VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

S Gajanana Nayak (1BM22CS227)

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
Feb-2025 to Jun-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 **S Gajanana Nayak (1BM22CS227),** 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

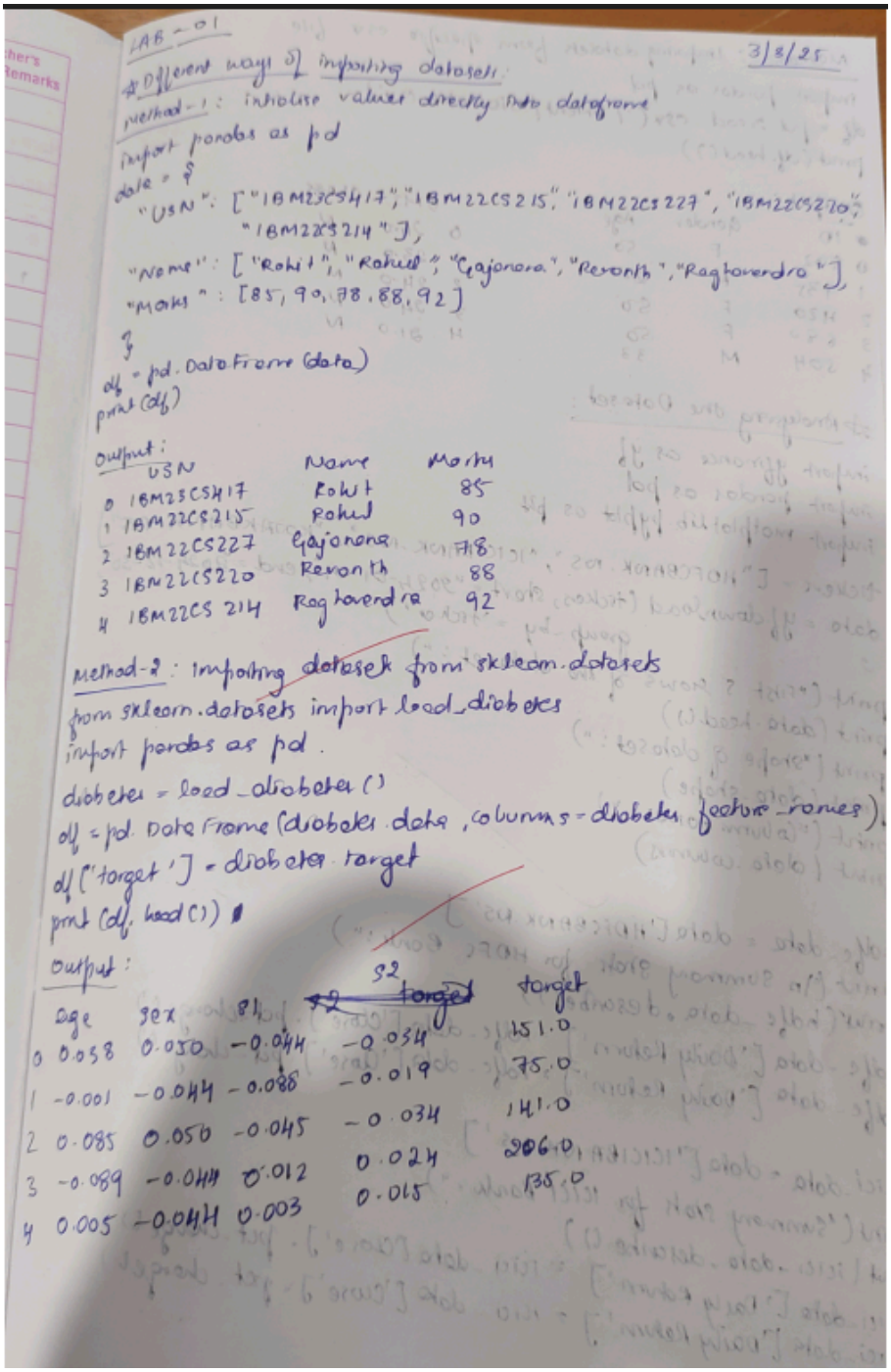
Index

Github Link:
https://github.com/Gajanana227/ML

Program 1

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

Screenshot:



Method 3: Importing datasets from specific .csv file

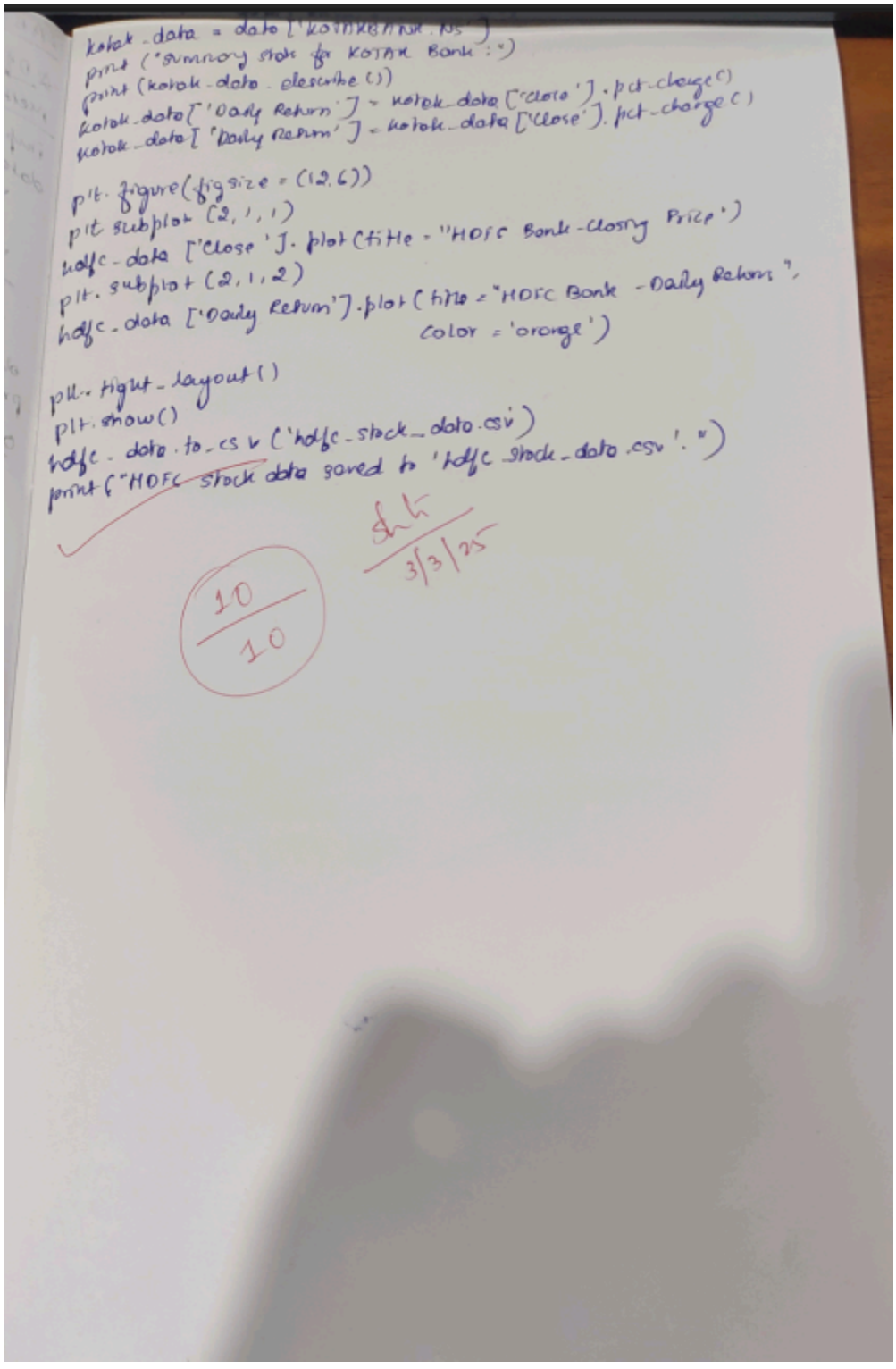
```
import pandas as pd
df = pd.read_csv('/content/diabetes.csv')
print(df.head())
```

Output:

	ID	Gender	Age	BMI	CLASS
0	502	F	50	24.0	N
1	735	M	26	23.0	N
2	420	F	50	24.0	N
3	680	F	50	24.0	N
4	504	M	33	21.0	N

Analysing the Dataset:

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-30",
                   group_by='ticker')
print("First 5 rows of the dataset:")
print(data.head())
print("Shape of dataset:")
print(data.shape)
print("Column names:")
print(data.columns)

hdfc_data = data['HDFCBANK.NS']
print("\n Summary stats for HDFC Bank:")
print(hdfc_data.describe())
hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()
hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()

icici_data = data['ICICIBANK.NS']
print("Summary stats for ICICI Bank:")
print(icici_data.describe())
icici_data['Daily Return'] = icici_data['Close'].pct_change()
icici_data['Daily Return'] = icici_data['Close'].pct_change()
```



```
kotak_data = data['KOTAKBANK.NS']
print("Summary stats for KOTAK Bank:")
print(kotak_data.describe())
kotak_data['Daily Return'] = kotak_data['Close'].pct_change()
kotak_data['Daily Return'] = kotak_data['Close'].pct_change()

plt.figure(figsize=(12,6))
plt.subplot(2,1,1)
hdfc_data['Close'].plot(title="HDFC Bank - Closing Price")
plt.subplot(2,1,2)
hdfc_data['Daily Return'].plot(title="HDFC Bank - Daily Return",
                               color='orange')

plt.tight_layout()
plt.show()
hdfc_data.to_csv('hdfc_stock_data.csv')
print("HDFC stock data saved to 'hdfc_stock_data.csv'.")
```

10/10

Sk
3/3/25

Code:

```
import pandas as pd
data = {
    "USN": ["1BM23CS417", "1BM22CS215", "1BM22CS227", "1BM22CS220", "1BM22CS214"],
    "Name": ["Rohit", "Rahul", "Gajanana", "Revanth", "Raghavendra"],
    "Marks": [85, 90, 78, 88, 92]
}
df = pd.DataFrame(data)
print(df)


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


import pandas as pd
df = pd.read_csv('/content/Diabetes .csv')
print(df.head())
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
```

```
data = yf.download(tickers, start="2024-01-01", end="2024-12-30",
group_by='ticker')
print("First 5 rows of the dataset:")
print(data.head())
print("\nShape of the dataset:")
print(data.shape)
print("\nColumn names:")
print(data.columns)

hdfc_data = data['HDFCBANK.NS']
print("\nSummary statistics for HDFC Bank:")
print(hdfc_data.describe())
hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()
hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()
icici_data = data['ICICIBANK.NS']
print("\nSummary statistics for ICICI Bank:")
print(icici_data.describe())
icici_data['Daily Return'] = icici_data['Close'].pct_change()
icici_data['Daily Return'] = icici_data['Close'].pct_change()
kotak_data = data['KOTAKBANK.NS']
print("\nSummary statistics for KOTAK Bank:")
print(kotak_data.describe())
kotak_data['Daily Return'] = kotak_data['Close'].pct_change()
kotak_data['Daily Return'] = kotak_data['Close'].pct_change()
```

```
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
hdfc_data['Close'].plot(title="HDFC Bank - Closing Price")
plt.subplot(2, 1, 2)
hdfc_data['Daily Return'].plot(title="HDFC Bank - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
hdfc_data.to_csv('hdfc_stock_data.csv')
print("HDFC stock data saved to 'hdfc_stock_data.csv'.")

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
icici_data['Close'].plot(title="ICICI Bank - Closing Price")
plt.subplot(2, 1, 2)
icici_data['Daily Return'].plot(title="ICICI Bank - Daily Returns", color='red')
plt.tight_layout()
plt.show()
icici_data.to_csv('icici_stock_data.csv')
print("ICICI stock data saved to 'icici_stock_data.csv'.")

plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
kotak_data['Close'].plot(title="Kotak Bank - Closing Price")
```

```
plt.subplot(2, 1, 2)
kotak_data['Daily Return'].plot(title="Kotak Bank - Daily Returns", color='green')
plt.tight_layout()
plt.show()
kotak_data.to_csv('kotak_stock_data.csv')
print("Kotak stock data saved to 'kotak_stock_data.csv'.")
```

Output:

```
          USN         Name  Marks
0  1BM23CS417        Rohit     85
1  1BM22CS215        Rahul     90
2  1BM22CS227     Gajanana     78
3  1BM22CS220      Revanth     88
4  1BM22CS214  Raghavendra     92
```

```
        age       sex       bmi        bp        s1        s2        s3  \
0  0.038076  0.050680  0.061696  0.021872 -0.044223 -0.034821 -0.043401
1 -0.001882 -0.044642 -0.051474 -0.026328 -0.008449 -0.019163  0.074412
2  0.085299  0.050680  0.044451 -0.005670 -0.045599 -0.034194 -0.032356
3 -0.089063 -0.044642 -0.011595 -0.036656  0.012191  0.024991 -0.036038
4  0.005383 -0.044642 -0.036385  0.021872  0.003935  0.015596  0.008142

         s4        s5        s6  target
0 -0.002592  0.019907 -0.017646   151.0
1 -0.039493 -0.068332 -0.092204    75.0
2 -0.002592  0.002861 -0.025930   141.0
3  0.034309  0.022688 -0.009362   206.0
4 -0.002592 -0.031988 -0.046641   135.0
```

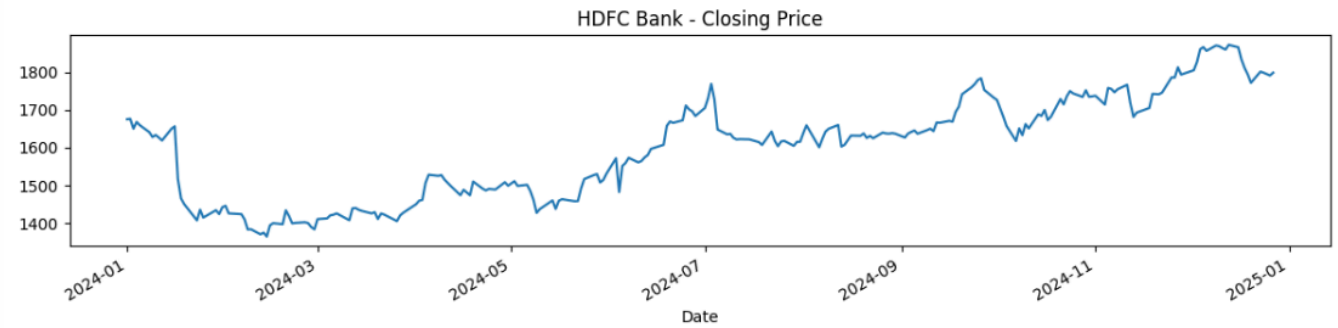
```
    ID  No_Pation Gender  AGE  Urea  Cr  HbA1c  Chol   TG  HDL  LDL  VLDL  \
0  502      17975      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
1  735      34221      M   26   4.5  62    4.9   3.7  1.4  1.1  2.1   0.6
2  420      47975      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
3  680      87656      F   50   4.7  46    4.9   4.2  0.9  2.4  1.4   0.5
4  504      34223      M   33   7.1  46    4.9   4.9  1.0  0.8  2.0   0.4

    BMI CLASS
0  24.0     N
1  23.0     N
2  24.0     N
3  24.0     N
4  21.0     N
```

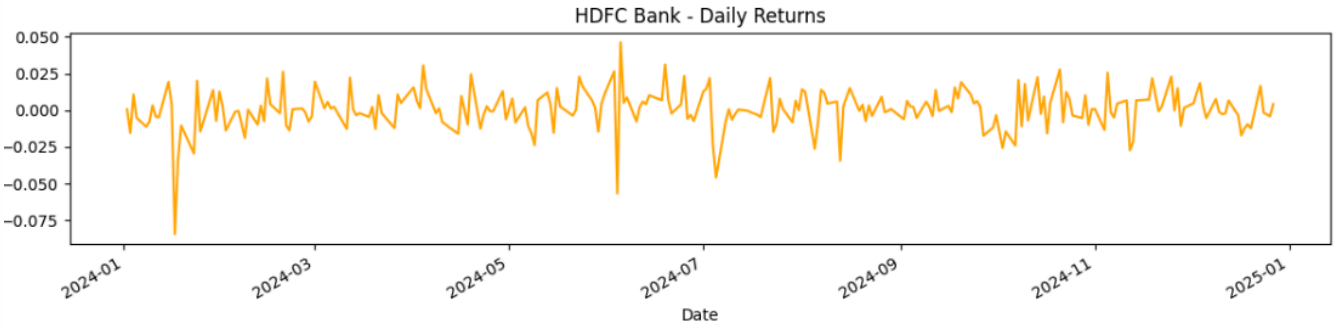
```
First 5 rows of the dataset:
Ticker      HDFCBANK.NS                                                \
Price             Open         High          Low        Close    Volume
Date
2024-01-01  1683.017598  1686.125187  1669.206199  1675.223999   7119843
2024-01-02  1675.914685  1679.860799  1665.950651  1676.210571  14621046
2024-01-03  1679.071480  1681.735059  1646.466666  1650.363525  14194881
2024-01-04  1655.394910  1672.116520  1648.193203  1668.071777  13367028
2024-01-05  1664.421596  1681.932477  1645.628180  1659.538208  15944735

Ticker     KOTAKBANK.NS                                                \
Price             Open         High          Low        Close   Volume
Date
2024-01-01  1906.909954  1916.899006  1891.027338  1907.059814  1425902
2024-01-02  1905.911108  1905.911108  1858.063525  1863.008179  5120796
2024-01-03  1861.959234  1867.952665  1845.627158  1863.857178  3781515
2024-01-04  1869.451068  1869.451068  1858.513105  1861.559692  2865766
2024-01-05  1863.457575  1867.852782  1839.383985  1845.577148  7799341

Ticker     ICICIBANK.NS
Price             Open        High         Low       Close    Volume
Date
2024-01-01   983.086778  996.273246  982.541485  990.869812   7683792
2024-01-02   988.490253  989.134730  971.883221  973.866150  16263825
2024-01-03   976.295294  979.567116  966.777197  975.650818  16826752
2024-01-04   977.980767  980.707295  973.519176  978.724365  22789140
2024-01-05   979.567084  989.779158  975.402920  985.218445  14875499


Shape of the dataset:
(244, 15)

Column names:
MultiIndex([( 'HDFCBANK.NS',   'Open'),
            ( 'HDFCBANK.NS',   'High'),
            ( 'HDFCBANK.NS',    'Low'),
            ( 'HDFCBANK.NS',  'Close'),
            ( 'HDFCBANK.NS', 'Volume'),
            ('KOTAKBANK.NS',   'Open'),
            ('KOTAKBANK.NS',   'High'),
            ('KOTAKBANK.NS',    'Low'),
            ('KOTAKBANK.NS',  'Close'),
            ('KOTAKBANK.NS', 'Volume'),
            ('ICICIBANK.NS',   'Open'),
            ('ICICIBANK.NS',   'High'),
            ('ICICIBANK.NS',    'Low'),
            ('ICICIBANK.NS',  'Close'),
            ('ICICIBANK.NS', 'Volume')],
           names=['Ticker', 'Price'])
```

