Evaluate Accuracy
Compare actual and predicted label
Compute Accuracy

$$\text{Accuracy} = \frac{\text{No of correct predictions}}{\text{Total test samples.}}$$

7) Display Results.

Print :
Predicted labels.
Actual labels
Classification accuracy

Code:

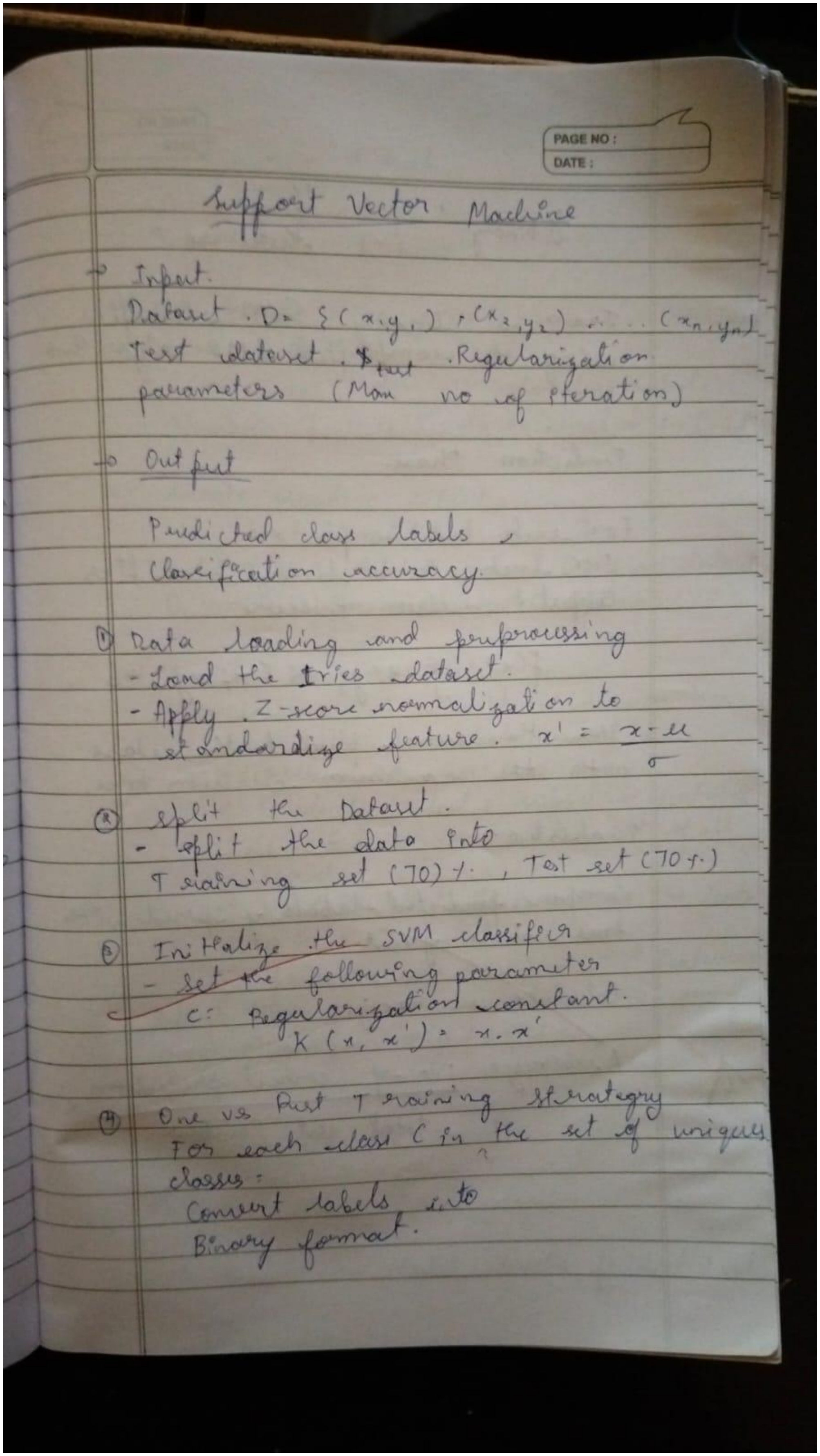
KNN

```
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:



PAGE NO :

DATE :

Support Vector Machine

→ Input.

Dataset . $D = \{(x_1, y_1), (x_2, y_2) \ldots (x_n, y_n)\}$

Test dataset . $\$_{test}$. Regularization parameters (Max no of iteration)

→ Output

Predicted class labels ,

Classification accuracy.

(1) Data loading and preprocessing

