# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**"JnanaSangama", Belgaum -590014, Karnataka.**

**LAB REPORT**
**on**

# Machine Learning (23CS6PCMAL)

***Submitted by***

**S M MRUNALINI (1BM22CS228)**

***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**

**<u>CERTIFICATE</u>**

This is to certify that the Lab work entitled “Machine Learning (23CS6PCMAL)” carried out by **S M Mrunalini(1BM22CS228),** 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.

| | |
|---|---|
| Saritha A N | Dr. Kavitha Sooda |
| Assistant Professor | Professor & HOD |
| Department of CSE , BMSCE | Department of CSE , BMSCE |

# Index


| Sl. No. | Date | Experiment Title | Page No. |
|---|---|---|---|
| 1 | 3-3-2025 | Write a python program to import and export data using Pandas library functions | 4 |
| 2 | 9-3-2025 | Demonstrate various data pre-processing techniques for a given dataset | 8 |
| 3 | 17-3-2025 | Implement Linear and Multi-Linear Regression algorithm using appropriate dataset | 10 |
| 4 | 17-3-2025 | Build Logistic Regression Model for a given dataset | 13 |
| 5 | 24-3-2025 | Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample. | 16 |
| 6 | 7-4-2025 | Build KNN Classification model for a given dataset. | 19 |
| 7 | 7-4-2025 | Build Support vector machine model for a given dataset | 22 |
| 8 | 21-4-2025 | Implement Random forest ensemble method on a given dataset. | 24 |
| 9 | 21-4-2025 | Implement Boosting ensemble method on a given dataset. | 25 |
| 10 | 21-4-2025 | Build k-Means algorithm to cluster a set of data stored in a .CSV file. | 27 |
| 11 | 21-4-2025 | Implement Dimensionality reduction using Principal Component Analysis (PCA) method. | 28 |

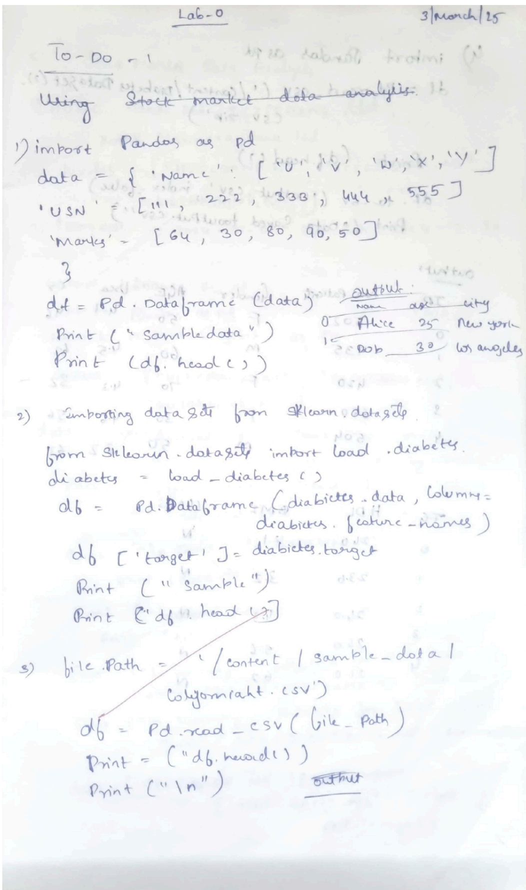
**Github Link:**

https://github.com/Mrunalinishettar/ml-lab

## Program 1

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

**Screenshots:**



Lab-0 3/March/25

To-Do - 1

Using ~~Stock market data analysis.~~

1) import Pandas as pd
data = { 'Name' : [ 'U', 'V', 'W', 'X', 'Y' ]
'USN' : [ 111, 222, 333, 444, 555 ]
'Marks' : [ 64, 30, 80, 90, 50 ]
}
df = Pd . Dataframe (data)
Print ( " sample data " )
Print ( df. head ( ) )

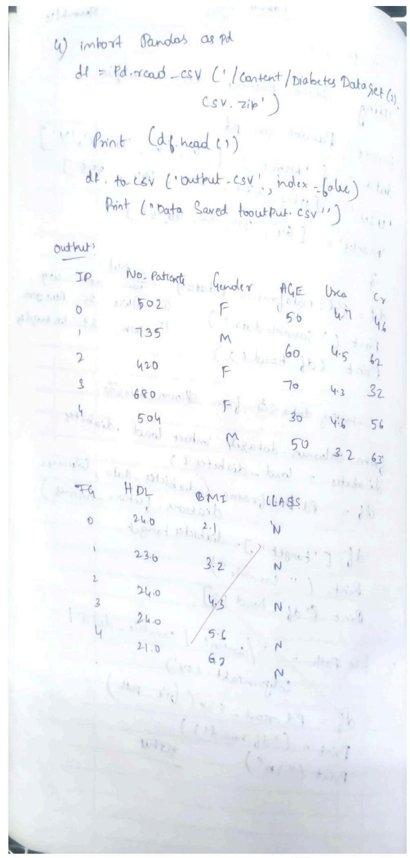
output
| | name | age | city |
|---|---|---|---|
| 0 | Alice | 25 | New york |
| 1 | Bob | 30 | Los angeles |

2) importing data set from sklearn datasets
from sklearn . datasets import load . diabetes
diabetes = load _ diabetes ( )
df = Pd. Dataframe ( diabetes . data , columns = diabetes . feature _ names )
df [ 'target' ] = diabetes . target
Print ( " sample " )
Print ( df . head ( ) )

3) file . Path = ' / content / sample _ data / california . csv ' )
df = Pd . read _ csv ( file _ Path )
Print = ( " df . head ( ) )
Print ( " \n " )

output





4) import Pandas as pd
df = Pd . read _ csv ( ' / content / Diabetes Dataset (1) . csv . zip ' )
Print ( df. head ( ) )
df . to csv ( ' output . csv ', index = false )
Print ( " Data Saved to output . csv " )

output

| ID | No. Patients | Gender | AGE | Urea | Cr |
|---|---|---|---|---|---|
| 0 | 502 | F | 50 | 4.7 | 46 |
| 1 | 735 | M | | | 46 |
| 2 | 420 | F | 60 | 4.5 | 62 |
| 3 | 680 | | 70 | 4.3 | 32 |
| 4 | 504 | F | 30 | 4.6 | 56 |
| | | M | 50 | 3.2 | 63 |

| FG | HDL | BMI | CLASS |
|---|---|---|---|
| 0 | 24.0 | 2.1 | N |
| 1 | 23.0 | 3.2 | N |
| 2 | 24.0 | | N |
| 3 | 24.0 | 4.3 | N |
| 4 | 21.0 | 5.6 | N |
| | | 6.2 | N |

5) Stock Market Data Analysis

1) HDFC Bank Ltd." ICICI BANK. Ltd,
Kotak Mahindra Bank Ltd.

tickers = ["HDFCBANK.NS", "ICICI BANK.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 dataset:")
    Print (data.head ())

Print ("\n shape of dataset: ")
    Print (data.shape)

Print ("\n Column names:")
    Print (data.columns)

