VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum- 590014, Karnataka.

LAB REPORT

on

Machine Learning (23CS6PCMAL)

Submitted by

Prajwal K K (1BM22CS199)

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 **Prajwal K K (1BM22CS199),** 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

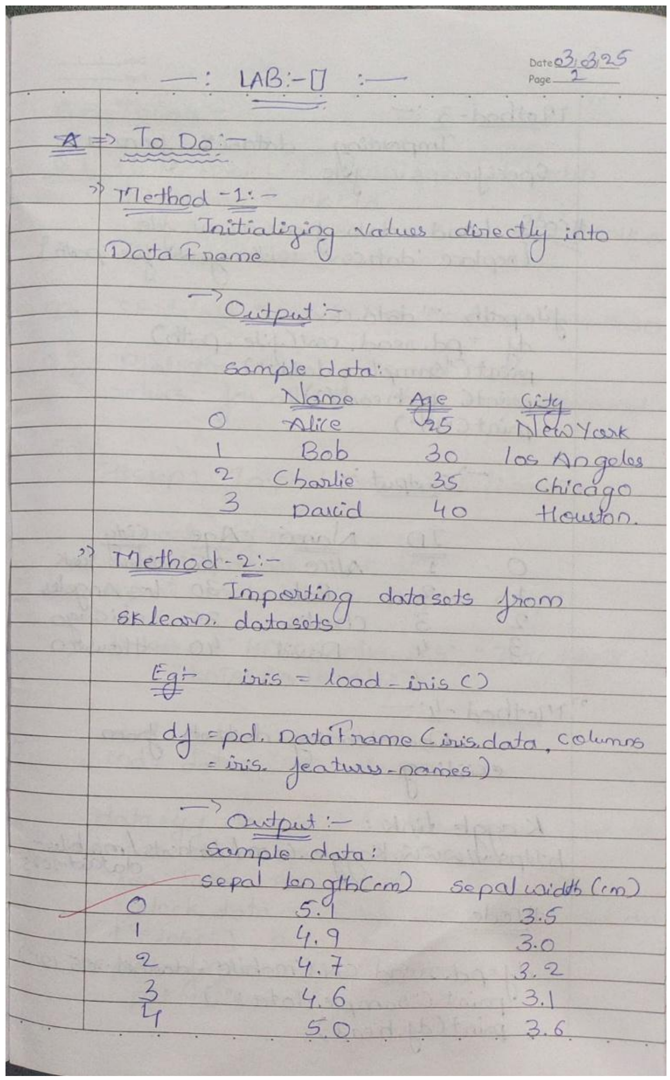
INDEX

Github Link: PRAJWALKK007/ML-LAB

Program 1

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

Screenshot:



Date 3.3.25
Page 2

—: LAB:-1 :—

A => To Do:-

» Method-1:-

Initializing values directly into Data Frame

→ Output:-

sample data:

	Name	Age	City
0	Alice	25	New York
1	Bob	30	Los Angeles
2	Charlie	35	Chicago
3	David	40	Houston.

» Method-2:-

Importing datasets from sklearn.datasets

Eg:- iris = load_iris()

df = pd.DataFrame(iris.data, columns = iris.feature_names)

→ Output:-

Sample data:

	sepal length(cm)	sepal width (cm)
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6

Date __/__/__
Page ______

>> Method-3 :—
Importing datasets from a Specific. csv file

#code Load data from a csv file
{replace 'data.csv' with your file path}.

```
file path ='data.csv'
df =pd.read_csv(file_path)
print("sample data")
print( df.head())
print("\n")
```

→ Output :-

	ID	Name	Age	City
0	1	Alice	25	New York
1	2	Bob	30	los Angeles
2	3	Charlie	35	chicago
3	4	David	40	Houston

>> Method-4:—
Downloading datasets from existing

Kaggle link :
https://www.kaggle.com/datasets/mobiles-dataset-2025

#code

```
df=pd.read_csv('mobile-dataset-2025.csv')
print("sample data:")
print(df.head)
```

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
```

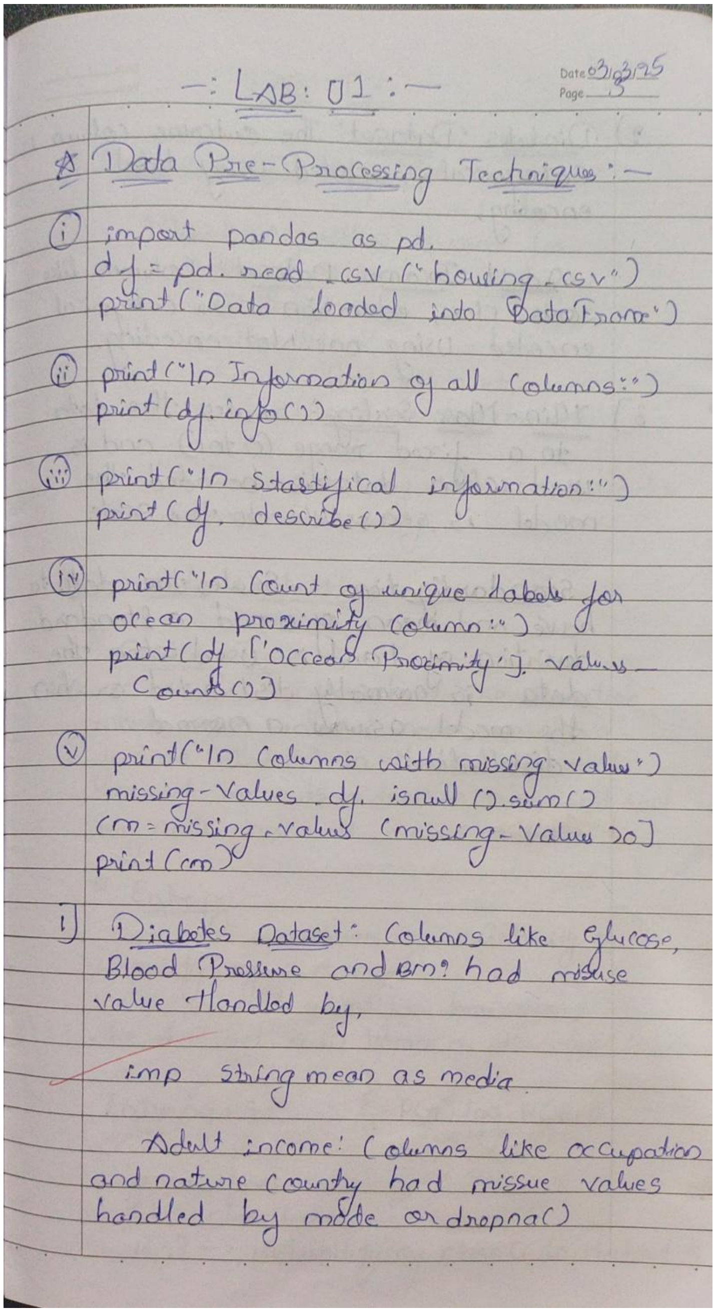
```
from sklearn.datasets import load_diabetes