```
Summary statistics for HDFC Bank:
Price         Open         High          Low        Close        Volume
count   244.000000   244.000000   244.000000   244.000000  2.440000e+02
mean   1601.375295  1615.443664  1588.221245  1601.898968  2.119658e+07
std     134.648125   134.183203   132.796819   133.748372  2.133860e+07
min    1357.463183  1372.754374  1345.180951  1365.404785  8.798460e+05
25%    1475.316358  1494.072805  1460.259509  1474.564087  1.274850e+07
50%    1627.724976  1638.350037  1616.000000  1625.950012  1.686810e+07
75%    1696.474976  1711.425018  1679.250000  1697.062531  2.295014e+07
max    1877.699951  1880.000000  1858.550049  1871.750000  2.226710e+08
```

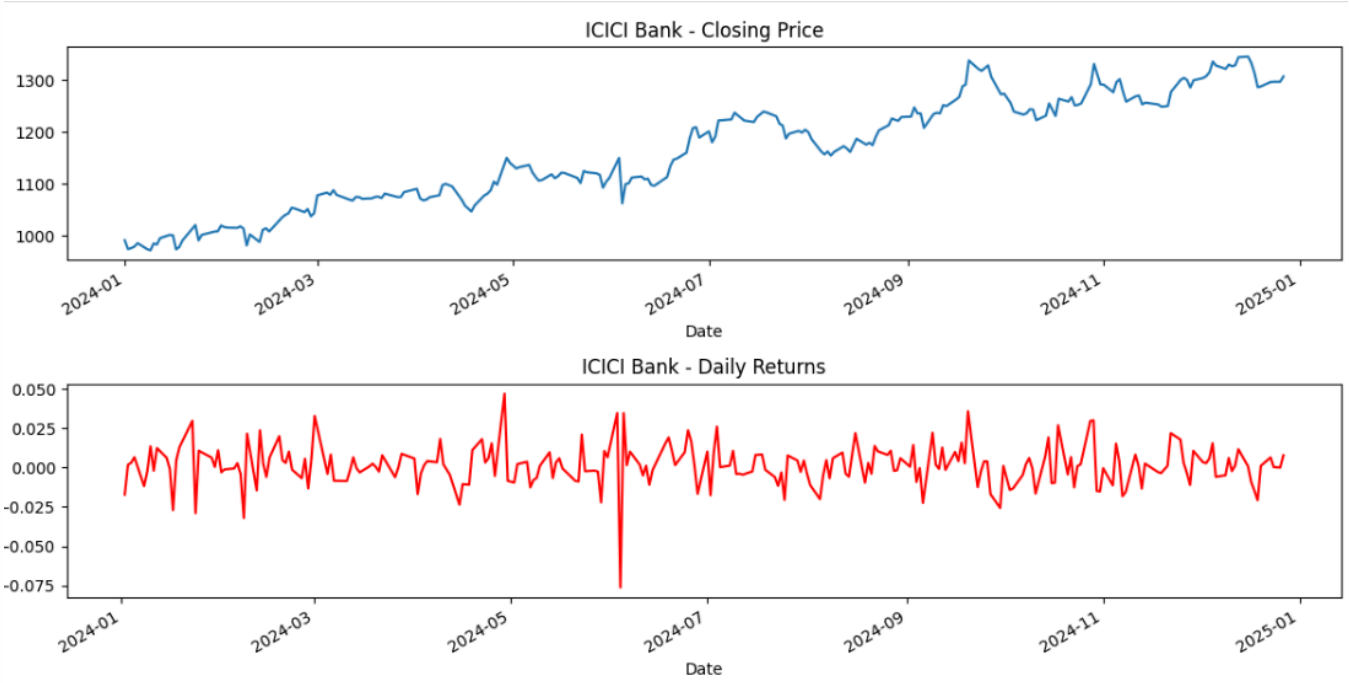
```
Summary statistics for ICICI Bank:
Price         Open         High          Low        Close        Volume
count   244.000000   244.000000   244.000000   244.000000  2.440000e+02
mean   1161.723560  1173.687900  1151.318979  1162.751791  1.539172e+07
std     104.905646   105.668229   105.083015   105.520481  9.503609e+06
min     965.637027   979.567116   961.869473   971.387512  1.007022e+06
25%    1073.818215  1085.368782  1067.386038  1075.107086  1.014533e+07
50%    1169.443635  1178.450012  1157.361521  1165.470703  1.291768e+07
75%    1248.512512  1261.399994  1236.649963  1250.812531  1.755770e+07
max    1344.900024  1362.349976  1340.050049  1346.099976  7.325777e+07
```

```
Summary statistics for KOTAK Bank:
Price         Open         High          Low        Close        Volume
count   244.000000   244.000000   244.000000   244.000000  2.440000e+02
mean   1771.245907  1787.548029  1754.395105  1770.792347  5.736598e+06
std      62.189675    61.978802    62.765980    62.594747  5.388927e+06
min    1581.266899  1586.161558  1542.159736  1545.006592  1.824890e+05
25%    1733.974927  1754.131905  1719.028421  1736.297058  3.300380e+06
50%    1769.500000  1789.450012  1758.099976  1773.681030  4.307680e+06
75%    1809.925018  1826.998164  1789.912506  1808.155670  6.159475e+06
max    1935.000000  1942.000000  1909.599976  1934.699951  6.617908e+07
```

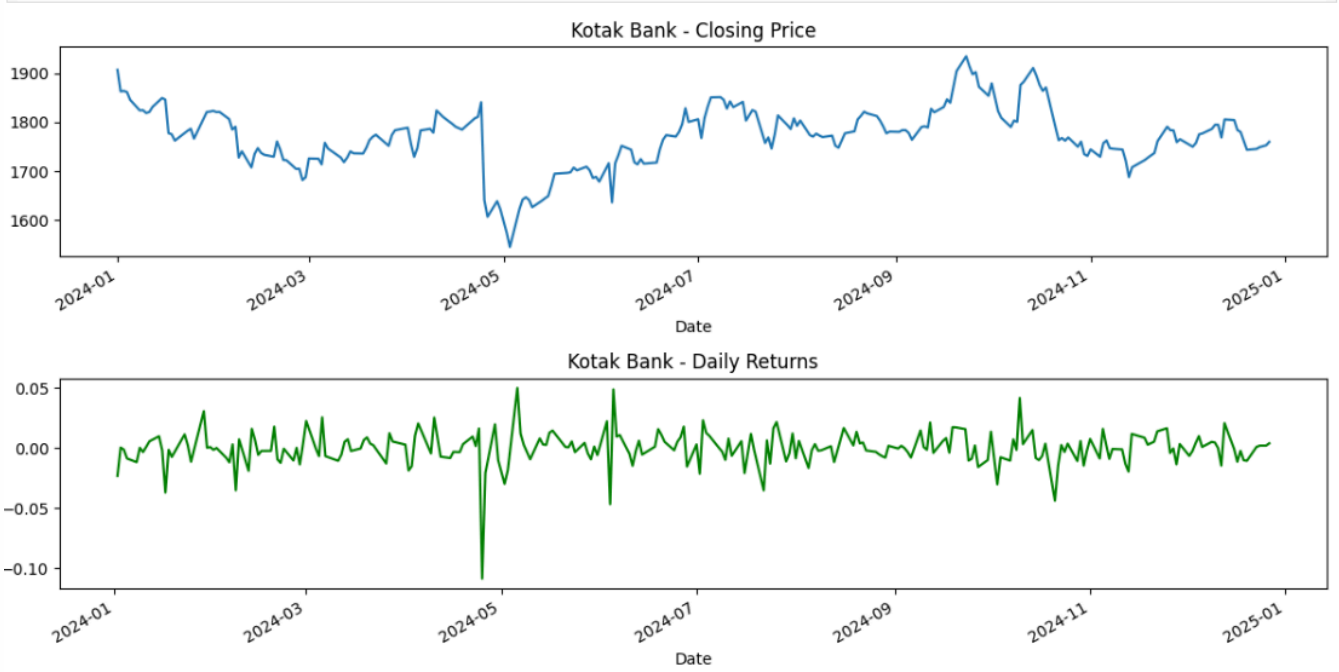





```
HDFC stock data saved to 'hdfc_stock_data.csv'.
```



:CICI stock data saved to 'icici_stock_data.csv'.

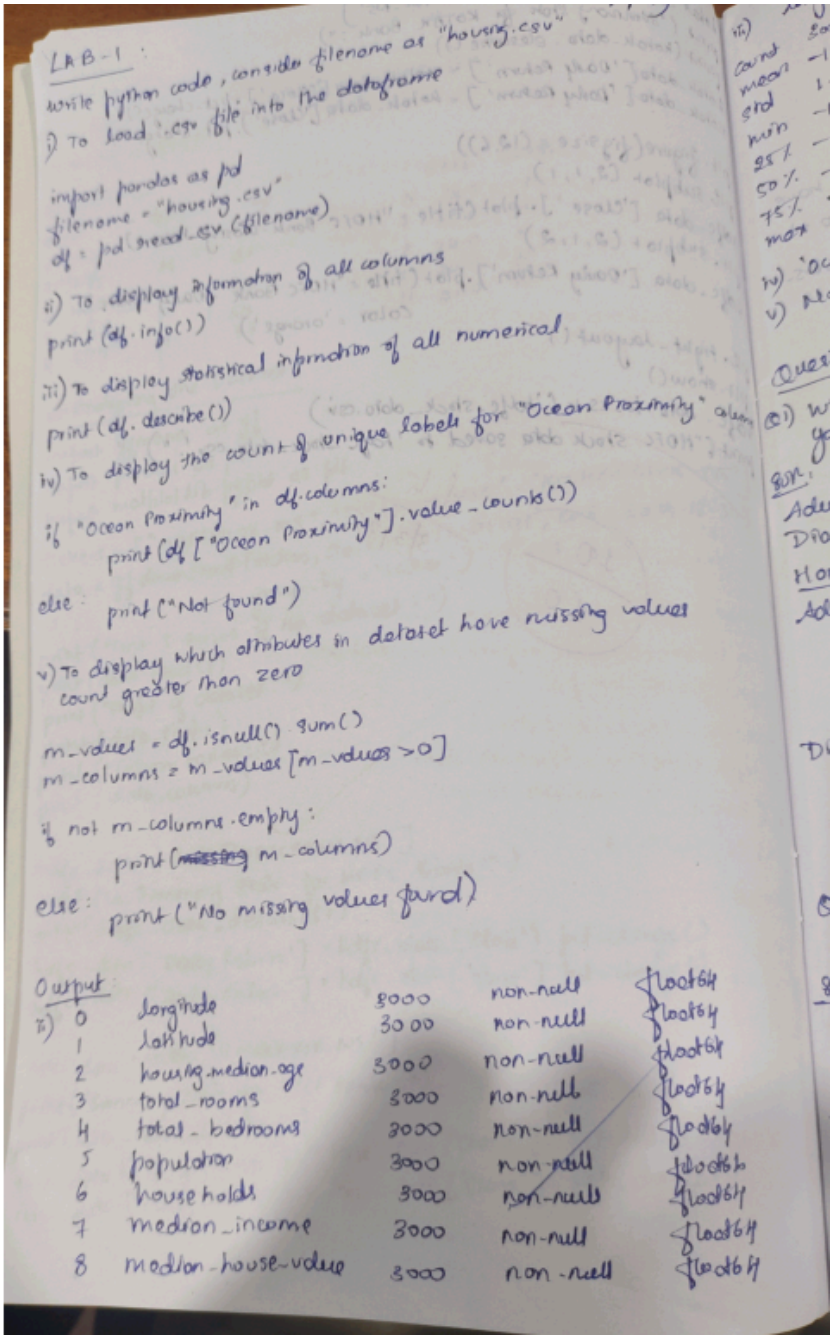


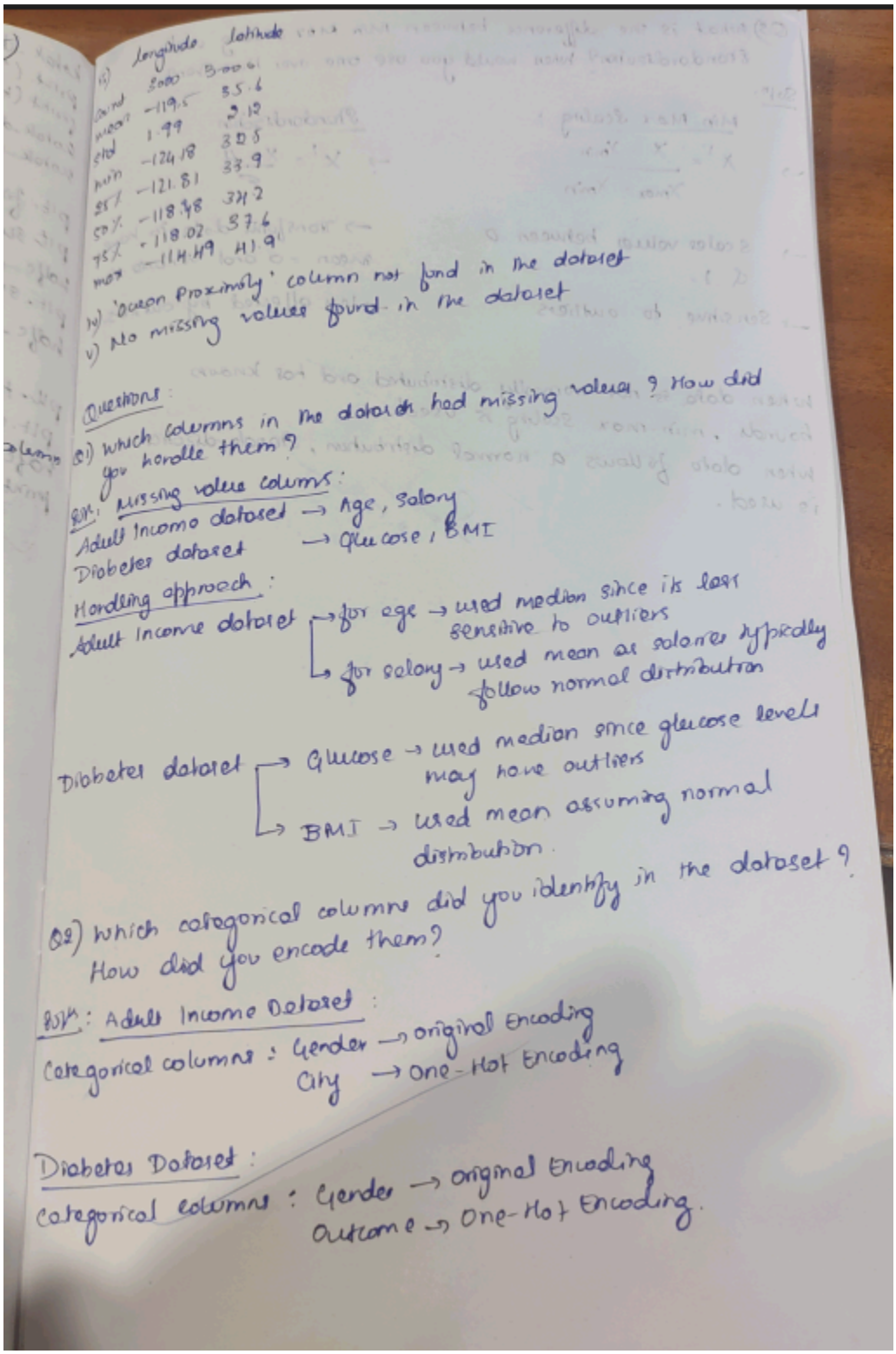
(otak stock data saved to 'kotak_stock_data.csv'.

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:





iii)

	Longitude	latitude
count	5000	5000
mean	-119.5	35.6
std	1.99	2.12
min	-124.18	32.5
25%	-121.81	33.9
50%	-118.48	34.2
75%	-118.02	37.6
max	-114.49	41.9

iv) 'Ocean Proximity' column not found in the dataset

v) No missing values found in the dataset

Questions:

Q1) Which columns in the dataset had missing values? How did you handle them?

Ans: Missing value columns:

Adult Income dataset → Age, Salary

Diabetes dataset → Glucose, BMI

Handling approach:

Adult Income dataset → for age → used median since it's less sensitive to outliers

→ for salary → used mean as salaries typically follow normal distribution

Diabetes dataset → Glucose → used median since glucose levels may have outliers

→ BMI → used mean assuming normal distribution.

Q2) Which categorical columns did you identify in the dataset? How did you encode them?

Ans: Adult Income Dataset:

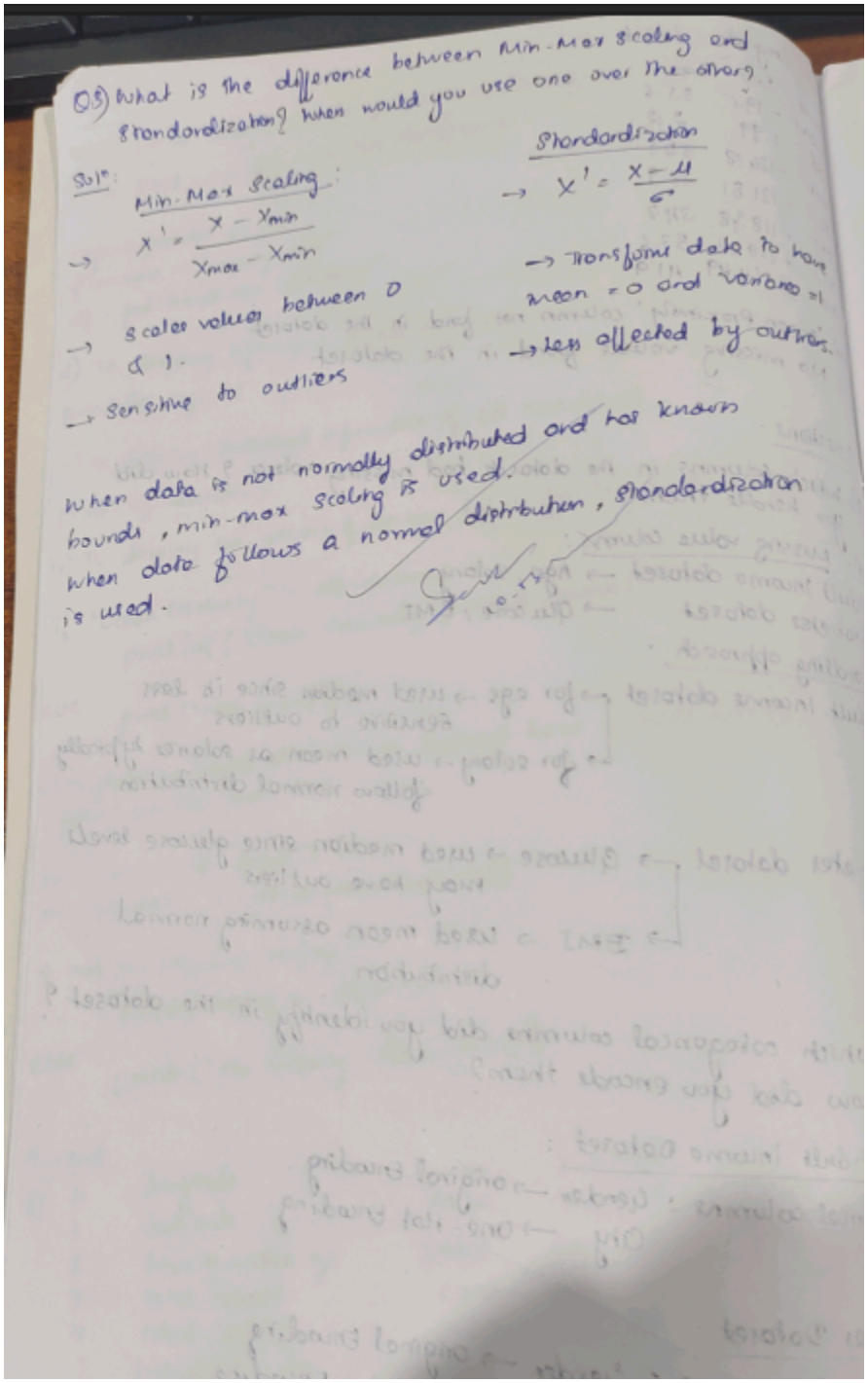
Categorical columns: Gender → original Encoding

City → One-Hot Encoding

Diabetes Dataset:

Categorical columns: Gender → original Encoding

Outcome → One-Hot Encoding.



Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt
```

```
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
filename = "/content/housing.csv"
df = pd.read_csv(filename)
print("Dataset Information:")
print(df.info())
print("\nStatistical Summary of Numerical Columns:")
print(df.describe())

if "Ocean Proximity" in df.columns:
    print("\nUnique Value Counts for 'Ocean Proximity':")
    print(df["Ocean Proximity"].value_counts())
else:
    print("\n'Ocean Proximity' column not found in the dataset.")

missing_values = df.isnull().sum()
missing_columns = missing_values[missing_values > 0]

if not missing_columns.empty:
    print("\nColumns with Missing Values:")
    print(missing_columns)
```

```
else:
    print("\nNo missing values found in the dataset.")
"'Write Python code to implement the following data preprocessing
techniques for Adult income data set"'



def createdata():
  data = {
     'Age': np.random.randint(18, 70, size=20),
     'Salary': np.random.randint(30000, 120000, size=20),
     'Purchased': np.random.choice([0, 1], size=20),
     'Gender': np.random.choice(['Male', 'Female'], size=20),
     'City': np.random.choice(['New York', 'San Francisco', 'Los Angeles'], size=20)
  }
  df = pd.DataFrame(data)
  return df
df = createdata()
df.head(10)
# Introduce some missing values for demonstration
df.loc[5, 'Age'] = np.nan
df.loc[10, 'Salary'] = np.nan
df.head(10)
```

```
#Data Cleaning

# Check for missing values in each column

missing_values = df.isnull().sum()

# Display columns with missing values

print(missing_values[missing_values > 0])

#Set the values to some value (zero, the mean, the median, etc.).

# Step 1: Create an instance of SimpleImputer with the median strategy for Age and mean stratergy for Salary

imputer1 = SimpleImputer(strategy="median")

imputer2 = SimpleImputer(strategy="mean")

df_copy=df

# Step 2: Fit the imputer on the "Age" and "Salary"column

# Note: SimpleImputer expects a 2D array, so we reshape the column

imputer1.fit(df_copy[["Age"]])

imputer2.fit(df_copy[["Salary"]])

# Step 3: Transform (fill) the missing values in the "Age" and "Salary"c column

df_copy["Age"] = imputer1.transform(df[["Age"]])

df_copy["Salary"] = imputer2.transform(df[["Salary"]])

# Verify that there are no missing values left

print(df_copy["Age"].isnull().sum())

print(df_copy["Salary"].isnull().sum())

#Handling Categorical Attributes

#Using Ordinal Encoding for gender COlumn and One-Hot Encoding for City Column

# Initialize OrdinalEncoder

ordinal_encoder = OrdinalEncoder(categories=[["Male", "Female"]])
```

```
# Fit and transform the data

df_copy["Gender_Encoded"] = ordinal_encoder.fit_transform(df_copy[["Gender"]])

# Initialize OneHotEncoder

onehot_encoder = OneHotEncoder()

# Fit and transform the "City" column

encoded_data = onehot_encoder.fit_transform(df[["City"]])

# Convert the sparse matrix to a dense array

encoded_array = encoded_data.toarray()

# Convert to DataFrame for better visualization

encoded_df = pd.DataFrame(encoded_array,
columns=onehot_encoder.get_feature_names_out(["City"]))

df_encoded = pd.concat([df_copy, encoded_df], axis=1)

df_encoded.drop("Gender", axis=1, inplace=True)

df_encoded.drop("City", axis=1, inplace=True)

print(df_encoded. head())


#Removing Outliers

# Outlier Detection and Treatment using IQR

#Pros: Simple and effective for mild outliers.

#Cons: May overly reduce variation if there are many extreme outliers.

df_encoded_copy1=df_encoded

df_encoded_copy2=df_encoded

df_encoded_copy3=df_encoded


Q1 = df_encoded_copy1['Salary'].quantile(0.25)
```

```
Q3 = df_encoded_copy1['Salary'].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR

df_encoded_copy1['Salary'] = np.where(df_encoded_copy1['Salary'] > upper_bound, upper_bound,

                    np.where(df_encoded_copy1['Salary'] < lower_bound, lower_bound,
df_encoded_copy1['Salary']))

df_encoded_copy1.head()


#Data Transformation

# Min-Max Scaler/Normalization (range 0-1)

#Pros: Keeps all data between 0 and 1; ideal for distance-based models.

#Cons: Can distort data distribution, especially with extreme outliers.

normalizer = MinMaxScaler()

df_encoded[['Salary']] = normalizer.fit_transform(df_encoded[['Salary']])

df_encoded.head()


# Standardization (mean=0, variance=1)

#Pros: Works well for normally distributed data; suitable for many models.

#Cons: Sensitive to outliers.

scaler = StandardScaler()

df_encoded[['Age']] = scaler.fit_transform(df_encoded[['Age']])

df_encoded.head()


'"Write Python code to implement the following data preprocessing
```

```
techniques for Diabetes data set"'
def create_diabetes_data():
    data = {
        'Age': np.random.randint(20, 80, size=20),
        'Glucose': np.random.randint(50, 200, size=20),
        'BMI': np.random.uniform(18.5, 45.0, size=20),
        'Gender': np.random.choice(['Male', 'Female'], size=20),
        'Outcome': np.random.choice([0, 1], size=20)
    }
    df = pd.DataFrame(data)
    # Introduce some missing values for demonstration
    df.loc[5, 'Glucose'] = np.nan
    df.loc[10, 'BMI'] = np.nan
    return df
# Load dataset
df = create_diabetes_data()
print("Original Data:\n", df.head())

# Data Cleaning - Handling Missing Values
imputer_glucose = SimpleImputer(strategy="median")
imputer_bmi = SimpleImputer(strategy="mean")
df_copy = df.copy()
df_copy["Glucose"] = imputer_glucose.fit_transform(df_copy[["Glucose"]])
df_copy["BMI"] = imputer_bmi.fit_transform(df_copy[["BMI"]])
```

```
print(df_copy["Glucose"].isnull().sum())

print(df_copy["BMI"].isnull().sum())


# Handling Categorical Attributes

# Ordinal Encoding for Gender

ordinal_encoder = OrdinalEncoder(categories=[["Male", "Female"]])

df_copy["Gender_Encoded"] = ordinal_encoder.fit_transform(df_copy[["Gender"]])

# One-Hot Encoding for Outcome column

onehot_encoder = OneHotEncoder()

encoded_outcome = onehot_encoder.fit_transform(df_copy[["Outcome"]]).toarray()

encoded_df = pd.DataFrame(encoded_outcome,
columns=onehot_encoder.get_feature_names_out(["Outcome"]))

df_encoded = pd.concat([df_copy, encoded_df], axis=1)

df_encoded.drop(["Gender", "Outcome"], axis=1, inplace=True)

print("\nAfter Encoding:\n", df_encoded.head())


# Removing Outliers using IQR (for Glucose)

Q1 = df_encoded["Glucose"].quantile(0.25)

Q3 = df_encoded["Glucose"].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR

df_encoded["Glucose"] = np.where(df_encoded["Glucose"] > upper_bound, upper_bound,

                        np.where(df_encoded["Glucose"] < lower_bound, lower_bound,
df_encoded["Glucose"]))
```

```
df_encoded.head()


# Data Transformation

# Min-Max Scaling for BMI

normalizer = MinMaxScaler()

df_encoded[['BMI']] = normalizer.fit_transform(df_encoded[['BMI']])

df_encoded.head()


# Standardization for Age

scaler = StandardScaler()

df_encoded[['Age']] = scaler.fit_transform(df_encoded[['Age']])

print("\nFinal Preprocessed Data:\n", df_encoded.head())
```