Print ("\n summary statistics for HDFC Bank:")
    Print (data ["HDFCBANK.NS' ].describe())
Print ("\n summary statistics for ICICI Bank:")
    Print (data ['ICICIBANK.NS' ].describe())

data ['HDFCBANK.NS' ] ['Daily return'] = data
        ['HDFCBANK.NS' ] ['close'].
                pct_change ()
```

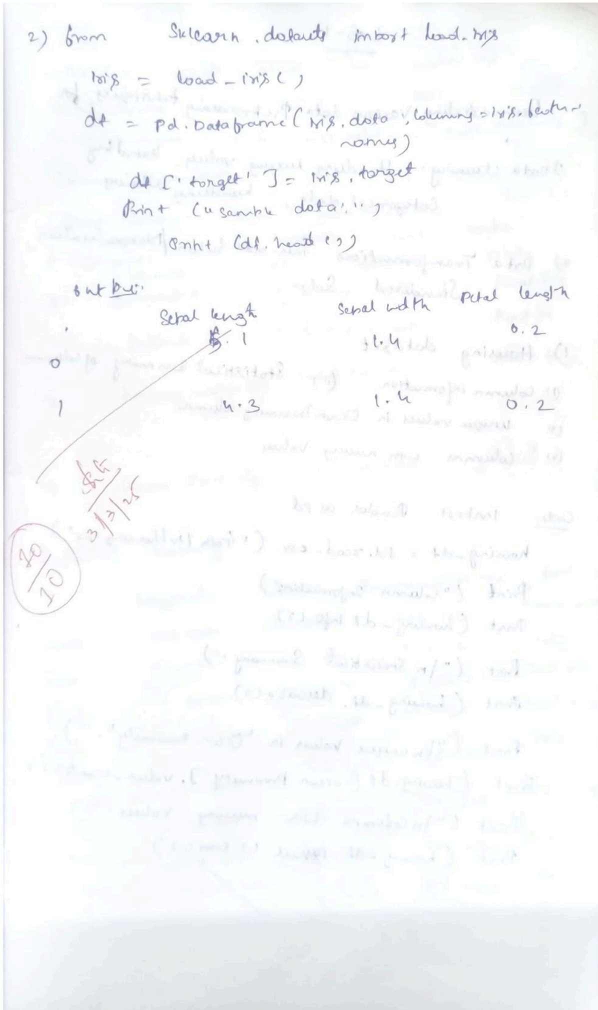
```
Plt. figure (figsize = (14, 10)

Plt. subplot (3, 2, 1 )
data ['HDFCBANK.NS']

Plt. subplot (3, 2, 1 )
data ['ICICIBANK.NS' ]['close']. Plot
        (title = "HDFCBANK - Closing P

data ["HDFCBANK.NS' ].to_csv (hdfc_bank_data

data ['ICICIBANK.NS' ].to_csv (icici_bank_data
                                csv)

Print ("\n Bank Stock data saved to
        csv file)
```

Output

```
(1) import Pandas as Pd

data = {
name : [Alice, bob, Charlie, David]
age : [25, 30, 40, 12]
city : [ New york, los angels, Chicago, Houston]
}
    df = Pd.Dataframe (data)
        Print (Sample data:")
            Print (df.head ())
```

Output :

| | name | age | city |
|---|---|---|---|
| 0 | Alice | 25 | New york |
| 1 | bob | 30 | Chicag |



2) 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 ( ))

Output:

| | Sepal length | Sepal width | Petal length |
|---|---|---|---|
| 0 | 5 . 1 | 1 . 4 | 0 . 2 |
| 1 | 4 . 3 | 1 . 4 | 0 . 2 |

3/3/25

10/10



**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())
file_path = 'data.csv'
df = pd.read_csv(file_path)
print("Sample data:")
print(df.head())
print("\n")
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())
from sklearn.datasets import load_diabetes
iris = load_diabetes()
df = pd.DataFrame(iris.data, columns=iris.feature_names)

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")
```

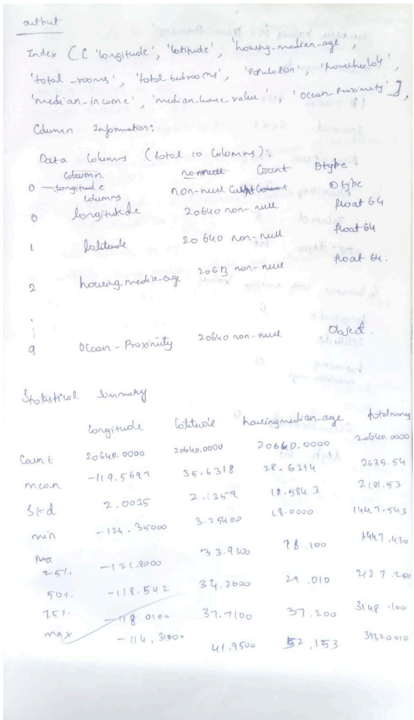
```
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="Reliance Industries - Daily Returns", color='orange')
plt.tight_layout()
plt.show()
reliance_data.to_csv('reliance_stock_data.csv')

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'.")

tickers = ["HDFCBANK.NS", "ICICIBANK.NS", "KOTAKBANK.NS"]
data = yf.download(tickers, start="2024-01-01", end="2024-12-30",
group_by='ticker')
print("First 5 rows of the dataset:")
print(data.head())
print("\nShape of the dataset:")
print(data.shape)
print("\nColumn names:")
print(data.columns)
print("\n")
reliance_data = data['ICICIBANK.NS']
print("\nSummary statistics for ICICI 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="ICICI Industries - Closing Price")
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="ICICI Industries - Daily Returns", color='BLACK')
plt.tight_layout()
```

```
plt.show()
reliance_data.to_csv('icici_stock_data.csv')
print("\nicici stock data saved to 'icici_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['KOTAKBANK.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="KOTAK Industries - Closing Price")
plt.subplot(2, 1, 2)
reliance_data['Daily Return'].plot(title="kotak Industries - Daily Returns", color='red')
plt.tight_layout()
plt.show()
reliance_data.to_csv('kotak_stock_data.csv')
print("\nkotak stock data saved to 'kotak_stock_data.csv'.")
```

# Program 2

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

## Screenshot:



output

Index ([ 'longitude', 'latitude', 'housing-median-age', 'total_rooms', 'total bedrooms', 'Population', 'households', 'median-income', 'median-house-value', 'ocean-Proximity' ],

Column Information:

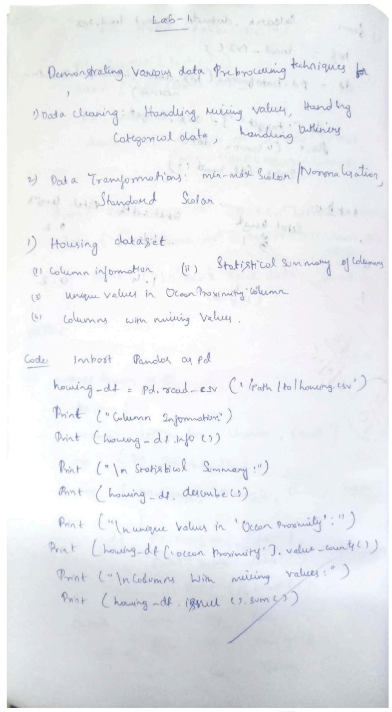
Data Columns (total 10 Columns):

| | Column | non-null Count | Dtype |
|---|---|---|---|
| 0 | longitude | 20640 non-null | float 64 |
| 1 | latitude | 20640 non-null | float 64 |
| 2 | housing median-age | 20613 non-null | float 64. |
| ⋮ | | | |
| 9 | Ocean - Proximity | 20640 non-null | Object. |

Statistical Summary

| | longitude | latitude | housingmedian_age | totalrooms |
|---|---|---|---|---|
| Count | 20640.0000 | 20640.0000 | 20640.0000 | 20640.0000 |
| mean | -119.5697 | 35.6318 | 28.6394 | 2635.54 |
| Std | 2.0035 | 2.1359 | 12.5843 | 2181.53 |
| min | -124.35000 | 3.25400 | 18.0000 | 1447.543 |
| 25% | -121.8000 | 33.9300 | 28.100 | 1447.430 |
| 50% | -118.542 | 34.2600 | 29.010 | 2127.250 |
| 75% | -118.0100 | 37.7100 | 37.200 | 3148.100 |
| max | -114.3100 | 41.9500 | 52.153 | 39320.010 |





Lab-

Demonstrating various data Preprocessing techniques for

1) Data cleaning: Handling missing values, Handling Categorical data, handling outliers

2) Data Transformations: min-max Scalar / Normalisation, Standard Scalar.

