VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Shipra Kumari (1BM22CS341)

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 **Shipra Kumari (1BM22CS341),** 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.

Sarala D V Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

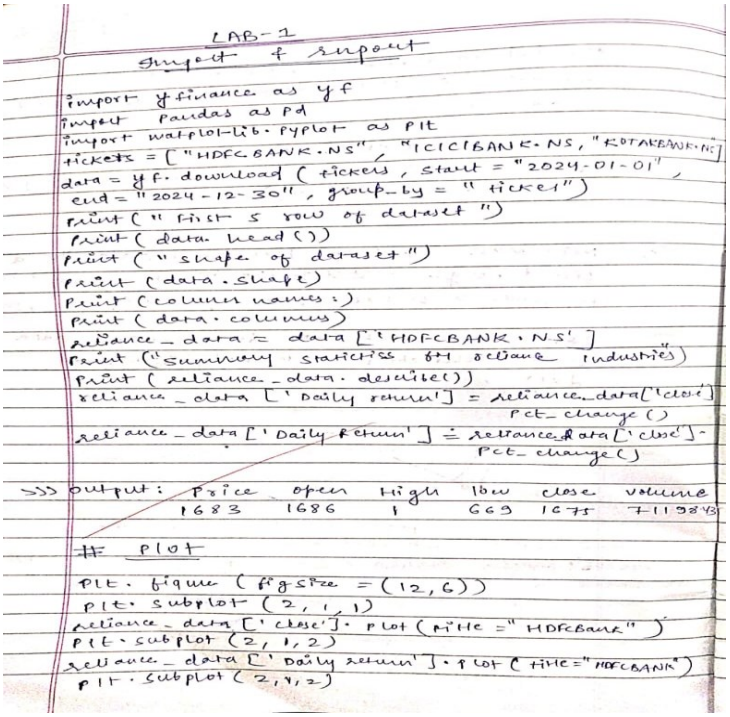
Index

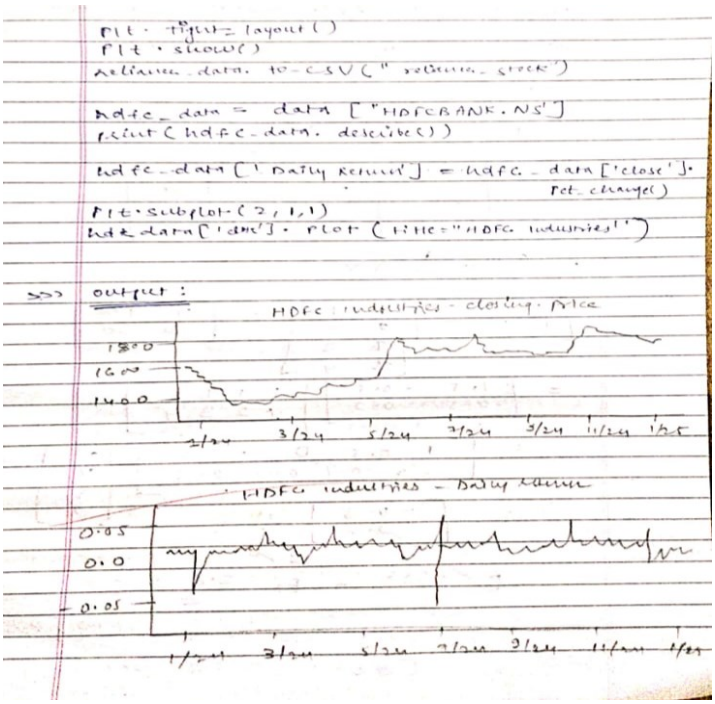
Github Link: https://github.com/singhhshipra/ML_LAB_

Program 1

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

Screenshot





Code:

```
import pandas as pd

# Create a DataFrame directly from a dictionary

data = {

'Name': ['Alice', 'Bob', 'Charlie', 'David'],

'Age': [25, 30, 35, 40],

'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']

}

df = pd.DataFrame(data)

print("Sample data:")

print(df.head())

from sklearn.datasets import load_iris

iris = load_iris()

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

df['target'] = iris.target

print("Sample data:")

print(df.head())


# Load data from a CSV file (replace 'data.csv' with your file path)

file_path = 'data.csv' # Ensure the file exists in the same directory

df = pd.read_csv(file_path)

print("Sample data:")

print(df.head())

print("\n")
```

```
import pandas as pd

# Reading data from a CSV file

df =pd.read_csv('sample_sales_data.csv')

# Displaying the first few rowsof the DataFrame

print(df.head())

# Writing the DataFrameto a CSV file

df.to_csv('output.csv',index=False)

print("Data saved to output.csv")


# Reading sales data from a CSV file

sales_df = pd.read_csv('sample_sales_data.csv')

# Displaying the first few rows of the dataset

print("First few rows of the sales data:")

print(sales_df.head())

# Grouping by Region and calculating total sales

sales_by_region =sales_df.groupby('Region')['Sales'].sum()

print("\nTotal sales by region:")

print(sales_by_region)

# Grouping by Product and calculating total quantity sold

best_selling_products = sales_df.groupby('Product')['Quantity'].sum().sort_values(ascending=False)

print("\nBest-selling products by quantity:")

print(best_selling_products)


# Saving the sales by region data to a CSV file
```

```
sales_by_region.to_csv('sales_by_region.csv')

# Saving the best-selling products data to a CSV file

best_selling_products.to_csv('best_selling_products.csv')

print("\nAnalysis results saved to CSV files.")


# Step 1: Import required libraries

import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt

# Step 2: Downloading Stock Market Data

# Define the ticker symbols for Indian companies

# Example: Reliance Industries (RELIANCE.NS), TCS (TCS.NS), Infosys (INFY.NS)

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())

# Step 3: Basic Data Exploration

# Check the shape of the dataset

print("\nShape of the dataset:")

print(data.shape)

# Check column names
```

```
print("\nColumn names:")
print(data.columns)
# 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()
# 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()
# Step 4: Saving the Processed Data to a New CSV File
# Save the Reliance data to a CSV file
reliance_data.to_csv('reliance_stock_data.csv')
print("\nReliance stock data saved to 'reliance_stock_data.csv'.")


# Step 1: Import required libraries
```

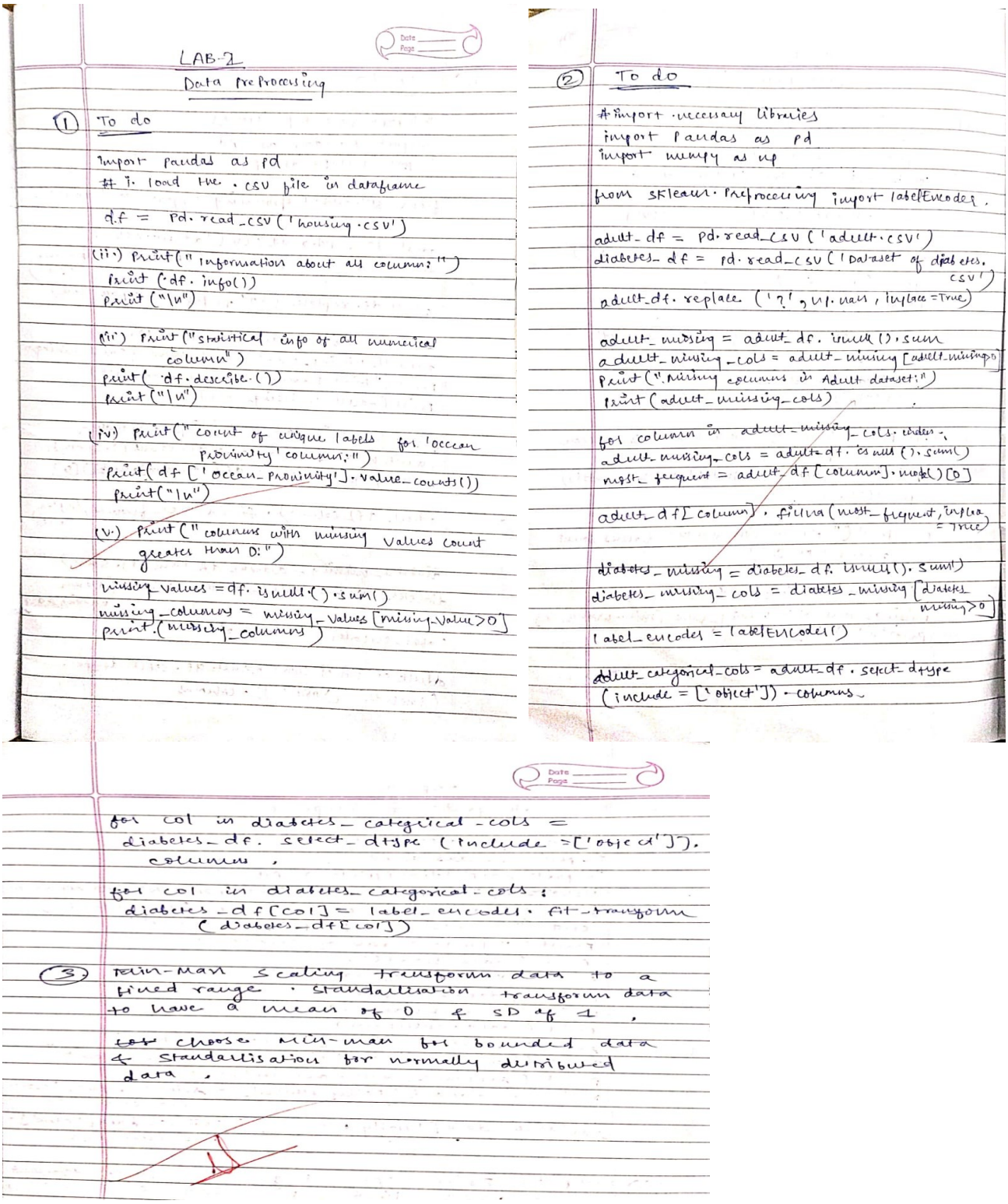
```
import yfinance as yf

import pandas as pd

import matplotlib.pyplot as plt

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

# Fetch historical data for the last 1 year

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

# Display the first 5 rows of the dataset

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

print(data.head())

# Step 3: Basic Data Exploration

# Check the shape of the dataset

print("\nShape of the dataset:")

print(data.shape)

# Check column names

print("\nColumn names:")

print(data.columns)

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

reliance_data = data['HDFCBANK.NS']

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

print(reliance_data.describe())

# Calculate daily returns

reliance_data['Daily Return'] = reliance_data['Close'].pct_change()

# 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="HDFCBANK - Closing Price")
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="HDFCBANK - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
# Step 4: Saving the Processed Data to a New CSV File
# Save the Reliance data to a CSV file
reliance_data.to_csv('reliance_stock_data.csv')
print("\nReliance stock data saved to 'reliance_stock_data.csv'.")
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset.

Screenshot



LAB-2
Data PreProcessing

(1) To do

```
import pandas as pd
# i. load the .csv file in dataframe
df = pd.read_csv('housing.csv')
(ii) print("information about all columns:")
print(df.info())
print("\n")
(ii) print("statistical info of all numerical columns")
print(df.describe())
print("\n")
(iv) print("count of unique labels for 'ocean_proximity' columns:")
print(df['ocean_proximity'].value_counts())
print("\n")
(v.) print("columns with missing values count greater than 0:")
missing_values = df.isnull().sum()
missing_columns = missing_values[missing_value>0]
print(missing_columns)
```

(2) To do

```
#import necessary libraries
import pandas as pd
import numpy as np

from sklearn.preprocessing import labelEncoder.

adult_df = pd.read_csv('adult.csv')
diabetes_df = pd.read_csv('Dataset of diabetes.csv')
adult_df.replace('?', np.nan, inplace=True)

adult_missing = adult_df.isnull().sum
adult_missing_cols = adult_missing[adult_missing>0]
print("missing columns in Adult dataset:")
print(adult_missing_cols)

for column in adult_missing_cols.index:
adult_missing_cols = adult=df.isnull().sum()
most_frequent = adult_df[column].mode()[0]

adult_df[column].fillna(most_frequent, inplace=True)

diabetes_missing = diabetes_df.isnull().sum()
diabetes_missing_cols = diabetes_missing[diabetes_missing>0]

label_encoder = labelEncoder()

adult_categorical_cols = adult_df.select_dtype(include=['object']).columns
```

```
for col in diabetes_categorical_cols = diabetes_df.select_dtype(include=['object']).columns,

for col in diabetes_categorical_cols:
diabetes_df[col] = label_encoder.fit_transform(diabetes_df[col])
```

(3) Min-Max Scaling transform data to a fixed range. Standardisation transform data to have a mean of 0 & SD of 1,

choose Min-max for bounded data & Standardisation for normally distributed data.

Code

```
import pandas as pd

# i. Load the .csv file into a DataFrame
df = pd.read_csv('housing.csv')

# ii. Display information of all columns
print("Information about all columns:")
print(df.info())
print("\n")

# iii. Display statistical information of all numerical columns
print("Statistical information of all numerical columns:")
print(df.describe())
print("\n")

# iv. Display the count of unique labels for the "Ocean Proximity" column
print("Count of unique labels for 'Ocean Proximity' column:")
print(df['ocean_proximity'].value_counts())
print("\n")

