VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Pannaga R Bhat (1BM22CS189)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Sep-2024 to Jan-2025

B.M.S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Machine Learning (23CS6PCMAL)" carried out by **Pannaga R Bhat (1BM22CS189),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

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

Index

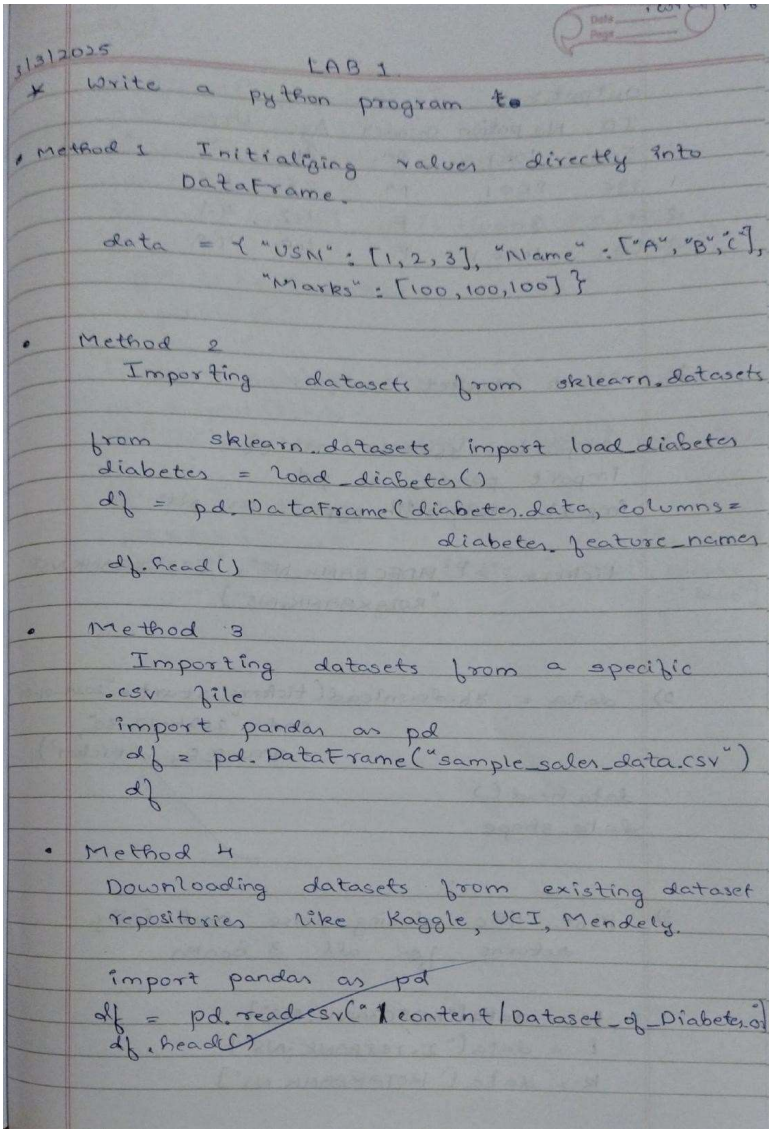
Github Link:

https://github.com/pannaga-rj/ML_Lab_1BM22CS189.git

Program 1

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

Screenshot:



31/12/2025

LAB 1

* Write a python program to

• Method 1 Initializing values directly into DataFrame.

```
data = {"USN": [1,2,3], "Name": ["A","B","C"],
        "Marks": [100,100,100]}
```

• Method 2
Importing datasets from sklearn.datasets

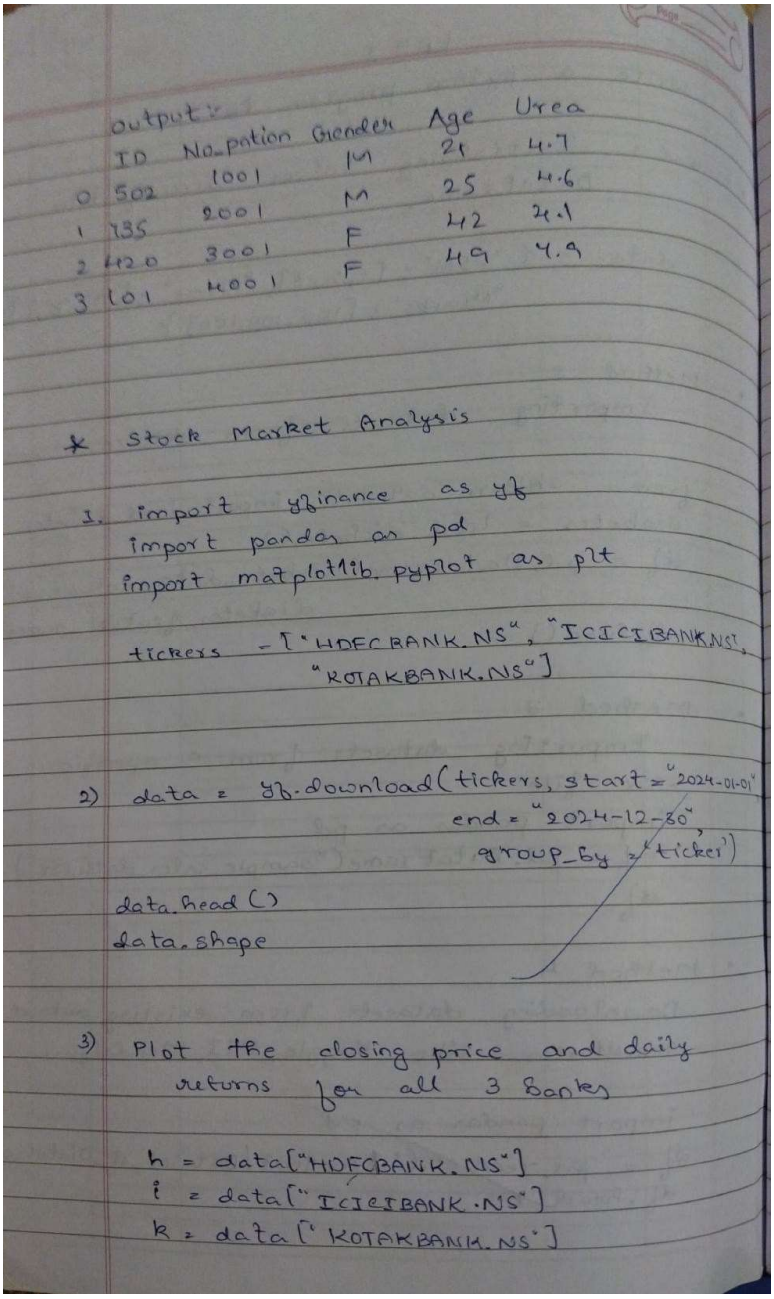
```
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
df = pd.DataFrame(diabetes.data, columns=
                  diabetes.feature_names
df.head()
```

• Method 3
Importing datasets from a specific .csv file

```
import pandas as pd
df = pd.DataFrame("sample_sales_data.csv")
df
```

• Method 4
Downloading datasets from existing dataset repositories like Kaggle, UCI, Mendely.

```
import pandas as pd
df = pd.read_csv("/content/Dataset_of_Diabetes.c
df.head()
```



output:

	ID	No-pation	Gender	Age	Urea
0	502	1001	M	21	4.7
1	735	2001	M	25	4.6
2	420	3001	F	42	4.1
3	101	4001	F	49	4.9

* Stock Market Analysis

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

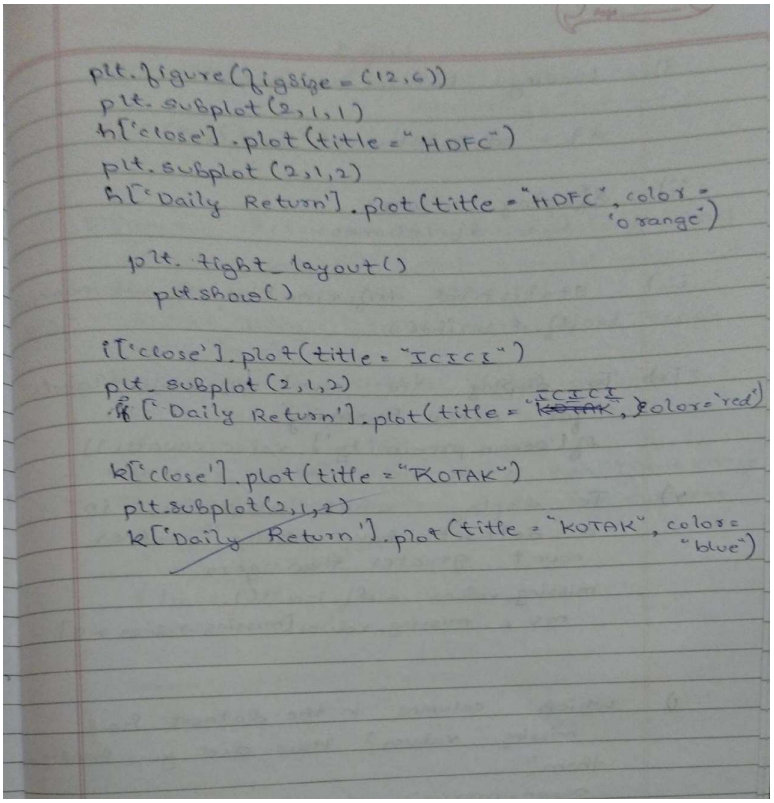
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]