1) Housing dataset

(i) Column information (ii) Statistical Summary of columns

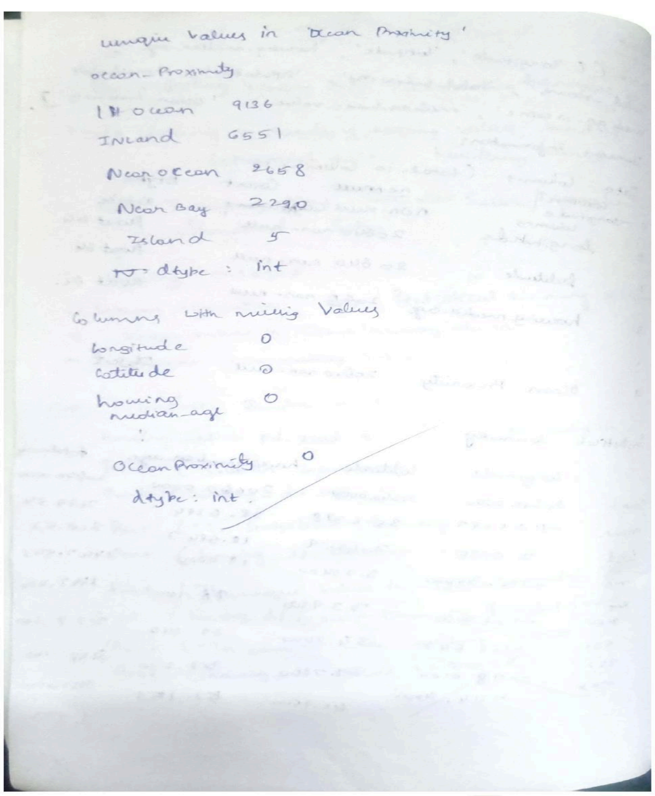
(3) unique values in Ocean Proximity column

(4) Columns with missing Values.

Code:

```
import Pandas as pd
housing_df = pd.read_csv ('/path/to/housing.csv')
Print ("Column Information:")
Print (housing_df.info ())
Print ("\n Statistical Summary:")
Print (housing_df. describe())
Print ("\n unique Values in 'Ocean Proximity':")
Print (housing_df['ocean Proximity']. value_counts())
Print ("\n Columns With missing values:")
Print (housing_df . isnull (). sum())
```





unique values in 'Ocean Proximity'

ocean_Proximity

| | |
|---|---|
| 1H ocean | 9136 |
| INLand | 6551 |
| Near ocean | 2658 |
| Near Bay | 2290 |
| Island | 5 |

dtype : int

Columns with missing Values

| | |
|---|---|
| longitude | 0 |
| latitude | 0 |
| housing median_age | 0 |
| ⋮ | |
| Ocean Proximity | 0 |

dtype : int

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

- Adult Income dataset (adult.csv)
  No columns had missing values
- Diabetes dataset (Dataset of diabetes.csv)
  No columns has missing values
  - Handling method: since no missing value were found, no imputation was performed

(2) Which categorical column did you identify in dataset? How did you encode them?

adult Income Dataset (adult.csv)

→ Categorical columns
  → workclass
  → education
  → occupation
  → race
  → gender
  → income
  → relationship



2) Python code to implement the following data preprocessing techniques for Diabetes & adult Income datasets.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv ("/content/dataset of diabetes.csv")
   df.head(10)
df = pd.read_csv ("/content/adult data1.csv")
   df.head(10)

import numpy as np
# introduce some missing values for demonstration.
df.loc[5, 'AGE'] = np.nan
df.loc[9, 'BMI'] = np.nan
df.head(10)
```

Output

| | ID | No_Pation | Gender | Age | Urea | Cr | BMI | Class |
|---|---|---|---|---|---|---|---|---|
| 0 | 502 | 17975 | F | 50.0 | 4.7 | 46 | NaN | N |
| 1 | 412 | 47975 | M | 43.0 | 4.5 | 62 | 0.17913 | N |
| 2 | 327 | 87656 | F | 32.0 | 4.2 | 72 | 0.17913 | N |
| 3 | 634 | 67643 | M | NaN | 4.7 | 2.3 | 0.69 | N |

Standardization is used when
- The dataset has ~~anything~~ varying units
a Gaussian distribution (normal)
- There are significant outliers,
as standardization is less
sensitive to them.

10.03.2025



**Code:**

```
from google.colab import files
diabetes=files.upload()

from google.colab import files
adult_income=files.upload()

df1=pd.read_csv("Dataset of Diabetes .csv")
df1.head()

df2=pd.read_csv("adult.csv")
df2.head()

df1.info()
df2.info()
```

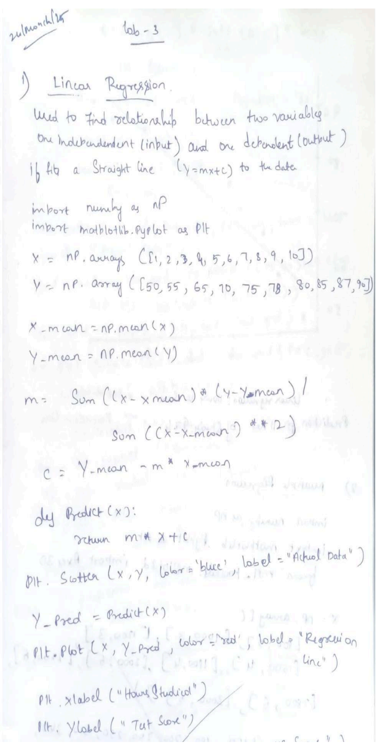
```
df1.describe()
df2.describe()

missing_values1 = df1.isnull().sum()
print(missing_values1)
missing_values2 = df2.isnull().sum()
print(missing_values2)

df1['Gender'] = df1['Gender'].replace('f', 'F')
ordinal_encoder = OrdinalEncoder(categories=[["F", "M"]])
df1["Gender_Encoded"] = ordinal_encoder.fit_transform(df1[["Gender"]])
onehot_encoder = OneHotEncoder()
encoded_data = onehot_encoder.fit_transform(df1[["CLASS"]])
encoded_array = encoded_data.toarray()
encoded_df = pd.DataFrame(encoded_array, columns=onehot_encoder.get_feature_names_out(["CLASS"]))
df_encoded = pd.concat([df1, encoded_df], axis=1)
df1 = pd.concat([df1, encoded_df], axis=1)
df1.drop("CLASS", axis=1, inplace=True)
df1.drop("Gender", axis=1, inplace=True)
print(df2.head())
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
df_copy2 = df2
ordinal_encoder = OrdinalEncoder(categories=[["Male","Female"]])
df_copy2["Gender_Encoded"] = ordinal_encoder.fit_transform(df_copy2[["gender"]])
print(df_copy2[["gender","Gender_Encoded"]])

onehot_encoder = OneHotEncoder()
encoded_data =
onehot_encoder.fit_transform(df2[["occupation","workclass","education","marital-status","relationship","race","
n ative-country","income"]])
encoded_array = encoded_data.toarray()
encoded_df = pd.DataFrame(encoded_array,
columns=onehot_encoder.get_feature_names_out(["occupation","workclass","education","marital-status","relatio
nship","race","native-country","income"]))
df_encoded = pd.concat([df_copy2, encoded_df], axis=1)

df_encoded.drop("gender", axis=1, inplace=True)
df_encoded.drop("occupation", axis=1, inplace=True)
df_encoded.drop("workclass", axis=1, inplace=True)
df_encoded.drop("education", axis=1, inplace=True)
df_encoded.drop("marital-status", axis=1, inplace=True)
df_encoded.drop("relationship", axis=1, inplace=True)
df_encoded.drop("race", axis=1, inplace=True)
df_encoded.drop("native-country", axis=1, inplace=True)
df_encoded.drop("income", axis=1, inplace=True)
print(df_encoded. head())

normalizer = MinMaxScaler()
df_encoded[["fnlwgt","educational-num","capital-gain","capital-loss","hours-per-week"]] =
normalizer.fit_transform(df_encoded[["fnlwgt","educational-num","capital-gain","capital-loss","hours-per-week"]
])
df_encoded.head()
```

