

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



LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Pranav Anantha Rao (1BM22CS201)

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 **Pranav Anantha Rao (1BM22CS201)**, who is a bonafide student of **B.M.S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

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

Index

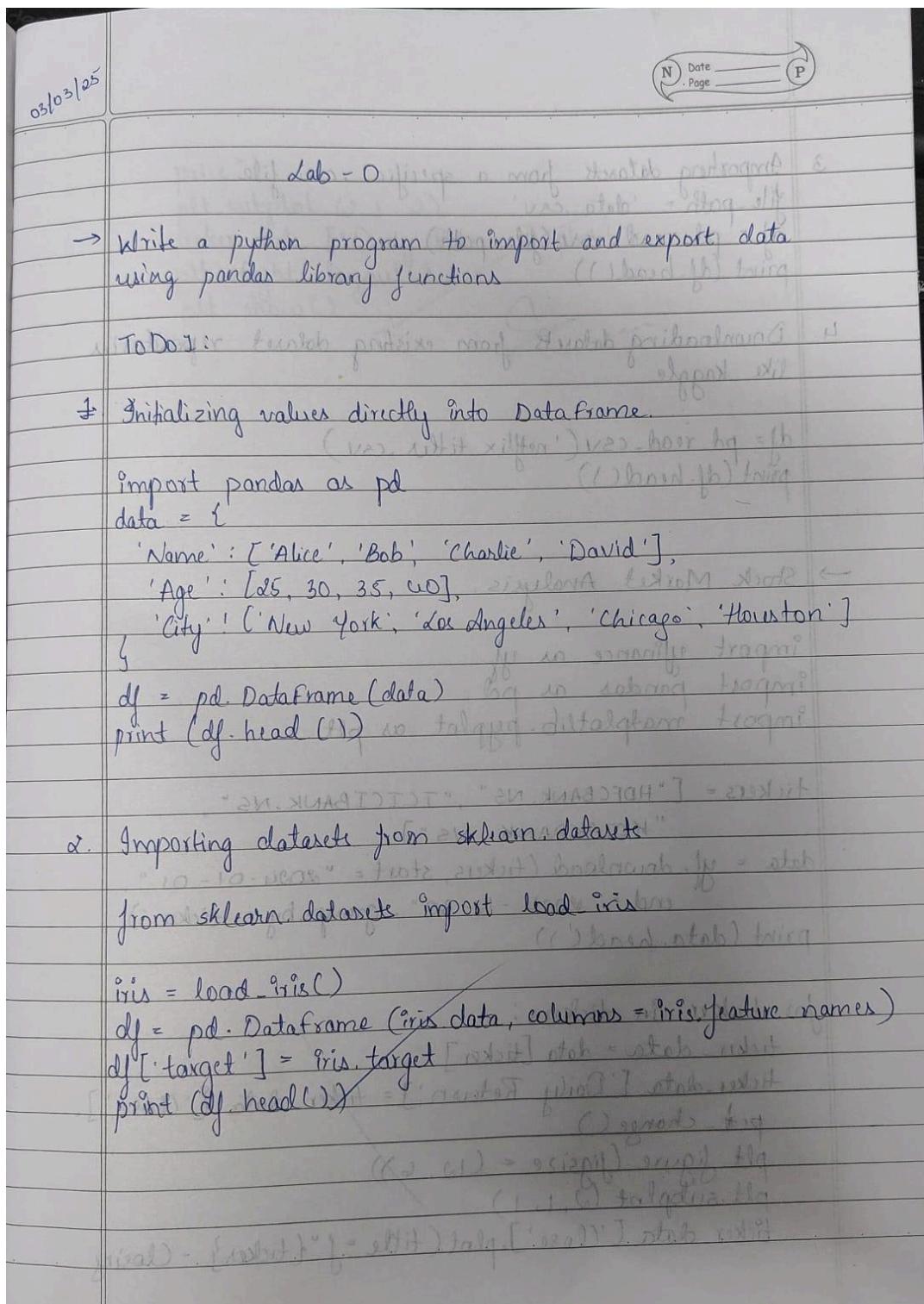
Sl. No.	Date	Experiment Title	Page No.
1	21-2-2025	Write a python program to import and export data using Pandas library functions	
2	3-3-2025	Demonstrate various data pre-processing techniques for a given dataset	
3	10-3-2025	Implement Linear and Multi-Linear Regression algorithm using appropriate dataset	
4	17-3-2025	Build Logistic Regression Model for a given dataset	
5	24-3-2025	Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample.	
6	7-4-2025	Build KNN Classification model for a given dataset.	
7	21-4-2025	Build Support vector machine model for a given dataset	
8	5-5-2025	Implement Random forest ensemble method on a given dataset.	
9	5-5-2025	Implement Boosting ensemble method on a given dataset.	
10	12-5-2025	Build k-Means algorithm to cluster a set of data stored in a .CSV file.	
11	12-5-2025	Implement Dimensionality reduction using Principal Component Analysis (PCA) method.	

Github Link: <https://github.com/PranavRao30/6D-ML-1BM22CS201>

Program 1

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

Screenshot:



Code:

```
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David'],
    'Age': [25, 30, 35, 40],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']
}
df = pd.DataFrame(data)
print("Sample data:")
print(df.head())
```

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

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

```
file_path = 'mobiles-dataset-2025.csv'
df = pd.read_csv(file_path, encoding='latin-1') # or 'cp1252' or other suitable encoding
print("Sample data:")
print(df.head())
```

```
import pandas as pd
data = {
```

```
'USN': ['IS001','IS002','IS003','IS004','IS005'],
'Name': ['Alice', 'Bob', 'Charlie', 'David','Eve'],
'Marks': [25, 30, 35, 40,45]
}

df = pd.DataFrame(data)

print("Sample data:")
print(df.head())


file_path = 'sample_sales_data.csv'
df = pd.read_csv(file_path)
print("Sample data:")
print(df.head())
print("\n")

df = pd.read_csv("/content/dataset-of-diabetes .csv",encoding='latin-1')
print("Sample data:")
print(df.head())
print("\n")

df =pd.read_csv('sample_sales_data.csv')
print("Sample data:")
print(df.head())
df.to_csv('output.csv',index=False)
print("Data saved to output.csv")

sales_df =pd.read_csv('sample_sales_data.csv')
print("Sample data:")
print(sales_df.head())
sales_by_region =sales_df.groupby('Region')['Sales'].sum()
print("\nTotal sales by region:")
print(sales_by_region)

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

```
print("\nBest-selling products by quantity:")
print(best_selling_products)
sales_by_region.to_csv('sales_by_region.csv')
best_selling_products.to_csv('best_selling_products.csv')
print("Data saved to sales_by_region.csv and best_selling_products.csv")
```

```
import yfinance as yf
import matplotlib.pyplot as plt
tickers = ["RELIANCE.NS", "TCS.NS", "INFY.NS"]
data = yf.download(tickers, start="2022-10-01", end="2023-10-01",
                   group_by='ticker')
print("First 5 rows of the dataset:")
print(data.head())
print("\nShape of the dataset:")
print(data.shape)
print("\nColumn names:")
print(data.columns)
print("\n")
reliance_data = data['RELIANCE.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
print("\n")
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
reliance_data['Close'].plot(title="Reliance Industries - Closing Price")
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="Reliance Industries - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
reliance_data.to_csv('reliance_stock_data.csv')
```

```
tickers = ["HDFCBANK.NS", "ICICI.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-30",
                   group_by='ticker')
print("First 5 rows of the dataset:")
print(data.head())
print("\nShape of the dataset:")
print(data.shape)
print("\nColumn names:")
print(data.columns)
print("\n")
reliance_data = data['HDFCBANK.NS']
print("\nSummary statistics for Reliance Industries:")
print(reliance_data.describe())
reliance_data['Daily Return'] = reliance_data['Close'].pct_change()
print("\n")
plt.figure(figsize=(12, 6))
plt.subplot(2, 1, 1)
reliance_data['Close'].plot(title="HDFC Industries - Closing Price")
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="HDFCIndustries - Daily Returns", color='red')
plt.tight_layout()
plt.show()
reliance_data.to_csv('hdfc_stock_data.csv')
print("\nhdfc stock data saved to 'hdfc_stock_data.csv'.")
```

Program 2

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

Screenshot:

Lab 1

1. housing.csv:

```
import pandas as pd
# i
df = pd.read_csv('housing.csv')
# ii
print(df.info())
# iii
print(df.describe())
# iv
print(df['ocean_proximity'].value_counts())
# v
missing_values = df.isnull().sum()
att = missing_values[missing_values > 0]
print(att)
```

→ Diabetes and adult income datasets:

1. Diabetes dataset: Glucose, Blood pressure, skin thickness etc had missing values. Handled by dropna()

adult income: Work, class, occupation had missing values. Handled by dropna()

2. Diabetes: No categorical column
adult Income: Work, class, education, occupation
3. Min max scaling scales data to a fixed range, usually [0, 1], while standardization centers data around 0 with a standard deviation of 1. ~~The min-max scaling when data has a known range and standardization when data has outliers.~~

Code:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.impute import SimpleImputer

try:
    diabetes_df = pd.read_csv('diabetes.csv')
    adult_df = pd.read_csv('adult.csv')
except FileNotFoundError:
    print("Error: Please upload 'diabetes.csv' and 'adult.csv' to your Google Colab environment.")
    exit()

diabetes_df.head(10)
adult_df.head(10)
```

```
diabetes_df.shape  
adult_df.shape
```