2) data = yf.download(tickers, start="2024-01-01", end="2024-12-30", group_by='ticker')

data.head()
data.shape

3) Plot the closing price and daily returns for all 3 banks

h = data["HDFCBANK.NS"]
i = data["ICICIBANK.NS"]
k = data['KOTAKBANK.NS']



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)
```

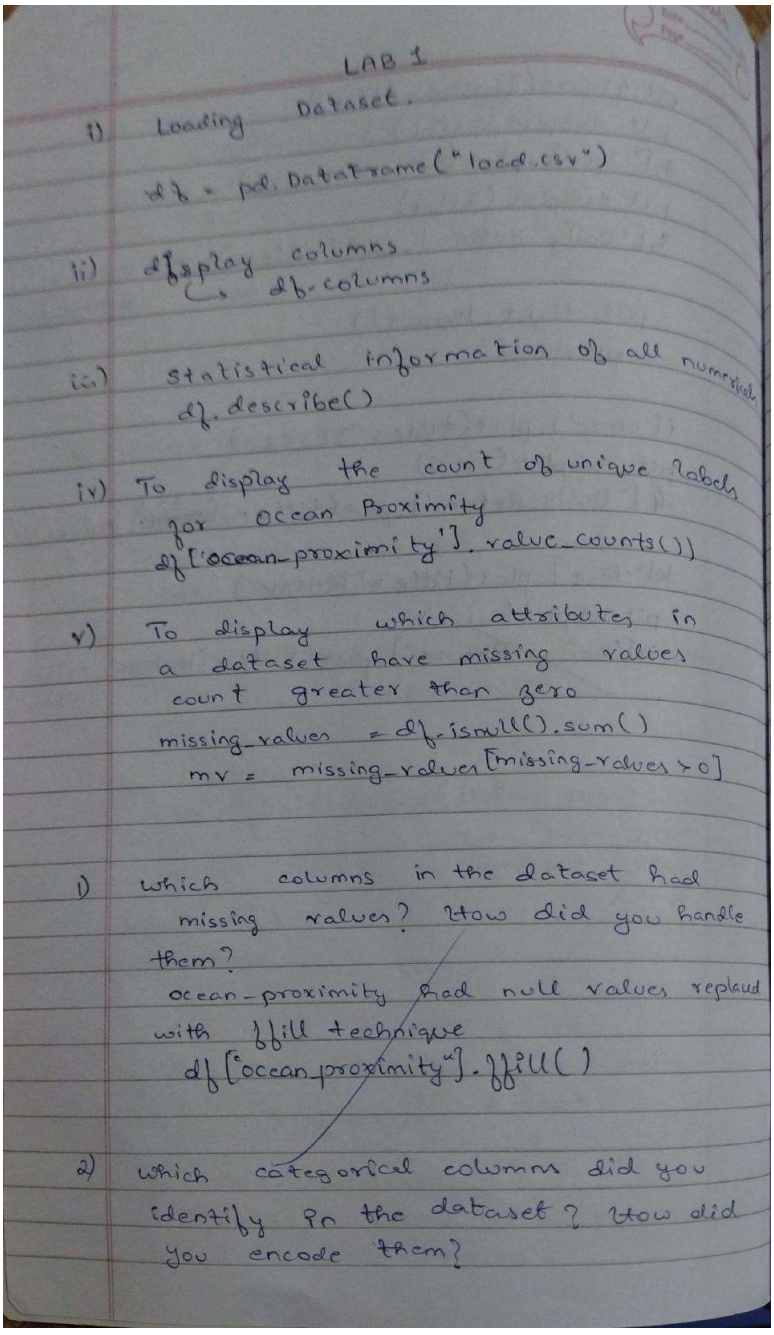
```
# Summary statistics for a specific stock (e.g., Reliance)

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

i) Loading Dataset.

df = pd.DataFrame("load.csv")

ii) display columns
df.columns

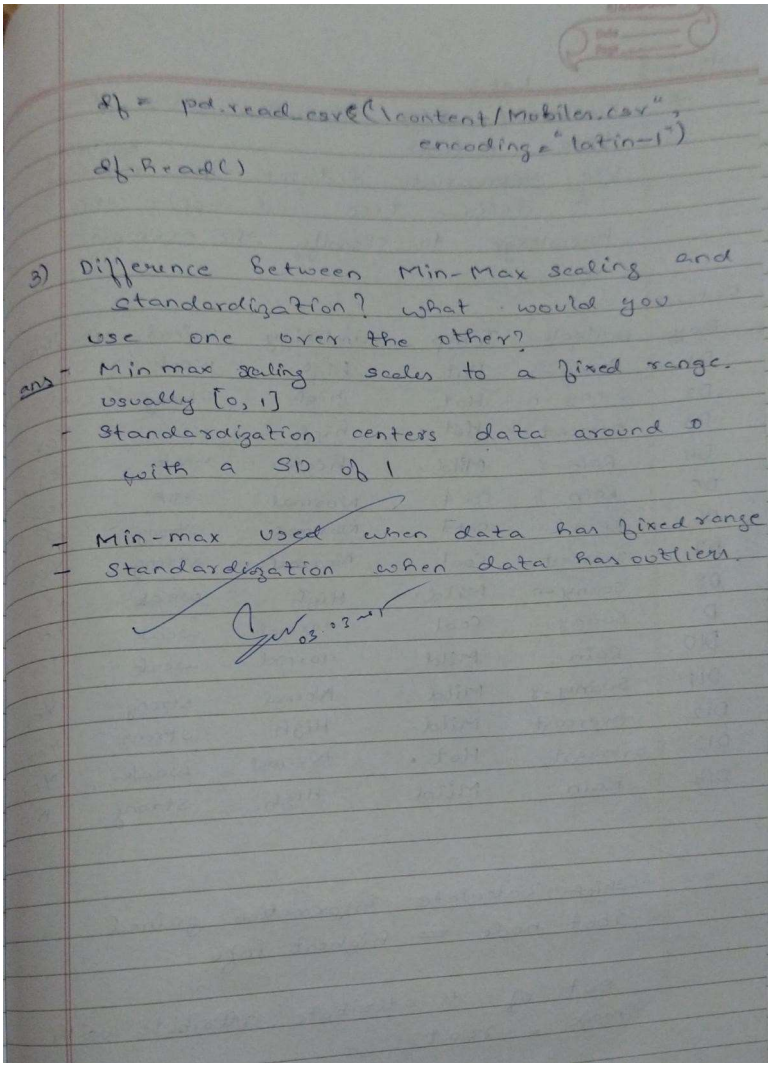
iii) Statistical information of all numerical
df.describe()

iv) To display the count of unique labels for Ocean Proximity
df['ocean_proximity'].value_counts()

v) To display which attributes in a dataset have missing values count greater than zero
missing_values = df.isnull().sum()
mv = missing_values[missing_values>0]

1) Which columns in the dataset had missing values? How did you handle them?
ocean_proximity had null values replaced with ffill technique
df["ocean_proximity"].ffill()

2) Which categorical columns did you identify in the dataset? How did you encode them?



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()
```

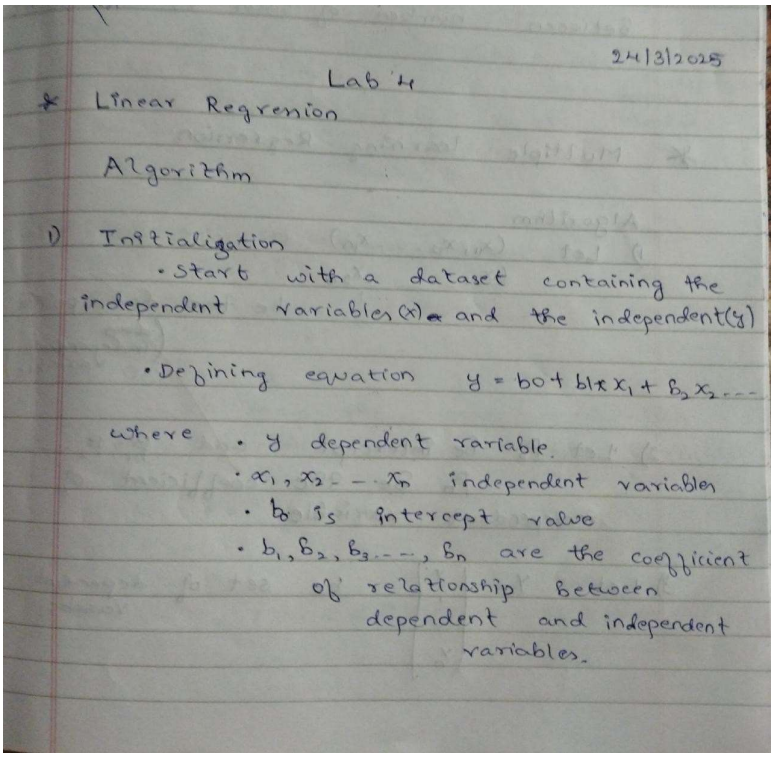
```
df[['Age', 'Salary']] = scaler.fit_transform(df[['Age', 'Salary']])


# 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

Implement Linear and Multi-Linear Regression algorithm appropriate dataset

Screenshot:



24/3/2025

Lab 4

* Linear Regression

Algorithm

1) Initialization
   - Start with a dataset containing the independent variables (x) and the independent(y)
   - Defining equation $y = b_0 + b_1 x_1 + b_2 x_2 \ldots$

where
- y dependent variable.
- $x_1, x_2 \ldots x_n$ independent variables
- b_0 is intercept value
- $b_1, b_2, b_3 \ldots, b_n$ are the coefficient of relationship between dependent and independent variables.