```
normalizer = MinMaxScaler()
df1[["No_Pation","AGE","Urea","Cr" , "HbA1c" , "Chol","TG","HDL","LDL","VLDL","BMI"]] =
normalizer.fit_transform(df1[["No_Pation","AGE","Urea","Cr" , "HbA1c" ,
"Chol","TG","HDL","LDL","VLDL","BMI"]])
df1.head()
```

## Program 3

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

**Screenshot:**



24/march/25

lab - 3

1) Linear Regression

Used to find relationship between two variables
one independent (input) and one dependent (output)
it fits a straight line (y=mx+c) to the data

import numpy as np
import matplotlib.pyplot as plt

x = np.array ([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
y = np.array ([50, 55, 65, 70, 75, 78, 80, 85, 87, 90])

x_mean = np.mean(x)
y_mean = np.mean(y)

m = sum ((x - x_mean) * (y - y_mean)) /
sum ((x - x_mean) ** 2)

c = y_mean - m * x_mean

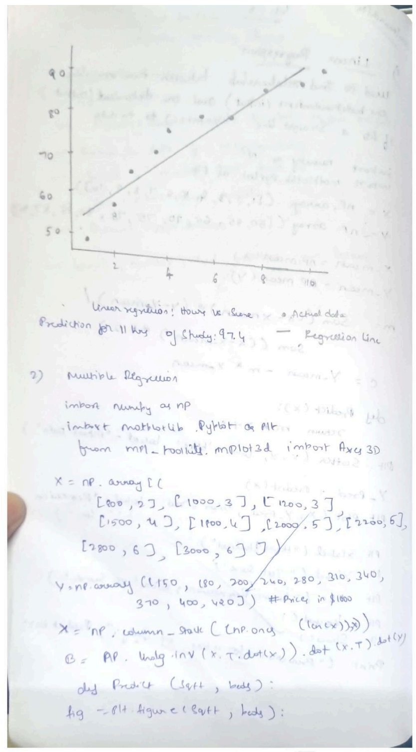
def predict (x):
return m * x + c

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

y_pred = predict (x)

plt.plot (x, y_pred, color = 'red', label = "Regression line")

plt.xlabel ("Hours Studied")
plt. ylabel (" Test Score")





Linear regression: Hours vs Score • Actual data
Prediction for 11 hrs of Study: 97.4 — Regression line

2) Multiple Regression

import numpy as np
import matplotlib .pyplot as plt
from mpl_toolkits. mplot3d import Axes 3D

x = np. array [(
[800, 2], [1000, 3], [1200, 3],
[1500, 4], [1800, 4], [2000, 5], [2200, 6],
[2800, 6], [3000, 6] ])

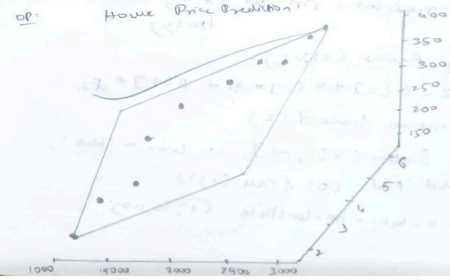
y = np.array ([150, 180, 200, 240, 280, 310, 340,
370, 400, 420]) # Prices in $1000

X = np. column_stack ((np.ones (len(x)), x))
B = np. linalg.inv (X.T.dot(X)).dot (X.T).dot(y)

def predict (sqft , beds):
fig = plt. figure (sqft , beds):

```
return B[0] + B[1] * sqft + B[2] * beds
    fig = plt.figure()
    ax = fig.add_subplot(111, projection = '3d')

    sqft_vals = x[:, 1]
    bed_vals = x[:, 2]
    price_vals = y

ax.scatter(sqft_vals, bed_vals, price_vals, color = 'blue',
label = "Actual Data")
sqft_grid, bed_grid = np.meshgrid(np.linspace(800,
        300, 10), np.linspace(2, 6, 5))
price_grid = predict(sqft_grid, bed_grid)
ax.plot_surface(sqft_grid, bed_grid, price_grid,
    color = 'red', alpha = 0.5)
ax.set_xlabel("Square Footage")
ax.set_ylabel("Bedrooms")
ax.set_zlabel("Price ($1000)")
ax.set_title("multiple linear regression:
            House Price Prediction")

plt.show()
print("Prediction for 2600 sqft, 5 bedrooms:",
        predict(2600, 5))
```

OP: House Price Prediction



**Code:**

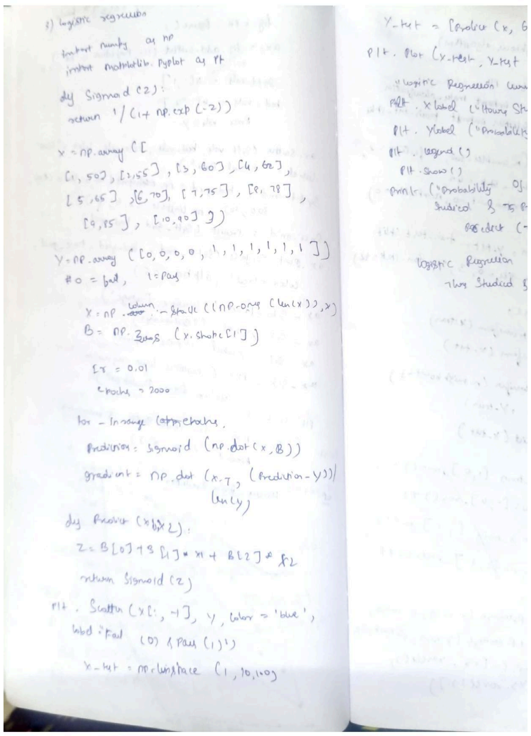
```
from google.colab import files
per_capita_income=files.upload()

from google.colab import files
salary=files.upload()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
from sklearn import linear_model

df1=pd.read_csv("canada_per_capita_income.csv")
df1.head()

df2=pd.read_csv("salary.csv")
df2.YearsExperience.median()
df2.YearsExperience =
df2.YearsExperience.fillna(df2.YearsExperience.median()) df2

plt.xlabel("year")
plt.ylabel("per capita income (US$)")
plt.scatter(df1.year, df1['per capita income (US$)'])

plt.xlabel("YearsExperience")
plt.ylabel("Salary")
plt.scatter(df2.YearsExperience, df2.Salary)

reg1 = linear_model.LinearRegression()
reg1.intercept_
reg1.predict([[2020]])

reg2 = linear_model.LinearRegression()
reg2.fit(df2.drop('Salary', axis='columns'), df2['Salary'])
reg2.coef_
reg2.intercept_
reg2.predict([[12]])

from google.colab import files
hiring=files.upload()

from google.colab import files
companies=files.upload()
```

```
df3=pd.read_csv("hiring.csv")
df3.head()

