

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT on

Machine Learning (23CS6PCMAL)

Submitted by

Maria Sayeema (1BM22CS151)

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 **Maria Sayeema (1BM22CS151)**, 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 Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE
--	--

Index

Sl. No.	Date	Experiment Title	Page No.
1	3-3-2025	Demonstrate various data pre-processing techniques for a given dataset	3
2	3-3-2025	Demonstrate the steps to build a machine-learning model that predicts the median housing price using the California housing price dataset.	21
3	10-3-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	27
4	17-3-2025	Build Logistic Regression Model for a given dataset	39
5	24-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.	47
6	7-4-2025	Build KNN Classification model for a given dataset.	55
7	21-4-2025	Build Support vector machine model for a given dataset	61
8	5-5-2025	Implement Random forest ensemble method on a given dataset.	68
9	5-5-2025	Implement Boosting ensemble method on a given dataset.	73
10	12-5-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file.	77
11	12-5-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method.	81

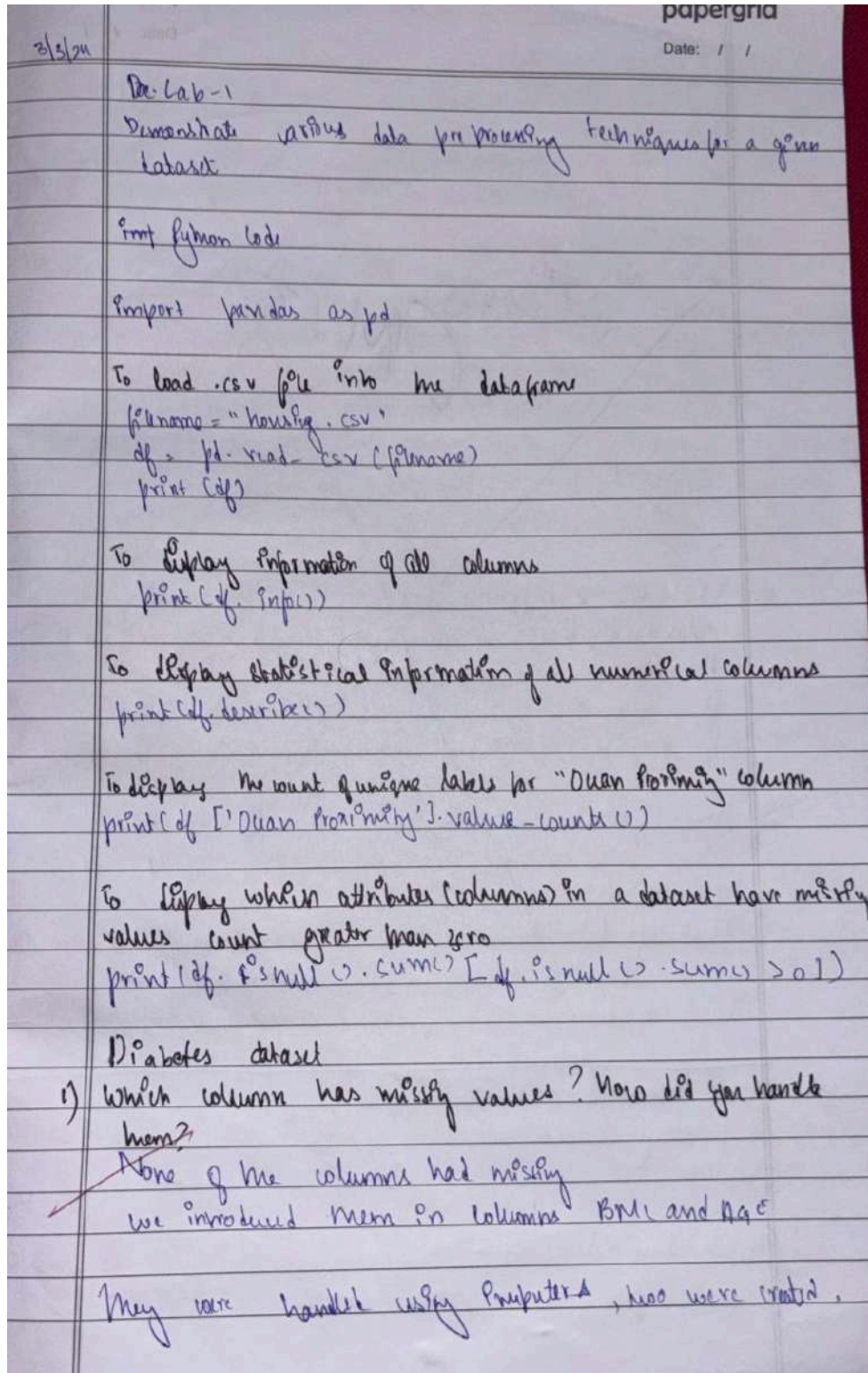
Github Link:

<https://github.com/mariasayeema/ML-LAB>

Program 1

Demonstrate various data pre-processing techniques for a given dataset

Screenshot



we had substituted median to handle missing values in Age column

and another that substituted mean to handle missing values in BMI column

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

Two categorical columns are present: Gender, CLASS. Gender was encoded using OrdinalEncoder with 'F' as 0 and 'M' as 1.

CLASS was encoded as OneHotEncoder in which each value in the column became a row i.e. CLASS-N, CLASS-P, CLASS-Y

3) What is the difference between Min-Max Scaling and Standardization? When would you use one over the other?

It was done on TG column, which forces all the values between 0 and 1

Standardization was applied for AGE column

Min-Max Scaling
keeps all the data between 0 and 1, ideal for distance based models

Standardization
works well for normally distributed data, suitable for many models
: $x - \mu / \sigma$

can distort data distribution especially with extreme outliers

sensitive to outliers
mean = 0 and SD = 1

Min-Max: distance-based models (evenly spread data and you want fixed bounds)

Standardization:

normally distributed data & regression based models or PCA where feature distribution affects output

Use standardization when there are more outliers

Adult Income data set

1) ~~the~~ which columns in the dataset had missing values?

How did you handle them

No missing values in any columns. We introduced missing values in educational-num and age

To handle these, we used two imputers, one that substitutes median to handle missing values in educational-num

and another one to handle missing values in age by substituting mean.

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

workclass, indusct, education, marital status, occupation, relationships, race, gender, capital-gain, native-country

We encoded them using two schemes:

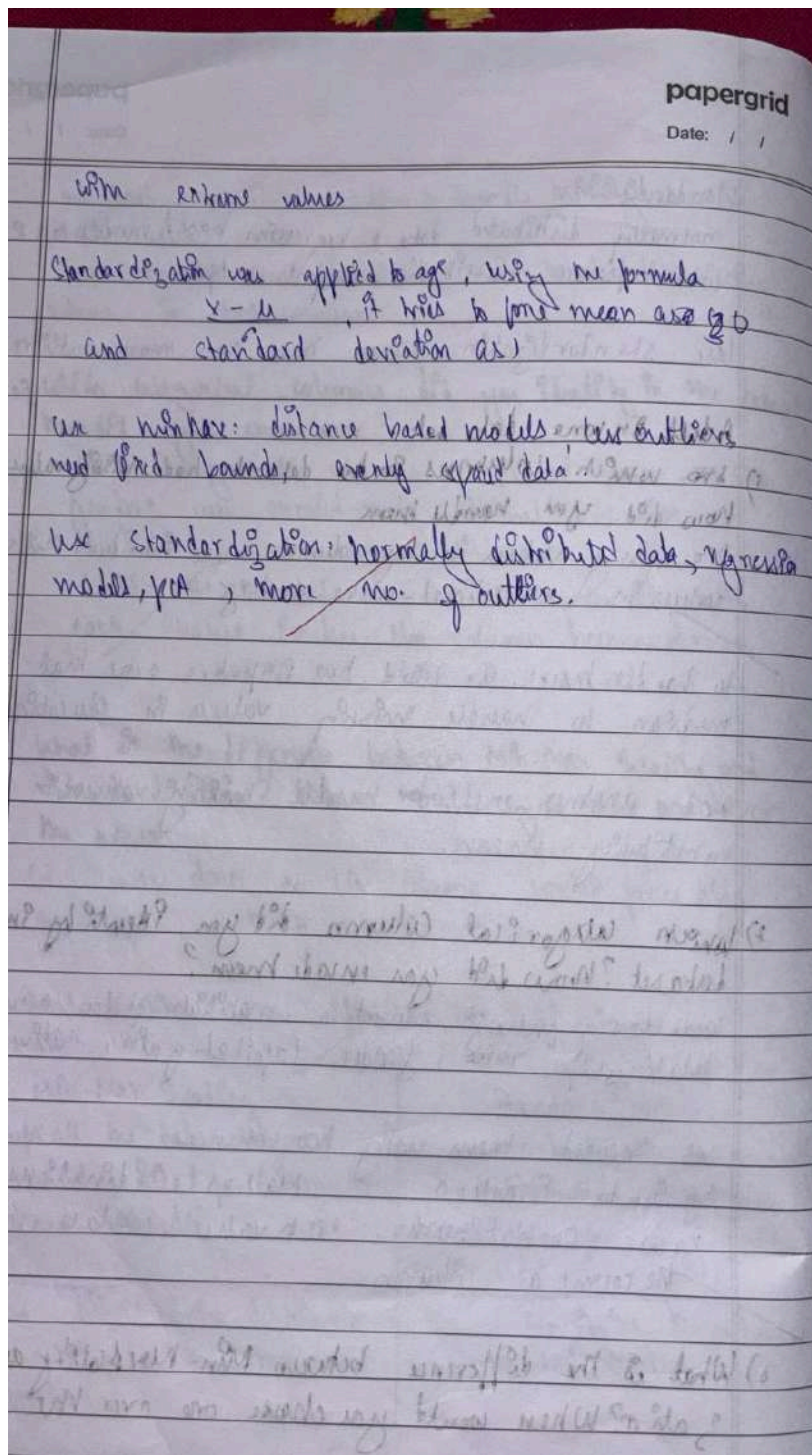
Eg: gender: female = 0 Male = 1 Ordinal encoder

race: Onehot encoder, each value i.e. black white etc.

became a column

What is the difference between Min-Max scaling and standardization? When would you choose one over the other

MinMax was applied to hours per week in which all values were forced between 0 to 1. Can distort data distribution



Code:

Data Processing :1 **Dataset of Diabetes**

from google.colab import files

uploaded = files.upload()

```

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

import pandas as pd

df = pd.read_csv('Dataset of Diabetes .csv')

df.head()

df.head(10)

df.shape

print(df.describe())

missing_values = df.isnull().sum()

print(missing_values[missing_values > 0])

import numpy as np

df.loc[5, 'AGE'] = np.nan

df.loc[10, 'BMI'] = np.nan # Set missing value for 'BMI' at row index 10

print(df.head(10))

df

print(df.describe())

missing_values = df.isnull().sum()

print(missing_values[missing_values > 0])

```



```

imputer1 = SimpleImputer(strategy="median")

imputer2 = SimpleImputer(strategy="mean")

df_copy=df

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

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

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

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

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

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

import pandas as pd

from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder

df['Gender'] = df['Gender'].str.upper() # Convert to uppercase

df['CLASS'] = df['CLASS'].str.upper()

ordinal_encoder = OrdinalEncoder(categories=[["F", "M"]]) # Encoding 'F' as 0, 'M' as 1

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

onehot_encoder = OneHotEncoder()

if 'CLASS' in df.columns:

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

    encoded_array = encoded_data.toarray()

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

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

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

else:

    df_encoded = df.copy()

```

```

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

print(df_encoded.head())

df_encoded

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

normalizer = MinMaxScaler()

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

print(df_encoded.head())

df_encoded

normalizer = MinMaxScaler()

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

df_encoded.head()

scaler = StandardScaler()

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

df_encoded.head()

df_encoded_copy1=df_encoded

df_encoded_copy2=df_encoded

df_encoded_copy3=df_encoded

Q1 = df_encoded_copy1['Chol'].quantile(0.25)

Q3 = df_encoded_copy1['Chol'].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR

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

