

Github Link:

https://github.com/SP212004/Machine_Learning-Lab

Program 1

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

Screenshot

Machine Learning
Lab - 0
05/03/25
Execute In To do.
Method - 1: Initializing values directly into Dataframe

```
import pandas as pd
data = {
    'VSN': [225, 230, 135, 140, 160],
    'Name': ['Shanthi', 'Deva', 'RishiKoh',
              'Varadha', 'CK Reddy'],
    'Marks': [97, 85, 76, 84, 92]
}
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())
```

Method - 2: Importing datasets from sklearn.datasets

```
from sklearn.datasets import load_diabetes
diabetet = load_diabetes()
df = pd.DataFrame(diabetet.data, columns=diabetet.feature_names)
df['target'] = diabetet.target
print("Sample data:")
print(df.head())
```

Method - 3: Importing datasets from a specific CSV file.

```
filepath = 'Sample-Sales-data.csv'  
df = pd.read_csv(filepath)  
print("sample data:")  
print(df.head())  
print("\n")
```

Method - 4: Downloading datasets from existing data repositories like Kaggle, UCI, Mendeley, KEEL, etc.

```
df = pd.read_csv('Dataset of Diabetes.csv',  
encoding = 'ISO-8859-1')  
print("sample data:")  
print(df.head())
```

Exercise - 2: To do.

Step 1: import required libraries.

import Y frame as Yf

import Pandas as pd

import matplotlib.pyplot as plt

Stock market Data Analysis, covering the
following:

1. HDFC Bank Ltd, ICICI Bank Ltd, Kotak
Mahindra Bank Ltd. tickers = ["HDFCBANK,
NS", "ICICIBANK,NS", "KOTAKBANK,NS"]

2. Start date : 2024-01-01, End date : 2024-12-31

3. plot the closing price and daily return for
all the three banks mentioned.

Step2: Downloading Stock Market Data

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

data = yf.download(tickers, start="2024-01-01",
end="2024-12-31",

group_by='ticker')

print("First 5 rows of the dataset")

print(data.head())

Step3: Basic Data Exploration.

print("n Shape of the dataset:")

print(data.shape)

print("n Column names:")

print(data.columns)

relance_data = data[["HDFCBANK,NS"]]

print("n Summary Statistics for HDFC Bank:")

print(relance_data.describe())

relance_data[["Daily Return"]] = relance_data[["Close"]].pct_change()

Step4: plot the closing price and daily return.

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

plt.subplot(2,1,1)

relance_data[["Close"]].plot(title="HDFC Bank Ltd.
(Closing Price)")

plt.subplot(2,1,2)

relance_data[["Daily Return"]].plot(title="HDFC Bank
Ltd. - Daily Return", color='orange')

plt.tight_layout()

plt.show()

drury 5/3/25

Code:

```
from sklearn.datasets import load_iris
import pandas as pd

iris = load_iris()
```

```

df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
print("sample data: ")
df.head()

# method 1
data = {
    'USN': ['A001', 'A002', 'A003', 'A004'],
    'Name': ['Amar', 'Akbar', 'Anthony', 'Venky'],
    'Marks': [34, 30, 31, 32]
}

df2 = pd.DataFrame(data)
df2

# method 2
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("sample data: ")
df.head()
# method 3

# Load data from a CSV file (replace 'data.csv' with your file path)
file_path = '/content/industry.csv' # Ensure the file exists in the same
directory
df2 = pd.read_csv(file_path)
print("Sample data:")
df2.head()

# method 4

file_path = '/content/Dataset of Diabetes .csv'
data3 = pd.read_csv(file_path)
df3 = pd.DataFrame(data3)

df3

```

```

#Using the code given in the above slides, do the exercise of the "Stock
Market Data Analysis", considering the following
# 1. HDFC Bank Ltd. , ICICI Bank Ltd , Kotak Mahindra Bank Ltd.
# tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
# 2. Start date: 2024-01-01, End date: 2024-12-30

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:")
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 Reliance Industries:")
print(hdfc_data.describe())
hdfc_data['Daily Return'] = hdfc_data['Close'].pct_change()

# icici bank
icici_data = data['ICICIBANK.NS']
print(hdfc_data.describe())
icici_data['Daily Return'] = icici_data['Close'].pct_change()

# Kotak bank
kotak_data = data['KOTAKBANK.NS']
print(hdfc_data.describe())
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()

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='orange')
plt.tight_layout()
plt.show()

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='orange')
plt.tight_layout()
plt.show()

# Save the Reliance data to a CSV file

hdfc_data.to_csv('hdfc_stock_data.csv')
icici_data.to_csv('icici_stock_data.csv')
kotak_data.to_csv('kotak_stock_data.csv')
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot

Lab - I Date : 05/03/25

Import pandas as pd.

- i) df = pd.read_csv("housing.csv")
- ii) df.info()
- iii) df.describe()
- iv) df[["Ocean Proximity"]].value_counts(), bins=7
- v) miss_val = df.isnull().sum()
val = miss_val[miss_val > 0]
print(val)

Diabetes:

Import pandas as pd.

Import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder

df = pd.read_csv("Dataset of Diabetes.csv")
print(df.head())

missing values
print(df.isnull().sum())

impute.
nc = df.select_dtypes(include=[float64, int64]).columns
imputer = SimpleImputer(strategy='mean')

$\text{df}[\text{enc}] = \text{imputer_fit_transform}(\text{df}[\text{enc}])$
 $\text{ad_c} = \text{df}[\text{c}]. \text{Select}[\text{type}[\text{include} = [\text{'object'}]]] \text{.columns}$
 $\text{imputer_cat} = \text{SimpleImputer}(\text{strategy} = \text{'most_frequent'})$
 $\text{df}[\text{Cat_c}] = \text{imputer_cat}. \text{fit_transform}(\text{df}[\text{Cat_c}])$

Handling categorical data.

$\text{label_encoder} = \text{LabelEncoder}()$
 $\text{df}[\text{'Gender'}] = \text{label_encoder}. \text{fit_transform}(\text{df}[\text{'Gender']})$
 $\text{df}[\text{'CLASS'}] = \text{label_encoder}. \text{fit_transform}(\text{df}[\text{'CLASS']})$

$Q_1 = \text{df}[\text{num_column}]. \text{quantile}(0.25)$
 $Q_3 = \text{df}[\text{num_column}]. \text{quantile}(0.75)$

$$IQR = Q_3 - Q_1$$

$\text{df_clean} = \text{df}[\text{enc}](\text{df}[\text{enc}] \leq (Q_1 - 1.5 * IQR))$
 $(\text{df}[\text{enc}] > (Q_3 + 1.5 * IQR)))$

Scaler_choice = 'minmax'

Scaler = MinMaxScaler

$\text{df_scaled} = \text{pd.DataFrame}(\text{Scaler}. \text{fit_transform}(\text{df_clean}[\text{enc}]), (\text{df_clean}[\text{enc}]))$

$\text{df_final} = \text{pd.concat}([\text{df_clean}[\text{Cat_c}], \text{df_scaled}], \text{axis}=1)$

- 1. No column.
- 2. Gender and CLASS

3. Min-Max-Scaling

If fixed range for feature, bounded range.

No outliers.

Standard Scaler,
* outliers exist
* Normal Distributed values

code:

```
# -*- coding: utf-8 -*-
"""LAB-1.ipynb
```

Automatically generated by Colab.

Original file is located at

<https://colab.research.google.com/drive/1LFIrPSjr6wkzvYXycyOlxEerHW0HtTTl2>

```
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

# Introduce some missing values for demonstration
df.loc[5, 'Age'] = np.nan
df.loc[10, 'Salary'] = np.nan
df.head(10)

# Basic information about the dataset
print(df.info())

# Summary statistics
print(df.describe())

#Code to Find Missing Values
# 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)

df_encoded. 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()

#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()

#Removing Outliers
# Z-score method
#Pros: Good for normally distributed data.
#Cons: Not suitable for non-normal data; may miss outliers in skewed
distributions.