df4=pd.read_csv("1000_Companies.csv")
df4.head()

df3.isnull().sum()
df4.isnull().sum()

df3_copy = df3.copy()
experience_mapping = {'two': 2, 'three': 3, 'five': 5, 'seven': 7, 'ten': 10, 'eleven': 11}
df3_copy['experience'] = df3_copy['experience'].map(experience_mapping)
median_experience = df3_copy['experience'].median()
df3_copy['experience'] = df3_copy['experience'].fillna(median_experience)
df3_copy
df3_copy['test_score(out of 10)'] = df3_copy['test_score(out of 10)'].fillna(df3_copy['test_score(out of
10)'].mean())
reg3 = linear_model.LinearRegression()
reg3.fit(df3_copy.drop('salary($)', axis='columns'), df3_copy['salary($)'])
reg3.coef_
reg3.intercept_
reg3.predict([[2,9,6]])
reg3.predict([[12,10,10]])
ohe = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
state_encoded = ohe.fit_transform(df4[['State']])
state_encoded_df = pd.DataFrame(state_encoded, columns=ohe.get_feature_names_out(['State']))

df4 = pd.concat([df4, state_encoded_df], axis=1).drop(columns=['State'])
print(df4)
reg4 = linear_model.LinearRegression()
reg4.fit(df4.drop('Profit',axis='columns'),df4.Profit)
print(reg4.coef_)
print(reg4.intercept_)
reg4.predict([[91694.48, 515841.3, 11931.24,0,1,0]])
```

<u>**Program 4**</u>

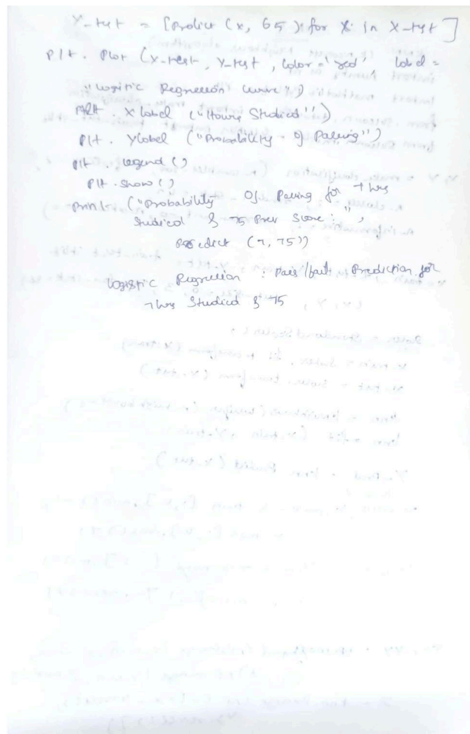
## Build Logistic Regression Model for a given dataset

## Screenshot:



```
3) logistic regression

import numpy as np
import matplotlib.pyplot as plt

def Sigmoid (z):
    return 1/(1+ np.exp (-z))

x = np.array ([
[1, 50], [2,55], [3, 60], [4, 62],
[5, 65], [6, 70], [7,75], [8, 78],
[9, 85], [10, 90] ])

Y = np.array ([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
# 0 = fail,  1 = pass

X = np.column_stack ([np.ones (len(x)), x])
B = np.zeros (x.shape[1])

lr = 0.01
epochs = 2000

for _ in range (epochs).
    Prediction = sigmoid (np.dot (x, B))
    gradient = np.dot (x.T, (Prediction - y)) / len(y)

def Predict (x1, x2):
    z = B[0] + B[1] * x1 + B[2] * x2
    return Sigmoid (z)

plt. Scatter (x[:, -1], y, color = 'blue',
label ='Fail (0) / Pass (1)')

x_test = np.linspace (1, 10, 100)
```





```
Y_test = [Predict (x, 65) for x in x_test]
plt. Plot (x_test, y_test, color = 'red' label =
    "Logistic Regression Curve")
plt. xlabel ("Hours Studied")
plt. ylabel ("Probability of Passing")
plt. legend ()
plt. show ()
print ("Probability of Passing for 7 hrs
    Studied & 75 Prev Score: ",
        predict (7, 75))

Logistic Regression : Pass/fail Prediction for
7 hrs Studied & 75
```

**Code:**

```
from google.colab import files
hr=files.upload()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy import stats
from sklearn import linear_model
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

df1=pd.read_csv("HR_comma_sep.csv")
df1.head()
df1.isnull().sum()
plt.figure(figsize=(12, 6))
sns.barplot(x='Department', y='left', data=df1)
plt.title('Employee Retention Rate by Department')
plt.xlabel('Department')
plt.ylabel('Proportion of Employees Left')
plt.xticks(rotation=45, ha='right')
plt.show()

ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
department_encoded = ohe.fit_transform(df1[['Department']])
department_encoded_df = pd.DataFrame(department_encoded,
columns=ohe.get_feature_names_out(['Department']))
df1 = pd.concat([df1, department_encoded_df], axis=1)
df1 = df1.drop('Department', axis=1)
ordinal_encoder = OrdinalEncoder(categories=[['low', 'medium', 'high']], dtype=np.int64)
salary_encoded = ordinal_encoder.fit_transform(df1[['salary']])
df1['salary_encoded'] = salary_encoded
df1 = df1.drop('salary', axis=1)
df1.head()

correlation_matrix = df1.corr()
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Features')
plt.show()
plt.figure(figsize=(8, 6))
sns.barplot(x='salary_encoded', y='left', data=df1)
plt.title('Impact of Employee Salary on Retention')
plt.xlabel('Salary Level (Encoded)')
plt.ylabel('Proportion of Employees Left')
plt.show()
```

```
df_copy = df1[['number_project', 'average_montly_hours', 'time_spend_company', 'left','salary_encoded',
'satisfaction_level','Work_accident']]
df_copy.head()
X = df_copy.drop('left', axis=1)
y = df_copy['left']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of the Logistic Regression model: {accuracy}")

from google.colab import files
zoodata=files.upload()
zootype=files.upload()

zoo_data = pd.read_csv('zoo-data.csv')
zoo_class = pd.read_csv('zoo-class-type.csv')
merged_data = pd.merge(zoo_data, zoo_class, left_on='class_type', right_on='Class_Number')
merged_data = merged_data.drop(['Animal_Names', 'Number_Of_Animal_Species_In_Class',
'Class_Number','class_type','animal_name'], axis=1)
X = merged_data.drop('Class_Type', axis=1)
y = merged_data['Class_Type']
print(merged_data.head())
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=np.unique(y_test))
disp.plot(cmap="Blues", values_format="d")
plt.title("Confusion Matrix")
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:**

```
ID3
Algorithm

import numpy as np
import pandas as pd
from graphviz import Digraph

def Entropy(dataset):
    class_count = dataset.iloc[:, -1].value_counts()
    Prob = class_count / len(dataset)
    return -np.sum(Prob * np.log2(Prob))

def information_gain(dataset, feature):
    total_entropy = entropy(dataset)
    feature_values = dataset[feature].value_counts()
    weighted_entropy = 0
    for value, count in feature_value.items():
        subset = dataset[dataset[feature] == value]
        weighted_entropy += (count / len(dataset)) * entropy(subset)
    return total_entropy - weighted_entropy

def best_feature(dataset):
    features = dataset.columns[:-1]
    best_feature = None
    for feature in features:
        info_gain = information_gain(dataset, feature)
        if info_gain > best_info_gain:
            best_feature = feature
    return best_feature
```