Output:

```
Statistical Summary of Numerical Columns:
         longitude    latitude  housing_median_age   total_rooms  \
count  3000.000000  3000.00000         3000.000000   3000.000000
mean   -119.589200    35.63539           28.845333   2599.578667
std       1.994936     2.12967           12.555396   2155.593332
min    -124.180000    32.56000            1.000000      6.000000
25%    -121.810000    33.93000           18.000000   1401.000000
50%    -118.485000    34.27000           29.000000   2106.000000
75%    -118.020000    37.69000           37.000000   3129.000000
max    -114.490000    41.92000           52.000000  30450.000000

       total_bedrooms    population  households  median_income  \
count     3000.000000   3000.000000  3000.00000    3000.000000
mean       529.950667   1402.798667   489.91200       3.807272
std        415.654368   1030.543012   365.42271       1.854512
min          2.000000      5.000000     2.00000       0.499900
25%        291.000000    780.000000   273.00000       2.544000
50%        437.000000   1155.000000   409.50000       3.487150
75%        636.000000   1742.750000   597.25000       4.656475
max       5419.000000  11935.000000  4930.00000      15.000100

       median_house_value
count          3000.00000
mean         205846.27500
std          113119.68747
min           22500.00000
25%          121200.00000
50%          177650.00000
75%          263975.00000
max          500001.00000

'Ocean Proximity' column not found in the dataset.

No missing values found in the dataset.
```

	Age	Salary	Purchased	Gender	City
0	21	98335	0	Female	San Francisco
1	29	107193	1	Female	San Francisco
2	40	65298	0	Male	Los Angeles
3	67	110619	0	Female	New York
4	35	35539	1	Female	Los Angeles
5	52	119090	1	Male	New York
6	40	87266	0	Male	San Francisco
7	20	118367	1	Male	Los Angeles
8	48	102695	0	Female	San Francisco
9	26	66367	0	Male	San Francisco

```
    Age    Salary  Purchased  Gender_Encoded  City_Los Angeles  City_New York  \
0  21.0   98335.0          0             1.0               0.0            0.0
1  29.0  107193.0          1             1.0               0.0            0.0
2  40.0   65298.0          0             0.0               1.0            0.0
3  67.0  110619.0          0             1.0               0.0            1.0
4  35.0   35539.0          1             1.0               1.0            0.0

   City_San Francisco
0                 1.0
1                 1.0
2                 0.0
3                 0.0
4                 0.0
```

	Age	Salary	Purchased	Gender_Encoded	City_Los Angeles	City_New York	City_San Francisco
0	21.0	0.745689	0	1.0	0.0	0.0	1.0
1	29.0	0.850876	1	1.0	0.0	0.0	1.0
2	40.0	0.353382	0	0.0	1.0	0.0	0.0
3	67.0	0.891559	0	1.0	0.0	1.0	0.0
4	35.0	0.000000	1	1.0	1.0	0.0	0.0

	Age	Glucose	BMI	Gender_Encoded	Outcome_0	Outcome_1
0	1.982577	93.0	0.991819	1.0	1.0	0.0
1	-0.110300	195.0	0.435823	0.0	0.0	1.0
2	0.229085	87.0	0.264387	1.0	0.0	1.0
3	-0.279993	113.0	0.785170	1.0	0.0	1.0
4	-1.015328	163.0	0.144165	0.0	0.0	1.0

```
Original Data:
    Age  Glucose        BMI  Gender  Outcome
0   77     93.0  44.640289  Female        0
1   40    195.0  30.589381    Male        1
2   46     87.0  26.256915  Female        1
3   37    113.0  39.417936  Female        1
4   24    163.0  23.218723    Male        1
```

	Age	Glucose	BMI	Gender_Encoded	Outcome_0	Outcome_1
0	77	93.0	0.991819	1.0	1.0	0.0
1	40	195.0	0.435823	0.0	0.0	1.0
2	46	87.0	0.264387	1.0	0.0	1.0
3	37	113.0	0.785170	1.0	0.0	1.0
4	24	163.0	0.144165	0.0	0.0	1.0

```
Final Preprocessed Data:
        Age  Glucose       BMI  Gender_Encoded  Outcome_0  Outcome_1
0  1.982577     93.0  0.991819             1.0        1.0        0.0
1 -0.110300    195.0  0.435823             0.0        0.0        1.0
2  0.229085     87.0  0.264387             1.0        0.0        1.0
3 -0.279993    113.0  0.785170             1.0        0.0        1.0
4 -1.015328    163.0  0.144165             0.0        0.0        1.0
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot:

LAB - 3

Simple Linear Regression

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

def estimate_coef(x,y):
    n = np.size(x)
    m_x = np.mean(x)
    m_y = np.mean(y)
    SS_xy = np.sum((x-m_x)*(y-m_y))
    SS_xx = np.sum((x-m_x)**2)
    b_1 = SS_xy/SS_xx
    b_0 = m_y - b_1*m_x
    return (b_0, b_1)

def plot(x,y,b):
    plt.scatter(x,y, color='m')
    y_pred = b[0] + b[1]*x
    plt.plot(x, y_pred, color='g')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.show()

file_path = input("Enter path:")
df = pd.read_csv(file_path)

x = df.iloc[:,0].values
y = df.iloc[:,1].values

b = estimate(x,y)
plot(x,y,b)
```

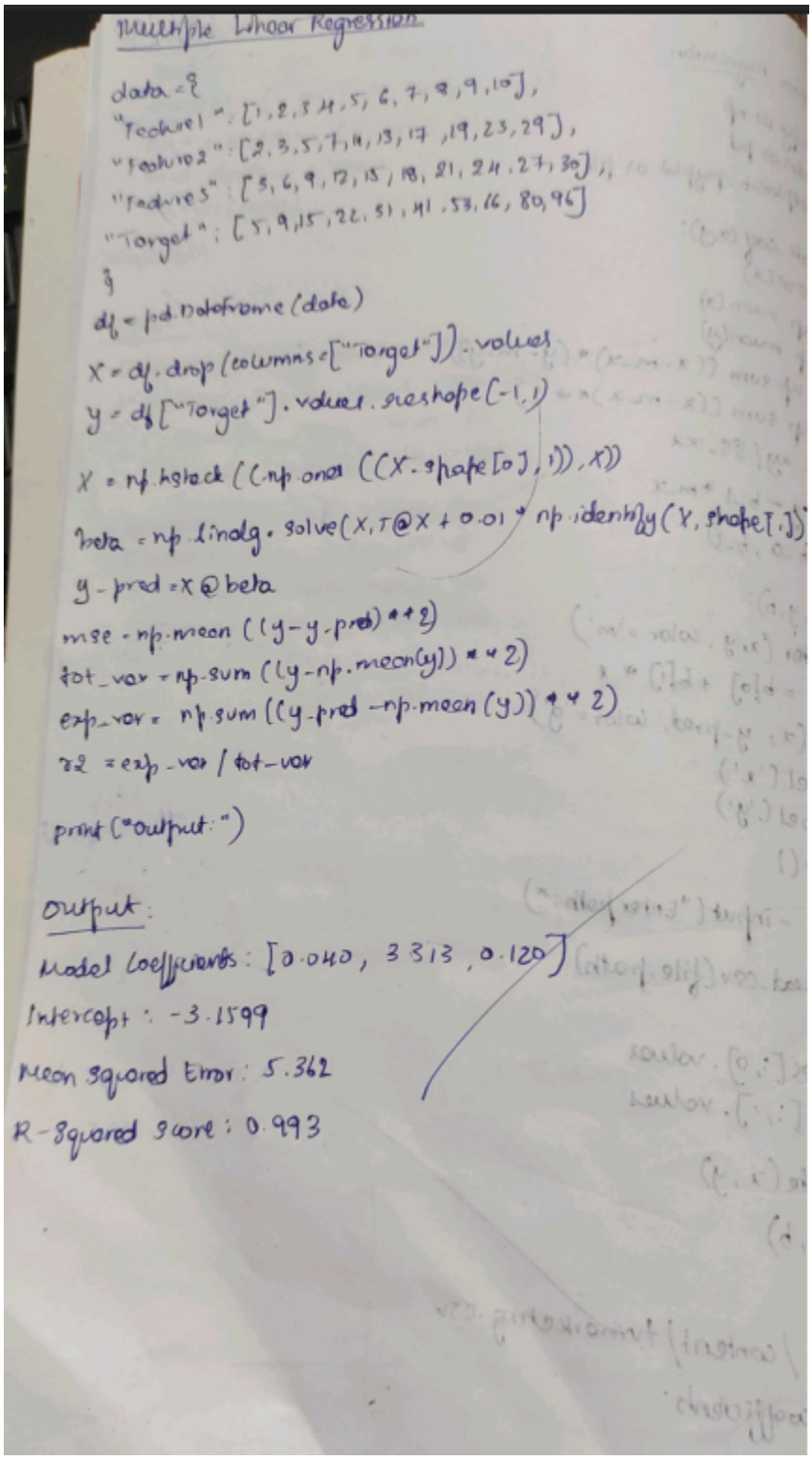
Output:

Enter path: /content/tvmarketing.csv

Estimated coefficients:

b_0 = 7.032

b_1 = 0.047



Multiple Linear Regression

```
data = {
"feature1": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
"feature2": [2, 3, 5, 7, 11, 13, 17, 19, 23, 29],
"feature3": [3, 6, 9, 12, 15, 18, 21, 24, 27, 30],
"Target": [5, 9, 15, 22, 31, 41, 53, 66, 80, 96]
}
df = pd.DataFrame(data)
X = df.drop(columns=["Target"]).values
y = df["Target"].values.reshape(-1,1)
X = np.hstack((np.ones((X.shape[0], 1)), X))
beta = np.linalg.solve(X.T@X + 0.01 * np.identity(X.shape[1]))
y_pred = X @ beta
mse = np.mean((y-y_pred)**2)
tot_var = np.sum((y-np.mean(y))**2)
exp_var = np.sum((y_pred - np.mean(y))**2)
r2 = exp_var / tot_var
print("output:")
```

Output:

Model Coefficients: [0.040, 3.313, 0.120]

Intercept: -3.1599

Mean Squared Error: 5.362

R-Squared Score: 0.993

Code:

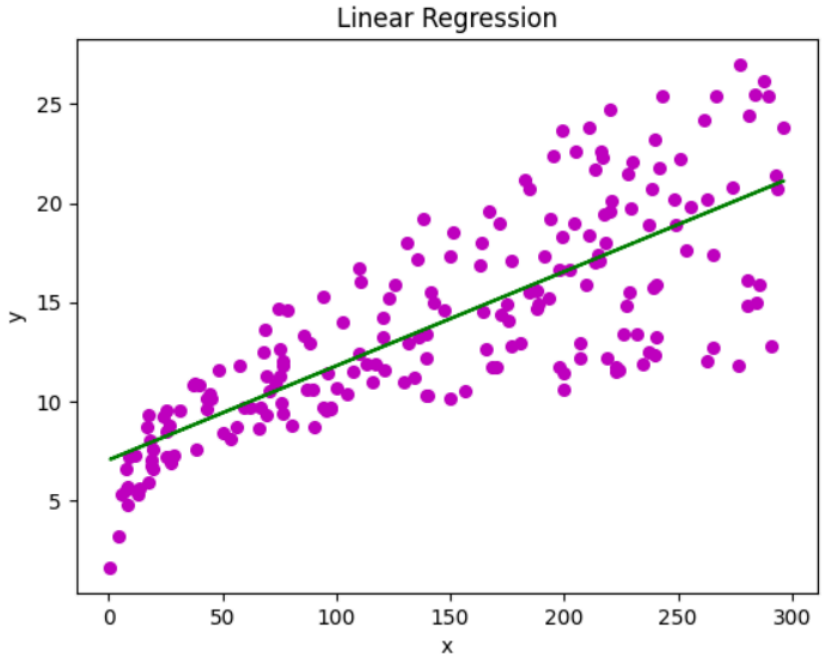
```
#Linear Regression
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
def estimate_coef(x, y):
    n = np.size(x)
    m_x = np.mean(x)
    m_y = np.mean(y)
    SS_xy = np.sum((x - m_x) * (y - m_y))
    SS_xx = np.sum((x - m_x) ** 2)
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1 * m_x
    return (b_0, b_1)
def plot_regression_line(x, y, b):
    plt.scatter(x, y, color="m", marker="o", s=30)
    y_pred = b[0] + b[1] * x
    plt.plot(x, y_pred, color="g")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title("Linear Regression")
    plt.show()
# Load dataset
file_path = input("Enter the path to the CSV file: ")
```

```
df = pd.read_csv(file_path)
# Assuming the dataset has two numerical columns: 'x' and 'y'
x = df.iloc[:, 0].values  # First column as x
y = df.iloc[:, 1].values  # Second column as y
b = estimate_coef(x, y)
print(f"Estimated coefficients:\nb_0 = {b[0]} \nb_1 = {b[1]}")
plot_regression_line(x, y, b)

#Multiple Regression
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# Sample dataset
data = {
    "Feature1": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
    "Feature2": [2, 3, 5, 7, 11, 13, 17, 19, 23, 29],
    "Feature3": [3, 6, 9, 12, 15, 18, 21, 24, 27, 30],
    "Target": [5, 9, 15, 22, 31, 41, 53, 66, 80, 96]
}
df = pd.DataFrame(data)
# Split dataset into features (X) and target variable (y)
X = df.drop(columns=["Target"]).values
y = df["Target"].values.reshape(-1, 1)
```

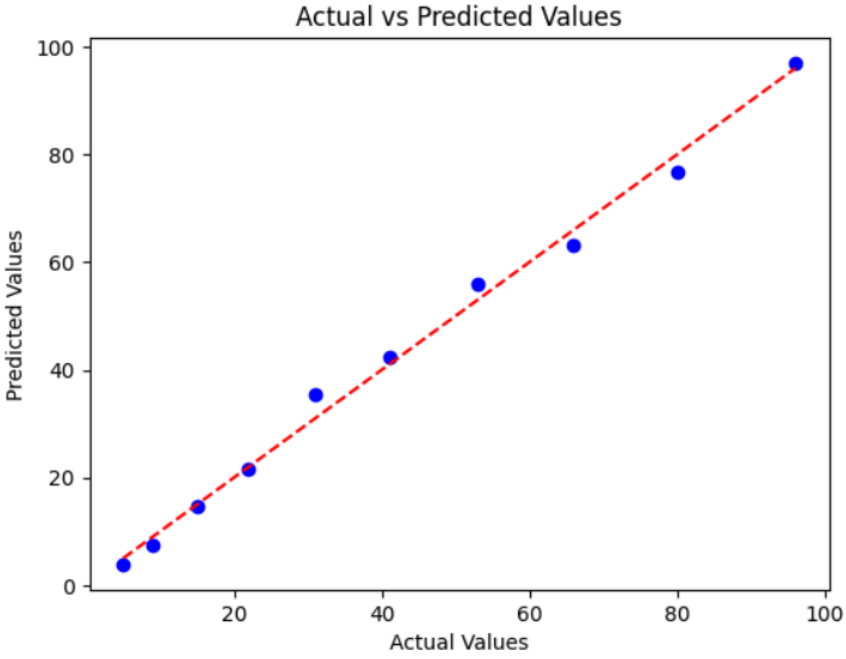
```
# Add intercept column (bias term)
X = np.hstack((np.ones((X.shape[0], 1)), X))
# Compute the coefficients using the Normal Equation
beta = np.linalg.solve(X.T @ X + 0.01 * np.identity(X.shape[1]), X.T @ y)
# Make predictions
y_pred = X @ beta
# Evaluate the model
mse = np.mean((y - y_pred) ** 2)
total_variance = np.sum((y - np.mean(y)) ** 2)
explained_variance = np.sum((y_pred - np.mean(y)) ** 2)
r2 = explained_variance / total_variance
# Display results
print("Model Coefficients:", beta[1:].flatten())
print("Intercept:", beta[0][0])
print("Mean Squared Error:", mse)
print("R-squared Score:", r2)
# Plot actual vs predicted values
plt.scatter(y, y_pred, color='blue')
plt.plot(y, y, color='red', linestyle='--')
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted Values")
plt.show()
```

Output:

```
Enter the path to the CSV file: /content/tvmarketing.csv
Estimated coefficients:
b_0 = 7.0325935491276965
b_1 = 0.047536640433019736
```



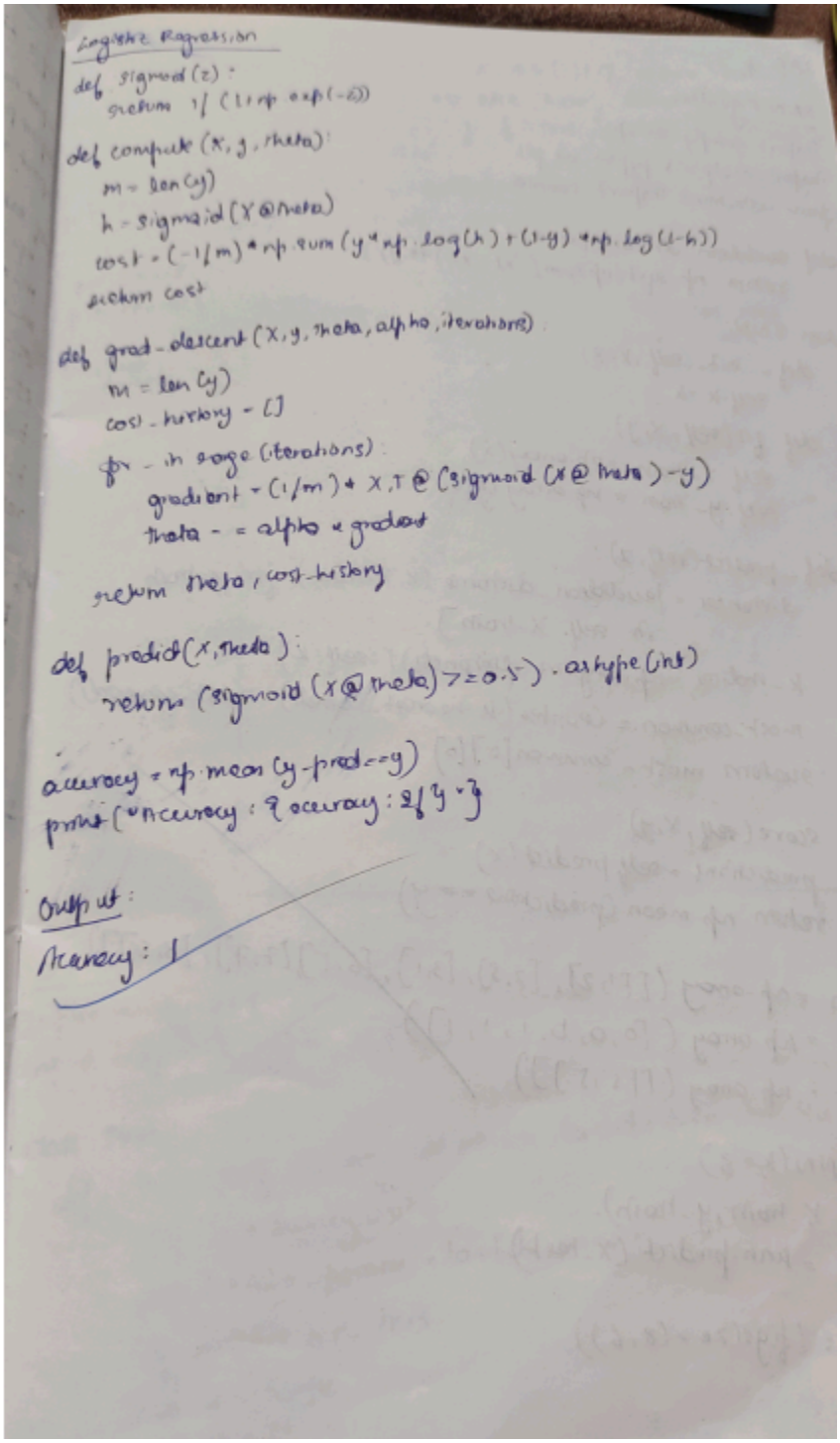

```
Model Coefficients: [0.04026032 3.31385993 0.12078097]
Intercept: -3.1599509492879228
Mean Squared Error: 5.362912814290877
R-squared Score: 0.9935326149415827
```



Program 4

Build Logistic Regression Model for a given dataset

Screenshot:



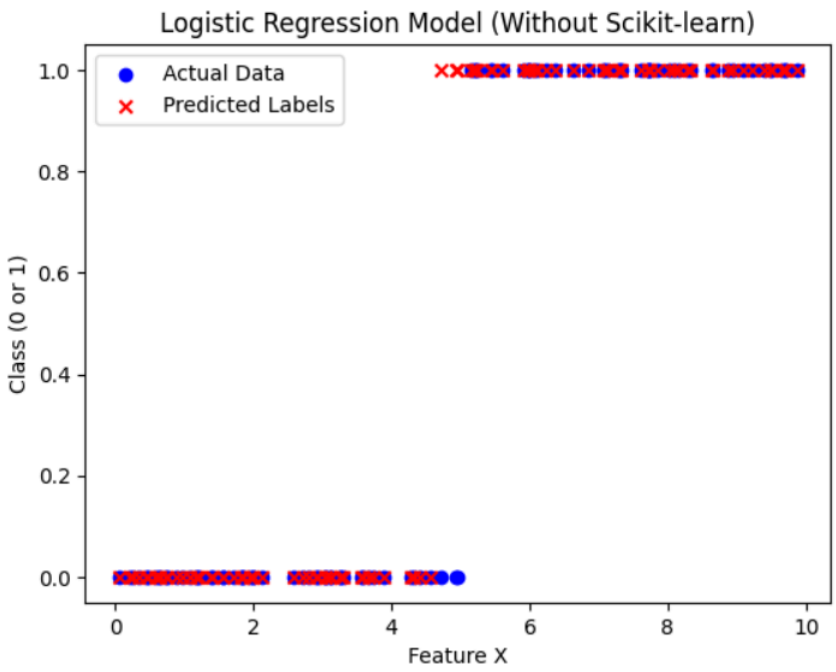
Code:

```
#Logistic Regression
import numpy as np
import matplotlib.pyplot as plt
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
def compute_cost(X, y, theta):
    m = len(y)
    h = sigmoid(X @ theta)
    cost = (-1/m) * np.sum(y * np.log(h) + (1 - y) * np.log(1 - h))
    return cost
def gradient_descent(X, y, theta, alpha, iterations):
    m = len(y)
    cost_history = []
    for _ in range(iterations):
        gradient = (1/m) * X.T @ (sigmoid(X @ theta) - y)
        theta -= alpha * gradient
        cost_history.append(compute_cost(X, y, theta))
    return theta, cost_history
def predict(X, theta):
    return (sigmoid(X @ theta) >= 0.5).astype(int)
# Generate synthetic binary classification data
np.random.seed(42)
X = np.random.rand(100, 1) * 10  # Feature values between 0 and 10
```

```
y = (X > 5).astype(int).ravel()  # Label: 1 if X > 5, else 0

# Add intercept term

X_b = np.c_[np.ones((X.shape[0], 1)), X]

# Initialize parameters

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

alpha = 0.1

iterations = 1000

# Train logistic regression using gradient descent

theta, cost_history = gradient_descent(X_b, y, theta, alpha, iterations)

# Make predictions

y_pred = predict(X_b, theta)

# Compute accuracy

accuracy = np.mean(y_pred == y)

print(f"Accuracy: {accuracy:.2f}")

# Plot the decision boundary

plt.scatter(X, y, color='blue', label='Actual Data')

plt.scatter(X, y_pred, color='red', marker='x', label='Predicted Labels')

plt.xlabel("Feature X")

plt.ylabel("Class (0 or 1)")

plt.legend()

plt.title("Logistic Regression Model (Without Scikit-learn)")

plt.show()
```

Output:

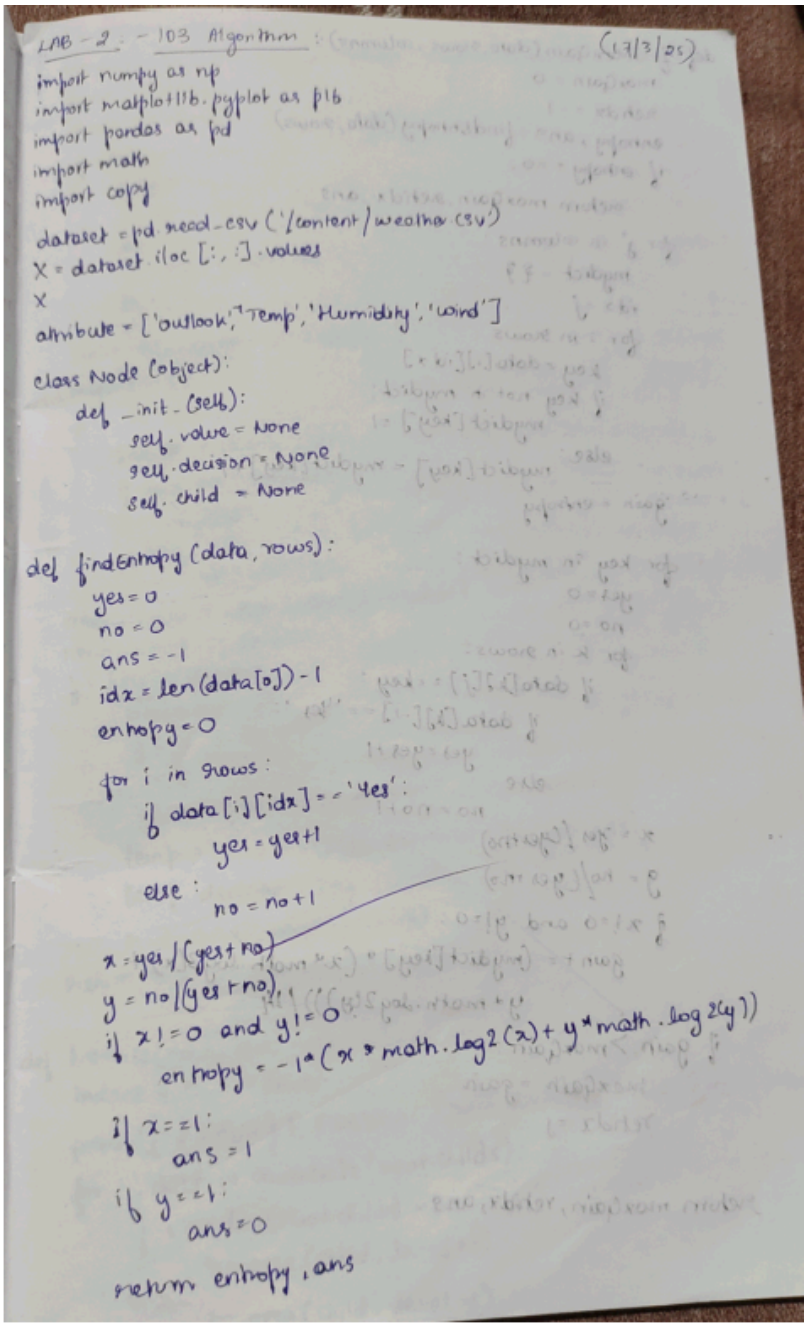
```
Accuracy: 0.97
```



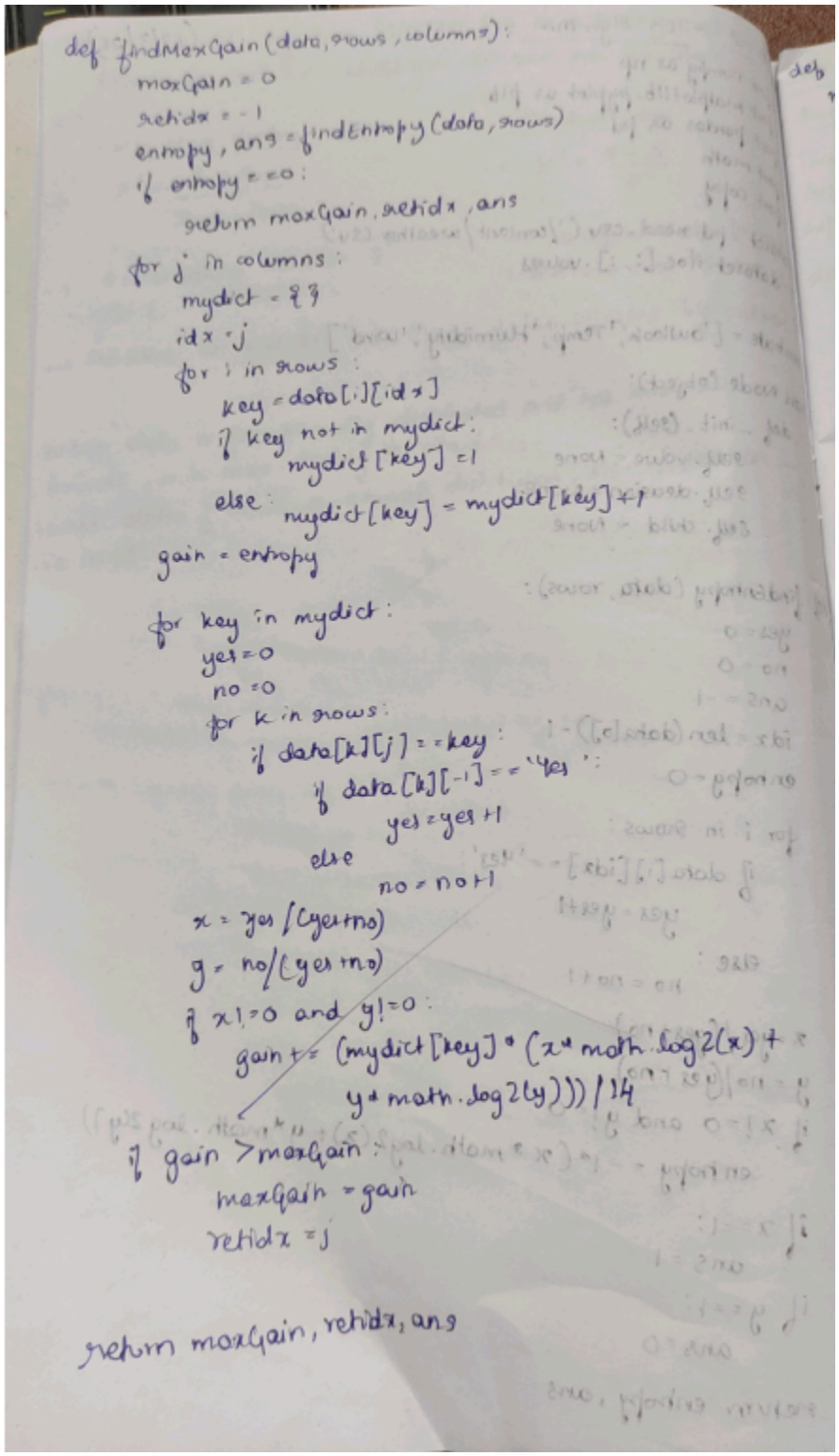
Program 5

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

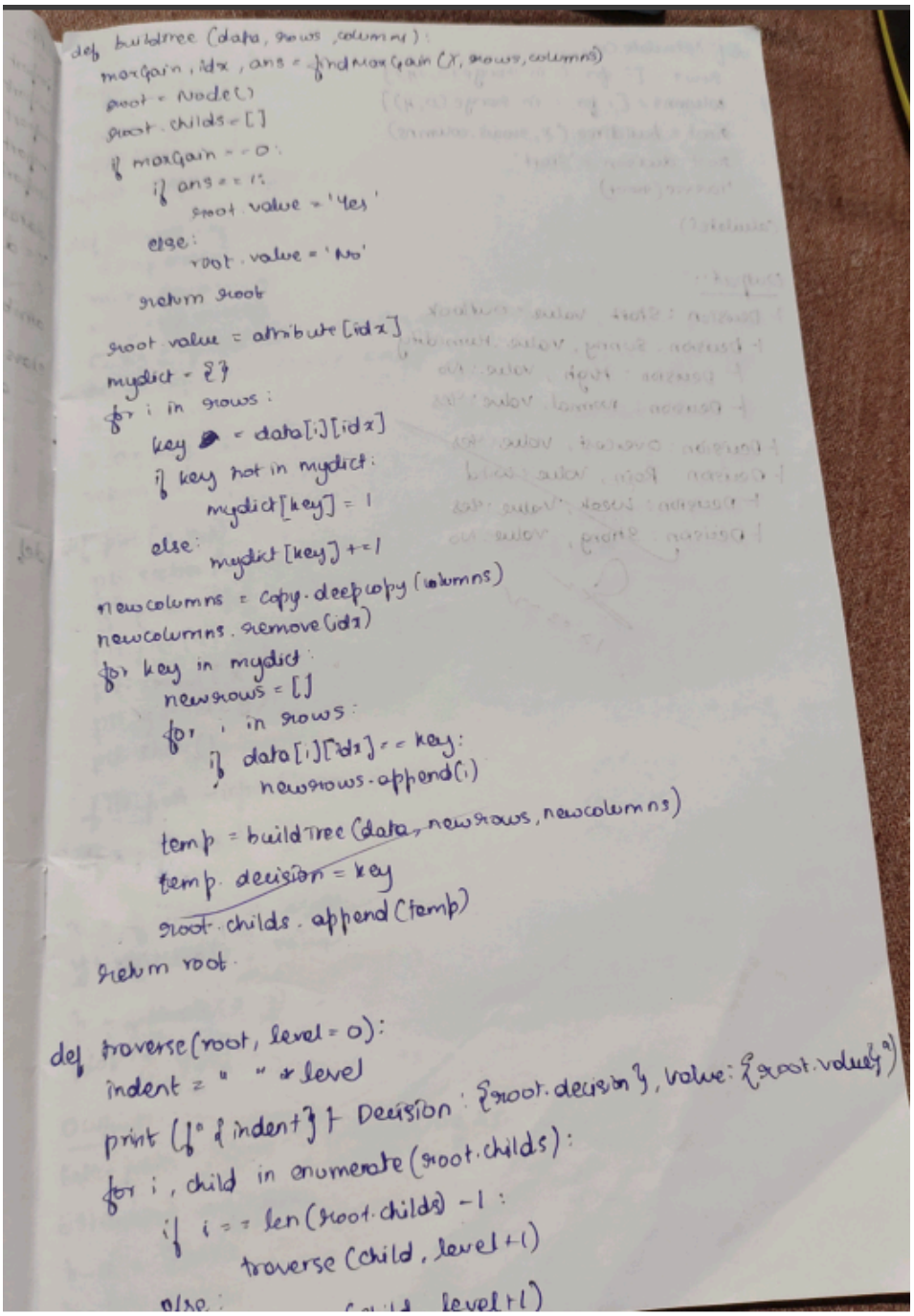
Screenshot:



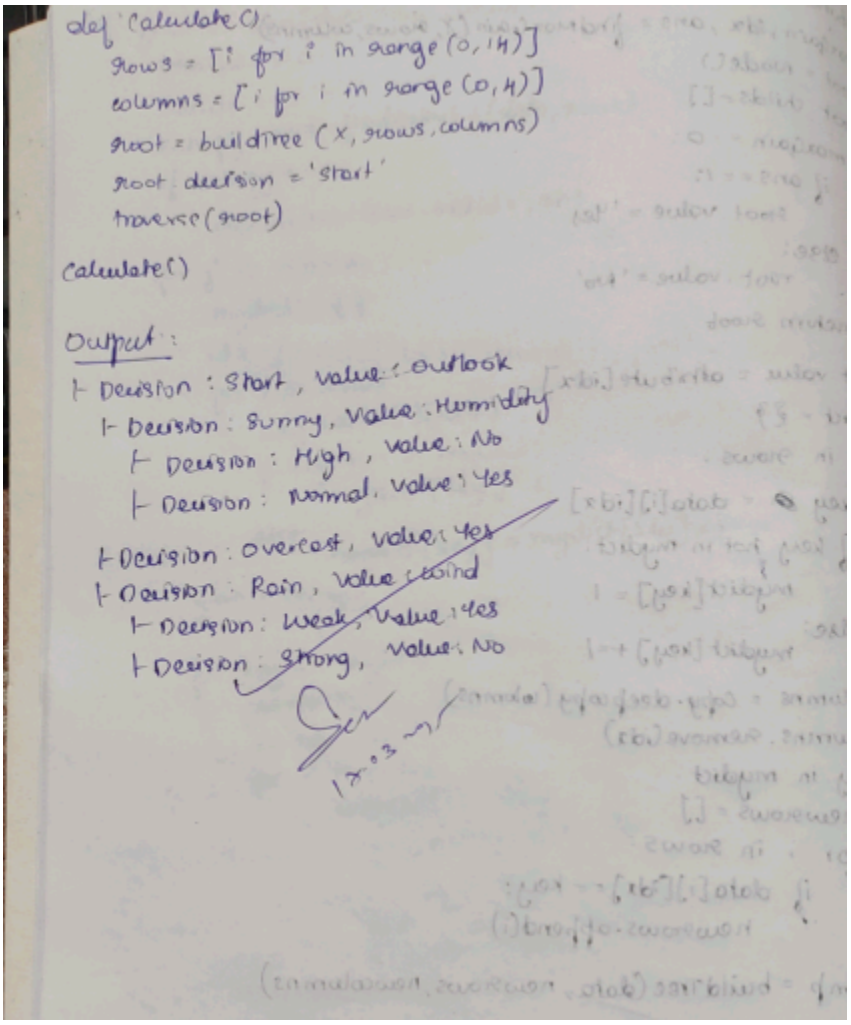
```
LAB - 2 : - ID3 Algorithm :                                    (17/3/25)
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
import copy
dataset = pd.read_csv('/content/weather.csv')
X = dataset.iloc[:, :].values
X
attribute = ['outlook', 'Temp', 'Humidity', 'wind']

class Node(object):
    def __init__(self):
        self.value = None
        self.decision = None
        self.child = None

def findEntropy(data, rows):
    yes = 0
    no = 0
    ans = -1
    idx = len(data[0]) - 1
    entropy = 0

    for i in rows:
        if data[i][idx] == 'yes':
            yes = yes + 1
        else:
            no = no + 1
    x = yes / (yes + no)
    y = no / (yes + no)
    if x != 0 and y != 0:
        entropy = -1 * (x * math.log2(x) + y * math.log2(y))

    if x == 1:
        ans = 1
    if y == 1:
        ans = 0

    return entropy, ans
```



```
def findMaxGain(data, rows, columns):
    maxGain = 0
    retidx = -1
    entropy, ans = findEntropy(data, rows)
    if entropy == 0:
        return maxGain, retidx, ans
    for j in columns:
        mydict = {}
        idx = j
        for i in rows:
            key = data[i][idx]
            if key not in mydict:
                mydict[key] = 1
            else:
                mydict[key] = mydict[key] + 1
        gain = entropy

        for key in mydict:
            yes = 0
            no = 0
            for k in rows:
                if data[k][j] == key:
                    if data[k][-1] == 'yes':
                        yes = yes + 1
                    else
                        no = no + 1
            x = yes/(yes+no)
            y = no/(yes+no)
            if x!=0 and y!=0:
                gain += (mydict[key] * (x * math.log2(x) +
                        y * math.log2(y)))/14
        if gain > maxGain:
            maxGain = gain
            retidx = j

    return maxGain, retidx, ans
```



```
def buildTree(data, rows, columns):
    maxGain, idx, ans = findMaxGain(X, rows, columns)
    root = Node()
    root.childs = []
    if maxGain == 0:
        if ans == 1:
            root.value = 'Yes'
        else:
            root.value = 'No'
        return root
    root.value = attribute[idx]
    mydict = {}
    for i in rows:
        key = data[i][idx]
        if key not in mydict:
            mydict[key] = 1
        else:
            mydict[key] += 1
    newcolumns = copy.deepcopy(columns)
    newcolumns.remove(idx)
    for key in mydict:
        newrows = []
        for i in rows:
            if data[i][idx] == key:
                newrows.append(i)
        temp = buildTree(data, newrows, newcolumns)
        temp.decision = key
        root.childs.append(temp)
    return root

def traverse(root, level = 0):
    indent = "  " * level
    print(f"{indent}Decision: {root.decision}, Value: {root.value}")
    for i, child in enumerate(root.childs):
        if i == len(root.childs) - 1:
            traverse(child, level+1)
        else:
            traverse(child, level+1)
```



Code:

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import math

import copy

dataset = pd.read_csv('/content/weather.csv')

X = dataset.iloc[:,:].values

X

attribute = ['Outlook', 'Temp', 'Humidity', 'Wind']
```

```
class Node(object):
    def __init__(self):
        self.value = None
        self.decision = None
        self.child = None
def findEntropy(data, rows):
    yes=0
    no=0
    ans=-1
    idx=len(data[0])-1
    entropy=0
    for i in rows:
        if data[i][idx]=='Yes':
            yes=yes+1
        else:
            no=no+1
    x=yes/(yes+no)
    y=no/(yes+no)
    if x!=0 and y!=0:
        entropy= -1*(x*math.log2(x)+y*math.log2(y))
    if x==1:
        ans = 1
    if y==1:
        ans = 0
```

```
    return entropy, ans
def findMaxGain(data, rows, columns):
    maxGain = 0
    retidx = -1
    entropy, ans = findEntropy(data, rows)
    if entropy == 0:
        """if ans == 1:
            print("Yes")
        else:
            print("No")"""
        return maxGain, retidx, ans
    for j in columns:
        mydict = {}
        idx = j
        for i in rows:
            key = data[i][idx]
            if key not in mydict:
                mydict[key] = 1
            else:
                mydict[key] = mydict[key] + 1
        gain = entropy
        # print(mydict)
        for key in mydict:
            yes = 0
```

```
            no = 0
            for k in rows:
                if data[k][j] == key:
                    if data[k][-1] == 'Yes':
                        yes = yes + 1
                    else:
                        no = no + 1
            # print(yes, no)
            x = yes/(yes+no)
            y = no/(yes+no)
            # print(x, y)
            if x != 0 and y != 0:
                gain += (mydict[key] * (x*math.log2(x) + y*math.log2(y)))/14
        # print(gain)
        if gain > maxGain:
            # print("hello")
            maxGain = gain
            retidx = j
    return maxGain, retidx, ans
def buildTree(data, rows, columns):
    maxGain, idx, ans = findMaxGain(X, rows, columns)
    root = Node()
    root.childs = []
    # print(maxGain)
```

```
    if maxGain == 0:
        if ans == 1:
            root.value = 'Yes'
        else:
            root.value = 'No'
        return root
    root.value = attribute[idx]
    mydict = {}
    for i in rows:
        key = data[i][idx]
        if key not in mydict:
            mydict[key] = 1
        else:
            mydict[key] += 1
    newcolumns = copy.deepcopy(columns)
    newcolumns.remove(idx)
    for key in mydict:
        newrows = []
        for i in rows:
            if data[i][idx] == key:
                newrows.append(i)
        # print(newrows)
        temp = buildTree(data, newrows, newcolumns)
```

```
        temp.decision = key
        root.childs.append(temp)
    return root
def traverse(root, level=0):
    # Create indentation based on the level
    indent = "    " * level  # 4 spaces per level
    # Print the decision and value of the current node with indentation
    print(f"{indent} ├── Decision: {root.decision}, Value: {root.value}")
    # Traverse the children, if they exist
    for i, child in enumerate(root.childs):
        if i == len(root.childs) - 1:  # Last child
            traverse(child, level + 1)
        else:
            traverse(child, level + 1)
def calculate():
    rows = [i for i in range(0, 14)]
    columns = [i for i in range(0, 4)]
    root = buildTree(X, rows, columns)
    root.decision = 'Start'
    traverse(root)
calculate()
from graphviz import Source
dot_code = """
digraph G {
```

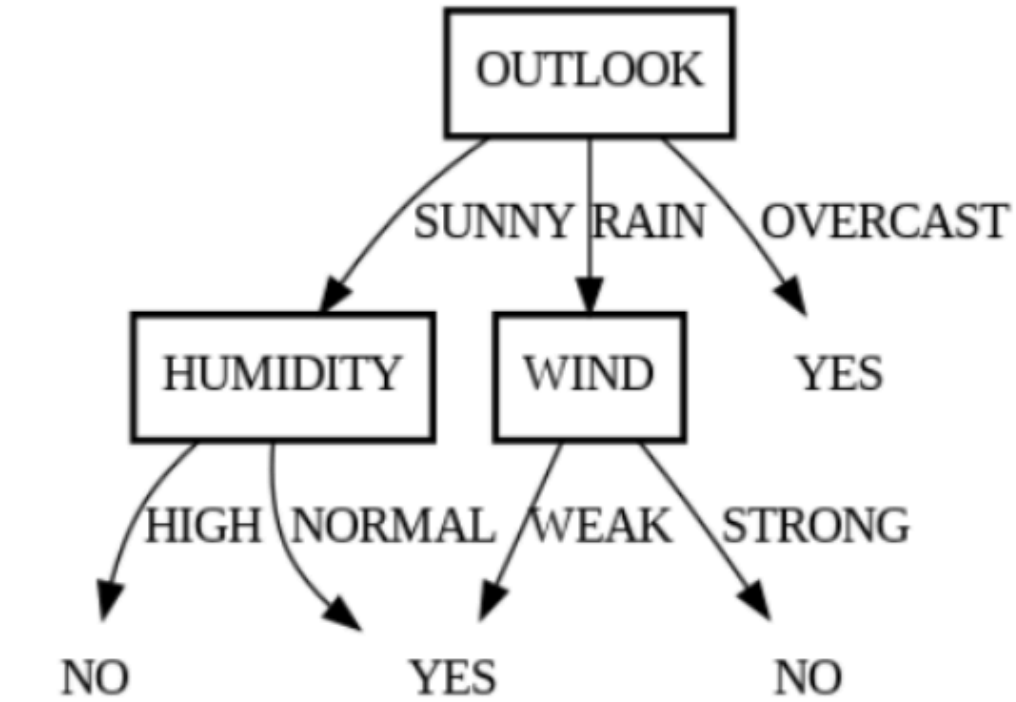
```
    edge [dir=forward]
    node [shape=box, style=bold]
    A [label="OUTLOOK"]
    B [label="HUMIDITY"]
    C [label="WIND"]
    D [label="NO", shape=plaintext]
    E [label="YES", shape=plaintext]
    F [label="YES", shape=plaintext]
    G [label="NO", shape=plaintext]
    A -> B [label="SUNNY"]
    A -> E [label="OVERCAST"]
    A -> C [label="RAIN"]
    B -> D [label="HIGH"]
    B -> F [label="NORMAL"]
    C -> F [label="WEAK"]
    C -> G [label="STRONG"]
}
"""
s = Source(dot_code, filename="decision_tree", format="png")
s.view()
```

Output:

```
array([['Sunny', 'Hot', 'High', 'Weak', 'No'],
       ['Sunny', 'Hot', 'High', 'Strong', 'No'],
       ['Overcast', 'Hot', 'High', 'Weak', 'Yes'],
       ['Rain', 'Mild', 'High', 'Weak', 'Yes'],
       ['Rain', 'Cool', 'Normal', 'Weak', 'Yes'],
       ['Rain', 'Cool', 'Normal', 'Strong', 'No'],
       ['Overcast', 'Cool', 'Normal', 'Strong', 'Yes'],
       ['Sunny', 'Mild', 'High', 'Weak', 'No'],
       ['Sunny', 'Cool', 'Normal', 'Weak', 'Yes'],
       ['Rain', 'Mild', 'Normal', 'Weak', 'Yes'],
       ['Sunny', 'Mild', 'Normal', 'Strong', 'Yes'],
       ['Overcast', 'Mild', 'High', 'Strong', 'Yes'],
       ['Overcast', 'Hot', 'Normal', 'Weak', 'Yes'],
       ['Rain', 'Mild', 'High', 'Strong', 'No']], dtype=object)
```