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']) # Replace outliers with NaN
df_encoded_copy2.head()

#Removing Outliers
# Median replacement for outliers
#Pros: Keeps distribution shape intact, useful when capping isn't feasible.
#Cons: May distort data if outliers represent real phenomena.
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'])
df_encoded_copy3.head()

'''

At the start of the Lab, in the Observation book, Write python code for the following considering filename as "housing.csv"
i. To load .csv file into the data frame
ii. To display information of all columns
iii. To display statistical information of all numerical
iv. To display the count of unique labels for "Ocean Proximity" column
v. To display which attributes (columns) in a dataset have missing values count greater than zero
Step-2: Show the observation book to lab batch faculty incharge.
Step-3: Do the "To Do" tasks given in the PPT
Step-4: At the end of the lab,
i. Write the answers for questions given in the PPT and show it to lab batch faculty incharge
ii. Should upload the code in your respective GitHub account.
File name format:yourUSN_Lab-1-DataProcessing.ipynb
'''
```

```

filename = "/content/housing (1).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.")

data2 = pd.read_csv("/content/Dataset_with_Nulls.csv")
data2.head()

data2.info()

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,
StandardScaler, MinMaxScaler
from scipy import stats

# Load dataset
file_path = "/content/Dataset_with_Nulls.csv"

```

```

df = pd.read_csv(file_path)

### Step 1: Handling Missing Values ####
# Identify missing values
print("Missing values before handling:\n", df.isnull().sum())

# Handling missing numerical columns - Median for 'Age', Mean for other
numerical values
num_imputer_median = SimpleImputer(strategy="median")
num_imputer_mean = SimpleImputer(strategy="mean")

df["AGE"] = num_imputer_median.fit_transform(df[["AGE"]])
for col in ["Urea", "Cr", "HbA1c", "Chol", "TG", "HDL", "LDL", "VLDL",
"BMI"]:
    df[col] = num_imputer_mean.fit_transform(df[[col]])

# Convert categorical columns to string type
df["Gender"] = df["Gender"].astype(str)
df["CLASS"] = df["CLASS"].astype(str)

# Handling missing categorical columns - Fill with Mode
cat_imputer = SimpleImputer(strategy="most_frequent")

df["Gender"] = cat_imputer.fit_transform(df[["Gender"]]).ravel()
df["CLASS"] = cat_imputer.fit_transform(df[["CLASS"]]).ravel()

print("Missing values after handling:\n", df.isnull().sum())

### Step 2: Handling Categorical Attributes ####
# Encode 'Gender' using Ordinal Encoding (Male = 0, Female = 1)
ordinal_encoder = OrdinalEncoder(categories=[["Male", "Female"]])
df['Gender'] = df['Gender'].replace({'F': 'Female', 'f': 'Female', 'M':
'Male'}) # Handle variations
# Replace NaN with Mode (most frequent) for Gender
df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0]) # Fill na with
mode if any

# Now apply ordinal encoding
ordinal_encoder = OrdinalEncoder(categories=[["Male", "Female"]])
# One-Hot Encoding for 'CLASS'
onehot_encoder = OneHotEncoder(sparse_output=False)

```

```

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

# Convert to DataFrame
class_encoded_df = pd.DataFrame(class_encoded,
columns=onehot_encoder.get_feature_names_out(["CLASS"]))

# Merge One-Hot Encoded Data and drop original categorical columns
df = pd.concat([df, class_encoded_df], axis=1)
df.drop(["Gender", "CLASS"], axis=1, inplace=True)

#### Step 3: Data Transformation ####
# Min-Max Scaling for Salary
minmax_scaler = MinMaxScaler()
df[["Urea", "Cr", "HbA1c", "Chol", "TG", "HDL", "LDL", "VLDL", "BMI"]] =
minmax_scaler.fit_transform(
    df[["Urea", "Cr", "HbA1c", "Chol", "TG", "HDL", "LDL", "VLDL", "BMI"]]
)

# Standardization for Age
standard_scaler = StandardScaler()
df[["AGE"]] = standard_scaler.fit_transform(df[["AGE"]])

#### Step 4: Removing Outliers ####
# IQR Method for 'Salary' (Replacing Outliers with Boundaries)
for col in ["Urea", "Cr", "HbA1c", "Chol", "TG", "HDL", "LDL", "VLDL",
"BMI"]:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[col] = np.where(df[col] > upper_bound, upper_bound, np.where(df[col] <
lower_bound, lower_bound, df[col]))

# Z-score method for 'AGE' (Replacing Outliers with NaN)
df["AGE_zscore"] = stats.zscore(df["AGE"])
df["AGE"] = np.where(df["AGE_zscore"].abs() > 3, np.nan, df["AGE"])

# Median Replacement for Outliers in 'AGE'
median_age = df["AGE"].median()
df["AGE"] = np.where(df["AGE"].isnull(), median_age, df["AGE"])

```

```
# Drop auxiliary columns
df.drop(columns=["AGE_zscore"], inplace=True)

df

print("Preprocessing Complete. Cleaned dataset saved!")
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot

Lab - 3

1.9/03/2025

Linear regression &
Straight line method

① Using $y = mx + c$ or $y = a + bx$ formula to find slope & intercept and to find the 5th & 6th week Sales based on the below table, by linear regression.

(Week)	(Sales in thousands)
1	2
2	4
3	5
4	9

~~import pandas as pd~~

~~# Read the data from the CSV file to load into a DataFrame~~

~~data = pd.read_csv('content/data.csv')~~

~~x = data['X(Week)']~~

~~y = data['Y(Sales in thousands)']~~

~~# Calculate the mean of X & Y.~~

~~$x_mean = x_mean$~~

~~$y_mean = y_mean$~~

~~# Calculate the slope (b_1)~~

~~numerator = $((x - x_mean) * (y - y_mean))$. sum()~~

~~denominator = $((x - x_mean)^2)$. sum()~~

~~$b_1 = \frac{\text{numerator}}{\text{denominator}}$~~

~~# calculate the Intercept (b_0)~~

~~$b_0 = y_mean - (b_1 * x_mean)$~~

~~print("Intercept:", round(b_0, 2))~~

~~print("Slope:", round(b_1, 2))~~

predict sales for the 5th and 7th week.

$$\text{week_5_sales} = b_0 + (b_1 * 5)$$

$$\text{week_7_sales} = b_0 + (b_1 * 7)$$

Print("predicted sales for week 5:", round(week_5_sales, 2))
Print("predicted sales for week 7:", round(week_7_sales, 2))

Output

Intercept: -0.5

Slope: 2.2

Predicted sales for week 5: 10.5

Predicted sales for week 7: 14.9

② Matrix method
Linear regression of data of weeks and product sales
throughout the matrix approach for finding slope(b_1),
Intercept(b_0) and then predicting sales for 5th and 7th weeks.

