

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



LAB REPORT

on

Machine Learning (23CS6PCMAL)

Submitted by

Manasvini Deepak (1BM22CS336)

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 **Manasvini Deepak (1BM22CS336)**, who is bonafide student of **B.M.S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum. The Lab report has been approved as it satisfies the academic requirements in respect of an Machine Learning (23CS6PCMAL) work prescribed for the said degree.

| | |
|---|--|
| Sarla DV 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 | 4 |
| 2 | 3-3-2025 | Demonstrate various data pre-processing techniques for a given dataset | 7 |
| 3 | 10-3-2025 | Implement Linear and Multi-Linear Regression algorithm using appropriate dataset | 13 |
| 4 | 17-3-2025 | Build Logistic Regression Model for a given dataset | 24 |
| 5 | 24-3-2025 | Use an appropriate data set for building the decision tree (ID3) and apply this knowledge to classify a new sample. | 28 |
| 6 | 7-4-2025 | Build KNN Classification model for a given dataset. | 33 |
| 7 | 21-4-2025 | Build Support vector machine model for a given dataset | 45 |
| 8 | 5-5-2025 | Implement Random forest ensemble method on a given dataset | 53 |
| 9 | 5-5-2025 | Implement Boosting ensemble method on a given dataset. | 56 |
| 10 | 12-5-2025 | Build k-Means algorithm to cluster a set of data stored in a .CSV file. | 59 |
| 11 | 12-5-2025 | Implement Dimensionality reduction using Principal Component Analysis (PCA) method. | 65 |

Github Link: <https://github.com/ManasviniDeepak/6thsem-ML-Lab>

Program 1

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

Screenshot

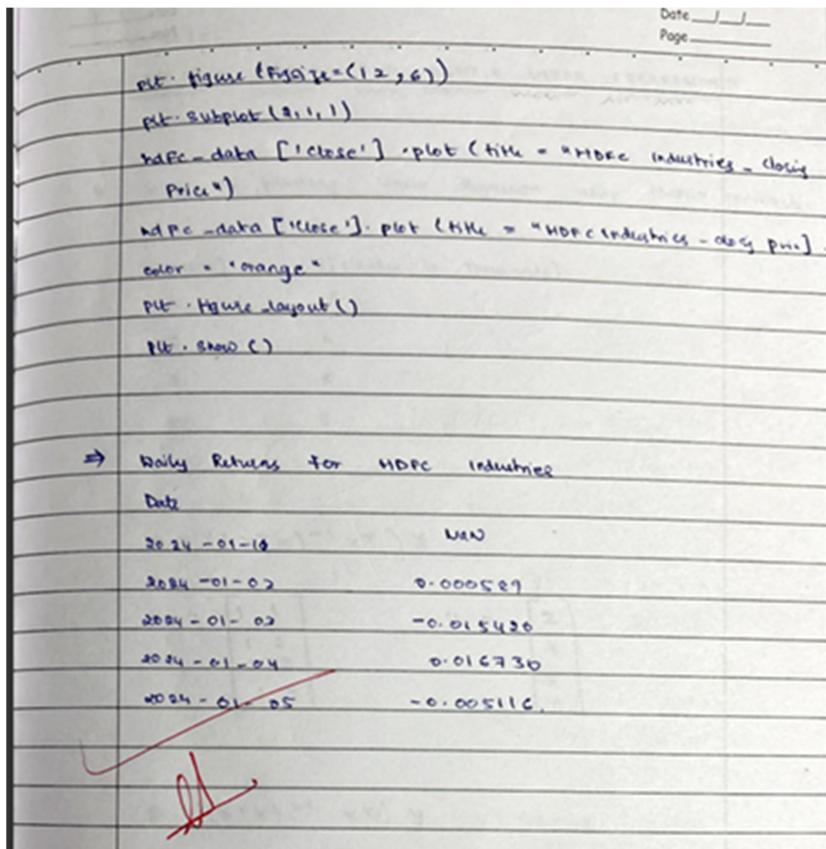
```
14-03-25
LAB 06: IMPORT AND EXPORT
Date _____
Page _____
Imports pandas as pd

1 df = pd.DataFrame([{"id": 1, "name": "A", "marks": 10}, {"id": 2, "name": "B", "marks": 20}, {"id": 3, "name": "C", "marks": 30}, {"id": 4, "name": "D", "marks": 40}, {"id": 5, "name": "E", "marks": 60}])
print(df)

      ID      Name  Marks
0    1        A     10
1    2        B     20
2    3        C     30
3    4        D     40
4    5        E     60

2 tickers = ["AAPL.NAS", "MSFT.NAS", "GOOG.NAS"]
data = yf.download(tickers, start="2024-01-01", end="2024-01-31", groupby=tickers)
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt

yf_data = data['AAPL.NAS']
yf_data['Daily Return'] = yf_data['Close'].pct_change()
print("In closing price for Apple Industries:")
print(yf_data['Daily Return'].head(10))
```



Code:

```

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

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

```

```

df.head()
df.shape
print(df.info())
# Summary statistics
print(df.describe())
missing_values=df.isnull().sum()
print(missing_values[missing_values > 0])
categorical_cols = df.select_dtypes(include=['object']).columns
print("Categorical columns identified:", categorical_cols)
if len(categorical_cols) > 0:
    df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
    print("\nDataFrame after one-hot encoding:")
    print(df.head())
else:
    print("\nNo categorical columns found in the dataset.")
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import pandas as pd

numerical_cols = df.select_dtypes(include=['number']).columns

scaler = MinMaxScaler()
df_minmax = df.copy() # Create a copy to avoid modifying the original
df_minmax[numerical_cols] = scaler.fit_transform(df[numerical_cols])

scaler = StandardScaler()
df_standard = df.copy()
df_standard[numerical_cols] = scaler.fit_transform(df[numerical_cols])
print("\nDataFrame after Min-Max Scaling:")
print(df_minmax.head())
print("\nDataFrame after Standardization:")
print(df_standard.head())

#**Adult Income Dataset**
df1=pd.read_csv('content/adult.csv')
df1.head()
df1.shape
print(df1.info())
# Summary statistics
print(df1.describe())
missing_values=df1.isnull().sum()
print(missing_values[missing_values > 0])
categorical_cols = df1.select_dtypes(include=['object']).columns
print("Categorical columns identified:", categorical_cols)
if len(categorical_cols) > 0:
    df1 = pd.get_dummies(df1, columns=categorical_cols, drop_first=True)
    print("\nDataFrame after one-hot encoding:")
    print(df1.head())
else:

```

```
print("\nNo categorical columns found in the dataset.")
from sklearn.preprocessing import MinMaxScaler, StandardScaler
import pandas as pd

numerical_cols = df1.select_dtypes(include=['number']).columns

scaler = MinMaxScaler()
df_minmax = df1.copy() # Create a copy to avoid modifying the original
df_minmax[numerical_cols] = scaler.fit_transform(df1[numerical_cols])

scaler = StandardScaler()
df_standard = df1.copy()
df_standard[numerical_cols] = scaler.fit_transform(df1[numerical_cols])
print("\nDataFrame after Min-Max Scaling:")
print(df_minmax.head())
print("\nDataFrame after Standardization:")
print(df_standard.head())
```

Program 2

Demonstrate various data pre-processing techniques for a given dataset

Screenshot

LAB 2: DATA PROCESSING

```

a) housing.csv:
import pandas as pd
df = pd.read_csv("housing.csv")

print("Information of all columns:")
print(df.info())
print("\n")
print("Statistical information of all numerical columns:")
print(df.describe())
print("\n")

print("Count of unique labels for 'Ocean Proximity' column")
print(df['ocean_proximity'].value_counts())
print("\n")

print("Columns with missing values count greater than zero")
print((df.isnull().sum()[(df.isnull().sum() > 0)])

```

OUTPUT

| Column | Type |
|-----------|---------|
| longitude | float64 |

```

Statistical information of all numerical columns:
      longitude      latitude      housing_median_age      total_rooms      ...
count    20640.000  20640.000  20640.000000  20640.00000
mean     2.339442   37.981850   6.01373000  1270.00000
std      1.429439   11.389680   1.78546000  1087.00000
min     -0.469219   1.000000   1.00000000  5.00000
25%     -0.124927   3.000000   3.00000000  10.00000
50%     0.000000   5.000000   5.00000000  15.00000
75%     0.459814   12.000000  12.00000000  20.00000
max     1.789054   46.000000  48.00000000  400.00000

```

total_bedrooms population households median_income median_house_value

| total_bedrooms | population | households | median_income | median_house_value |
|----------------|------------|------------|---------------|--------------------|
| 20640.000 | 2064000 | 3936000 | 58360.000 | 454320.000 |

count of unique labels for 'Ocean Proximity' column

ocean_proximity

| NEAR OCEAN | INLAND |
|------------|--------|
| 9130 | 6510 |

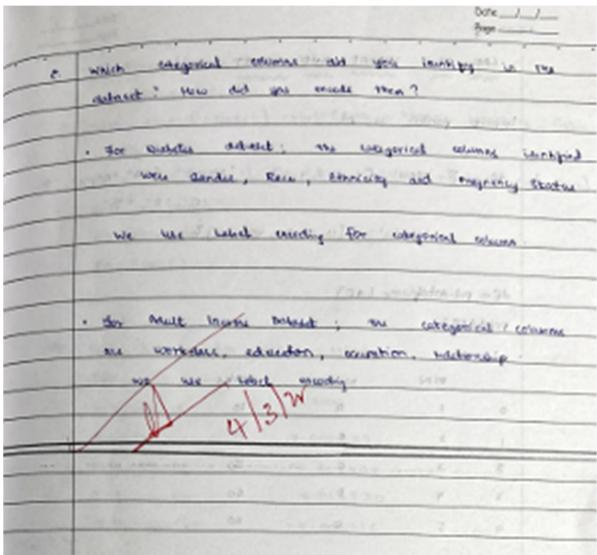
columns with missing values count greater than zero:

total_bedrooms 204

ANALYSIS OF DIABETES AND ADULT INCOME DATASET

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

- For diabetes dataset: numerical columns which were missing were glucose and blood pressure. We handled it by replacing with mean.
- Categorical columns which were missing are gender or we handled it by most frequent imputation.
- For adult income dataset the numerical columns which were missing were workclass, age. We handled it by mean imputation. The categorical columns missing were marital status, education. We handled it by most frequent imputation.



Code:

```

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

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

df.head(2)

df.describe()

df.info()

sns.histplot(df['median_income'], kde=True, color='green')

sns.histplot(df['housing_median_age'])

from sklearn.model_selection import train_test_split

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

y = df["median_house_value"]

```

```

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

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

y = df["median_house_value"]

df["income_cat"] = pd.cut(df["median_house_value"],

bins=[0, 100000, 200000, 300000, 400000, np.inf],

labels=[1, 2, 3, 4, 5])

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=df["income_cat"])

train_set = X_train.copy()

train_set["median_house_value"] = y_train

train_set.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4, s=train_set["population"]/100, label="population", figsize=(10,7), c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True)

plt.legend()

numerical_columns = df.select_dtypes(include=['float64', 'int64'])

correlation_matrix = numerical_columns.corr()

print(correlation_matrix["median_house_value"].sort_values(ascending=False))

df.plot(kind="scatter", x="median_income", y="median_house_value", alpha=0.1)

# Combine 'median_income' and 'households'

df["income_households"] = df["median_income"] * df["households"]

numerical_columns = df.select_dtypes(include=['float64', 'int64'])

correlation_matrix = numerical_columns.corr()

print(correlation_matrix["median_house_value"].sort_values(ascending=False))

df.plot(kind="scatter", x="income_households", y="median_house_value", alpha=0.1)