- Load the Iries dataset.
- Apply . Z-score normalization to standardize feature. $x' = \frac{x - \mu}{\sigma}$

(2) split the Dataset .

- split the data into

T raining set (70) %. , Test set (70 %.)

(3) Initialize the SVM classifier

- set the following parameter

C: Regularization constant.

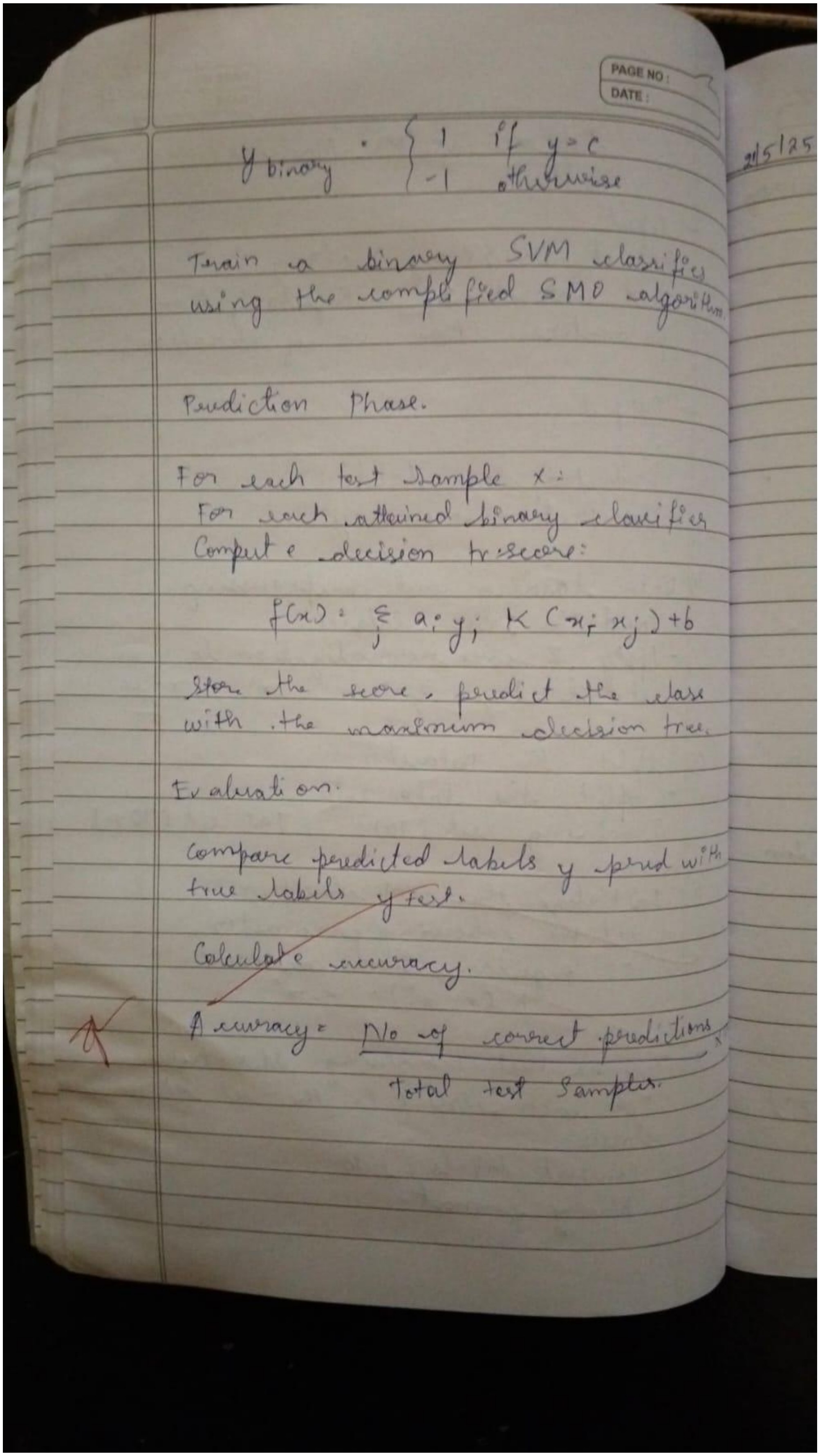
$K(x, x') = x \cdot x'$

(4) One vs Rest T raining Strategy

For each class C in the set of uniques classes :

Convert labels into

Binary format.



2/5/25

$$y_{binary} = \begin{cases} 1 & \text{if } y = c \\ -1 & \text{otherwise} \end{cases}$$

Train a binary SVM classifier using the simplified SMO algorithm.

Prediction Phase.