```

```

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

print(df_encoded_copy1.head())

df_encoded_copy2['Chol_zscore'] = stats.zscore(df_encoded_copy2['Chol'])

df_encoded_copy2['Chol'] = np.where(df_encoded_copy2['Chol_zscore'].abs() > 3, np.nan,
df_encoded_copy2['Chol'])

print(df_encoded_copy2.head())

df_encoded_copy3['Chol_zscore'] = stats.zscore(df_encoded_copy3['Chol'])

median_salary = df_encoded_copy3['Chol'].median()

df_encoded_copy3['Chol'] = np.where(df_encoded_copy3['Chol'].abs() > 3, median_salary,
df_encoded_copy3['Chol'])

print(df_encoded_copy3.head())

```

2 Adult:

```

from google.colab import files

uploaded = files.upload()

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

import pandas as pd

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

```

```

df.head()

df.head(10)

df.shape

print(df.describe())

missing_values = df.isnull().sum()

print(missing_values[missing_values > 0])

import numpy as np

df.loc[5, 'educational-num'] = np.nan

df.loc[7, 'age'] = np.nan

print(df.head(10))

df

print(df.describe())

missing_values = df.isnull().sum()

print(missing_values[missing_values > 0])

imputer1 = SimpleImputer(strategy="median")

imputer2 = SimpleImputer(strategy="mean")

df_copy = df

imputer1.fit(df_copy[["educational-num"]])

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

df_copy["educational-num"] = imputer1.transform(df[["educational-num"]])

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

print(df_copy["educational-num"].isnull().sum())

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

```



```

import pandas as pd

from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder

df['gender'] = df['gender'].str.upper()

df['race'] = df['race'].str.upper()

ordinal_encoder = OrdinalEncoder(categories=[["FEMALE", "MALE"]])

df["gender_Encoded"] = ordinal_encoder.fit_transform(df[["gender"]])

onehot_encoder = OneHotEncoder()

if 'race' in df.columns:

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

    encoded_array = encoded_data.toarray()

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

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

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

else:

    df_encoded = df.copy()

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

print(df_encoded.head())

df_encoded

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

normalizer = MinMaxScaler()

df_encoded[["hours-per-week"]] = normalizer.fit_transform(df_encoded[["hours-per-week"]])

print(df_encoded.head())

df_encoded

```

```

scaler = StandardScaler()

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

df_encoded.head()

df_encoded_copy1 = df_encoded

df_encoded_copy2 = df_encoded

df_encoded_copy3 = df_encoded

Q1 = df_encoded_copy1['fnlwgt'].quantile(0.25)

Q3 = df_encoded_copy1['fnlwgt'].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR

df_encoded_copy1['fnlwgt'] = np.where(

    df_encoded_copy1['fnlwgt'] > upper_bound,

    upper_bound,

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

)

print(df_encoded_copy1.head())

df_encoded_copy2['fnlwgt_zscore'] = stats.zscore(df_encoded_copy2['fnlwgt'])

df_encoded_copy2['fnlwgt'] = np.where(df_encoded_copy2['fnlwgt_zscore'].abs() > 3, np.nan,
df_encoded_copy2['fnlwgt'])

print(df_encoded_copy2.head())

df_encoded_copy3['fnlwgt_zscore'] = stats.zscore(df_encoded_copy3['fnlwgt'])

median_salary = df_encoded_copy3['fnlwgt'].median()

df_encoded_copy3['fnlwgt'] = np.where(df_encoded_copy3['fnlwgt_zscore'].abs() > 3,
median_salary, df_encoded_copy3['fnlwgt'])

```

```
print(df_encoded_copy3.head())
```

Preprocessing:

```
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
```

```
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)
```

```
df.shape
```

```

df.loc[5, 'Age'] = np.nan

df.loc[10, 'Salary'] = np.nan

df.head(10)

print(df.info())

print(df.describe())

missing_values = df.isnull().sum()

print(missing_values[missing_values > 0])

imputer1 = SimpleImputer(strategy="median")

imputer2 = SimpleImputer(strategy="mean")

df_copy = df

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

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

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

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

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

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

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

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

onehot_encoder = OneHotEncoder()

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

encoded_array = encoded_data.toarray()

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())

normalizer = MinMaxScaler()

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

df_encoded.head()

scaler = StandardScaler()

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

df_encoded.head()

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'])

)

print(df_encoded_copy1.head())

df_encoded_copy2['Salary_zscore'] = stats.zscore(df_encoded_copy2['Salary'])

```

```

df_encoded_copy2['Salary'] = np.where(df_encoded_copy2['Salary_zscore'].abs() > 3, np.nan,
df_encoded_copy2['Salary'])

print(df_encoded_copy2.head())

df_encoded_copy3['Salary_zscore'] = stats.zscore(df_encoded_copy3['Salary'])

median_salary = df_encoded_copy3['Salary'].median()

df_encoded_copy3['Salary'] = np.where(df_encoded_copy3['Salary_zscore'].abs() > 3,
median_salary, df_encoded_copy3['Salary'])

print(df_encoded_copy3.head())

```

Preprocessing:

```

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

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.shape

df.loc[5, 'Age'] = np.nan

df.loc[10, 'Salary'] = np.nan

df.head(10)

print(df.info())

print(df.describe())

missing_values = df.isnull().sum()

print(missing_values[missing_values > 0])

imputer1 = SimpleImputer(strategy="median")

imputer2 = SimpleImputer(strategy="mean")

df_copy = df

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

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

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

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

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

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

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

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

onehot_encoder = OneHotEncoder()

```

```

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

encoded_array = encoded_data.toarray()

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())

normalizer = MinMaxScaler()

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

df_encoded.head()

scaler = StandardScaler()

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

df_encoded.head()

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"]))

```



```
print(df_encoded_copy1.head())

df_encoded_copy2['Salary_zscore'] = stats.zscore(df_encoded_copy2['Salary'])

df_encoded_copy2['Salary'] = np.where(df_encoded_copy2['Salary_zscore'].abs() > 3, np.nan,
df_encoded_copy2['Salary'])

print(df_encoded_copy2.head())

df_encoded_copy3['Salary_zscore'] = stats.zscore(df_encoded_copy3['Salary'])

median_salary = df_encoded_copy3['Salary'].median()

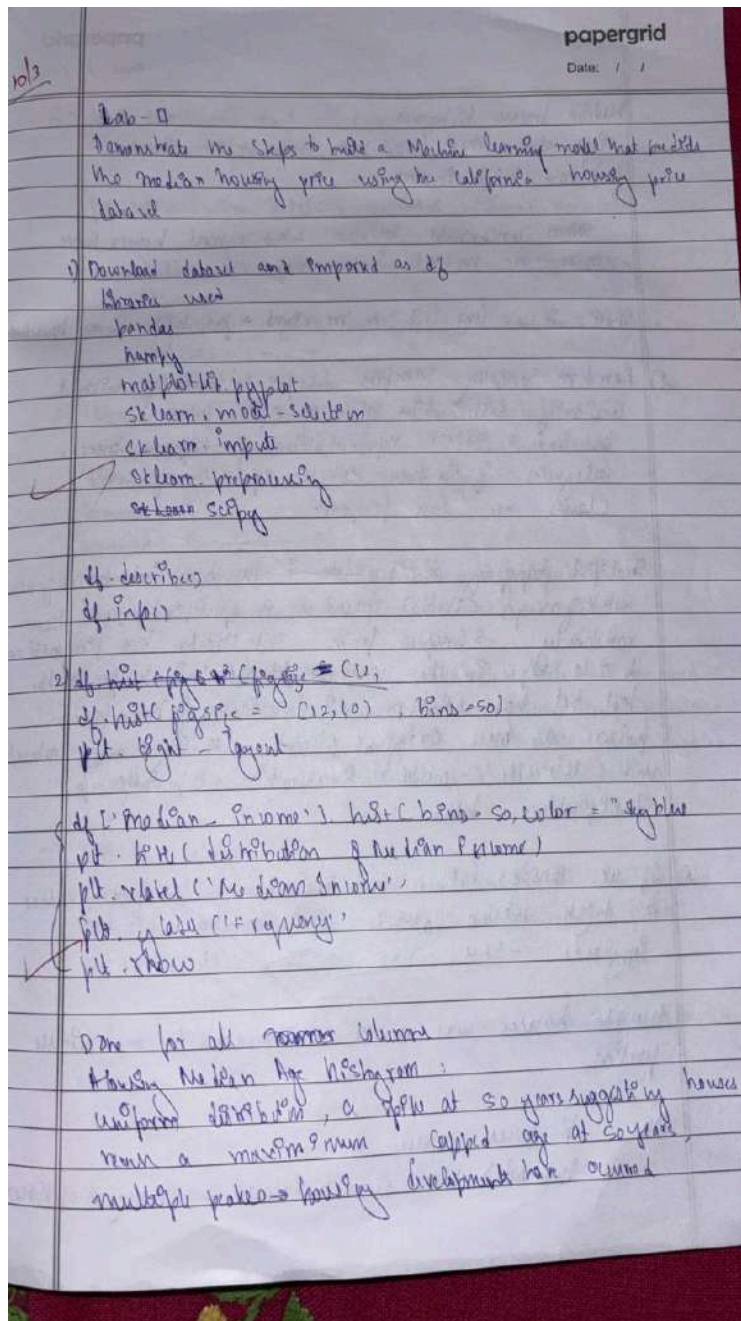
df_encoded_copy3['Salary'] = np.where(df_encoded_copy3['Salary_zscore'].abs() > 3,
median_salary, df_encoded_copy3['Salary'])

print(df_encoded_copy3.head())
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot:



Median Income histogram:

Right skew: most houses have lower median income
few have high income

Values concentrated between 2 to 6: most houses have
lower - to - moderate income range

There is a long tail on the right & few higher income households

- 2) Random Sampling: samples data points randomly without
considering distribution of a specific feature
can lead to uneven representation of different classes or
categories if in the test set especially if some
classes are less frequent

Stratified Sampling: splits data into homogeneous
subgroups (strata) based on a specific feature then
randomly samples from each strata in proportion to
its size in the overall dataset this helps to the
test set has similar distribution of the chosen
feature as the original dataset. leading to more robust
and reliable model evaluation especially for
classification tasks

- 3) If we consider latitude and longitude: median house value
is high between latitude 36, 37 and
longitude -124, -122, -120, -118

most houses are concentrated in the middle
portion

The house value is

Inland > near ocean > Island > near bay > ocean

5) total rooms and total bedrooms with
total bedrooms, population, household

population with total-rooms, total-bedrooms, household

household with total-rooms, total-bedrooms, population

most correlated with housing price = median-income
correlation = 0.865

6) rooms per household = total-rooms / households

bedrooms per household = total-bedrooms / households

rooms per person = total-rooms / population

income per

household income = median

income per person = median-income / population

household-income = median-income * households

Most of the data points are concentrated below \$ as median
income which means most house prices. median house
values are associated with low to moderate median
income, there is a large positive correlation

7) Missing values:

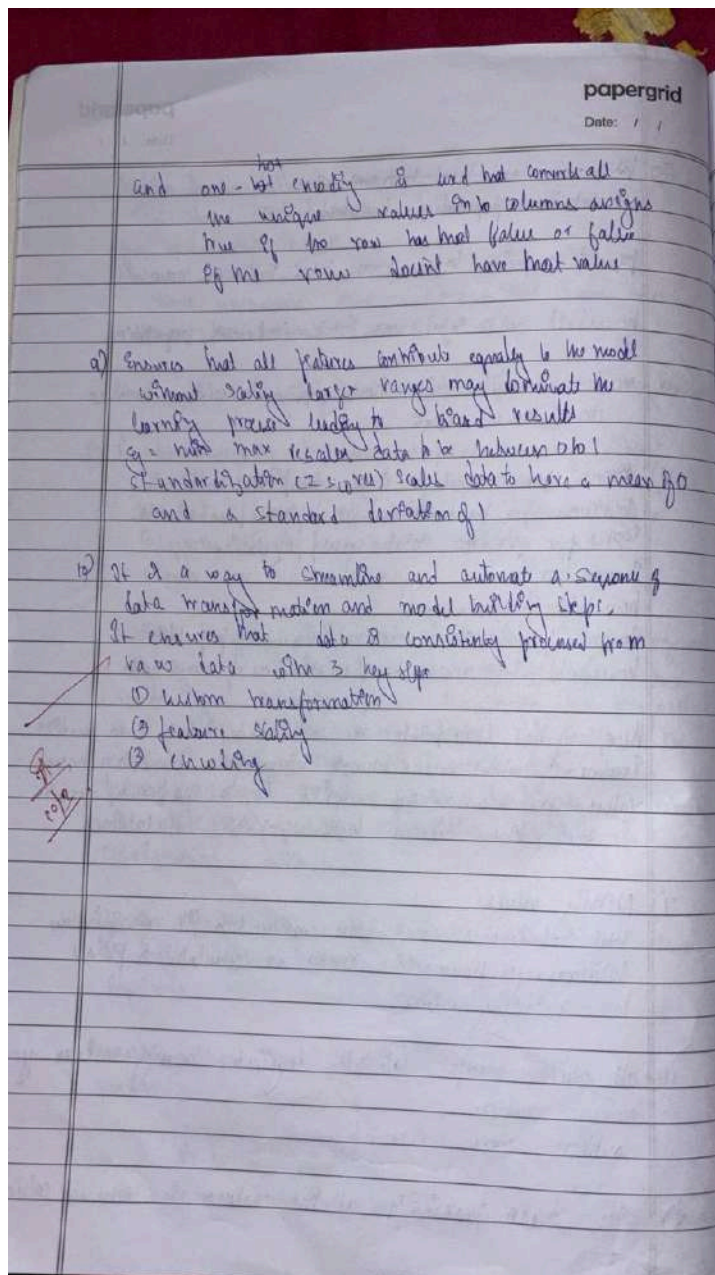
total bedrooms → median was used to fill missing values

bedrooms per household: mean was calculated to fill all
the missing values

all columns except latitude, longitude, housing-median-age
have outliers

outliers were removed

8) Yes upon processing all the values are taken as columns



Code:

```
import pandas as pd

from sklearn.base import BaseEstimator, TransformerMixin

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.pipeline import Pipeline
```

```

from sklearn.impute import SimpleImputer

df.columns = df.columns.str.strip()

class CustomFeatureTransformer(BaseEstimator, TransformerMixin):

    def fit(self, X, y=None):

        return self

    def transform(self, X):

        X['rooms_per_household'] = X['total_rooms'] / X['households']

        return X

numerical_features = ['housing_median_age', 'total_rooms', 'total_bedrooms', 'population',
'households', 'median_income', 'median_house_value']

categorical_features = ['ocean_proximity']

preprocessor = ColumnTransformer([

    ('num', Pipeline([

        ('imputer', SimpleImputer(strategy='median')),

        ('scaler', StandardScaler())

    ]), numerical_features),

    ('cat', Pipeline([

        ('imputer', SimpleImputer(strategy='most_frequent')),

        ('encoder', OneHotEncoder(handle_unknown='ignore'))

    ]), categorical_features)

])

pipeline = Pipeline([

    ('custom', CustomFeatureTransformer()),

    ('preprocessor', preprocessor)

])

```

```

processed_data = pipeline.fit_transform(df)

num_cols = numerical_features + ['rooms_per_household']

cat_cols =
pipeline.named_steps['preprocessor'].transformers_[1][1].named_steps['encoder'].get_feature_names_
_out(categorical_features)

all_columns = num_cols + list(cat_cols)

if processed_data.shape[1] == len(all_columns):

    processed_df = pd.DataFrame(processed_data, columns=all_columns)

else:

    print(f"Mismatch in number of columns: processed data has {processed_data.shape[1]} columns,
expected {len(all_columns)}.")

    print("Adjusted column names:", all_columns)

```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot

17/03/24 18-03: Implement Linear and multi-linear regression on the given data set

papergrid
Date: / /

x_i (area) y_i (Sales in thousands)

1	2
2	4
3	5
4	9

Linear Regression

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} \quad Y = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix}$$
$$X^T Y = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix} \begin{pmatrix} 2 \\ 4 \\ 5 \\ 9 \end{pmatrix} = \begin{pmatrix} 4 & 10 \\ 10 & 30 \end{pmatrix}$$
$$(X^T X)^{-1} = \begin{pmatrix} 4 & 10 \\ 10 & 30 \end{pmatrix}^{-1} = \begin{pmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{pmatrix}$$
$$(X^T X)^{-1} X^T = \begin{pmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.5 & -0.1 & 0.1 & 0.3 \end{pmatrix}$$
$$(X^T X)^{-1} X^T Y = \begin{pmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.5 & -0.1 & 0.1 & 0.3 \end{pmatrix} \begin{pmatrix} 2 \\ 4 \\ 5 \\ 9 \end{pmatrix} = \begin{pmatrix} -1.5 \\ 2.2 \end{pmatrix}$$

$$y = 9.5 + 2.2x$$

The predicted sales when $x=5$ is

$$y = 9.5 + 2.2 \times 5 = 20.5$$

Diameter (x) in inches	Price (y) in dollars	x^2	xy
8	10	64	80
10	13	100	130
12	16	144	192
Sum	39	308	402
mean	13	102.67	134

$$b_1 = \frac{(134) - (10)(39)}{102.67 - (10)^2} = 1.5$$

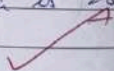
$$b_0 = 13 - (1.5) \times 10 = -2$$

$$\text{At } x = 20$$

$$y = -2 + (1.5 \times 20)$$

$$= -2 + 30 = 28$$

Price of 20 inch pizza is 28



1) (QATAR) A - per - CAPITAL - income.csv

① No missing values
no need for handling missing values

② Scaling/encoding → not necessary as features are in appropriate numerical format for linear regression. There were no categorical variables that required encoding.

2) Salary.csv:

① Yes: 2 missing values in Years Experience, filled them with mean as they are numerical values

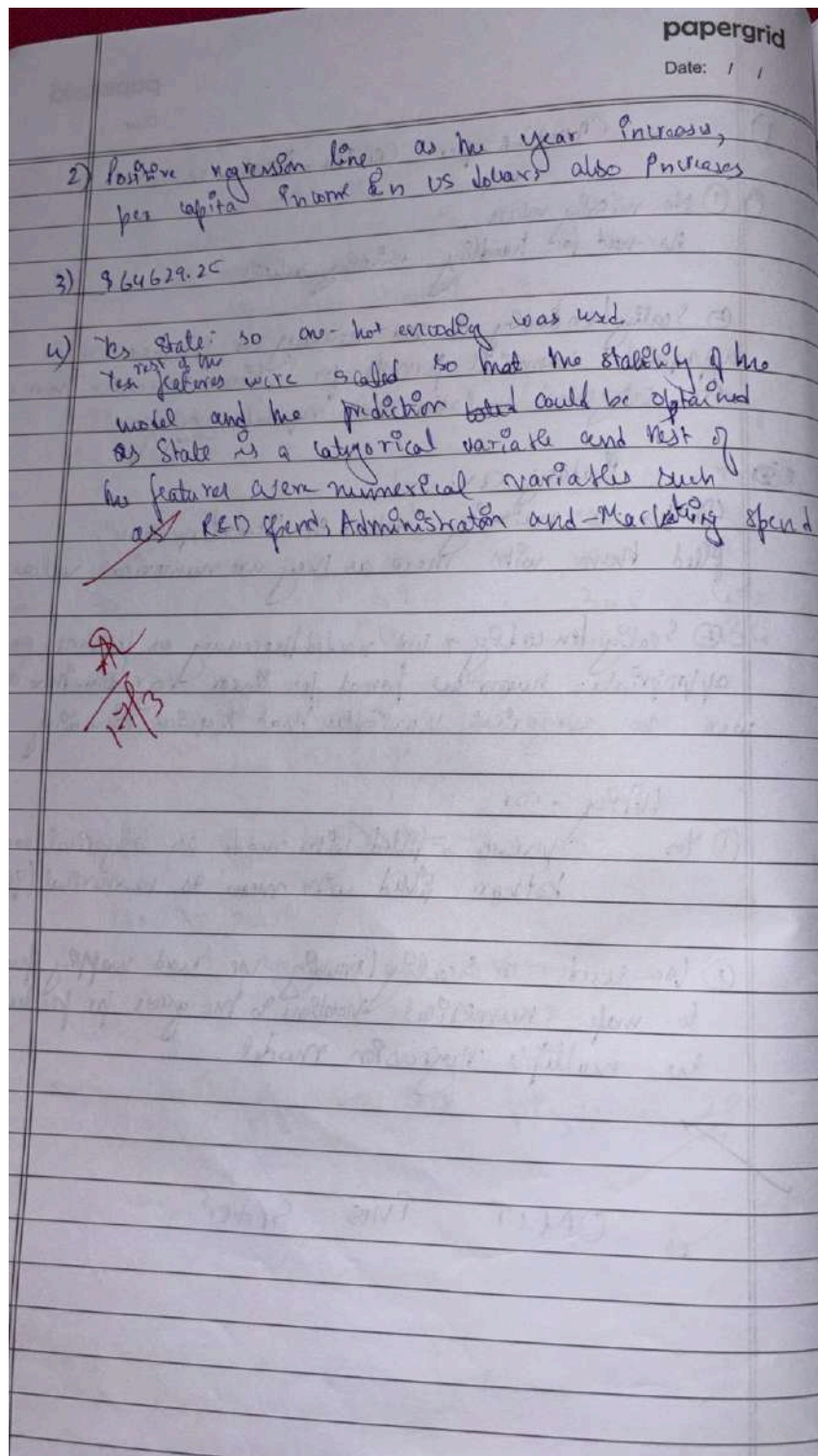
② Scaling/encoding → not needed/necessary as features are in appropriate numerical format for linear regression. There were no categorical variables that required encoding.

Wipac.csv

① Yes, Experience → filled with mode as categorical and kstcare → filled with mean as numerical (quantitative)

② We tried to scaling/encoding: we used mapping function to map numerical notations to the years for producing the multiple regression model.

ON IT THIS SPACE



Code:

Linear Regression:

HOUSING:

```

import pandas as pd

import numpy as np

from sklearn import linear_model

import matplotlib.pyplot as plt

from google.colab import files

uploaded = files.upload()

import pandas as pd

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

df.head()

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

reg = linear_model.LinearRegression()

reg.fit(new_df, price)

reg.predict([[3300]])

reg.coef_

reg.intercept_

reg = linear_model.LinearRegression()

reg.fit(new_df, price)

reg.predict([[3300]])

```

```
reg.coef_
```

```
reg.intercept_
```

```
3300 * 135.78767123 + 180616.43835616432
```

```
reg.predict([[5000]])
```

SALARY

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn import linear_model
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.linear_model import LinearRegression
```

```
from sklearn.model_selection import train_test_split
```

```
from google.colab import files
```

```
uploaded = files.upload()
```

```
df_salary = pd.read_csv('salary.csv')
```

```
print(df_salary.head())
```

```
print(df_salary.isnull().sum())
```

```
mean_years_experience = df_salary['YearsExperience'].mean()
```

```
df_salary['YearsExperience'].fillna(mean_years_experience, inplace=True)
```

```
print(df_salary.isnull().sum())
```

```
plt.scatter(df_salary['YearsExperience'], df_salary['Salary'], color='blue')
```

```
plt.xlabel('Years of Experience')
```

```
plt.ylabel('Salary')
```

```
plt.title('Years of Experience vs Salary')
```

```

plt.show()

X_salary = df_salary['YearsExperience'].values.reshape(-1, 1) # Independent variable (Years of
Experience)

y_salary = df_salary['Salary'].values # Dependent variable (Salary)

X_train, X_test, y_train, y_test = train_test_split(X_salary, y_salary, test_size=0.2,
random_state=42)

salary_model = LinearRegression()

salary_model.fit(X_train, y_train)

predicted_salary_12_years = salary_model.predict([[12]])

print(f'Predicted salary for 12 years of experience: ${predicted_salary_12_years[0]:.2f}')

plt.scatter(df_salary['YearsExperience'], df_salary['Salary'], color='blue')

plt.plot(df_salary['YearsExperience'], salary_model.predict(X_salary), color='red') # Regression line

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.title('Years of Experience vs Salary with Regression Line')

plt.show()