#Handling Missing Values

```
diabetes_numeric_cols = diabetes_df.select_dtypes(include=[np.number]).columns  
diabetes_categorical_cols = diabetes_df.select_dtypes(exclude=[np.number]).columns
```

```
adult_numeric_cols = adult_df.select_dtypes(include=[np.number]).columns  
adult_categorical_cols = adult_df.select_dtypes(exclude=[np.number]).columns
```

```
diabetes_numeric_imputer = SimpleImputer(strategy='mean')  
adult_numeric_imputer = SimpleImputer(strategy='mean')  
diabetes_df[diabetes_numeric_cols] =  
diabetes_numeric_imputer.fit_transform(diabetes_df[diabetes_numeric_cols])  
adult_df[adult_numeric_cols] = adult_numeric_imputer.fit_transform(adult_df[adult_numeric_cols])
```

```
diabetes_categorical_imputer = SimpleImputer(strategy='most_frequent')  
adult_categorical_imputer = SimpleImputer(strategy='most_frequent')  
diabetes_df[diabetes_categorical_cols] =  
diabetes_categorical_imputer.fit_transform(diabetes_df[diabetes_categorical_cols])  
adult_df[adult_categorical_cols] =  
adult_categorical_imputer.fit_transform(adult_df[adult_categorical_cols])
```

```
print("Missing values in Diabetes dataset after imputation:")  
print(diabetes_df.isnull().sum())  
print("Missing values in Adult Income dataset after imputation:")  
print(adult_df.isnull().sum())
```

```
adult_df.replace("?", np.nan, inplace=True)  
print("Missing values in Adult Income dataset after replacing '?'")  
print(adult_df.isnull().sum())
```

```

from sklearn.impute import SimpleImputer

# Identify numeric and categorical columns
adult_numeric_cols = adult_df.select_dtypes(include=[np.number]).columns
adult_categorical_cols = adult_df.select_dtypes(exclude=[np.number]).columns

# Handle missing values in numeric columns using mean imputation
adult_numeric_imputer = SimpleImputer(strategy='mean')
adult_df[adult_numeric_cols] = adult_numeric_imputer.fit_transform(adult_df[adult_numeric_cols])

# Handle missing values in categorical columns using most frequent imputation
adult_categorical_imputer = SimpleImputer(strategy='most_frequent')
adult_df[adult_categorical_cols] =
adult_categorical_imputer.fit_transform(adult_df[adult_categorical_cols])
print("Missing values in Adult Income dataset after imputation:")
print(adult_df.isnull().sum())

from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

# Encode categorical columns in Diabetes dataset
for col in diabetes_categorical_cols:
    diabetes_df[col] = label_encoder.fit_transform(diabetes_df[col])

# Encode categorical columns in Adult Income dataset
for col in adult_categorical_cols:
    adult_df[col] = label_encoder.fit_transform(adult_df[col])
print("Encoded columns in Diabetes dataset:")
print(diabetes_df.head())
print("Encoded columns in Adult Income dataset:")
print(adult_df.head())

```

#Handling outliers

```
def remove_outliers(df):
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
    df_no_outliers = df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
    return df_no_outliers
```

```
diabetes_df_no_outliers = remove_outliers(diabetes_df)
```

```
adult_df_no_outliers = remove_outliers(adult_df)
```

```
print("Diabetes dataset shape after removing outliers:", diabetes_df_no_outliers.shape)
```

```
print("Adult Income dataset shape after removing outliers:", adult_df_no_outliers.shape)
```

#Min-max scaling

```
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
diabetes_scaled_minmax = pd.DataFrame(min_max_scaler.fit_transform(diabetes_df_no_outliers),
columns=diabetes_df_no_outliers.columns)
adult_scaled_minmax = pd.DataFrame(min_max_scaler.fit_transform(adult_df_no_outliers),
columns=adult_df_no_outliers.columns)
print("Diabetes dataset after Min-Max scaling:")
print(diabetes_scaled_minmax.head())
print("Adult Income dataset after Min-Max scaling:")
print(adult_scaled_minmax.head())
```

Initialize Standard Scaler

```
from sklearn.preprocessing import StandardScaler
standard_scaler = StandardScaler()
diabetes_scaled_standard = pd.DataFrame(standard_scaler.fit_transform(diabetes_df_no_outliers),
columns=diabetes_df_no_outliers.columns)
adult_scaled_standard = pd.DataFrame(standard_scaler.fit_transform(adult_df_no_outliers),
```

```
columns=adult_df_no_outliers.columns)
print("Diabetes dataset after Standard scaling:")
print(diabetes_scaled_standard.head())
print("Adult Income dataset after Standard scaling:")
print(adult_scaled_standard.head())
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset.

Screenshot:

→ Linear Regression :

Algorithm :

1. Let $(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)$ be the datapoints
2. Let β_0 be intercept and β_1 be coefficient.
3. $y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$ $\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$ $x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$
4. Rearrange :
5. Final $\beta = ((x^T \cdot x)^{-1} \cdot x^T \cdot y) = y$
6. Calculate β_0 and β_1 from β and draw the best fit line.

→ Multiple linear Regression

Algorithm:

1. Let $(x_{11}, x_{12}, \dots, x_{1m}, y_1)$,
 $(x_{21}, x_{22}, \dots, x_{2m}, y_2)$,

\vdots
 $(x_{n1}, x_{n2}, \dots, x_{nm}, y_n)$ be the

datapoint.

$\beta_0, \beta_1, \dots, \beta_n$

2. Let β_0 be intercept and β_1, \dots, β_n be the coefficient.

3. $Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}, \quad X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1m} \\ x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & & & \\ x_{n1} & x_{n2} & \dots & x_{nm} \end{bmatrix}$

4. Rearrange:

$$Y = \begin{bmatrix} 1 + x_{11} + x_{12} + \dots + x_{1m} \\ 1 + x_{21} + x_{22} + \dots + x_{2m} \\ \vdots \\ 1 + x_{n1} + x_{n2} + \dots + x_{nm} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

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

5. Obtain values of $\beta_0, \beta_1, \dots, \beta_n$ from β

Code:

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

df = pd.read_csv('housing_area_price.csv')
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
reg = linear_model.LinearRegression()
reg.fit(new_df, price)
```

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

```
reg.predict([[3300]])
reg.coef_
reg.intercept_
3300 * 135.78767123 + 180616.43835616432
```

#(2) Predict price of a home with area = 5000 sqr ft

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

```
df = pd.read_csv('homeprices_Multiple_LR.csv')
df.bedrooms.median()
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())
reg = linear_model.LinearRegression()
reg.fit(df.drop('price', axis='columns'), df.price)
reg.coef_
reg.intercept_
```

```
#Find price of home with 3000 sqr ft area, 3 bedrooms, 40 year old  
reg.predict([[3000, 3, 40]])  
112.06244194*3000 + 23388.88007794*3 + -3231.71790863*40 + 221323.00186540384
```

```
df = pd.read_csv('canada_per_capita_income.csv')  
print(df.head())  
X = df[['year']]  
y = df['per capita income (US$)']  
reg = LinearRegression()  
reg.fit(X, y)  
predicted_income_2020 = reg.predict([[2020]])  
print(f'Predicted per capita income for Canada in 2020: {predicted_income_2020[0]:.2f}')
```

```
plt.scatter(X, y, color='blue')  
plt.plot(X, reg.predict(X), color='red')  
plt.xlabel('Year')  
plt.ylabel('Per Capita Income')  
plt.title('Per Capita Income in Canada Over the Years')  
plt.show()
```

```
df = pd.read_csv('salary.csv')  
print(df.head())  
print("Missing values in the dataset:")  
print(df.isnull().sum())
```

```
df['YearsExperience'] = df['YearsExperience'].fillna(df['YearsExperience'].median())  
print("\nMissing values after filling:")  
print(df.isnull().sum())  
X = df[['YearsExperience']]  
y = df['Salary']  
reg = LinearRegression()
```

```

reg.fit(X, y)
predicted_salary_12_years = reg.predict([[12]])
print(f"\nPredicted salary for an employee with 12 years of experience:
${predicted_salary_12_years[0]:,.2f}")

plt.scatter(X, y, color='blue')
plt.plot(X, reg.predict(X), color='red')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.title('Salary vs. Years of Experience')
plt.show()

def convert_to_numeric(value):
    word_to_num = {
        'zero': 0, 'one': 1, 'two': 2, 'three': 3, 'four': 4, 'five': 5,
        'six': 6, 'seven': 7, 'eight': 8, 'nine': 9, 'ten': 10,
        'eleven': 11, 'twelve': 12, 'thirteen': 13, 'fourteen': 14,
        'fifteen': 15
    }
    return word_to_num.get(value.lower(), value) if isinstance(value, str) else value

df_hiring = pd.read_csv('hirings.csv')
print(df.head())
df_hiring['experience'] = df_hiring['experience'].apply(convert_to_numeric)

df_hiring['experience'].fillna(0, inplace=True)
df_hiring['test_score(out of 10)'].fillna(df_hiring['test_score(out of 10)'].median(), inplace=True)
df_hiring['interview_score(out of 10)'].fillna(df_hiring['interview_score(out of 10)'].median(),
inplace=True)
X_hiring = df_hiring[['experience', 'test_score(out of 10)', 'interview_score(out of 10)']]
y_hiring = df_hiring['salary($)']
reg_hiring = LinearRegression()