# v. Display which attributes (columns) have missing values count greater than zero
print("Columns with missing values count greater than zero:")
```

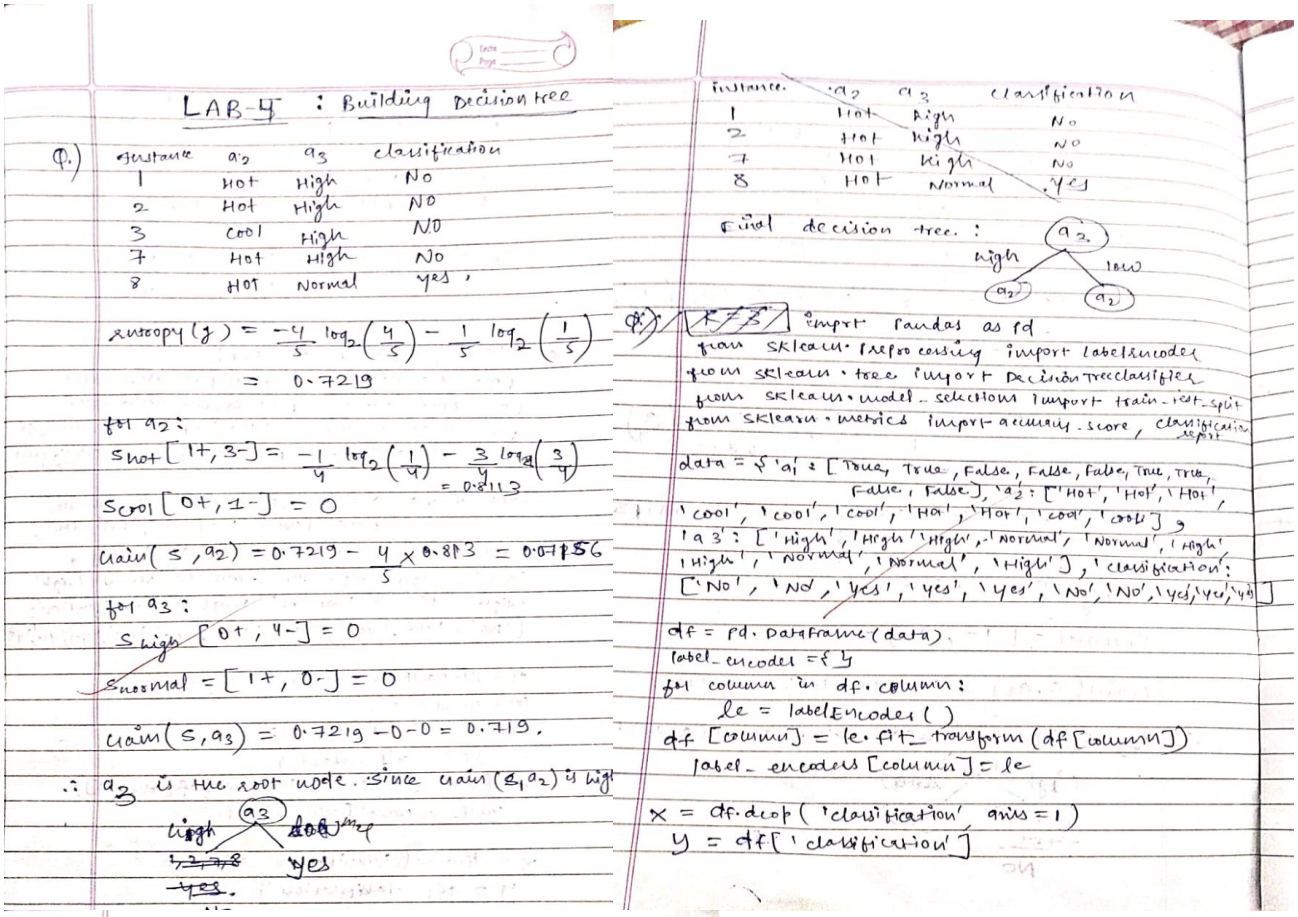
```
missing_values = df.isnull().sum()
missing_columns = missing_values[missing_values > 0]
print(missing_columns)
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
# Load the datasets
adult_df = pd.read_csv('adult.csv')
diabetes_df = pd.read_csv('Dataset of Diabetes .csv')

# ---- Part 1: Handling Missing Values ----
# Replace '?' with NaN in the Adult Income dataset
adult_df.replace('?', np.nan, inplace=True)
# Check for missing values in Adult Income dataset
adult_missing = adult_df.isnull().sum()
adult_missing_cols = adult_missing[adult_missing > 0]
print("Missing columns in Adult Income dataset:")
print(adult_missing_cols)

# Handle missing values in Adult Income dataset by replacing with mode for categorical columns
for column in adult_missing_cols.index:
    most_frequent = adult_df[column].mode()[0]
    adult_df[column].fillna(most_frequent, inplace=True)
```

```
# Check for missing values in Diabetes dataset
diabetes_missing = diabetes_df.isnull().sum()
diabetes_missing_cols = diabetes_missing[diabetes_missing > 0]
print("\nMissing columns in Diabetes dataset:")
print(diabetes_missing_cols)

# ---- Part 2: Encoding Categorical Columns ----

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Identify and encode categorical columns in Adult Income dataset
adult_categorical_cols = adult_df.select_dtypes(include=['object']).columns
for col in adult_categorical_cols:
    adult_df[col] = label_encoder.fit_transform(adult_df[col])
print("\nCategorical columns in Adult Income dataset encoded.")

# Identify and encode categorical columns in Diabetes dataset
diabetes_categorical_cols = diabetes_df.select_dtypes(include=['object']).columns
for col in diabetes_categorical_cols:
    diabetes_df[col] = label_encoder.fit_transform(diabetes_df[col])
print("Categorical columns in Diabetes dataset encoded.")
```

Program 3

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

Screenshot



Code

```
# -*- coding: utf-8 -*-
"""Decision_Tree.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1RXDK8CR1doVCMHgkaXpJsNLAvzOIaXdd
"""

import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Create the dataset
data = {
    'a1': [True, True, False, False, False, True, True, True, False, False],
    'a2': ['Hot', 'Hot', 'Hot', 'Cool', 'Cool', 'Cool', 'Hot', 'Hot', 'Cool', 'Cool'],
    'a3': ['High', 'High', 'High', 'Normal', 'Normal', 'High', 'High', 'Normal', 'Normal', 'High'],
    'Classification': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes']
}
```

```
data


# Convert to DataFrame
df = pd.DataFrame(data)


# Convert categorical data to numerical data
label_encoders = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le


# Split the dataset into features and target
X = df.drop('Classification', axis=1)
y = df['Classification']


# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


# Initialize the Decision Tree Classifier with entropy as the criterion
clf = DecisionTreeClassifier(criterion='entropy')


# Train the classifier
clf.fit(X_train, y_train)
```

```
# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred, target_names=['No', 'Yes']))

# Optionally, visualize the decision tree
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=['No', 'Yes'])
plt.show()# -*- coding: utf-8 -*-
"""Decision_Tree.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1RXDK8CR1doVCMHgkaXpJsNLAvzOIaXdd
"""
```

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Create the dataset
data = {
    'a1': [True, True, False, False, False, True, True, True, False, False],
    'a2': ['Hot', 'Hot', 'Hot', 'Cool', 'Cool', 'Cool', 'Hot', 'Hot', 'Cool', 'Cool'],
    'a3': ['High', 'High', 'High', 'Normal', 'Normal', 'High', 'High', 'Normal', 'Normal', 'High'],
    'Classification': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes']
}

data

# Convert to DataFrame
df = pd.DataFrame(data)

# Convert categorical data to numerical data
label_encoders = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
```

```
    label_encoders[column] = le

# Split the dataset into features and target
X = df.drop('Classification', axis=1)
y = df['Classification']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the Decision Tree Classifier with entropy as the criterion
clf = DecisionTreeClassifier(criterion='entropy')

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred, target_names=['No', 'Yes']))

# Optionally, visualize the decision tree
```

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=['No', 'Yes'])
plt.show()
# -*- coding: utf-8 -*-
"""Decision_Tree.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1RXDK8CR1doVCMHgkaXpJsNLAvzOIaXdd
"""

import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Create the dataset
data = {
    'a1': [True, True, False, False, False, True, True, True, False, False],
```

```
    'a2': ['Hot', 'Hot', 'Hot', 'Cool', 'Cool', 'Cool', 'Hot', 'Hot', 'Cool', 'Cool'],
    'a3': ['High', 'High', 'High', 'Normal', 'Normal', 'High', 'High', 'Normal', 'Normal', 'High'],
    'Classification': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes']
}

data

# Convert to DataFrame
df = pd.DataFrame(data)

# Convert categorical data to numerical data
label_encoders = {}
for column in df.columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Split the dataset into features and target
X = df.drop('Classification', axis=1)
y = df['Classification']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Initialize the Decision Tree Classifier with entropy as the criterion
clf = DecisionTreeClassifier(criterion='entropy')

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred, target_names=['No', 'Yes']))

# Optionally, visualize the decision tree
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=['No', 'Yes'])
plt.show()
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

```
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

from sklearn.preprocessing import LabelEncoder


# Load IRIS dataset

iris_df = pd.read_csv("iris (1).csv")

X_iris = iris_df.iloc[:, :-1]  # Features (all columns except last)

y_iris = iris_df.iloc[:, -1]   # Target (last column)


# Encode categorical target variable if necessary

y_iris = LabelEncoder().fit_transform(y_iris)


# Split data into train (80%) and test (20%)

X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris, test_size=0.2,
random_state=42)


# Train Decision Tree model

clf_iris = DecisionTreeClassifier()

clf_iris.fit(X_train_iris, y_train_iris)


# Predictions

y_pred_iris = clf_iris.predict(X_test_iris)


# Evaluation

accuracy_iris = accuracy_score(y_test_iris, y_pred_iris)
```

```
conf_matrix_iris = confusion_matrix(y_test_iris, y_pred_iris)


print("IRIS Dataset Results:")
print(f"Accuracy: {accuracy_iris:.2f}")
print("Confusion Matrix:")
print(conf_matrix_iris)


# Load Drug dataset
drug_df = pd.read_csv("drug.csv")


# Encode categorical features
label_encoders = {}
for column in drug_df.columns[:-1]:  # Encode only feature columns if necessary
    if drug_df[column].dtype == 'object':
        le = LabelEncoder()
        drug_df[column] = le.fit_transform(drug_df[column])
        label_encoders[column] = le


X_drug = drug_df.iloc[:, :-1]  # Features (all columns except last)
y_drug = LabelEncoder().fit_transform(drug_df.iloc[:, -1])  # Encode target variable


# Split data into train (80%) and test (20%)
X_train_drug, X_test_drug, y_train_drug, y_test_drug = train_test_split(X_drug, y_drug,
test_size=0.2, random_state=42)
```

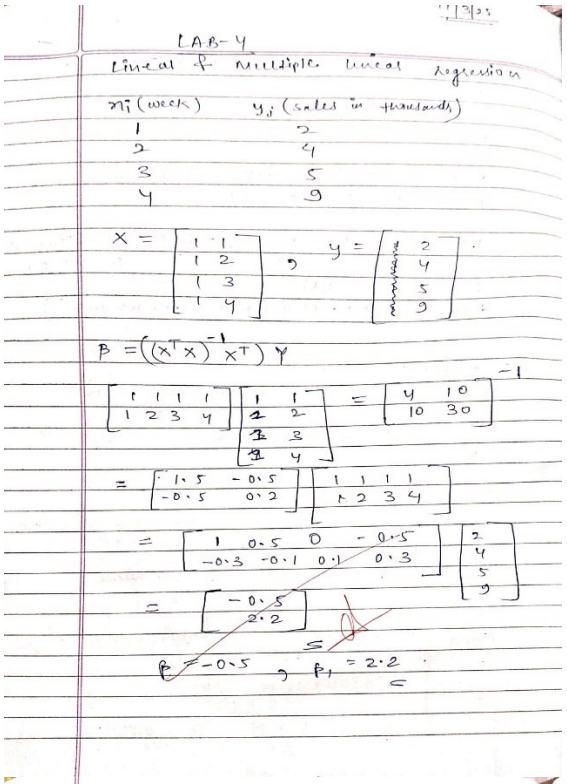
```
# Train Decision Tree model
clf_drug = DecisionTreeClassifier()
clf_drug.fit(X_train_drug, y_train_drug)

# Predictions
y_pred_drug = clf_drug.predict(X_test_drug)

# Evaluation
accuracy_drug = accuracy_score(y_test_drug, y_pred_drug)
conf_matrix_drug = confusion_matrix(y_test_drug, y_pred_drug)

print("\nDrug Dataset Results:")
print(f"Accuracy: {accuracy_drug:.2f}")
print("Confusion Matrix:")
print(conf_matrix_drug)

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Load dataset
iris_df = pd.read_csv("petrol_consumption (1).csv")
```