```

```

plt.show()

missing_values = df.isnull().sum()

print(missing_values[missing_values > 0])

h=df

h.dropna(subset=["total_bedrooms"])

from sklearn.preprocessing import OneHotEncoder

df1=pd.read_csv('housing.csv')

hc=df1[['ocean_proximity']]

encoder=OneHotEncoder()

hc_encoded=encoder.fit_transform(hc).toarray()

hc_1hot_df = pd.DataFrame(hc_encoded, columns=encoder.get_feature_names_out(hc.columns))

hc_1hot_df.head()

```

Feature scaling is crucial in machine learning for several reasons, particularly when using algorithms that are sensitive to the scale of features. Here's a breakdown of its importance:

1. Improved Performance of Distance-Based Algorithms:
2. Faster Convergence of Gradient Descent:
3. Improved Regularization:
4. Better Interpretation of Coefficients:
5. Numerical Stability:

```

from sklearn.base import BaseEstimator, TransformerMixin

from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer

from sklearn.preprocessing import StandardScaler

```

```

# Custom transformer to add engineered attributes

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):

    def __init__(self, add_bedrooms_per_room=True):
        self.add_bedrooms_per_room = add_bedrooms_per_room

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

    def transform(self, X):
        # Assumes X is a NumPy array with the following columns:
        # total_rooms (index 3), total_bedrooms (index 2), population (index 4), households (index 5)

        rooms_per_household = X[:, 3] / X[:, 5]
        population_per_household = X[:, 4] / X[:, 5]

        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, 2] / X[:, 3]

        return np.c_[X, rooms_per_household, population_per_household, bedrooms_per_room]

    else:
        return np.c_[X, rooms_per_household, population_per_household]

# Identify numerical and categorical columns

num_attribs = df1.drop("ocean_proximity", axis=1).columns # All numeric columns
cat_attribs = ["ocean_proximity"]

# Build numerical pipeline: impute missing values, add new attributes, then scale

num_pipeline = Pipeline([

```

```
('imputer', SimpleImputer(strategy="median"))),  
('attribs_adder', CombinedAttributesAdder()),  
('std_scaler', StandardScaler()),  
  
# Build the full pipeline combining numerical and categorical processing  
full_pipeline = ColumnTransformer([  
    ("num", num_pipeline, num_attribs),  
    ("cat", OneHotEncoder(), cat_attribs),  
  
    # Process the dataset using the pipeline  
housing_prepared = full_pipeline.fit_transform(housing)  
  
print("Shape of processed data:", housing_prepared.shape)
```

Program 3

Implement Linear and Multi-Linear Regression algorithm using appropriate dataset

Screenshot

| | |
|--|--|
| <p>Date _____ Page _____</p> <p>LAB 04 - LINEAR AND MULTIPLE LINEAR REGRESSION</p> <p>Given below is a simple dataset</p> <p>solve the following Linear Regression using Matrix approach</p> <p>Variables) y: Value (in thousands) price (in \$)</p> <p>1 2 2 4 3 6 4 8</p> <p>($x_1^2 = 1 + 4 + 9 + 16 = 30$) ($x_1x_2 = 2 \cdot 4 + 4 \cdot 6 + 6 \cdot 8 = 64$)</p> <p>$\hat{y} = ((x^T x)^{-1} x^T) y$</p> <p>$x = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}$ $y = \begin{bmatrix} 2 \\ 4 \\ 6 \\ 8 \end{bmatrix}$</p> <p>$x^T x = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix} = \begin{bmatrix} 10 & 10 \\ 10 & 30 \end{bmatrix}$</p> <p>$(x^T x)^{-1} = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$</p> <p>$(x^T x)^{-1} x^T = \begin{bmatrix} 2 & 0.5 & 0 & -0.5 \\ -0.5 & 0.1 & 0.1 & 0.3 \end{bmatrix}$</p> <p>$((x^T x)^{-1} x^T) y = \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix}$ $\hat{y} = -0.5 + 2.2$</p> | <p>housing-area-price = [Linear Regression]</p> <pre> import pandas as pd import numpy as np from sklearn.linear_model import LinearRegression import matplotlib.pyplot as plt df = pd.read_csv('housing-area-price.csv') plt.plot(df['Area'], df['Price']) plt.xlabel('Area') plt.ylabel('Price (\$k)') plt.show() X = df['Area'].values Y = df['Price'].values reg = LinearRegression() reg.fit(X, Y) print("Coefficient:", reg.coef_) print("Intercept:", reg.intercept_) area = 3200 + reg.predict([[3200]]) print(f"Predicted area for 3200 sqft is {area} k\$") area = 5000 + reg.predict([[5000]]) print(f"Predicted area for 5000 sqft is {area} k\$") </pre> |
|--|--|

Output:
 Coefficient : [-135.39361 11. -199610.438356]
 Intercept : 199610.438356
 Predicted output for 3200 sq ft is 22083.85 22200
 Predicted output for 5000 sq ft is 25945.75 25950.

 ⇒
 Output : multiple_lr/home price = (-135.39361 * area)
 + (11. * no. of rooms) + 199610.438356
 Coefficient : [11.000000000000002, 43835.6320394, -135.39361]
 Intercept : 199610.438356
 43835.6320394 * 3200 + -135.39361 * 5000 + 199610.438356
 11.000000000000002 * 3200 + 199610.438356

 • Multiple Linear Regression : Numpy CSV
 import pandas as pd
 import numpy as np
 from sklearn.linear_model import LinearRegression
 df = pd.read_csv('Numpy.csv')
 df = df.rename(columns = {
 'rat-score(df[10])': 'rat-score',
 'interview-score(df[10])': 'interview-score',
 'salary(\$)': 'salary\$', inplace = True)
 df['current_to_income'] =
 word_no_rooms = 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

df['experience'] = df['experience'].apply(lambda x: x / 1000000000)
 df['test-score'] = df['test-score'].apply(lambda x: x / 1000000000)
 df['experience'] = df['experience'].values()
 df['test-score'] = df['test-score'].values()
 df['salary'] = df['salary'].values()