```

```

reg_hiring.fit(X_hiring, y_hiring)
candidates = np.array([[2, 9, 6], [12, 10, 10]])
predicted_salaries = reg_hiring.predict(candidates)

for i, candidate in enumerate(candidates):
    print(f"\nPredicted salary for candidate with {candidate[0]} yrs experience, {candidate[1]} test score,
{candidate[2]} interview score: {predicted_salaries[i]:.2f} USD")
plt.scatter(y_hiring, reg_hiring.predict(X_hiring), color='blue', label='Predicted vs Actual')
plt.xlabel("Actual Salary")
plt.ylabel("Predicted Salary")
plt.title("Actual vs Predicted Salary")
plt.legend()
plt.show()

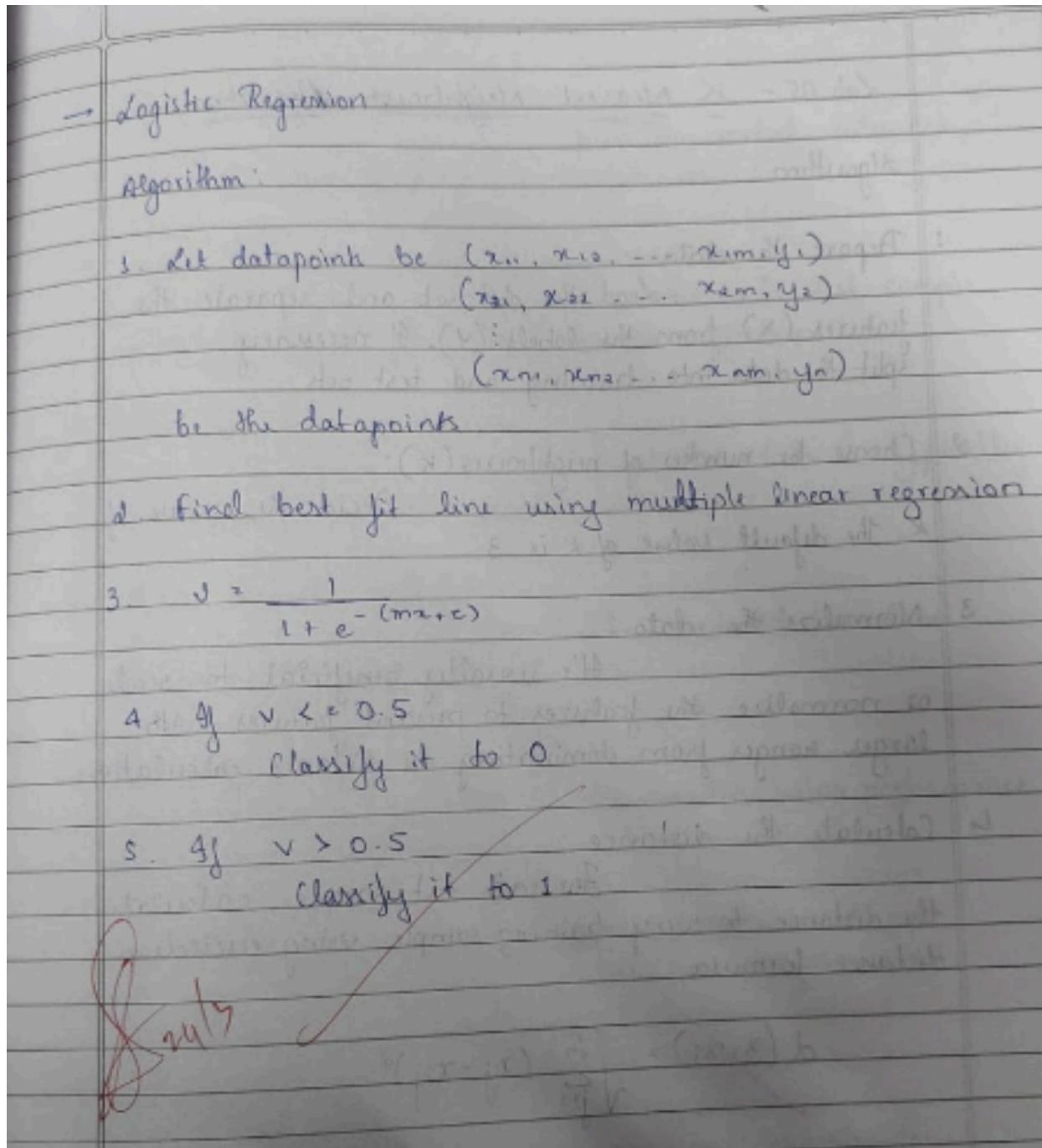
df_companies = pd.read_csv('1000_Companies.csv')
print(df.head())
label_encoder = LabelEncoder()
df_companies['State'] = label_encoder.fit_transform(df_companies['State'])
X_companies = df_companies[['R&D Spend', 'Administration', 'Marketing Spend', 'State']]
y_companies = df_companies['Profit']
df_companies.fillna(df_companies.median(), inplace=True)
reg_companies = LinearRegression()
reg_companies.fit(X_companies, y_companies)
input_data = np.array([[91694.48, 515841.3, 11931.24, label_encoder.transform(['Florida'])[0]]])
predicted_profit = reg_companies.predict(input_data)
print(f"Predicted profit: {predicted_profit[0]:.2f} USD")
plt.scatter(y_companies, reg_companies.predict(X_companies), color='blue', label='Predicted vs
Actual')
plt.xlabel("Actual Profit")
plt.ylabel("Predicted Profit")
plt.title("Actual vs Predicted Profit")
plt.legend()
plt.show()

```

Program 4

Build Logistic Regression Model for a given dataset.

Screenshot:



Code:

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

df = pd.read_csv("HR_comma_sep.csv")
print(df.info())
numericCols = df.select_dtypes(include=['float64', 'int64']).columns
plt.figure(figsize=(10, 8))
sns.heatmap(df[numericCols].corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Matrix (Numeric Features)")
plt.show()

plt.figure(figsize=(8, 6))
sns.countplot(x='salary', hue='left', data=df)
plt.title("Impact of Salary on Employee Retention")
plt.xlabel("Salary Level")
plt.ylabel("Employee Count")
plt.show()

import pandas as pd
df = pd.read_csv("zoo-data.csv")
print(df.info())
print(df.head())
print(df.isnull().sum())
df.drop(columns=['animal_name'], inplace=True)
X = df.drop(columns=['class_type'])
y = df['class_type']

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
```

```
X_train, X_test, y_train, y_test = train_test_split(  
    X, y, test_size=0.2, random_state=42, stratify=y)  
scaler = StandardScaler()  
X_train = scaler.fit_transform(X_train)  
X_test = scaler.transform(X_test)  
logreg = LogisticRegression(max_iter=200, multi_class='multinomial', solver='lbfgs')  
logreg.fit(X_train, y_train)  
  
from sklearn.metrics import accuracy_score  
y_pred = logreg.predict(X_test)  
accuracy = accuracy_score(y_test, y_pred)  
print(f"Model Accuracy: {accuracy:.2f}")  
  
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay  
import matplotlib.pyplot as plt  
cm = confusion_matrix(y_test, y_pred)  
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=logreg.classes_)  
disp.plot(cmap=plt.cm.Blues)  
plt.title("Confusion Matrix for Zoo Animal Classification")  
plt.show()  
  
y_pred = logreg.predict(X_test)  
pred_classes = [class_mapping[pred] for pred in y_pred]  
print("Predicted Classes:", pred_classes)  
  
