VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum- 590014, Karnataka.

LAB REPORT

on

Machine Learning (23CS6PCMAL)

Submitted by

RAGHAVENDRA N (1BM22CS213)

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 **Raghavendra N (1BM22CS213),** 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 Incharges:	
Sneha S Bagalkot Assistant Professor Department of CSE BMSCE	**Dr. Kavitha Sooda** Professor & HOD Department of CSE, BMSCE

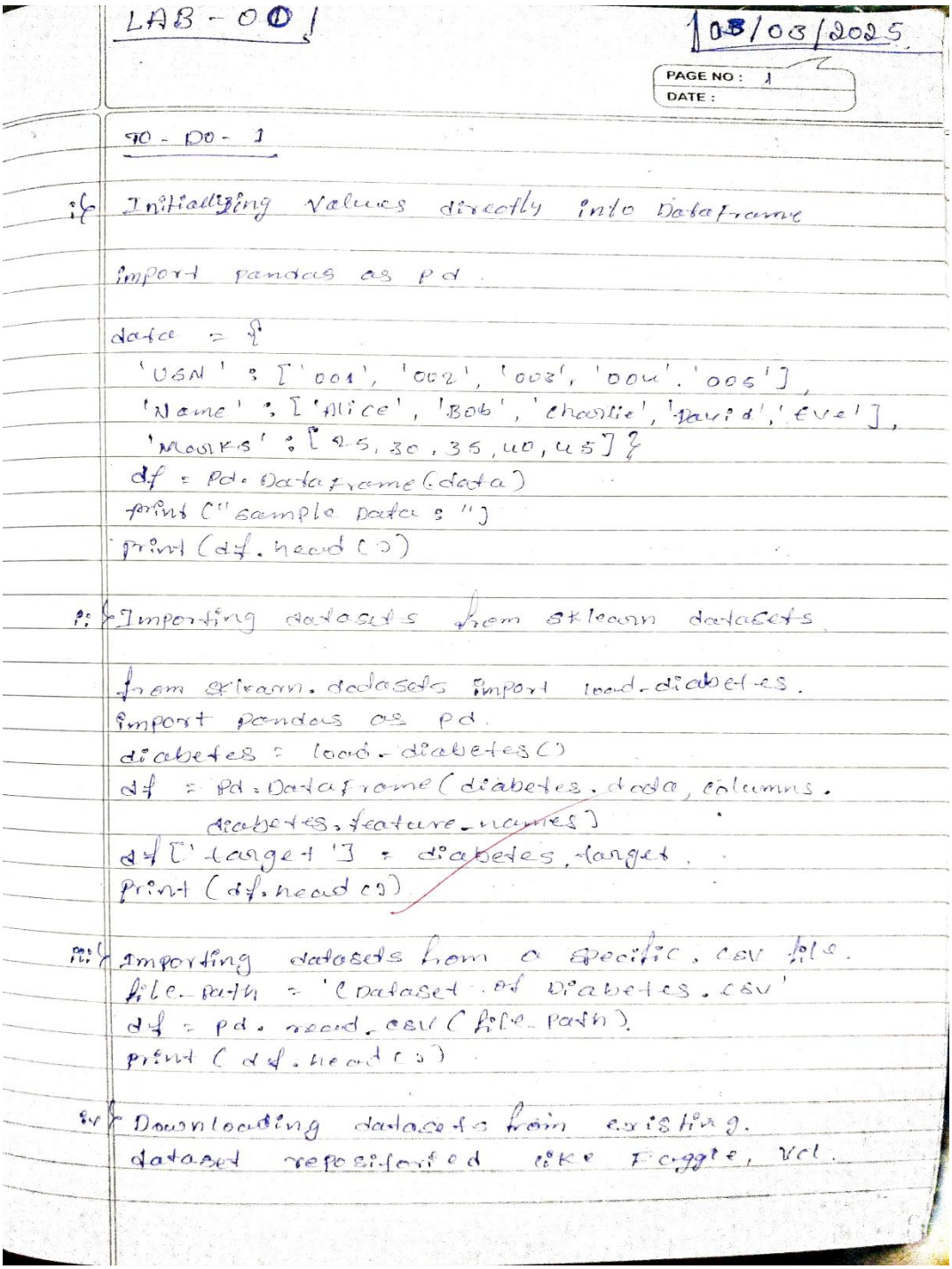
INDEX

Github Link: https://github.com/RaghavendraN-cs213/1BM22CS213_6TH_SEM_ML_LAB

Program 1

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

Screenshot:



LAB - 001

08/03/2025

PAGE NO : 1
DATE :

TO - DO - 1

i) Initializing values directly into Dataframe

```
import pandas as pd

data = {
    'USN' : ['001', '002', '003', '004', '005'],
    'Name' : ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
    'Marks' : [25, 30, 35, 40, 45] }
df = pd.DataFrame(data)
print("Sample Data : ")
print(df.head())
```

ii) Importing datasets from sklearn datasets

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

iii) Importing datasets from a specific .csv file.

```
file_path = 'Dataset of Diabetes.csv'
df = pd.read_csv(file_path)
print(df.head())
```

iv) Downloading datasets from existing dataset repositories like Kaggle, UCI.

PAGE NO : 2
DATE :

```
df = pd.read_csv('/Dataset of Diabetes (1).csv')
print("Sample data:")
print(df.head())
```

Task-Do-2

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

tickers = ['HDFCBANK.NS', "ICICIBANK.NS",
           "KOTAKBANK.NS"]
start_date = "2024-01-01"
end_date = "2024-12-30"

data = yf.download(tickers, start = start_date,
       end = end_date, group_by = 'tickers')
for ticker in tickers:
    stock_data = data[ticker]
    stock_data = ['Daily Return'] = stock_data
    ['close'].pct_change()
    plt.subplot(2,1,1)
    stock_data['close'].plot(title = f"{ticker}
        closing_price")
    plt.ylabel("Price (INR)")
    plt.subplot(2,1,2)
    stock_data['Daily Returns", color = 'orange')
    f"{ticker} - Daily Returns", color = 'orange')
    plt.ylabel("Daily Return")
    plt.tight_layout()
    plt.show()
```

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 | 10/08/2025

Data pre-processing techniques.

PAGE NO: 3

DATE:

(i) import pandas as pd.
df = pd.read_csv("housing.csv")
print("Data loaded into Data Frame")

(ii) print("\n Stastical information: ")
print(df.describe())

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

(iv) print("\n Count of unique labels for ocean proximity column: ")
print(df['ocean proximity'].value_counts())

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

1 Diabetes Dataset: Columns like Glucose Blood pressure and BMI had null handled by imputing mean and median

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

(d) Diabetes Dataset: the outcome column in categorical, encoded using label Encoding.

PAGE NO : 4
DATE :

Adult Income Dataset columns like work class, education were categorical & removed using one-Hot Encoding.

(3) Min-max Scaling scale the data to a fixed range (0 to 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 assumed a normal distribution.

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']])
```

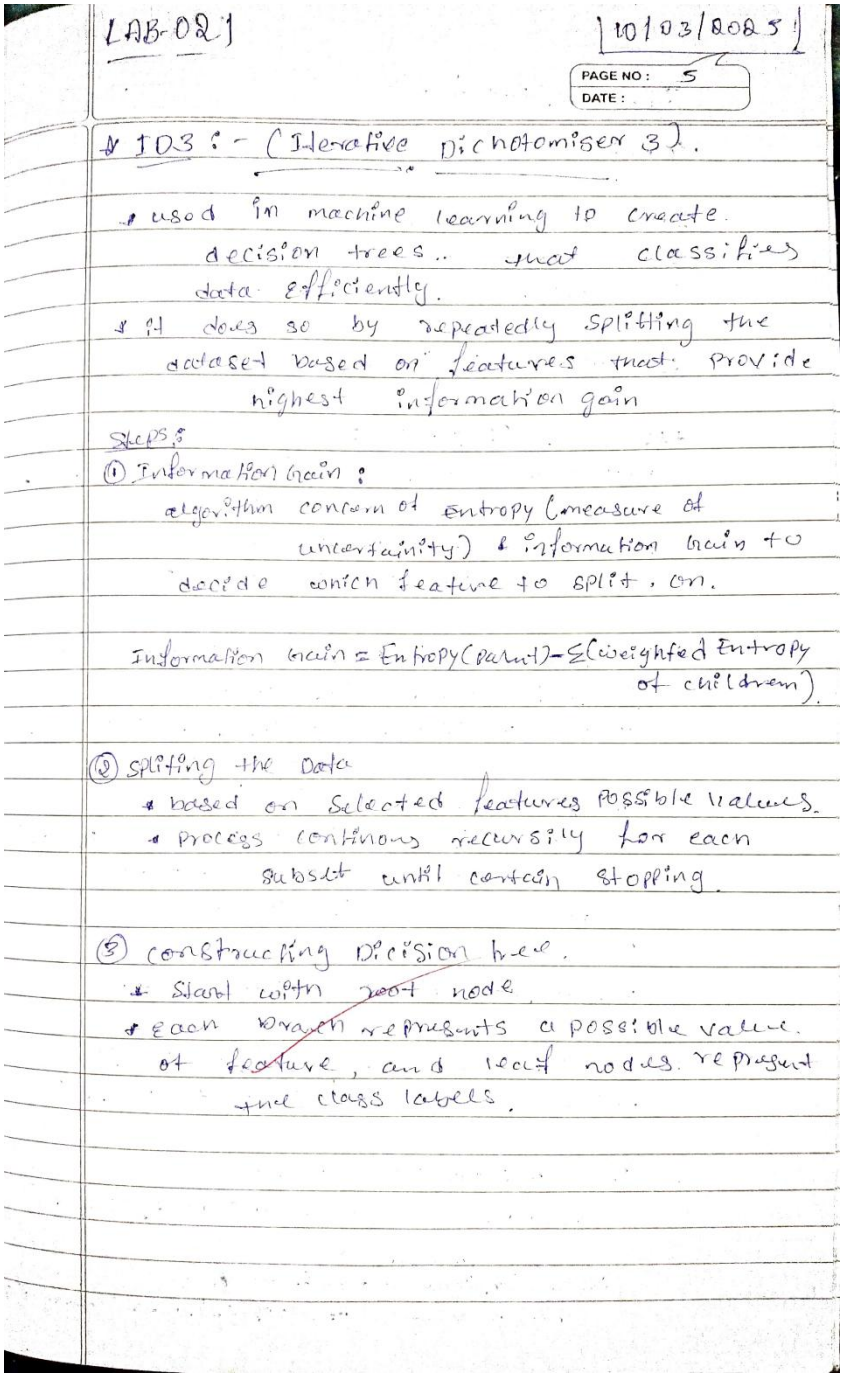
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler


# 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-02 | 10/03/2025

PAGE NO : 5
DATE :

* ID3 :- (Iterative Dichotomiser 3).