```
For each test sample x:
 For each attained binary classifier
 Compute decision tr:score:
     $f(x) = \sum_j a_i y_i K(x_i, x_j) + b$
 Store the score, predict the class
 with the maximum decision tree.
```

Evaluation.

Compare predicted labels y pred with true labels y test.

Calculate accuracy.

$$\text{Accuracy} = \frac{\text{No of correct predictions}}{\text{Total test samples}} \times$$

Code:

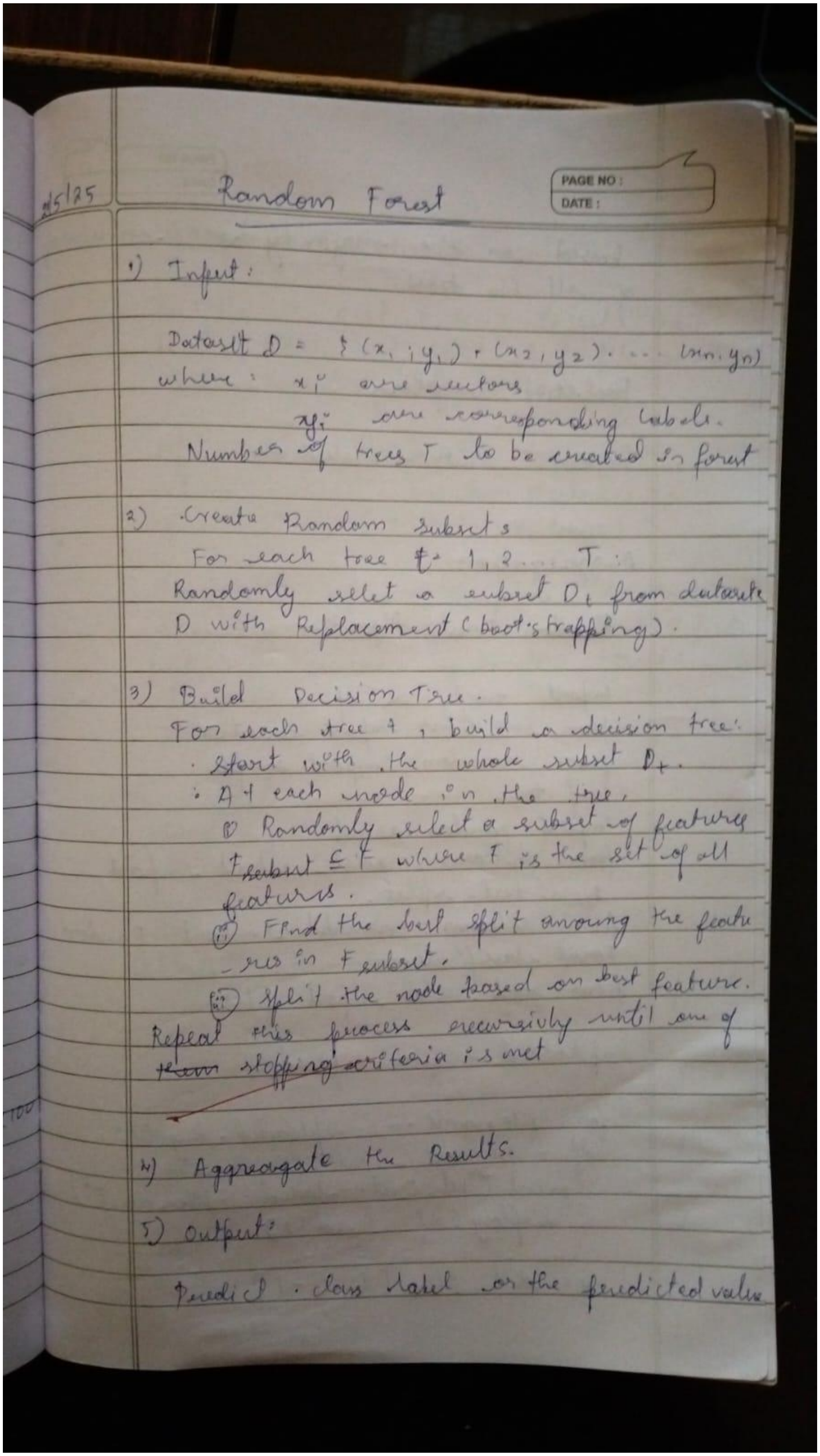
```
from sklearn import datasets
from sklearn.model_selection import
train_test_split from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# 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:



5/25

Random Forest

PAGE NO :
DATE :

1) Input:

Dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$
where: x_i are vectors
y_i are corresponding labels.
Number of trees T to be created in forest

2) Create Random subsets
For each tree $t = 1, 2 \ldots T$:
Randomly select a subset D_t from dataset D with Replacement (boot-strapping).

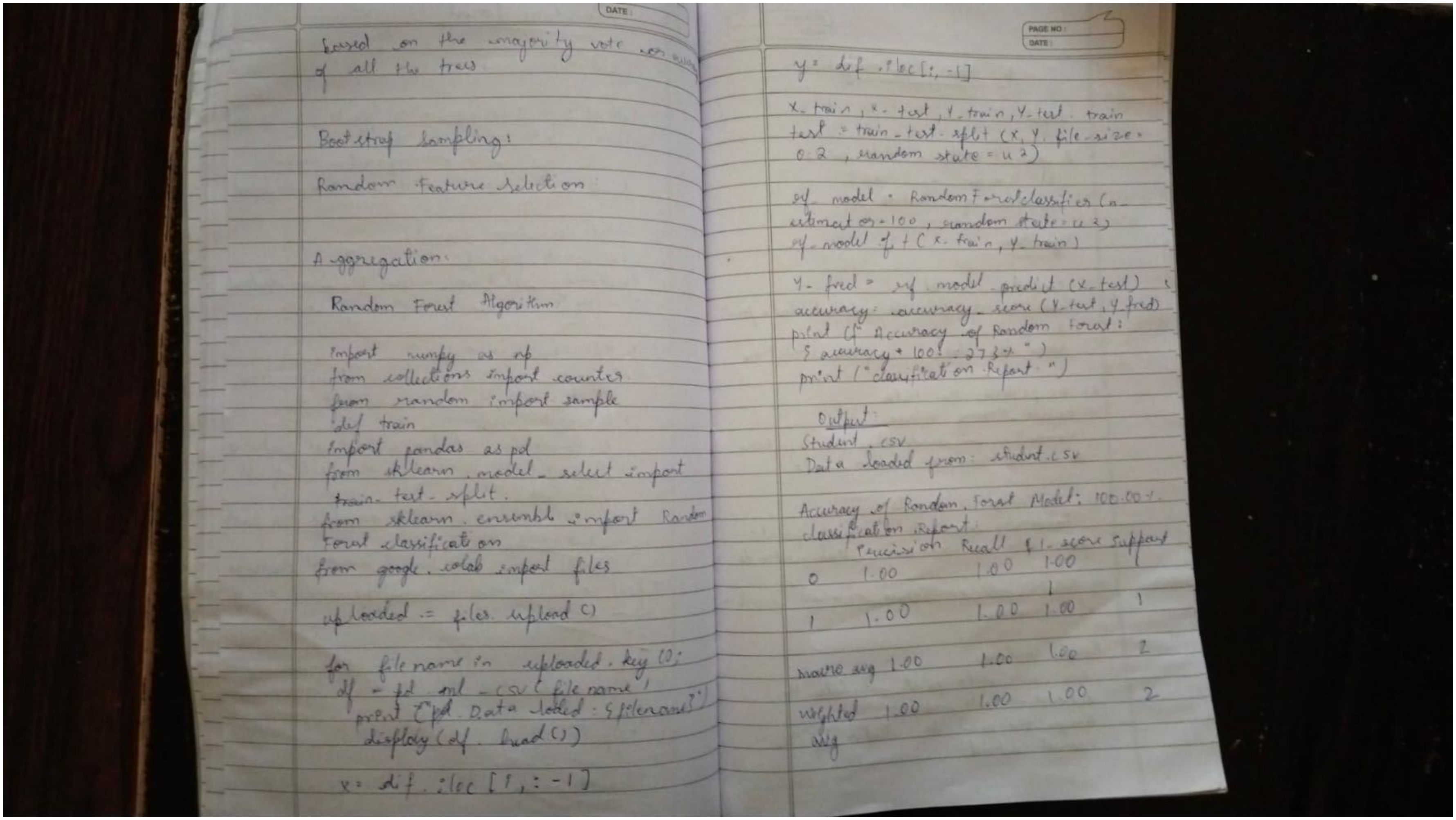
3) Build Decision Tree.
For each tree t, build a decision tree:
- Start with the whole subset D_t.
- At each node in the tree,
  (i) Randomly select a subset of features $F_{subset} \subseteq F$ where F is the set of all features.
  (ii) Find the best split among the features in F_{subset}.
  (iii) Split the node based on best feature.

Repeat this process recursively until one of ~~the~~ stopping criteria is met

4) Aggreagate the Results.

5) Output:

Predict class label or the predicted value



DATE :

based on the majority vote on avera
of all the trees

Bootstrap Sampling:

Random Feature Selection

Aggregation:

Random Forest Algorithm

```
import numpy as np
from collections import counter
from random import sample
def train
import pandas as pd
from sklearn.model_select import
 train_test_split.
from sklearn.ensembl import Random
Forest classification
from google.colab import files

uploaded := files.upload()

for filename in uploaded.key ():
 df = pd.read_csv(filename')
  print (f"pd.Data loded : {filename}")
  display(df.head())

x= dif.iloc [1, : -1]
```

PAGE NO :
DATE :

```
y= dif .iloc[:, -1]

x_train, x_test, y_train, y_test. train
test = train_test_split (x, y. file_size=
0.2 , random state = 42)

rf_model = RandomForestclassifier (n_
estimators=100, random state = 42)
rf_model fit (x_train, y_train)

y_pred = rf.model.predict (x_test)
accuracy= accuracy_score (y_test, y_pred)
print (f" Accuracy of Random forest:
 {accuracy * 100 : 2f}% ")
print ("classification Report ")
```