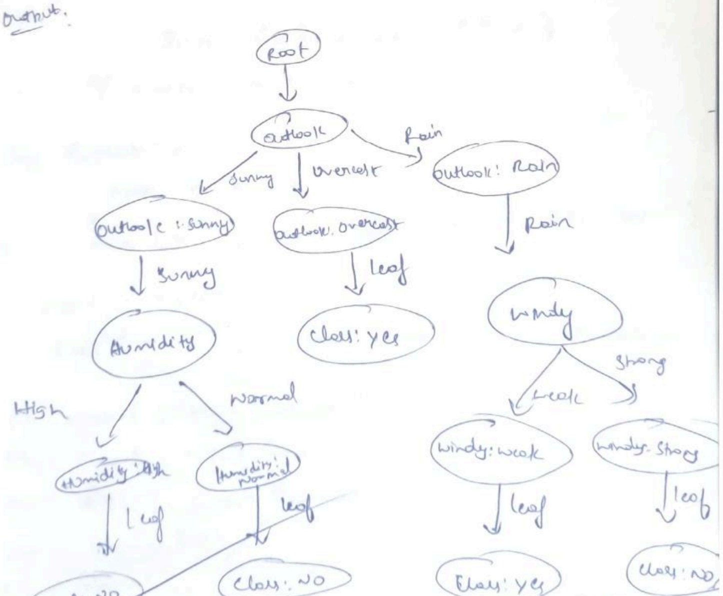


```
def id3(dataset, max_depth = None, depth = 0):
    if len(dataset.iloc[:, -1].unique()) == 1:
        return dataset.iloc[0, -1]
    if len(dataset.columns) == 1:
        return dataset.iloc[:, -1].mode()[0]
    if max_depth is not None and depth >= max_depth
        return dataset.iloc[:, -1].mode()[0]
    best = best_feature(dataset)
    tree = {best: {}}
    for value in dataset[best].unique():
        subset = dataset[dataset[best] == value]
        tree[best][value] = id3(subset.drop(columns=[best]), max_depth=max_depth,
            depth=depth+1)

def create_tree_diagram(tree, dot = None,
        parent_name = "Root", parent_value = ""):
    if dot is None
        dot = Digraph(format = "png", engine = "dot")
    if isinstance(tree, dict):
        for feature, branches in tree.items():
            feature_name = f"{parent_name}_{feature}"
            dot.node(feature_name, feature)
            dot.node(parent_name, feature_name,
                label = str(value))
            create_tree_diagram(subtree, dot, value_name,
                str(value))
```

else:

```
dot.node (Parent_name + "_class", f"class: {tree}")
dot.edge (Parent_name, Parent_name + "_class", label = "leaf")
return dot

data = Pd.read_csv ("/content/weather_forecast.csv")
df = Pd.DataFrame (data)
tree = id3 (df, max_depth = 3)
dot = Create_tree_diagram (tree)
dot.render ("decision_tree", view = True)
```

output:



**Code:**

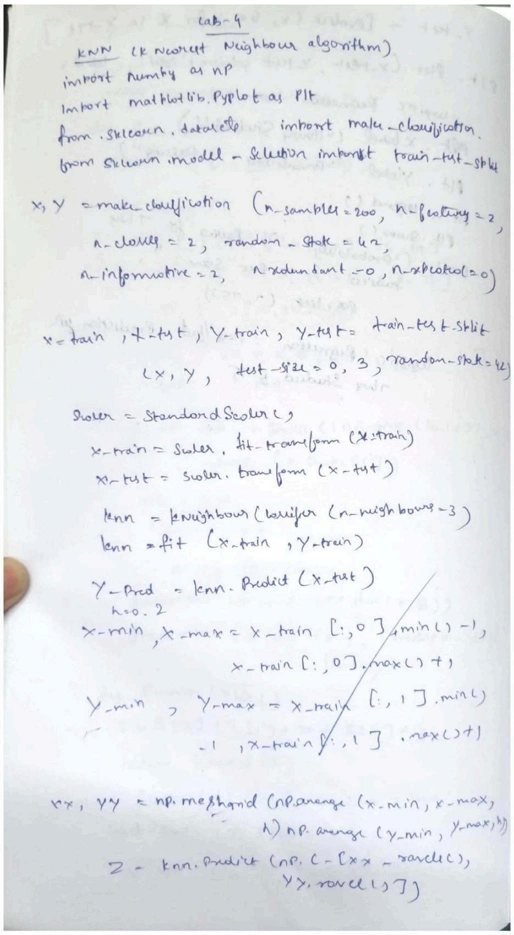
```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris.csv")
df1.head()

df1.isnull().sum()

X = df1.drop('species', axis=1)
y = df1['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred))
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=y.unique())
plt.show()

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
cmap = plt.cm.get_cmap('PuBuGn')
disp.plot(cmap=cmap)
plt.show()

drug=files.upload()
df2=pd.read_csv("drug.csv")
df2.head()
df2.isnull().sum()

label_encoders = {}
for column in df2.columns:
    le = LabelEncoder()
    df2[column] = le.fit_transform(df2[column])
    label_encoders[column] = le
X = df2.drop('Drug', axis=1)
y = df2['Drug']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
print(classification_report(y_test, y_pred))
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=X.columns, class_names=[str(c) for c in y.unique()])
plt.show()

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
```

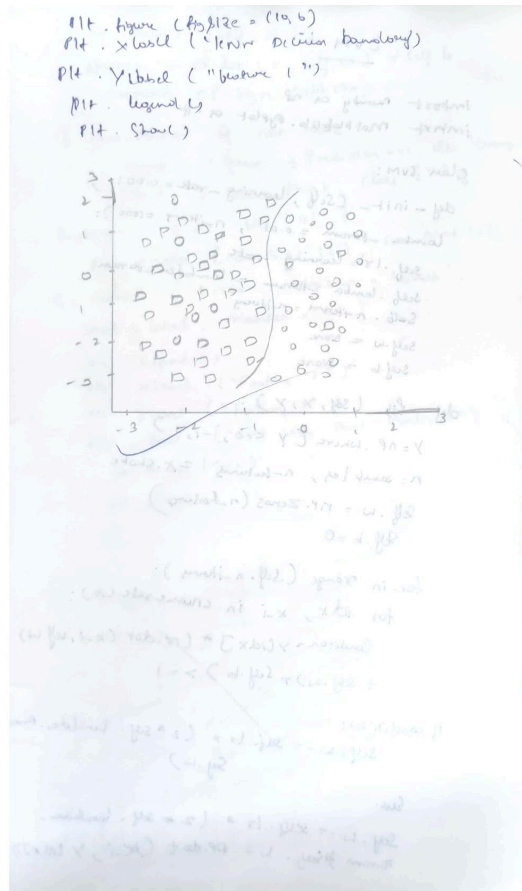
```
cmap = plt.cm.Blues
disp.plot(cmap=cmap)
plt.show()

pc=files.upload()
df3=pd.read_csv("petrol_consumption.csv")
df3.head()
df3.isnull().sum()
X = df3.drop('Petrol_Consumption', axis=1)
y = df3['Petrol_Consumption']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
regressor = DecisionTreeRegressor(random_state=42)
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error: {mse:.2f}')
print(f'Root Mean Squared Error: {rmse:.2f}')
print(f'Mean Absolute Error: {mae:.2f}')
print(f'R-squared: {r2:.2f}')
plt.figure(figsize=(30, 30))
plot_tree(regressor, filled=True, feature_names=X.columns, fontsize=10)
plt.show()
```