 X = df[['experience', 'test-score', 'interview-score']]
 Y = df['salary']

 model = LinearRegression()
 model.fit(X, Y)
 print("Coeficient : ", model.coef_)
 print("Intercept : ", model.intercept_)

 salary_1 = model.predict([[3, 9, 0.7]])
 salary_2 = model.predict([[12, 10, 0.7]])

 print("Predicted salary_1 : ", salary_1)
 print("Predicted salary_2 : ", salary_2)

 Output:
 Coefficient : [2010.98494 1845.4453 2205.8612]
 Intercept : 19972.312

Code:

```
# -*- coding: utf-8 -*-
```

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn import linear_model
```

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

```
df = pd.read_csv('/content/housing_area_price.csv')
```

```
df

# Commented out IPython magic to ensure Python compatibility.

# %matplotlib inline

plt.xlabel('area')

plt.ylabel('price')

plt.scatter(df.area,df.price,color='red',marker='+')
```

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

```
new_df
```

```
price = df.price
```

```
price
```

```
# Create linear regression object
```

```
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_
```

```
"""Y = m * X + b (m is coefficient and b is intercept)"""

3300*135.78767123 + 180616.43835616432
```

```
"""(1) Predict price of a home with area = 5000 sqr ft"""

16
```

```
reg.predict([[5000]])  
# -*- coding: utf-8 -*-  
  
import pandas as pd  
  
import numpy as np  
  
from sklearn import linear_model  
  
  
df = pd.read_csv('/content/homeprices_Multiple_LR.csv')  
df  
  
"""Data Preprocessing: Fill NA values with median value of a column"""  
  
  
df.bedrooms.median()  
  
  
df.bedrooms = df.bedrooms.fillna(df.bedrooms.median())  
df  
  
  
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

import pandas as pd

from sklearn.linear_model import LinearRegression

# Load the dataset
df1 = pd.read_csv('/content/canada_per_capita_income.csv')

# Prepare the data
X = df1.year.values.reshape(-1, 1) # Features (year)
y = df1['per capita income (US$)'] # Target (per capita income)

# Create and train the linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict per capita income for 2020
year_2020 = [[2020]]
predicted_income = model.predict(year_2020)

print(f'Predicted per capita income for Canada in 2020: {predicted_income[0]:.2f}')

import pandas as pd

from sklearn.linear_model import LinearRegression

import matplotlib.pyplot as plt
```

```

# Load the dataset (canada_per_capita_income.csv)

df1 = pd.read_csv('/content/canada_per_capita_income.csv')

# Prepare the data

X = df1.year.values.reshape(-1, 1) # Features (year)

y = df1['per capita income (US$)'] # Target (per capita income)

# Create and train the linear regression model

model = LinearRegression()

model.fit(X, y)

# Create the plot

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

plt.scatter(X, y, color='blue', label='Data Points') # Now using the correct X and y

plt.plot(X, model.predict(X), color='red', label='Regression Line')

plt.xlabel('Year')

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

plt.title('Per Capita Income in Canada over Time')

plt.legend()

plt.grid(True)

plt.show()

import pandas as pd

from sklearn.linear_model import LinearRegression

```

```
from sklearn.impute import SimpleImputer

# Load the dataset
df = pd.read_csv('content/salary.csv')

# Prepare the data
X = df.iloc[:, :-1].values # Features (years of experience)
y = df.iloc[:, 1].values # Target (salary)

# Impute missing values with the mean
imputer = SimpleImputer(strategy='mean') # Create an imputer object with strategy as mean
X = imputer.fit_transform(X) # Fit and transform the imputer on feature data 'X'

# Create and train the linear regression model
model = LinearRegression()
model.fit(X, y)

# Predict salary for 12 years of experience
years_experience = [[12]]
predicted_salary = model.predict(years_experience)

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

import pandas as pd

from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
```

```

# Load the dataset

df = pd.read_csv('/content/hiring.csv')

# Handle missing values

# Convert 'experience' column to numeric, replacing non-numeric with NaN

df['experience'] = pd.to_numeric(df['experience'], errors='coerce')

imputer = SimpleImputer(strategy='mean')

df['experience'] = imputer.fit_transform(df[['experience']])

df['test_score(out of 10)'] = imputer.fit_transform(df[['test_score(out of 10)']])

# Prepare the data

X = df.drop('salary($)', axis='columns')

y = df['salary($)']

# Create and train the linear regression model

model = LinearRegression()

model.fit(X, y)

# Predict salaries for the given candidates

candidate1 = [[2, 9, 6]]

candidate2 = [[12, 10, 10]]


predicted_salary1 = model.predict(candidate1)

```

```

predicted_salary2 = model.predict(candidate2)

print(f"Predicted salary for candidate 1: ${predicted_salary1[0]:.2f}")
print(f"Predicted salary for candidate 2: ${predicted_salary2[0]:.2f}")

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import LabelEncoder, OneHotEncoder

from sklearn.compose import ColumnTransformer

# Load the dataset

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

# Separate features (X) and target (y)

X = df.iloc[:, :-1].values

y = df.iloc[:, 4].values

# Encode categorical data (State)

labelencoder = LabelEncoder()

X[:, 3] = labelencoder.fit_transform(X[:, 3])

ct = ColumnTransformer(

    transformers=[('encoder', OneHotEncoder(), [3])],

    remainder='passthrough'

)

X = ct.fit_transform(X)

```

```
# Avoid dummy variable trap (remove one encoded column)
X = X[:, 1:]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Create and train the multiple linear regression model
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predict profit for the given values
new_prediction = regressor.predict([[1, 0, 91694.48, 515841.3, 11931.24]])

print(f"Predicted Profit: {new_prediction[0]:.2f}")
```

Program 4

Build Logistic Regression Model for a given dataset

Screenshot

1/12/25
Date _____
Page _____

Ques 03: Logistic Regression Model

a) $\alpha_0 = -15 \quad \alpha_1 = 0.8 \quad n = 2$

b) $p(x) = \frac{1}{1 + e^{-(\alpha_0 + \alpha_1 \cdot x)}}$

$$= \frac{1}{1 + e^{-(15 + 0.8 \cdot 2)}}$$

c) $n = 2 \quad \frac{1}{1 + e^{-(15 + 0.8 \cdot 2)}} = 0.04$

d) since $p(x) > 0.5$ class = pass.

e) $x = [2, 1, 0]$

$\text{softmax}(x_1) = \frac{e^{x_1}}{e^{x_1} + e^{x_2}} = 0.6252$

$\text{softmax}(x_2) = \frac{e^{x_2}}{e^{x_1} + e^{x_2}} = 0.3747$

$\text{softmax}(x_3) = \frac{e^{x_3}}{e^{x_1} + e^{x_2} + e^{x_3}} = 0.0001$

100
10⁻⁵