* used in machine learning to create decision trees.. that classifies data efficiently.
* It does so by repeatedly splitting the dataset based on features that provide highest information gain

Steps:

① Information Gain :
algorithm concern of Entropy (measure of uncertainity) & information Gain to decide which feature to split, on.

Information Gain = Entropy(Parent) - Σ(weighted Entropy of children).

② Spliting the Data
* based on selected features possible values.
* Process continous recursily for each subset until certain stopping.

③ constructing Decision tree.
* Start with root node
* each branch represents a possible value of feature, and leaf nodes represent the class labels.

LAB-08)

17/03/2025

PAGE NO : 6
DATE :

ID3: implementation

Code:-

```
import numpy as np
import pandas as pd
from collections import Counter

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

def entropy(target_col):
    values, counts = np.unique(target_col, return_
                               counts = true)
    probs = counts / counts.sum()
    return -np.sum(probs * np.log2(probs))

def information_gain(df, feature, target = "Decision"):
    total_entropy = entropy(df[target])
    values, counts = np.unique(df[feature], return_
                               counts = True)
    weighted_entropy = sum(
        (counts[i] / sum(counts)) * entropy(df[df[
            feature] == values[i]][target])
        for i in range(len(values))
    )
    return total_entropy - weighted_entropy

def id3(df, features, target = "Decision"):
    unique_classes = np.unique(df[target])

    if len(unique_classes) == 1:
        return unique_classes = np.unique(df[target])
    if len(features) == 0:
        return Counter(df[target]).most_common(1)
```

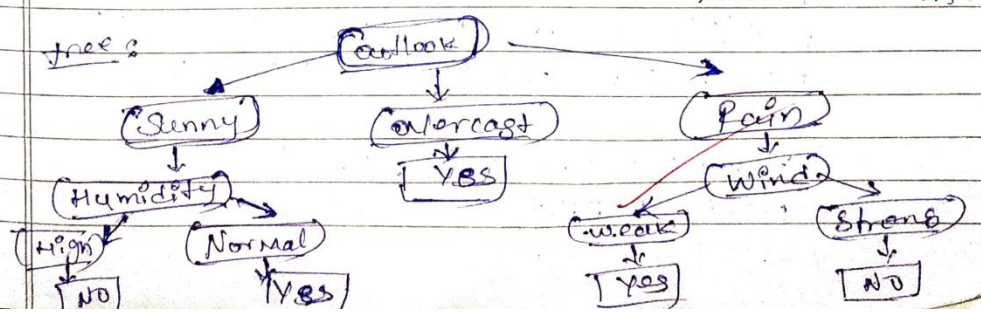
PAGE NO :
DATE :

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

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

tree = {best_feature : {}}
for value in np.unique(df[best_feature]):
    subset = df[df[best_feature] == value].
        drop(columns = [best_feature])
    tree[best_feature][value] = id3(subset,
        [f for f in features if f != best_feature],
        target)

return tree

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

import pprint
pprint.pprint(decision_tree)
```

output :

```
{'outlook': {'Overcast': 'Yes',
             'Rain': {'wind': {'Strong': 'No',
                               'weak': 'Yes'}},
             'Sunny': {'Humidity': {'High': 'No',
                                    'Normal': 'Yes'}}}}
```

tree :



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):
```

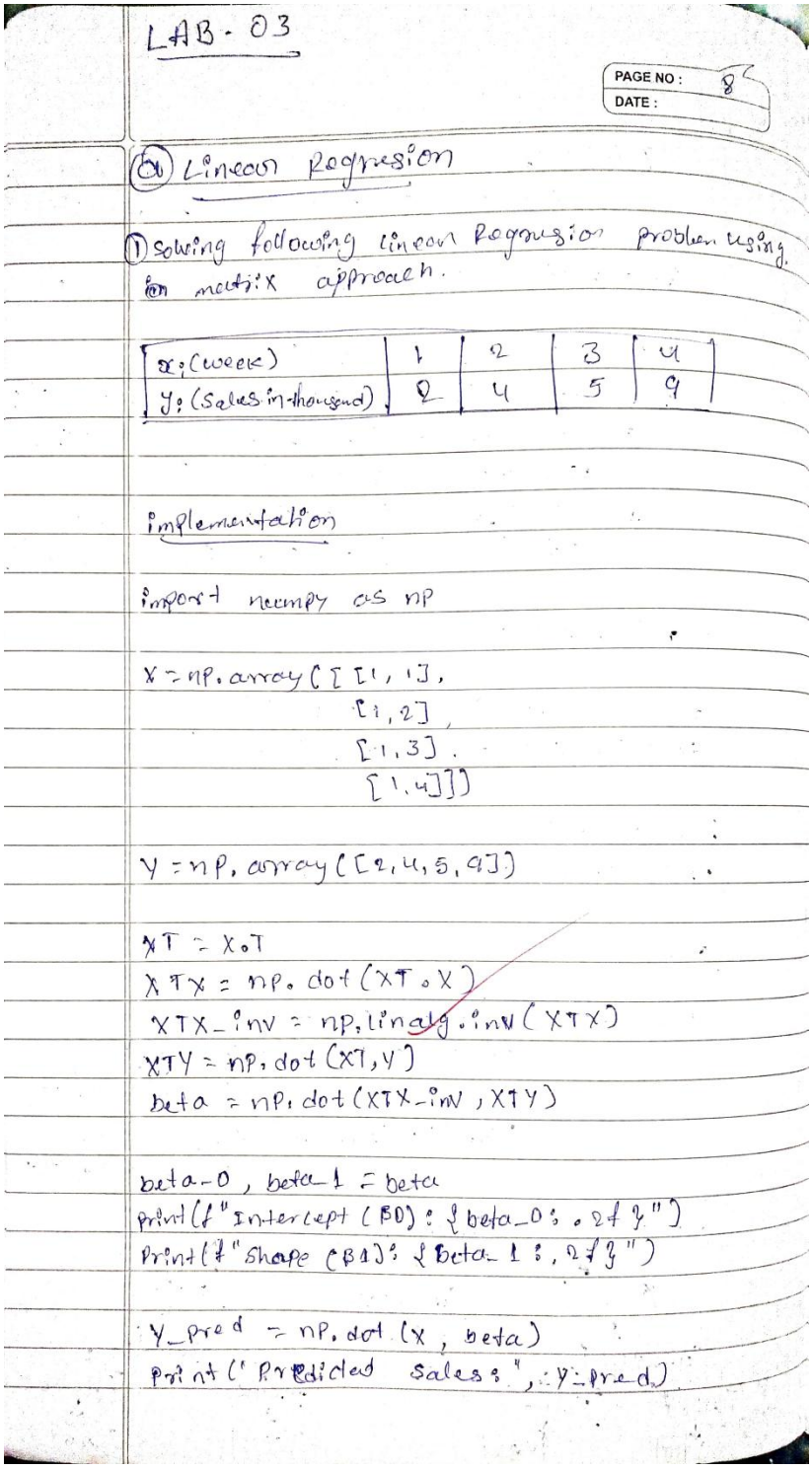
```
    if not isinstance(tree, dict):
        return tree
    feature = list(tree.keys())[0]
    value = example[feature]
    return classify(tree[feature][value], example)

# 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:



LAB - 03

PAGE NO : 8
DATE :

(a) Linear Regression

(1) solving following linear Regression problem using matrix approach.

x: (week)	1	2	3	4
y: (sales in thousand)	2	4	5	9

implementation

```
import numpy as np

X = np.array([[1, 1],
              [1, 2],
              [1, 3],
              [1, 4]])

Y = np.array([2, 4, 5, 9])

XT = X.T
XTX = np.dot(XT, X)
XTX_inv = np.linalg.inv(XTX)
XTY = np.dot(XT, Y)
beta = np.dot(XTX_inv, XTY)

beta_0, beta_1 = beta
print(f"Intercept (B0): {beta_0:.2f}")
print(f"Shape (B1): {beta_1:.2f}")