<u>**Program 6**</u>

**Build KNN Classification model for a given dataset.**

**Screenshot:**








**Code:**

```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris (2).csv")
df1.head()
df1.isnull().sum()
X = df1.drop('species', axis=1)
y = df1['species']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
best_k = 1
best_accuracy = 0
for k in range(1, 11):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy for k={k}: {accuracy}, Error Rate for k={k}: {1-accuracy}")
```

```
    if             accuracy             >
        best_accuracy:
        best_accuracy = accuracy
        best_k = k
print(f"Best k value: {best_k}")
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

diabetes=files.upload()
df2=pd.read_csv("diabetes.csv")
df2.head()
df2.isnull().sum()
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df2.drop('Outcome', axis=1))
X_train, X_test, y_train, y_test = train_test_split(X_scaled, df2['Outcome'], test_size=0.2, random_state=42)
best_k = 1
best_accuracy = 0
for k in range(1, 11):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy for k={k}: {accuracy}")
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k
print(f"Best k value: {best_k}")

knn = KNeighborsClassifier(n_neighbors=best_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
```

```
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
print("\nClassification
Report:")
print(classification_report(y_test, y_pred))

heart=files.upload()
df3=pd.read_csv("heart.csv")
df3.head()
df3.isnull().sum()
X = df3.drop('target', axis=1)
y = df3['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
best_k = 1
best_accuracy = 0
for k in range(1, 11):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"Accuracy for k={k}: {accuracy}, Error Rate for k={k}: {1-accuracy}")
    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_k = k
print(f"Best k value: {best_k}")
knn = KNeighborsClassifier(n_neighbors=optimal_k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=knn.classes_, yticklabels=knn.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

## Program7

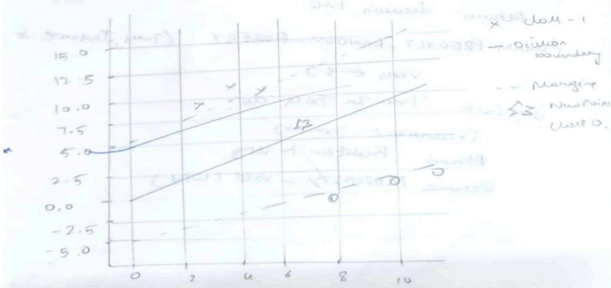
**Build Support vector machine model for a given dataset**

**Screenshot:**

SVM

```
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 )
    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.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]))
```

```
def Predict (self, x):
    approx = np.dot (x, self.w) + self.b
    return np.sign (approx)
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, edgecolor = 'black',
    label = label, marker = '*' )
    ax.legend ()
    plt.xlabel ("Feature 1")
    plt.ylabel ("Feature 2")
    plt.title ("SVM with Point Prediction")
    plt.grid (True)
    plt.show ()
```






**Code:**

```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris (1).csv")
df1.head()
X = df1.drop('species', axis=1)
y = df1['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
rbf_svm = SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)
rbf_y_pred = rbf_svm.predict(X_test)
print("RBF Kernel SVM:")
print("Accuracy:", accuracy_score(y_test, rbf_y_pred))
cm = confusion_matrix(y_test, rbf_y_pred)
sns.heatmap(cm, annot=True, fmt='d',cmap="Blues")
plt.title('Confusion Matrix for RBF Kernel SVM')
plt.xlabel('Predicted')
```

```
plt.ylabel('True')
plt.show()
print(classification_report(y_test, rbf_y_pred))
linear_svm = SVC(kernel='linear')
linear_svm.fit(X_train, y_train)
linear_y_pred = linear_svm.predict(X_test)
print("\nLinear Kernel SVM:")
print("Accuracy:", accuracy_score(y_test, linear_y_pred))
cm = confusion_matrix(y_test, linear_y_pred)
sns.heatmap(cm, annot=True, fmt='d',cmap="Blues")
plt.title('Confusion Matrix for Linear Kernel SVM')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
print(classification_report(y_test, linear_y_pred))
letter=files.upload()
df2=pd.read_csv("letter-recognition.csv")
df2.head()
X = df2.drop('letter', axis=1)
y = df2['letter']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
svm_classifier = SVC(kernel='linear', probability=True)
svm_classifier.fit(X_train, y_train)
y_pred = svm_classifier.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10,10))
sns.heatmap(cm, annot=True, fmt='d', cmap="Blues")
plt.title('Confusion Matrix for SVM')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
lb = LabelBinarizer()
lb.fit(y_test)
y_test_lb = lb.transform(y_test)
y_pred_prob = svm_classifier.predict_proba(X_test)
fpr = {}
tpr = {}
thresh ={}
roc_auc = dict()
n_class = y_test_lb.shape[1]
for i in range(n_class):
    fpr[i], tpr[i], thresh[i] = roc_curve(y_test_lb[:,i], y_pred_prob[:,i])
    roc_auc[i] = auc(fpr[i], tpr[i])
plt.plot(fpr[0], tpr[0], linestyle='--',color='orange', label='SVM (AUC = %0.2f)' % roc_auc[0])
plt.title('ROC Curve for Class 0')
plt.xlabel('False Positive
Rate') plt.ylabel('True Positive
rate') plt.legend(loc='best')
plt.show()
print(f"AUC score for class 0: {roc_auc[0]}")
```

## Program 8

**Implement Random forest ensemble method on a given dataset**

**Screenshot:**

Lab-5

Random Forest Algorithm

Function RANDOM_FOREST (Data D, Integer T, Integer m):

Trees ← [ ]

for t from 1 to T do:

Bootstrapsample ← Sample-with-Replacement

Tree ← Build-Decision-tree (Bootstrap Sam

Append Tree to Trees

Return Trees

Function Build-Decision-tree (Data D, Integer m):

At each node in tree:

Randomly select m features from all features

find the best split using only these m features

Recursively split until stopping condition is met

return decision tree

Function PREDICT_RANDOM_FOREST (Trees, Instance x

votes ← [ ]

for each Tree in Trees do:

Prediction ← Tree(x)

Append Prediction to votes

Return MAJORITY-VOTE (votes)

Scanned with OKEN Scanner

**Code:**

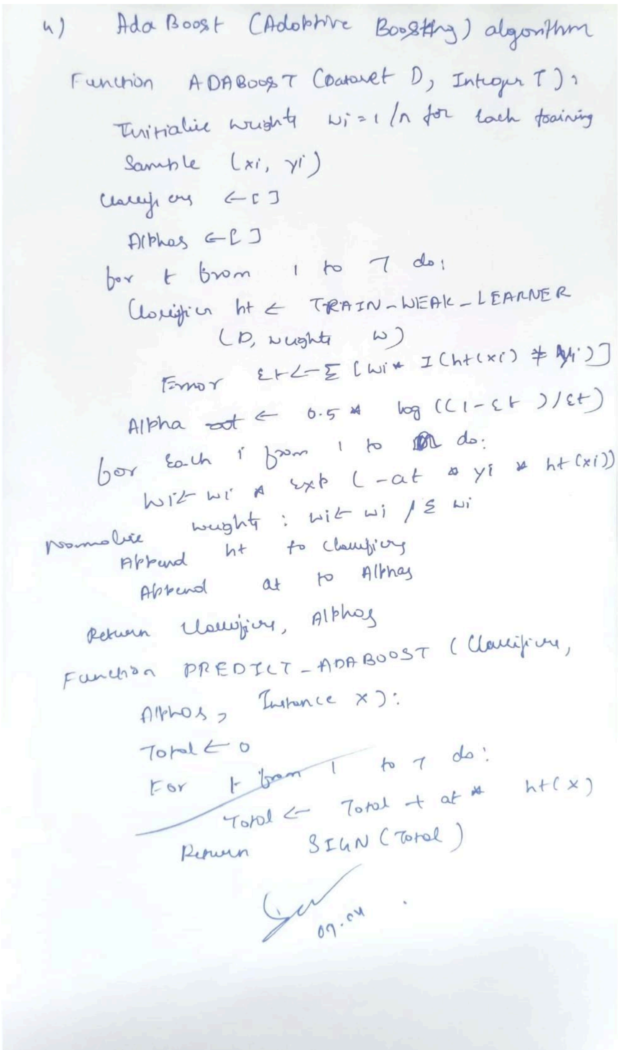
```
from google.colab import files
iris=files.upload()
df1=pd.read_csv("iris (4).csv")
df1.head()
X = df1.drop('species', axis=1)
y = df1['species']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
rf_classifier = RandomForestClassifier(random_state=0)
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)
default_accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with default n_estimators: {default_accuracy}")
best_accuracy = 0
best_n_estimators = 0
for n_estimators in range(1, 101):
   rf_classifier = RandomForestClassifier(n_estimators=n_estimators, random_state=0)
   rf_classifier.fit(X_train, y_train)
   y_pred = rf_classifier.predict(X_test)
   accuracy = accuracy_score(y_test, y_pred)
   if accuracy > best_accuracy:best_accuracy
   = accuracy best_n_estimators =
   n_estimators
print(f"\nBest accuracy: {best_accuracy} achieved with n_estimators = {best_n_estimators}")
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
         xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

```
        best_accuracy = accuracy
        best_n_estimators = n_estimators
print(f"\nBest accuracy: {best_accuracy} achieved with n_estimators = {best_n_estimators}")
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=np.unique(y_test), yticklabels=np.unique(y_test))
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

**Program 9**

**Implement Boosting ensemble method on a given dataset**

**Screenshot:**



4) Ada Boost (Adaptive Boosting) algorithm

Function ADABOOST (Dataset D, Integer T):
Initialize weights wi = 1/n for each training sample (xi, yi)
Classifiers ← [ ]
Alphas ← [ ]
for t from 1 to T do:
Classifier ht ← TRAIN-WEAK-LEARNER (D, weights w)
Error εt ← Σ [wi * I(ht(xi) ≠ yi)]
Alpha ~~αt~~ ← 0.5 * log ((1 − εt)/εt)
for each i from 1 to n do:
wi ← wi * exp (−at * yi * ht(xi))
normalize weight : wi ← wi / Σ wi
Append ht to classifiers
Append at to Alphas
Return classifiers, Alphas
Function PREDICT-ADABOOST (Classifiers, Alphas, Instance x):
Total ← 0
For t from 1 to T do:
Total ← Total + at * ht(x)
Return SIGN (Total)

09.04



**Code:**

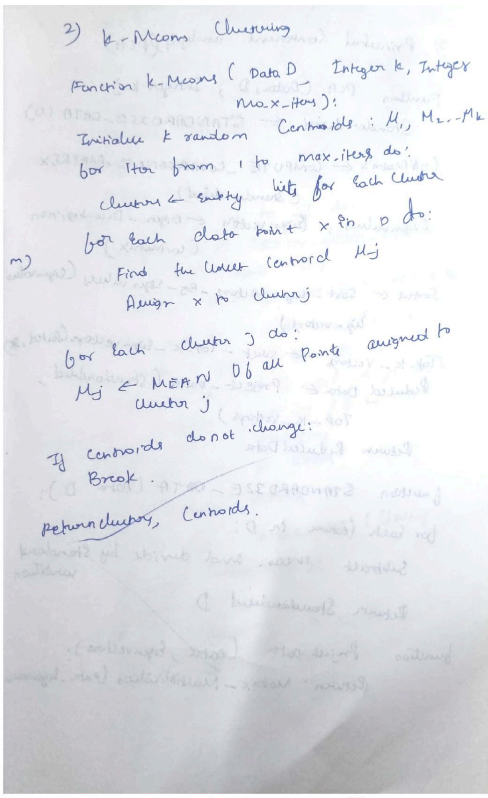
```
from google.colab import files
income=files.upload()
df1=pd.read_csv("income.csv")
df1.head()
X = df1.drop('income_level', axis
```