X (Weeks)	Y (Sales in thousands)
1	2
2	4
3	5
4	9

import pandas as pd

import numpy as np

Importing

data = pd.read_csv('/content/data.csv')

prepare the data for linear regression

x = data['x(weeks)'].values

y = data['y (Sales in thousands)'].values

Create the design matrix.

$$X_{\text{matrix}} = np.vstack([np.ones(len(X)), X]).T$$

calculate the coefficient along the matrix method.

$$\text{coefficients} = np.linalg.inv(X_{\text{matrix}} @ X_{\text{matrix}}) @ y$$

Extract the intercept and slope

$$\text{intercept} = \text{coefficients}[0]$$

$$\text{slope} = \text{coefficients}[1]$$

print("Intercept:", round(intercept, 2))

print("Slope:", round(slope, 2))

predict sales for 5th and 7th week

$$\text{week_5_sales} = \text{intercept} + \text{slope} * 5$$

$$\text{week_7_sales} = \text{intercept} + \text{slope} * 7$$

print("predicted
sales for week 5:")

print("predicted
sales for week 7:")

Output

Intercept: -0.5

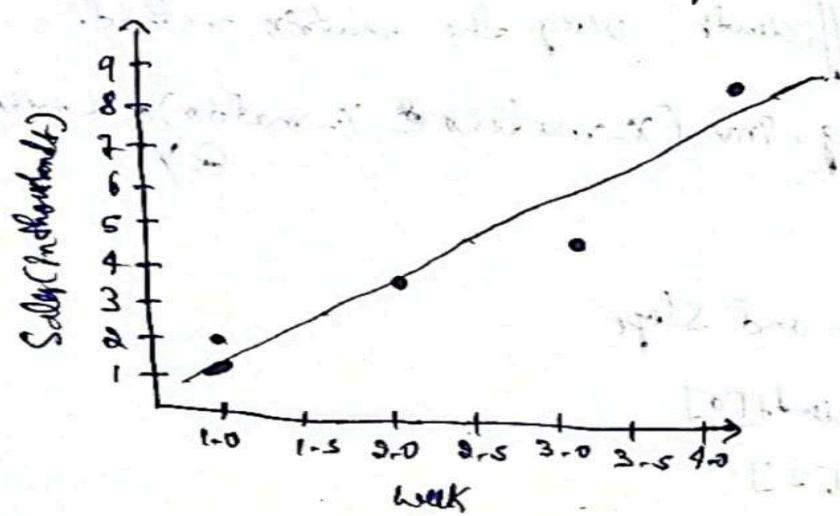
Slope: 2.2

Predicted sales for week 5: 10.5

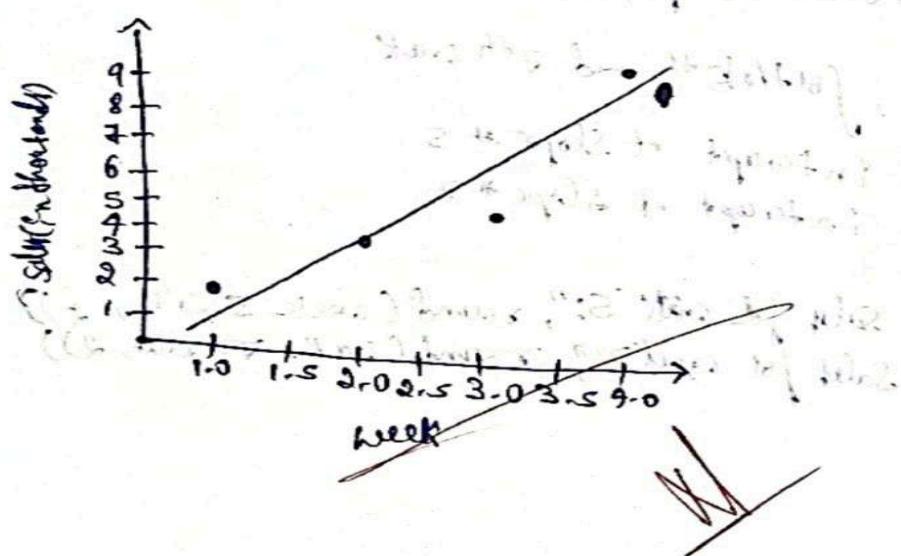
Predicted sales for week 7: 14.9

④ Straight line method graph

Salt vs week regression



⑤ Matrix method graph



```

code:
import numpy as np
import matplotlib.pyplot as plt

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x = np.mean(x)
    m_y = np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum((x - m_x) * (y - m_y))
    SS_xx = np.sum((x - m_x) ** 2)

    # calculating regression coefficients
    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):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color="m", marker="o", s=30)

    # predicted response vector
    y_pred = b[0] + b[1] * x

    # plotting the regression line
    plt.plot(x, y_pred, color="g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title("Linear Regression")
    plt.show()

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

```

```

# estimating coefficients
b = estimate_coef(x, y)
print(f"Estimated coefficients:\nb_0 = {b[0]} \nb_1 = {b[1]}")

# plot regression line
plot_regression_line(x, y, b)

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)

```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot

Lab-4
Logistic regression

To solve

- Consider a binary classification problem where we want to predict whether a student will pass or fail based on their study hours. The logistic regression model has been fitted, and the learned parameters are $a_0 = -5$ (intercept) and $a_1 = 0.8$ (coefficient for Study hours)
 - Write the logistic regression equation for this problem.
 - Calculate the probability that a student who studies for 7 hours will pass.
 - Determine the predicted class (pass or fail) for this student based on threshold of 0.5.
- Consider $x = [2, 1, 0]$ for three classes. Apply softmax function to find the probability vector of these classes.

Given logistic regression equation :-

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

where,

$a_0 = -5$ (Intercept)
 $a_1 = 0.8$ (coefficient for Study hours).
 x represent Study hours.

(a) logistic regression equation

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

(b) Probability of passing for 7 hours of study.
 we substitute, $x = 7$:

$$P(\text{pass}) = \frac{1}{1 + e^{(-5 + 0.8 \times 7)}}$$

$$P(\text{part}) = \frac{1}{1 + e^{-0.546}}$$

$$P(\text{part}) = \frac{1}{1 + e^{-0.6}}$$

$$P(\text{part}) = \frac{1}{1 + e^{-0.598}} = 0.645$$

Classification chart

- If $P(\text{part}) > 0.5$, the student is predicted to pass.
- If $P(\text{part}) < 0.5$, the student is predicted to fail.
- So based on this, $P(\text{part}) = 0.645$ approximately. Therefore, the student is pass.