Y_pred = np.dot(X, beta)
print("Predicted sales:", Y_pred)
```

PAGE NO : 9
DATE :

Output :

Intercept (B_0) : −0.50

slope (B_1) : 2.20

predicted Sales : [1.7, 3.9, 6.1, 8.3]

③ Multiple Linear Regression

Q. predict Glucose level of diabetes_data.csv using MLR.

diabetes_data.csv :

Age	BMI	BP	Insulin	D.PedigreFun	Glucose.
50	25.3	80	150	0.5	140
⋮	⋮	⋮	⋮	⋮	⋮
33	29.1	81	145	0.67	160.

implementation

```
import numpy as np
import pandas as pd
df = pd.read_csv("diabetes_data.csv");

X = df[['Age', 'BMI', 'BP', 'Insulin', 'D.PedigreeFun']].
                                        values
y = df['Glucose'].values

X = np.c_[np.ones(X.shape[0]), X]
```

PAGE NO :
DATE :

```
train_size = int(0.8 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

theta = np.linalg.inv(X_train.T @ X_train) @
        X_train.T @ y_train.

y_pred = X_test @ theta

mse = np.mean((y_test - y_pred) ** 2)
r2 = 1 - (np.sum((y_test - y_pred) ** 2) / np.sum
     ((y_test - np.mean(y_test)) ** 2))

print("Multiple Linear Regression model (Age, BMI, I, DP→Hb)")
print(f"Mean Squared Error : {mse}")
print(f"R^2 score : {r2}")
```

Output :-

MLR :-

Mean Squared Error : 41.23512336576399

R^2 score : 0.9312747443903934

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

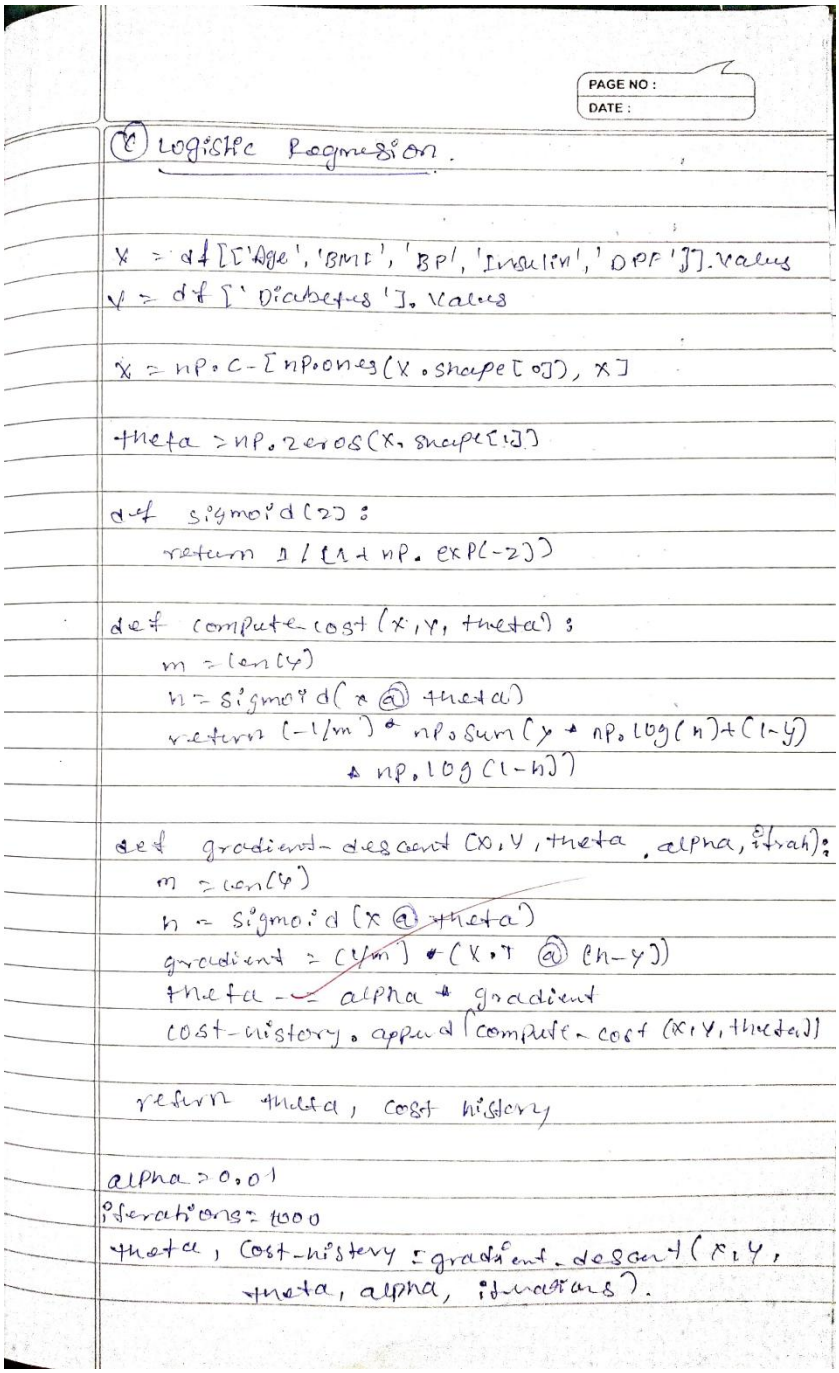
```
import pandas as pd

# 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_
```

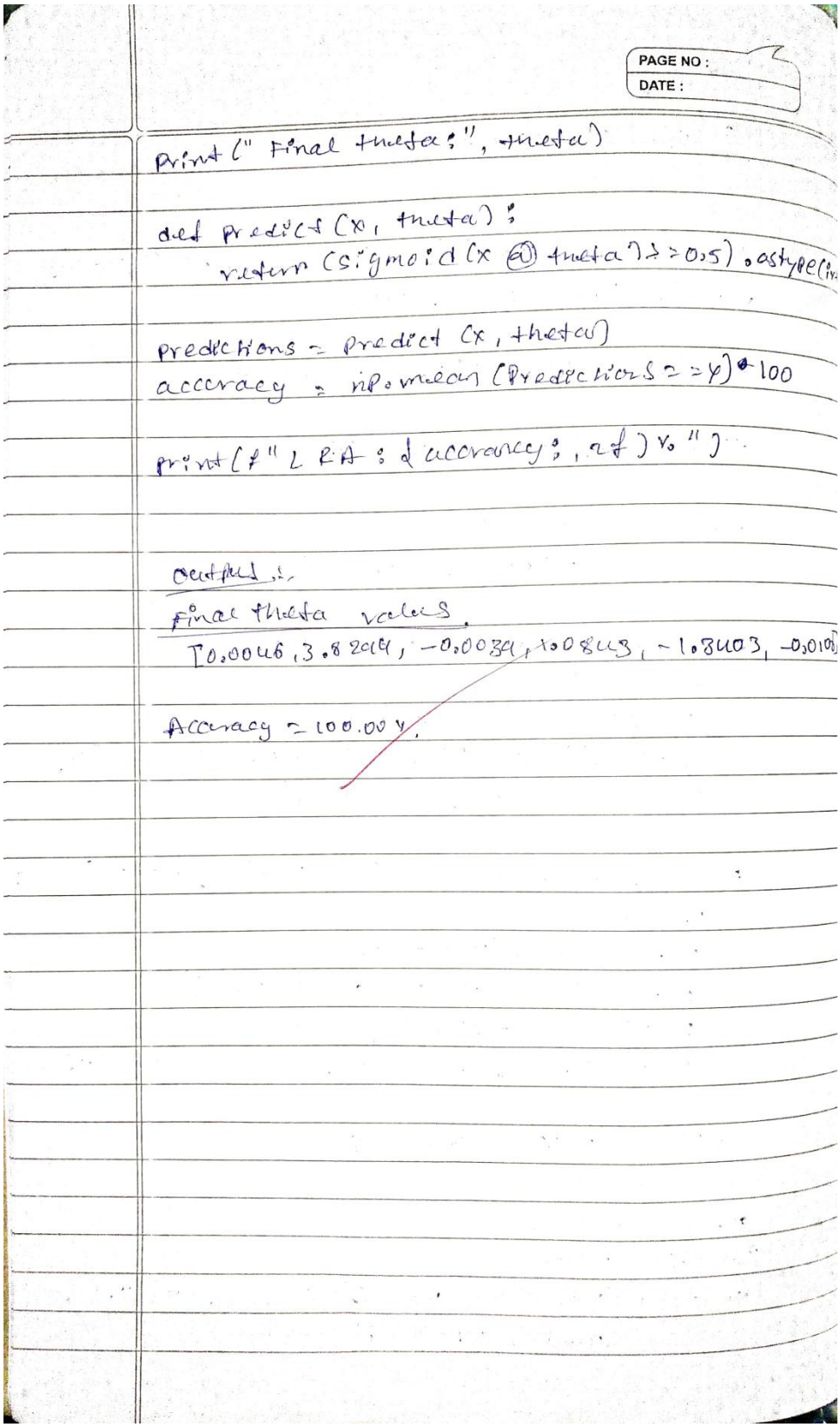
Program 5

Build Logistic Regression Model for a given dataset

Screenshot:



```
PAGE NO :
DATE :

(8) Logistic Regression.