2) Finding the Best Line which minimizes the sum of squared errors between predicted and actual values

$$\Sigma(y_i - \bar{y}_i)^2$$

3) Predict the value.

Ex TV_Sales Dataset where we can find relationship between number of sales vs cost spent

* Multiple learning Regression

Algorithm

1) Let $(x_{11}, x_{12} \dots x_{1n})$
$(x_{21} \dots x_{2n})$
⋮ ⋮ be the data points (independent variable)

2) Let β_0 be intercept and $\beta_1, \beta_2 \dots \beta_n$ be the coefficient of independent variable

3) Let $Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}$ be set of dependent Variables

3] After rearranging

$$Y = \begin{bmatrix} 1 + x_1 + x_2 - & + x_{1n} \\ 1 + x_{21} + x_2 - & x_{2n} \\ \vdots & \\ 1 + 1 & x_{mn} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_n \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

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

4] Obtain values of $\beta_0, \beta_1 \ldots \beta_n$

Ex:- Housing dataset where we have CRIM, ZN, INDUS, PTRATIO are ~~depen~~ independent upon which we calculate 'Medv'

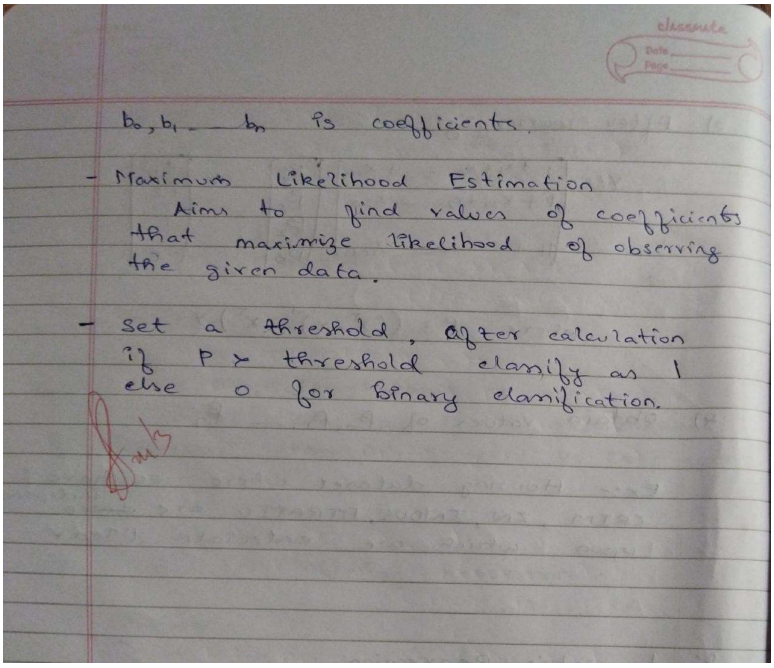
* Logistic Regression.

Algorithm

- Define a Sigmoid function to map the linear combination of independent and dependent variables.

$$y = b_0 + b_1 x_1 + \ldots b_n x_n$$

$$p = \frac{1}{1 + e^{-y}}$$



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

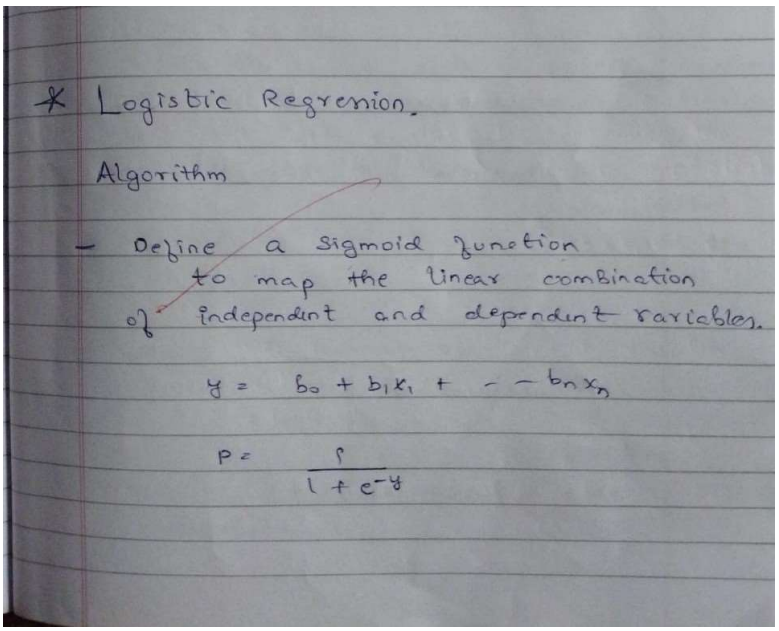
```
import pandas as pd

# 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)
```

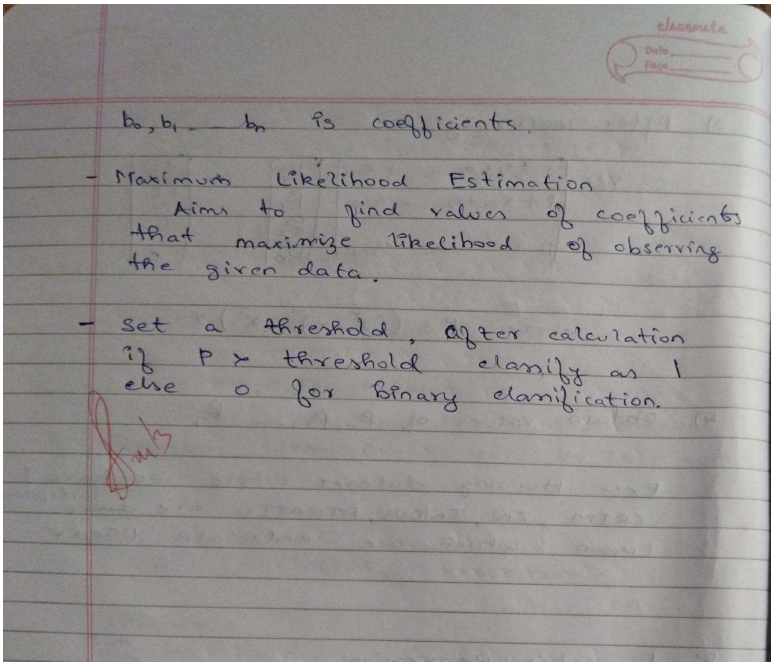
```
# Step 5 : select model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
# Step 6 : train or fit model
model.fit(X_train,y_train)
model.intercept_


model.coef_
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot:





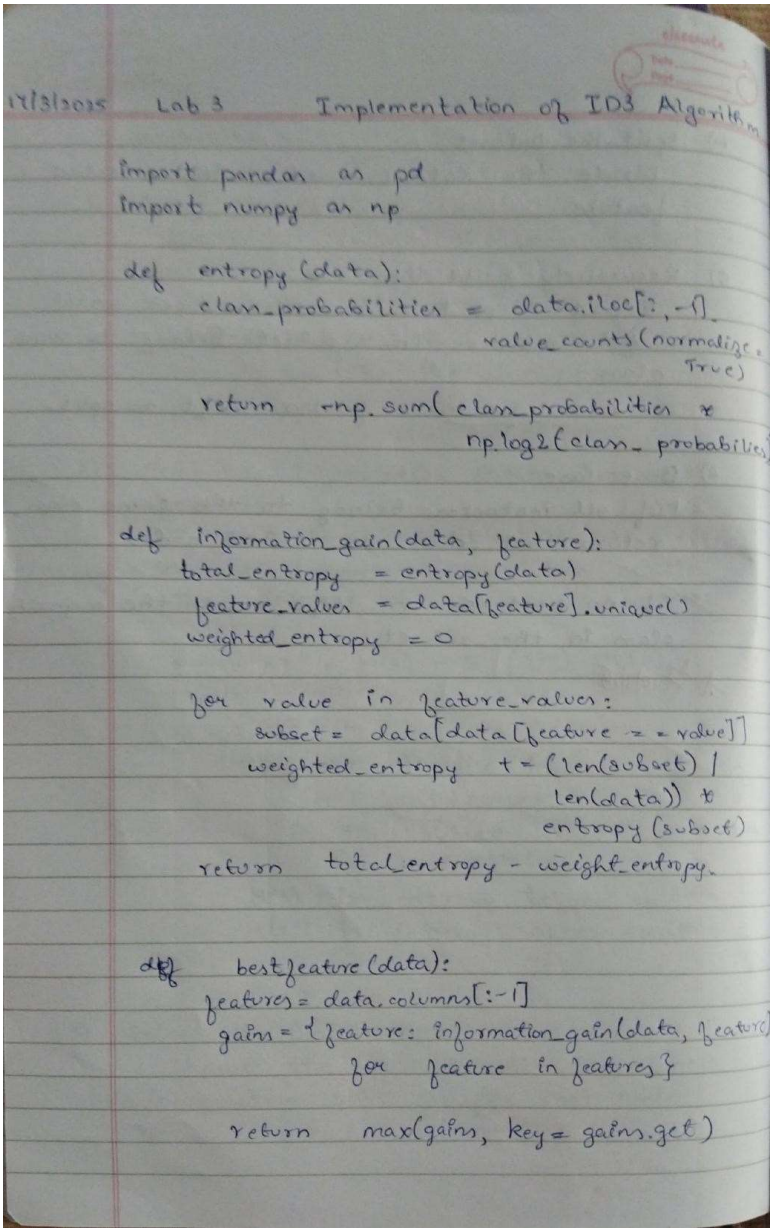
Code:

```
from sklearn.linear_model import LogisticRegression

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score


# Load sample dataset (binary classification - Iris with only 2 classes)

iris = load_iris()

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

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


# Train/Test split

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


# Logistic Regression model

model = LogisticRegression()

model.fit(X_train, y_train)


# Predict and evaluate

y_pred = model.predict(X_test)

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

Program 5

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

Screenshot:



```
17/3/2025   Lab 3   Implementation of ID3 Algorithm

import pandas as pd
import numpy as np

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

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 - weight_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)
```

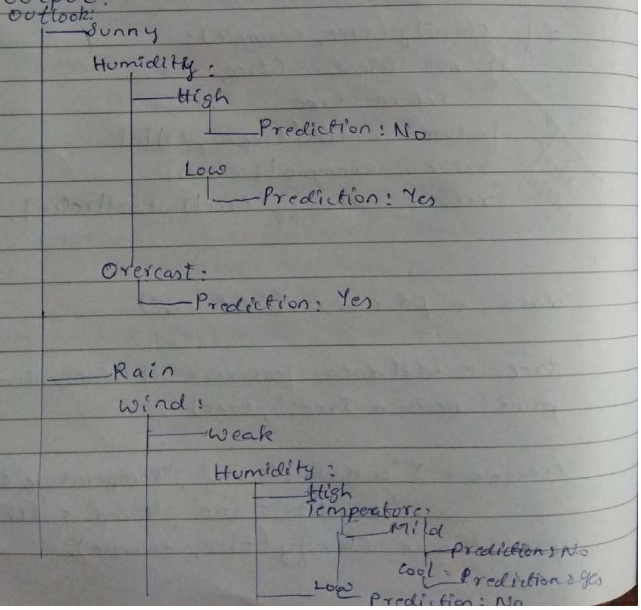
```
def id3(data, features=None):
    if len(data.iloc[:,-1].unique()) == 1:
        return data.iloc[:,-1].mode()[0]

    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.
```

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

```
data = pd.read_csv("data.csv")

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

example = {"outlook": 'Sunny', 'Temperature': 'cool',
           'Humidity': 'low', 'Wind': 'Strong'}
prediction = classify(tree, example).
```

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

def print_tree(tree, indent=" "):
    if isinstance(tree, dict):
        for key, value in tree.items():
            print(f"{indent}{key} :")
            for sub_key, sub_tree in value.items
                print(f"{indent}  {sub_key}")
                print_tree(sub_tree, indent + "
    else:
        print(f"{indent}  Prediction: {tree}")
```

output:



Strong
Prediction: Yes

Prediction for example: Yes

* End to end machine learning project working with real data. Look at the big picture get the data discover. Visualize the data Program the data, select and train the model ~~for~~ fine tune your model.

1) Get the data
Import pandas as pd
housing = pd.read_csv("/sample_data/california.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'])
plt.show()

plt.scatter(house['median_income'], housing['median_house_value'])

```
plt.show()
sns.heatmap(housing.corr(), annot = True)
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 OneHotEn

x = housing.drop('median_house_value', axis=1)
y = housing['median_house_value']

x_train, x_test, y_train, y_test = train_test_s
        (x, y, test_size = 0.2, random_st
                                    = 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_erro
import numpy as np

y_pred = model.predict(x_test)
rmse = root_mean_squared_error(y_test, y_pred)
print(f'RMSE: {rmse}')
```

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):
```

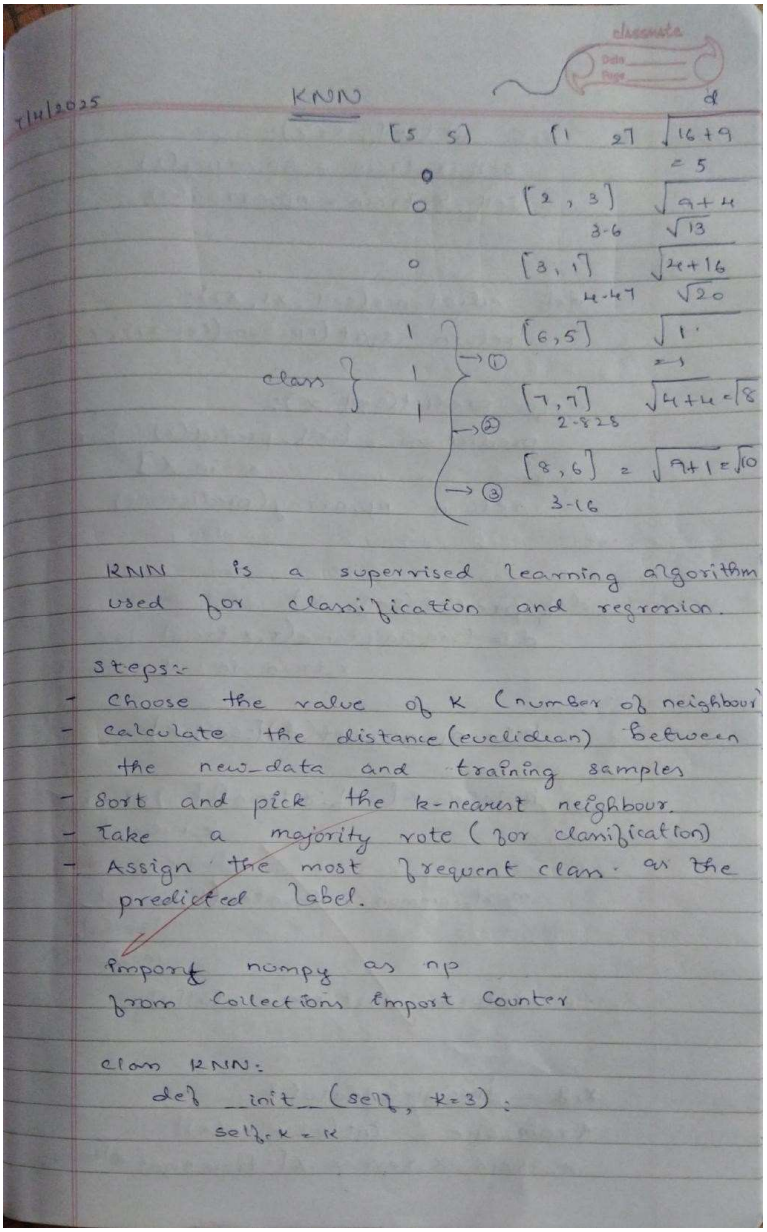
```
    if not isinstance(tree, dict):
        return tree
    feature = list(tree.keys())[0]
    value = example[feature]
    return classify(tree[feature][value], example)

# 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
```

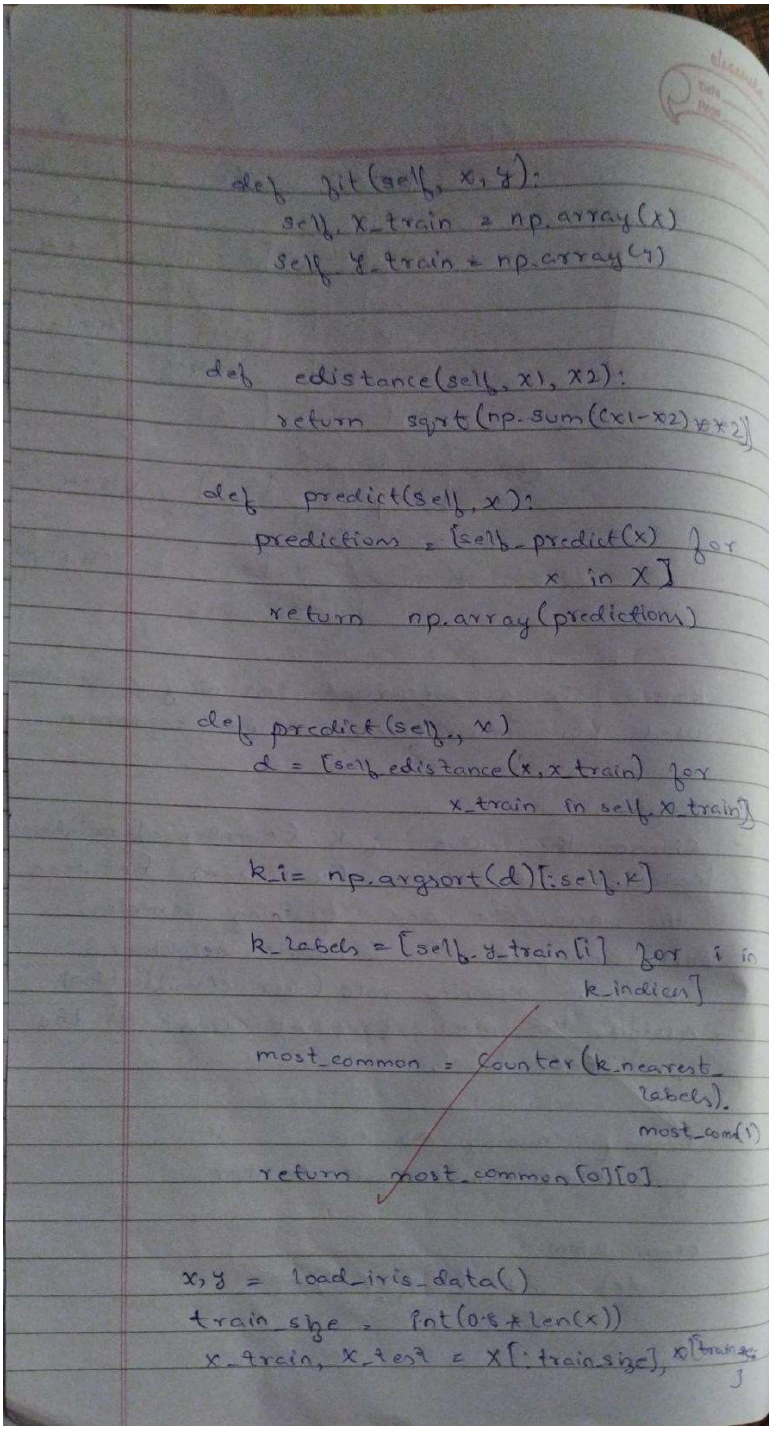
```
        dot.node(parent_name + "_class", f"Class: {tree}")

        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 6

Build KNN Classification model for a given dataset

Screenshot:





```
def fit(self, X, y):
    self.X_train = np.array(X)
    self.y_train = np.array(y)

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

def predict(self, X):
    predictions = [self._predict(x) for
                   x in X]
    return np.array(predictions)

def prredict(self, x)
    d = [self.edistance(x, x_train) for
         x_train in self.X_train]

    k_i = np.argsort(d)[:self.k]

    k_labels = [self.y_train[i] for i in
                k_indices]

    most_common = Counter(k_nearest_
                          labels).
                  most_com(1)

    return most_common[0][0]

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

y_train, y_test = y[:trainsize], y[trainsize:]

km = KNN(k=3)
knn.fit(x_train, y_train)
predicts = knn.predict(x_test)

* Support Vector Machine. (SVM)

Steps :-

1) Input the data
   - Feature matrix X consisting of (samples x features)
   - Target labels y (multi-class 0, 1, 2)
   - Regularization constant c
   - max_iterations

2) Split the Dataset into training set (70%) and testing_set (30%)

3) Initialize the SVM classifier
   set the following parameters.
   C : regularization constant
   max_iter : Number of training iterations
   Kernel - linear defined $K(x \cdot x') = x \cdot x'$

4) One vs Rest Training Strategy
   For each class c in the set of unique classes: convert labels into binary format

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

- Train a binary SVM classifier using the sequential minimal optimization

5) Binary SVM Training

Initialize

$\alpha = 0$ Langrange multipliers

$\beta = 0$ Bias term

Repeat for max_iter: iterations

For each training sample i:

- Randomly select another index
- compute prediction errors $j \neq i$

E_i and E_j

- save old values α_i, α_j
- compute bounds L, H

if $L == H$ continue

- compute

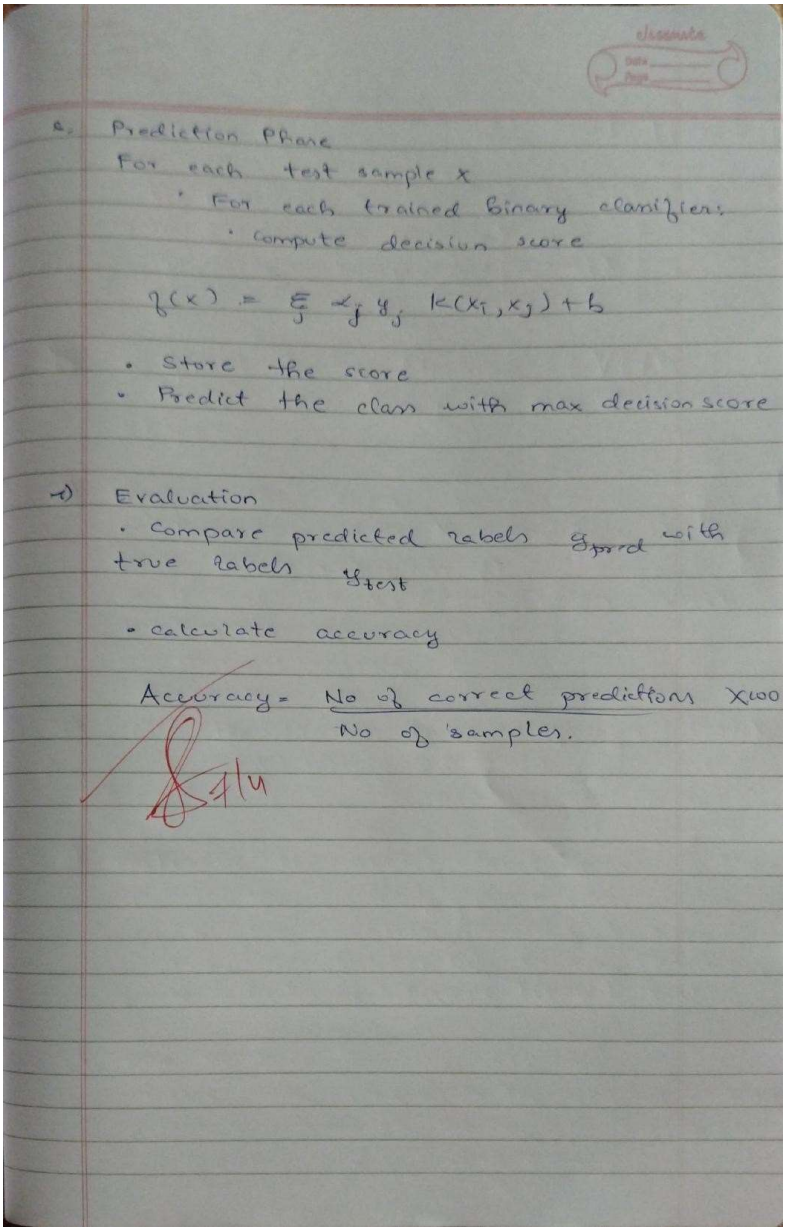
$$\eta = 2K(x_i, x_j) - K(x_i, x_i) - K(x_j, x_j)$$

if $\eta \geq 0$ skip update

- update α_j

$$\alpha_j = \alpha_j + y_j \frac{(E_i - E_j)}{\eta}$$

- clip $\alpha_j \in [L, H]$
- update α_i
- compute b



Code:

KNN

```
import numpy as np

from collections import Counter


class KNN:
```

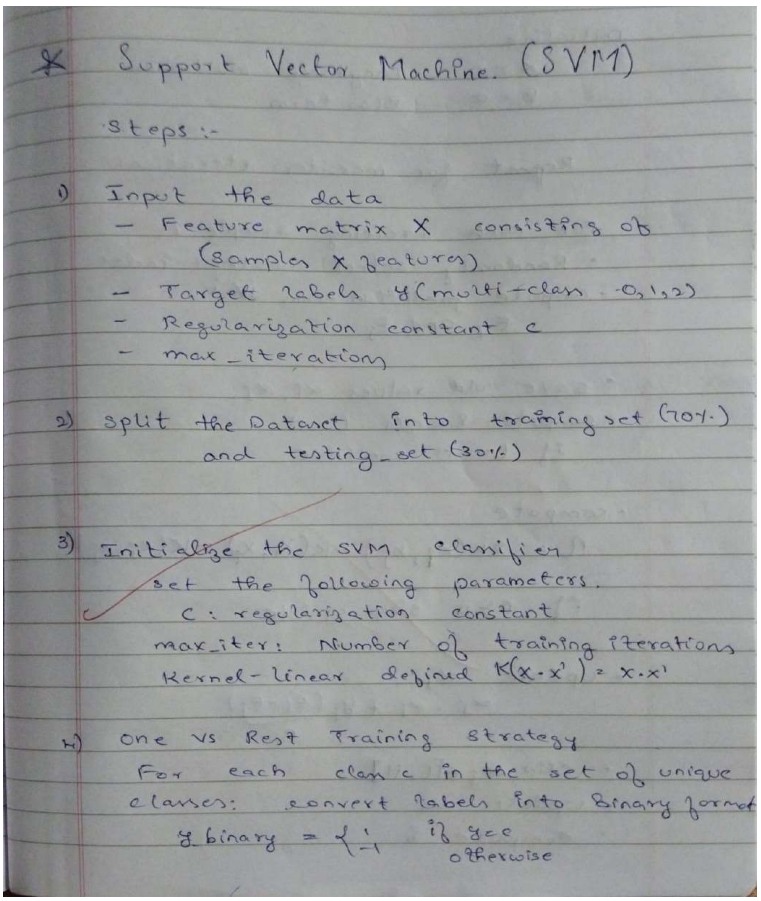
```
def __init__(self, k=3):
    self.k = k

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:



- Train a binary SVM classifier using the sequential minimal optimization

5) Binary SVM Training

Initialize
$\alpha = 0$ Langrange multipliers
$\beta = 0$ Bias term

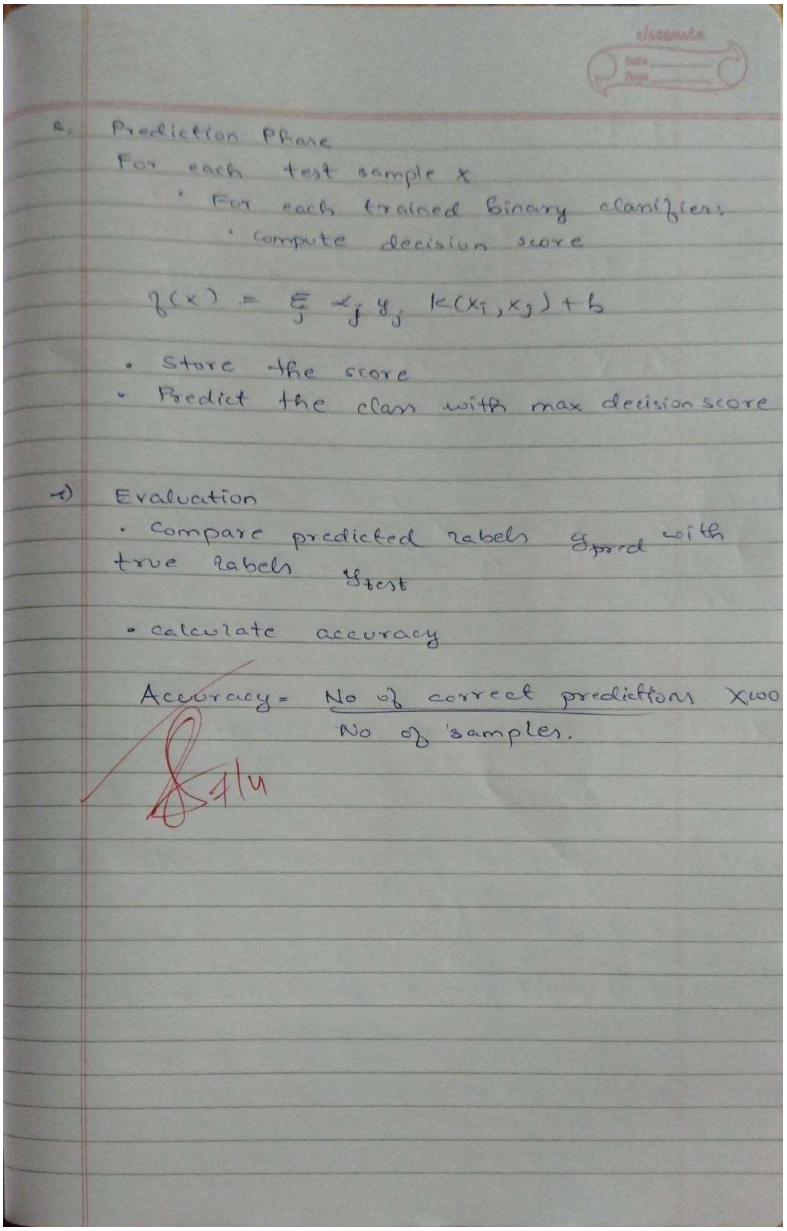
Repeat for max_iter: iterations

For each training sample i:
- Randomly select another index
- compute prediction errors $j \neq i$
  E_i and E_j

- save old values α_i, α_j
- compute bounds L, H
  if $L == H$ continue

- compute
$$\eta = 2K(x_i, x_j) - K(x_i, x_i) - K(x_j, x_j)$$
if $\eta \geq 0$ skip update

- update α_j
$$\alpha_j = \alpha_j + y_j \frac{(E_i - E_j)}{\eta}$$
- clip $\alpha_j \in [L, H]$
- update α_i
- compute b



Code:

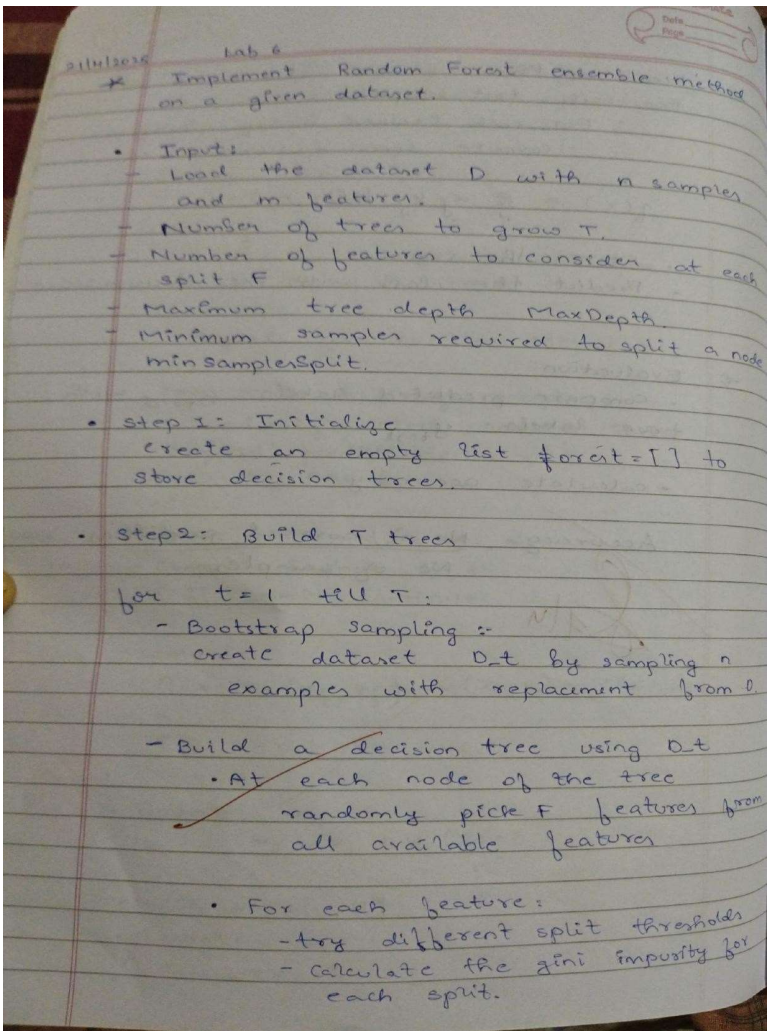
```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import matplotlib.pyplot as plt
```

```
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 6

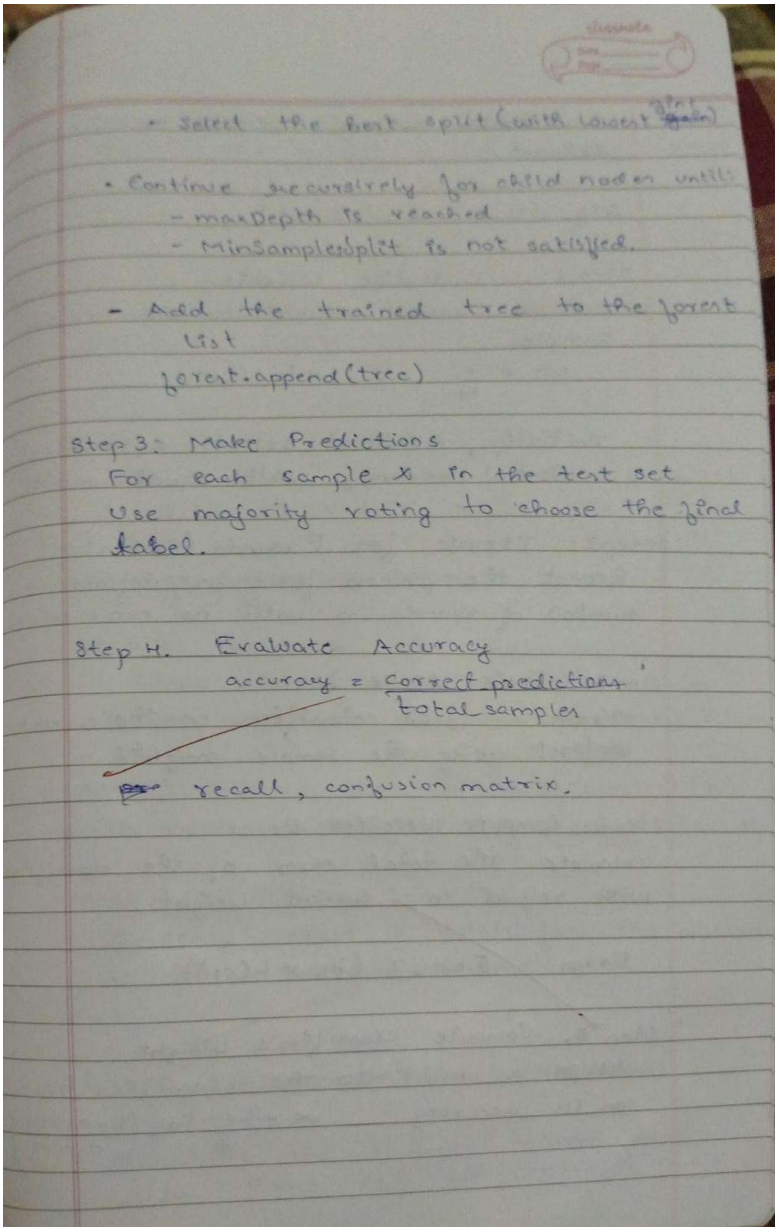
* Implement Random Forest ensemble method on a given dataset.

- Input:
  - Load the dataset D with n samples and m features.
  - Number of trees to grow T.
  - Number of features to consider at each split F
  - Maximum tree depth MaxDepth.
  - Minimum samples required to split a node minSamplesSplit.

- Step 1: Initialize
  create an empty list forest = [] to store decision trees.

- Step 2: Build T trees

for t = 1 till T:
  - Bootstrap sampling :-
    create dataset D_t by sampling n examples with replacement from D.
  - Build a decision tree using D_t
    - At each node of the tree randomly pick F features from all available features
    - For each feature:
      - try different split thresholds
      - calculate the gini impurity for each split.