```

CANADA_PER_CAPITA_INCOME:

```

import pandas as pd

import numpy as np

from sklearn import linear_model

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from google.colab import files

```

```

uploaded = files.upload()

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

print(df.head())

print(df.isnull().sum())

df.dropna(inplace=True)

df.shape

plt.scatter(df['year'], df['per capita income (US$)'], color='blue')

plt.xlabel('Year')

plt.ylabel('Per Capita Income (US$)')

plt.title('Year vs Per Capita Income')

plt.show()

X = df['year'].values.reshape(-1, 1)

y = df['per capita income (US$)'].values

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

model = LinearRegression()

model.fit(X_train, y_train)

predicted_income_2020 = model.predict([[2020]])

print(f'Predicted per capita income for 2020: ${predicted_income_2020[0]:.2f}')

plt.scatter(df['year'], df['per capita income (US$)'], color='blue')

plt.plot(df['year'], model.predict(X), color='red')

plt.xlabel('Year')

plt.ylabel('Per Capita Income (US$)')

plt.title('Year vs Per Capita Income with Regression Line')

plt.show()

```


Multilinear Regression:

HIRING:

```
import pandas as pd

import numpy as np

from sklearn import linear_model

import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from google.colab import files

uploaded = files.upload()

df_hiring = pd.read_csv('hiring.csv')

print("Missing values in the dataset:")

print(df_hiring.isnull().sum())

df_hiring['experience'].fillna(df_hiring['experience'].mode()[0], inplace=True)

df_hiring['test_score(out of 10)'].fillna(df_hiring['test_score(out of 10)'].mean(), inplace=True)

print("Missing values in the dataset:")

print(df_hiring.isnull().sum())

experience_mapping = {

    'two': 2,

    'three': 3,

    'five': 5,

    'seven': 7,

    'eight': 8,

    'ten': 10,
```



```

'eleven': 11
}

df_hiring['experience'] = df_hiring['experience'].replace(experience_mapping)

X_hiring = df_hiring[['experience', 'test_score(out of 10)', 'interview_score(out of 10)']]

y_hiring = df_hiring['salary($)']

X_train_hiring, X_test_hiring, y_train_hiring, y_test_hiring = train_test_split(
    X_hiring, y_hiring, test_size=0.2, random_state=42)

hiring_model = LinearRegression()

hiring_model.fit(X_train_hiring, y_train_hiring)

predicted_salary_12_10_10 = hiring_model.predict([[12, 10, 10]])

print(f"\nPredicted salary for a candidate with 12 years of experience, 10 test score, and 10 interview
score: ${predicted_salary_12_10_10[0]:.2f}")

predicted_salary_2_9_6 = hiring_model.predict([[2, 9, 6]])

print(f"Predicted salary for a candidate with 2 years of experience, 9 test score, and 6 interview
score: ${predicted_salary_2_9_6[0]:.2f}")

```

1000_COMPANIES:

```

import pandas as pd

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

from google.colab import files

uploaded = files.upload()

```

```

df = pd.read_csv('1000_Companies.csv')

missing_values = df.isnull().sum()

print(f'Missing values in each column:\n{missing_values}')

X = df[['R&D Spend', 'Administration', 'Marketing Spend', 'State']]

y = df['Profit']

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

preprocessor = ColumnTransformer(

    transformers=[

        ('state', OneHotEncoder(), ['State']),

        ('num', 'passthrough', ['R&D Spend', 'Administration', 'Marketing Spend'])

    ])

pipeline = Pipeline(steps=[

    ('preprocessor', preprocessor),

    ('scaler', StandardScaler()),

    ('regressor', LinearRegression())

])

pipeline.fit(X_train, y_train)

new_data = pd.DataFrame({

    'R&D Spend': [91694.48],

    'Administration': [515841.3],

    'Marketing Spend': [11931.24],

    'State': ['Florida']

})

predicted_profit = pipeline.predict(new_data)

```

```
print(f'Predicted Profit: {predicted_profit[0]}")
```

HOME_PRICES_MULTIPLE_LR:

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn import linear_model
```

```
import matplotlib.pyplot as plt
```

```
from google.colab import files
```

```
uploaded = files.upload()
```

```
import pandas as pd
```

```
df = pd.read_csv('homeprices_Multiple_LR.csv')
```

```
df.head()
```

```
df.bedrooms.median()
```

```
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
```

```
dfreg = linear_model.LinearRegression()
```

```
reg.fit(df.drop('price', axis='columns'), df.price)
```

```
reg.coef_
```

```
reg.intercept_
```

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

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

Program 4

Build Logistic Regression Model for a given dataset

Screenshot

24/5

papergrid
Date: / /

Lab - 03.1

Train logistic Regression Problem

1) $a_0 = -5$
 $a_1 = 0.8$

$$P(\text{pass}) = \frac{1}{1 + e^{-(a_0 + a_1 x)}}$$
$$P(\text{pass}) = \frac{1}{1 + e^{-(-5 + 0.8x)}}$$

2) $x = 7$

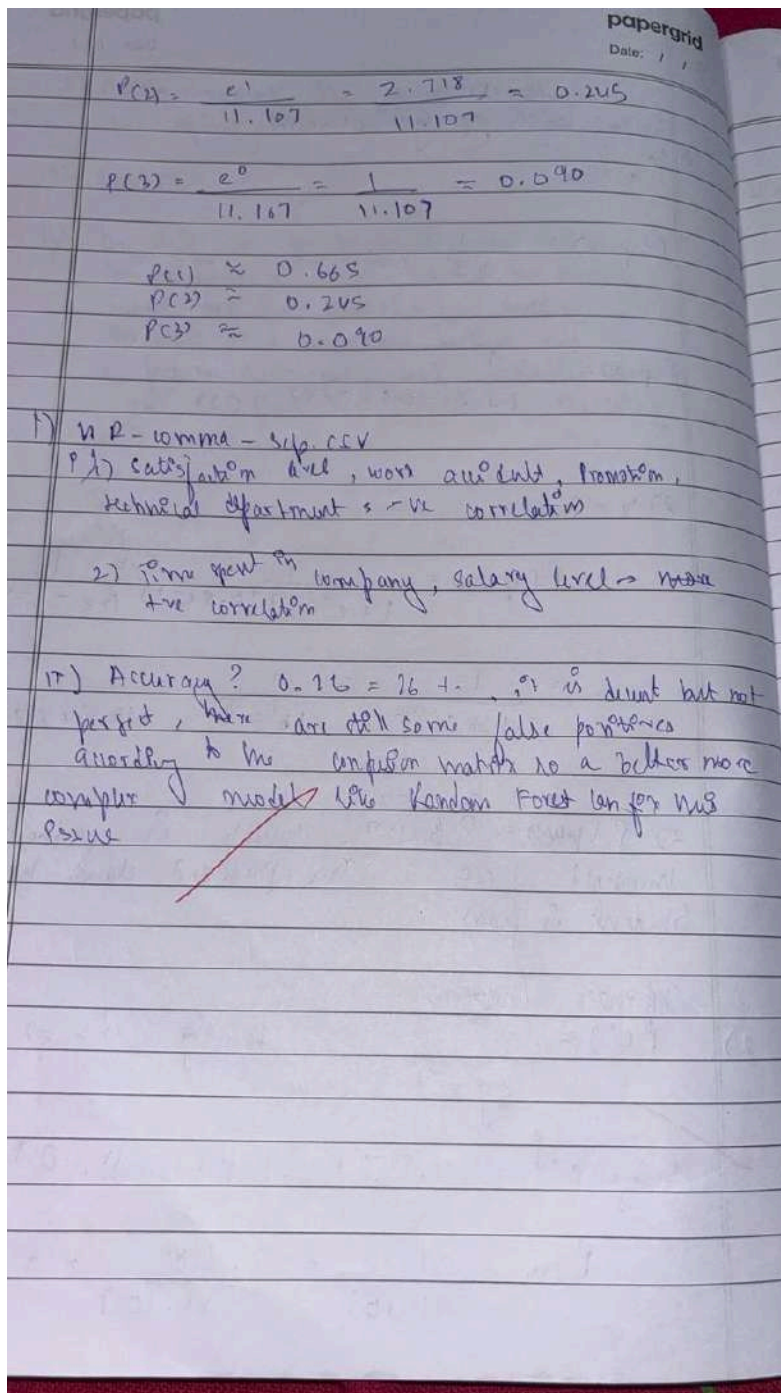
$$P(\text{pass}) = \frac{1}{1 + e^{-(-5 + 0.8(7))}} = \frac{1}{1 + e^{-(-5 + 5.6)}}$$
$$= \frac{1}{1 + e^{-0.6}} = \frac{1}{1 + 0.5488}$$
$$= 0.6457$$

3) $P(\text{pass}) = 0.6457$, which is greater than the threshold of 0.5, the predicted class for this student is pass

Softmax Problem.

2) $P(i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$ @ $\sum_j e^{z_j} = e^2 + e^1 + e^0$

$$\sum_j e^{z_j} = 7.389 + 2.718 + 1 = 11.107$$
$$P(1) = \frac{e^2}{11.107} = \frac{7.389}{11.107} = 0.665$$



Code:

HR_COMMA_SEPARATED.CSV:

```
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 LabelEncoder

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report

from google.colab import files

uploaded = files.upload()

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

df

df.info()

df.isnull().sum()

label_enc = LabelEncoder()

df["salary"] = label_enc.fit_transform(df["salary"])

plt.figure(figsize=(12, 6))

sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")

plt.title("Feature Correlation Heatmap")

plt.show()

correlation = df.corr()["left"].sort_values(ascending=False)

print("Correlation with Employee Retention:\n", correlation)

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

sns.countplot(x="salary", hue="left", data=df, palette="coolwarm")

plt.xlabel("Salary Level")

plt.ylabel("Number of Employees")

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

```

```

plt.xticks(ticks=[0, 1, 2], labels=["Low", "Medium", "High"])

plt.legend(["Stayed", "Left"])

plt.show()

plt.figure(figsize=(12, 6))

dept_retention = df.groupby("left")[df.columns[df.columns.str.startswith("Department_")]].sum().T

dept_retention.plot(kind="bar", figsize=(12, 6), colormap="coolwarm", edgecolor="black")

plt.xlabel("Department")

plt.ylabel("Number of Employees")

plt.title("Department-wise Employee Retention")

plt.xticks(rotation=45)

plt.legend(["Stayed", "Left"], title="Employee Status")

plt.show()

features = ["satisfaction_level", "last_evaluation", "number_project",
            "average_monthly_hours", "time_spend_company", "salary"]

features += [col for col in df.columns if "Department_" in col]

X = df[features]

y = df["left"]

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

model = LogisticRegression(max_iter=500)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

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

print("\nClassification Report:\n", classification_report(y_test, y_pred))

```

```

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt="d", cmap="coolwarm", xticklabels=["Stayed", "Left"],
yticklabels=["Stayed", "Left"])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

```

INSURANCE:

```

import pandas as pd

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score

from matplotlib import pyplot as plt

from google.colab import files

uploaded = files.upload()

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

df

```



```

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_

model.intercept_

import math

def sigmoid(x):

    return 1 / (1 + math.exp(-x))

def prediction_function(age):

    z = 0.127 * age - 4.973

```

```

    y = sigmoid(z)

    return y

age = 35

prediction_function(age)

MULTICLASS:

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

iris = pd.read_csv("/content/drive/MyDrive/Colab
Notebooks/ML-Course-6thSem-Feb-2025/Unit-2/iris.csv")

iris.head()