X = df[['Age', 'BMI', 'BP', 'Insulin', 'DPF']].values
y = df['Diabetes'].values

X = np.c_[np.ones(X.shape[0]), X]

theta = np.zeros(X.shape[1])

def sigmoid(z):
    return 1/(1+np.exp(-z))

def compute_cost(X, y, theta):
    m = len(y)
    h = sigmoid(X @ theta)
    return (-1/m) * np.sum(y * np.log(h) + (1-y)
                * np.log(1-h))

def gradient_descent(X, y, theta, alpha, iterations):
    m = len(y)
    h = sigmoid(X @ theta)
    gradient = (1/m) * (X.T @ (h-y))
    theta -= alpha * gradient
    cost_history.append(compute_cost(X, y, theta))

    return theta, cost_history

alpha = 0.01
iterations = 1000
theta, cost_history = gradient_descent(X, y,
        theta, alpha, iterations).
```



PAGE NO :
DATE :

print (" Final theta :", theta)

def predict (x, theta) :
    return (sigmoid (x @ theta) >= 0.5).astype(in

predictions = predict (x, theta)
accuracy = np.mean (predictions == y) * 100

print (f" LRA : {accuracy :.2f} % ")

Output :-
Final theta values.
[0.0046, 3.8204, -0.0034, 1.0843, -1.8403, -0.0108]

Accuracy = 100.00 %.

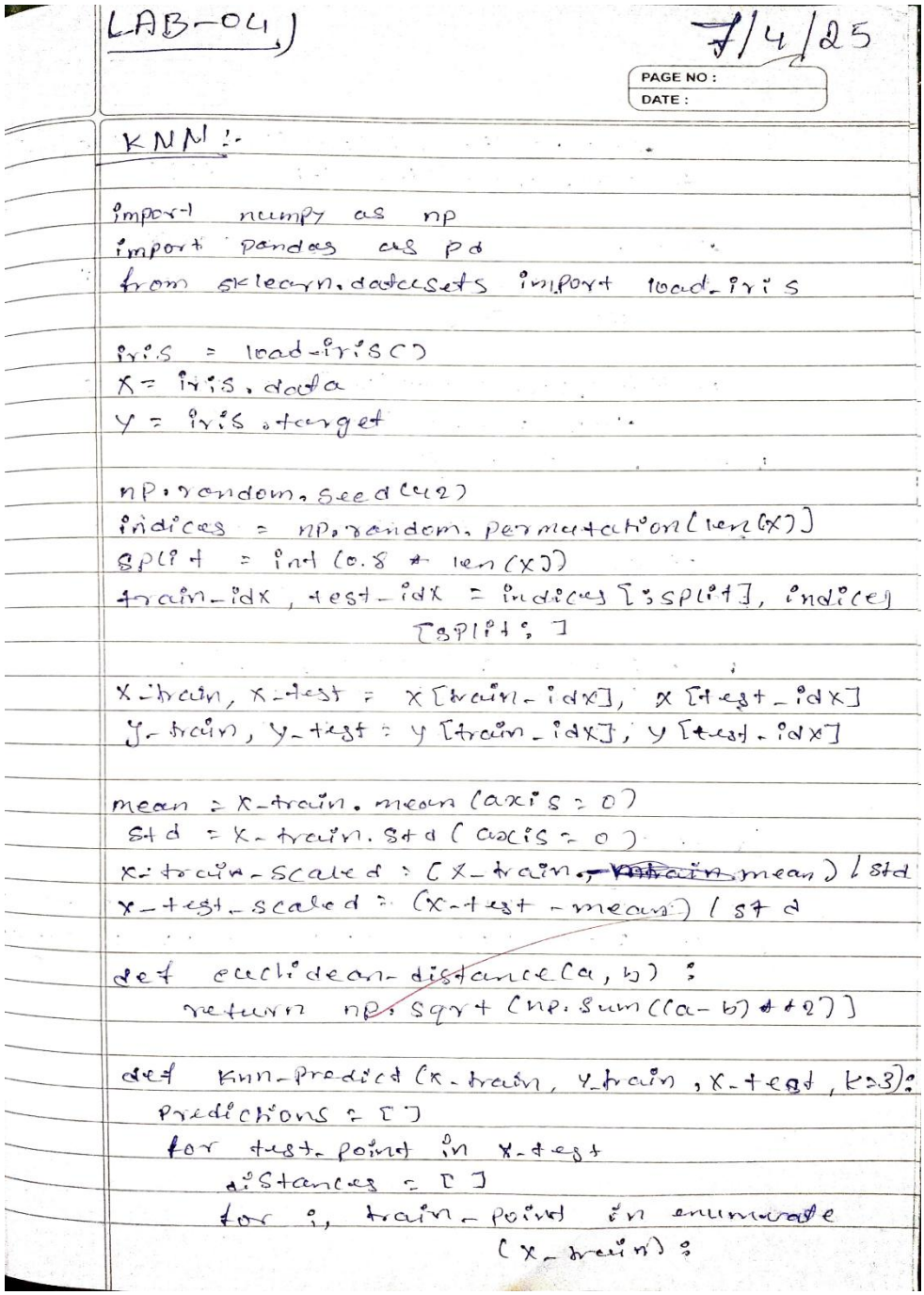
Code:

```
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:



LAB-04) 7/4/25

PAGE NO :
DATE :

KNN :-

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()
X = iris.data
y = iris.target

np.random.seed(42)
indices = np.random.permutation(len(X))
split = int(0.8 * len(X))
train_idx, test_idx = indices[:split], indices
                      [split:]

X_train, X_test = X[train_idx], X[test_idx]
y_train, y_test = y[train_idx], y[test_idx]

mean = X_train.mean(axis=0)
std = X_train.std(axis=0)
X_train_scaled = (X_train - mean) / std
X_test_scaled = (X_test - mean) / std

def euclidean_distance(a, b):
    return np.sqrt(np.sum((a-b)**2))

def knn_predict(X_train, y_train, X_test, k=3):
    predictions = []
    for test_point in X_test
        distances = []
        for i, train_point in enumerate
                    (X_train):
```

PAGE NO :
DATE :

```
dist = Euclidian_distance (test_point, train_point)
distances.append((dist, y_train[i]))

distances.sort(key=lambda x: x[0])
K_nearest_labels = [label for _, label in
    distances[:K]]

predicted_label = max(set(K_nearest_labels),
    key=K_nearest_labels.count)
return np.array(predictions)

y_pred = knn_predict(X_train_scaled, y_train,
    X_test_scaled, K=3)
accuracy = np.mean(y_pred == y_test)

print(f"Accuracy: {accuracy:.2f}")
print("Predictions:", y_pred)
print("True labels:", y_test
```

Output →

Accuracy: 0.97

Predictions: 1 0 2 1 0 1 2 2 0 1 2 2 0 2 0 1 2 1 1 2 2 0 1 0 1 2

true label: 1 0 1 1 0 1 2 2 0 1 2 2 0 2 0 1 2 1 1 2 2 0 1 2 0 1 2

Code:

KNN

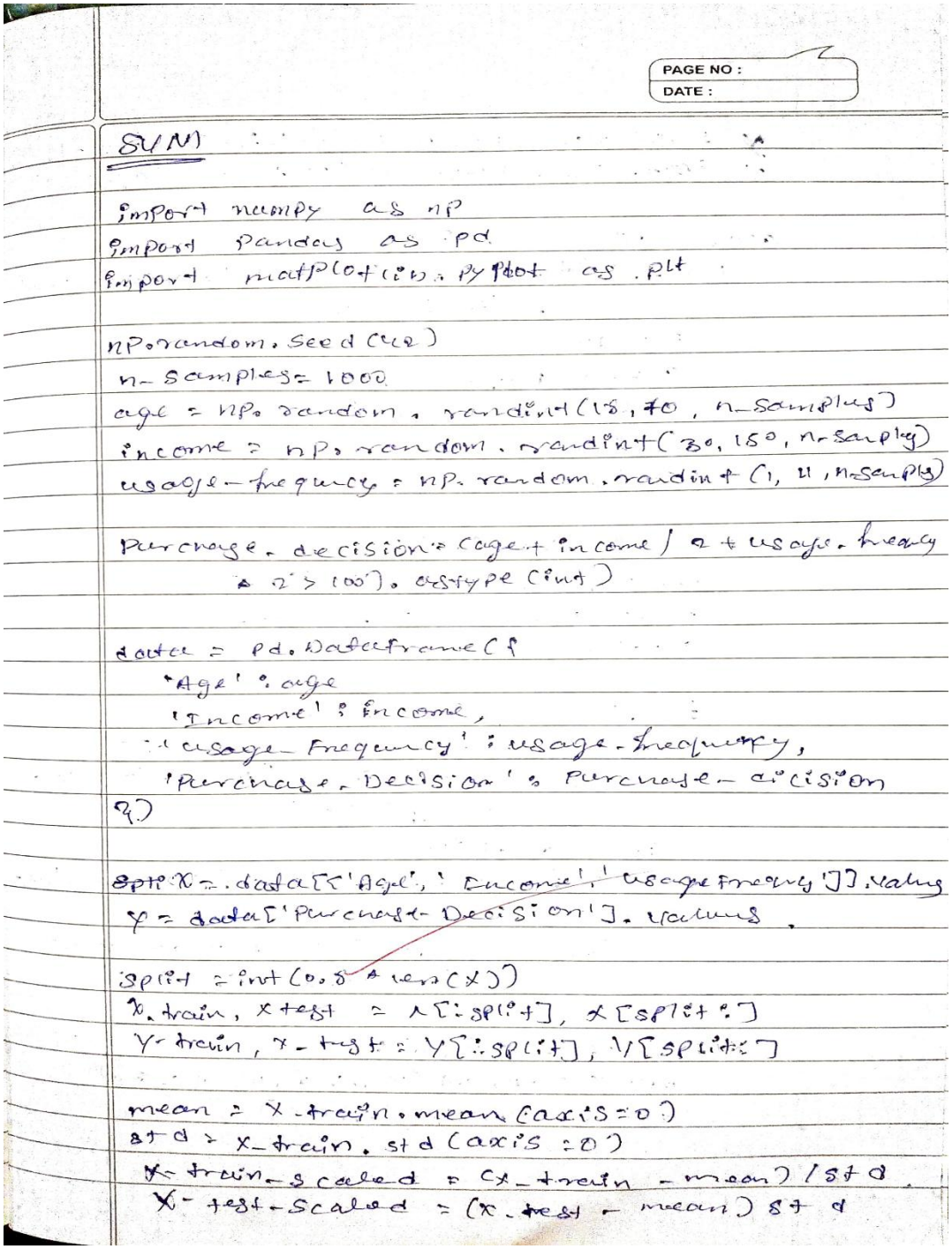
```
import numpy as np
from collections import Counter