Code:

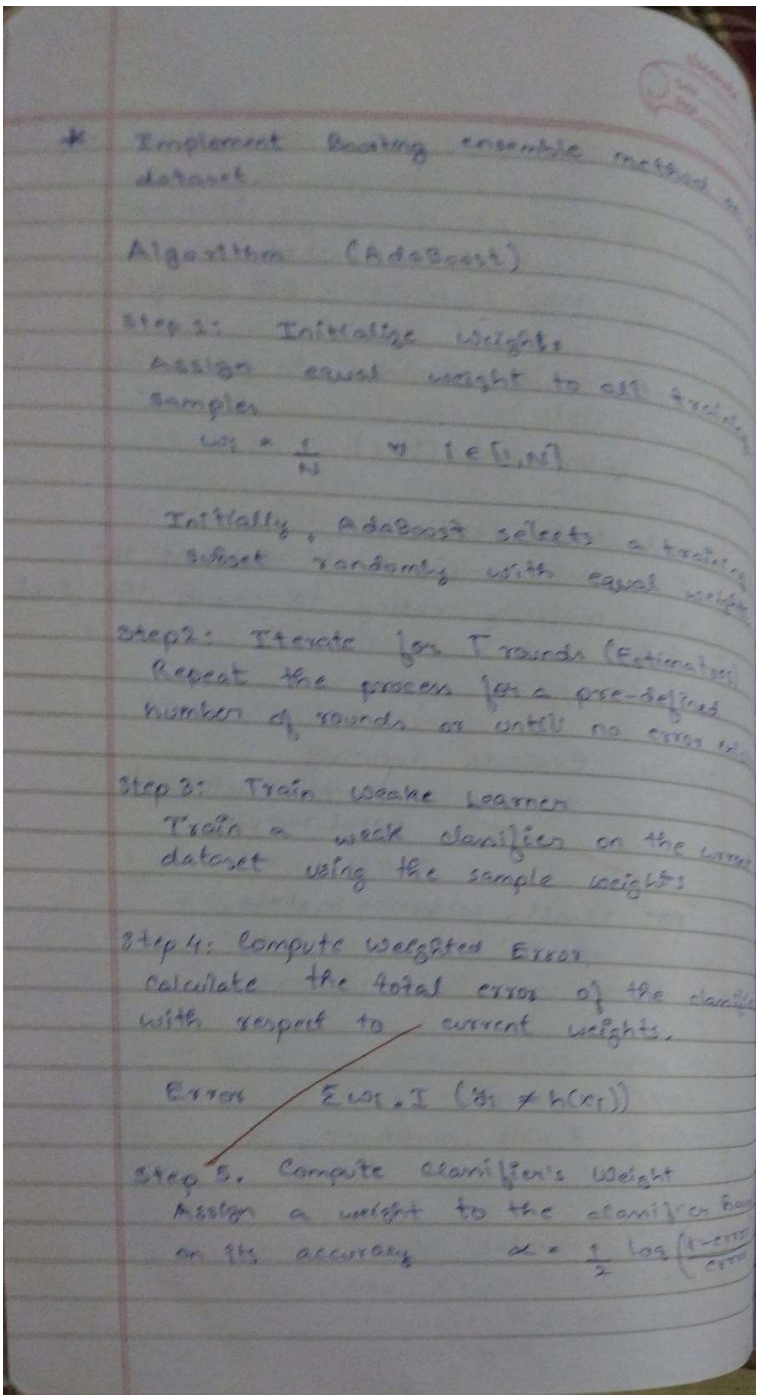
```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score
```

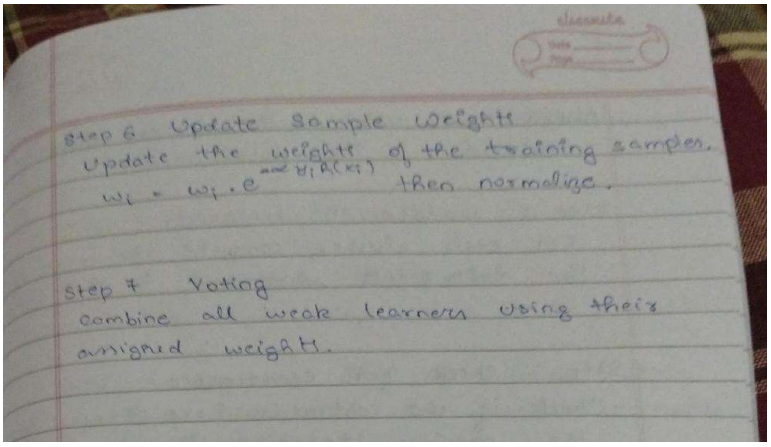
```
# Load sample dataset
iris = load_iris()
X, y = iris.data, iris.target

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

# Initialize Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Predict and evaluate
y_pred = rf.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot:





Code:

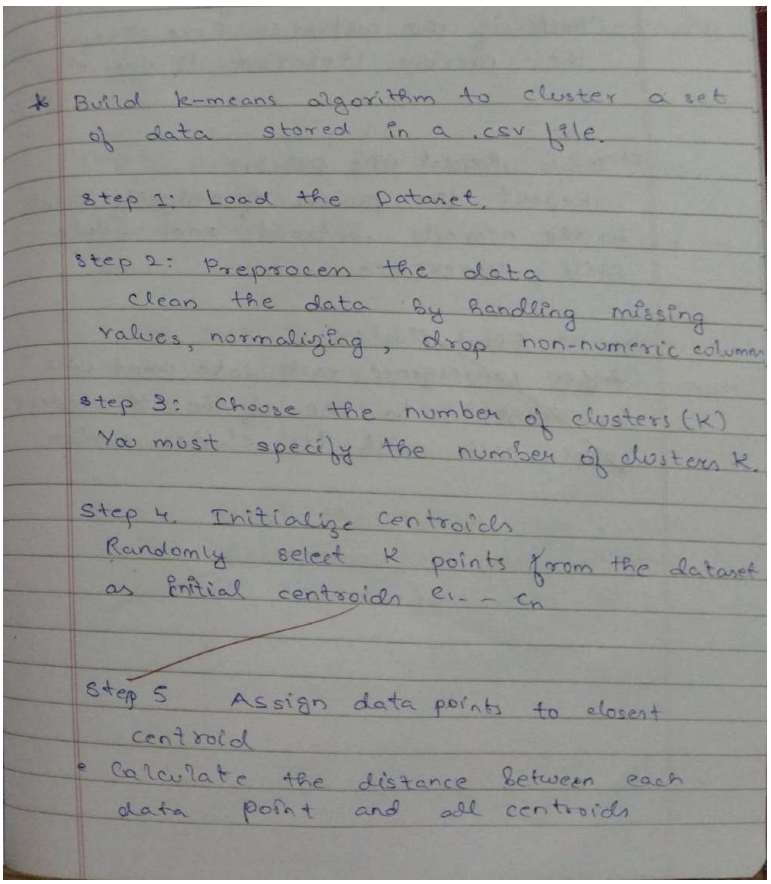
```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load Iris dataset
iris = load_iris()
X, y = iris.data, iris.target
```

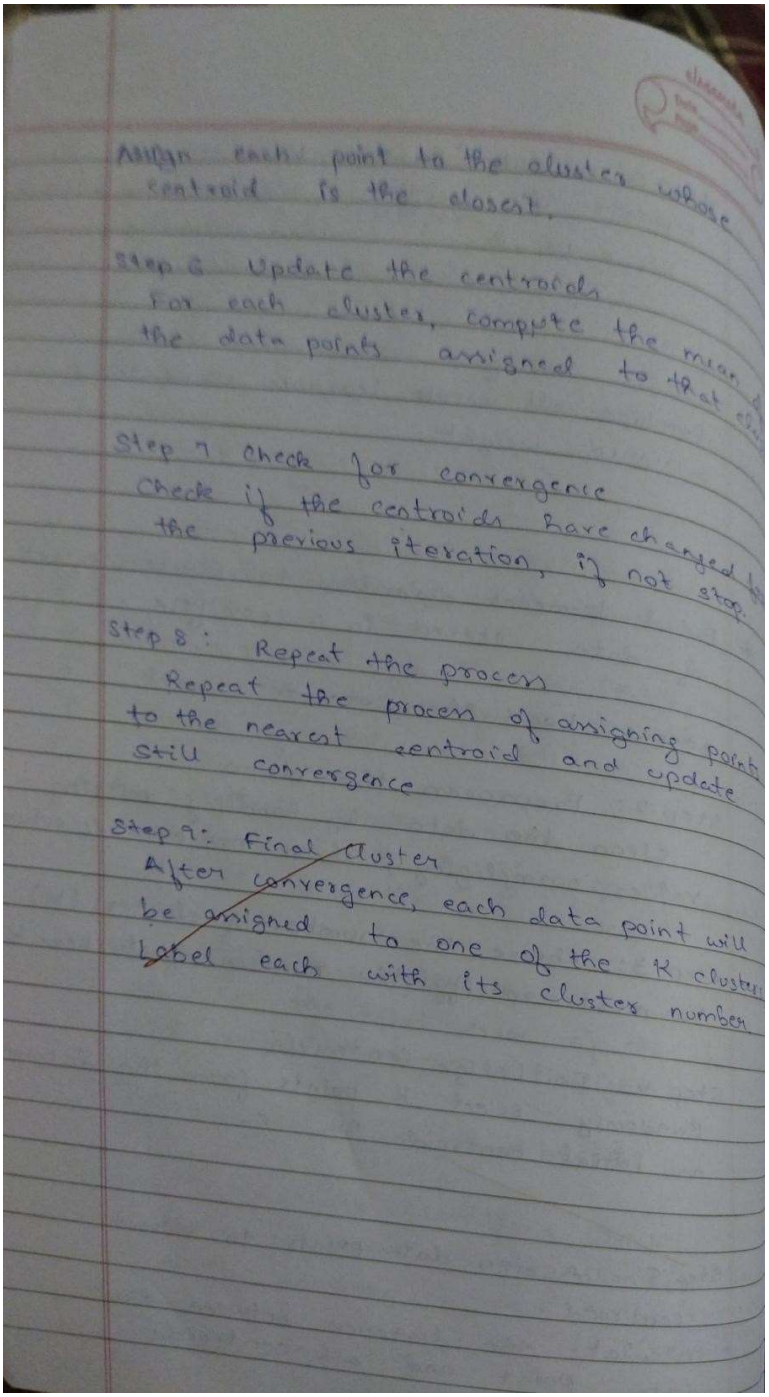
```
# For AdaBoost, we'll use binary classification
# Convert to binary (setosa vs. not-setosa)
y = (y == 0).astype(int)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train AdaBoost
model = AdaBoostClassifier(n_estimators=50, learning_rate=1.0, random_state=42)
model.fit(X_train, y_train)