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: 01 :-

Date 03/03/25
Page 3

Data Pre-Processing Techniques :-

(i)
```
import pandas as pd.
df = pd. read .csv ("housing.csv")
print("Data loaded into DataFrame")
```

(ii)
```
print("\n Information of all Columns:")
print(df.info())
```

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

(iv)
```
print("\n Count of unique labels for
ocean proximity Column:")
print(df ['Ocean Proximity']. values_
  Counts()]
```

(v)
```
print("\n Columns with missing values")
missing-Values . df. isnull ().sum()
cm = missing.values (missing-Values >0]
print(cm)
```

i] Diabetes Dataset: Columns like Glucose, Blood Pressure and BMI had missue value Handled by,

imp string mean as media.

Adult income: Columns like occupation and nature country had missue values handled by mode or dropna()

Date ___/___/___
Page ______

2) **Diabetes Dataset**: The outcome column is categorical, encoded using label encoding:

Adult Income Dataset: Columns like work class, education were categorical encoded using one-Hot encoding.

3) **Min-Max Scaling**: scales the data to a fixed range (0 to 1) and is used when data is bounded the model is sensstive to the scale.

Standardization: Scales the data to have and 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 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()
```

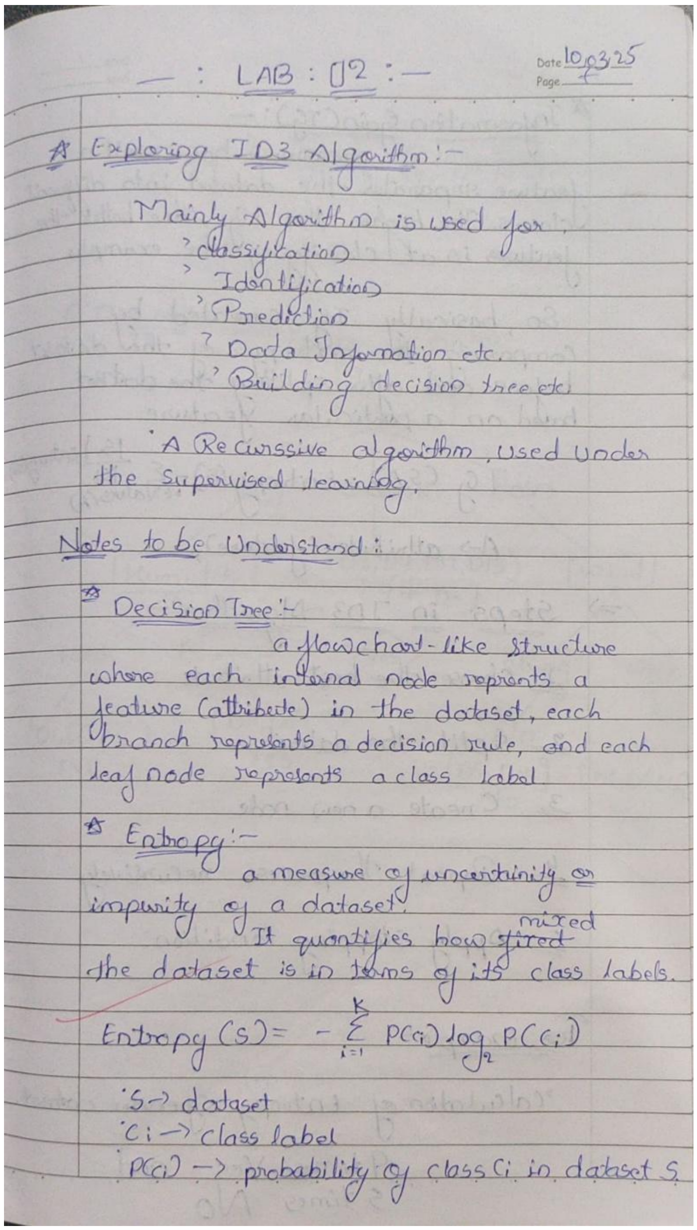
```
df[['Age', 'Salary']] = scaler.fit_transform(df[['Age', 'Salary']])

# 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()
```

```python
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 : —

Date 10.03.25
Page 4

★ Exploring ID3 Algorithm :-

Mainly Algorithm is used for :

- Classification
- Identification
- Prediction
- Data Information etc.
- Building decision tree etc.

· A Recurssive algorithm, used under the supervised learning.

Notes to be Understand :

★ Decision Tree :- a flowchart-like structure where each internal node reprents a feature (attribute) in the dataset, each branch represents a decision rule, and each leaf node represents a class label

★ Entropy :- a measure of uncertainity or impurity of a dataset. It quantifies how ~~stired~~ mixed the dataset is in terms of its class labels.

$$\text{Entropy}(S) = -\sum_{i=1}^{K} P(c_i)\log_2 P(c_i)$$

· S → dataset
· c_i → class label
· $P(c_i)$ → probability of class c_i in dataset S

Date __/__/__
Page ____

Information Gain (IG): —

measures how well a feature separates the dataset into different classes. The higher the IG, the better the features is at classifying the examples.

So, basically IG created by comparing the entropy of the dataset before and after splitting the dataset based on a particular feature.

$$IG(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

A → attribute (feature),

⇒ **Steps in ID3 Algorithm:** —

1. Choose the best attribute
2. Split the dataset
3. Create a new node
4. Repeat the process recursively
5. Apply stopping condition.