```
# Logistic Regression - Binary
import pandas as pd
from matplotlib import pyplot as plt
df = pd.read_csv('https://raw.githubusercontent.com/ageron/handson-ml2/master/datasets/binary.csv')
df.head()

# df = pd.read_csv('https://raw.githubusercontent.com/ageron/handson-ml2/master/datasets/binary.csv')
# df = pd.read_csv('https://raw.githubusercontent.com/ageron/handson-ml2/master/datasets/binary.csv')

X_train, X_test, y_train, y_test = train_test_split(
    df[['age']], df['bought'], test_size=0.9, random_state=42)
X_train.shape

# from sklearn.linear_model import LogisticRegression
# model = LogisticRegression()
# model.fit(X_train, y_train)

y_pred = model.predict(X_test)
model.score(X_test, y_test)
model.predict_proba(X_test)

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

def predict_pass(age):
    z = 0.182 * age - 4.923
    y = sigmoid(z)
    return y

age = 25
print(predict_pass(age))
OUTPUT: 0.3709839
```

```

Logistic Regression - multiclass

import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import metrics
import matplotlib.pyplot as plt

iris = pd.read_csv("iris.csv")
iris.head(3)

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

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

model = LogisticRegression(multi_class = 'multinomial')

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

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy of multinomial Logistic Regression Model: " + str(accuracy))

OUTPUT:
Accuracy of multinomial Logistic Regression: 1.00

```

* **Quercus:**

- (a) HR dataset:
- (i) * satisfaction level
- * number of projects
- * avg. monthly hours
- * time spent in training
- * work accidents
- * salary level
- * department

(ii) Accuracy: 92.61%

(iii) See dataset:

- (iv) Yes - data properties
- (v) No - data - found missing or inconsistent
- (vi) very high accuracy.

(vii) * Reproce (class 3)

* Risk (class 4)

Code:

```

import pandas as pd

import numpy as np

df=pd.read_csv("/content/HR_comma_sep.csv")

df.head(3)

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

print(df.groupby('left').mean(numeric_only=True))

print(df.groupby('salary').mean(numeric_only=True))

import matplotlib.pyplot as plt

pd.crosstab(df.salary,df.left).plot(kind='bar')

```

```

plt.title('Employee Retention vs Salary')

plt.xlabel('Salary')

plt.ylabel('Number of Employees')

plt.show()

pd.crosstab(df.Department, df.left).plot(kind='bar')

plt.title('Employee Retention vs Department')

plt.xlabel('Department')

plt.ylabel('Number of Employees')

plt.show()

salary_dummies = pd.get_dummies(df.salary, prefix="salary")

dept_dummies = pd.get_dummies(df.Department, prefix="dept")

df_with_dummies = pd.concat([df, salary_dummies, dept_dummies], axis=1)

df_with_dummies = df_with_dummies.drop(['salary', 'Department'], axis=1)

X_features = ['satisfaction_level', 'last_evaluation', 'number_project', 'average_montly_hours',
'time_spend_company', 'Work_accident', 'promotion_last_5years'] + list(salary_dummies.columns) +
list(dept_dummies.columns)

X = df_with_dummies[X_features]

y = df_with_dummies.left

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

```

```
from sklearn.linear_model import LogisticRegression  
  
model = LogisticRegression()  
  
model.fit(X_train, y_train)
```

```
from sklearn.metrics import accuracy_score
```

```
y_pred = model.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
```

```
print("Accuracy of the model:", accuracy)
```

Program 5

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

Screenshot

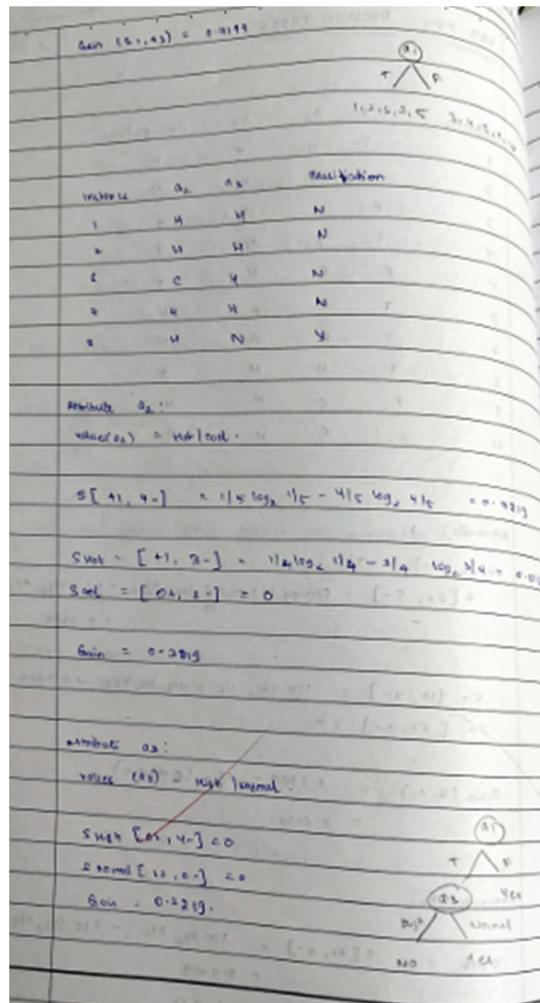
| Date _____ Page _____ | | | | |
|--------------------------|----------------|----------------|----------------|-------------|
| LAB 05: DECISION TREES | | | | |
| Instance | A ₁ | A ₂ | A ₃ | Resignation |
| 1 | T | H | H | N |
| 2 | T | H | H | N |
| 3 | F | H | H | Y |
| 4 | F | C | N | Y |
| 5 | F | C | N | Y |
| 6 | T | C | H | N |
| 7 | T | H | H | N |
| 8 | T | H | N | Y |
| 9 | F | C | N | Y |
| 10 | F | C | H | Y |

Attribute: A₁
 value (A₁) = T {F, C} condition [T, C]
 $G[A_1, S] = \text{Entropy}(S) = -\frac{1}{2} \log_2 \frac{1}{2}/\frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2}/\frac{1}{2}$
 $= 0.9209$

$S_T[A_1, Y=1] = 1/5 \log_2 1/5 + 4/5 \log_2 4/5 > 0.7312$
 $S_T[A_1, Y=0] = 0$

$\text{Gain}(S, A_1) = 0.9209 - 0.7312 = 0.1890$

Attribute: A₂:
 $S_{H1} = S[A_2, H=1] = 3/10 \log_2 3/10 + 7/10 \log_2 7/10 = 0.9405$
 $S_{H2} = S[A_2, H=2] = 0.921$
 $\text{Gain}(S, A_2) = 0.4199$