Output:

```
Student.csv
Data loaded from: student.csv

Accuracy of Random Forest Model: 100.00%
classification Report
            Precision  Recall  f1-score  support
0           1.00       1.00    1.00      1

1           1.00       1.00    1.00      1

macro avg   1.00       1.00    1.00      2

weighted    1.00       1.00    1.00      2
avg
```

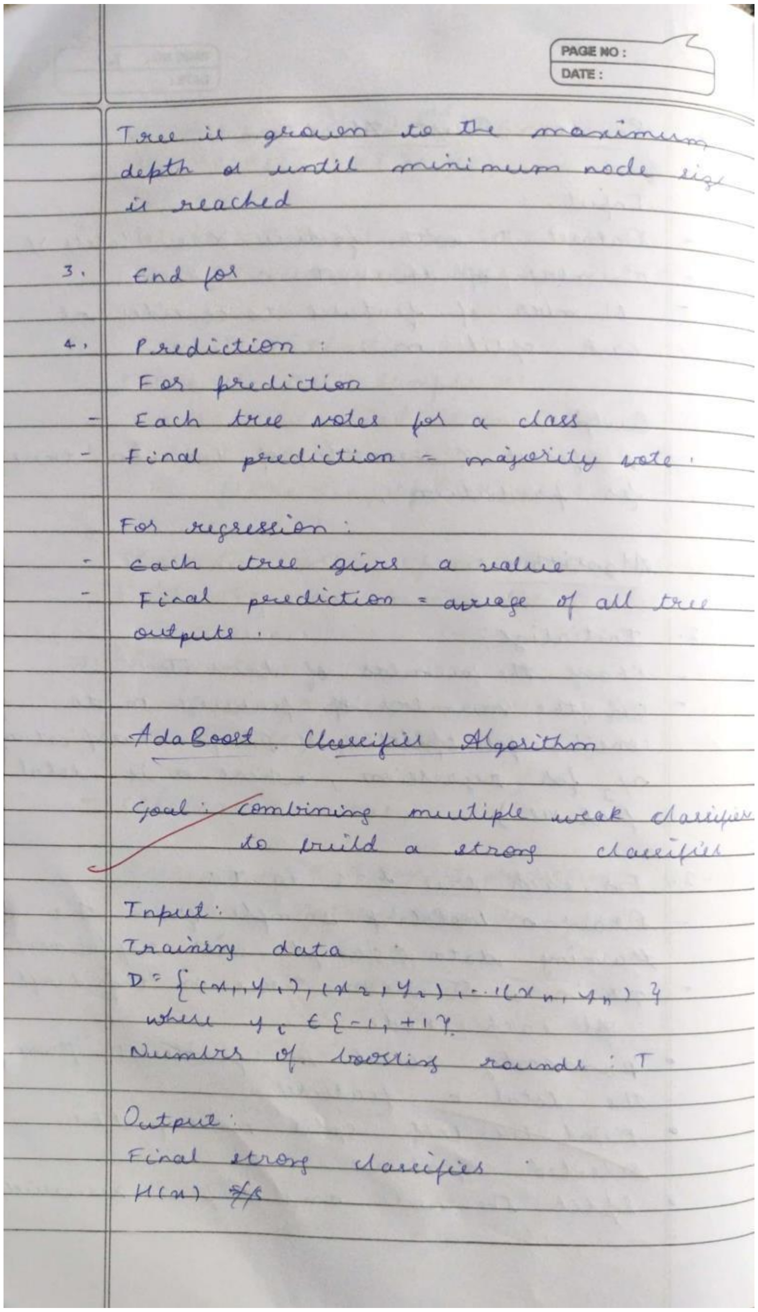
Code:

```
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:



PAGE NO :
DATE :

Tree is grown to the maximum depth or until minimum node size is reached

3. End for

4. Prediction :

For prediction

- Each tree votes for a class
- Final prediction = majority vote.

For regression :

- Each tree gives a value
- Final prediction = average of all tree outputs.

AdaBoost Classifier Algorithm

Goal : combining multiple weak classifier to build a strong classifier

Input :

Training data :

$D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$

where $y_i \in \{-1, +1\}$.

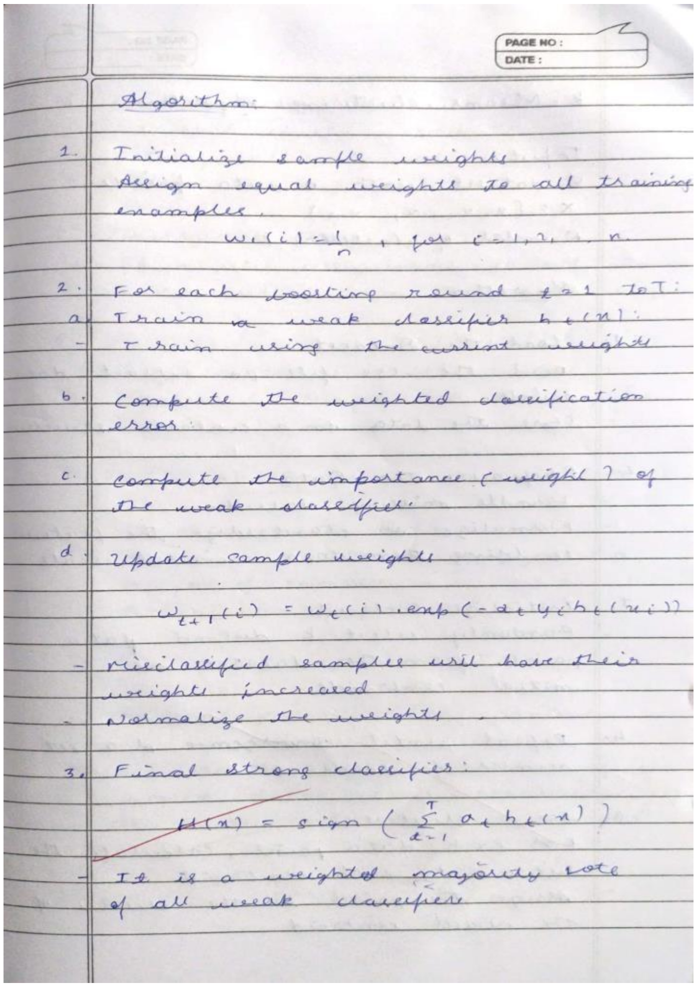