Q.Sold

The softmax function for unit $i = n$:

$$P_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

$$\text{for } i = [2, 1, 0]$$

$$P_1 = \frac{e^{2.89}}{e^{2.89} + e^0} = \frac{e^{2.89}}{11.108} = 0.65$$

$$P_2 = \frac{e^{2.18}}{e^{2.89} + e^0} = \frac{e^{2.18}}{11.108} = 0.645$$

$$P_3 = \frac{e^0}{e^{2.89} + e^0} = \frac{1}{11.108} = 0.099$$

Rejected

After building the logistic regression model, write the answers for following questions in your observation book.

(i) Get dataset file "HR-comma-sp.csv".

(ii) Which variables did you identify as having direct and clear impact on employee retention? Why?

Sal (GT Salary)

• Employee with low salary had a higher chance of leaving compared to those with medium or higher salary.

• Its scatter plot showed a negative correlation between salary and attrition rate.

(iii) Department (e.g. sales, technical) had higher attrition.

(iv) Satisfaction level (e.g. low satisfaction, long hours, less promotion) increased attrition.

(v) What was the accuracy of your logistic regression model? Do you think it is a good accuracy? Why or why not?

• Accuracy: 78.13%

• Good enough if above 80%, it's good if below 70% improvements are needed.

• Factors like feature selection, using a different model (decision tree, random forest etc.) or handling imbalanced data could help enhance performance.

Q. for 200 dataset.

(i) Did you perform any data pre-processing step? If yes, what were they and why were they necessary? If no, why not? (removed irrelevant columns and applied one-hot encoding)

(ii) Were there any missing or inconsistent values in the dataset? If yes, how did you handle them?

No missing values found. They can be handled with filling with mean, median or mode methods.

(iii) What does the confusion matrix tell you about the performance of your model?

More correct predictions along the diagonal mean better model performance.

(iv) Which class types were most frequently misclassified? Why do you think that happened?

Similar features, imbalanced data, and model limitations in handling complex problems.

```

code:
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()
```

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

② Use an appropriate data set for building the decision tree (ID3) and apply that knowledge to classify a new sample.

Step 1: Import necessary libraries.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import metrics
import matplotlib.pyplot as plt
```

Step 2: Load the weather dataset from the csv file.

```
df = pd.read_csv("/content/fennet.csv")
```

Step 3: Display the first few rows of the dataset to understand its structure.

```
print("First 5 rows of the dataset:")
print(df.head())
```

Step 4: Pre processing

```
le_outlook = LabelEncoder()
le_temp = LabelEncoder()
le_humidity = LabelEncoder()
le_windy = LabelEncoder()
le_play = LabelEncoder()
```

```

# Encoding the relevant column (all lowercase for you)
df[["outlook"]]=lo_outlook, df["IsTeenager"] = df[["outlook"]]

df[["temp"]]=le_temp, df["transform"] = df[["temp"]]

df[["humidity"]]=le_humidity, df["transform"] = df[["humidity"]]

df[["windy"]]=le_windy, df["transform"] = df[["windy"]]

df[["play"]]=le_play, df["transform"] = df[["play"]]

## Step 5: Define features(X) and the target variable(y)
X=df[['outlook', 'temp', 'humidity', 'windy']]
y=df['play']

## Step 6: Split the dataset
x_train, x_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=42)

## Step 7: Create a decision tree classifier
clf = DecisionTreeClassifier(criterion='entropy', random_state=42)

## Step 8: Train the classifier
clf.fit(x_train, y_train)

## Step 9: Predict the labels
y_pred = clf.predict(x_test)

## Step 10: Evaluate the model's accuracy
accuracy = metrics.accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100 :.2f} %")

## Step 11: Classify a new weather sample.
new_sample=[lo_outlook.transform(['Sunny']), le_temp.transform(['cool']), le_humidity.transform(['high']), le_windy.transform(['fall'])]

```

predicted_clf = clf, predict(new_sample)
output the predicted class.
Print(f"Predicted class for the new sample: {y[0]} if
le-play, Inverse-transform ([predicted_class[0]])
[0] = yes, else 'no', y[0])"

Step 12: Visualize the decision tree.

```

plt.figure(figsize=(10,8))
plot_tree(clf, feature_names=['outlook', 'temp', 'humidity',
'windy'], class_names=le_play.classes_, filled=True, rounded=True)
plt.title("Decision tree for weather data")
plt.show()

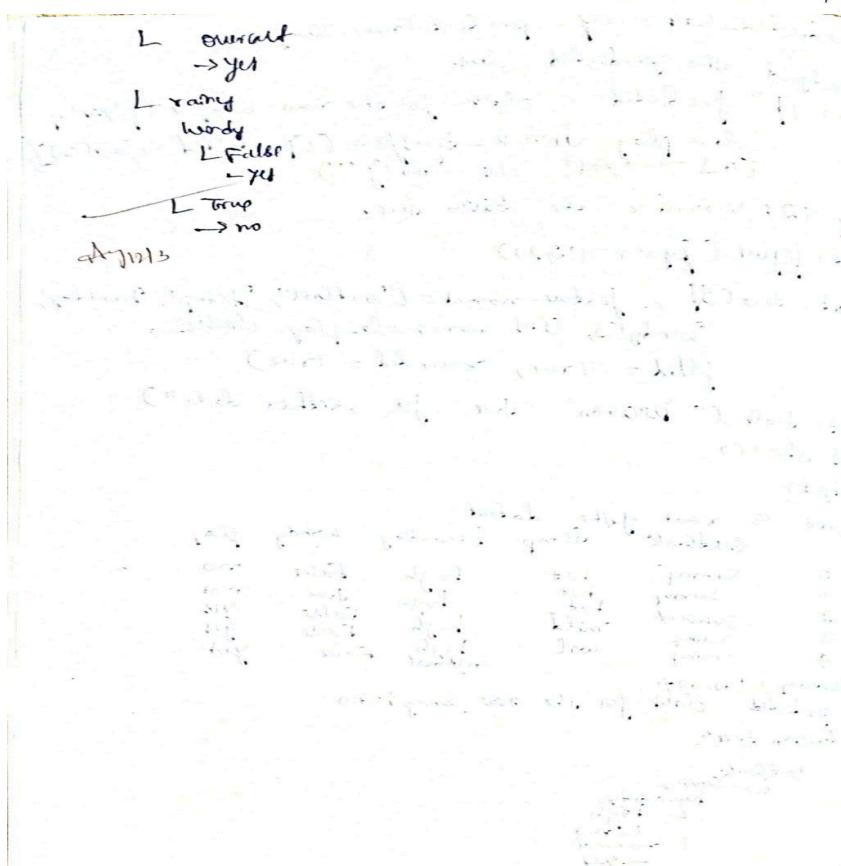
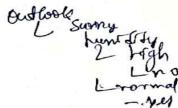
```

Output:

	outlook	temp	humidity	windy	play
0	Sunny	hot	high	false	no
1	Sunny	hot	high	true	no
2	overcast	hot	high	false	yes
3	rainy	mild	high	false	yes
4	rainy	cool	normal	false	yes

Accuracy: 100.00%
Predicted class for the new Sample: no

Decision tree:



```

code:
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import math
import copy

dataset = pd.read_csv('/content/Tennis.csv')
X = dataset.iloc[:, :].values
print(X)

dataset

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):
    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)

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()
```

Program 6

Build KNN Classification model for a given dataset

Screenshot

Lab-5
KNN Classification model

Step1 Consider the following dataset for $K = 3$ and test data ($35, 100$) at
 (Gender, Age, Salary) solve using Knn classifier model
 and predict the target.

Gender	Age	Salary	Target
A	18	50	N
B	23	55	N
C	24	70	N
D	41	60	Y
E	43	70	Y
F	38	40	Y
X	35	100	Y

Step2 Compute Euclidean distance

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

A $\sqrt{(35-18)^2 + (100-50)^2} = \sqrt{97} \approx 9.8$
 B $\sqrt{(35-23)^2 + (100-55)^2} = \sqrt{465} \approx 21.5$
 C $\sqrt{(35-24)^2 + (100-70)^2} = \sqrt{380} \approx 19.5$
 D $\sqrt{(35-41)^2 + (100-60)^2} = \sqrt{404} \approx 20.1$
 E $\sqrt{(35-43)^2 + (100-40)^2} = \sqrt{310} \approx 17.6$
 F $\sqrt{(35-38)^2 + (100-40)^2} = \sqrt{609} \approx 24.6$

Step3 Select $K=3$ Nearest Neighbour.
 Considering the distances, the 3 nearest neighbour are
 1. E $(31.05, Y)$
 2. C $(38.0, N)$
 3. D $(40.45, Y)$

Step 3: Majority voting

Among the 3 nearest neighbors

- 2 are $\gamma(E, D)$
- 1 is $N(C)$.

Since γ is the majority (2 out of 3), the predicted target for x is γ .

Final Predictor

for given $\mathcal{N}(S, 100)$. The predicted target is γ .

To answer

1. For grid search:

How to choose the K value? Demonstrate using accuracy rate and error rate.

2. For Principal Subsets

What is the purpose of feature scaling? How to perform it?

(1) to test different K values (e.g., 1 to 20).

• plot accuracy and error rate vs K .

• small $K \rightarrow$ overfitting, large $K \rightarrow$ underfitting.

• optimal K (5-10) balanced accuracy and error

(2) Feature Scaling

• Standardize features (contribution of all features).

• prevent large values from dominating distance-based models

• methods

• Min-Max Scaler: Scale to $[0, 1]$ (use for non-Gaussian data)

• Standardization: $x' = (x - \mu)/\sigma$, $\mu = \text{mean}$, $\sigma^2 = \text{variance}$

for stability data - scale Standardization is preferred.

code:

```
import math
import matplotlib.pyplot as plt

# Step 1: Distance calculation (Euclidean)
def distance(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

# Step 2: KNN Function
def knn(training_data, test_point, k):
    distances = []
    for point, label in training_data:
        d = distance(point, test_point)
        distances.append((d, label))
    distances.sort()
    k_nearest = distances[:k]
    labels = [label for _, label in k_nearest]
    prediction = max(set(labels), key=labels.count)
    return prediction

# Step 3: Visualization Function
def visualize_knn(data, test_point, predicted_label, new_point=None,
new_label=None):
    colors = {'A': 'blue', 'B': 'red'}
    markers = {'A': 'o', 'B': 's'}

    # Plot training data
    for point, label in data:
        plt.scatter(point[0], point[1], color=colors[label],
marker=markers[label], label=label if f"train_{label}" not in
plt.gca().get_legend_handles_labels()[1] else "")

    # Plot test point
    plt.scatter(test_point[0], test_point[1], color='green', marker='*',
s=200, label=f'Test → {predicted_label}')

    # Plot new point if provided
    if new_point is not None and new_label is not None:
        plt.scatter(new_point[0], new_point[1], color='orange', marker='x',
s=150, label=f'New → {new_label}')



```

```

plt.legend()
plt.grid(True)
plt.title("KNN Classification")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()

# Step 4: Run everything
if __name__ == "__main__":
    # Training data
    data = [
        ([1, 2], 'A'),
        ([2, 3], 'A'),
        ([3, 1], 'A'),
        ([6, 5], 'B'),
        ([7, 7], 'B'),
        ([8, 6], 'B')
    ]

    # Test point
    test = [2, 2]
    result = knn(data, test, k=3)
    print("Predicted class for test:", result)

    # New point to classify
    new_point = [7, 5]
    new_result = knn(data, new_point, k=3)
    print("Predicted class for new point:", new_result)