```

→ code for problem.
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

df = pd.read_csv('petrol_consumption.csv')
x = df.drop(['Petrol_Consumption'], axis=1)
y = df['Petrol_Consumption']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=42)

regressor = DecisionTreeRegressor()
regressor.fit(x_train, y_train)
y_pred = regressor.predict(x_test)

mae = mean_absolute_error(y_true=y_true, y_pred=y_pred)
mse = mean_squared_error(y_true=y_true, y_pred=y_pred)
rmse = np.sqrt(mse)

DP
mean absolute error : 46.3
mean squared error : 17452.2
Root mean squared error : 132.62457

```

Date _____
Page _____

Q1. The accuracy score for 100% is 1.0
Confusion matrix says that there is no misclassification.
All classes correctly labelled.

Q2. We need to visualize the tree, examine nodes,
Identify important features in decision tree - which
data values are predicted for each feature. In regression
tree, we assign a unique value for each instance
to aim to make a split that minimizes the variance b
target variable

Code:

```

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

from sklearn import tree

import matplotlib.pyplot as plt

```

```
iris = load_iris()
```

```
X = iris.data
```

```
y = iris.target

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

clf = DecisionTreeClassifier()

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

conf_matrix = confusion_matrix(y_test, y_pred)

print("Accuracy:", accuracy)

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

plt.figure(figsize=(12, 8))

tree.plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)

plt.show()

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score, confusion_matrix

from sklearn import tree
```

```
import matplotlib.pyplot as plt

iris = load_iris()
X = iris.data
y = iris.target

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

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

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

plt.figure(figsize=(12, 8))
tree.plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.show()

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.tree import DecisionTreeRegressor

from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```

import numpy as np # import numpy

data = pd.read_csv("petrol_consumption.csv")

X = data[['Petrol_tax', 'Average_income', 'Paved_Highways',
          'Population_Driver_licence(%)']]

y = data['Petrol_Consumption']

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

regressor = DecisionTreeRegressor()

regressor.fit(X_train, y_train)

y_pred = regressor.predict(X_test)

mae = mean_absolute_error(y_test, y_pred)

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

print("Mean Absolute Error:", mae)

print("Mean Squared Error:", mse)

print("Root Mean Squared Error:", rmse)

from sklearn.tree import plot_tree

```

```
import matplotlib.pyplot as plt

plt.figure(figsize=(15, 10))

# Assuming 'data' is your original pandas DataFrame

plot_tree(regressor, feature_names=data[['Petrol_tax', 'Average_income', 'Paved_Highways',
'Population_Driver_licence(%)']].columns, filled=True, rounded=True)

plt.show()
```

Program 6

Build KNN Classification model for a given dataset.

Screenshot

Mon 04-08-2023
LAB 00: KNN
Date _____
Page _____

→ handle the following dataset for $k=3$.
test data (35, 100) - predict target

| Index | Age | Salary | Target | Distance | Rank |
|-------|-----|--------|--------|----------|------|
| A | 18 | 50 | N | 62.81 | 1 |
| B | 33 | 55 | N | 46.53 | 2 |
| C | 24 | 30 | N | 31.95 | 3 |
| D | 41 | 60 | Y | 46.46 | 4 |
| E | 42 | 30 | Y | 31.69 | 5 |
| F | 38 | 40 | Y | 40.87 | 6 |
| X | 35 | 100 | ? | | |

$X = 3$
Rank 1 = Y
Rank 2 = N
Rank 3 = Y

The target predicted is Y.

→ code:

```
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

iris = load_iris()
X = iris.data[:, :-1]
y = iris.target

Xtrain, Xtest, Ytrain, Ytest = train_test_split(X, y, test_size=0.2,
                                              random_state=42)
```

Date _____
Page _____

```
clf = KNeighborsClassifier(n_neighbors=3)
clf.fit(Xtrain, Ytrain)
y_pred = clf.predict(Xtest)

print("Accuracy: ", accuracy_score(ytest, ypred))
print(confusion_matrix(ytest, ypred))
print(classification_report(ytest, ypred))

op: Accuracy 1.0
confusion matrix [[10 0 0]
                  [0 9 0]
                  [0 0 11]]
```

Q1: For choosing k values loop through values from 1 to 10 & choose one with max. accuracy

Q2: To normalize or standardize the range of values for different features / Needles to reduce no single feature dominates the fitting process due to its scale which can improve model performance & stability.

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```

import seaborn as sns
import matplotlib.pyplot as plt

try:
    data = pd.read_csv('/content/iris (1).csv')
except FileNotFoundError:
    print("Error: 'iris.csv' not found. Please upload the file to your Colab environment.")
    exit()

X = data.drop('species', axis=1)
y = data['species']

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

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()

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

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

from sklearn.preprocessing import StandardScaler

import seaborn as sns

import matplotlib.pyplot as plt

try:
    diabetes = pd.read_csv('diabetes.csv')
except FileNotFoundError:
    print("Error: 'diabetes.csv' not found. Please ensure the file is in the current directory.")
    exit()

X = diabetes.drop('Outcome', axis=1)
y = diabetes['Outcome']

scaler = StandardScaler()
X = scaler.fit_transform(X)