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

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

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

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: {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()
```

Program 5

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

Screenshot:

papergrid
Date: / /

LAB-5
DECISION TREE

find splitting node a_1 or a_2

Instance	a_1	a_2	Classification
1	hot	high	Yes
2	hot	normal	No
6	cool	high	No
7	hot	high	No
8	hot	normal	Yes

$H(S) = -p_1 \log_2 p_1 - p_2 \log_2 p_2$
 $= -\left(\frac{4}{5} \log_2 \frac{4}{5} + \frac{1}{5} \log_2 \frac{1}{5}\right)$
 $= -(0.8 \log_2 0.8 + 0.2 \log_2 0.2)$
 $= -(0.8(-0.3219) + 0.2(-2.3219))$
 $= -(1 - 0.2575 - 0.4644)$
 $= 0.721$

Entropy for attribute a_1

$H(\text{hot}) = -0.75 \log_2 0.75 - 0.25 \log_2 0.25 = 0.918$
 $H(\text{cool}) = -(1 \log_2 1) = 0$

Weighted entropy for a_1

$H(a_1) = \left(\frac{4}{5} \times 0.918\right) + (0)$

$$= 0.649$$

$$I_G(a_2) = H(S) - H(a_2) = 0.721 - 0.649 = 0.072$$

Entropy for a_2

$$H(\text{High}) = -\left(\frac{4}{5} \log_2 \frac{4}{5}\right) = 0$$

$$H(\text{Normal}) = -\left(\frac{1}{5} \log_2 \frac{1}{5}\right) = 0$$

$$\text{Weighted entropy for } a_2 = H(a_2) = \left(\frac{4}{5} \times 0\right) + \left(\frac{1}{5} \times 0\right) = 0$$

$$I_G(a_2) = H(S) - H(a_2) = 0.721 - 0 = 0.721$$

$$I_G(a_2) = 0.72 < I_G(a_1) = 0.721$$

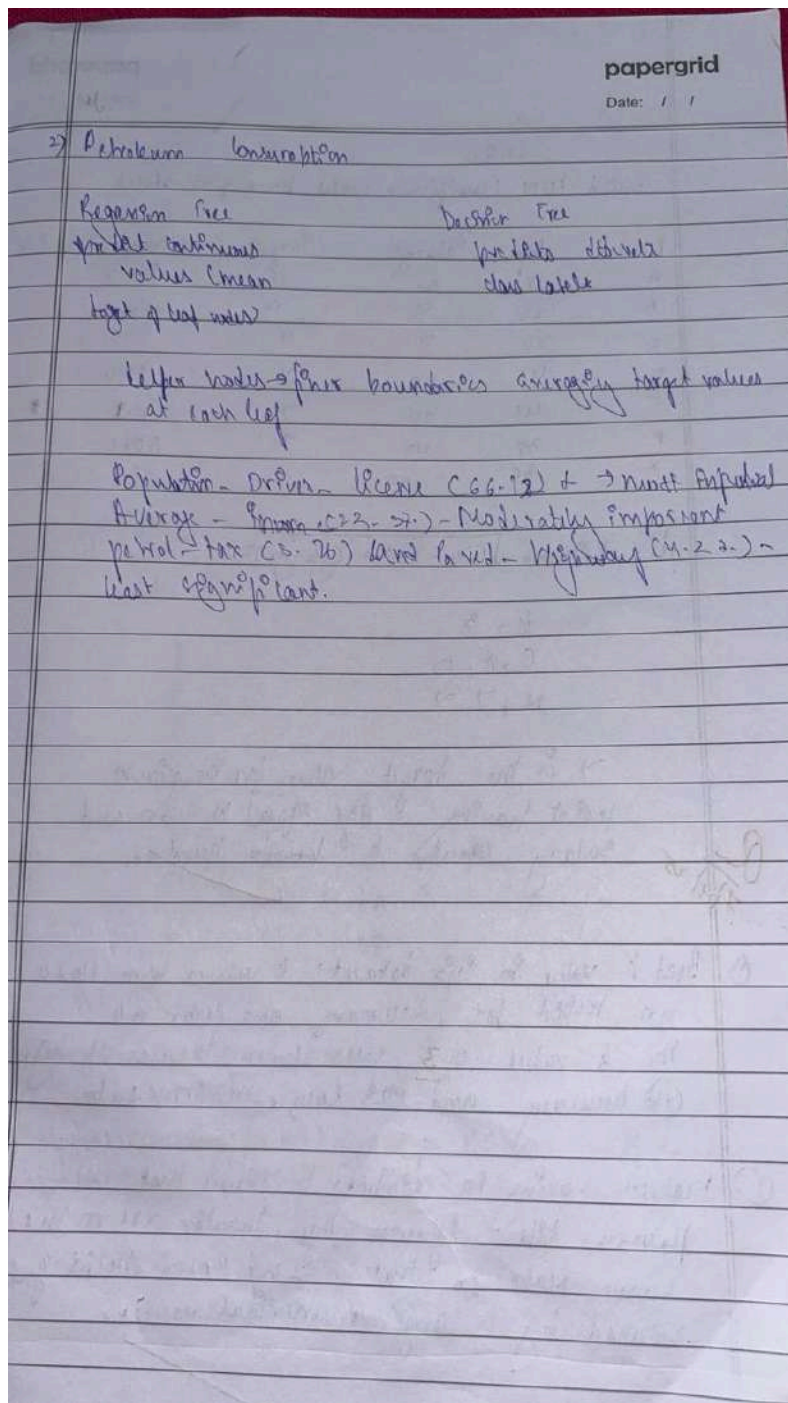
a_1 (High / Normal) should be the splitting Node

1) Accuracy RIS

$$1) \text{ Accuracy} = 1.00 = 100\%$$

Confusion matrix shows perfect classification with no misclassifications. All classes were correctly predicted, leading to 100% accuracy. No classes were confused.

2/13



Code:

PETROLEUM_CONSUMPTION:

```
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

from google.colab import files

import matplotlib.pyplot as plt

from sklearn.tree import plot_tree

uploaded = files.upload()

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

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

y = df['Petrol_Consumption']

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

model = DecisionTreeRegressor()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

print(f"Mean Absolute Error (MAE): {mae}")

print(f"Mean Squared Error (MSE): {mse}")

print(f"Root Mean Squared Error (RMSE): {rmse}")

feature_importance = model.feature_importances_

features = X.columns

print("\nFeature Importance:")

for feature, importance in zip(features, feature_importance):

```

```

    print(f'{feature}: {importance:.4f}')

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

plt.barh(features, feature_importance, color='skyblue')

plt.xlabel("Feature Importance Score")

plt.ylabel("Features")

plt.title("Feature Importance for Petrol Consumption Prediction")

plt.show()

plt.figure(figsize=(12, 6))

plot_tree(model, feature_names=features, filled=True, rounded=True)

plt.show()

```

DRUG_TEST:

```

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

from google.colab import files

uploaded = files.upload()

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

df['Sex'] = df['Sex'].map({'F': 0, 'M': 1})

df['BP'] = df['BP'].map({'LOW': 0, 'NORMAL': 1, 'HIGH': 2})

df['Cholesterol'] = df['Cholesterol'].map({'NORMAL': 0, 'HIGH': 1})

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

y = df['Drug']

```



```

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

model = DecisionTreeClassifier()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

print(f'Accuracy Score: {accuracy:.4f}')

print("Confusion Matrix:")

print(conf_matrix)

```

IRIS:

```

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

from google.colab import files

uploaded = files.upload()

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

X = iris.iloc[:, :-1]

y = iris.iloc[:, -1]

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

clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

```

```

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

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

print("Confusion Matrix:")

print(conf_matrix)

```

DECISION TREE CODE:

```

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

from sklearn.tree import plot_tree

import matplotlib.pyplot as plt

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']

}

df = pd.DataFrame(data)

label_encoders = {}

for column in df.columns:

    le = LabelEncoder()

    df[column] = le.fit_transform(df[column])

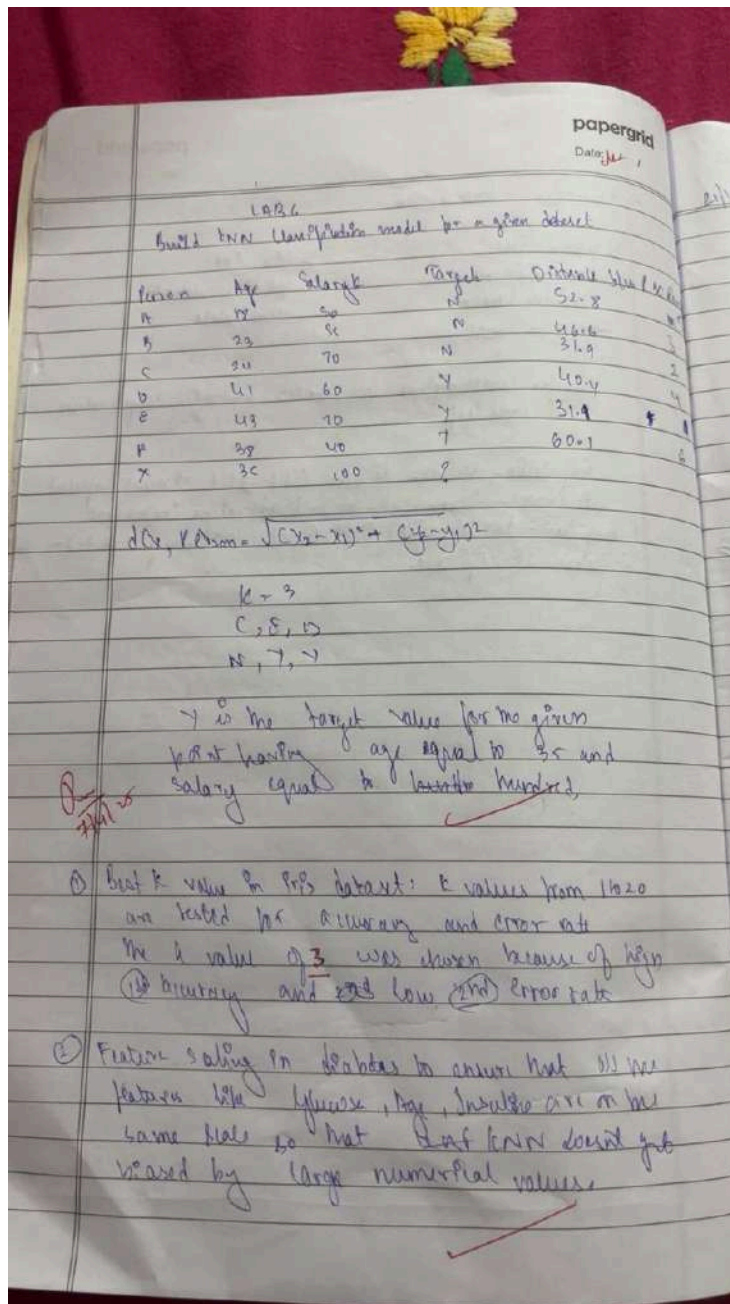
```

```
label_encoders[column] = le
X = df.drop('Classification', axis=1)
y = df['Classification']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred, target_names=['No', 'Yes']))
plt.figure(figsize=(12,8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=['No', 'Yes'])
plt.show()
```

Program 6

Build KNN Classification model for a given dataset.

Screenshot:



Code:

DIABETES:

```

import numpy as np

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 StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

from google.colab import files

uploaded = files.upload()

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

X = df.drop('Outcome', axis=1)

y = df['Outcome']

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

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

knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

print("Accuracy Score:", accuracy)

print("Confusion Matrix:\n", conf_matrix)

print("\nClassification Report:\n", classification_report(y_test, y_pred))

```

```

plt.figure(figsize=(6,4))

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No Diabetes', 'Diabetes'],
yticklabels=['No Diabetes', 'Diabetes'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

```

HEART.CSV:

```

import numpy as np

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 StandardScaler

from sklearn.neighbors import KNeighborsClassifier

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

from google.colab import files

uploaded = files.upload()

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

X = df.drop("target", axis=1)

y = df["target"]

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

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

```

```

X_test_scaled = scaler.transform(X_test)

scores = []

k_values = range(1, 21)

for k in k_values:

    knn = KNeighborsClassifier(n_neighbors=k)

    knn.fit(X_train_scaled, y_train)

    score = knn.score(X_test_scaled, y_test)

    scores.append(score)