Numbers of boosting rounds : T

Output :

Final strong classifier :

$H(x)$ ~~s/s~~

Code:



PAGE NO :
DATE :

Algorithm:

1. Initialize sample weights
Assign equal weights to all training examples.

$w_1(i) = \frac{1}{n}$, for $i = 1, 2, .., n$.

2. For each boosting round $t = 1$ to T:

a. Train a weak classifier $h_t(x)$:
- Train using the current weights

b. Compute the weighted classification error

c. Compute the importance (weight) of the weak classifier.

d. Update sample weights

$$w_{t+1}(i) = w_t(i) . \exp(-\alpha_t y_i h_t(x_i))$$

- misclassified samples will have their weights increased
- Normalize the weights.

3. Final strong classifier:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

- It is a weighted majority vote of all weak classifiers

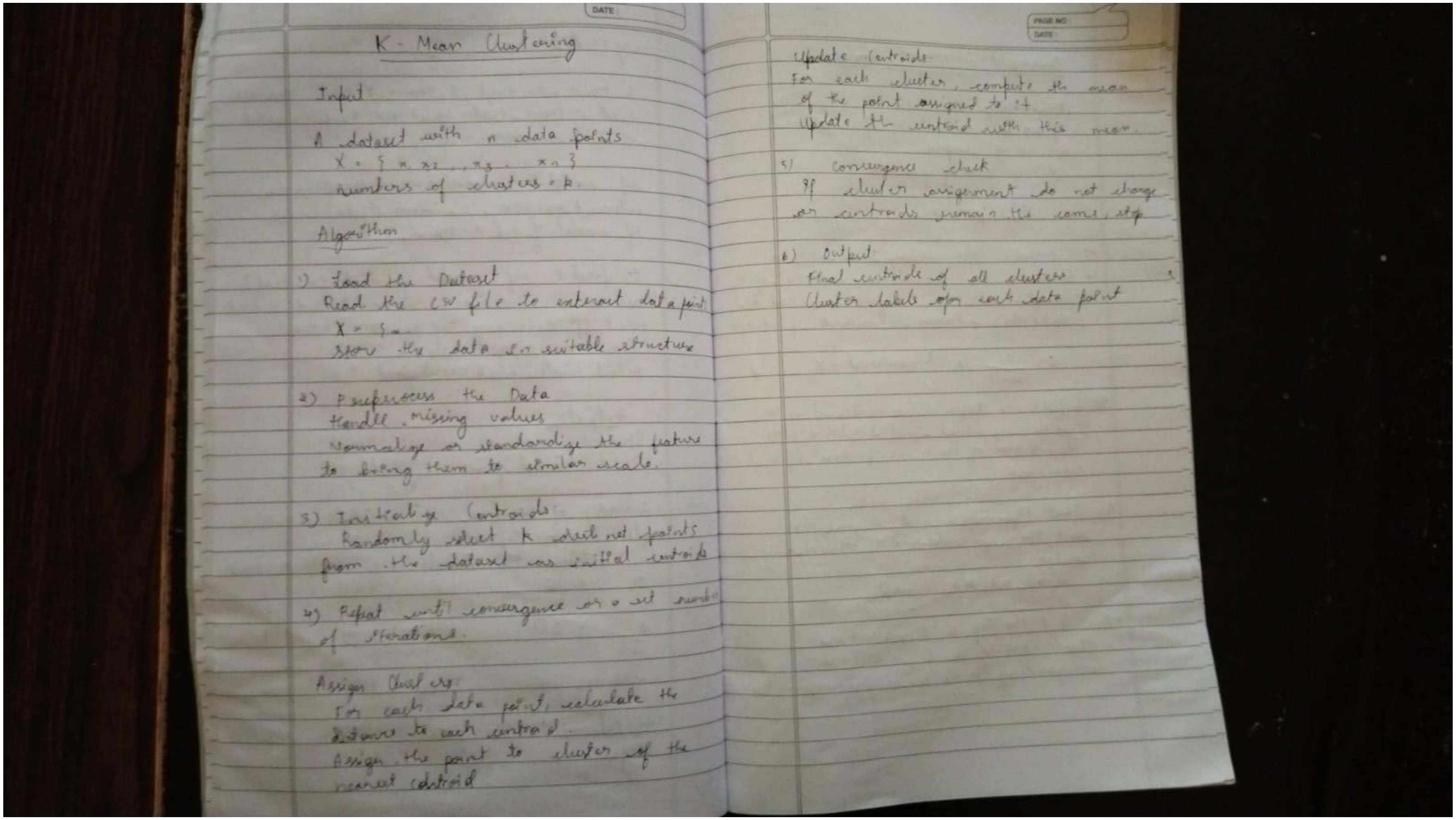
```
from
sklearn.ensemble import AdaBoostClassifier from
sklearn.datasets import load_iris from
sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# 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:



K - Mean Clustering

Input

A dataset with n data points
$X = \{ x_1, x_2, \ldots, x_3, \ldots x_n \}$
numbers of clusters = k.

Algorithm

1) Load the Dataset
Read the csv file to extract data point
$X = \{ \ldots$
stor the data in suitable structure

2) Preprocess the Data
Handle missing values
Normalize or standardize the feature
to bring them to similar scale.

3) Initialize Centroids
Randomly select k distinct points
from the dataset as initial centroids

4) Repeat until convergence or a set number
of iterations.

Assign Clusters:
For each data point, calculate the
distance to each centroid.
Assign the point to cluster of the
nearest centroid

Update Centroids
For each cluster, compute the mean
of the point assigned to it.
Update the centroid with this mean.

5) Convergence check
If cluster assignment do not change
or centroids remain the same, stop

6) Output
Final centroids of all clusters
Cluster labels for each data point

Code:

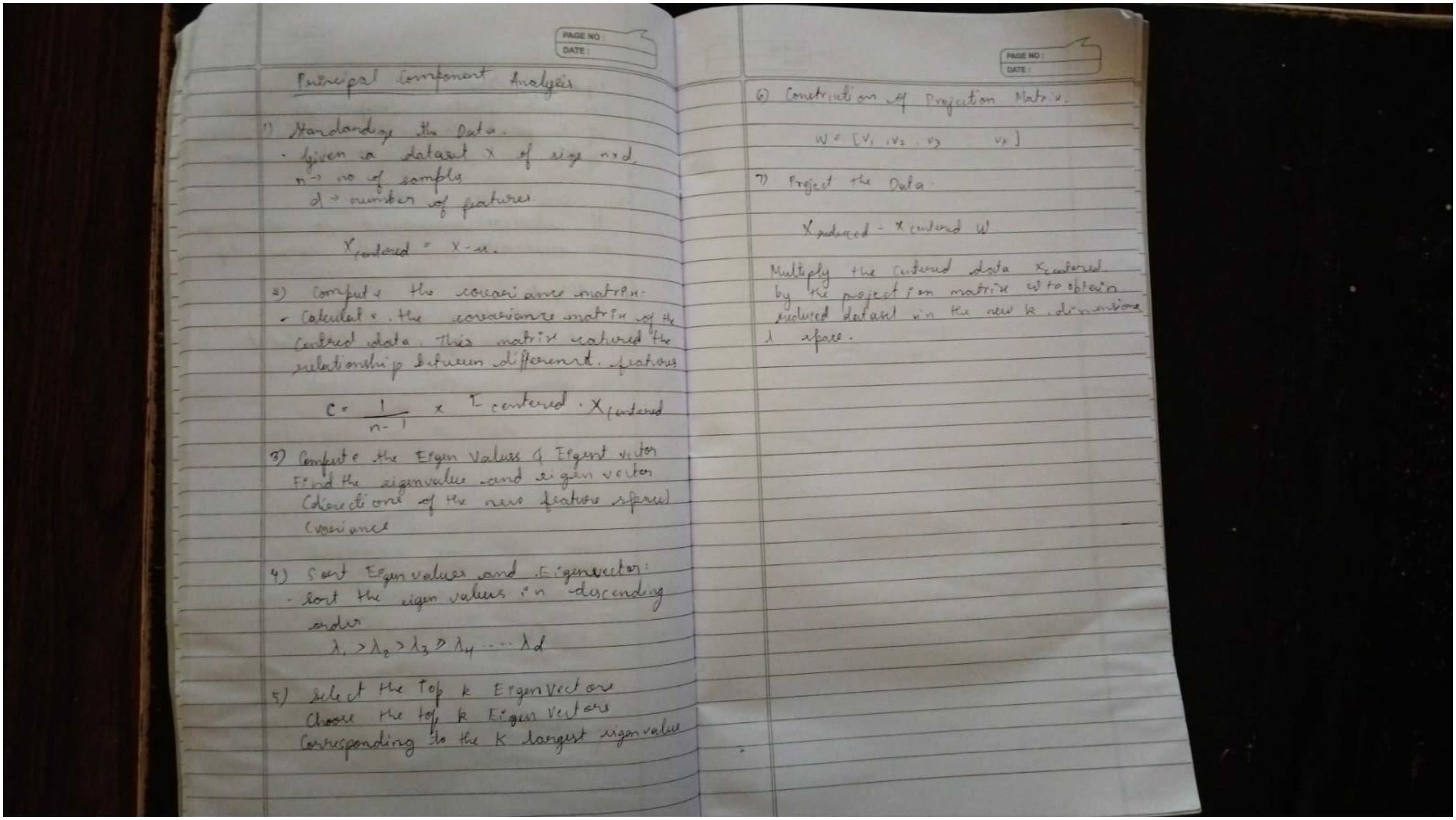
```
import pandas as pd

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:



PAGE NO :
DATE :

Principal Component Analysis

1) Standarding the Data.

- given a dataset x of size n×d.
  n→ no of sampls
  d→ number of features