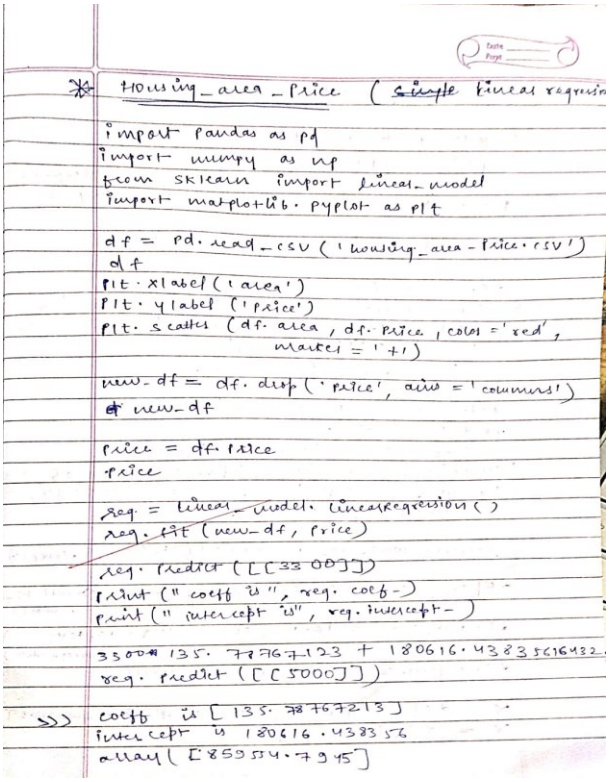
```
# Assuming the last column is the target variable 'Petrol_Consumption'
X = df.iloc[:, :-1]  # Features (all columns except the last one)
y = df.iloc[:, -1]   # Target variable

# Split dataset into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and train DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)

# Predict on test data
y_pred = regressor.predict(X_test)

# Evaluate model performance
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
```

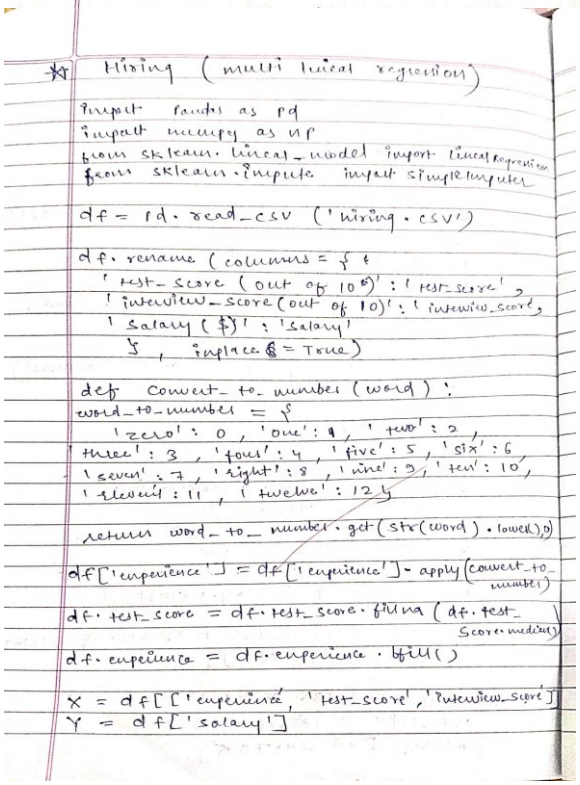
Program 4

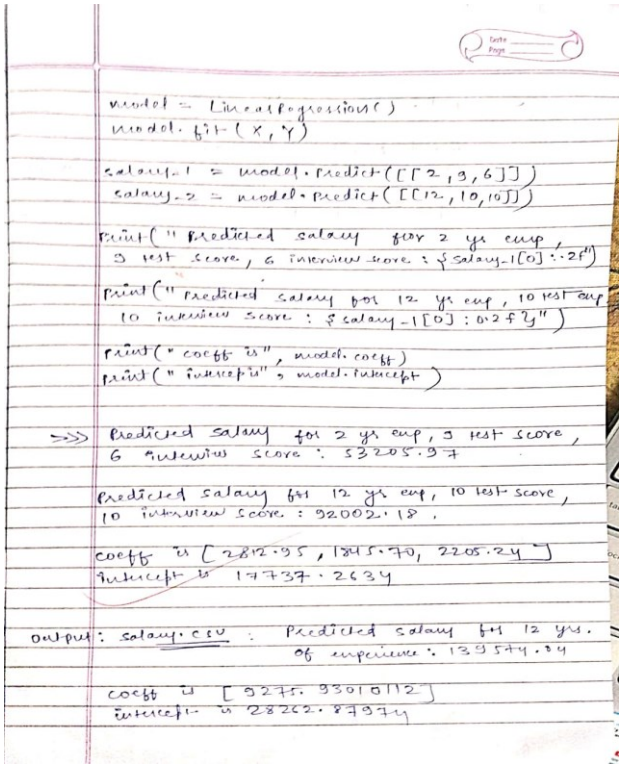
Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

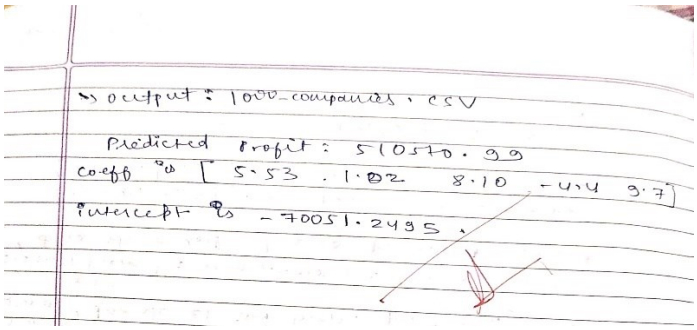
Screenshot











Code

```
import pandas as pd
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

df = pd.read_csv('housing_area_price.csv')
df

# Commented out IPython magic to ensure Python compatibility.
# %matplotlib inline
plt.xlabel('area')
plt.ylabel('price')
plt.scatter(df.area,df.price,color='red',marker='+')

new_df = df.drop('price',axis='columns')
new_df

price = df.price
price

# Create linear regression object
reg = linear_model.LinearRegression()
reg.fit(new_df,price)

"""(1) Predict price of a home with area = 3300 sqr ft"""

reg.predict([[3300]])
```

```
print("coeff is",reg.coef_)

print("intercept is",reg.intercept_)

"""Y = m * X + b (m is coefficient and b is intercept)"""

3300*135.78767123 + 180616.43835616432

"""(1) Predict price of a home with area = 5000 sqr ft"""

reg.predict([[5000]])


# -*- coding: utf-8 -*-
"""Multiple_LR_HomePrice.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1fK78C8TPV44HdvT6lsMhaau2wMtKXquQ
"""

import pandas as pd
import numpy as np
from sklearn import linear_model

df = pd.read_csv('homeprices_Multiple_LR.csv')
df
"""Data Preprocessing: Fill NA values with median value of a column"""

df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
df
reg = linear_model.LinearRegression()
reg.fit(df.drop('price',axis='columns'),df.price)
print("coeff is",reg.coef_)
print("intercept is",reg.intercept_)
```

```
"""Find price of home with 3000 sqr ft area, 3 bedrooms, 40 year old"""

reg.predict([[3000, 3, 40]])

112.06244194*3000 + 23388.88007794*3 + -3231.71790863*40 + 221323.00186540384

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
# Load dataset
df = pd.read_csv('canada_per_capita_income.csv')
# Check dataset structure
print(df.head())

# Renaming columns if needed
df.columns = ["year", "per_capita_income"]

# Reshape data
X = df[['year']]
y = df['per_capita_income']

# Train the model
model = LinearRegression()
model.fit(X, y)

# Predict per capita income for 2020
predicted_income = model.predict([[2020]])
print(f"Predicted per capita income for 2020: {predicted_income[0]:.2f}")

# Plotting results
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, model.predict(X), color='red', label='Regression Line')
plt.scatter([[2020]], predicted_income, color='green', label='Prediction (2020)')
plt.xlabel("Year")
plt.ylabel("Per Capita Income")
plt.title("Canada Per Capita Income Prediction")
plt.legend()
```

```
plt.show()
print("coeff is",model.coef_)
print("intercept is",model.intercept_)



import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Load dataset

df = pd.read_csv('salary.csv')

# Check for missing values
print("Missing values in dataset:\n", df.isnull().sum())

# Drop rows with missing values
df.Salary=df.Salary.fillna(df.Salary.median())
df.YearsExperience=df.YearsExperience.fillna(df.YearsExperience.median())

# Reshape data
X = df[['YearsExperience']]
Y = df['Salary']

# Train the model
model = LinearRegression()
model.fit(X,Y)

# Predict salary for 12 years of experience
predicted_salary = model.predict([[12]])
print(f"Predicted salary for 12 years of experience: {predicted_salary[0]:.2f}")

# Plotting results
plt.scatter(X, y, color='blue', label='Actual Data')
plt.plot(X, model.predict(X), color='red', label='Regression Line')
plt.scatter([[12]], predicted_salary, color='green', label='Prediction (12 years)')
plt.xlabel("Years of Experience")
```

```
plt.ylabel("Salary")
plt.title("Salary Prediction Model")
plt.legend()
plt.show()
print("coeff is",model.coef_)
print("intercept is",model.intercept_)

import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer

# Load dataset
df = pd.read_csv('hiring.csv')

# Rename columns for easier access
df.rename(columns={
    'test_score(out of 10)': 'test_score',
    'interview_score(out of 10)': 'interview_score',
    'salary($)': 'salary'
}, inplace=True)

# Convert experience column from text to numbers
def convert_to_number(word):
    word_to_number = {
        'zero': 0,
        'one': 1,
        'two': 2,
        'three': 3,
        'four': 4,
        'five': 5,
        'six': 6,
        'seven': 7,
        'eight': 8,
        'nine': 9,
        'ten': 10,
        'eleven': 11,
```

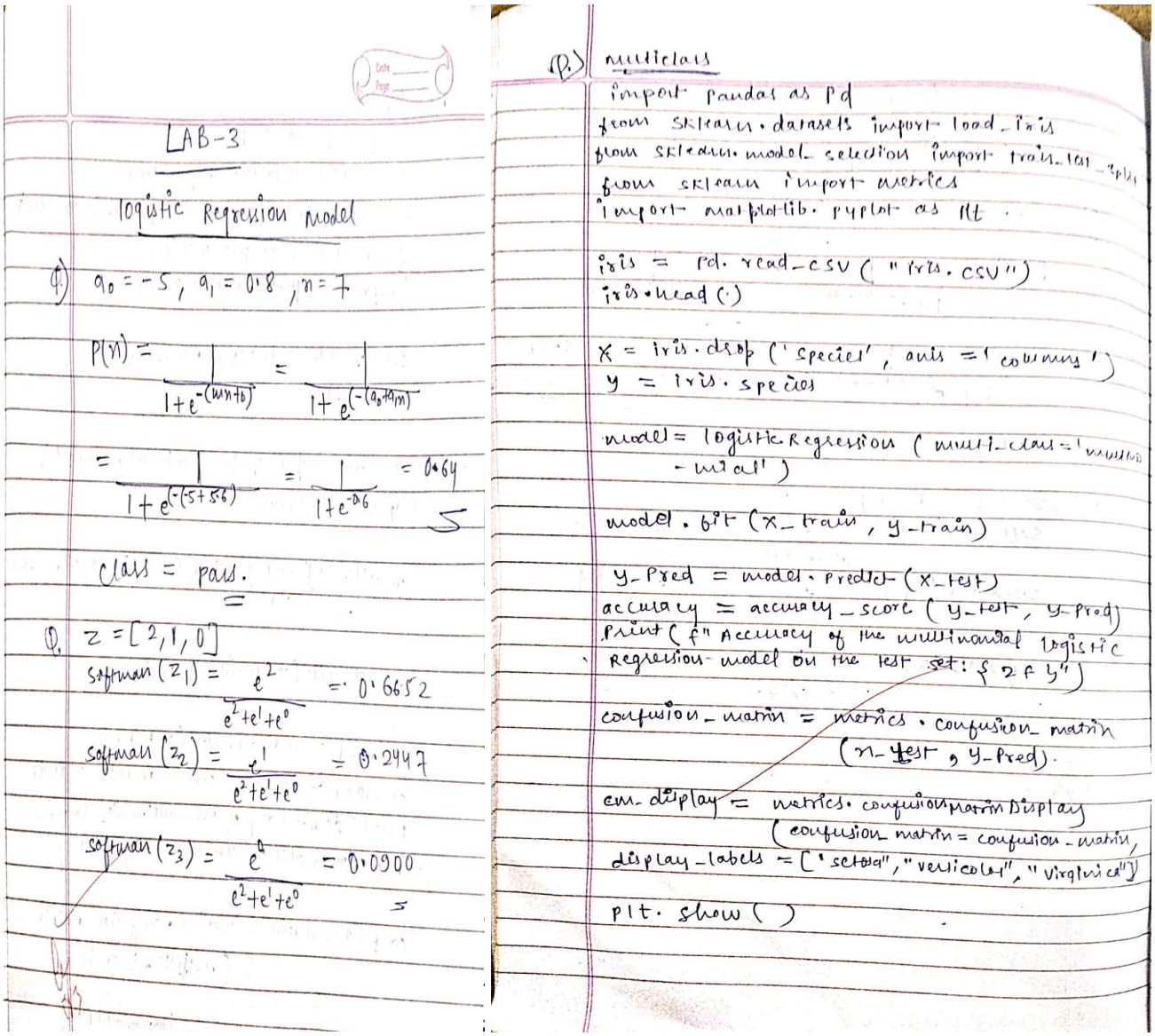
```
        'twelve': 12
    }
    return word_to_number.get(str(word).lower(), 0)

df['experience'] = df['experience'].apply(convert_to_number)

# Handle missing values
df.test_score = df.test_score.fillna(df.test_score.median())
df.experience = df.experience.bfill()

# Prepare data
X = df[['experience', 'test_score', 'interview_score']]
y = df['salary']

# Train model
model = LinearRegression()
model.fit(X, y)

# Predict salaries
salary_1 = model.predict([[2, 9, 6]])
salary_2 = model.predict([[12, 10, 10]])

print(f"Predicted salary for 2 yr exp, 9 test score, 6 interview score: {salary_1[0]:.2f}")
print(f"Predicted salary for 12 yr exp, 10 test score, 10 interview score: {salary_2[0]:.2f}")
print("coeff is",model.coef_)
print("intercept is",model.intercept_)

import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder

# Load dataset
df = pd.read_csv("1000_Companies.csv")

# Select relevant features
X = df[["R&D Spend", "Administration", "Marketing Spend", "State"]]
```

```
y = df["Profit"]

# One-hot encode categorical 'State' column
encoder = OneHotEncoder(drop="first", sparse_output=False)
state_encoded = encoder.fit_transform(X[["State"]])

# Convert encoded data to DataFrame and merge with X
state_encoded_df = pd.DataFrame(state_encoded, columns=encoder.get_feature_names_out(["State"]))
X = X.drop("State", axis=1)
X = pd.concat([X, state_encoded_df], axis=1)

# Train the model
model = LinearRegression()
model.fit(X, y)

# Prepare input data for prediction
input_data = pd.DataFrame([[91694.48, 515841.3, 11931.24, "Florida"]], columns=["R&D Spend",
"Administration", "Marketing Spend", "State"])

# One-hot encode the state in input data
input_state_encoded = encoder.transform(input_data[["State"]])
input_state_df = pd.DataFrame(input_state_encoded, columns=encoder.get_feature_names_out(["State"]))

# Drop state column and merge with input data
input_data = input_data.drop("State", axis=1)
input_data = pd.concat([input_data, input_state_df], axis=1)

# Predict profit
predicted_profit = model.predict(input_data)
print(f"Predicted Profit: {predicted_profit[0]:.2f}")
print("coeff is",model.coef_)
print("intercept is",model.intercept_)
```