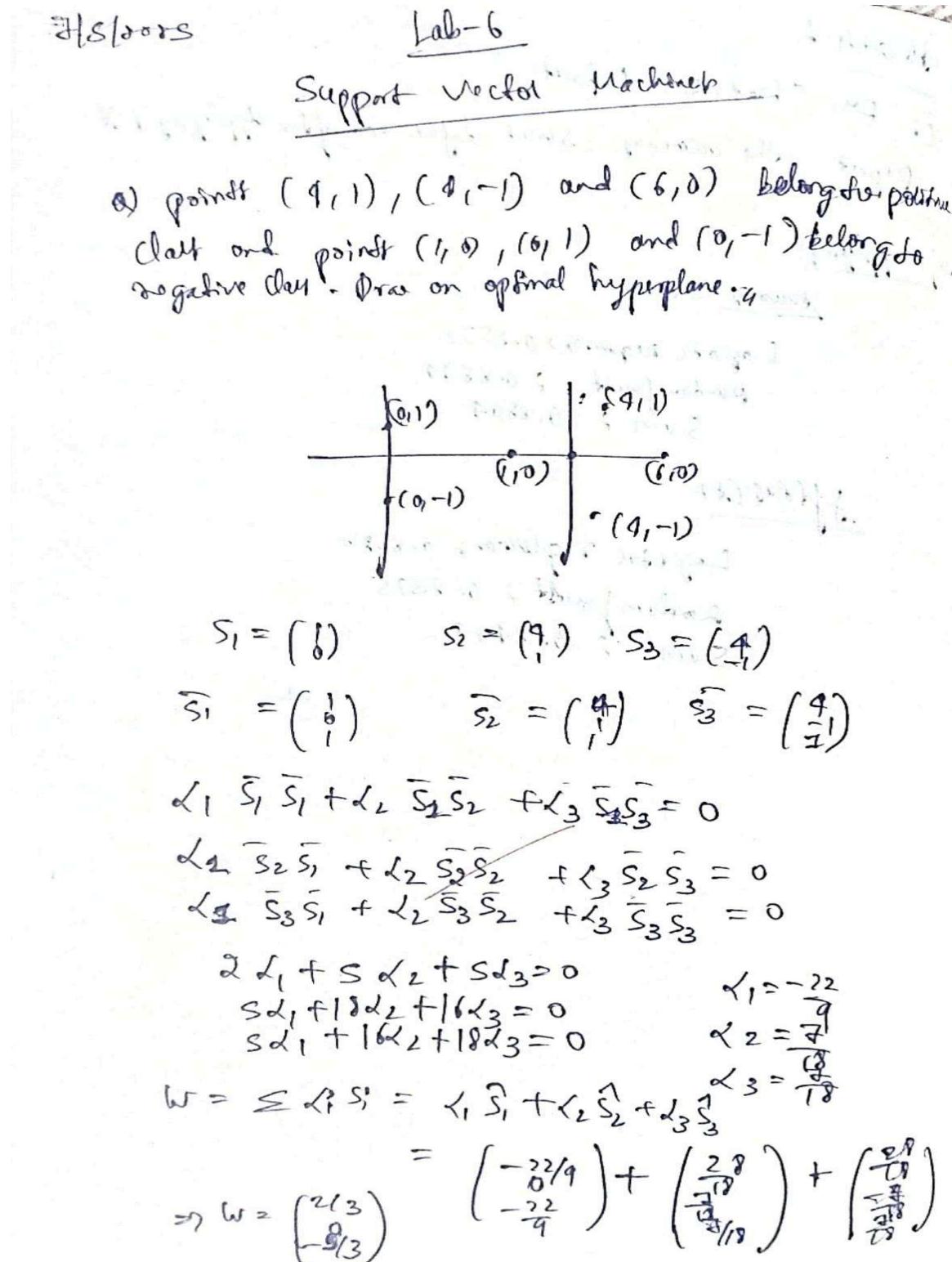
    # Visualize
    visualize_knn(data, test, result, new_point=new_point,
new_label=new_result)

```

Program 7

Build Support vector machine model for a given dataset

Screenshot



$$\text{Eqn} \Rightarrow \frac{2}{3}x_1 + 0.x_2 + (-\frac{5}{3}) = 0$$

$$\frac{2}{3}x_1 - \frac{5}{3} = 0$$

$$2x_1 - 5 = 0$$

$$x_1 = \frac{5}{2}$$

$$x_1 = 2.5$$

Sign of hyperplane and boundary, $x_1 = 2.5$

(Q) For linearly separable what is accuracy of classifier using linear and RBF kernel? Which of them has better performance.

(A) Linear Kernel = 100%
RBF Kernel = 100%

Generally, RBF performs better than linear method.
Small datasets get equal accuracy.
With non-linearly separable datasets,
Linear kernel draws a straight hyperplane and
Struggles with curved boundaries.
RBF allows for more accurate and flexible
boundaries.

(Q) For digit recognition (SV) dataset present
Indirect conclusion matrix, one share only
Specific test rapidly confused class others
What AUC score did it reflect model
Performance? How does performance of SVM
model of dataset differ to its performance on

(M) The IRIS dataset
1. Test data predicted ok, how may that look
2. Actual vs Predicted
3. Confusion matrix

Interpretations:

High values on diagonal = good performance
Off-diagonal values = misclassification.
Letter like O, C and G, A might get confused
Similar features.

• AUC Score measure how well the model
distinguishes between classes
An AUC \rightarrow 1 (Perfect). If $AUC = 0.5$ (Randomly).

Underinterpretations:

• perfect: when $AUC = 0.95$ for 1000
• failure: when $AUC = 0.85 - 0.93$

IRIS model is much smaller and simpler dataset somewhat
letter recognition is much more complex.
More classes
more complex
more ambiguous

SVM performs better on IRIS, but still does not perform well.

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 usage
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]

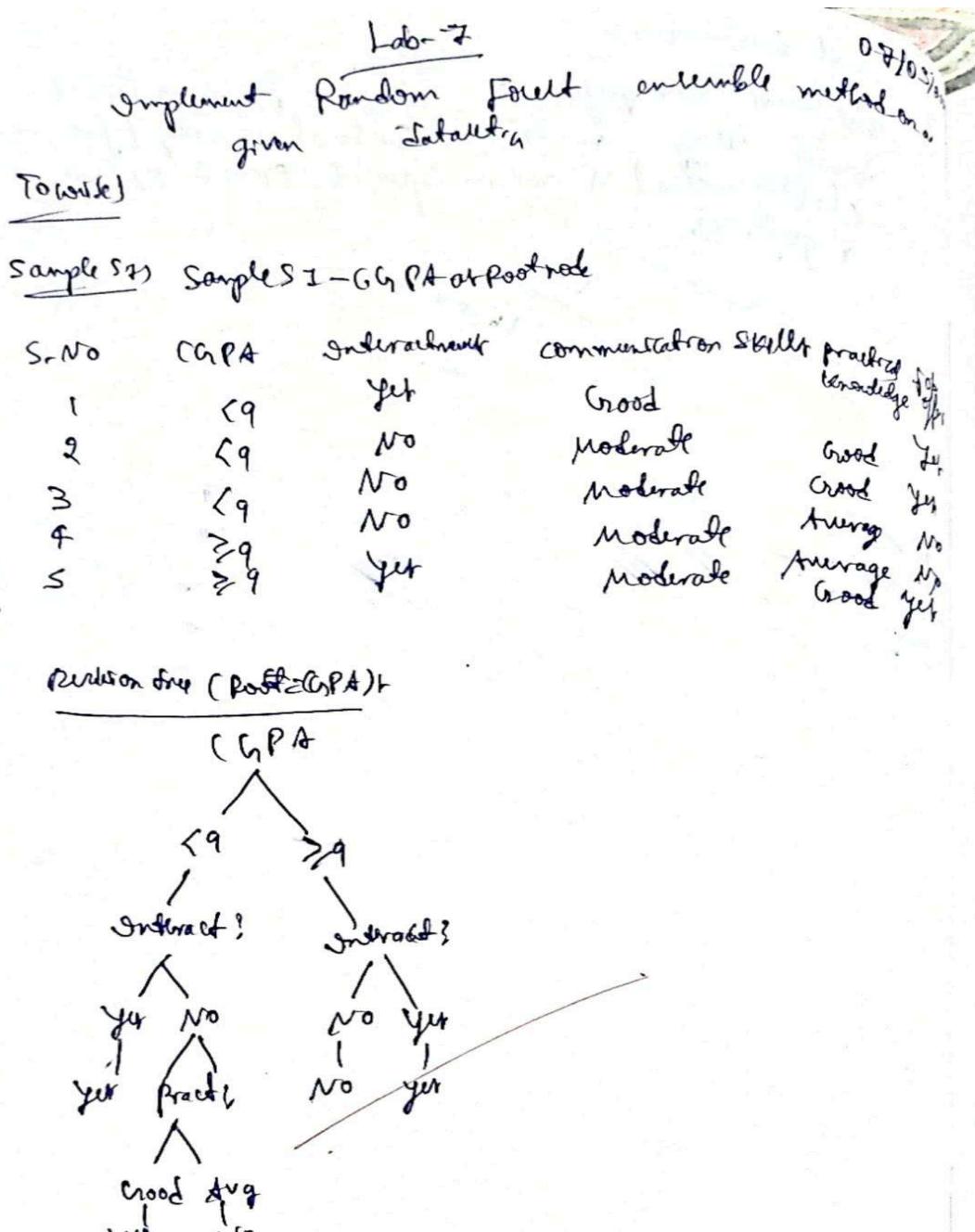
# Visualize
svm.visualize(X, y, new_point=new_point[0], prediction=prediction)

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

Program 8

Implement Random forest ensemble method on a given dataset

Screenshot



For GPA < 9:

- v of $\text{GPA} < 9 \rightarrow \text{JobOffer} = \text{Yes}$
- v of $\text{GPA} < 9 \rightarrow \text{JobOffer} = \text{No}$
- v of $\text{practical Knowledge Good} \rightarrow \text{JobOffer} = \text{Yes}$
- v of $\text{EU} (\text{Average}) \rightarrow \text{JobOffer} = \text{No}$.

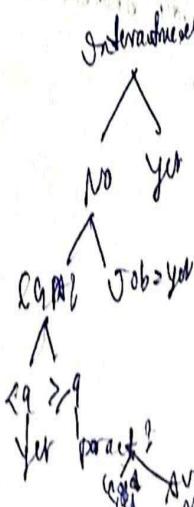
v For $\text{GPA} \geq 9$:

- v of $\text{GPA} \geq 9 \rightarrow \text{JobOffer} = \text{No}$
- v of $\text{GPA} \geq 9 \rightarrow \text{JobOffer} = \text{Yes}$

Sampled 6 \rightarrow $\text{GPA} \geq 9$ at Root Node

Sl.No	GPA	Interactions	Communication	Practical Knowledge	JobOffer
2	< 9	No	Skilled	Good	Yes
3	≥ 9	No	Moderate	Average	No
4	≥ 9	No	Moderate	Average	No
5	≥ 9	Yet	Moderate	Good	Yes
6	≥ 9	Yet	Moderate	Good	Yes

Precision tree ($\text{Root} = \text{GPA} \geq 9$)



+ If $\text{GPA} < 9 \rightarrow \text{JobOffer} = \text{Yes}$

+ If $\text{GPA} \geq 9 \rightarrow \text{JobOffer} = \text{No}$

+ of $\text{GPA} \geq 9 \rightarrow \text{JobOffer} = \text{Yes}$

+ of $\text{GPA} \geq 9$:

+ of $\text{practical Knowledge Good} \rightarrow \text{JobOffer} = \text{Yes}$

+ of $\text{EU} (\text{Average}) \rightarrow \text{JobOffer} = \text{No}$,

To write

After building the RF model, write the answer for the following question in your solution book,

1. for "GPA" dataset

What is the best accuracy score and confusion matrix of the classifier you obtained and why having such?

& Best accuracy Score: 1.0 (or 100%)

& Number of Trees (n_estimators): 1.

& Confusion matrix:

$[1.0 \ 0.0]$

$[0.0 \ 1.0]$

$[0.0 \ 1.0]$

This result was obtained by first finding the number of trees from 100, the best accuracy was achieved with just 1 tree which indicates that the dataset is highly and easily separable.

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))
```