```
├── Decision: Start, Value: Outlook
    ├── Decision: Sunny, Value: Humidity
        ├── Decision: High, Value: No
        ├── Decision: Normal, Value: Yes
    ├── Decision: Overcast, Value: Yes
    ├── Decision: Rain, Value: Wind
        ├── Decision: Weak, Value: Yes
        ├── Decision: Strong, Value: No
```

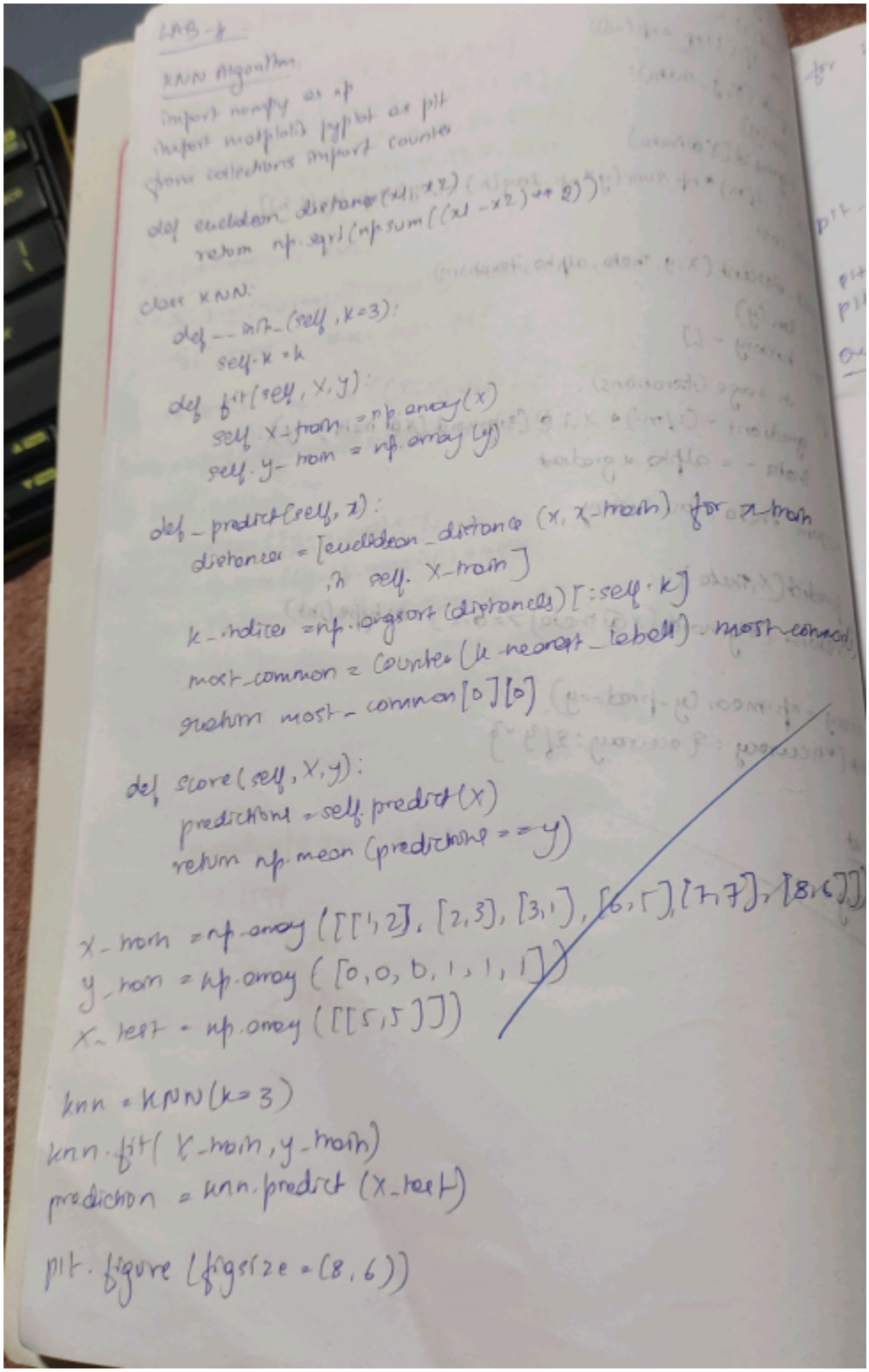
decision_tree.png

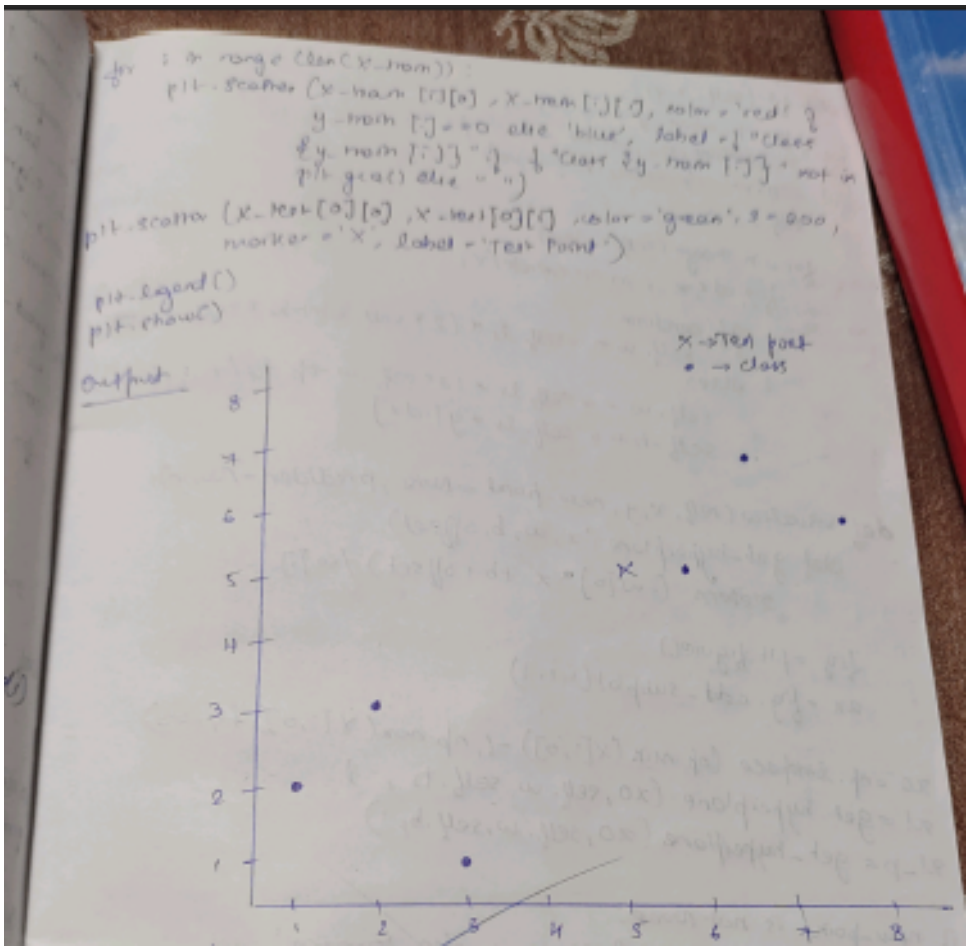


Program 6

Build KNN Classification model for a given dataset

Screenshot:





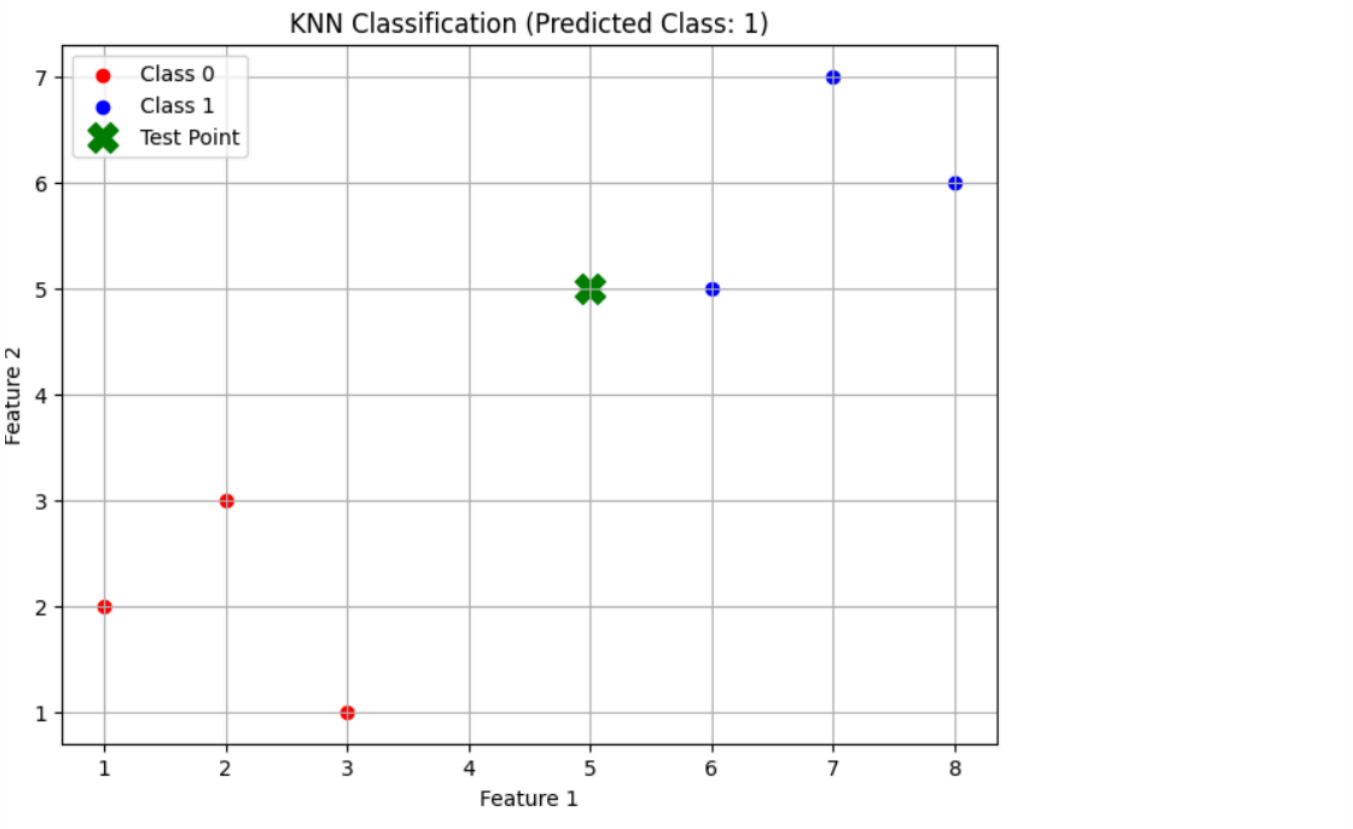
Code:

```
import numpy as np

import matplotlib.pyplot as plt

from collections import Counter


# Euclidean distance function

def euclidean_distance(x1, x2):

    return np.sqrt(np.sum((x1 - x2) ** 2))

# KNN Class

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 predict(self, X):
        return [self._predict(x) for x in X]
    def _predict(self, x):
        distances = [euclidean_distance(x, x_train) for x_train in self.X_train]
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]
    def score(self, X, y):
        predictions = self.predict(X)
        return np.mean(predictions == y)
# Sample data
X_train = np.array([[1, 2], [2, 3], [3, 1], [6, 5], [7, 7], [8, 6]])
y_train = np.array([0, 0, 0, 1, 1, 1])
X_test = np.array([[5, 5]])
# Instantiate and fit KNN
knn = KNN(k=3)
knn.fit(X_train, y_train)
prediction = knn.predict(X_test)
```

```
# Plotting

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

# Training data

for i in range(len(X_train)):

    plt.scatter(X_train[i][0], X_train[i][1],

                color='red' if y_train[i] == 0 else 'blue',

                label=f"Class {y_train[i]}" if f"Class {y_train[i]}" not in
plt.gca().get_legend_handles_labels()[1] else "")

# Test point

plt.scatter(X_test[0][0], X_test[0][1], color='green', s=200, marker='X', label='Test Point')

plt.title(f"KNN Classification (Predicted Class: {prediction[0]})")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.legend()

plt.grid(True)

plt.show()
```

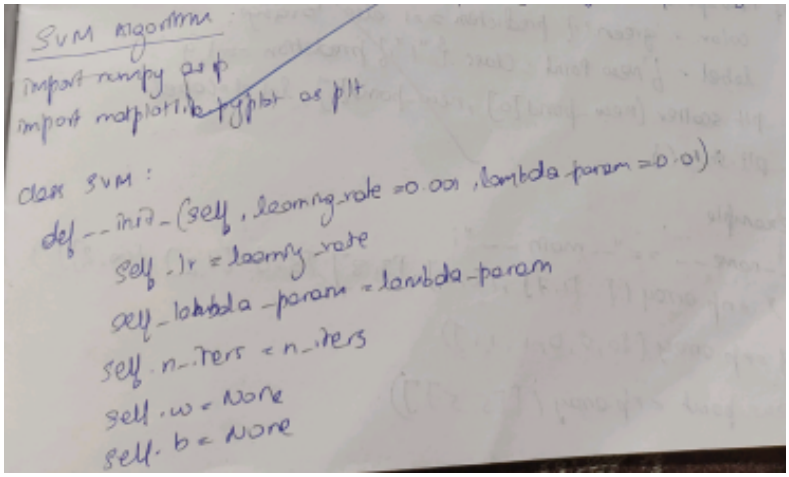
Output:

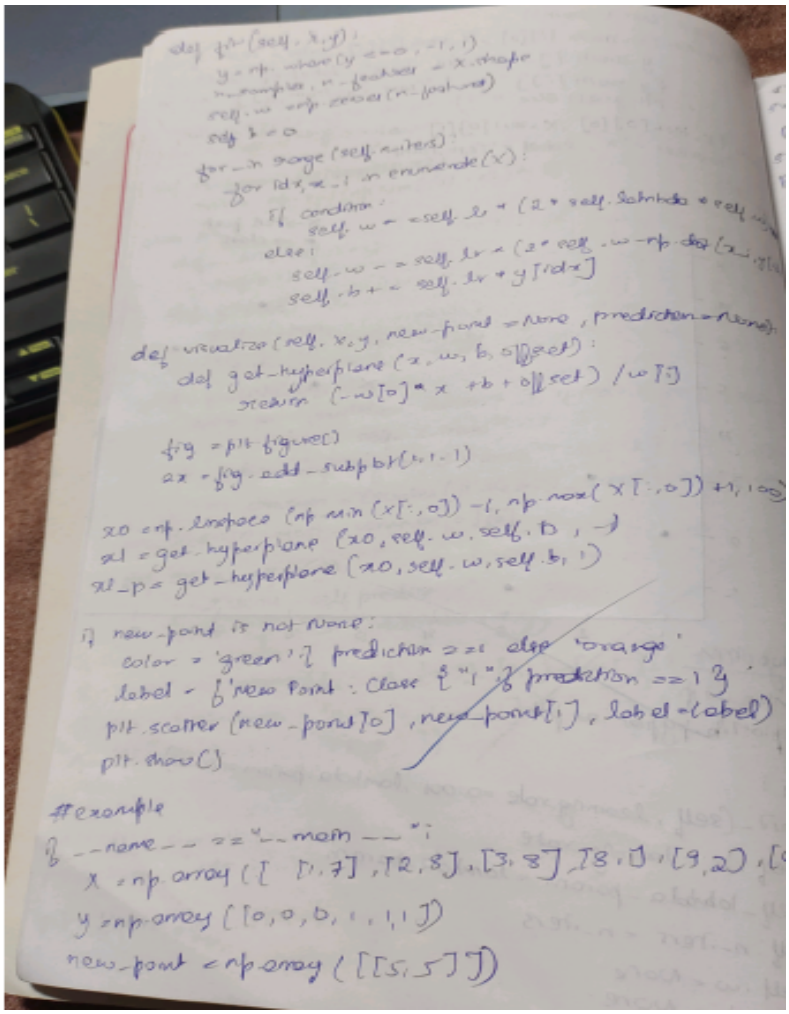

KNN Classification (Predicted Class: 1)
Class 0
Class 1
Test Point
Feature 1
Feature 2
1
2
3
4
5
6
7
8

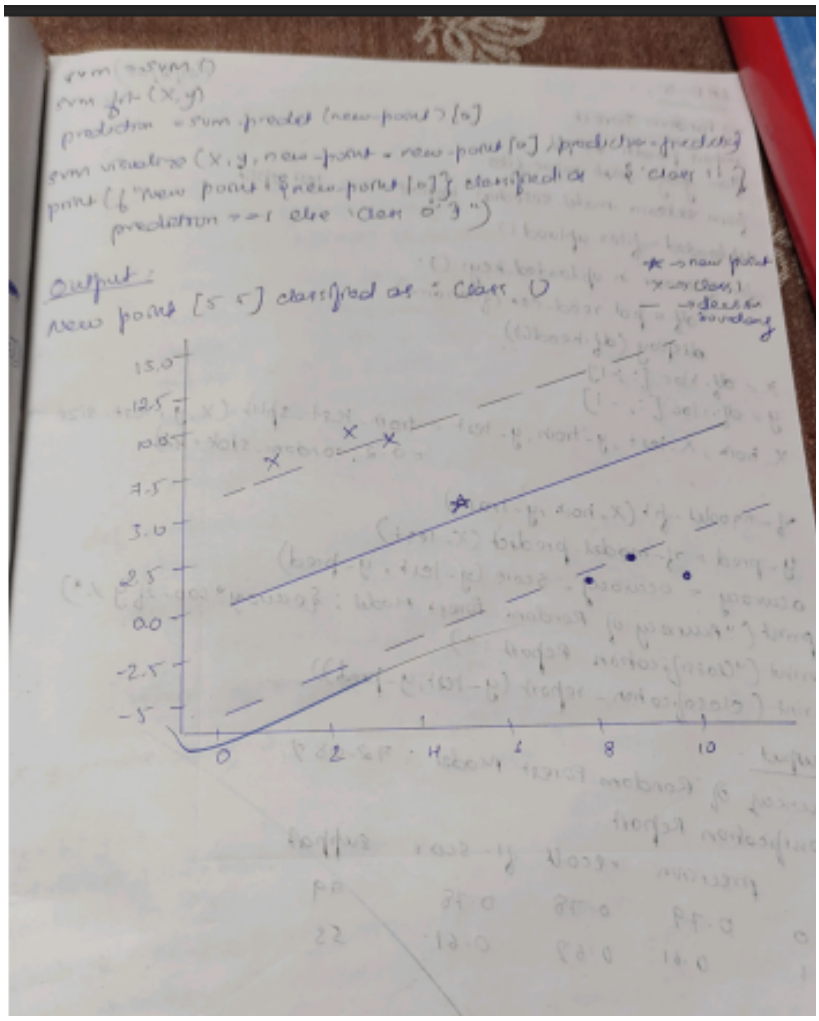
Program 7

Build Support vector machine model for a given dataset

Screenshot:







Code:

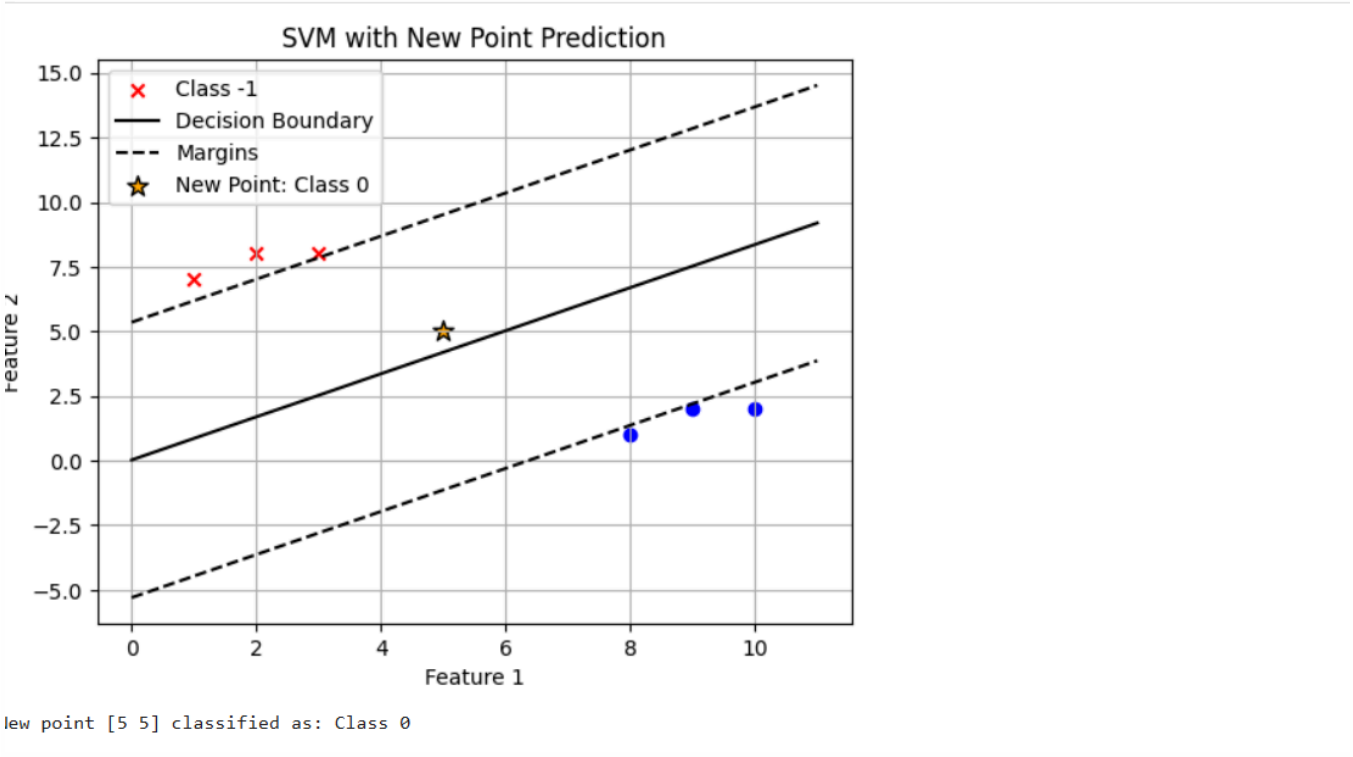
```
import numpy as np