Program 5

Build Logistic Regression Model for a given dataset

Screenshot



LAB-3

Logistic Regression model

Q) $a_0 = -5$, $a_1 = 0.8$, $x = 7$

$$P(x) = \frac{1}{1+e^{-(wx+b)}} = \frac{1}{1+e^{(-(a_0+a_1x)}}$$

$$= \frac{1}{1+e^{-(-5+5.6)}} = \frac{1}{1+e^{-0.6}} = 0.64$$

class = pass.

Q) $z = [2, 1, 0]$

$$\text{Softmax}(z_1) = \frac{e^2}{e^2+e^1+e^0} = 0.6652$$

$$\text{Softmax}(z_2) = \frac{e^1}{e^2+e^1+e^0} = 0.2447$$

$$\text{Softmax}(z_3) = \frac{e^0}{e^2+e^1+e^0} = 0.0900$$

Q) multiclass

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn import metrics
import matplotlib.pyplot as plt

iris = pd.read_csv("iris.csv")
iris.head()

X = iris.drop('Species', axis='columns')
y = iris.species

model = LogisticRegression(multi_class='multinomial')

model.fit(X_train, y_train)

y_Pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the multinomial logistic Regression model on the test set: {2f %}")

confusion_matrix = metrics.confusion_matrix(y_test, y_Pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = ['setosa', 'versicolor', 'virginica'])

plt.show()
```

Binary

```
import pandas as pd
from matplotlib import pyplot as plt

df = pd.read_csv("insurance_data.csv")
df.head()

plt.scatter(df.age, df.bought_insurance,
            marker='+', color='red')

from sklearn.model_selection import train_test_split
X_test

model = LogisticRegression()
model.fit(X_train, y_train)
X_test
y_test

y_predicted = model.predict(X_test)
model.score(X_test, y_test)
y_predicted = model.predict([[60]])

def sigmoid(x):
    return 1/(1+math.exp(-x))

def prediction_fun(age):
    z = 0.127 * age - 4.973
    y = sigmoid
    return y.
```

Q 1. (i)
- Satisfaction level
- Number of projects
- Average monthly hours
- Time spent in company
- work accident
- salary level
- department.

(ii) Accuracy = 76.6 %

(2)

(i) Yes, no data preprocessing

(ii) ~~would it missing or inconsistent data~~

(iii) No, data found which were missing or inconsistent.

(iii) very high accuracy.

(iv)
- Reptiles (class 3)
- Fish (class 4)

Code

```
# -*- coding: utf-8 -*-

"""LogisticRegression_Multiclass.ipynb


Automatically generated by Colab.


Original file is located at

    https://colab.research.google.com/drive/1anBybVXILenh0a_R4aM_ZemLrEqYWnJl

"""


# Import necessary libraries

import pandas as pd

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

from sklearn import metrics

import matplotlib.pyplot as plt


# Load the Iris dataset

iris =  pd.read_csv("iris.csv")

iris.head()


X=iris.drop('species',axis='columns')# Features (sepal length, sepal width, petal length, petal width)
```

```
y = iris.species # Target labels (0: Setosa, 1: Versicolor, 2: Virginica)

# Split the dataset into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Multinomial Logistic Regression model
# Use 'multinomial' for multi-class classification and 'lbfgs' solver
model = LogisticRegression(multi_class='multinomial')

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate the accuracy of the model on the test data
accuracy = accuracy_score(y_test, y_pred)

# Display the accuracy
print(f"Accuracy of the Multinomial Logistic Regression model on the test set: {accuracy:.2f}")

confusion_matrix = metrics.confusion_matrix(y_test, y_pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusion_matrix,
display_labels = ["Setosa", "Versicolor", "Virginica"])
```

```
cm_display.plot()
plt.show()


# -*- coding: utf-8 -*-
"""LogisticRegression_Binary.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1M8PXdcmPsrQtqyVXpET3sgghAMr_MCg5
"""

# Commented out IPython magic to ensure Python compatibility.
import pandas as pd
from matplotlib import pyplot as plt
# %matplotlib inline
#"%matplotlib inline" will make your plot outputs appear and be stored within the notebook.

df = pd.read_csv("insurance_data.csv")
df.head()

plt.scatter(df.age,df.bought_insurance,marker='+',color='red')
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(df[['age']],df.bought_insurance,train_size=0.9,random_state=10)

X_train.shape



X_test



from sklearn.linear_model import LogisticRegression

model = LogisticRegression()



model.fit(X_train, y_train)



X_test



y_test



y_predicted = model.predict(X_test)

y_predicted



model.score(X_test,y_test)



model.predict_proba(X_test)



y_predicted = model.predict([[60]])

y_predicted
```

```
#model.coef_ indicates value of m in y=m*x + b equation
model.coef_

#model.intercept_ indicates value of b in y=m*x + b equation
model.intercept_

#Lets defined sigmoid function now and do the math with hand
import math
def sigmoid(x):
  return 1 / (1 + math.exp(-x))

def prediction_function(age):
   z = 0.127 * age - 4.973 # 0.12740563 ~ 0.0127 and -4.97335111 ~ -4.97
   y = sigmoid(z)
   return y

age = 35
prediction_function(age)

"""0.37 is less than 0.5 which means person with 35 will not buy the insurance"""

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

```
import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score


# Load the dataset

file_path = "HR_comma_sep.csv"  # Update the path if necessary

df = pd.read_csv(file_path)


# Exploratory Data Analysis (EDA)

df.info()

print(df.head())


# Plot bar chart for salary vs retention

plt.figure(figsize=(8, 5))

sns.barplot(x="salary", y="left", data=df, estimator=lambda x: (1 - x.mean()), order=["low",
"medium", "high"])

plt.xlabel("Salary Level")

plt.ylabel("Retention Rate")

plt.title("Impact of Salary on Employee Retention")

plt.show()


# Plot bar chart for department vs retention

plt.figure(figsize=(10, 5))
```

```
sns.barplot(x="Department", y="left", data=df, estimator=lambda x: (1 - x.mean()),
order=df["Department"].unique())

plt.xticks(rotation=45)

plt.xlabel("Department")

plt.ylabel("Retention Rate")

plt.title("Correlation Between Department and Employee Retention")

plt.show()


# Encode categorical variables

df_encoded = df.copy()

df_encoded["salary"] = df_encoded["salary"].map({"low": 0, "medium": 1, "high": 2})

df_encoded = pd.get_dummies(df_encoded, columns=["Department"], drop_first=True)


# Select features based on EDA

features = [

    "satisfaction_level",

    "last_evaluation",

    "number_project",

    "average_montly_hours",

    "time_spend_company",

    "Work_accident",

    "promotion_last_5years",

    "salary",

]
```

```
# Include department dummies in features
features += [col for col in df_encoded.columns if col.startswith("Department_")]

# Define X and y
X = df_encoded[features]
y = df_encoded["left"]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Train logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predict and evaluate accuracy
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Logistic Regression Model Accuracy: {accuracy:.4f}")



import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

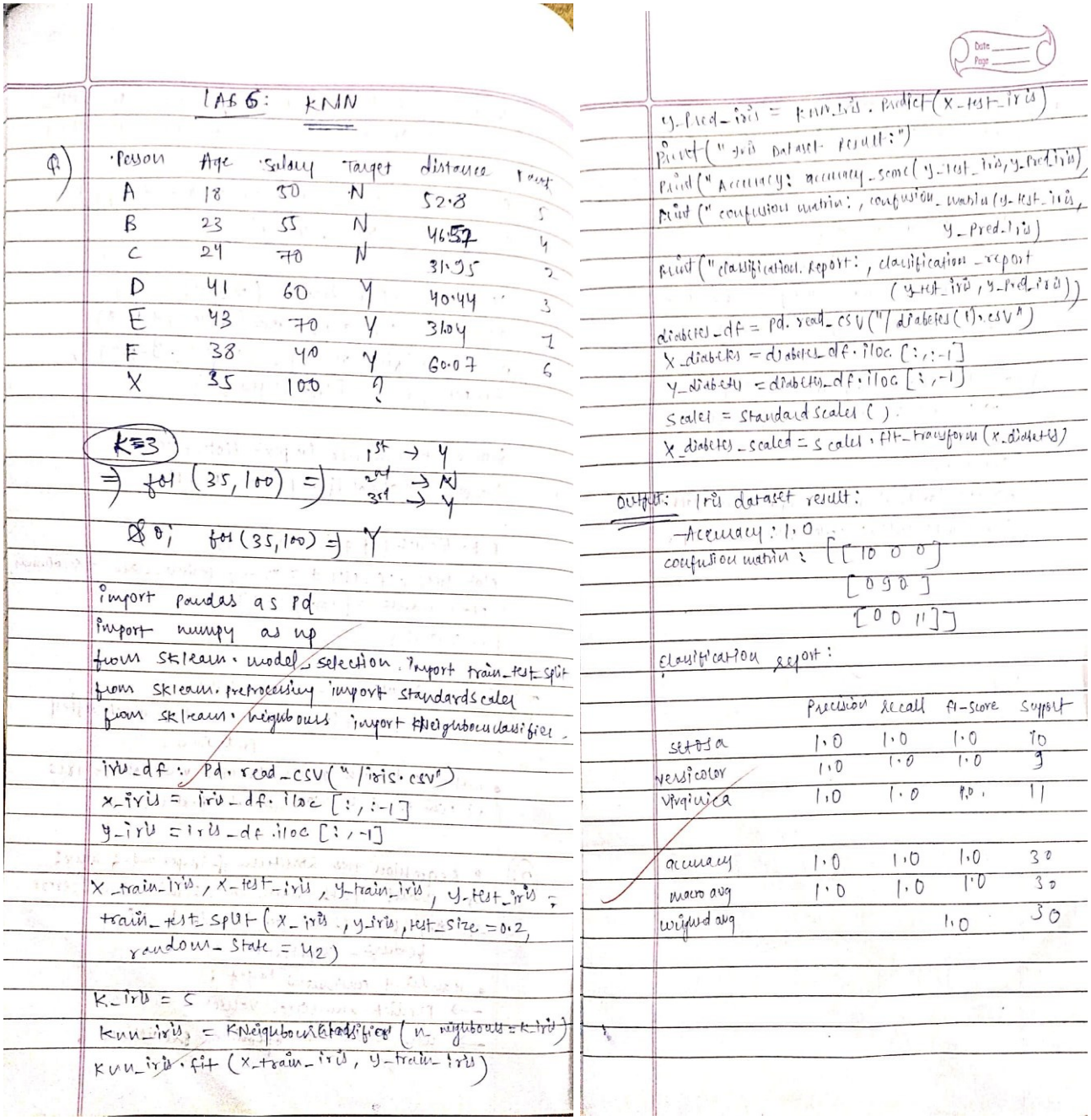
```
import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report