```
y = df1['income_level']
X = pd.get_dummies(X)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
abc = AdaBoostClassifier(n_estimators=10, random_state=42)
abc.fit(X_train, y_train)
y_pred = abc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Initial AdaBoost accuracy (10 trees): {accuracy}")
param_grid = {'n_estimators': [50, 100, 150, 200]}
grid_search = GridSearchCV(AdaBoostClassifier(random_state=42), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
print(f"Best parameters: {grid_search.best_params_}")
print(f"Best cross-validation score: {grid_search.best_score_}")
best_abc = grid_search.best_estimator_
y_pred_best = best_abc.predict(X_test)
best_accuracy = accuracy_score(y_test, y_pred_best)
print(f"Accuracy of the best model on the test set: {best_accuracy}")
cm = confusion_matrix(y_test, y_pred_best)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['<=50K', '>50K'], yticklabels=['<=50K', '>50K'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

# Program 10

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

**Screenshot:**


2) k-Means Clustering

Function k-Means ( Data D Integer k, Integer max_iters ):
Initialize k random Centroids : $\mu_1, \mu_2, ..\mu_k$
for Iter from 1 to max_iters do:
clusters ← empty lists for each cluster
for each data point x in D do:
Find the closest centroid $\mu_j$
Assign x to cluster j
for each cluster j do:
$\mu_j$ ← MEAN of all points assigned to cluster j
If centroids do not change:
Break.
Return clusters, Centroids.

Scanned with OKEN Scanner

**Code:**

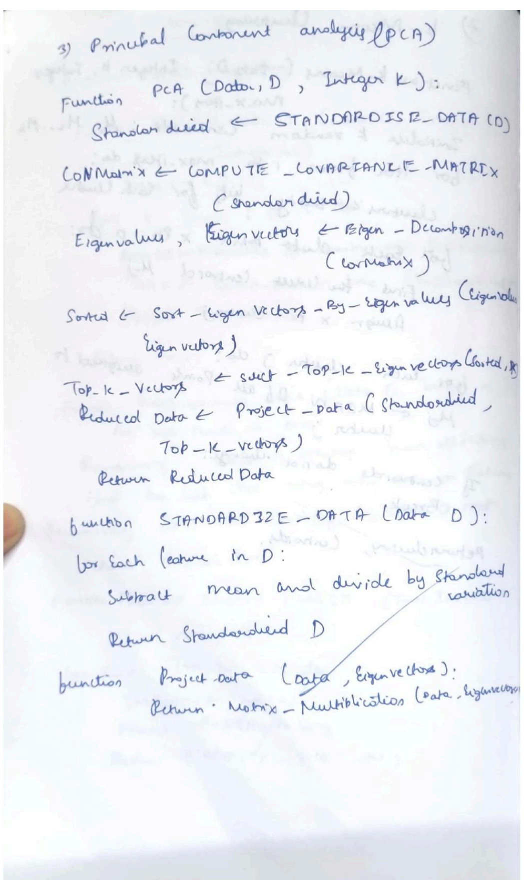
```
from google.colab import files
iris=files.upload()
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from scipy import stats
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import 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 (4).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:**



3) Principal Component analysis (PCA)

Function PCA (Data, D , Integer K):

Standardised ← STANDARDISE_DATA (D)

CovMatrix ← COMPUTE_COVARIANCE_MATRIX (Standardised)

Eigenvalues, Eigenvectors ← Eigen_Decomposition (CovMatrix)

Sorted ← Sort_Eigen Vectors_By_Eigen values (Eigenvalues, Eigen vectors)

Top_k_Vectors ← Select_Top_k_Eigen vectors (Sorted, k)

Reduced Data ← Project_Data (Standardised, Top_k_vectors)

Return Reduced Data

function STANDARDIZE_DATA (Data D):

for each feature in D:

Subtract mean and divide by standard variation

Return Standardised D

function Project_data (data, Eigenvectors):

Return: Matrix_Multiplication (data, Eigenvectors)

**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 (1).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]}")
```