# Predict and evaluate
y_pred = model.predict(X_test)
print("AdaBoost Accuracy (sklearn):", accuracy_score(y_test, y_pred))
```

Program 10

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

Screenshot:





Code:

import pandas as pd

from sklearn.cluster import KMeans

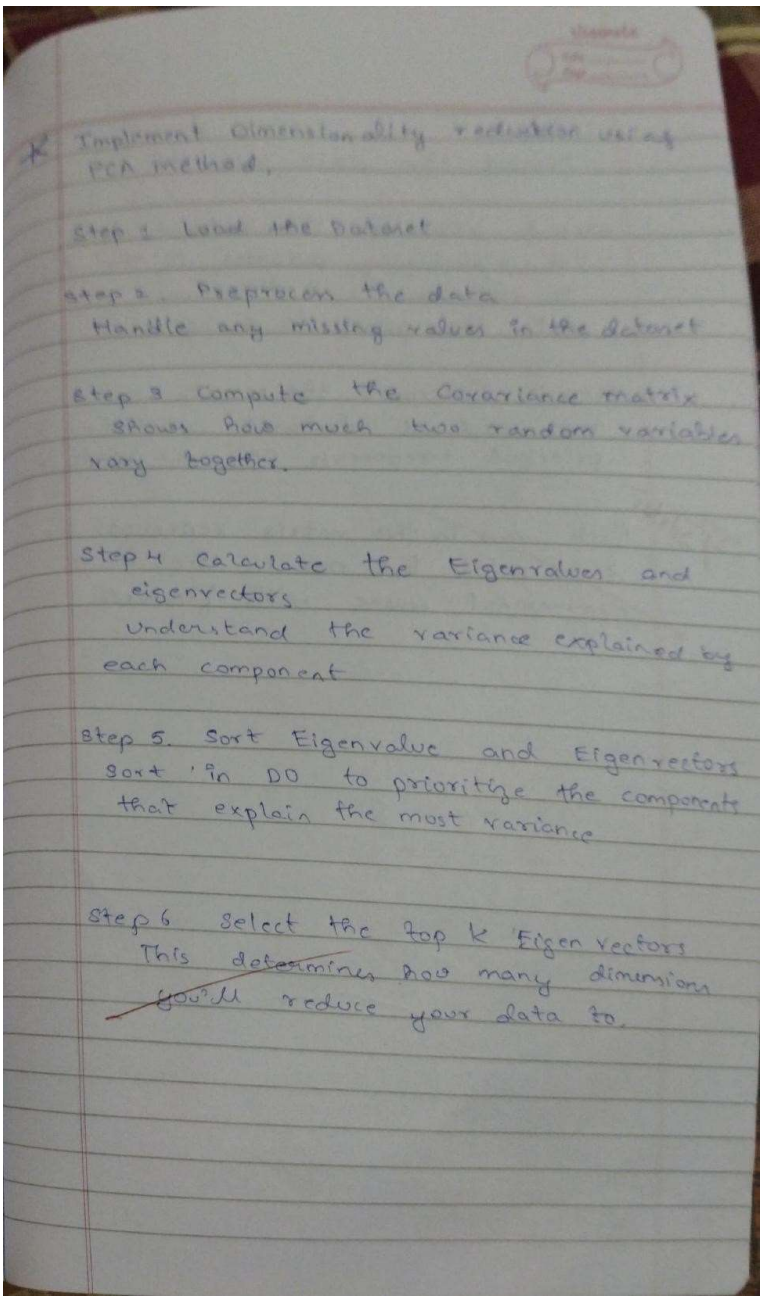
```
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris # Import load_iris


# Step 1: Load the Iris dataset directly

iris = load_iris()

# Create a DataFrame from the data and target

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

# Add the target column for potential reference, though not used for clustering

data['target'] = iris.target




# Step 2: Extract only numeric columns (or select required features)

# All features in the Iris dataset are numeric

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


# Step 3: Apply KMeans

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

kmeans = KMeans(n_clusters=3, random_state=42, n_init=10) # Added n_init to suppress future
warnings

data['Cluster'] = kmeans.fit_predict(X)


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

# Iris data has 4 features. We will plot the first two features for visualization.

if X.shape[1] >= 2:

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

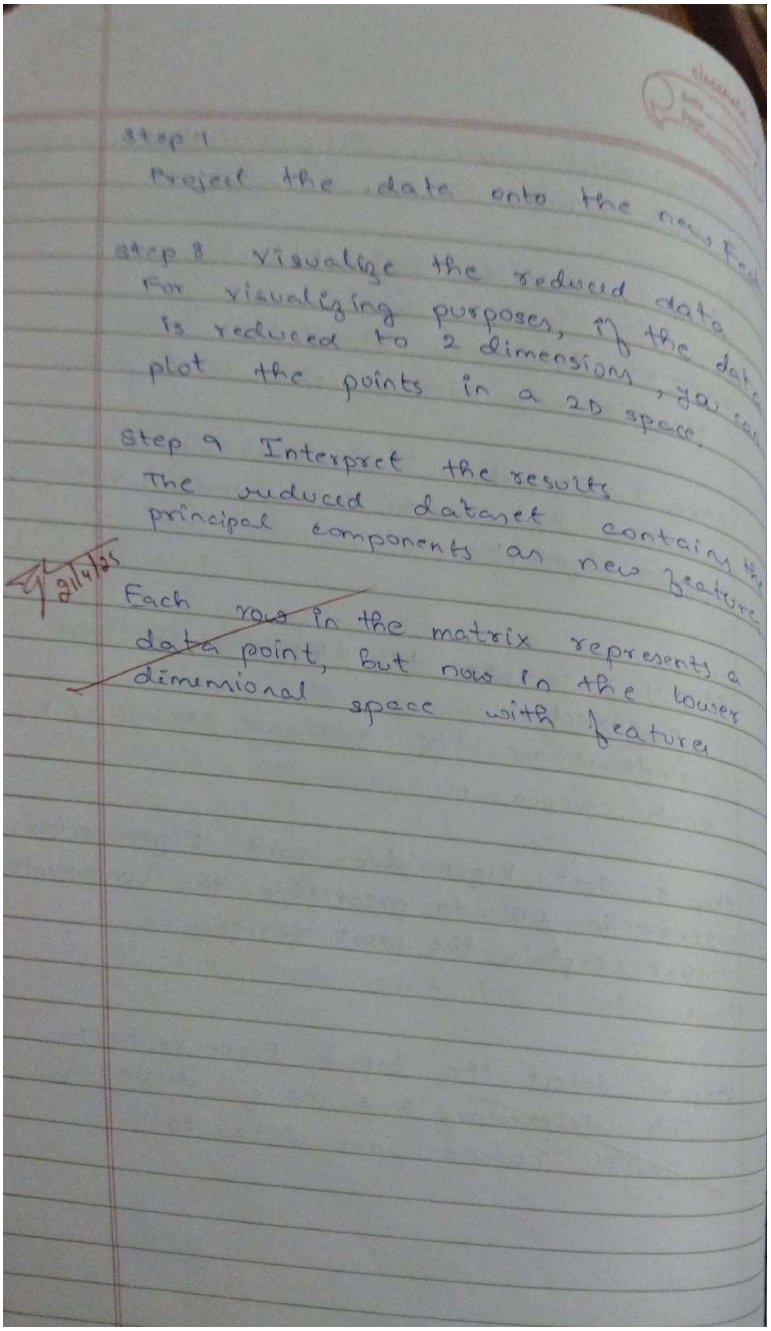
```
    plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], color='red', marker='x',
s=200)

    plt.title("K-Means Clustering of Iris Dataset")

    plt.xlabel(iris.feature_names[0]) # Label with actual feature name

    plt.ylabel(iris.feature_names[1]) # Label with actual feature name

    plt.show()

else:

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

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot:





Code:

```
import pandas as pd

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler
```

```
import matplotlib.pyplot as plt

# Load dataset
data = pd.read_csv("your_data.csv")  # Replace with your file
X = data.select_dtypes(include=['float64', 'int64'])

# Step 1: Standardize
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 2: Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Print explained variance ratio
print("Explained variance ratio:", pca.explained_variance_ratio_)

# Visualize
plt.scatter(X_pca[:, 0], X_pca[:, 1], c='blue', alpha=0.5)
plt.title("PCA - 2D Projection")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```