# Load the datasets

zoo_data_path = "zoo-data.csv"

zoo_class_path = "zoo-class-type.csv"


# Read the CSV files

zoo_df = pd.read_csv(zoo_data_path)

class_df = pd.read_csv(zoo_class_path)

# Create a mapping of class numbers to class names

class_mapping = dict(zip(class_df["Class_Number"], class_df["Class_Type"]))


# Drop the 'animal_name' column as it's not useful for modeling

zoo_df = zoo_df.drop(columns=["animal_name"])

# Define features (X) and target (y)

X = zoo_df.drop(columns=["class_type"])

y = zoo_df["class_type"]

# Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Train logistic regression model for multiclass classification

multi_class_model = LogisticRegression(max_iter=1000, multi_class="multinomial",
solver="lbfgs")
```

```
multi_class_model.fit(X_train, y_train)

# Predict on the test set

y_pred = multi_class_model.predict(X_test)

# Calculate accuracy

multi_class_accuracy = accuracy_score(y_test, y_pred)

print(f"Logistic Regression Model Accuracy: {multi_class_accuracy:.4f}\n")

# Compute confusion matrix

conf_matrix = confusion_matrix(y_test, y_pred)

# Map class numbers to class names

class_labels = [class_mapping[i] for i in np.unique(y)]

# Print classification report (includes precision, recall, F1-score)

print("Classification Report:")

print(classification_report(y_test, y_pred, target_names=class_labels))

# Plot the confusion matrix

plt.figure(figsize=(8, 6))

sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", xticklabels=class_labels,
yticklabels=class_labels)

plt.xlabel("Predicted Class")

plt.ylabel("Actual Class")

plt.title("Confusion Matrix for Multiclass Logistic Regression")

plt.xticks(rotation=45)

plt.yticks(rotation=45)

plt.show()
```

Program 6

Build KNN Classification model for a given dataset.

Screenshot



LAB 6: KNN

Q)

Person	Age	Salary	Target	distance	rank
A	18	50	N	52.8	5
B	23	55	N	46.57	4
C	24	70	N	31.95	2
D	41	60	Y	40.44	3
E	43	70	Y	31.04	1
F	38	40	Y	60.07	6
X	35	100	?		

K=3

⇒ for (35,100) ⇒ 1st → Y, 2nd → N, 3rd → Y

⇒ for (35,100) ⇒ Y

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import standardscaler
from sklearn.neighbours import KNeighbourclassifier

iris_df = pd.read_csv("/iris.csv")
x_iris = iris_df.iloc[:,:-1]
y_iris = iris_df.iloc[:,-1]

X_train_iris, X_test_iris, y_train_iris, y_test_iris =
train_test_split(X_iris, y_iris, test_size=0.2,
    random_state = 42)

K_iris = 5
knn_iris = KNeighbourclassifier(n_neighbours=K_iris)
knn_iris.fit(X_train_iris, y_train_iris)
y_Pred_iris = knn_iris.Predict(X_test_iris)
Print("iris Dataset result:")
Print("Accuracy:", accuracy_score(y_test_iris, y_Pred_iris))
Print("confusion matrix:", confusion_matrix(y_test_iris,
                                    y_Pred_iris))
Print("classification report:", classification_report
                    (y_test_iris, y_Pred_iris))
diabetes_df = pd.read_csv("/diabetes(1).csv")
X_diabetes = diabetes_df.iloc[:,:-1]
y_diabetes = diabetes_df.iloc[:,-1]
scalei = standardscaler()
X_diabetes_scaled = scalei.fit_transform(X_diabetes)
```

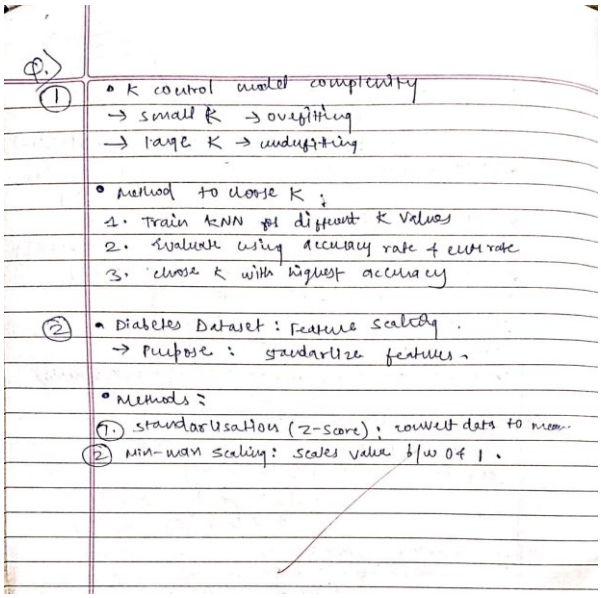
Output: Iris dataset result:

Accuracy: 1.0

confusion matrix: [[10 0 0]
[0 9 0]
[0 0 11]]

classification report:

	Precision	Recall	f1-score	support
setosa	1.0	1.0	1.0	10
versicolor	1.0	1.0	1.0	9
virginica	1.0	1.0	1.0	11
accuracy	1.0	1.0	1.0	30
macro avg	1.0	1.0	1.0	30
weighted avg			1.0	30



Code

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report



# Load Iris dataset

iris_df = pd.read_csv("/iris.csv")

X_iris = iris_df.iloc[:, :-1]  # Features

y_iris = iris_df.iloc[:, -1]   # Target



# Split Iris dataset

X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris, test_size=0.2,
random_state=42)
```

```
# Train KNN model for Iris
k_iris = 5  # Optimal k value can be tuned further
knn_iris = KNeighborsClassifier(n_neighbors=k_iris)
knn_iris.fit(X_train_iris, y_train_iris)

# Predictions for Iris
y_pred_iris = knn_iris.predict(X_test_iris)

# Evaluation for Iris
print("Iris Dataset Results:")
print("Accuracy:", accuracy_score(y_test_iris, y_pred_iris))
print("Confusion Matrix:\n", confusion_matrix(y_test_iris, y_pred_iris))
print("Classification Report:\n", classification_report(y_test_iris, y_pred_iris))

# Load Diabetes dataset
diabetes_df = pd.read_csv("/diabetes (1).csv")
X_diabetes = diabetes_df.iloc[:, :-1]  # Features
y_diabetes = diabetes_df.iloc[:, -1]   # Target

# Feature Scaling for Diabetes dataset
scaler = StandardScaler()
X_diabetes_scaled = scaler.fit_transform(X_diabetes)
```

```
# Split Diabetes dataset

X_train_diabetes, X_test_diabetes, y_train_diabetes, y_test_diabetes =
train_test_split(X_diabetes_scaled, y_diabetes, test_size=0.2, random_state=42)


# Train KNN model for Diabetes

k_diabetes = 7  # Optimal k value can be tuned further

knn_diabetes = KNeighborsClassifier(n_neighbors=k_diabetes)

knn_diabetes.fit(X_train_diabetes, y_train_diabetes)


# Predictions for Diabetes

y_pred_diabetes = knn_diabetes.predict(X_test_diabetes)


# Evaluation for Diabetes

print("\nDiabetes Dataset Results:")

print("Accuracy:", accuracy_score(y_test_diabetes, y_pred_diabetes))

print("Confusion Matrix:\n", confusion_matrix(y_test_diabetes, y_pred_diabetes))


import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

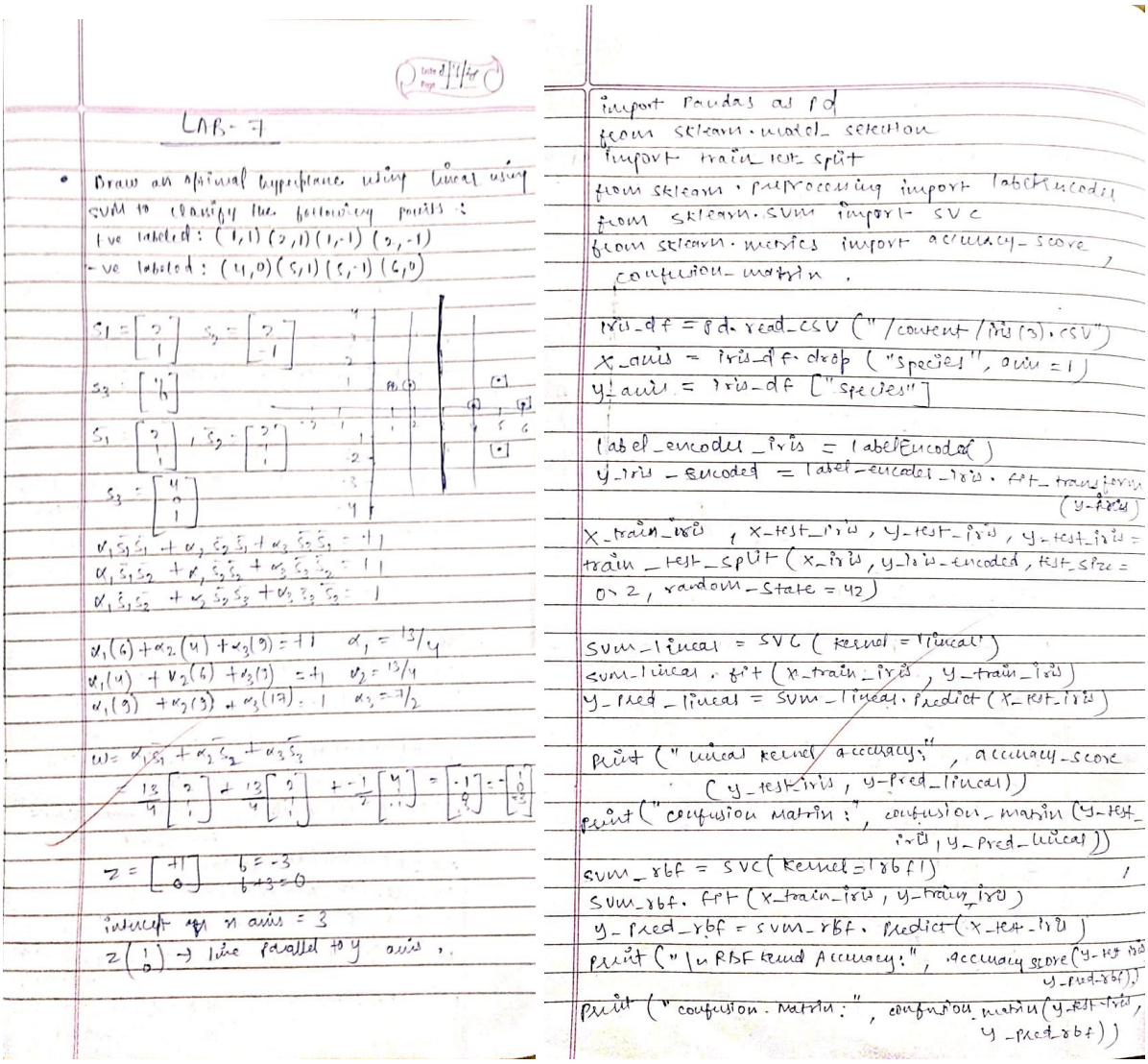
```
# Load dataset

df = pd.read_csv("/heart.csv")


# Separate features and target

X = df.drop(columns=['target'])

y = df['target']


# Normalize features

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Split dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42,
stratify=y)


# Find the best k

best_k = 1

best_score = 0

scores = {}


for k in range(1, 21):

    knn = KNeighborsClassifier(n_neighbors=k)

    knn.fit(X_train, y_train)

    y_pred = knn.predict(X_test)
```

```
    acc = accuracy_score(y_test, y_pred)
    scores[k] = acc
    if acc > best_score:
        best_score = acc
        best_k = k

print(f"Best k: {best_k}, Best Accuracy: {best_score}")

# Train KNN with best k
knn_best = KNeighborsClassifier(n_neighbors=best_k)
knn_best.fit(X_train, y_train)
y_pred_best = knn_best.predict(X_test)
# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_best)
# Plot confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=["No Disease", "Disease"],
yticklabels=["No Disease", "Disease"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
# Print classification report
report = classification_report(y_test, y_pred_best, target_names=["No Disease", "Disease"])
print(report)
```

Program 7

Build Support vector machine model for a given dataset

Screenshot



LAB:- 7

- Draw an optimal hyperplane using linear svm to classify the following points :

+ve labeled : (3,1) (3,-1) (6,1) (6,-1)

-ve labeled : (1,0) (2,1) (2,-1) (3,0)

$s_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ $s_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$ $s_3 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$

$s_1 = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$ $s_2 = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$ $s_3 = \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix}$