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

plt.plot(k_values, scores, marker='o', linestyle='--')

plt.title('KNN Accuracy for different k values')

plt.xlabel('Number of Neighbors (k)')

plt.ylabel('Accuracy')

plt.grid(True)

plt.show()

best_k = k_values[np.argmax(scores)]

print(f'Best k: {best_k} with accuracy: {max(scores):.4f}')

best_knn = KNeighborsClassifier(n_neighbors=best_k)

best_knn.fit(X_train_scaled, y_train)

y_pred = best_knn.predict(X_test_scaled)

cm = confusion_matrix(y_test, y_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=best_knn.classes_)

disp.plot(cmap='Blues')

plt.title('Confusion Matrix')

```

```
plt.show()

report = classification_report(y_test, y_pred, output_dict=False)

print("Classification Report:")

print(report)
```

IRIS.CSV:

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

from google.colab import files

uploaded = files.upload()

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

df['species'] = df['species'].astype('category').cat.codes

X = df.drop('species', axis=1)

y = df['species']

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

param_grid = {'n_neighbors': list(range(1, 21))}

knn = KNeighborsClassifier()

grid = GridSearchCV(knn, param_grid, cv=5)

grid.fit(X_train, y_train)
```



```
best_k = grid.best_params_['n_neighbors']

print(f"Best k value: {best_k}")

knn = KNeighborsClassifier(n_neighbors=best_k)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

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

conf_matrix = confusion_matrix(y_test, y_pred)

print("Confusion Matrix:")

print(conf_matrix)

print("Classification Report:")

print(classification_report(y_test, y_pred))

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('Actual')

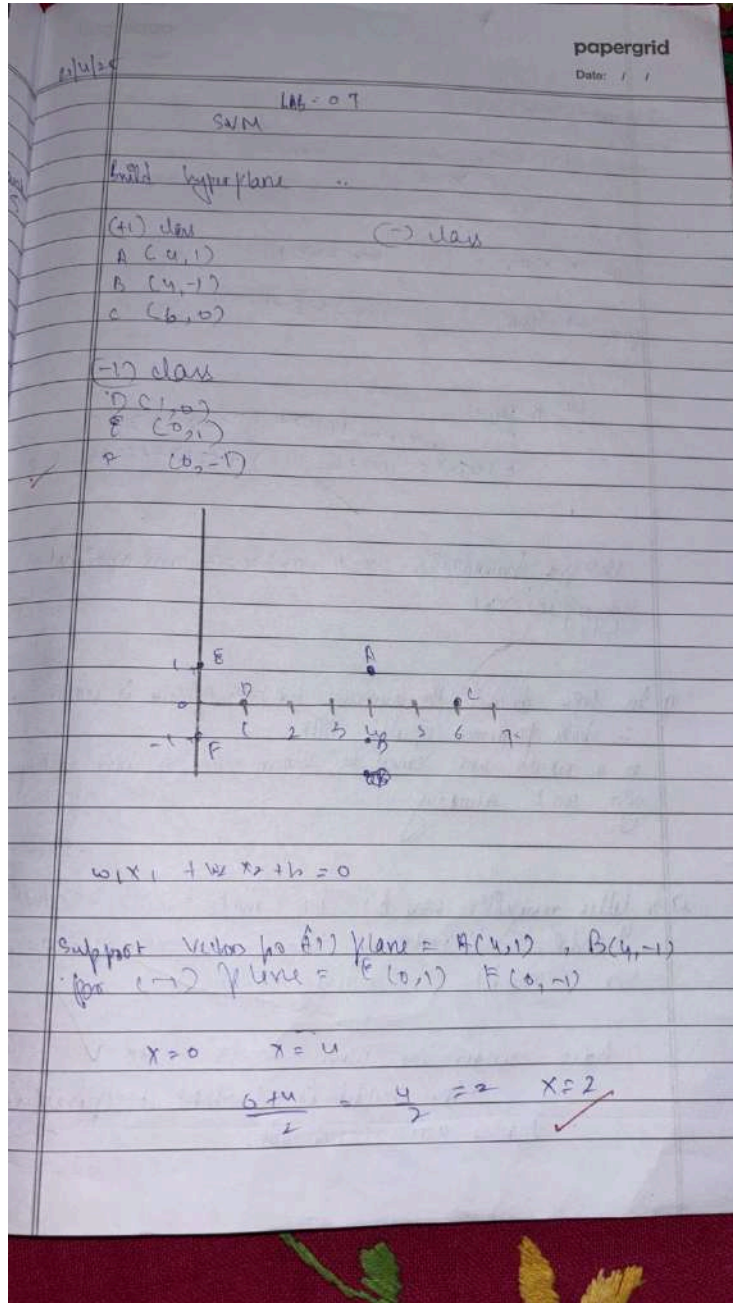
plt.title('Confusion Matrix')

plt.show()
```

Program 7

Build Support vector machine model for a given dataset

Screenshot



$$w = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$1x_1 + 0x_2 - 2 = 0 \quad x_1 = 2$$

for $x=1$ class

$$wx + b \geq 1$$

for $x=-1$ class

$$wx + b \leq -1$$

$$\text{Let } h(1,1)$$

$$= wx + b = 1(1) + 0(1) - 2 = 2 \geq 1$$

$$h(0,1) = wx + b = 1(0) + 0(1) - 2 = -2 \leq -1$$

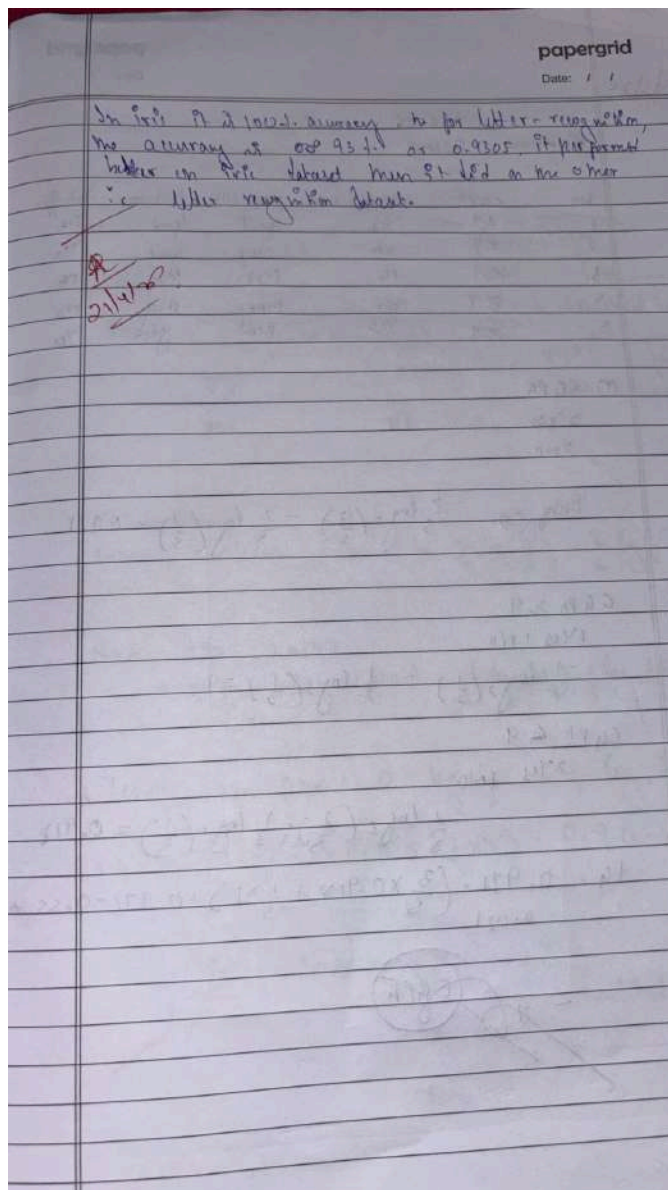
margin boundaries $x=1$ and $x=-3$ and optimal
hyperplane $x=2$

1) In 1st dataset: The accuracy for both of them is 100%.
∴ both proposed equations will
as a result one cannot be chosen over the other as they both
give 100% accuracy

2) In 2nd dataset: For F and P, S and Z, V and W,
M and R, K and R as some of the examples where the
model is getting confused

$$\text{Area under the curve} = 0.99 > 0.95$$

∴ the model is excellent at separating the
classes across thresholds.



Code:

IRIS.CSV:

```
import numpy as np
```

```
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.svm import SVC

from sklearn.metrics import accuracy_score, confusion_matrix

from google.colab import files

uploaded = files.upload()

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

le = LabelEncoder()

df['species'] = le.fit_transform(df['species'])

X = df.drop('species', axis=1)

y = df['species']

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

svm_linear = SVC(kernel='linear')

svm_linear.fit(X_train, y_train)

y_pred_linear = svm_linear.predict(X_test)

print("Linear Kernel SVM:")

print("Accuracy:", accuracy_score(y_test, y_pred_linear))

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_linear))

svm_rbf = SVC(kernel='rbf')

svm_rbf.fit(X_train, y_train)

y_pred_rbf = svm_rbf.predict(X_test)

print("\nRBF Kernel SVM:")

print("Accuracy:", accuracy_score(y_test, y_pred_rbf))

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rbf))

def plot_confusion_matrix(y_true, y_pred, kernel_name):

```

```

cm = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(5, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
             xticklabels=le.classes_, yticklabels=le.classes_)

plt.title(f'Confusion Matrix - {kernel_name} Kernel')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

plot_confusion_matrix(y_test, y_pred_linear, "Linear")

plot_confusion_matrix(y_test, y_pred_rbf, "RBF")

```

LETTER_RECOGNITION.CSV:

```

import numpy as np

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 LabelBinarizer

from sklearn.svm import SVC

from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, roc_curve

from google.colab import files

uploaded = files.upload()

df = pd.read_csv('letter-recognition.csv')

X = df.drop('letter', axis=1)

y = df['letter']

```

```

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

svm_model = SVC(kernel='rbf', probability=True)

svm_model.fit(X_train, y_train)

y_pred = svm_model.predict(X_test)

acc = accuracy_score(y_test, y_pred)

print("Accuracy:", acc)

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(14, 10))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',

            xticklabels=sorted(df['letter'].unique()),

            yticklabels=sorted(df['letter'].unique()))

plt.title("Confusion Matrix - SVM on Letter Recognition")

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.show()

lb = LabelBinarizer()

y_test_bin = lb.fit_transform(y_test)

y_score = svm_model.predict_proba(X_test)

auc = roc_auc_score(y_test_bin, y_score, average="macro", multi_class="ovr")

print("AUC Score (macro-averaged):", auc)

fpr = {}

tpr = {}

plt.figure(figsize=(10, 7))

for i, letter in enumerate(lb.classes_[:5]):

```

```

fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])

plt.plot(fpr[i], tpr[i], label=f'ROC curve for {letter}')

plt.plot([0, 1], [0, 1], 'k--')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curves (First 5 Letters)')

plt.legend()

plt.grid(True)

plt.show()

```

SVM,BASICS,CSV:

```

import pandas as pd

from sklearn.datasets import load_digits

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

digits = load_digits()

df = pd.DataFrame(digits.data, digits.target)

df['target'] = digits.target

X_train, X_test, y_train, y_test = train_test_split(df.drop('target', axis='columns'), df['target'],
test_size=0.3)

rbf_model = SVC(kernel='rbf')

rbf_model.fit(X_train, y_train)

print("RBF Kernel Accuracy:", rbf_model.score(X_test, y_test))

linear_model = SVC(kernel='linear')

linear_model.fit(X_train, y_train)

print("Linear Kernel Accuracy:", linear_model.score(X_test, y_test))

```


Program 8

Implement Random forest ensemble method on a given dataset

Screenshot

papergrid
Date: / /

9/5/20

LAB-08
RANDOM FOREST

No	CGPA	Internship	Internship	Internship	Job Offer
1	< 9	Yes	Good	Good	Yes
2	< 9	No	Med	Good	Yes
3	< 9	No	Med	Med	No
4	> 9	No	Med	Med	No
5	> 9	Yes	Med	Good	Yes

(1) CGPA

3 Yes
2 No

$$Entropy = -\frac{3}{5} \log_2 \left(\frac{3}{5} \right) - \frac{2}{5} \log_2 \left(\frac{2}{5} \right) = 0.971$$

CGPA > 9

1 Yes 1 No

$$-\frac{1}{2} \log_2 \left(\frac{1}{2} \right) - \frac{1}{2} \log_2 \left(\frac{1}{2} \right) = 1$$