import seaborn as sns  
import matplotlib.pyplot as plt  
sns.countplot(x='class_type', data=df)  
plt.title("Class Distribution of Animals in Zoo Dataset")  
plt.xlabel("Class Type")  
plt.ylabel("Count")  
plt.show()
```

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
cm = confusion_matrix(y_test, y_pred)
class_labels = [class_mapping[num] for num in logreg.classes_]
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=class_labels)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix with Class Names")
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:

01/31/25

Lab 2

Implement ID3 algorithm using decision tree algorithm

the appropriate dataset for reading, building a decision tree and applying this knowledge to classify a new example

→ algorithm

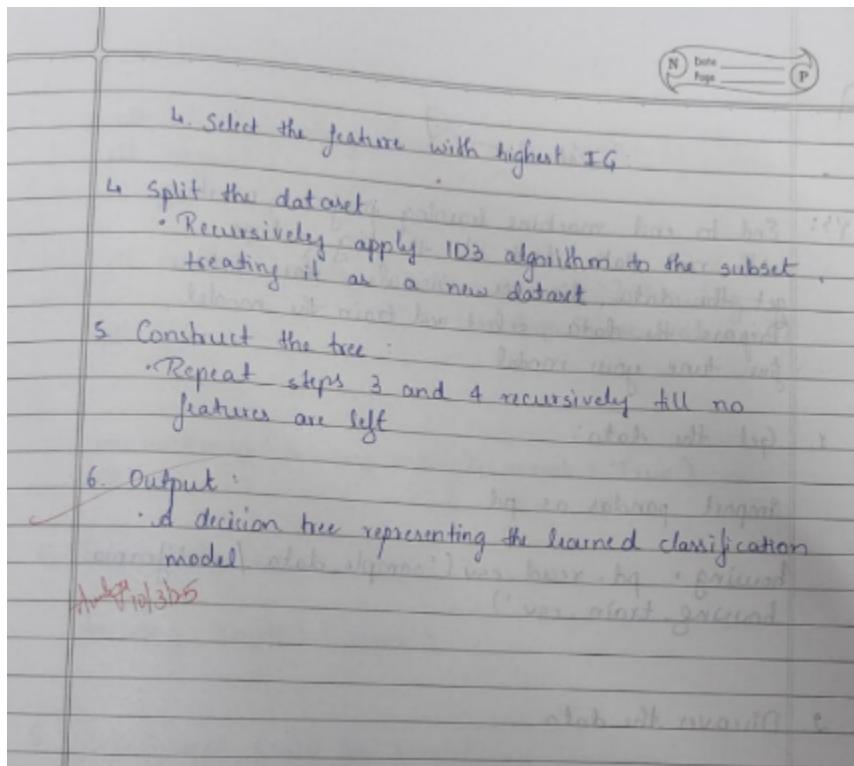
1. Input:
 - a set of training examples, each with a target class
 - a set of features with possible values
2. Initialize:
 - Start with the entire dataset as the root node
3. Recursion:
 - for each node
 - 2. Base cases:
 - If all instances in the dataset belong to the same classes, create a leaf node with that class
 - If there are no features left to split on, create a leaf node with the majority class
 - 3. Calculate Entropy:

$$H(S) = - \sum_{i=1}^n p_i \log_2 p_i$$

$p_i \rightarrow$ probability of class i

3. Calculate Information Gain:

$$IG(S, A) = H(S) - \sum_{A \in Values(A)} \frac{|S_A|}{|S|} H(S_A)$$



Code:

```

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score, f1_score
from sklearn.preprocessing import LabelEncoder

def train_and_evaluate_iris():
    iris_df = pd.read_csv("iris.csv")
    X = iris_df.drop(columns=["species"])
    y = iris_df["species"]
    y_le = LabelEncoder()
    y = y_le.fit_transform(y)
  
```

```

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Evaluating the model
acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, average='weighted')
rec = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
cm = confusion_matrix(y_test, y_pred)

print("IRIS Dataset Classification:")
print(f"Accuracy Score: {acc:.4f}")
print(f"Precision Score: {prec:.4f}")
print(f"Recall Score: {rec:.4f}")
print(f"F1 Score: {f1:.4f}")

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=y_le.classes_,
yticklabels=y_le.classes_)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix: iris.csv")
plt.show()

train_and_evaluate_iris()

def train_and_evaluate_drug():
    drug_df = pd.read_csv("drug.csv")
    categorical_features = ["Sex", "BP", "Cholesterol"]
    label_encoders = {}

```

```

for col in categorical_features:
    le = LabelEncoder()
    drug_df[col] = le.fit_transform(drug_df[col])
    label_encoders[col] = le

X = drug_df.drop(columns=["Drug"])
y = drug_df["Drug"]
y_le = LabelEncoder()
y = y_le.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

acc = accuracy_score(y_test, y_pred)
prec = precision_score(y_test, y_pred, average='weighted')
rec = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')
cm = confusion_matrix(y_test, y_pred)

print("Drug Dataset Classification:")
print(f"Accuracy Score: {acc:.4f}")
print(f"Precision Score: {prec:.4f}")
print(f"Recall Score: {rec:.4f}")
print(f"F1 Score: {f1:.4f}")

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=y_le.classes_,
            yticklabels=y_le.classes_)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix: drug.csv")

```

```
plt.show()
```

```
train_and_evaluate_drug()
```

```
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.tree import DecisionTreeRegressor, plot_tree
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

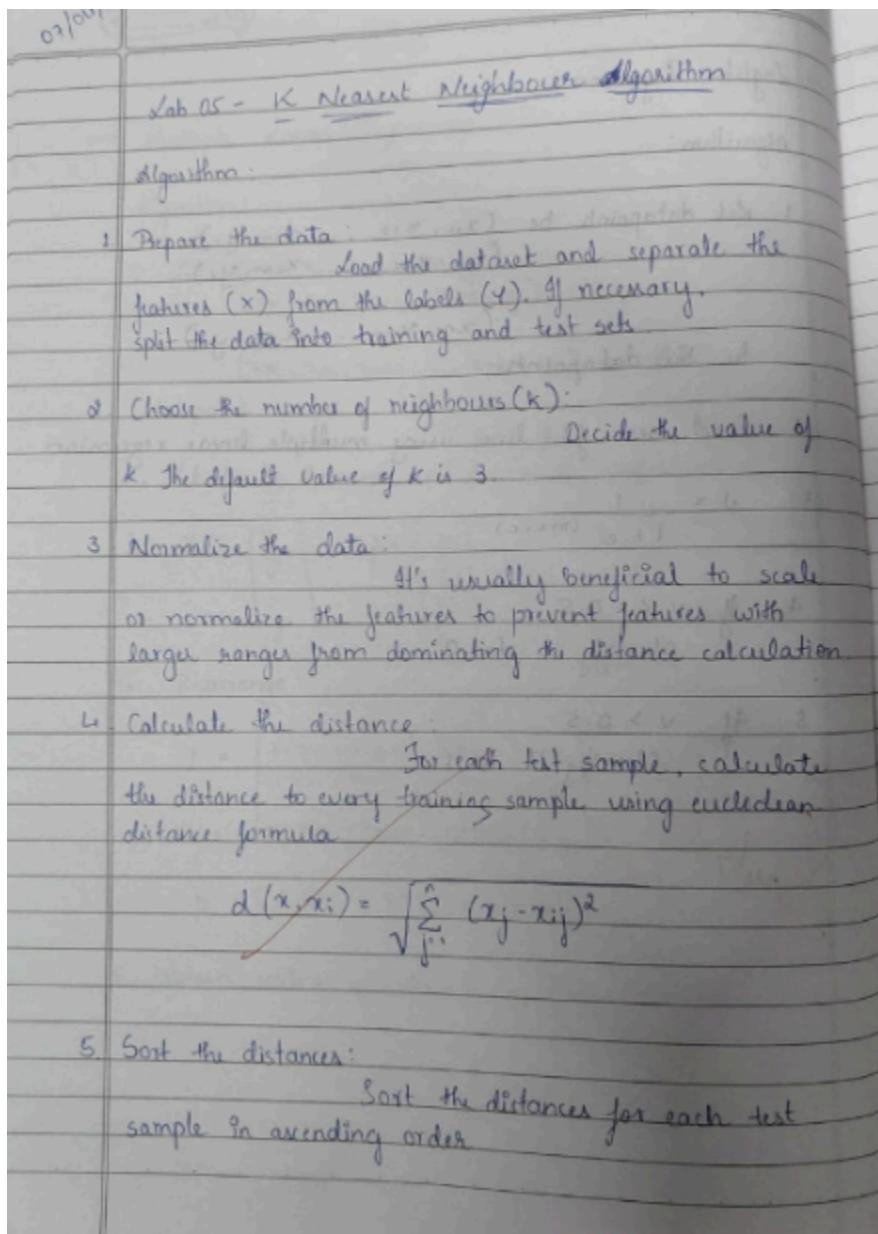
```
petrol_df = pd.read_csv("petrol_consumption.csv")
X = petrol_df.drop(columns=["Petrol_Consumption"])
y = petrol_df["Petrol_Consumption"]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
model = DecisionTreeRegressor(max_depth=5, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("Petrol Consumption Regression:")
print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error (MSE):", mean_squared_error(y_test, y_pred))
print("Root Mean Squared Error (RMSE):", np.sqrt(mean_squared_error(y_test, y_pred)))
```

Program 6

Build KNN Classification model for a given dataset.

Screenshot:



- N Date _____ P
6. Select the K nearest neighbours
from the sorted distances.
select the top k nearest neighbours \rightarrow
 7. Count the class labels of neighbours:
For each test sample,
count the class labels of k nearest neighbours
 8. Determine the predicted class:
The predicted class is typically determined by the majority vote
 9. Make the prediction:
Assign the predicted class label to the test sample
 10. Evaluate the model:
Evaluate the model using performance metrics like accuracy.