$\alpha_1 s_1 s_1 + \alpha_2 s_2 s_1 + \alpha_3 s_3 s_1 = -1$

$\alpha_1 s_1 s_2 + \alpha_2 s_2 s_2 + \alpha_3 s_3 s_2 = 1$

$\alpha_1 s_1 s_3 + \alpha_2 s_2 s_3 + \alpha_3 s_3 s_3 = 1$

$\alpha_1(6) + \alpha_2(4) + \alpha_3(9) = +1 \quad \alpha_1 = 13/4$

$\alpha_1(4) + \alpha_2(6) + \alpha_3(9) = +1 \quad \alpha_2 = 13/4$

$\alpha_1(9) + \alpha_2(9) + \alpha_3(17) = -1 \quad \alpha_3 = -7/2$

$w = \alpha_1 s_1 + \alpha_2 s_2 + \alpha_3 s_3$

$\frac{13}{4}\begin{bmatrix} 3 \\ 1 \end{bmatrix} + \frac{13}{4}\begin{bmatrix} 3 \\ -1 \end{bmatrix} + \frac{-7}{2}\begin{bmatrix} 4 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = -\begin{bmatrix} -1 \\ 0 \end{bmatrix}$

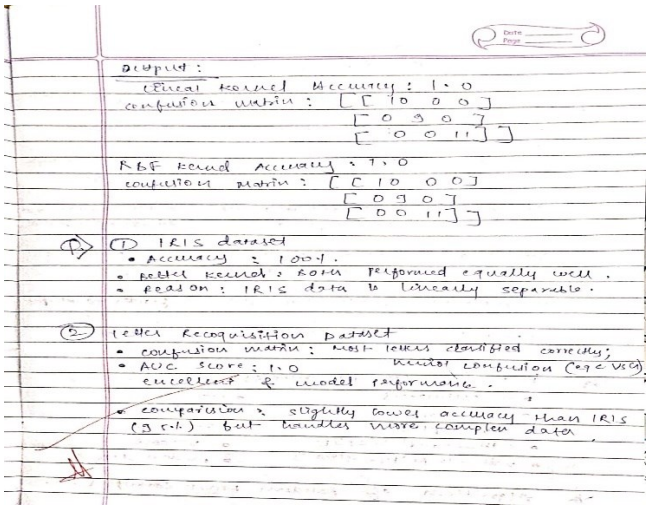
$z = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \quad b = -3$

$-1 + 3 = 0$

intercept on x axis = 3

$z\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ → line parallel to y axis.

```
import pandas as pd
from sklearn.model_selection
import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,
    confusion_matrix.

iris_df = pd.read_csv("/content/Iris(3).csv")
X_axis = iris_df.drop("species", axis=1)
y_axis = iris_df["species"]

label_encoder_iris = LabelEncoder()
y_iris_encoded = label_encoder_iris.fit_transform
                    (y_iris)
X_train_iris, X_test_iris, y_test_iris, y_test_iris =
train_test_split(X_iris, y_iris_encoded, test_size =
0.2, random_state = 42)

svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train_iris, y_train_iris)
y_pred_linear = svm_linear.predict(X_test_iris)

print("Linear Kernel Accuracy:", accuracy_score
        (y_test_iris, y_pred_linear))
print("confusion matrix:", confusion_matrix(y_test_
                    iris, y_pred_linear))
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train_iris, y_train_iris)
y_pred_rbf = svm_rbf.predict(X_test_iris)
print("\n RBF Kernel Accuracy:", accuracy_score(y_test_iris,
                                    y_pred_rbf))
print("confusion matrix:", confusion_matrix(y_test_iris,
                    y_pred_rbf))
```



Code

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, confusion_matrix


# Load IRIS dataset

iris_df = pd.read_csv("iris.csv")


# Prepare features and labels

X_iris = iris_df.drop("species", axis=1)

y_iris = iris_df["species"]


# Encode target labels

label_encoder_iris = LabelEncoder()

y_iris_encoded = label_encoder_iris.fit_transform(y_iris)
```

```
# Split into training and testing data
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(
    X_iris, y_iris_encoded, test_size=0.2, random_state=42)

# Train and evaluate Linear SVM
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train_iris, y_train_iris)
y_pred_linear = svm_linear.predict(X_test_iris)

print("Linear Kernel Accuracy:", accuracy_score(y_test_iris, y_pred_linear))
print("Confusion Matrix:\n", confusion_matrix(y_test_iris, y_pred_linear))

# Train and evaluate RBF SVM
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train_iris, y_train_iris)
y_pred_rbf = svm_rbf.predict(X_test_iris)

print("\nRBF Kernel Accuracy:", accuracy_score(y_test_iris, y_pred_rbf))
print("Confusion Matrix:\n", confusion_matrix(y_test_iris, y_pred_rbf))



import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import LabelEncoder, StandardScaler, label_binarize
from sklearn.svm import SVC
from sklearn.multiclass import OneVsRestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("letter-recognition.csv")

# Assume first column is label, rest are features
X = df.iloc[:, 1:]
y = df.iloc[:, 0]
# Encode labels and binarize for ROC
le = LabelEncoder()
y_encoded = le.fit_transform(y)
y_binarized = label_binarize(y_encoded, classes=range(len(le.classes_)))
# Train-test split
X_train, X_test, y_train, y_test, y_train_bin, y_test_bin = train_test_split(
    X, y_encoded, y_binarized, test_size=0.2, random_state=42)
# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# SVM with RBF kernel in One-vs-Rest strategy
```

```
model = OneVsRestClassifier(SVC(kernel='rbf', probability=True))
model.fit(X_train_scaled, y_train_bin)
# Predictions and probabilities
y_pred_bin = model.predict(X_test_scaled)
y_pred_proba = model.predict_proba(X_test_scaled)
# Evaluation
y_pred_labels = y_pred_bin.argmax(axis=1)
y_test_labels = y_test_bin.argmax(axis=1)
print("Accuracy:", accuracy_score(y_test_labels, y_pred_labels))
print("Confusion Matrix:\n", confusion_matrix(y_test_labels, y_pred_labels))
print("AUC Score:", roc_auc_score(y_test_bin, y_pred_proba, multi_class='ovr'))
# Plot ROC curves for a few classes (to keep plot readable)
plt.figure(figsize=(10, 7))
for i in range(min(5, y_test_bin.shape[1])):  # Plot only first 5 classes
    fpr, tpr, _ = roc_curve(y_test_bin[:, i], y_pred_proba[:, i])
    plt.plot(fpr, tpr, label=f"Class {le.inverse_transform([i])[0]}")
plt.plot([0, 1], [0, 1], 'k--')
plt.title('Multiclass ROC Curve - Letter Recognition')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Program 8
Implement Random forest ensemble method on a given dataset.

Screenshot

LAB-8

Random forest

* Difference b/w Decision tree & Random forest classifier

Feature	Decision tree	Random forest
Definition	A single tree used for classification or regression	An ensemble multiple decision tree.
Accuracy	lower accuracy due to overfitting	Higher accuracy due to averaging results
Overfitting	~~low accuracy~~ More to overfitting	less overfitting
Interpretability	easy to interpret	harder to interpret
Stability	unstable	stable

* ~~Parameters~~ of RandomForest

- n_estimators
- criterion
- max_depth
- min_samples_split
- min_impurity_decrease
- oob_score
- n_jobs
- bootstrap
- verbose
- random_state

* Algorithm of Random Forest classifier.

(1) Input: Dataset D with features X & labels Y, number of trees N.

(2) For each tree (i=1 to N)

- Randomly select a bootstrap sample from the dataset.
- Train a Decision Tree on this sample.
- When splitting nodes:
  - → Instead of considering all features, select
  - a random subset of features.
  - → Choose the best feature to split from this dataset.
- Grow the tree fully or until a stopping criterion is met.

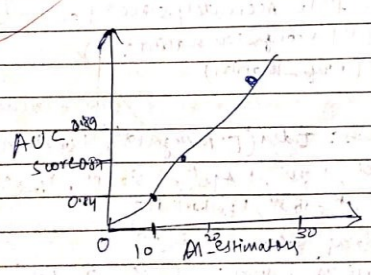
(3) Prediction:

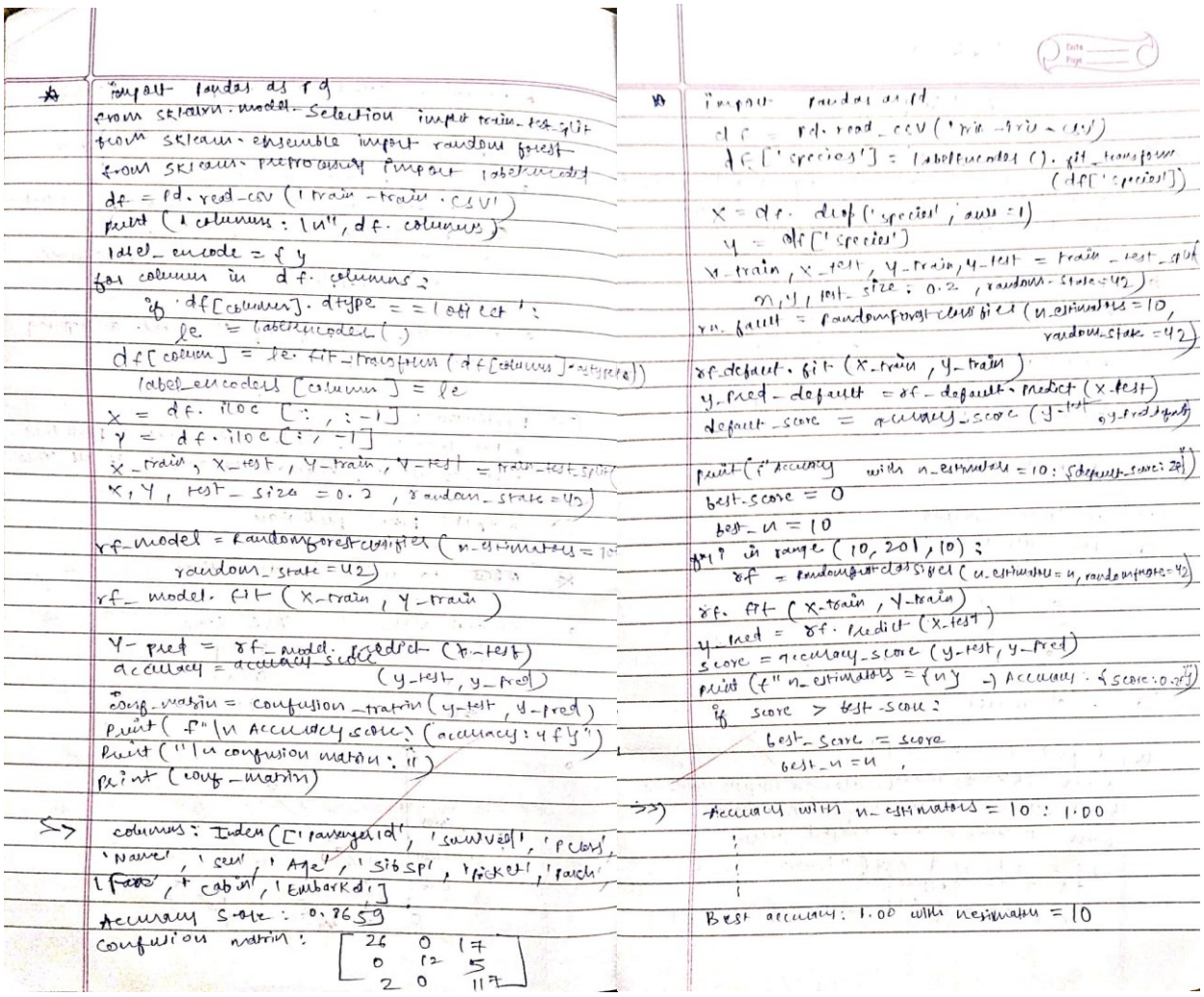
- For classification: Each tree gives a prediction.
- For regression: Average the outputs of all trees

(4) output: Final prediction.