CGPA < 9

2 Yes 1 No

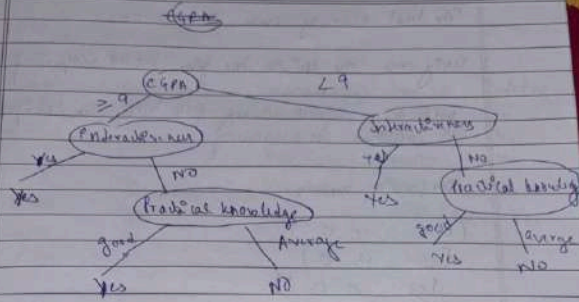
$$-\frac{2}{3} \log_2 \left(\frac{2}{3} \right) - \frac{1}{3} \log_2 \left(\frac{1}{3} \right) = 0.918$$

$$I_G = 0.971 - \left(\frac{3}{5} \times 0.918 + \frac{2}{5} \times 1 \right) = 0.971 - 0.855$$

$$= 0.116$$

< 9

(CGPA)



2) Interactiveness

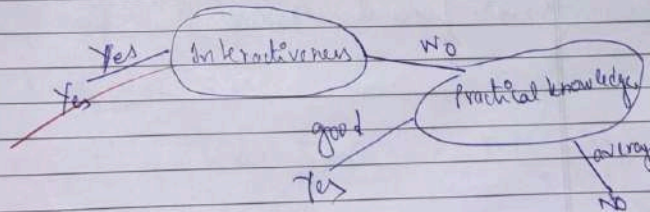
Entropy (Yes, No) = $-\frac{3}{5} \log_2 \left(\frac{3}{5} \right) - \frac{2}{5} \log_2 \left(\frac{2}{5} \right) = 0.971$

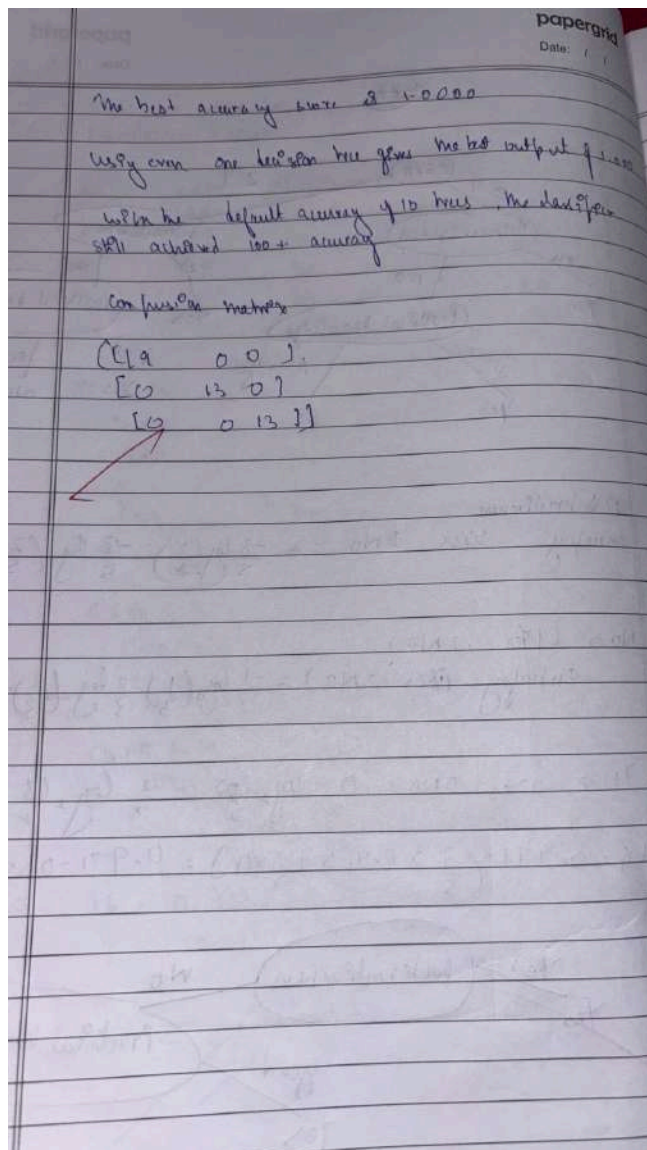
No (1 Yes, 1 No)

Entropy (Yes, No) = $-\frac{1}{3} \log_2 \left(\frac{1}{3} \right) - \frac{2}{3} \log_2 \left(\frac{2}{3} \right) = 0.918$

Yes (2 Yes, 0 No) = $0 - \log_2 (0) - \frac{2}{2} \log_2 \left(\frac{2}{2} \right) = 0$

$IG = 0.971 - \left(\frac{3}{5} \times 0.918 + \frac{2}{5} \times 0 \right) = 0.971 - 0.5508 = 0.4202$





Code:

IRIS.CSV:

```
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.ensemble import RandomForestClassifier
```

```

from sklearn.metrics import accuracy_score, confusion_matrix

from google.colab import files

uploaded = files.upload()

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

X = df.drop('species', axis=1)

y = df['species']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

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'Default RF Accuracy (n_estimators=10): {default_score:.4f}')

scores = []

tree_range = range(1, 101)

for n in tree_range:

    rf = RandomForestClassifier(n_estimators=n, random_state=42)

    rf.fit(X_train, y_train)

    y_pred = rf.predict(X_test)

    acc = accuracy_score(y_test, y_pred)

    scores.append(acc)

best_score = max(scores)

best_n = tree_range[scores.index(best_score)]

print(f'Best Accuracy: {best_score:.4f} with {best_n} trees')

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

```

```

plt.plot(tree_range, scores, marker='o')

plt.title('Random Forest Accuracy vs Number of Trees')

plt.xlabel('Number of Trees')

plt.ylabel('Accuracy')

plt.grid(True)

plt.show()

rf_best = RandomForestClassifier(n_estimators=1, random_state=42)

rf_best.fit(X_train, y_train)

y_pred_best = rf_best.predict(X_test)

cm = confusion_matrix(y_test, y_pred_best)

print("Confusion Matrix:\n", cm)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=rf_best.classes_,
            yticklabels=rf_best.classes_)

plt.title("Confusion Matrix")

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.show()

```

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot

2/5/25

papergrid
Date: / /

LAB-08
AdaBoost

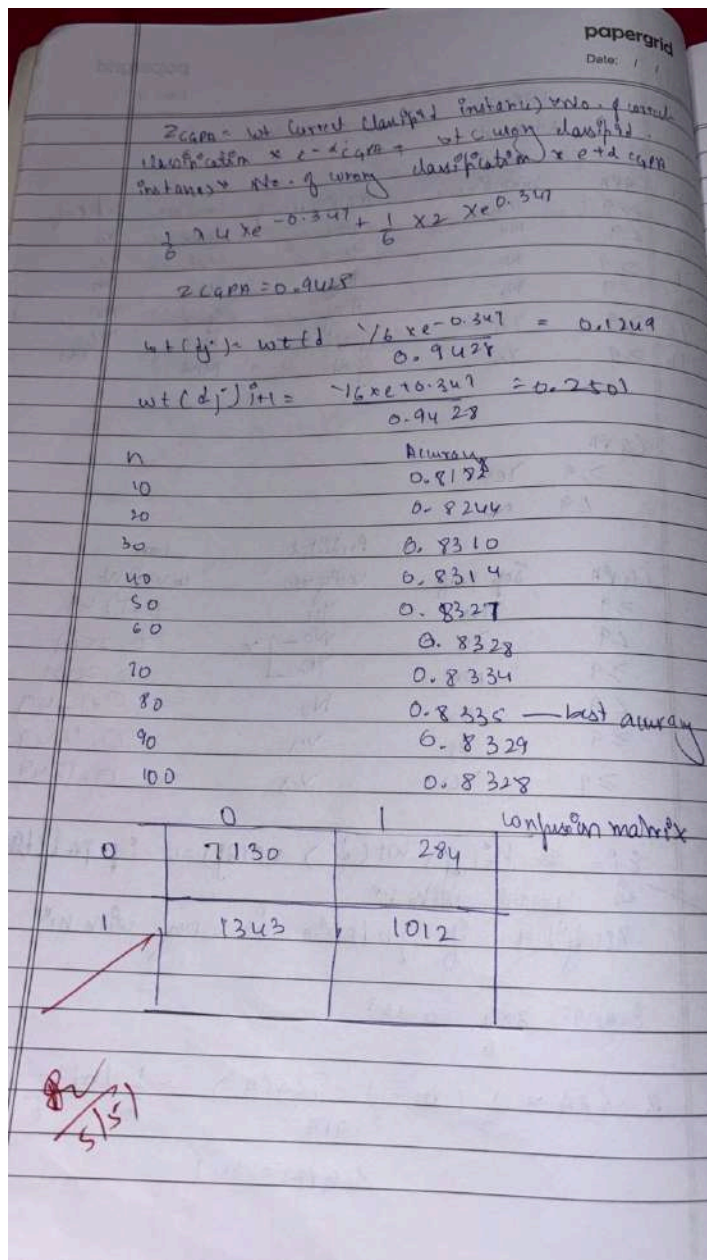
Weighted	CGPA	Intermediate	Practical knowledge	Language	Job Ref.
1/6	> 9	Yes	Good	Good	Yes
1/6	< 9	No	Good	Med	Yes
1/6	> 9	No	Avg	Med	No
1/6	< 9	No	Avg	Good	No
1/6	> 9	Yes	Good	Med	Yes
1/6	> 9	Yes	Good	Med	Yes

CGPA	Job Ref.	Weighted	Weight
> 9	Yes	Yes	0.1249
< 9	Yes	No	0.2501
> 9	No	Yes	0.2501
< 9	No	No	0.1249
> 9	Yes	Yes	0.1249
> 9	Yes	Yes	0.1249

$\epsilon_i = \sum_{j=1}^n w_j (y_j - h_j(x_j))$ if prediction is wrong with h_j
 $w_j(x_j) = 1$ if prediction is wrong with h_j

$\epsilon_{CGPA} = \frac{2 \times 1}{6} = 0.333$

$\alpha_{CGPA} = \frac{1}{2} \left(\ln \frac{1 - \epsilon_{CGPA}}{\epsilon_{CGPA}} \right) = \frac{1}{2} \ln \left(\frac{1 - 0.333}{0.333} \right)$
 $\alpha_{CGPA} = 0.347$



Code:

```

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.ensemble import AdaBoostClassifier

```

```

from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay

from google.colab import files

uploaded = files.upload()

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

X = df.drop("income_level", axis=1)

y = df["income_level"]

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

model_10 = AdaBoostClassifier(n_estimators=10, random_state=42)

model_10.fit(X_train, y_train)

y_pred_10 = model_10.predict(X_test)

score_10 = accuracy_score(y_test, y_pred_10)

print(f'Accuracy with 10 estimators: {score_10:.4f}')

estimator_range = range(10, 101, 10)

scores = []

for n in estimator_range:

    model = AdaBoostClassifier(n_estimators=n, random_state=42)

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)

    scores.append(acc)

    print(f'n_estimators={n}, Accuracy={acc:.4f}')

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

plt.plot(estimator_range, scores, marker='o')

plt.title("AdaBoost Accuracy vs Number of Estimators")

```



```

plt.xlabel("Number of Estimators")

plt.ylabel("Accuracy")

plt.grid(True)

plt.show()

best_n = estimator_range[scores.index(max(scores))]

best_score = max(scores)

print(f"\nBest Accuracy: {best_score:.4f} with n_estimators={best_n}")

best_model = AdaBoostClassifier(n_estimators=best_n, random_state=42)

best_model.fit(X_train, y_train)

y_best_pred = best_model.predict(X_test)

cm = confusion_matrix(y_test, y_best_pred)

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

disp.plot()

plt.title(f"Confusion Matrix (n_estimators = {best_n})")

plt.show()

```

Program 10

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

Screenshot

12/5/20

papergrid
Date: / /

LAL 10

k-means clustering

	$\text{dist}(x, k_1)$	$\text{dist}(x, k_2)$	cluster
(1, 1)	0	7.21	C ₁
(1.5, 2)	1.12	6.10	C ₁
(3, 4)	3.6	4.24	C ₁
(5, 7)	7.2	0	C ₂
(3.5, 5)	4.72	2.56	C ₂
(4.5, 5)	3.32	2.24	C ₂
(5.5, 4.5)	4.30	3.20	C ₂