Code:

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

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

```

```

le = LabelEncoder()
iris_df['species'] = le.fit_transform(iris_df['species'])
X = iris_df.drop('species', axis=1)
y = iris_df['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

error_rates = []
accuracies = []
k_values = range(1, 10)
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred_k = knn.predict(X_test)
    error = 1 - accuracy_score(y_test, y_pred_k)
    error_rates.append(error)
    accuracies.append(accuracy_score(y_test, y_pred_k))

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(k_values, accuracies, marker='o', color='blue')
plt.title("Accuracy vs K")
plt.xlabel("K Value")
plt.ylabel("Accuracy")

plt.subplot(1, 2, 2)
plt.plot(k_values, error_rates, marker='o', color='red')
plt.title("Error Rate vs K")
plt.xlabel("K Value")
plt.ylabel("Error Rate")
plt.tight_layout()
plt.show()

best_k = k_values[accuracies.index(max(accuracies))]

```

```

print(f"Best K: {best_k} with Accuracy: {max(accuracies):.2f}")

knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

# Evaluation
print("\n==== Final Evaluation on IRIS Dataset ====")
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred, labels=[0, 1, 2], target_names=le.classes_))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title("Confusion Matrix - IRIS")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

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, stratify=y
)
accuracy_scores = []
k_range = range(1, 21)

```

```

for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred_k = knn.predict(X_test)
    acc = accuracy_score(y_test, y_pred_k)
    accuracy_scores.append(acc)

plt.figure(figsize=(8, 5))
plt.plot(k_range, accuracy_scores, marker='o', color='purple')
plt.title("Accuracy vs K (Diabetes Dataset)")
plt.xlabel("K Value")
plt.ylabel("Accuracy")
plt.xticks(k_range)
plt.grid()
plt.show()

best_k = k_range[accuracy_scores.index(max(accuracy_scores))]
print(f"Best K: {best_k} with Accuracy: {max(accuracy_scores):.2f}")

knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print("==== Final Evaluation (Diabetes Dataset) ====")
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Purples', xticklabels=['No Diabetes', 'Diabetes'],
            yticklabels=['No Diabetes', 'Diabetes'])
plt.title("Confusion Matrix - Diabetes")
plt.xlabel("Predicted")

```

```
plt.ylabel("Actual")
plt.show()

heart_df = pd.read_csv('heart.csv')
X = heart_df.drop('target', axis=1)
y = heart_df['target']
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, stratify=y
)
accuracy_scores = []
k_range = range(1, 21)
```

for k in k_range:

```
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred_k = knn.predict(X_test)
    acc = accuracy_score(y_test, y_pred_k)
    accuracy_scores.append(acc)
```

```
plt.figure(figsize=(8, 5))
plt.plot(k_range, accuracy_scores, marker='o', color='red')
plt.title("Accuracy vs K (Heart Dataset)")
plt.xlabel("K Value")
plt.ylabel("Accuracy")
plt.xticks(k_range)
plt.grid()
plt.show()
```

```
best_k = k_range[accuracy_scores.index(max(accuracy_scores))]
print(f'Best K: {best_k} with Accuracy: {max(accuracy_scores):.2f}')
```

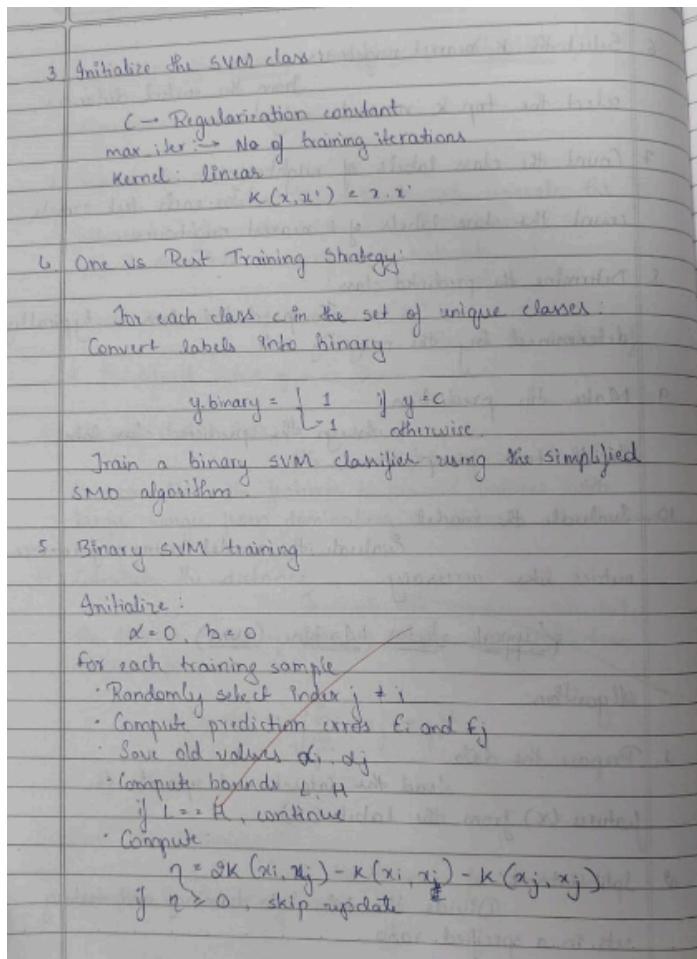
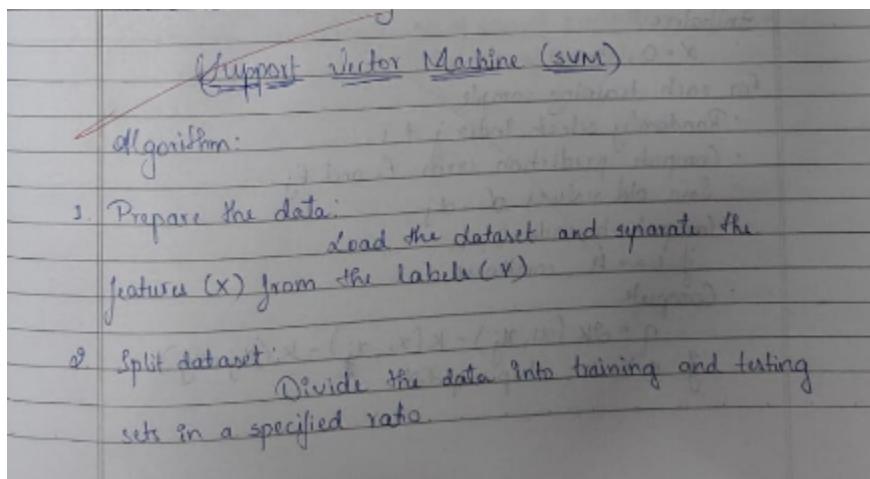
```
knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)

print("==== Final Evaluation (Heart Dataset) ====")
print("\nAccuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['No Disease', 'Disease']))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', xticklabels=['No Disease', 'Disease'],
            yticklabels=['No Disease', 'Disease'])
plt.title("Confusion Matrix - Heart Disease")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Program 7

Build Support vector machine model for a given dataset.

Screenshot:



(N) Date _____ (P) Page _____

- Update α_j
- $$\alpha_j = \alpha_j + \frac{y_j}{\eta} (E_i - E_j)$$
- Clip α_j
- Update α_i
- Compute b , the bias term
- Store model for each class

c. Prediction Phase

Compute decision score:

$$f(x) = \sum_j x_j y_j K(x_i, x_j) + b$$

Store the score.

Predict the class with maximum decision score.

7. Evaluation:

Calculate accuracy = $\frac{\text{No of correct predictions}}{\text{Total no of samples}} \times 100$

~~8th~~

Code:

```

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, roc_curve
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt
import seaborn as sns

```

```
iris = pd.read_csv("iris.csv")
label_encoder = LabelEncoder()
iris['species'] = label_encoder.fit_transform(iris['species'])
class_names_iris = label_encoder.classes_
X_iris = iris.drop('species', axis=1)
y_iris = iris['species']
X_train_iris, X_test_iris, y_train_iris, y_test_iris = train_test_split(X_iris, y_iris, test_size=0.2,
random_state=42)
```

```
scaler = StandardScaler()
X_train_iris = scaler.fit_transform(X_train_iris)
X_test_iris = scaler.transform(X_test_iris)
svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train_iris, y_train_iris)
y_pred_linear = svm_linear.predict(X_test_iris)
acc_linear = accuracy_score(y_test_iris, y_pred_linear)
cm_linear = confusion_matrix(y_test_iris, y_pred_linear)
```

```
plt.figure(figsize=(6,4))
sns.heatmap(cm_linear, annot=True, fmt='d', cmap='Blues', xticklabels=class_names_iris,
yticklabels=class_names_iris)
plt.title(f'IRIS SVM Linear Kernel\nAccuracy: {acc_linear:.2f}')
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.tight_layout()
plt.show()
```

```
svm_rbf = SVC(kernel='rbf')
svm_rbf.fit(X_train_iris, y_train_iris)
y_pred_rbf = svm_rbf.predict(X_test_iris)
acc_rbf = accuracy_score(y_test_iris, y_pred_rbf)
cm_rbf = confusion_matrix(y_test_iris, y_pred_rbf)
```

```
plt.figure(figsize=(6,4))
sns.heatmap(cm_rbf, annot=True, fmt='d', cmap='Greens', xticklabels=class_names_iris,
yticklabels=class_names_iris)
plt.title(f'IRIS SVM RBF Kernel\nAccuracy: {acc_rbf:.2f}')
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.tight_layout()
plt.show()
```

```
letters = pd.read_csv("letter-recognition.csv")
X_letters = letters.drop('letter', axis=1)
y_letters = letters['letter']
label_encoder_letters = LabelEncoder()
y_letters_encoded = label_encoder_letters.fit_transform(y_letters)
class_names_letters = label_encoder_letters.classes_
X_train_letters, X_test_letters, y_train_letters, y_test_letters = train_test_split(
    X_letters, y_letters_encoded, test_size=0.2, random_state=42)
```

```
scaler_letters = StandardScaler()
X_train_letters = scaler_letters.fit_transform(X_train_letters)
X_test_letters = scaler_letters.transform(X_test_letters)
svm_letters = SVC(kernel='rbf', probability=True)
svm_letters.fit(X_train_letters, y_train_letters)
```

```
y_pred_letters = svm_letters.predict(X_test_letters)
acc_letters = accuracy_score(y_test_letters, y_pred_letters)
cm_letters = confusion_matrix(y_test_letters, y_pred_letters)
```

```
plt.figure(figsize=(14, 12))
sns.heatmap(cm_letters, annot=True, fmt='d', cmap='Purples',
```

```

xticklabels=class_names_letters,
yticklabels=class_names_letters,
annot_kws={"size": 8},
cbar=True)

plt.title(f'Letter Recognition - SVM RBF Kernel\nAccuracy: {acc_letters*100:.2f}%', fontsize=16)
plt.xlabel("Predicted Label", fontsize=14)
plt.ylabel("True Label", fontsize=14)
plt.xticks(rotation=45)
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()