For Example:—

Calculation of Entropy of entire dataset.

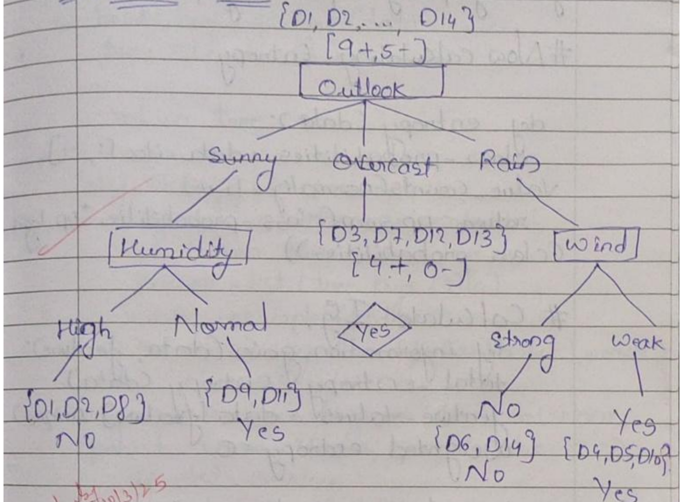
9 times Yes and
5 times No

Date ___ Page 9

$$(S) = -\left(\frac{9}{14}\log_2\frac{9}{14} + \frac{5}{14}\log_2\frac{5}{14}\right)$$

$$S \approx 0.940$$

Decision Tree:-



PAGE NO: 8
DATE:

17|03|24 ID3 code:

```
import pandas as pd
import numpy as np

def entropy (data):
    class_prob = data.iloc [:, -1]. value_
        counts (normalize = True)
    return -np.sum (class_prob * np.log2
    (class_prob))

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

def best_feature (data):
    features = data.columns [:-1]
    gains = {feature: information_gain(
    data, feature) for feature in features}
    return max(gains, key = gains.get)

def id3 (data, features = None):
    if len (data.iloc [:, -1].unique ()) == 1
        return data.iloc [:, -1].iloc[0]

    if len (features) == 0:
```

—: LAB: 03 :—

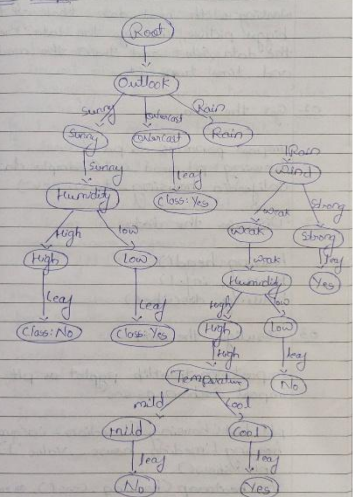
Implementing the ID3 Algorithm :—

#Code

```
import pandas as pd
import numpy as np
from graphviz import Digraph

#Now calculating Entropy

def entropy (data):
    class_probabilities = data.iloc[:, -1].
value_counts(normaliz=True)
    return -np.sum(class_probabilities * np.log2
(class_probabilities))

# Calculate IG.
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

def best_feature (data):
    features = data.columns [:-1]
    gains = {feature : information_gain(data,
        feature) for feature in features}
    return max(gains, key=gains.get)
```

Page 13

⇒ 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:

—: LAB-04 :—

Date __/__/__
Page ______

A 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 the independent variables and dependent variables.

Involving only one independent variable.

To find a line $y = mx + c$ i.e. minimizes the error b/w the predicted values (y) and actual values.

Algo

01 Define the Problem:

Select the dependent variable y (target variable) and the independent variables x, (features)

02 Initialize Parameters:

Set initial values for the co-efficients (β_0....), also the learning rate.

MSE → Average of squared difference b/w actual and predicted values.

Date __/__/__
Page 17

03 Compute Prediction:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon$$

04 Calculate MSE: {Mean Squared Error}

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

$N \rightarrow$ observations
$\hat{y}_i \rightarrow$ predicted values
$y_i \rightarrow$ actual values

Applications:

- Analyzing risk in financial systems
- Forecasting sales or revenue
- Estimating trends in data
- Predicting Student Satisfaction

Pseudocode for Linear Regression:

```
Function LinearRegression(x, y):
    #step 1: Add a column of ones to x for
    the intercept term
    x = AddColumnOfOnes(x)

    #step 2: Compute the coefficients using
    the OLS formula
    # beta = (X^T * X)^-1 * X^T * Y
    X_transpose = Transpose(X)
    XTX = Multiply(X_transpose, X)
    XTX_inverse = Inverse(XTX)
```