* To write in observation

n_estimators	AUC Score
10	0.84
20	0.87
30	0.89





Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder
# Load the dataset
df = pd.read_csv("train - train.csv")
```

```
# Show column names to identify target (optional)
print("Columns:\n", df.columns)
# Encode all object (categorical) columns to numeric
label_encoders = {}
for column in df.columns:
    if df[column].dtype == 'object':
        le = LabelEncoder()
        df[column] = le.fit_transform(df[column].astype(str))
        label_encoders[column] = le
# Assume last column is target
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
# Train-test split (80-20)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
# Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
# Predictions
y_pred = rf_model.predict(X_test)
# Evaluation
accuracy = accuracy_score(y_test, y_pred)
```

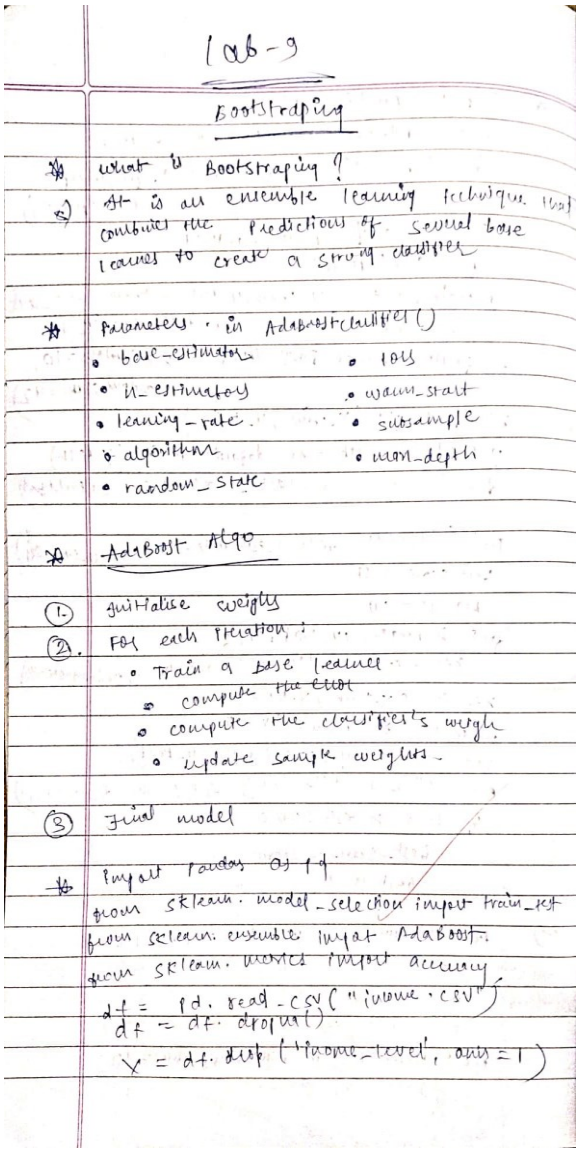
```
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"\nAccuracy Score: {accuracy:.4f}")
print("\nConfusion Matrix:")
print(conf_matrix)



import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder

# Load the dataset
df = pd.read_csv('iris - iris.csv')

# Encode target labels
df['species'] = LabelEncoder().fit_transform(df['species'])

# Split features and target
X = df.drop('species', axis=1)
y = df['species']

# Train-test split (80-20)
```

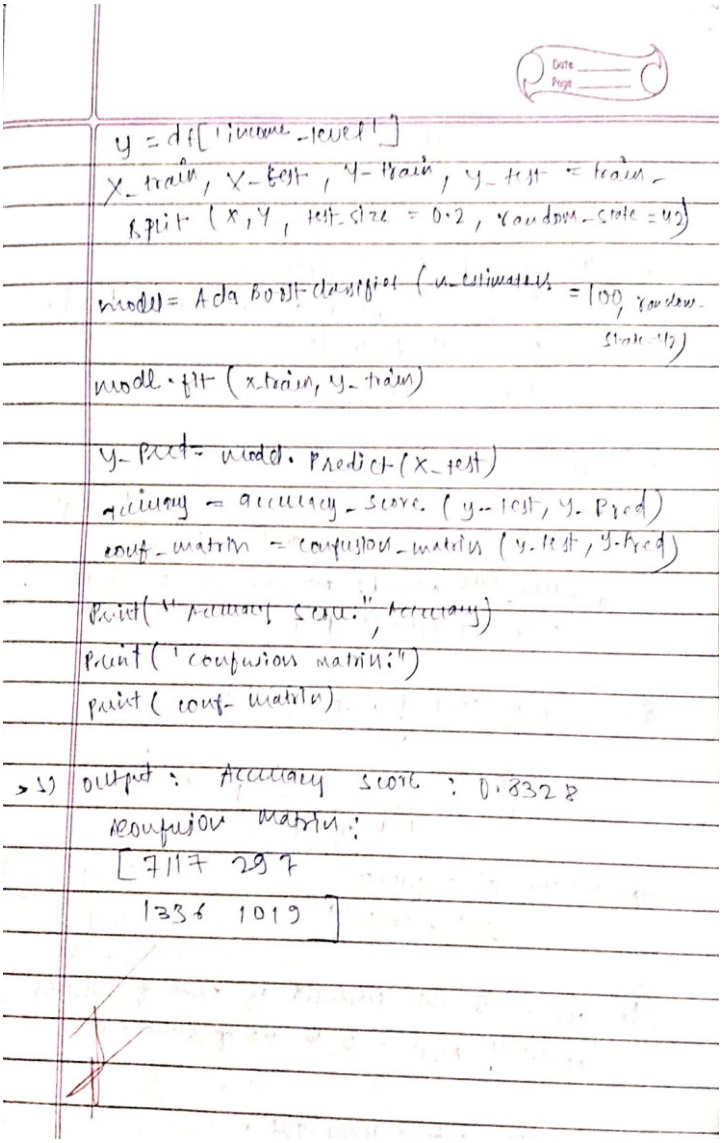
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Default model with n_estimators=10
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)
default_score = accuracy_score(y_test, y_pred_default)

print(f"Accuracy with n_estimators=10: {default_score:.2f}")

# Fine-tuning: Try different values for n_estimators
best_score = 0
best_n = 10
for n in range(10, 201, 10):
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    print(f"n_estimators={n} => Accuracy: {score:.2f}")
    if score > best_score:
        best_score = score
        best_n = n

print(f"\nBest Accuracy: {best_score:.2f} with n_estimators = {best_n}")
```

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot





Code

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the dataset
df = pd.read_csv("income.csv")

# Drop rows with missing values (if any)
df = df.dropna()

# Separate features and target variable
X = df.drop('income_level', axis=1)
y = df['income_level']

# Split the dataset: 80% training, 20% testing
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Initialize and train the AdaBoost classifier
model = AdaBoostClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

```
# Predict on test data
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Display results
print("Accuracy Score:", accuracy)
print("Confusion Matrix:")
print(conf_matrix)



import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
```

```
# Load your dataset
df = pd.read_csv("iris - iris.csv")

# Check column names to identify feature and target
print(df.columns)

# Encode target if needed
if df['species'].dtype == 'object':
    label_encoder = LabelEncoder()
    df['species'] = label_encoder.fit_transform(df['species'])

# Features and target
X = df.drop('species', axis=1)
y = df['species']

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

# Parameters to tune
estimators_range = [10, 50, 100, 150]
learning_rates = [0.01, 0.1, 0.5, 1.0]
results = []
```

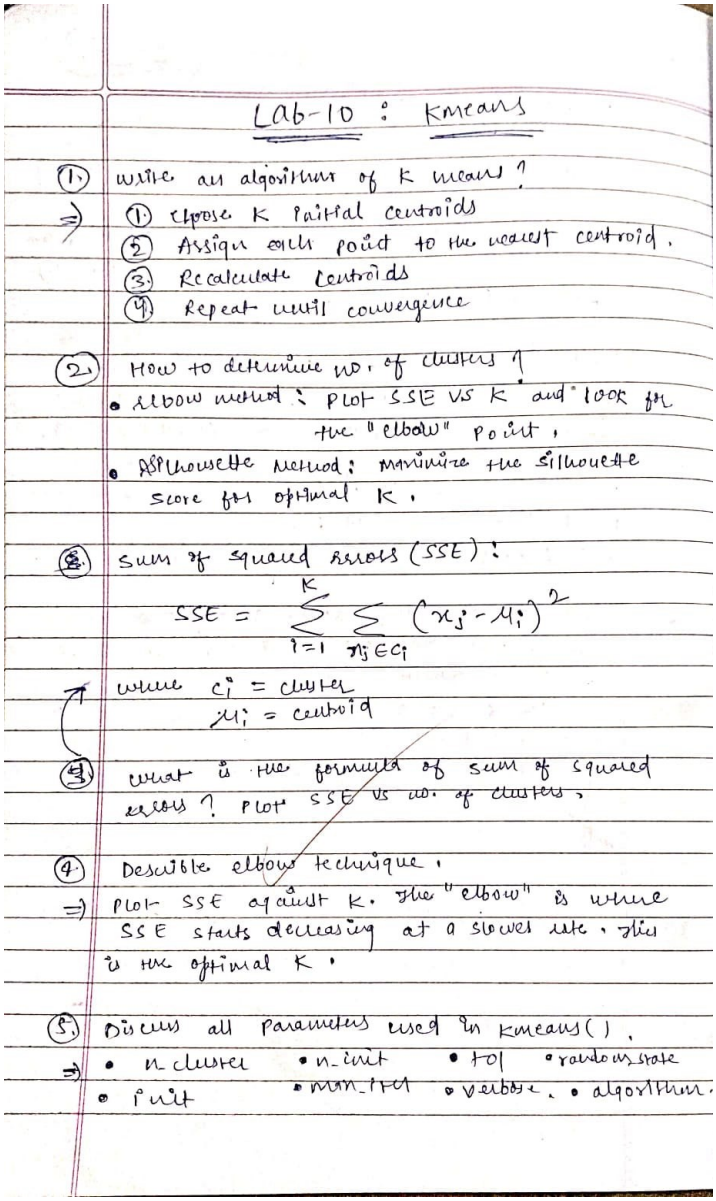
```
# AdaBoost with DecisionTree
for n in estimators_range:
    for lr in learning_rates:
        model = AdaBoostClassifier(
            estimator=DecisionTreeClassifier(max_depth=1),
            n_estimators=n,
            learning_rate=lr,
            random_state=42
        )
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results.append(('Decision Tree', n, lr, acc))

# AdaBoost with LogisticRegression
for n in estimators_range:
    for lr in learning_rates:
        model = AdaBoostClassifier(
            estimator=LogisticRegression(solver='liblinear'),
            n_estimators=n,
            learning_rate=lr,
            random_state=42
        )
        model.fit(X_train, y_train)
```

```
        y_pred = model.predict(X_test)
        acc = accuracy_score(y_test, y_pred)
        results.append(('Logistic Regression', n, lr, acc))

# Display results
df_results = pd.DataFrame(results, columns=['Base Estimator', 'n_estimators', 'learning_rate', 'accuracy'])
print("Top 5 Models by Accuracy:\n")
print(df_results.sort_values(by='accuracy', ascending=False).head())

# Plot
plt.figure(figsize=(12, 6))
sns.lineplot(data=df_results, x='n_estimators', y='accuracy', hue='Base Estimator', style='learning_rate', markers=True)
plt.title("AdaBoost Accuracy on Iris Dataset")
plt.xlabel("Number of Estimators")
plt.ylabel("Accuracy")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Program 10

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

Screenshot



LAb-10 : Kmeans

(1.) write an algorithm of K means ?

⇒ (1) Choose K initial centroids
(2) Assign each point to the nearest centroid.
(3) Recalculate centroids
(4) Repeat until convergence

(2.) How to determine no. of clusters ?

- Elbow method : plot SSE vs K and look for the "elbow" point.
- Silhouette Method : Maximize the Silhouette score for optimal K.

(3.) Sum of squared errors (SSE) :

$$SSE = \sum_{i=1}^{K} \sum_{x_j \in C_i} (x_j - \mu_i)^2$$

where C_i = cluster
μ_i = centroid

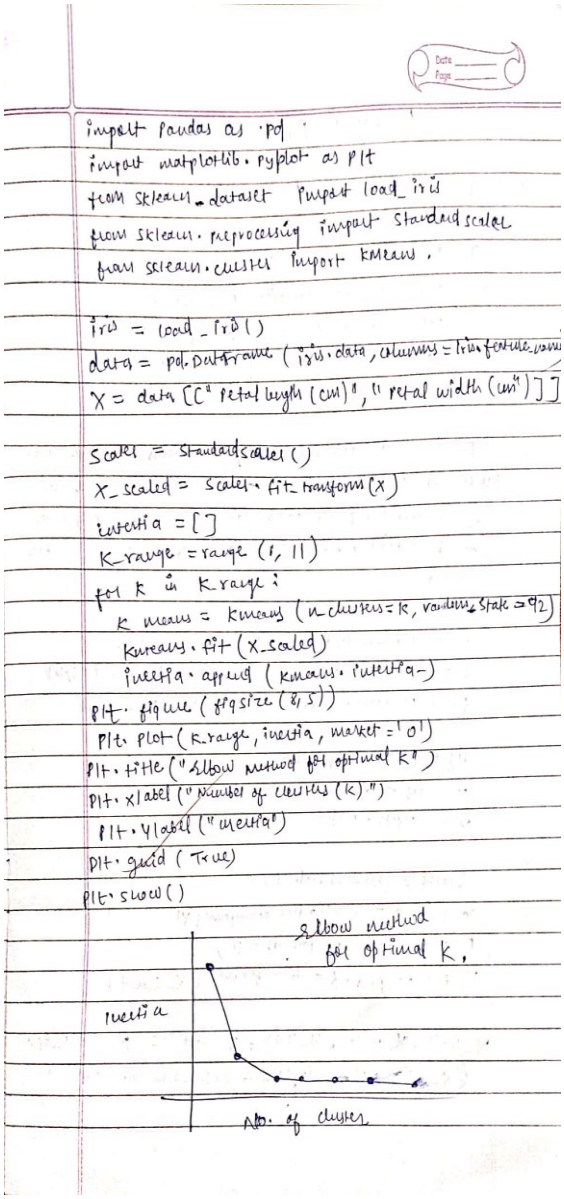
(3.) What is the formula of sum of squared errors ? Plot SSE vs no. of clusters.

(4.) Describe elbow technique.

⇒ Plot SSE against K. The "elbow" is where SSE starts decreasing at a slower rate. This is the optimal K.

(5.) Discuss all parameters used in Kmeans().

⇒ • n_cluster • n_init • tol • random_state
• init • max_iter • verbose • algorithm.