```

```

y_test_binarized = label_binarize(y_test_letters, classes=np.arange(len(class_names_letters)))
y_score = svm_letters.predict_proba(X_test_letters)
auc_score = roc_auc_score(y_test_binarized, y_score, average='macro')
fpr = dict()
tpr = dict()
for i in range(len(class_names_letters)):
    fpr[i], tpr[i], _ = roc_curve(y_test_binarized[:, i], y_score[:, i])
plt.figure(figsize=(8, 6))
for i in range(0, len(class_names_letters), 4): # Plot every 4th class
    plt.plot(fpr[i], tpr[i], lw=1.5, label=f'Class {class_names_letters[i]}')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title(f'Multi-Class ROC Curve (Macro AUC = {auc_score:.6f})')
plt.legend(loc="lower right", fontsize='small')
plt.grid()
plt.tight_layout()
plt.show()

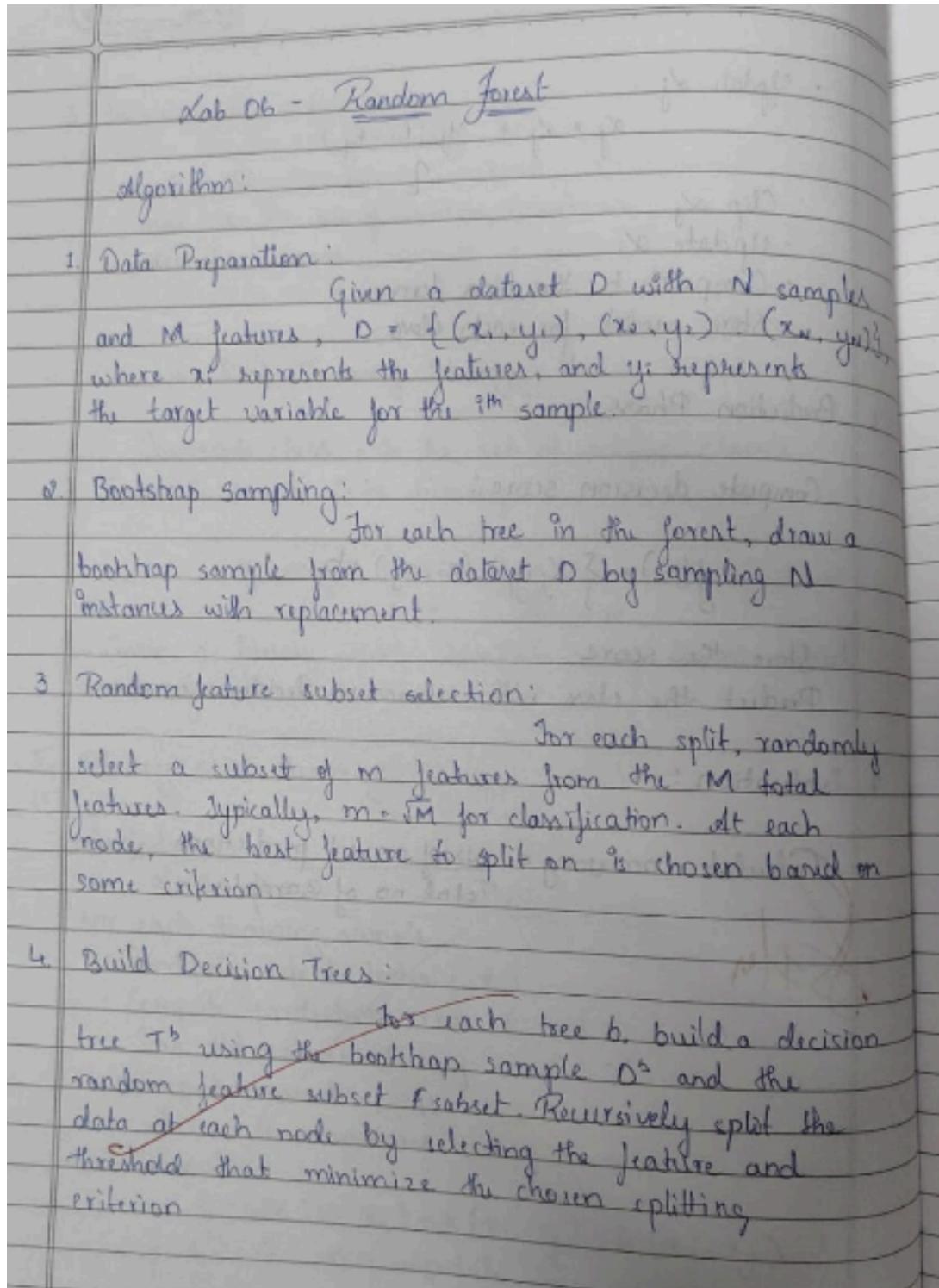
print(f'Exact AUC Score = {auc_score}')

```

Program 8

Implement Random forest ensemble method on a given dataset.

Screenshot:



- (N) Page _____ (P)
5. Calculate Impurity:
For classification, compute Gini
Impurity for each possible split of the dataset
 6. Calculate Mean Squared Error:
For regression, calculate mean squared error for each node
 7. Recursive Splitting:
Recursively splitting the data at each node of the tree based on the feature that provides the best split according to the impurity or MSE
 8. Repeat for Multiple Trees:
Repeat steps 4 to 7 B times
to construct Random Forest.
 9. Make Predictions
Classification: Majority vote
Regression: Average
 - ~~10. Evaluate the model:~~
Classification: Accuracy
~~Regression: MSE, MAE~~

Code:

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix

```

```
iris_df = pd.read_csv("iris.csv")
X = iris_df.drop('species', axis=1)
y = iris_df['species']
le = LabelEncoder()
y_encoded = le.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.3, random_state=42)
rf_model = RandomForestClassifier(n_estimators=10, random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
print("Random Forest Accuracy with 10 trees:", accuracy_score(y_test, y_pred))

scores = []
n_range = range(1, 101)
best_model = None
best_preds = None

for n in n_range:
    model = RandomForestClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    acc = accuracy_score(y_test, preds)
    scores.append(acc)
    if acc == max(scores):
        best_model = model
        best_preds = preds