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

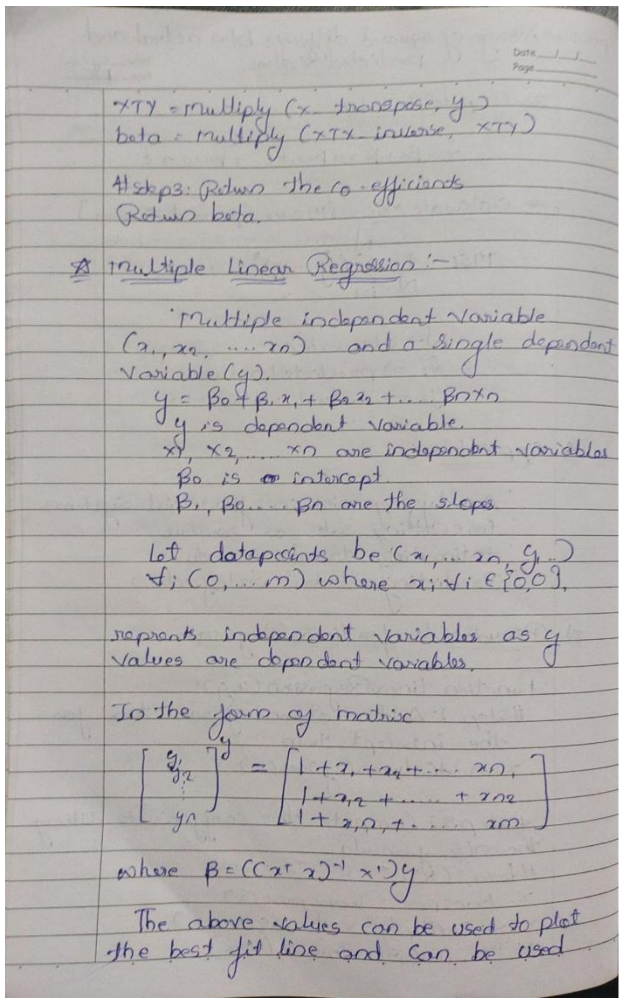
```
import pandas as pd

# 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:



XTY = multiply (x_transpose, y)
beta = multiply (XTX_inverse, XTY)

step3: Return the co-efficients
Return beta.

Multiple Linear Regression :-

Multiple independent variable $(x_1, x_2, \ldots x_n)$ and a single dependent variable (y).

$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots \beta_n x_n$

y is dependent variable.

$x_1, x_2, \ldots x_n$ are independent variables

β_0 is intercept.

$\beta_1, \beta_2 \ldots \beta_n$ are the slopes

Let datapoints be $(x_1, \ldots x_n, y_i \ldots)$ $\forall_i (0, \ldots m)$ where $x_i \forall i \in \{0, 0\}$.

represents independent variables as y values are dependent variables.

In the form of matrix

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 + x_1 + x_2 + \ldots x_{n1} \\ 1 + x_{12} + \ldots + x_{n2} \\ 1 + x_{1n} + \ldots x_m \end{bmatrix}$$

where $\beta = ((x^T x)^{-1} x^T) y$

The above values can be used to plot the best fit line and can be used

Date __/__/__
Page 19

to predict future values.

☆ Logistic Regression :-

- Logistic regression approach operates on Sigmoid curve rather than best fit line, we got a value $\in [0,1]$ (Binary Classification) and then classify into +ve or -ve by comparing with median.

Let data points be $(x_i, y_i) \; \forall \; i \in [0, n]$, 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 $V < 0.5 \rightarrow$ then "no"
- If $V > 0.5 \rightarrow$ then "Yes".

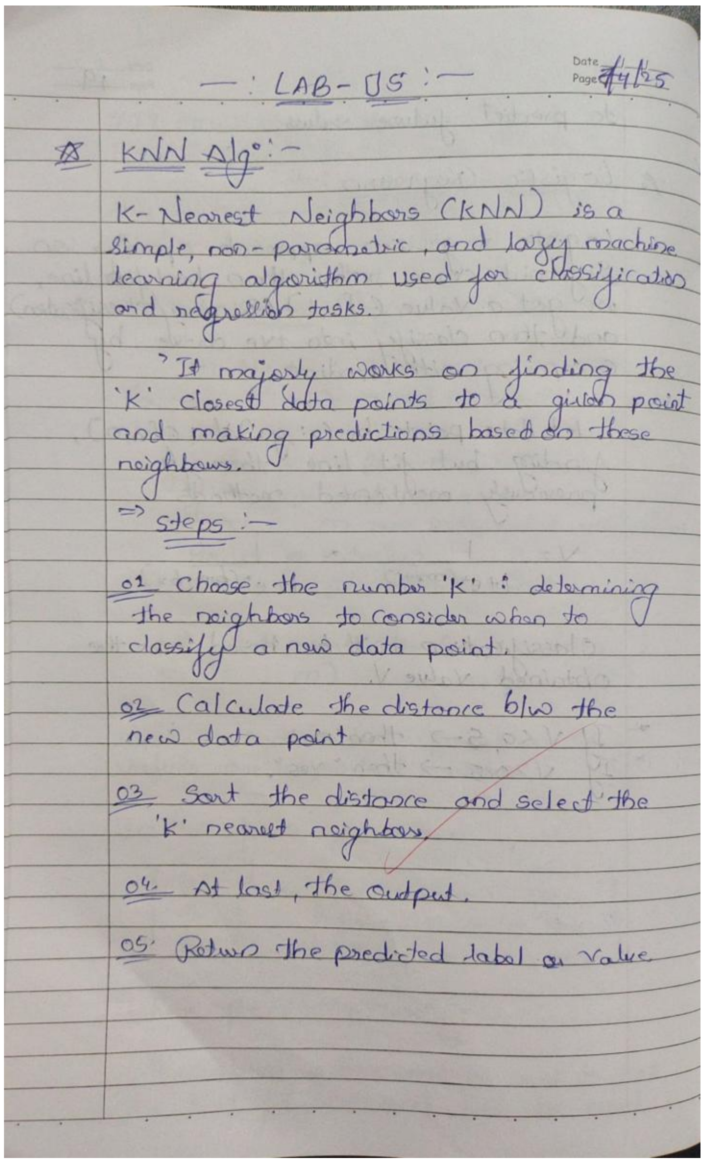
Code:

```
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:



— : LAB - 05 : —

Date 7/4/25

KNN Alg° :—

K- Nearest Neighbors (KNN) is a simple, non-parametric, and lazy machine learning algorithm used for classification and regression tasks.

It majorly works on finding the 'K' closest data points to a given point and making predictions based on these neighbours.

=> Steps :—

01 Choose the number 'K' : determining the neighbors to consider when to classify a new data point.

02 Calculate the distance b/w the new data point

03 Sort the distance and select the 'K' nearest neighbors.

04 At last, the output.

05 Return the predicted label or value

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

Date __/__/__
Page 21

Code

Using sklearn

```
iris = load_iris()
x = iris.data
y = iris.target

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

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

knn = KNeighborsClassifier(n_neighbors = 3)

knn.fit(x_train_scaled, y_train)

y_pred = knn.predict(x_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
```

=> Output:-

```
Accuracy of KNN classifier: 1.00
Predictions: [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0
              1 2 1 1 2 0 2 0 2 0 2 2 2 2 2 0]
True labels: [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0
              1 2 1 1 2 0 2 0 2 2 2 2 0 0]
```

Tuning 'K':

- If K is too small, the model may be noisy and overfit the data (High Variance)

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:

Date __/__/__
Page ____

- If K is too large, the model may be too simple and underfit the data.
- A common practice is to use cross-validation to find the optimal value for K

△ SVM {Support Vector Machine}:-

a powerful, supervised machine learning algorithm commonly used for classification and regression tasks.