Program 9

Implement Boosting ensemble method on a given dataset

Screenshot

Lab 8
Implementation of Boosting ensemble method
on given dataset.

To write
Step by step solution for decision stump on CGPA

Step 1 Data Input

Index	CGPA	Actual Label	Predicted Label	Comments	Step 1
0	$>=9$	Yes	Good	Good	Yes
1	<9	No	Good	Moderate	Yes
2	$>=9$	No	Average	Moderate	No
3	<9	No	Average	Good	No
4	$>=9$	Yes	Good	Moderate	Yes
5	$>=9$	Yes	Good	Moderate	Yes

Step 1 calculate step weights

$$w_1 = \frac{1}{5} \approx 0.167 \text{ for each sample.}$$

Step 2.1 Build decision stump on CGPA

* condition $CGPA >= 9 \rightarrow$ predict Yes, else predict No

Apply first rule for all rows

Index	CGPA	Actual Label	Predicted Label	Comments
0	$>=9$	Yes	Yes	✓
1	<9	No	No	✗
2	$>=9$	No	Yes	✗
3	<9	No	No	✓
4	$>=9$	Yes	Yes	✓
5	$>=9$	Yes	Yes	✓

Step 2 Calculate weighted error.

$$\text{Error} = \sum_{\text{all classes}} w_1 + w_2 = \frac{1}{6} + \frac{1}{6} = \frac{1}{3}$$

Step 3 Compute stump weight (α_{stump})

$$\alpha_{\text{stump}} = \frac{1}{2} \ln \left(\frac{1 - \text{Error}}{\text{Error}} \right) = \frac{1}{2} \ln \left(\frac{1 - 0.333}{0.333} \right) = \frac{1}{2} \ln(2) \approx 0.3966$$

Final accuracy rate in observation booklet

Decision Stump in C4.5

* Rule: if $(C4.5)_{\text{leaf}} \rightarrow$ predict yes, else no;
+ weighted error: 0.3966

+ Alpha (stump weight): 0.3966

✓ correctly classified points: 0, 3, 9, 5

✗ misclassified points: 7, 2, 4

Towards

1. Fit "standardized" data points

what is the first accuracy score and confusion matrix
the classifier you obtained and along how many
fruits?

1. wrong classification > 10%

Accuracy: 8 / 80 %

Confusion matrix:

[0 78 63]

[1 14 10 11]

9. But without doing fine tuning (in estimate for range 1-73,
steps 2, 3)

but accuracy: 83.33%

but confusion matrix:

[0 0 1 3 9]

[1 2 3 11 8]

code:

```
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()

```

Program 10

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

Screenshot

Lab-9
Build k-Means algorithm to cluster a set of data stored in a CSV file.