best_score = max(scores)
best_n = n_range[scores.index(best_score)]
print(f'Best Random Forest Accuracy: {best_score:.4f} with {best_n} trees')
```

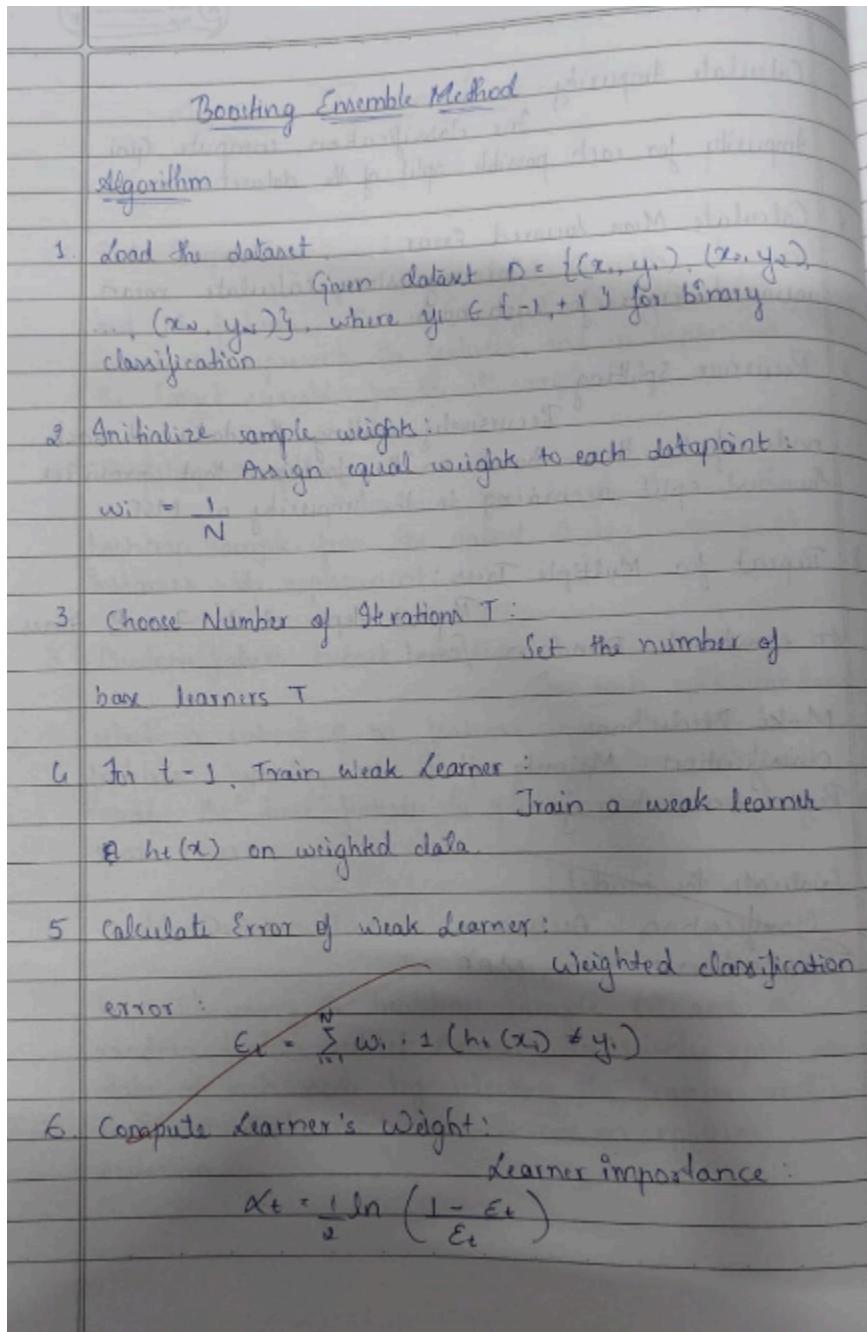
```
plt.figure(figsize=(10, 5))
plt.plot(n_range, scores, marker='o', linestyle='-', color='blue')
plt.title('Random Forest Accuracy vs Number of Trees (Iris Dataset)')
plt.xlabel('Number of Trees')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()

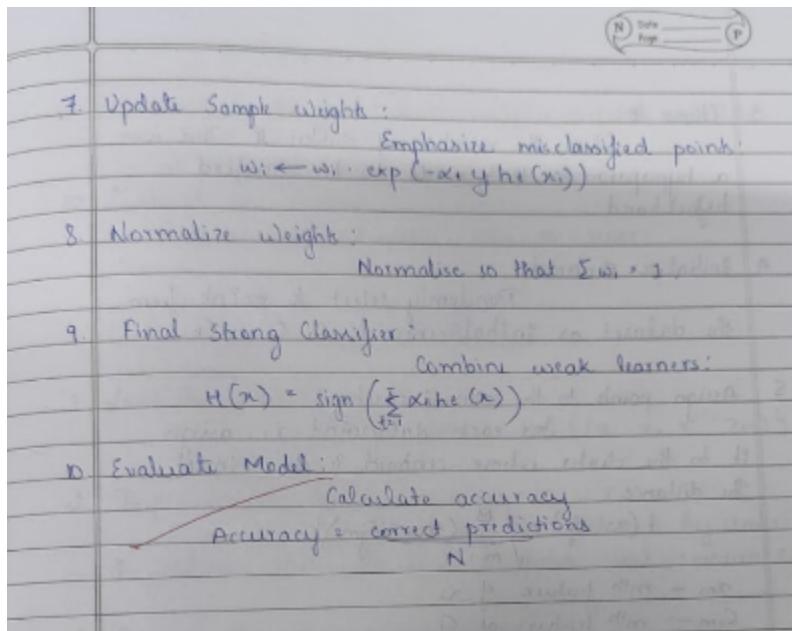
cm = confusion_matrix(y_test, best_preds)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_, yticklabels=le.classes_)
plt.title(f"Confusion Matrix for Best Random Forest Model ({best_n} Trees)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot:





Code:

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix

income_df = pd.read_csv("income.csv")
X_income = income_df.drop('income_level', axis=1)
y_income = income_df['income_level']
X_train_i, X_test_i, y_train_i, y_test_i = train_test_split(X_income, y_income, test_size=0.3,
random_state=42)

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

```

```

ada_model.fit(X_train_i, y_train_i)
y_pred_i = ada_model.predict(X_test_i)
print("AdaBoost Accuracy with 10 estimators:", accuracy_score(y_test_i, y_pred_i))

scores_ada = []
n_range_ada = range(1, 51)
best_model_ada = None
best_preds_ada = None

for n in n_range_ada:
    model = AdaBoostClassifier(n_estimators=n, random_state=42)
    model.fit(X_train_i, y_train_i)
    preds = model.predict(X_test_i)
    acc = accuracy_score(y_test_i, preds)
    scores_ada.append(acc)
    if acc == max(scores_ada):
        best_model_ada = model
        best_preds_ada = preds

best_score_ada = max(scores_ada)
best_n_ada = n_range_ada[scores_ada.index(best_score_ada)]
print(f'Best AdaBoost Accuracy: {best_score_ada:.4f} with {best_n_ada} estimators')

plt.figure(figsize=(10, 5))
plt.plot(n_range_ada, scores_ada, marker='o', linestyle='-', color='orange')
plt.title('AdaBoost Accuracy vs Number of Estimators (Income Dataset)')
plt.xlabel('Number of Estimators')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()

cm_ada = confusion_matrix(y_test_i, best_preds_ada)

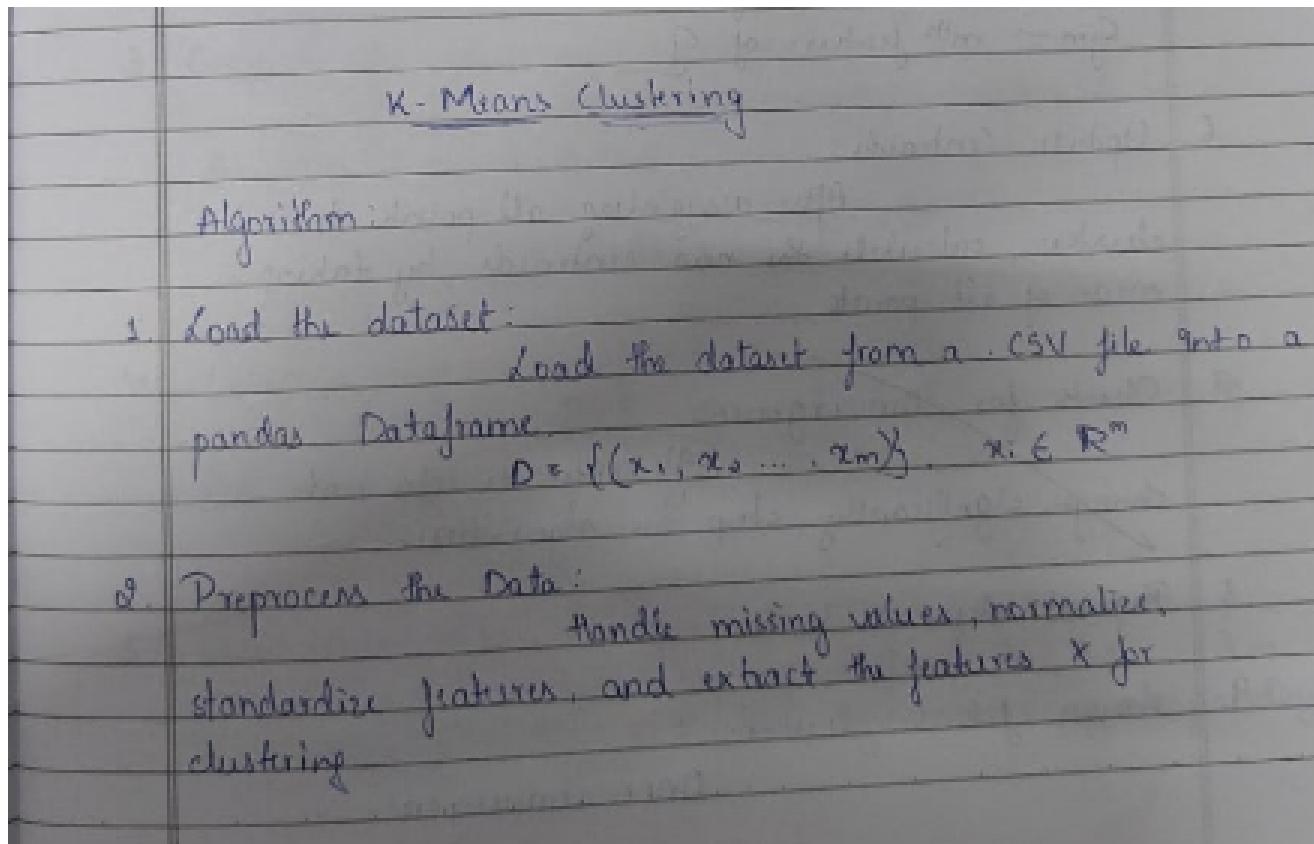
```

```
plt.figure(figsize=(6, 5))
sns.heatmap(cm_ada, annot=True, fmt='d', cmap='Oranges', xticklabels=[0, 1], yticklabels=[0, 1])
plt.title(f"Confusion Matrix for Best AdaBoost Model ({best_n_ada} Estimators)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Program 10

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

Screenshot:



3. Choose k :
 Select the number of clusters k . This is a hyperparameter that needs to be specified beforehand.
4. Initialize centroids:
 Randomly select k points from the dataset as initial centroids c_1, c_2, \dots, c_k .
5. Assign points to the nearest cluster:
 For each datapoint x_i , assign it to the cluster whose centroid is the closest.
 The distance:

$$d(x_i, c_j) = \sqrt{\sum_{m=1}^M (x_{im} - c_{jm})^2}$$
- $x_{im} \rightarrow m^{\text{th}}$ feature of x_i
 $c_{jm} \rightarrow m^{\text{th}}$ feature of c_j
6. Update centroids:
 After assigning all points to clusters, calculate the new centroids by taking mean of all points.
7. Check for convergence:
 If centroids do not change significantly, stop the algorithm.
8. Repeat steps 5 to 7.
9. Assign final clusters:
 Once convergence is reached, the final cluster assignments and centroids are obtained.