$\text{dist}(x, y) = (x - x_1)^2 + (y - y_1)^2$

centroid:

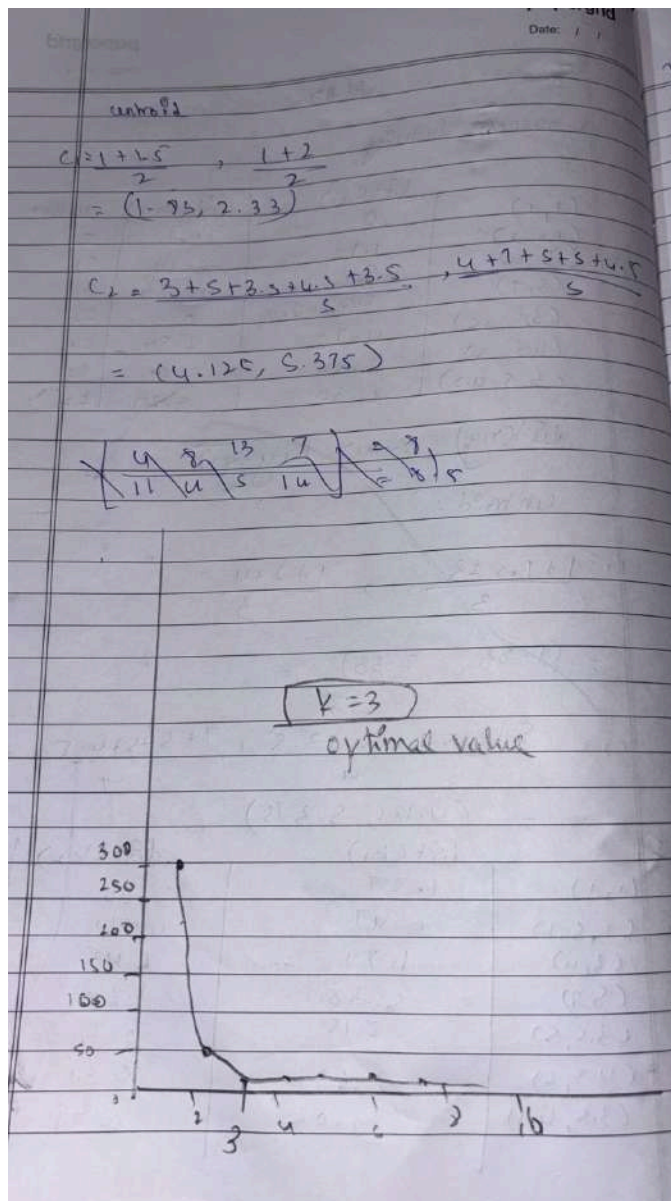
$C_1 = \left(\frac{1+1.5+3}{3}, \frac{1+2+4}{3} \right)$

$= (1.83, 2.33)$

$C_2 = \left(\frac{5+3.5+4.5+5.5}{4}, \frac{7+5+4+4.5}{4} \right)$

$= (4.125, 5.375)$

	$\text{dist}(x, k_1)$	$\text{dist}(x, k_2)$	
(1, 1)	1.57	5.64	C ₁
(1.5, 2)	0.47	4.72	C ₁
(3, 4)	1.99	1.48	C ₂
(5, 7)	5.38	1.68	C ₂
(3.5, 5)	2.15	0.72	C ₂
(4.5, 5)	2.69	0.51	C ₂
(5.5, 4.5)	2.03	0.99	C ₂



Code:

IRIS.:

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
from sklearn.cluster import KMeans
```

```
from sklearn.preprocessing import StandardScaler
```

```
from google.colab import files
```

```

uploaded = files.upload()

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

X = df[['petal_length', 'petal_width']]

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

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_)

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()

optimal_k = 3

kmeans = KMeans(n_clusters=optimal_k, random_state=42)

clusters = kmeans.fit_predict(X_scaled)

df['cluster'] = clusters

plt.scatter(kmeans.cluster_centers_[ :, 0], kmeans.cluster_centers_[ :, 1],

            s=200, c='black', marker='X', label='Centroids')

```

```
plt.title("K-Means Clusters on Petal Features")  
plt.xlabel("Petal Length (scaled)")  
plt.ylabel("Petal Width (scaled)")  
plt.legend()  
plt.grid(True)  
plt.show()
```

Program 11

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

Screenshot

12/5/25

papergrid
Date: / /

LAB-11 PREG
PCA

Feature	x_1	x_2	x_3	x_4
x_1	4	8	13	7
x_2	11	4	8	14

$$X = \begin{bmatrix} 4 & 8 & 13 & 7 \\ 11 & 4 & 8 & 14 \end{bmatrix}$$

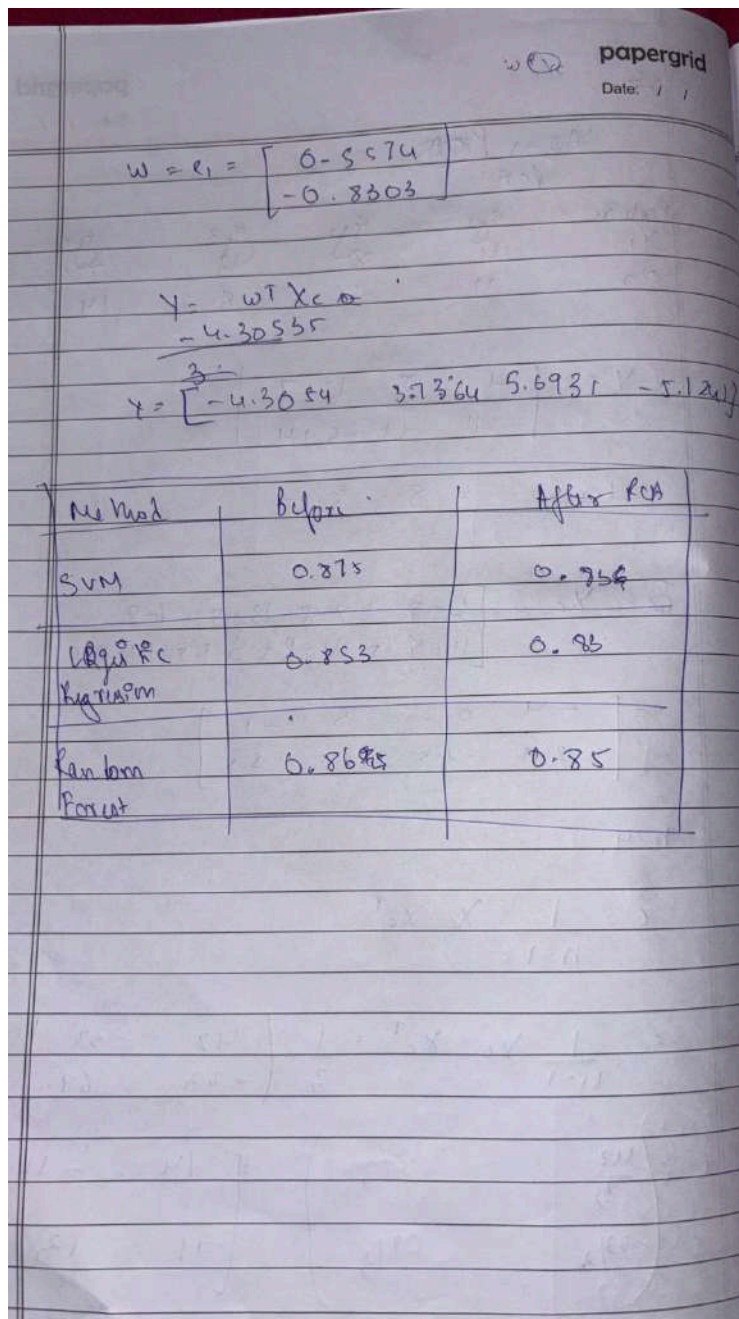
Means = 8, 8.5

$$X_c = \begin{bmatrix} 4-8 & 8-8.5 & 13-8 & 7-8 \\ 11-8 & 4-8.5 & 8-8.5 & 14-8.5 \end{bmatrix}$$
$$= \begin{bmatrix} -4 & -0.5 & 5 & -1 \\ 3 & -4.5 & -0.5 & 5.5 \end{bmatrix}$$

$n=4$

$$C = \frac{1}{n-1} X_c X_c^T$$
$$= \frac{1}{4-1} X_c X_c^T = \frac{1}{3} \begin{bmatrix} 42 & -33 \\ -33 & 69 \end{bmatrix}$$
$$= \begin{pmatrix} \frac{42}{3} & -\frac{33}{3} \\ -\frac{33}{3} & \frac{69}{3} \end{pmatrix} = \begin{bmatrix} 14 & -11 \\ -11 & 23 \end{bmatrix}$$

$\lambda_1 = 30.3$ $\lambda_2 = 6.6$



Code:

HEART.

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
```

```
from sklearn.model_selection import train_test_split
```

```

from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt

import seaborn as sns

from google.colab import files

uploaded = files.upload()

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

label_encoder = LabelEncoder()

df['Sex'] = label_encoder.fit_transform(df['Sex'])

df['FastingBS'] = label_encoder.fit_transform(df['FastingBS'])

df['ExerciseAngina'] = label_encoder.fit_transform(df['ExerciseAngina'])

df['HeartDisease'] = label_encoder.fit_transform(df['HeartDisease'])

print(df.head())

df = pd.get_dummies(df, columns=['ChestPainType', 'RestingECG', 'ST_Slope'], drop_first=True)

print(df.head())

X = df.drop('HeartDisease', axis=1)

y = df['HeartDisease']

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

print(X_scaled[:5])

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

```



```
svm_model = SVC()

svm_model.fit(X_train, y_train)

y_pred_svm = svm_model.predict(X_test)

accuracy_svm = accuracy_score(y_test, y_pred_svm)

print(f'SVM Accuracy: {accuracy_svm}')

log_reg_model = LogisticRegression()

log_reg_model.fit(X_train, y_train)

y_pred_log_reg = log_reg_model.predict(X_test)

accuracy_log_reg = accuracy_score(y_test, y_pred_log_reg)

print(f'Logistic Regression Accuracy: {accuracy_log_reg}')

rf_model = RandomForestClassifier()

rf_model.fit(X_train, y_train)

y_pred_rf = rf_model.predict(X_test)

accuracy_rf = accuracy_score(y_test, y_pred_rf)

print(f'Random Forest Accuracy: {accuracy_rf}')

pca = PCA(n_components=5)

X_train_pca = pca.fit_transform(X_train)

X_test_pca = pca.transform(X_test)

print(f'Explained variance ratio by PCA: {pca.explained_variance_ratio_}')

svm_model_pca = SVC()

svm_model_pca.fit(X_train_pca, y_train)

y_pred_svm_pca = svm_model_pca.predict(X_test_pca)

accuracy_svm_pca = accuracy_score(y_test, y_pred_svm_pca)

print(f'SVM Accuracy with PCA: {accuracy_svm_pca}')
```

```
log_reg_model_pca = LogisticRegression()

log_reg_model_pca.fit(X_train_pca, y_train)

y_pred_log_reg_pca = log_reg_model_pca.predict(X_test_pca)

accuracy_log_reg_pca = accuracy_score(y_test, y_pred_log_reg_pca)

print(f"Logistic Regression Accuracy with PCA: {accuracy_log_reg_pca}")

rf_model_pca = RandomForestClassifier()

rf_model_pca.fit(X_train_pca, y_train)

y_pred_rf_pca = rf_model_pca.predict(X_test_pca)

accuracy_rf_pca = accuracy_score(y_test, y_pred_rf_pca)

print(f"Random Forest Accuracy with PCA: {accuracy_rf_pca}")

print("\nAccuracy Comparison:")

print(f"SVM Accuracy: {accuracy_svm}")

print(f"Logistic Regression Accuracy: {accuracy_log_reg}")

print(f"Random Forest Accuracy: {accuracy_rf}")

print(f"SVM Accuracy with PCA: {accuracy_svm_pca}")

print(f"Logistic Regression Accuracy with PCA: {accuracy_log_reg_pca}")

print(f"Random Forest Accuracy with PCA: {accuracy_rf_pca}")
```