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:



SVM

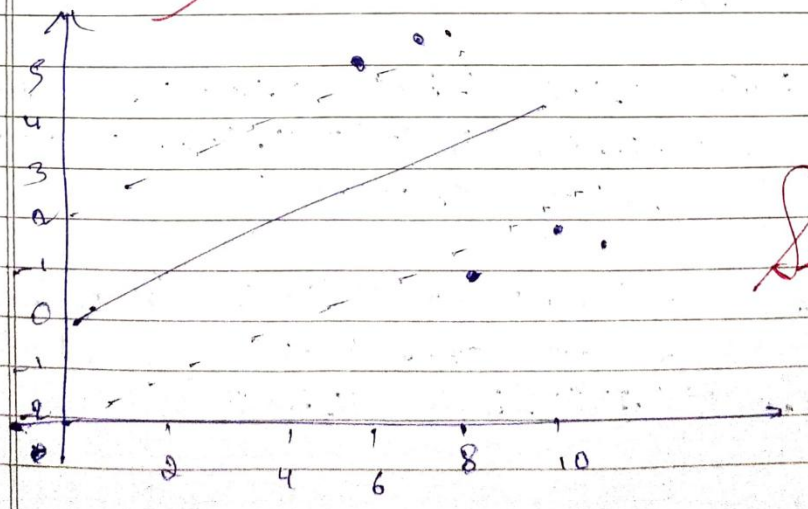
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

np.random.seed(42)
n_samples = 1000
age = np.random.randint(18, 70, n_samples)
income = np.random.randint(30, 150, n_samples)
usage_frequency = np.random.randint(1, 11, n_samples)

Purchase_decision = (age + income / 2 + usage_frequency
    * 2 > 100).astype(int)

data = pd.DataFrame({
    'Age': age,
    'Income': income,
    'Usage_Frequency': usage_frequency,
    'Purchase_Decision': Purchase_decision
})

split = int(0.8 * len(X))
X_train, X_test = X[:split], X[split:]
y_train, y_test = y[:split], y[split:]

mean = X_train.mean(axis=0)
std = X_train.std(axis=0)
X_train_scaled = (X_train - mean) / std
X_test_scaled = (X_test - mean) / std
```

PAGE NO :
DATE :

```
svm = LinearSVM()
svm.fit(X_train_scaled, y_train_svm)

predictions = svm.predict(X_test_scaled)
predicted_labels = np.where(predictions >= 1, 0, 1)

accuracy = np.mean(predicted_labels == y_test)
print("Accuracy is {accuracy:.2f}")

plt.figure(figsize=(8, 6))
plt.scatter(X_test[:, 0], X_test
plt.xlabel('Age (scaled)')
plt.ylabel('Income (scaled)')
plt.title('Custom SVM clan: Age vs Inco')
```

Output :

Accuracy of custom SVM: 0.52



SB
7/4/25

PAGE NO :
DATE :

```
y_train_svm = np.where(y_train == 0, -1, 1)
y_test_svm = np.where(y_test == 0, -1, 1)

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

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.w = np.zeros(n_features)
        self.b = 0

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

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

Code:

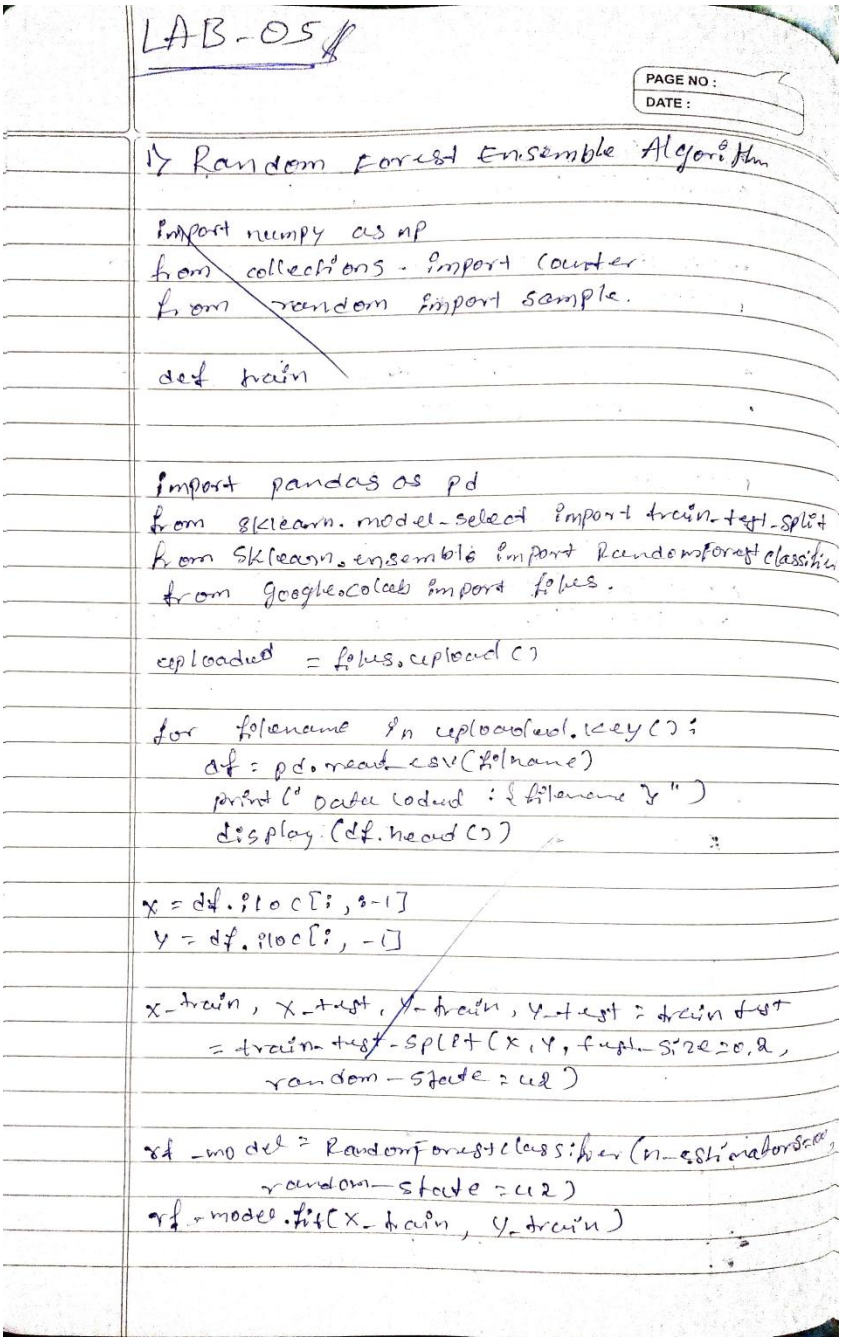
```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

# 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:

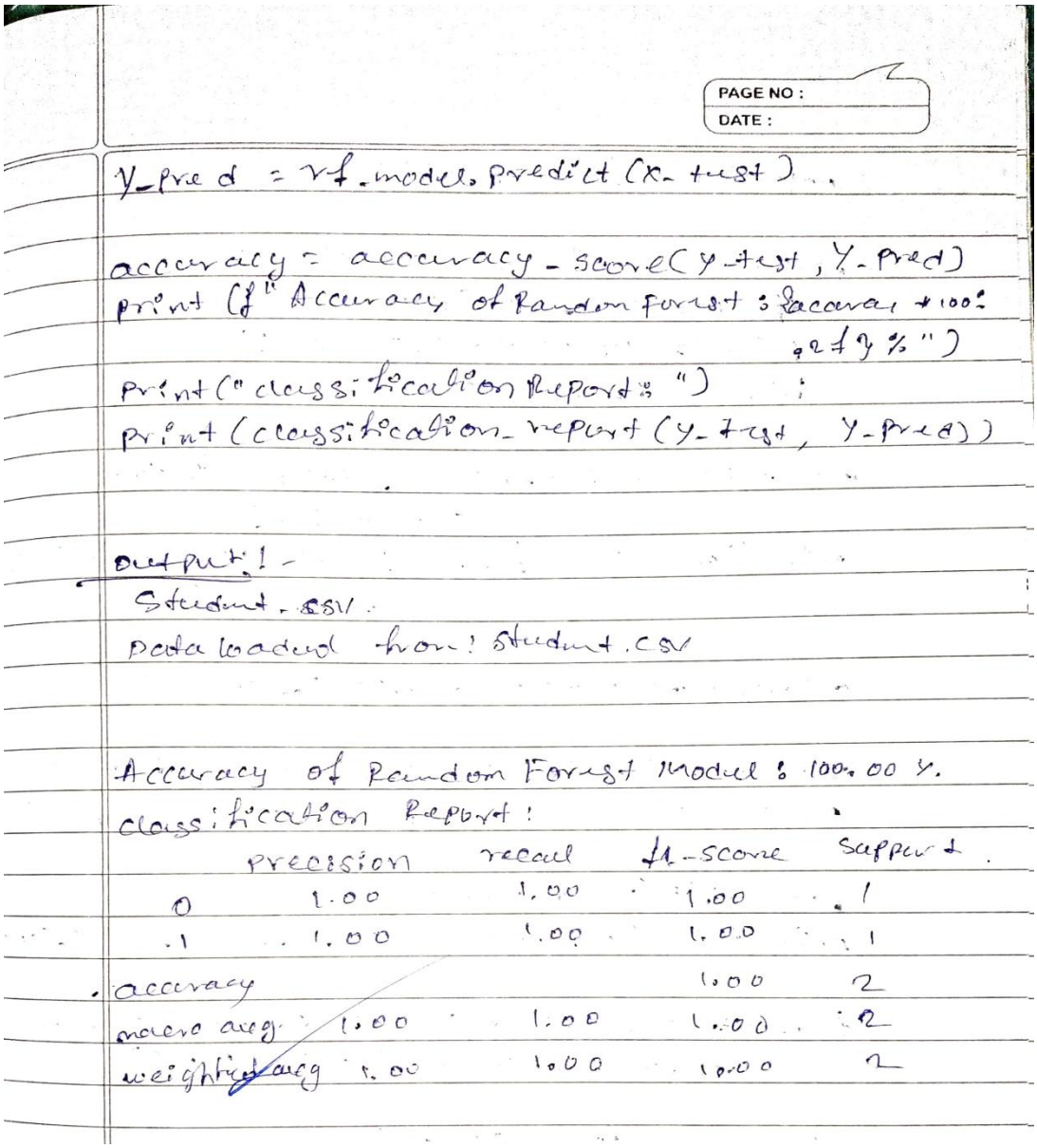


LAB-05

PAGE NO :
DATE :

1) Random Forest Ensemble Algorithm