→ Algorithm:-

01 Input: A dataset with labeled examples.

02 Output: A hyperplane that best separates the classes in the feature space.

03 Training:

- Find the optimal hyperplane by maximizing the margin b/w classes.
- For non-linearly separable data, apply kernal functions to transform the data into higher dimensions, where a hyperplane can be found.

⇒ Classification:

1 -> Yes
0 -> No

Date __/__/__
Page ______

- **Input:** A set of labeled data points.
- **Output:** A class label based on the optimal hyperplane.
- **Goal:** Maximize the margin between classes while minimizing misclassification

Code:-

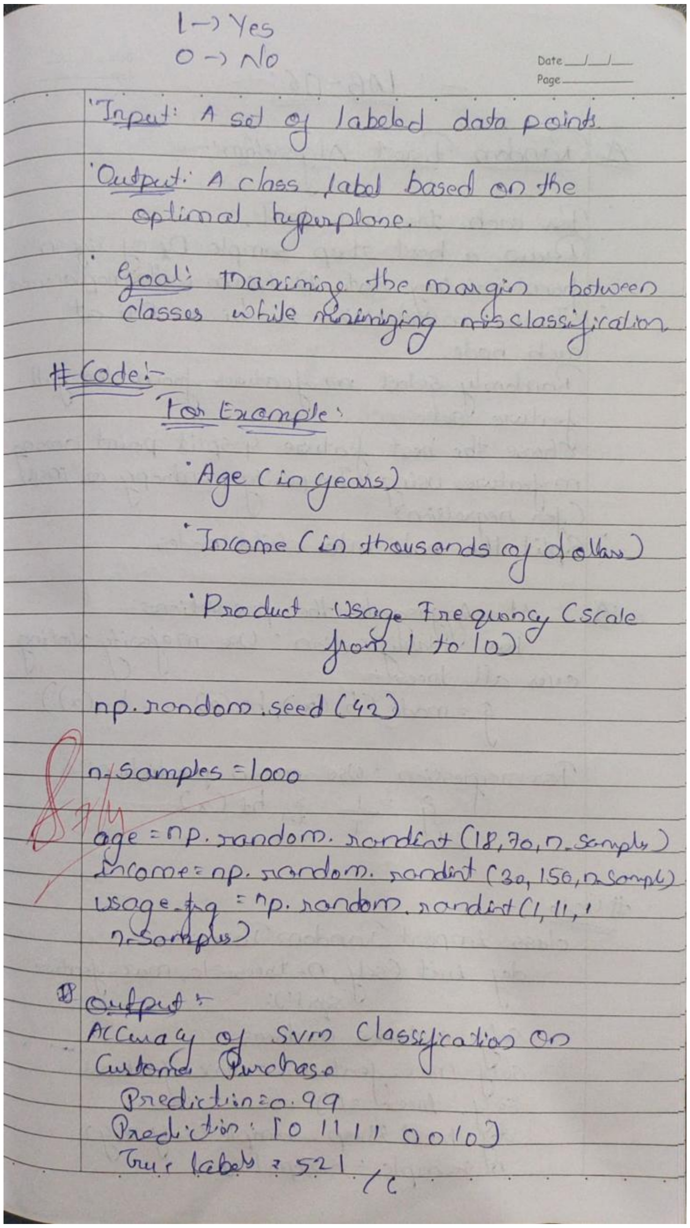
For Example:

- Age (in years)
- Income (in thousands of dollars)
- Product Usage Frequency (scale from 1 to 10)

```
np.random.seed(42)

n_samples = 1000

age = np.random.randint(18, 70, n_samples)
income = np.random.randint(30, 150, n_samples)
usage_freq = np.random.randint(1, 11,
n_samples)
```

Output:-

Accuracy of SVM Classification on Customer Purchase
Prediction: 0.99
Prediction: [0 1 1 1 1 0 0 1 0]
True labels: 521 [C


1 → Yes
0 → No

Date __/__/__
Page ____

- Input: A set of labeled data points
- Output: A class label based on the optimal hyperplane.
- Goal: Maximize the margin between classes while minimizing misclassification

Code:-

For Example:

- Age (in years)
- Income (in thousands of dollars)
- Product Usage Frequency (scale from 1 to 10)

np.random.seed(42)

n_samples = 1000

age = np.random.randint(18, 70, n_samples)
income = np.random.randint(30, 150, n_samples)
usage_fq = np.random.randint(1, 11, n_samples)

Output:-

Accuracy of SVM Classification on Customer Purchase Prediction: 0.99
Prediction: [0 1 1 1 1 0 0 1 0]
True labels: 521

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:

Date __/__/__
Page ____

—: LAB – 06 :—

A. Random Forest Algorithm:—

- For each tree $t = 1$ to T,
- Draw a boot strap sample D_t of size n from raining data D (sam with replacement)
- Train a decision tree h_i on D_i at each node.
 Randomly select m features from the full feature set.
 Choose the best feature & split point among m features using Impurity, Entropy or MSE (for regression)
- Split the node into child nodes.

B. At Aggregate the predictions:
For classification: Use majority voting over all trees:

$$\hat{y} = \text{mode}(h_1(x), h_2(x), \ldots h_n(x))$$

For regression: Use mean prediction:

$$\hat{y} = \frac{1}{T} \sum_{t=1} h_t(x)$$

Code

```
class import RandomForest:
    def init (self, n-trees =10, max-feature
                        sqrt'):
    self.n-trees n-trees
    self.max-features = max-features
    self.trees =[]
    def boost strap_sample (self. x,y):
    n-sample = x.slope[0]
```

Date __/__/__
Page ______

```
    size = n-sample . replace = True
    return x [indexs], y [indexs]
def - get - max - features (self, n - features):
    if self . max - features == sqrt:
        return int (np. sqrt (n - features))
    elif is instances (self. max - features . int):
        return self. max - feature
def fit (self, x, y):
    self . tree = []
    n - features = x. shape[:]
    max - feature = self . get - max - features
```

=> K-Means Algorithm:-

01. Initialize centroids:
Randomly choose k data pointer from x as initial cluster centroids:

$\mu_1, \mu_2, \ldots \mu_k$.