```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.dataset import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans.

iris = load_iris()
data = pd.DataFrame(iris.data, columns = iris.feature_names
X = data["petal length (cm)", "petal width (cm)"]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
inertia = []
K_range = range(1, 11)
for k in K_range:
    K means = Kmeans(n_clusters=k, random_state=42)
    Kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
plt.figure(figsize(8,5))
plt.plot(K_range, inertia, marker='o')
plt.title("Elbow method for optimal K")
plt.xlabel("Number of clusters (k)")
plt.ylabel("inertia")
plt.grid(True)
plt.show()
```

Code

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import adjusted_rand_score
import matplotlib.pyplot as plt
import random

# Step 1: Create income.csv with dummy data
np.random.seed(42)
names = [f"Person_{i}" for i in range(1, 51)]
ages = np.random.randint(20, 60, 50)
incomes = np.random.randint(20000, 100000, 50)

df = pd.DataFrame({
    'Name': names,
    'Age': ages,
    'Income': incomes
})
df.to_csv('income.csv', index=False)
print("✅ 'income.csv' created successfully.")
```

```
# Uncomment these lines if you're using Google Colab and want to download the file
from google.colab import files
files.download('income.csv')

# Step 2: Load data (excluding 'Name')
data = pd.read_csv('income.csv')
X = data[['Age', 'Income']]

# Step 3: Train-test split (80% train, 20% test)
X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)

# Step 4: Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Step 5: Plot SSE vs number of clusters (Elbow method)
sse = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_train_scaled)
    sse.append(kmeans.inertia_)
```

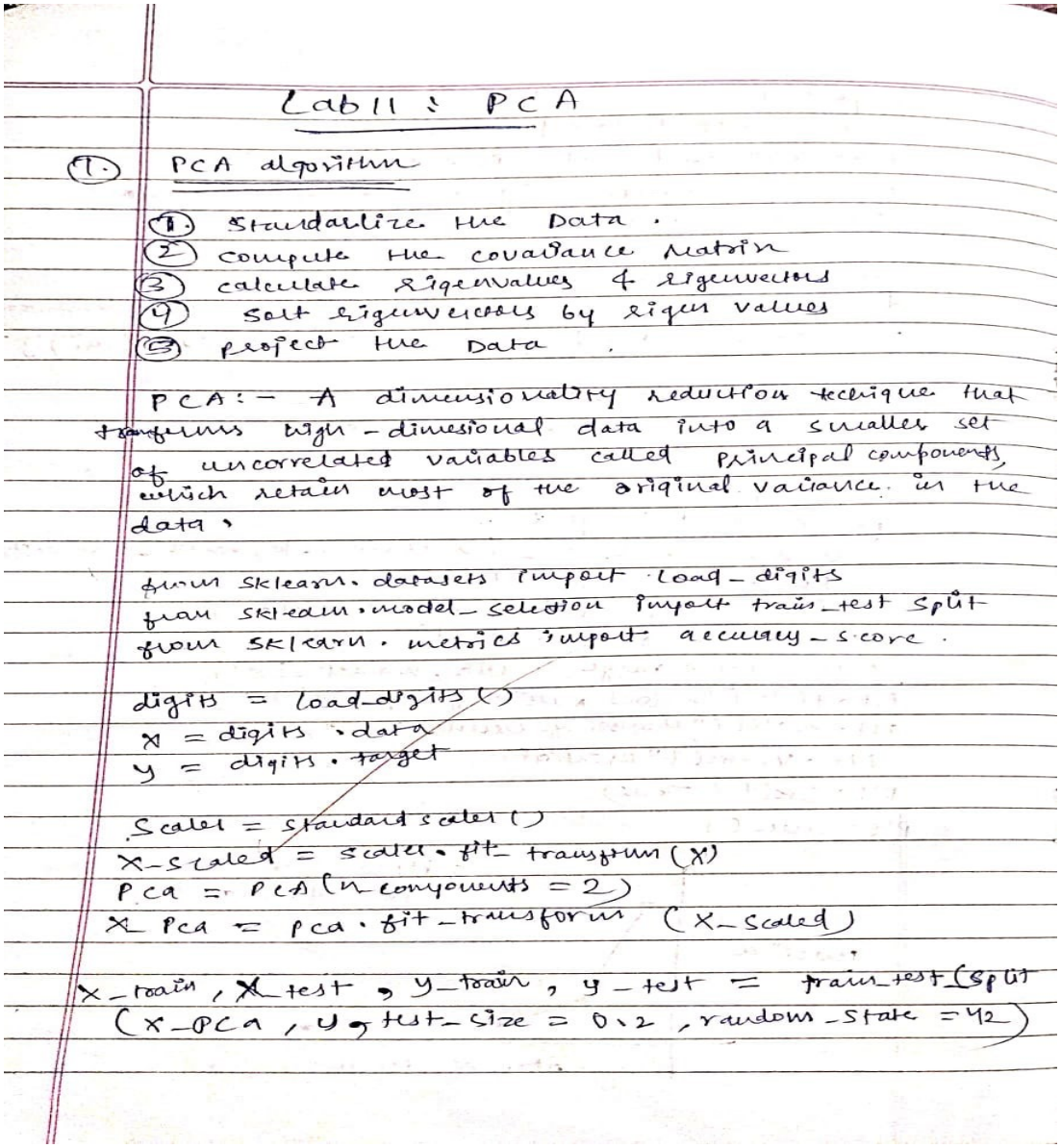
```
plt.plot(k_range, sse, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('SSE (Inertia)')
plt.title('Elbow Method: SSE vs Number of Clusters')
plt.grid(True)
plt.show()

# Step 6: Fit KMeans model with chosen k
k = 3  # Change this based on the elbow plot
model = KMeans(n_clusters=k, random_state=42)
model.fit(X_train_scaled)

# Step 7: Predict clusters
train_preds = model.predict(X_train_scaled)
test_preds = model.predict(X_test_scaled)
# Step 8: Evaluate with Adjusted Rand Index (simulated labels)
true_labels_train = [random.randint(0, k-1) for _ in range(len(train_preds))]
accuracy = adjusted_rand_score(true_labels_train, train_preds)
print("🔍 Adjusted Rand Index (proxy accuracy):", round(accuracy, 2))
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
```

```
# Step 1: Load the Iris dataset and select petal length and width
iris = load_iris()
data = pd.DataFrame(iris.data, columns=iris.feature_names)
X = data[["petal length (cm)", "petal width (cm)"]]
# Step 2: Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Elbow Method to find optimal k
inertia = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
# Plot the elbow graph
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, marker='o')
plt.title("Elbow Method for Optimal k")
plt.xlabel("Number of clusters (k)")
plt.ylabel("Inertia")
plt.grid(True)
plt.show()
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method.

Screenshot



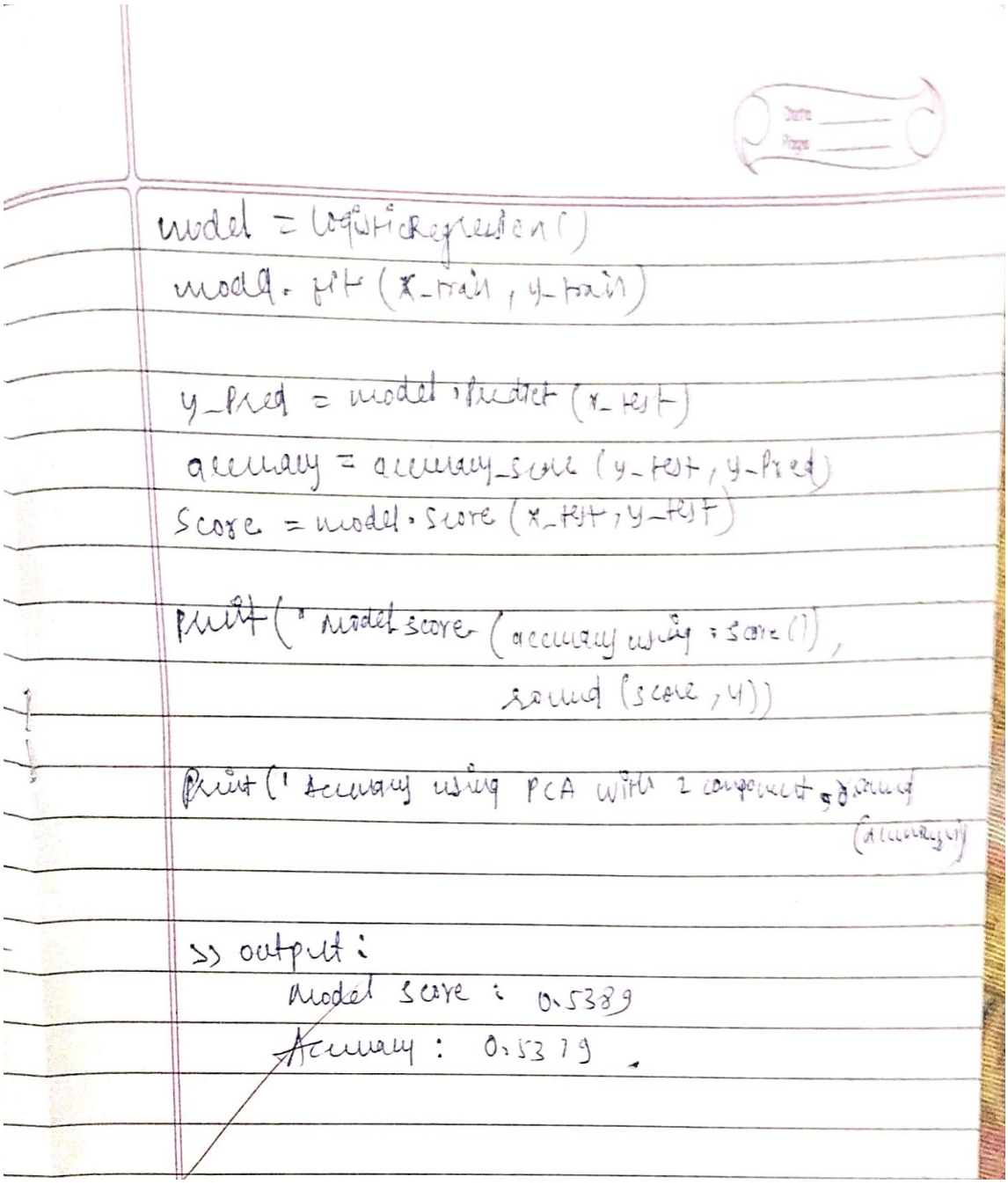
Lab 11 : PCA

1. PCA algorithm

1. Standardize the Data.
2. Compute the covariance Matrix
3. calculate Eigenvalues & Eigenvectors
4. Sort Eigenvectors by Eigen values
5. Project the Data.

PCA :- A dimensionality reduction technique that transforms high-dimensional data into a smaller set of uncorrelated variables called Principal components, which retain most of the original variance in the data.

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score.

digits = load_digits()
X = digits.data
y = digits.target

Scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
Pca = PCA(n_components = 2)
X_Pca = pca.fit_transform (X_scaled)

X_train, X_test, y_train, y_test = train_test_split
(X_PCA, y, test_size = 0.2, random_state = 42)
```



Code

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
# Step 1: Load the digits dataset
digits = load_digits()
X = digits.data
y = digits.target

# Step 2: Scale the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 3: Apply PCA (reduce to 2 components)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Step 4: Split into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X_pca, y, test_size=0.2, random_state=42
)

# Step 5: Train Logistic Regression on PCA-transformed data
model = LogisticRegression()
model.fit(X_train, y_train)

# Step 6: Predict and evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
score = model.score(X_test, y_test)
print("✅ Model Score (accuracy using .score()):", round(score, 4))

print("✅ Accuracy using PCA with 2 components:", round(accuracy, 4))


import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from scipy.stats import zscore

# Step 1: Load dataset
df = pd.read_csv("/content/heart.csv")  # Adjust path if needed

# Step 2: Remove outliers using Z-score
```

```
z_scores = np.abs(zscore(df.select_dtypes(include=[np.number])))
df = df[(z_scores < 3).all(axis=1)]

# Step 3: Convert text columns to numbers
df_encoded = df.copy()
for col in df_encoded.select_dtypes(include=["object"]).columns:
    if df_encoded[col].nunique() <= 2:
        le = LabelEncoder()
        df_encoded[col] = le.fit_transform(df_encoded[col])
    else:
        df_encoded = pd.get_dummies(df_encoded, columns=[col], drop_first=True)

# Step 4: Apply scaling
X = df_encoded.drop("target", axis=1)  # Replace 'target' if it's named differently
y = df_encoded["target"]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 5: Train/test split and model building
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC()
}

print("Model Accuracies (without PCA):")
for name, model in models.items():
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    acc = accuracy_score(y_test, preds)
    print(f"{name}: {acc:.4f}")

# Step 6: Apply PCA and re-evaluate
pca = PCA(n_components=0.95)  # Retain 95% variance
X_pca = pca.fit_transform(X_scaled)
X_train_pca, X_test_pca, _, _ = train_test_split(X_pca, y, test_size=0.2, random_state=42)

print("\nModel Accuracies (with PCA):")
for name, model in models.items():
    model.fit(X_train_pca, y_train)
    preds = model.predict(X_test_pca)
    acc = accuracy_score(y_test, preds)
    print(f"{name}: {acc:.4f}")
```