```
import numpy as np
from collections import Counter
from random import sample.

def train


import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from google.colab import files.

uploaded = files.upload()

for filename in uploaded.keys():
    df = pd.read_csv(filename)
    print('Data loaded : {filename}')
    display(df.head())

x = df.iloc[:, :-1]
y = df.iloc[:, -1]

x_train, x_test, y_train, y_test = train_test
    = train_test_split(x, y, test_size=0.2,
        random_state = 42)

rf_model = RandomForestClassifier(n_estimators=100,
        random_state=42)
rf_model.fit(x_train, y_train)
```



PAGE NO :
DATE :

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 : ")
print(classification_report(y_test, Y_pred))

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
accuracy			1.00	2
macro avg	1.00	1.00	1.00	2
weighted avg	1.00	1.00	1.00	2

Code:

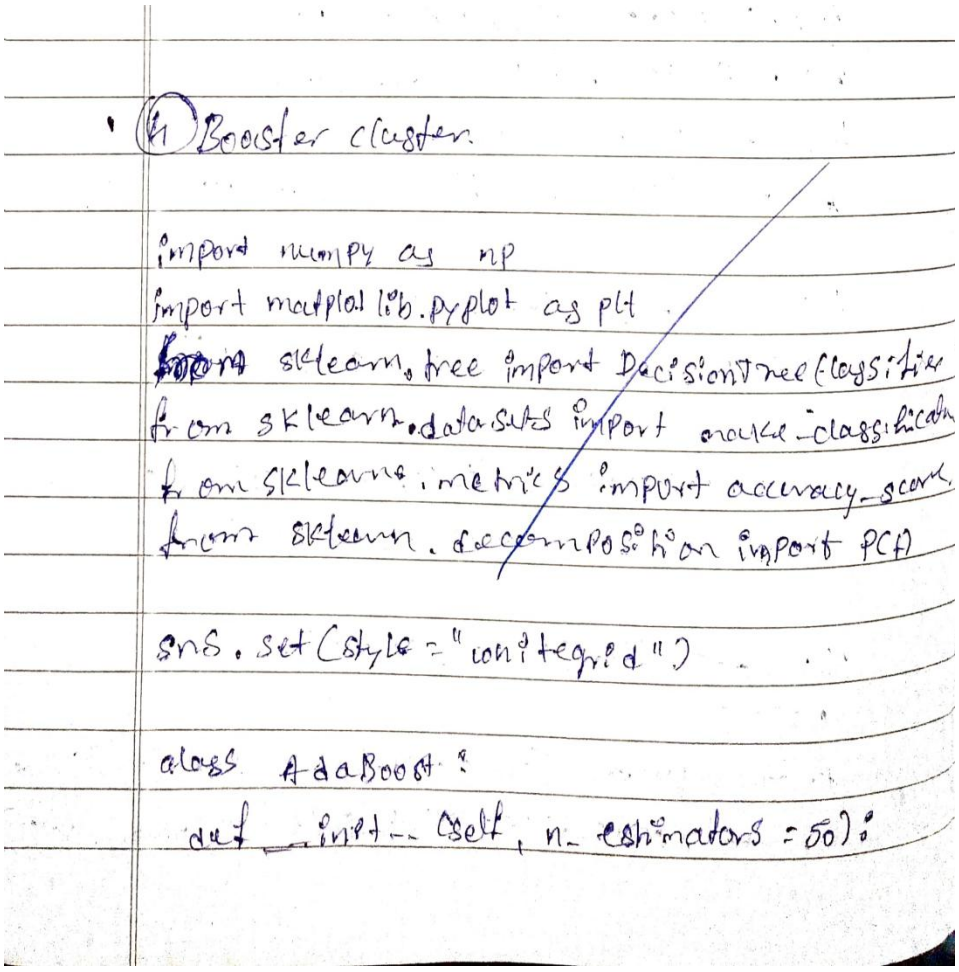
```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score