02. Repeat until convergence.
(a) Repeat each data point to the nearest centroid. For each point x_i, find the closet centroid μ_j based on distance

$$\mu_j = \frac{1}{|C_i|} \sum_{x_i \in C_i} x_i$$

03. check for convergence
- if cluster assignments don't chang
- Centroids don't move significantly, the stop

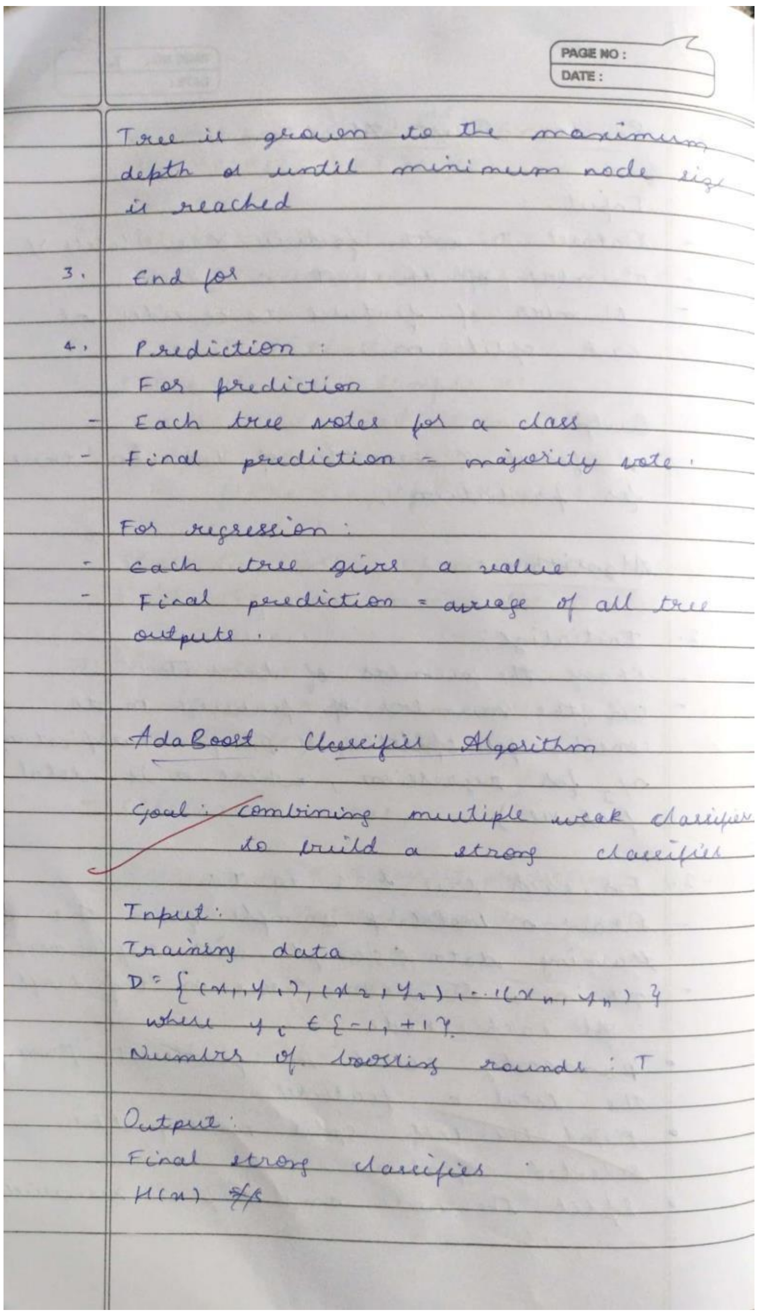
Code:

```
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 $y_i \in \{-1, +1\}$
Number of boosting rounds : T

Output :
Final strong classifier
$H(x)$ ~~s/s~~

Code:

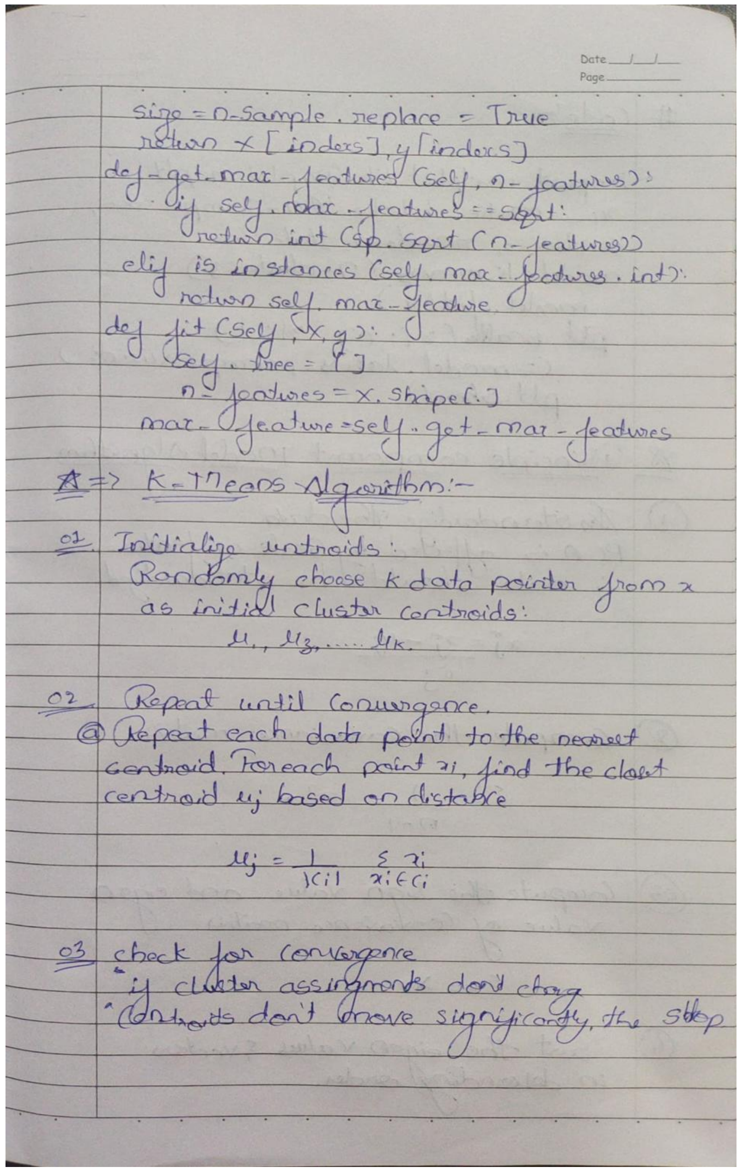
```
from sklearn.ensemble import AdaBoostClassifier