10. Evaluate the clusters:
 Sum of squared errors, silhouette score.

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from scipy import stats
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

df1=pd.read_csv("iris.csv")
df1.head()
df = df1.drop(['sepal_length','sepal_width','species'],axis=1)
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df)
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(scaled_df)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

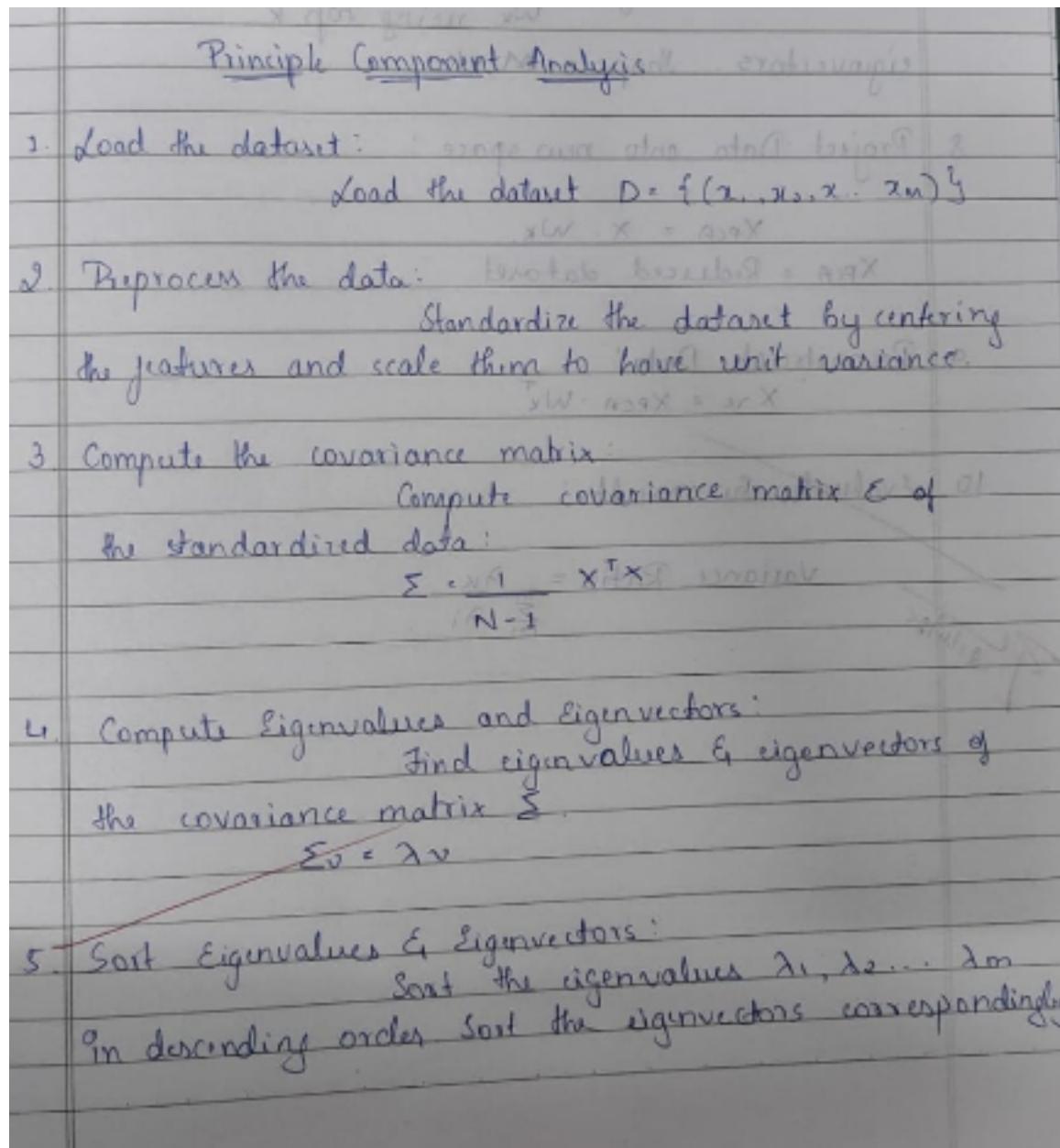
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10, random_state=0)
pred_y = kmeans.fit_predict(scaled_df)
df['cluster'] = pred_y
```

```
plt.scatter(df['petal_length'], df['petal_width'], c=df['cluster'])  
plt.title('Clusters of Iris Flowers')  
plt.xlabel('Petal Length')  
plt.ylabel('Petal Width')  
plt.show()
```

Program 11

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

Screenshot:



6. Select Top k Eigenvectors:

Select the top k eigenvectors to reduce the dimensionality to k dimensions.

7. Construct the Projection Matrix:

W_k using top k eigenvectors. Shape = $M \times k$.

8. Project Data onto new space:

$$X_{PCA} = X \cdot W_k$$

X_{PCA} = Reduced dataset.

9. Reconstruct Data:

$$X_{re} = X_{PCA} \cdot W_k^T$$

10. Evaluate the results:

$$\text{Variance Ratio} = \frac{\lambda_k}{\sum_{i=1}^n \lambda_i}$$

Code:

```
from google.colab import files  
heart=files.upload()  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model_selection import train_test_split
```

```
from scipy import stats
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.decomposition import PCA

df1=pd.read_csv("heart.csv")
df1.head()
text_cols = df1.select_dtypes(include=['object']).columns
label_encoder = LabelEncoder()
for col in text_cols:
    df1[col] = label_encoder.fit_transform(df1[col])
print(df1.head())
X = df1.drop('HeartDisease', axis=1)
y = df1['HeartDisease']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Support Vector Machine
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)
svm_predictions = svm_model.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predictions)
print(f'SVM Accuracy: {svm_accuracy}')
```

```
# Logistic Regression
lr_model = LogisticRegression(random_state=42)
lr_model.fit(X_train, y_train)
lr_predictions = lr_model.predict(X_test)
lr_accuracy = accuracy_score(y_test, lr_predictions)
print(f"Logistic Regression Accuracy: {lr_accuracy}")

# Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
rf_accuracy = accuracy_score(y_test, rf_predictions)
print(f"Random Forest Accuracy: {rf_accuracy}")

models = {
    "SVM": svm_accuracy,
    "Logistic Regression": lr_accuracy,
    "Random Forest": rf_accuracy}
best_model = max(models, key=models.get)
print(f"\nBest Model: {best_model} with accuracy {models[best_model]}")

pca = PCA(n_components=0.95)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
svm_model_pca = SVC(kernel='linear', random_state=42)
svm_model_pca.fit(X_train_pca, y_train)
svm_predictions_pca = svm_model_pca.predict(X_test_pca)
svm_accuracy_pca = accuracy_score(y_test, svm_predictions_pca)
print(f"SVM Accuracy (with PCA): {svm_accuracy_pca}")

lr_model_pca = LogisticRegression(random_state=42)
```

```
lr_model_pca.fit(X_train_pca, y_train)
lr_predictions_pca = lr_model_pca.predict(X_test_pca)
lr_accuracy_pca = accuracy_score(y_test, lr_predictions_pca)
print(f"Logistic Regression Accuracy (with PCA): {lr_accuracy_pca}")

rf_model_pca = RandomForestClassifier(random_state=42)
rf_model_pca.fit(X_train_pca, y_train)
rf_predictions_pca = rf_model_pca.predict(X_test_pca)
rf_accuracy_pca = accuracy_score(y_test, rf_predictions_pca)
print(f"Random Forest Accuracy (with PCA): {rf_accuracy_pca}")

models_pca = {
    "SVM": svm_accuracy_pca,
    "Logistic Regression": lr_accuracy_pca,
    "Random Forest": rf_accuracy_pca}
best_model_pca = max(models_pca, key=models_pca.get)
print(f"\nBest Model (with PCA): {best_model_pca} with accuracy {models_pca[best_model_pca]}")
```