To work
initial cluster centers
 $C_1 = (1, 0, 1, 0)$
 $C_2 = (5, 5, 2, 0)$

$R_1 = (1, 0, 1, 0)$
 $R_2 = (1, 5, 2, 0)$
 $R_3 = (3, 0, 1, 0)$
 $R_4 = (5, 0, 2, 0)$
 $R_5 = (5, 5, 1, 0)$
 $R_6 = (4, 5, 5, 0)$
 $R_7 = (3, 5, 1, 0)$

cluster after iteration 1
 $C_1 \approx R_1, R_2, R_3$
 $C_2 \approx R_4, R_5, R_6, R_7$

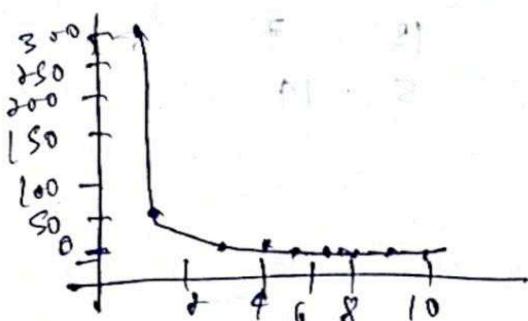
New centers
 $C_1 \approx (1.83, 2.33)$
 $C_2 \approx (4.625, 5.375)$

Iteration 2
Final cluster after iteration 2
 $C_1: R_1, R_2, R_3$
 $C_2: R_4, R_5, R_6, R_7$

TODAY

(r) For "CIFINCON" dataset draw elbow plot.

draw the elbow plot at what the optimal K value obtained



optimal K value obtained was $K = 3.4$



done by keeping back one testing record

[Testing] = 3 misclassified

Overall classification accuracy: $(16 - 3) / 16 = 81\%$

code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import datasets
import seaborn as sns
from sklearn.cluster import KMeans

iris = datasets.load_iris()
print("Dataset loaded successfully")

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

#Top values of Dataset
# Data.head()
x=Data.iloc[:,0:3].values

css=[]

# # Finding inertia on various k values
# for i in range(1,8):
#     kmeans=KMeans(n_clusters = i, init = 'k-means++',
#                 max_iter = 100, n_init = 10, random_state = 0).fit(x)
#     css.append(kmeans.inertia_)

#Applying Kmeans classifier
kmeans = KMeans(n_clusters=3,init = 'k-means++', max_iter = 100, n_init = 10,
random_state = 0)
y_kmeans = kmeans.fit_predict(x)
kmeans.cluster_centers_

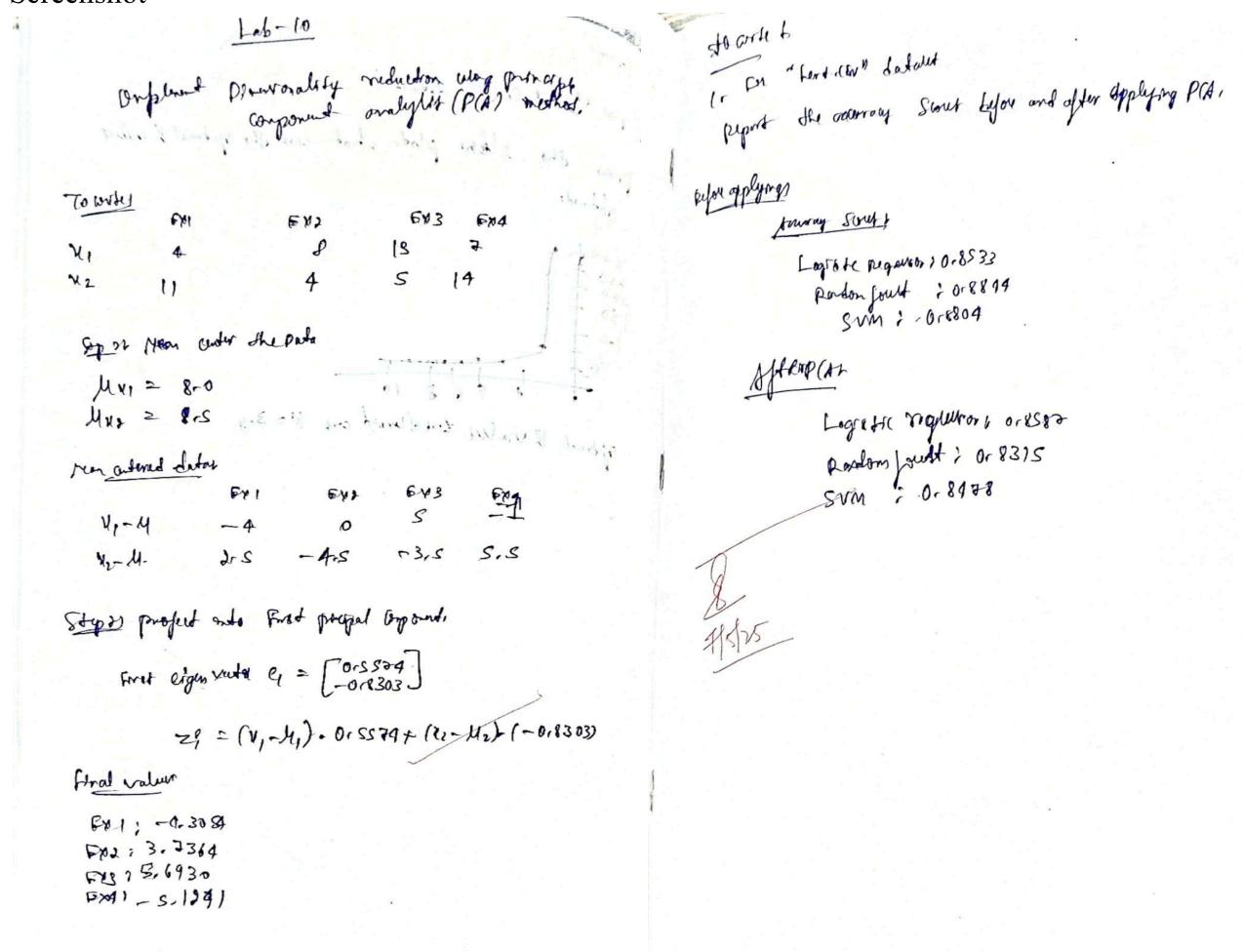
# Visualising the clusters - On the first two columns
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1],
            s = 100, c = 'red', label = 'Iris-setosa')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1],
            s = 100, c = 'blue', label = 'Iris-versicolour')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1],
            s = 100, c = 'green', label = 'Iris-virginica')

# Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1],
            s = 100, c = 'black', label = 'Centroids')
plt.legend()
```

Program 11

Implement Dimensionality reduction using Principal Component Analysis (PCA) method

Screenshot



code:

```
# 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_)

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

```

**VISVESVARAYA TECHNOLOGICAL
UNIVERSITY**
“JnanaSangama”, Belgaum -590014, Karnataka.



**LAB REPORT
on**

Machine Learning (23CS6PCMAL)

Submitted by

SHASHANK PATEL C J (1BM22CS255)

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 June-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 **Shashank Patel C J (1BM22CS255)**, who is a bona fide 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: Prof. Shravya A R Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Sl. No.	Date	Experiment Title	Page No.
1	5-03-2025	Write a python program to import and export data using Pandas library functions	1-6
2	5-3-2025	Demonstrate various data pre-processing techniques for a given dataset	7-16
3	12-3-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	17-22
4	19-3-2025	Build Logistic Regression Model for a given dataset	23-26
5	2-4-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample	27-33
6	2-4-2025	Build KNN Classification model for a given dataset	34-38
7	9-4-2025	Build Support vector machine model for a given dataset	39-43
8	16-5-2025	Implement Random forest ensemble method on a given dataset	44-46
9	7-5-2025	Implement Boosting ensemble method on a given dataset	47-51
10	7-5-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file	52-54
11	7-5-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method	55-56