$$X_{centered} = X - u.$$

2) Compute the covariance matrix:

- Calculate. the covariance matrix of the Centred data. This matrix cahured the relationship between differenrt. features

$$C = \frac{1}{n-1} \times {}^{T}\text{centered} \cdot X_{centered}$$

3) Compute the Eigen Values & Eigent vector
Find the eigenvalue and eigen vector (directions of the new feature space) (variance

4) Sort Eigenvalues and Eigenvector:

- sort the eigen values in descending order

$$\lambda_1 > \lambda_2 > \lambda_3 > \lambda_4 \cdots \lambda d$$

5) select the Top k EigenVector
Choose the top k Eigen Vectors Corresponding to the K largest eigenvalue

PAGE NO :
DATE :

6) Construction of Projection Matrix.

$$W = [v_1, v_2, v_3 \quad v_k]$$

7) Project the Data

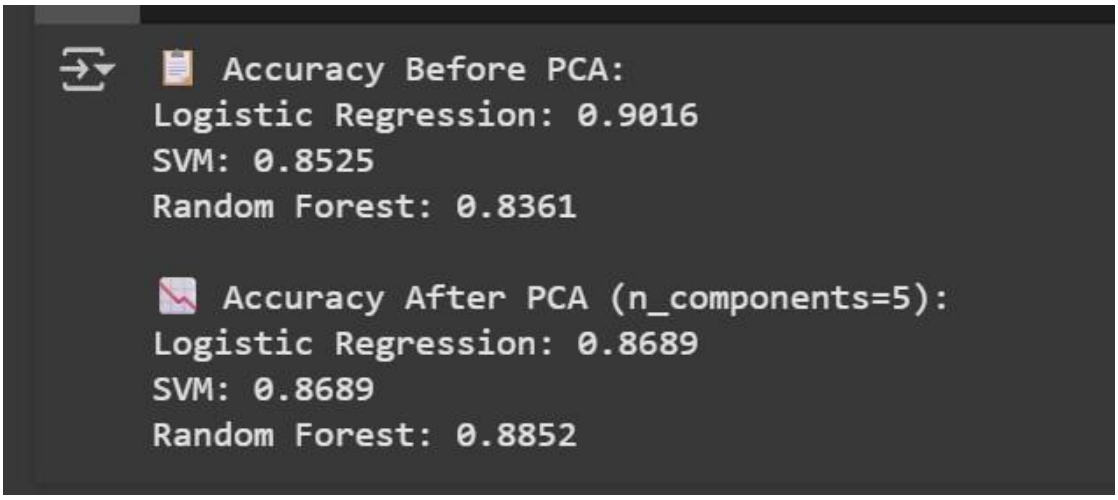
$$X_{reduced} = X_{centered} W$$

Multiply the centered data $X_{centered}$ by the projection matrix W to obtain reduced dataset in the new k-dimensional space.

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()
```



```
📋 Accuracy Before PCA:
Logistic Regression: 0.9016
SVM: 0.8525
Random Forest: 0.8361

📉 Accuracy After PCA (n_components=5):
Logistic Regression: 0.8689
SVM: 0.8689
Random Forest: 0.8852
```