```

```

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

knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f'Accuracy: {accuracy}')

cm = confusion_matrix(y_test, y_pred)

print("Confusion Matrix:")

print(cm)

sns.heatmap(cm, annot=True, fmt="d")

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

print("Classification Report:")

print(classification_report(y_test, y_pred))

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

import seaborn as sns

import matplotlib.pyplot as plt

```

```
from sklearn.preprocessing import StandardScaler

try:
    heart = pd.read_csv('heart.csv')
except FileNotFoundError:
    print("Error: 'heart.csv' not found. Please ensure the file is in the current directory.")
    exit()

X = heart.drop('target', axis=1)
y = heart['target']

scaler = StandardScaler()
X = scaler.fit_transform(X)

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, 21):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    if accuracy > best_accuracy:
        best_accuracy = accuracy
```

```

best_k = k

print(f"Best k: {best_k} with accuracy {best_accuracy}")

knn = KNeighborsClassifier(n_neighbors=best_k)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Accuracy: {accuracy}")

cm = confusion_matrix(y_test, y_pred)

print("Confusion Matrix:")

print(cm)

sns.heatmap(cm, annot=True, fmt="d")

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

print("Classification Report:")

print(classification_report(y_test, y_pred))

import matplotlib.pyplot as plt

import seaborn as sns

```

```
from sklearn.metrics import classification_report, confusion_matrix

cm = confusion_matrix(y_test, y_pred)

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

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

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()

print(classification_report(y_test, y_pred))
```

```
# prompt: For Iris dataset

# How to choose the k value? Demonstrate using accuracy rate and error

# rate. Give theory
```

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
```

```
# Load the Iris dataset
```

```

try:
    data = pd.read_csv('/content/iris (1).csv')
except FileNotFoundError:
    print("Error: 'iris (1).csv' not found. Please upload the file to your Colab environment.")
    exit()

# Prepare the data
X = data.drop('species', axis=1)
y = data['species']

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

# Scale the data (important for KNN)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Find the optimal k value
error_rates = []
for k in range(1, 31): # Test k values from 1 to 30
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    error_rates.append(1 - accuracy_score(y_test, y_pred)) # Error rate = 1 - accuracy

```

```

# Plot error rates

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

plt.plot(range(1, 31), error_rates, color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)

plt.title('Error Rate vs. K Value')

plt.xlabel('K')

plt.ylabel('Error Rate')

plt.show()

# Theory for choosing k:

# The optimal 'k' value minimizes the error rate.

# Very small k (e.g., 1) can lead to overfitting, being too sensitive to noise.

# Very large k (e.g., 30) can lead to underfitting, smoothing out the decision boundaries too much.

# We seek a k that balances these extremes, as shown by the error rate plot.

#Select k based on the minimum error rate observed in the plot

best_k = error_rates.index(min(error_rates)) + 1 #Add 1 as the index starts from 0

# Train and evaluate the model with the best k

knn = KNeighborsClassifier(n_neighbors=best_k)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

# Evaluate the model

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()

```

```

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt

```

```

# Load data

df = pd.read_csv('/content/iris (1).csv')

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

y = df.iloc[:, -1]

```

```

# Train-test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

# Store accuracy and error rate

accuracy = []
error_rate = []

# Try k from 1 to 20

for k in range(1, 21):

    knn = KNeighborsClassifier(n_neighbors=k)

    knn.fit(X_train, y_train)

    preds = knn.predict(X_test)

    acc = accuracy_score(y_test, preds)

    accuracy.append(acc)

    error_rate.append(1 - acc)

# Plot

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

plt.plot(range(1, 21), accuracy, label='Accuracy')

plt.plot(range(1, 21), error_rate, label='Error Rate')

plt.xlabel('K Value')

plt.ylabel('Rate')

plt.title('K vs Accuracy and Error Rate')

plt.legend()

```

```
plt.show()

import pandas as pd

from sklearn.preprocessing import StandardScaler

# Load data

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

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

y = df['Outcome'] # Target

# Perform scaling

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)