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:



Date __/__/__
Page ____

```
size = n-Sample . replace = True
return x [indexs], y [indexs]
def -get-max-features (self, n-features):
    if self.max-features == sqrt:
        return int (np.sqrt (n-features))
    elif is instances (self.max-features. int):
        return self.max-feature
def fit (self, x, y):
    self. tree = [ ]
    n-features = x. shape[:]
    max-feature = self.get-max-features
```

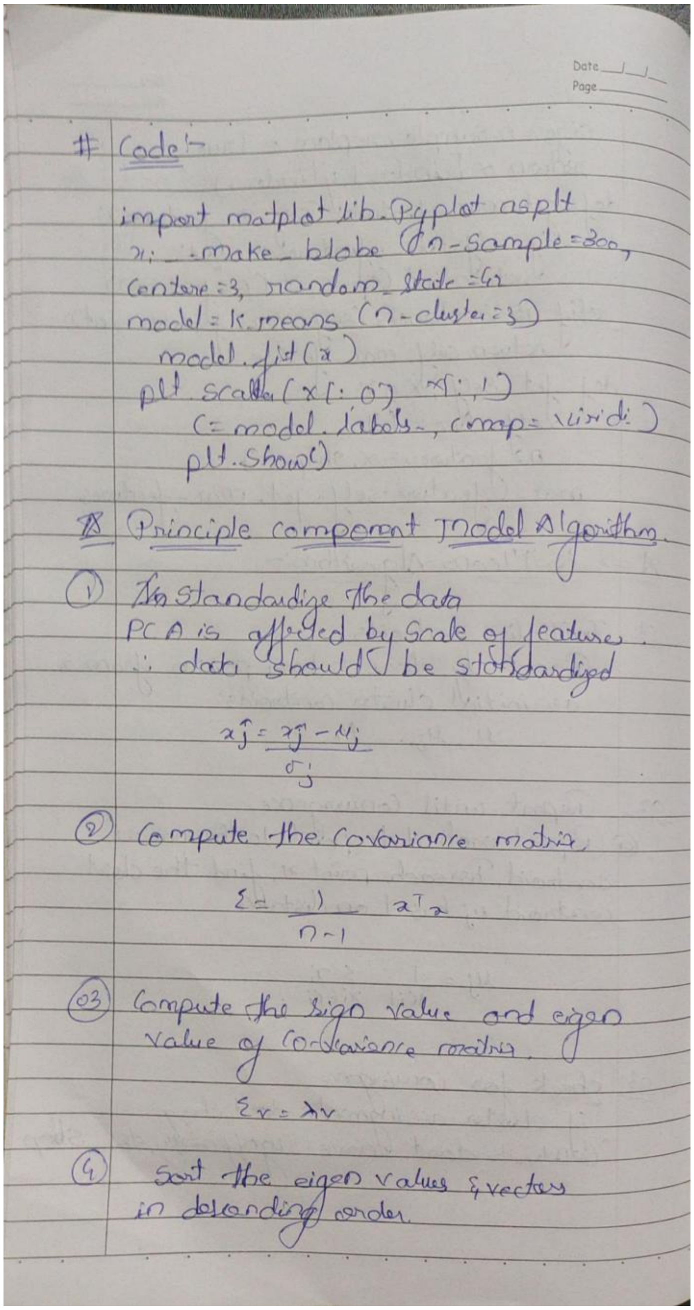
A => K-Means Algorithm:-

01. Initialize centroids:
Randomly choose k data pointer from x
as initial cluster centroids:
$\mu_1, \mu_2, \ldots \mu_k$.

02. Repeat until convergence.
(a) Repeat each data point to the nearest centroid. For each point x_i, find the closest centroid μ_j based on distance

$$\mu_j = \frac{1}{|C_i|} \sum_{x_i \in C_i} x_i$$

03 check for convergence
· if cluster assignments don't change
· Centroids don't move significantly, the stop



Date ___/___/___
Page ______

Code :-

```
import matplotlib.pyplot as plt
x, _ = make_blobs (n-sample=300,
centers=3, random_state=42
model = K means (n-clusters=3)
  model.fit(x)
  plt.scatter (x[:,0], x[:,1]
      c=model.labels_, cmap='viridis')
    plt.show()
```

Principle component Model Algorithm.

① To standardize the data
PCA is affected by scale of features.
∴ data should be standardized

$$x_j^{\hat{}} = \frac{x_j - \mu_j}{\sigma_j}$$

② Compute the covariance matrix.

$$\Sigma = \frac{1}{n-1} x^T x$$

③ Compute the sign value and eigen value of co-variance matrix.

$$\Sigma v = \lambda v$$

④ Sort the eigen values & vectors in descending order.

Code:

import pandas as pd

from sklearn.cluster import KMeans

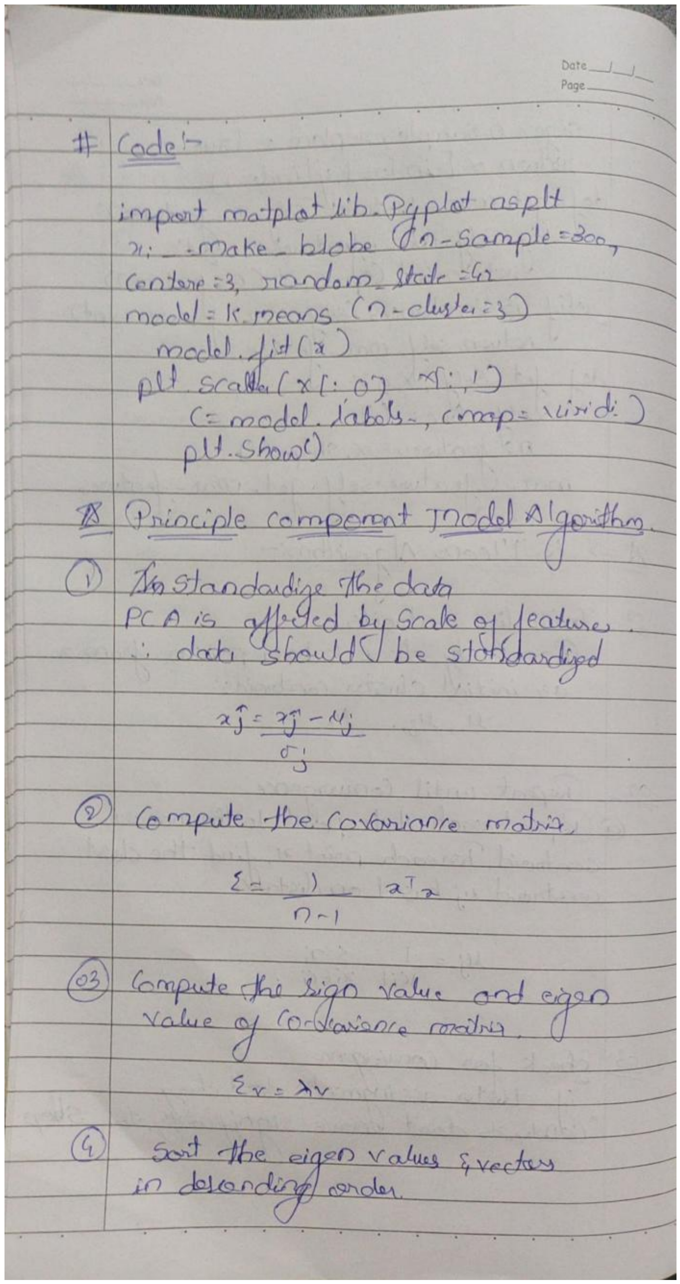
```
import matplotlib.pyplot as plt

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:


Date / /
Page

Code:-

```
import matplotlib.Pyplot as plt
x: _ make_blobe (n-sample=300,
centere=3, random_state=42
model=k.means (n-cluster=3)
 model.fit(x)
plt.scatter(x[:0] x[:,1]
   c=model.labels_, cmap=viridi)
 plt.show()
```

Principle component Model Algorithm.

(1) Standardize the data
PCA is affected by Scale of feature.
∴ data should be standardized

$$\hat{x_j} = \frac{x_j - \mu_j}{\sigma_j}$$

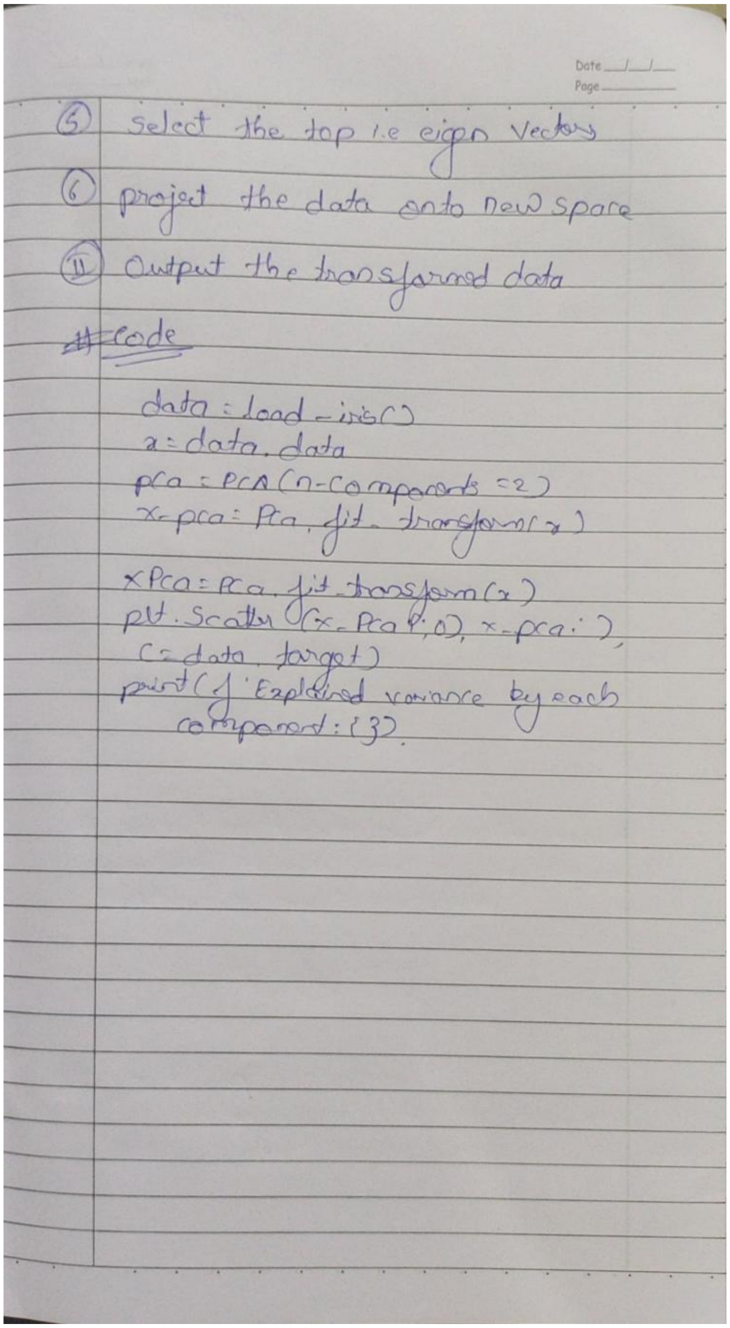
(2) Compute the covariance matrix

$$\Sigma = \frac{1}{n-1} x^T x$$

(03) Compute the sign value and eigen value of covariance matrix.

$$\Sigma v = \lambda v$$

(4) Sort the eigen values & vectors in descending order.



⑤ select the top i.e eigen vectors

⑥ project the data onto new space

⑦ Output the transformed data

code

```
data = load_iris()
x = data.data
pca = PCA(n_components = 2)
x_pca = pca.fit_transform(x)

xPca = Pca.fit_transform(x)
plt.Scatter(x_Pca[:,0], x_pca[: ],
c=data.target)
print(f'Explained variance by each
    component: {}')
```

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
```