# Load sample dataset
```

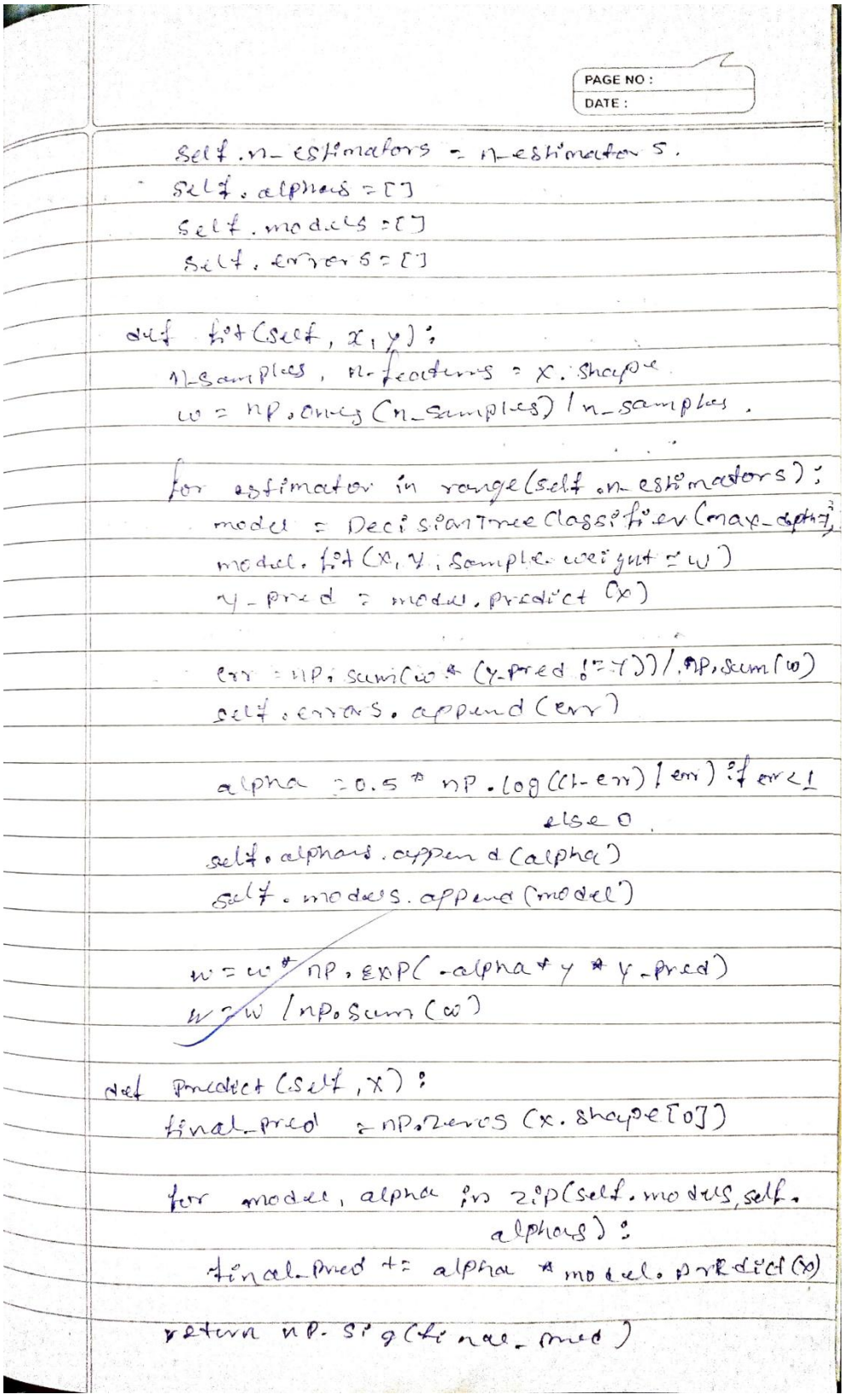
```
iris = load_iris()

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

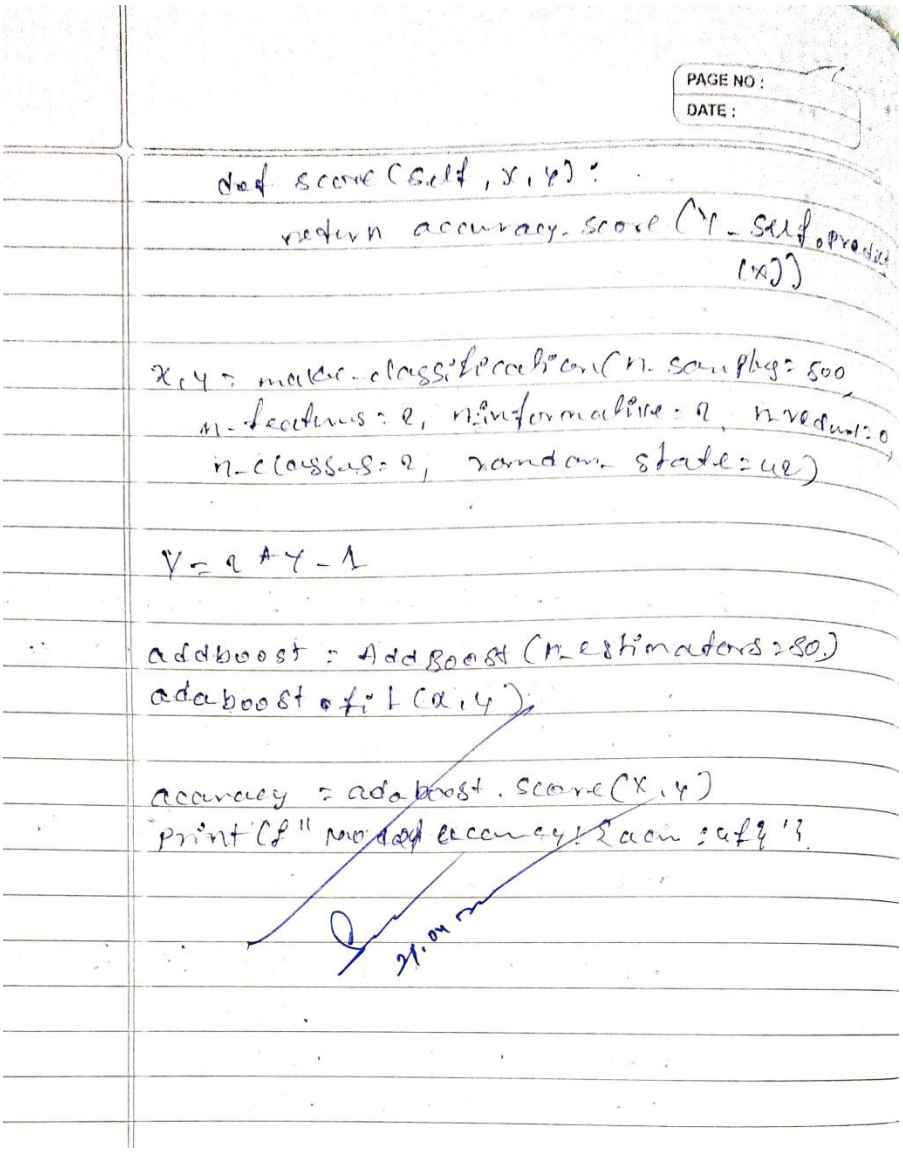
Screenshot:





```
PAGE NO :
DATE :

self.n_estimators = n_estimators.
self.alphas = []
self.models = []
self.errors = []

def fit(self, x, y):
    n_samples, n_features = x.shape
    w = np.ones(n_samples) / n_samples.

    for estimator in range(self.n_estimators):
        model = DecisionTreeClassifier(max_depth=1)
        model.fit(x, y, sample_weight = w)
        y_pred = model.predict(x)

        err = np.sum(w * (y_pred != y)) / np.sum(w)
        self.errors.append(err)

        alpha = 0.5 * np.log((1-err) / err) if err < 1
                                        else 0.
        self.alphas.append(alpha)
        self.models.append(model)

        w = w * np.exp(-alpha * y * y_pred)
        w = w / np.sum(w)

def predict(self, x):
    final_pred = np.zeros(x.shape[0])

    for model, alpha in zip(self.models, self.
                                    alphas):
        final_pred += alpha * model.predict(x)

    return np.sign(final_pred)
```

V



Code:

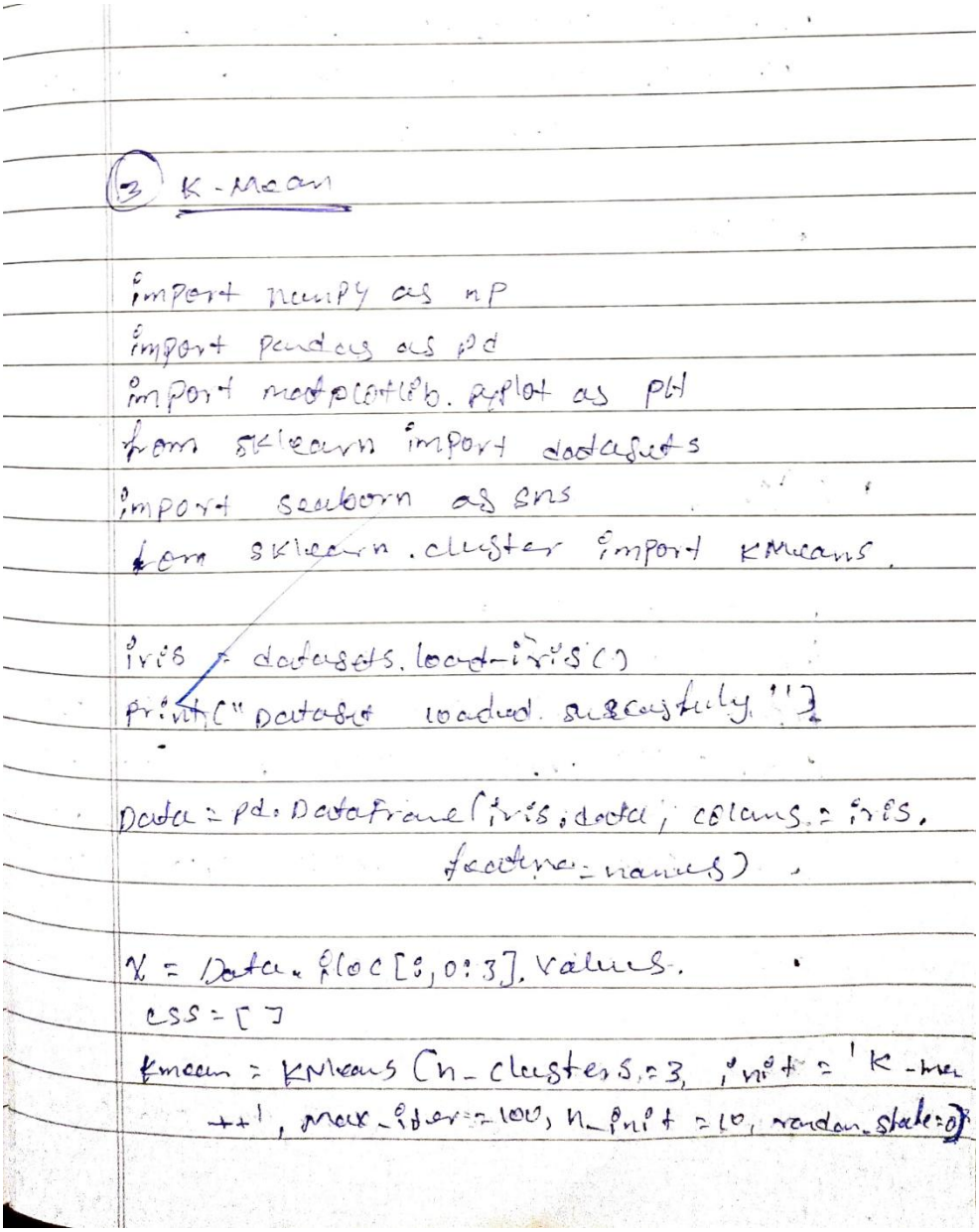
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```
# 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))
```

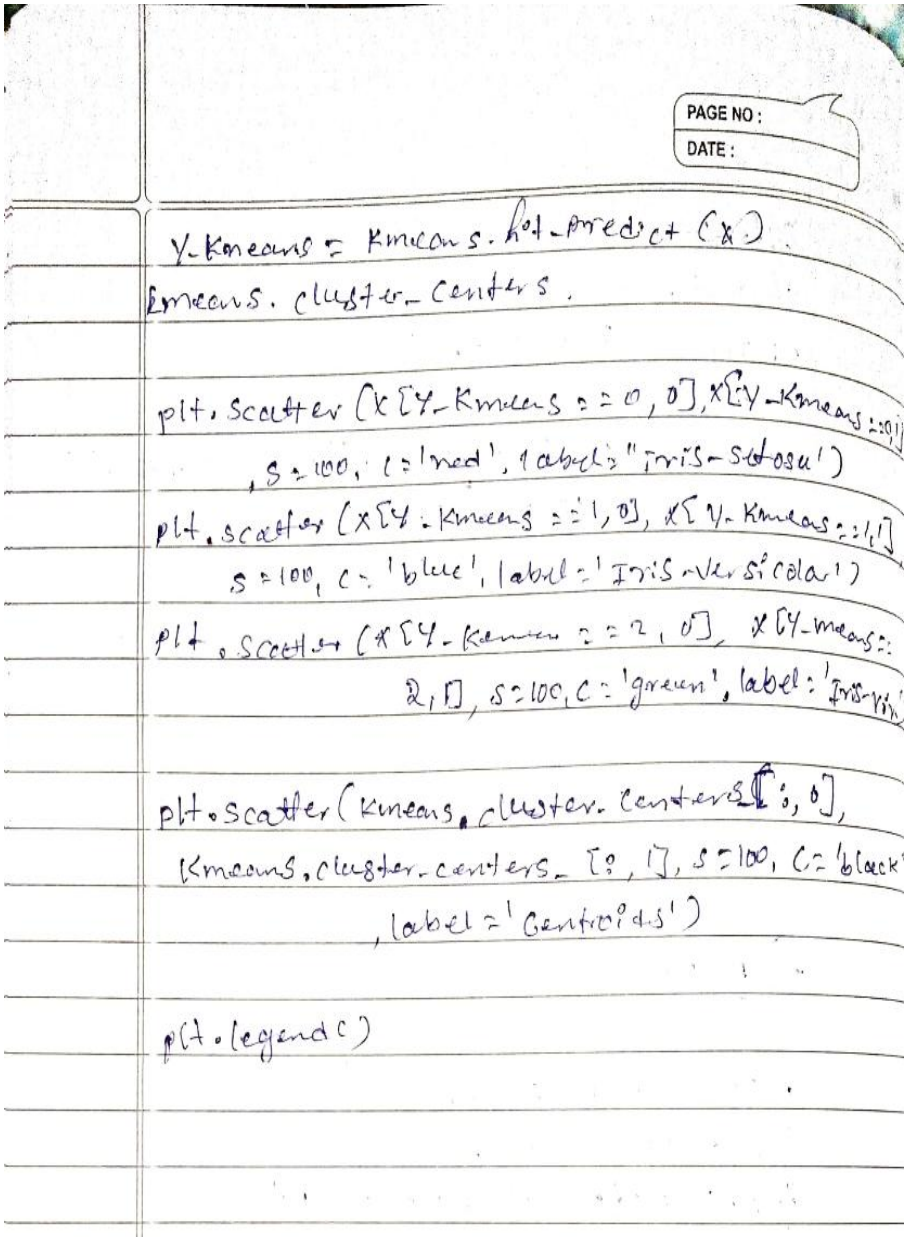
Program 10

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

Screenshot:



```
3) K-Mean

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
import seaborn as sns
from sklearn.cluster import KMeans.

iris = datasets.load_iris()
print("Dataset loaded successfully.");

Data = pd.DataFrame(iris.data, colums = iris.
                    feature_names)

X = Data.iloc[:, 0:3].values.
css = []
kmean = KMeans(n_clusters=3, init = 'k-me
    ++', max_iter=100, n_init = 10, random_state=0)
```



Code:

```
import pandas as pd

from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris # Import load_iris
```

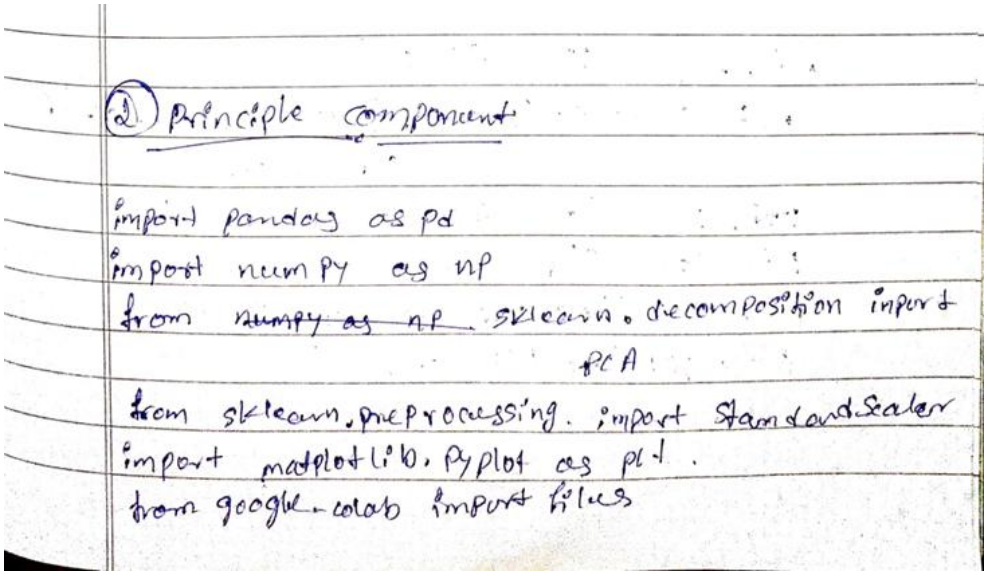
```
# Step 1: Load the Iris dataset directly

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

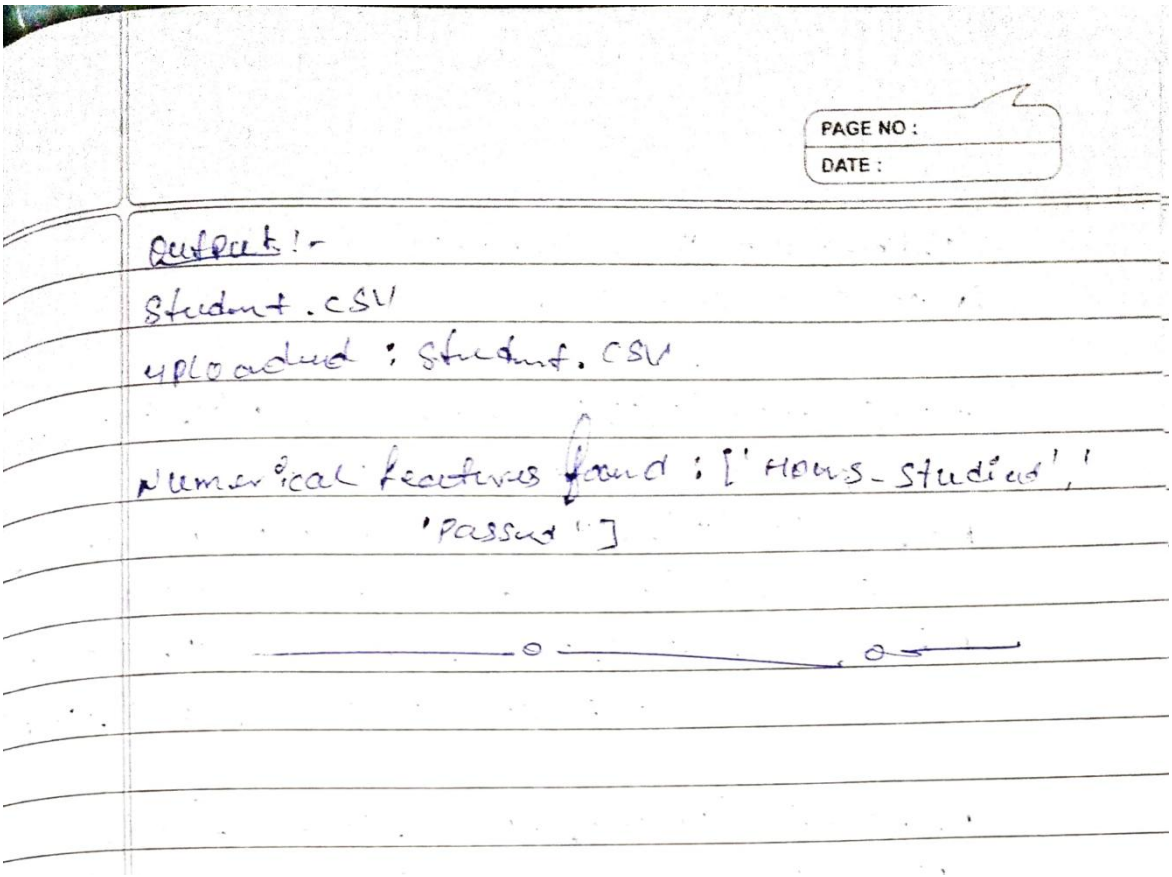
Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot:



PAGE NO :
DATE :

```
uploaded = files.upload()

for filename in uploaded.keys():
  df = pd.read_csv(filename)
  print(f"uploaded: {filename}")
  display(df.head())

numeric_df = df.select_dtypes(include=
                [np.number])
print("Numerical features found:", list(numeric_df.
                columns))

selected_features = numeric_df.columns

X = numeric_df[selected_features].dropna()
X_scaled = StandardScaler().fit_transform(X)

pca = PCA(n_components = 2)
principal_components = pca.fit_transform(X_scale

pca_df = pd.DataFrame(data = principal_
    components, columns =['PC1', 'PC2'])

plt.figure(figsize=(8,6))
plt.scatter(pca_df['PC1'], pca_df['PC2'],
                alpha=0.7)
plt.xlabel('Principal Component-1')
plt.ylabel('Principal Component 2')
plt.title('2D PCA visualization')
plt.grid(True)
plt.show()
```



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()
```

Choose Files Student.csv

- **Student.csv**(text/csv) - 41 bytes, last modified: 5/18/2025 - 100% done

```
Saving Student.csv to Student.csv
Uploaded file: Student.csv
```

	Hours_Studied	Passes
0	1	0
1	2	0
2	3	1
3	4	1
4	5	1

```
Numerical features found: ['Hours_Studied', 'Passes']
```