import matplotlib.pyplot as plt


class SVM:

    def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000):

        self.lr = learning_rate

        self.lambda_param = lambda_param

        self.n_iters = n_iters
```

```
        self.w = None
        self.b = None
    def fit(self, X, y):
        y = np.where(y <= 0, -1, 1)  # Convert labels to -1 and 1
        n_samples, n_features = X.shape
        self.w = np.zeros(n_features)
        self.b = 0
        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                condition = y[idx] * (np.dot(x_i, self.w) + self.b) >= 1
                if condition:
                    self.w -= self.lr * (2 * self.lambda_param * self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y[idx]))
                    self.b += self.lr * y[idx]
    def predict(self, X):
        approx = np.dot(X, self.w) + self.b
        return np.sign(approx)
    def visualize(self, X, y, new_point=None, prediction=None):
        def get_hyperplane(x, w, b, offset):
            return (-w[0] * x + b + offset) / w[1]
        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)
```

```
        # Plot existing data points
        for i, sample in enumerate(X):
            if y[i] == 1:
                plt.scatter(sample[0], sample[1], marker='o', color='blue', label='Class +1' if i == 0 else "")
            else:
                plt.scatter(sample[0], sample[1], marker='x', color='red', label='Class -1' if i == 0 else "")
        # Plot decision boundary
        x0 = np.linspace(np.min(X[:, 0])-1, np.max(X[:, 0])+1, 100)
        x1 = get_hyperplane(x0, self.w, self.b, 0)
        x1_m = get_hyperplane(x0, self.w, self.b, -1)
        x1_p = get_hyperplane(x0, self.w, self.b, 1)
        ax.plot(x0, x1, 'k-', label='Decision Boundary')
        ax.plot(x0, x1_m, 'k--', label='Margins')
        ax.plot(x0, x1_p, 'k--')
        # Plot the new point
        if new_point is not None:
            color = 'green' if prediction == 1 else 'orange'
            label = f'New Point: Class {"1" if prediction == 1 else "0"}'
            plt.scatter(new_point[0], new_point[1], c=color, s=100, edgecolors='black', label=label,
marker='*')
        ax.legend()
        plt.xlabel("Feature 1")
        plt.ylabel("Feature 2")
        plt.title("SVM with New Point Prediction")
        plt.grid(True)
```

```
        plt.show()
# Example
if __name__ == "__main__":
    # Training data
    X = np.array([
        [1, 7],
        [2, 8],
        [3, 8],
        [8, 1],
        [9, 2],
        [10, 2]
    ])
    y = np.array([0, 0, 0, 1, 1, 1])  # 0 -> -1, 1 -> +1
    # New point to classify
    new_point = np.array([[5, 5]])
    # Train and predict
    svm = SVM()
    svm.fit(X, y)
    prediction = svm.predict(new_point)[0]
    # Visualize
    svm.visualize(X, y, new_point=new_point[0], prediction=prediction)

    # Print prediction
    print(f"New point {new_point[0]} classified as: {'Class 1' if prediction == 1 else 'Class 0'}")
```

Output:

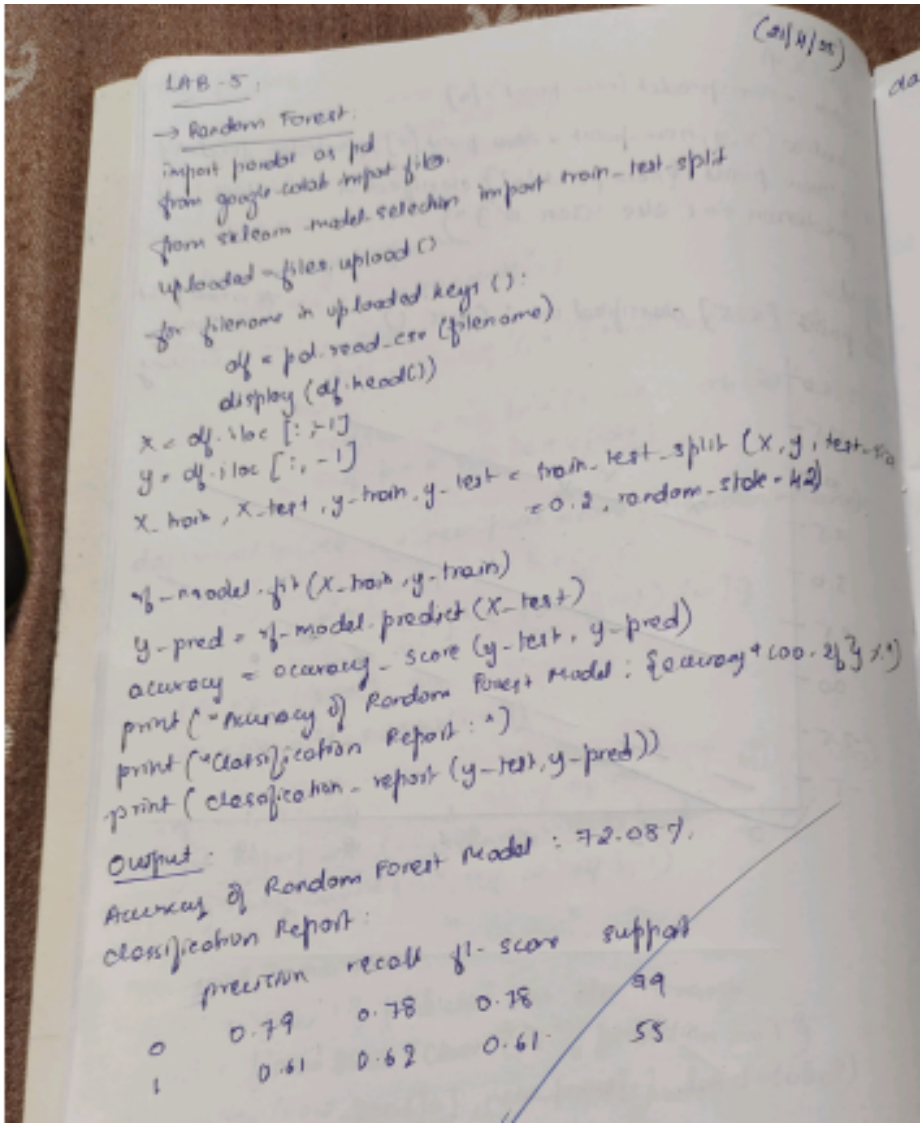



```
lew point [5 5] classified as: Class 0
```

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot:



Code:

#RANDOM FOREST

 # STEP 1: Import required libraries

import pandas as pd

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from google.colab import files

# STEP 2: Upload your dataset
uploaded = files.upload()

# STEP 3: Load the dataset (assuming it's a CSV)
for filename in uploaded.keys():
    df = pd.read_csv(filename)
    print(f"Data loaded from: {filename}")
    display(df.head())  # Display first 5 rows of data

# STEP 4: Preprocessing
# Assume the last column is the target variable (label)
X = df.iloc[:, :-1]  # Features (all rows, all columns except last)
y = df.iloc[:, -1]   # Target (last column)

# STEP 5: Split the 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)

# STEP 6: Initialize and train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)  # 100 trees in the forest
```

```
rf_model.fit(X_train, y_train)

# STEP 7: Make predictions on the test set
y_pred = rf_model.predict(X_test)

# STEP 8: Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Random Forest Model: {accuracy * 100:.2f}%")

# STEP 9: Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Output:

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving diabetes.csv to diabetes.csv
Data loaded from: diabetes.csv
```

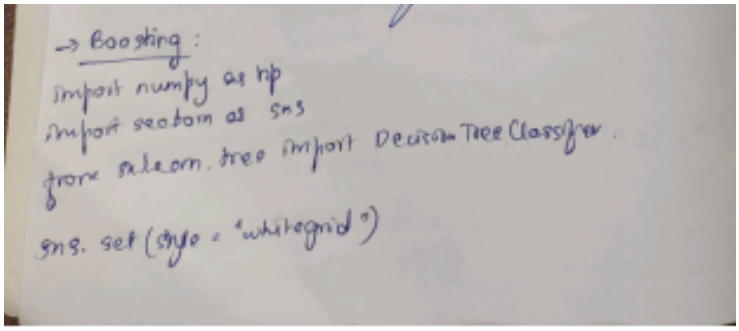
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

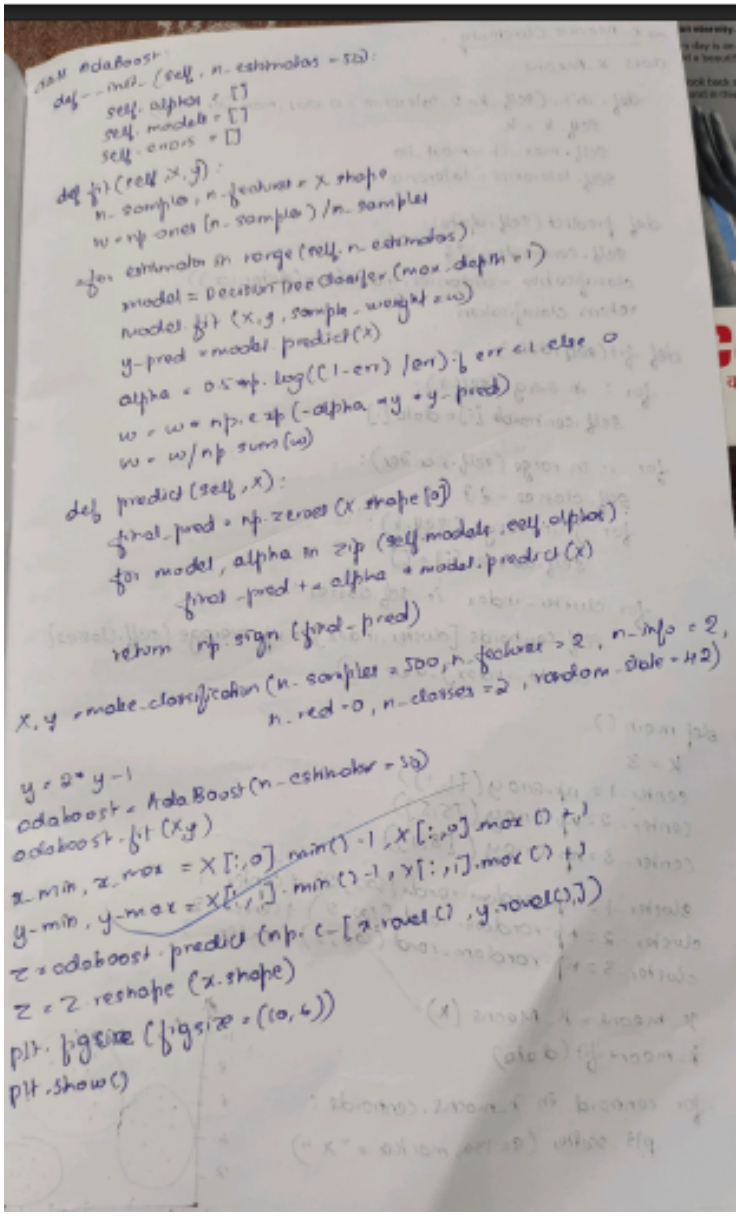
```
Accuracy of Random Forest Model: 72.08%
Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.78      0.78        99
           1       0.61      0.62      0.61        55

    accuracy                           0.72       154
   macro avg       0.70      0.70      0.70       154
weighted avg       0.72      0.72      0.72       154
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot:





Code:

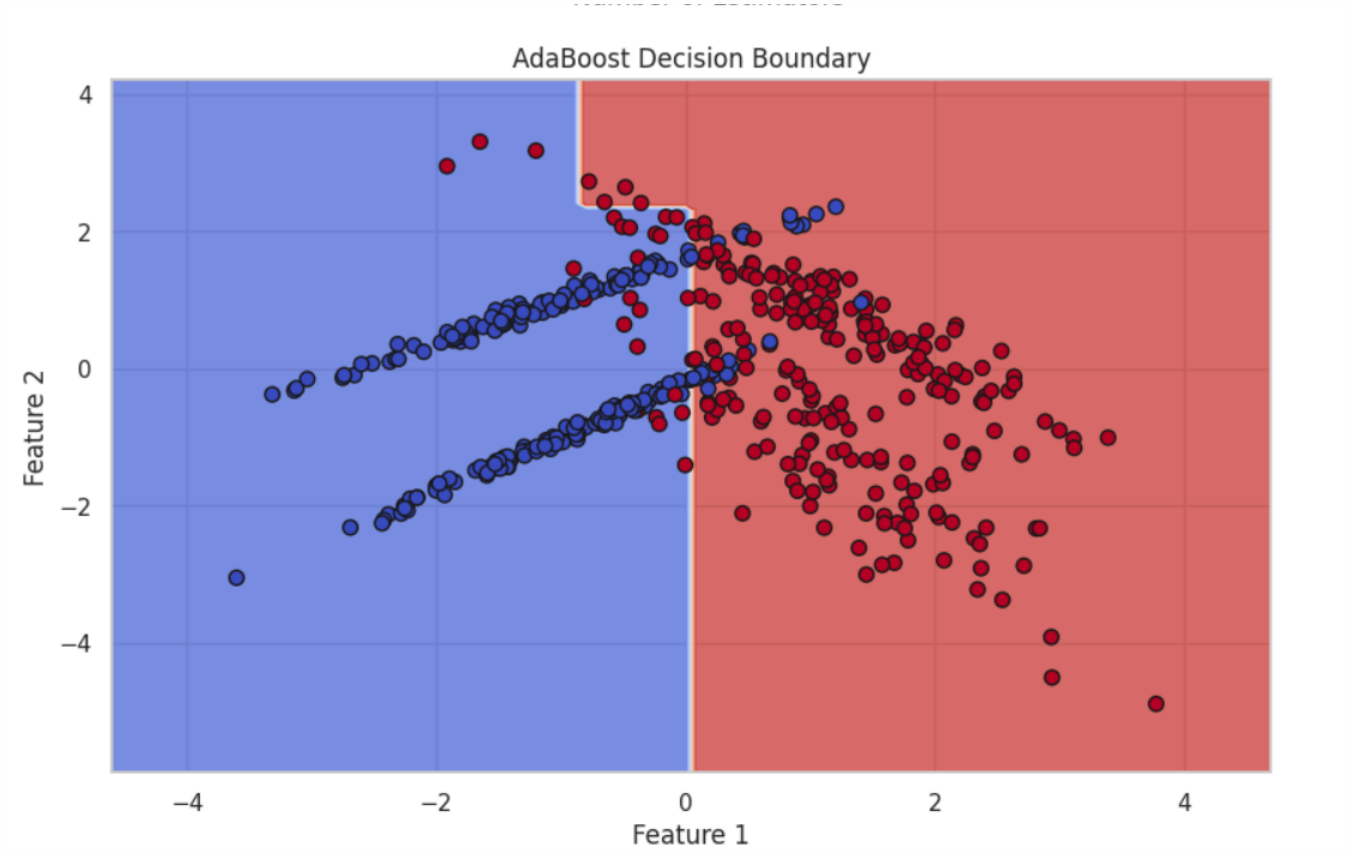
```
#BOOSTING
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA


# Set up plot style
sns.set(style="whitegrid")
class AdaBoost:
    def __init__(self, n_estimators=50):
        self.n_estimators = n_estimators
        self.alphas = []  # Weights of each weak classifier
        self.models = []  # Weak classifiers (e.g., decision stumps)
        self.errors = []  # List to store error for each estimator
    def fit(self, X, y):
        # Initialize weights for each data point
        n_samples, n_features = X.shape
        w = np.ones(n_samples) / n_samples  # Equal weights initially
        for estimator in range(self.n_estimators):
            # Train weak classifier (decision stump)
```

```
        model = DecisionTreeClassifier(max_depth=1)  # Decision stump
        model.fit(X, y, sample_weight=w)
        y_pred = model.predict(X)
        # Calculate error rate
        err = np.sum(w * (y_pred != y)) / np.sum(w)
        self.errors.append(err)
        # Compute alpha (weight for the classifier)
        alpha = 0.5 * np.log((1 - err) / err) if err < 1 else 0
        self.alphas.append(alpha)
        self.models.append(model)
        # Update weights for misclassified samples
        w = w * np.exp(-alpha * y * y_pred)  # Update weights based on classifier performance
        w = w / np.sum(w)  # Normalize the weights

    def predict(self, X):
        # Initialize the final prediction
        final_pred = np.zeros(X.shape[0])
        for model, alpha in zip(self.models, self.alphas):
            final_pred += alpha * model.predict(X)
        # Return the sign of the final prediction
        return np.sign(final_pred)
    def score(self, X, y):
        # Return accuracy of the model
        return accuracy_score(y, self.predict(X))
```

```
# Generate a synthetic binary classification dataset with 2 informative features

X, y = make_classification(n_samples=500, n_features=2, n_informative=2, n_redundant=0,
n_classes=2, random_state=42)

#Convert labels to -1 and 1 for AdaBoost

y = 2 * y - 1

# Create and train AdaBoost model

adaboost = AdaBoost(n_estimators=50)

adaboost.fit(X, y)

# Evaluate the model

accuracy = adaboost.score(X, y)

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

# Plot error over iterations

plt.figure(figsize=(10, 6))

plt.plot(range(1, adaboost.n_estimators + 1), adaboost.errors, marker='o', linestyle='-', color='b')

plt.title('Error vs. Number of Estimators')

plt.xlabel('Number of Estimators')

plt.ylabel('Error')

plt.grid(True)

plt.show()

# Plot decision boundary for final model

x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),

                     np.arange(y_min, y_max, 0.1))
```

```
Z = adaboost.predict(np.c_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.figure(figsize=(10, 6))

plt.contourf(xx, yy, Z, alpha=0.75, cmap='coolwarm')

plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o', s=50, cmap='coolwarm')

plt.title('AdaBoost Decision Boundary')

plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()
```

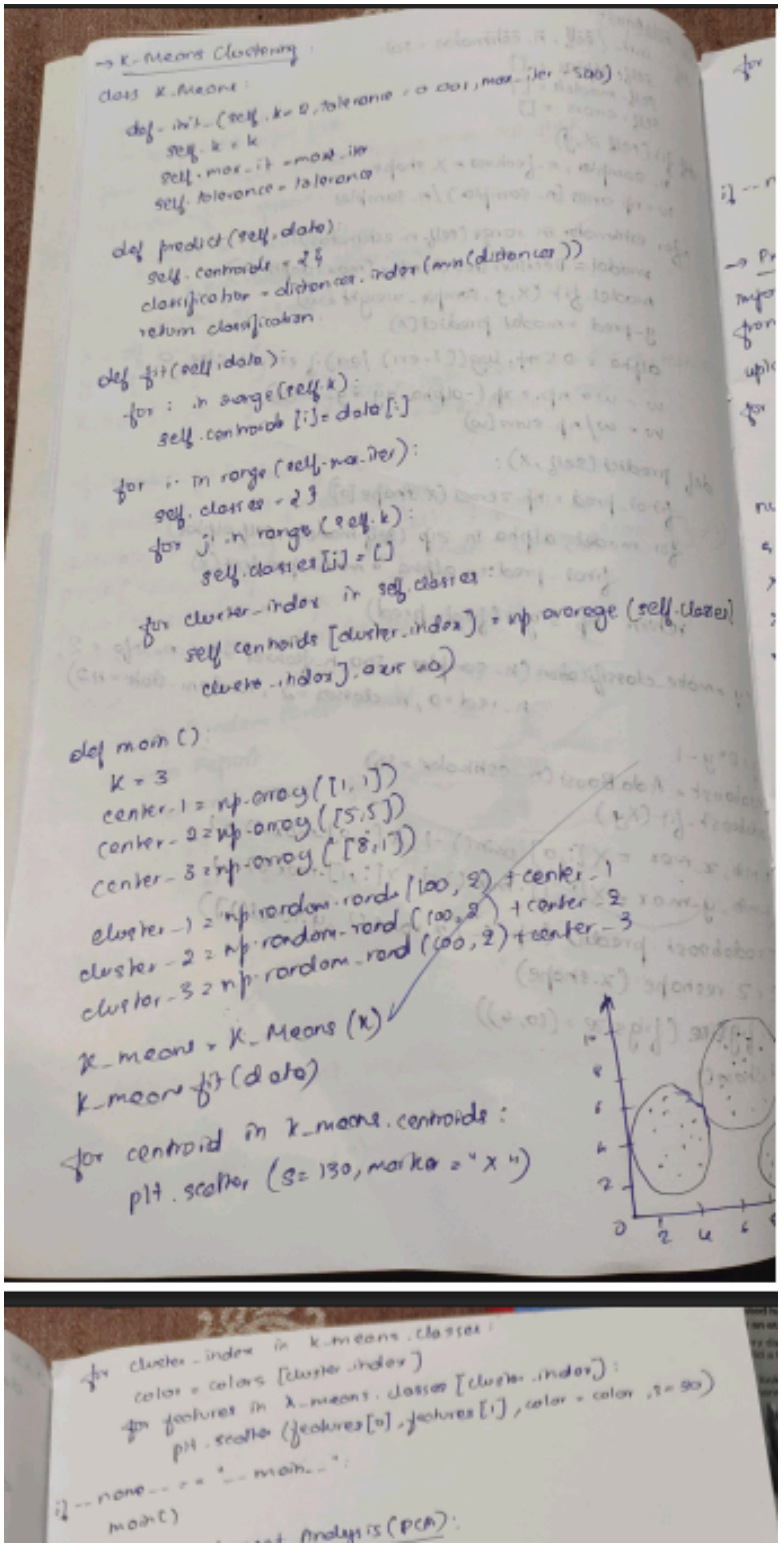
Output:



Program 10

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

Screenshot:



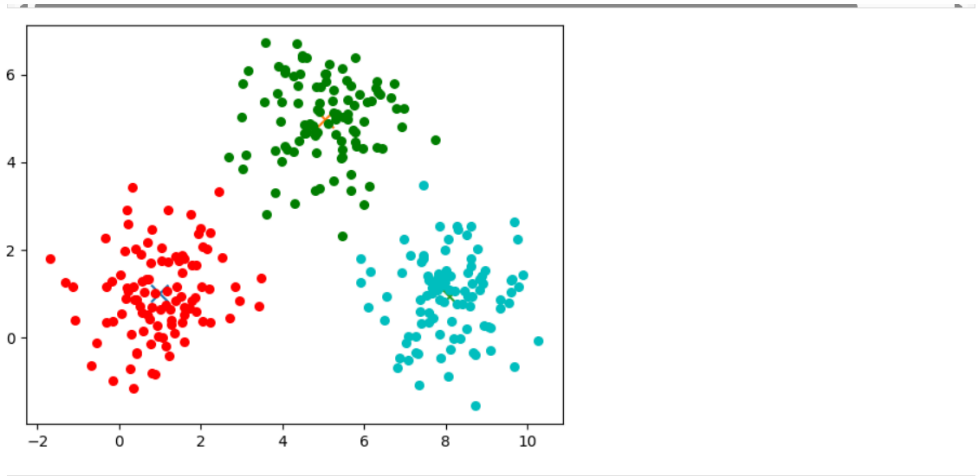
Code:

```
#K-MEANS

class K_Means:
    def __init__(self, k=2, tolerance = 0.001, max_iter = 500):
        self.k = k
        self.max_iterations = max_iter
        self.tolerance = tolerance
    def euclidean_distance(self, point1, point2):
        #return math.sqrt((point1[0]-point2[0])**2 + (point1[1]-point2[1])**2 +
(point1[2]-point2[2])**2)   #sqrt((x1-x2)^2 + (y1-y2)^2)
        return np.linalg.norm(point1-point2, axis=0)
    def predict(self,data):
        distances = [np.linalg.norm(data-self.centroids[centroid]) for centroid in self.centroids]
        classification = distances.index(min(distances))
        return classification
    def fit(self, data):
        self.centroids = {}
        for i in range(self.k):
            self.centroids[i] = data[i]
        for i in range(self.max_iterations):
            self.classes = {}
            for j in range(self.k):
                self.classes[j] = []
            for point in data:
                distances = []
```

```
            for index in self.centroids:
                distances.append(self.euclidean_distance(point,self.centroids[index]))
            cluster_index = distances.index(min(distances))
            self.classes[cluster_index].append(point)
        previous = dict(self.centroids)
        for cluster_index in self.classes:
            self.centroids[cluster_index] = np.average(self.classes[cluster_index], axis = 0)
        isOptimal = True
        for centroid in self.centroids:
            original_centroid = previous[centroid]
            curr = self.centroids[centroid]
            if np.sum((curr - original_centroid)/original_centroid * 100.0) > self.tolerance:
                isOptimal = False
        if isOptimal:
            break
def main():
    K=3
    center_1 = np.array([1,1])
    center_2 = np.array([5,5])
    center_3 = np.array([8,1])
    # Generate random data and center it to the three centers
    cluster_1 = np.random.randn(100, 2) + center_1
    cluster_2 = np.random.randn(100,2) + center_2
    cluster_3 = np.random.randn(100,2) + center_3
```

```
    data = np.concatenate((cluster_1, cluster_2, cluster_3), axis = 0)
    k_means = K_Means(K)
    k_means.fit(data)
    # Plotting starts here
    colors = 10*["r", "g", "c", "b", "k"]
    for centroid in k_means.centroids:
        plt.scatter(k_means.centroids[centroid][0], k_means.centroids[centroid][1], s = 130, marker =
"x")
    for cluster_index in k_means.classes:
        color = colors[cluster_index]
        for features in k_means.classes[cluster_index]:
            plt.scatter(features[0], features[1], color = color,s = 30)


if __name__ == "__main__":
    main()
```

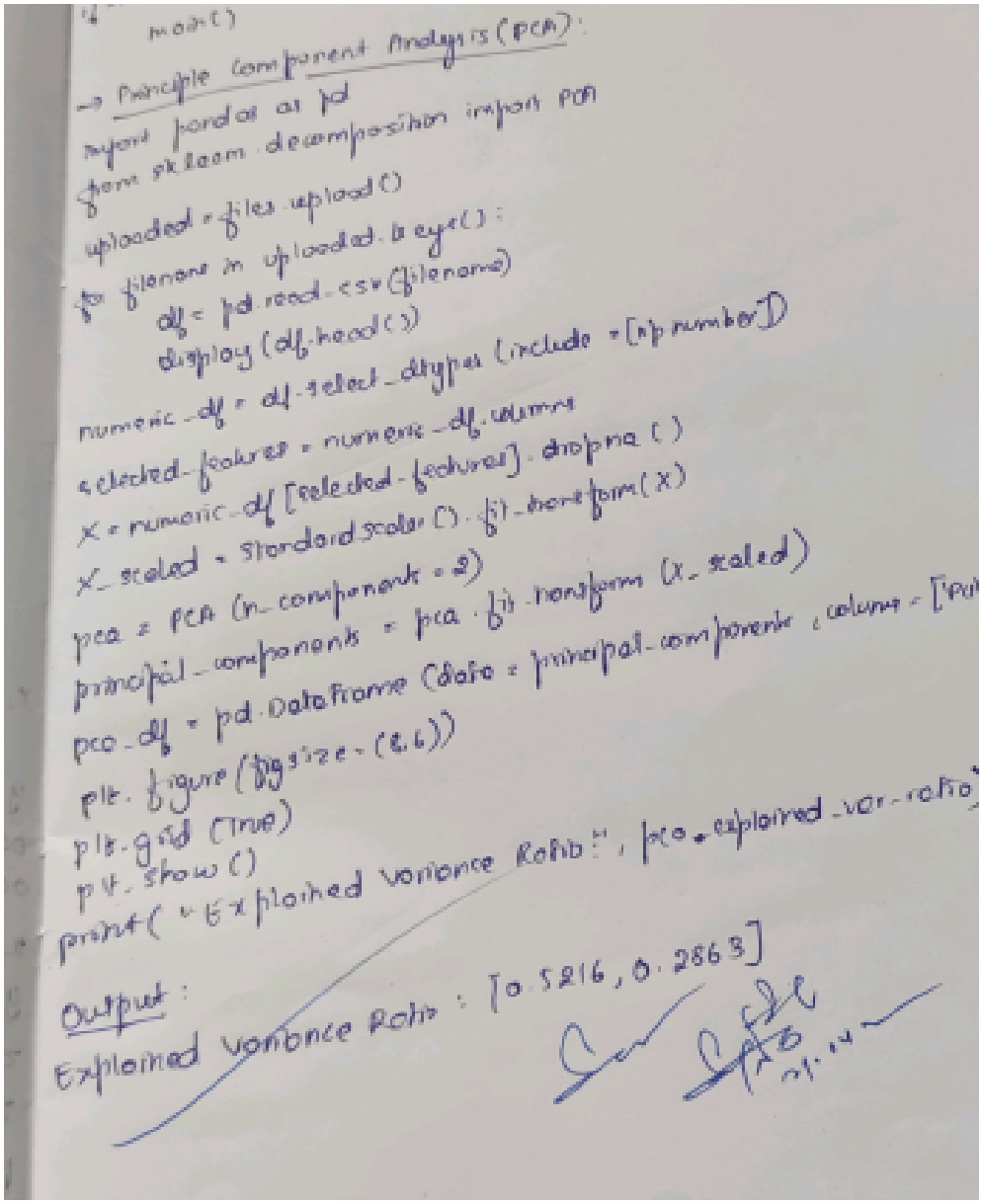
Output:



Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot:

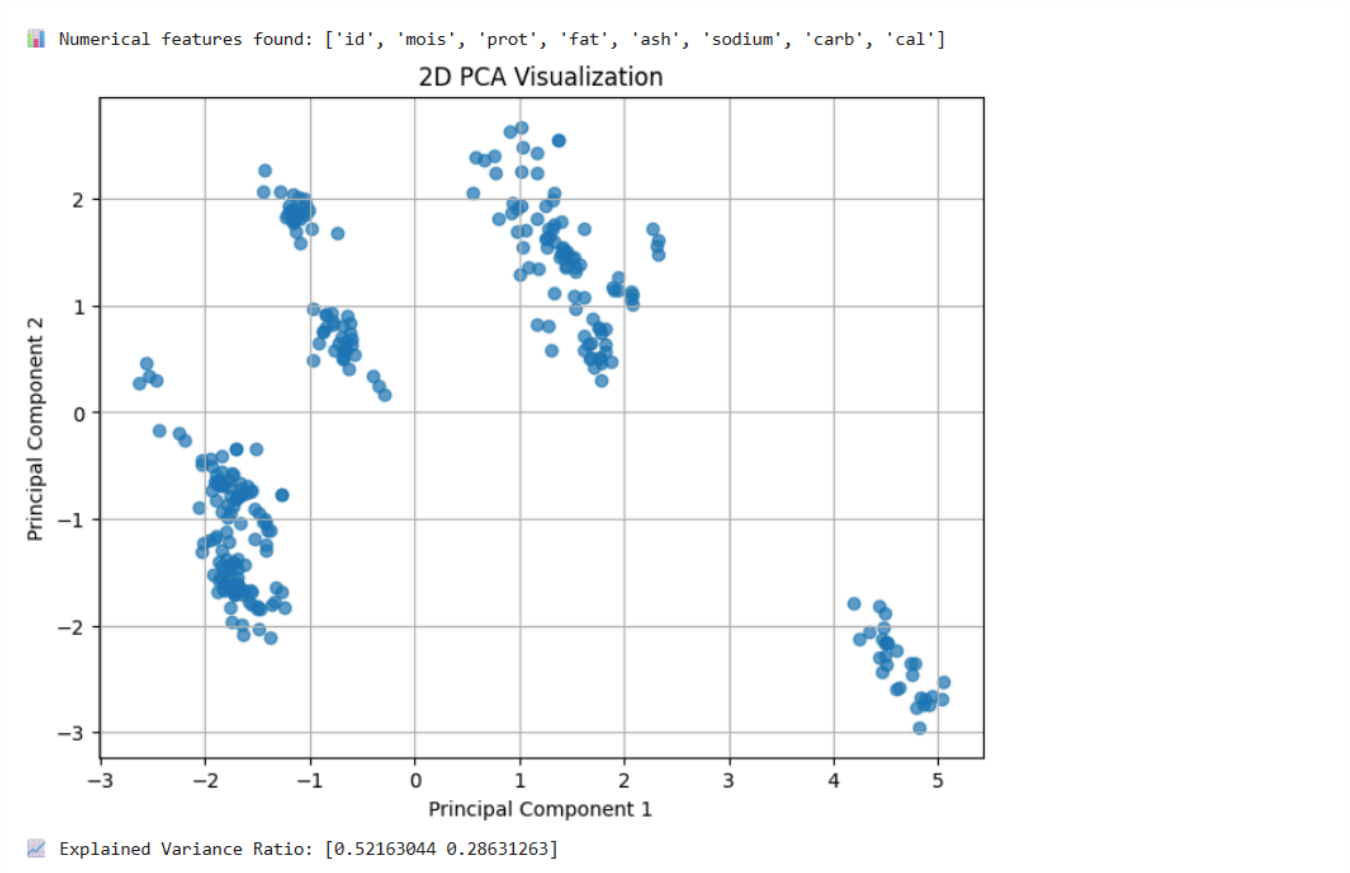


Code:

```
#PCA

# STEP 1: Import packages

import pandas as pd

import numpy as np

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

import matplotlib.pyplot as plt

from google.colab import files


# STEP 2: Upload the CSV file

uploaded = files.upload()


# STEP 3: Load the dataset

for filename in uploaded.keys():

    df = pd.read_csv(filename)

    print(f" Uploaded: {filename}")

    display(df.head())


# STEP 4: Select numerical columns

numeric_df = df.select_dtypes(include=[np.number])

print("Numerical features found:", list(numeric_df.columns))


# OPTIONAL: Manually select columns if needed
```

```
# selected_features = ['feature1', 'feature2', ...]
selected_features = numeric_df.columns  # use all numeric features for now

# STEP 5: Standardize data
X = numeric_df[selected_features].dropna()
X_scaled = StandardScaler().fit_transform(X)

# STEP 6: Apply PCA
pca = PCA(n_components=2)
principal_components = pca.fit_transform(X_scaled)

# STEP 7: Create DataFrame for components
pca_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])

# STEP 8: Visualize the first two principal components
plt.figure(figsize=(8,6))
plt.scatter(pca_df['PC1'], pca_df['PC2'], alpha=0.7)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('2D PCA Visualization')
plt.grid(True)
plt.show()
# STEP 9: Explained variance ratio
print("Explained Variance Ratio:", pca.explained_variance_ratio_)
```

Output:


Numerical features found: ['id', 'mois', 'prot', 'fat', 'ash', 'sodium', 'carb', 'cal']
2D PCA Visualization
Principal Component 2
Principal Component 1
Explained Variance Ratio: [0.52163044 0.28631263]