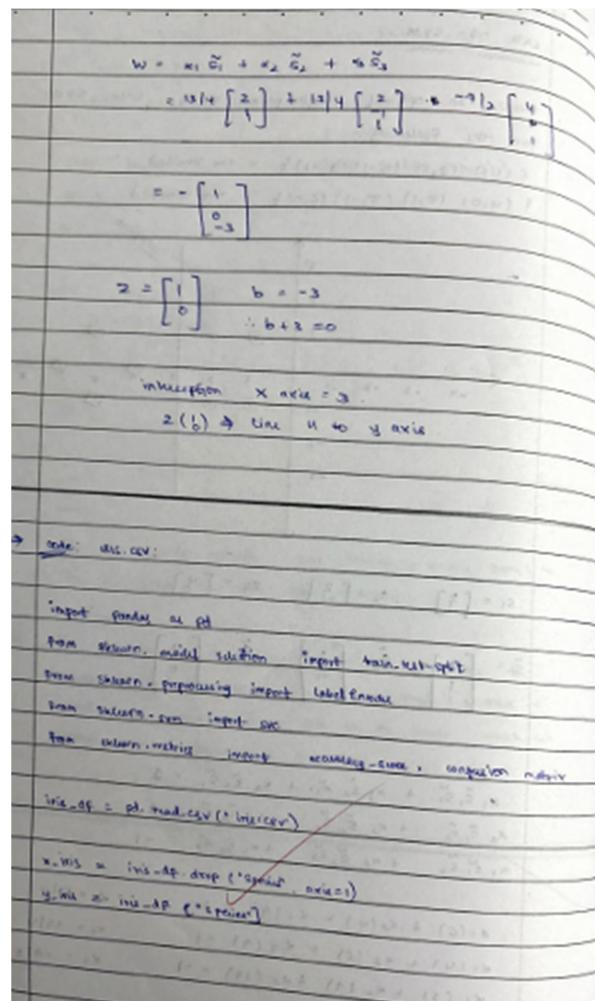
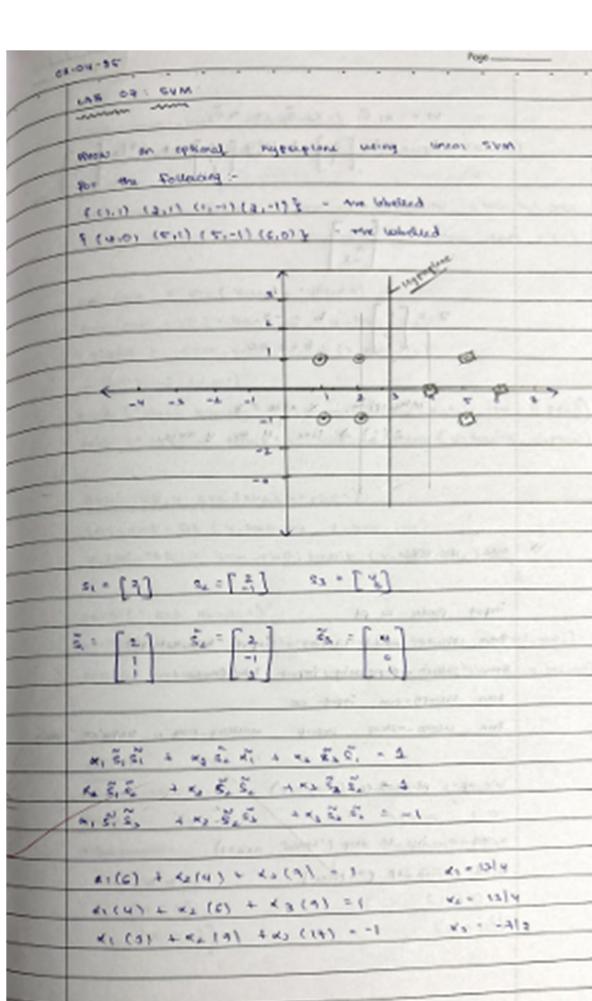
# Convert back to DataFrame (optional)

X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
```

Program 7

Build Support vector machine model for a given dataset

Screenshot



```

# Load a dataset - iris
# Using encoded = label=encoder=iris fit_transform (Y-axis)
# Using split function to split training and testing data
# X-train-iris, Y-train-iris, Y-test-iris = train-test-split
# (X-iris, Y-iris-encoded, test-size = 0.5, random_state = 42)

svm_linear = SVC(kernel='linear')
svm_linear.fit(X_train_iris, Y_train_iris)
y_pred = svm_linear.predict(X_test_iris)
print("Kernel: linear")
print("Accuracy: ", accuracy_score(Y_test_iris, y_pred))
print("Confusion matrix: ", confusion_matrix(Y_test_iris, y_pred))

# SVM-RBF = SVC(kernel='rbf')
# SVM-RBF = SVC()
y_pred_rbf = SVM_RBF.predict(X_test_iris)
print("Kernel: rbf")
print("Accuracy: ", accuracy_score(Y_test_iris, y_pred_rbf))
print("Confusion matrix: ", confusion_matrix(Y_test_iris, y_pred_rbf))

# OUTPUT:
# linear
# Kernel: linear
# Accuracy: 1.0
# Confusion matrix:
# [[10 0 0]
# [0 9 0]
# [0 0 10]]
# RBF - Kernel
# Accuracy: 1.0
# Confusion matrix:
# [[10 0 0]
# [0 9 0]
# [0 0 10]]

```

| | |
|-----|--|
| a1: | iris dataset: |
| | <ul style="list-style-type: none"> accuracy: 100% confusion matrix: Both performed equally well Reason: IRIS data is linearly separable |
| a2: | Wine Recognition dataset: |
| | <ul style="list-style-type: none"> accuracy: ~60% confusion matrix: most wines classified correctly, minor confusion (e.g. red vs white) <p>Excellent model performance</p> <ul style="list-style-type: none"> comparison: slightly lower accuracy than RLS (95%) but handles more complex data |

Code:

```

import numpy as np

import matplotlib.pyplot as plt

positive_class = np.array([[4, 1], [4, -1], [6, 0]])

negative_class = np.array([[1, 0], [0, 1], [0, -1]])

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

plt.scatter(positive_class[:, 0], positive_class[:, 1], color='red', label='Positive Class', s=100, edgecolors='black')

plt.scatter(negative_class[:, 0], negative_class[:, 1], color='blue', label='Negative Class', s=100, edgecolors='black')

all_points = np.concatenate([positive_class, negative_class])

```

```

labels = ["(4,1)", "(4,-1)", "(6,0)", "(1,0)", "(0,1)", "(0,-1)"]

for i, txt in enumerate(labels):
    plt.annotate(txt, (all_points[i][0], all_points[i][1]), textcoords="offset points", xytext=(0,5),
    ha='center', fontsize=10)

x_values = np.linspace(-1, 7, 100)
y_values = np.zeros_like(x_values)

plt.plot(x_values, y_values, color='black', linestyle='--', label='Optimal Hyperplane (y = 0)')
plt.plot(x_values, y_values + 1, color='gray', linestyle=':', label='Margin at y = 1')
plt.plot(x_values, y_values - 1, color='gray', linestyle=':', label='Margin at y = -1')

plt.title('Optimal Hyperplane for SVM (Visual Approximation)', fontsize=14)
plt.xlabel('x1')
plt.ylabel('x2')
plt.xlim(-1, 7)
plt.ylim(-2, 2)

plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)

plt.legend()

plt.grid(True)
plt.show()

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

```

```

from sklearn.metrics import accuracy_score, confusion_matrix

import seaborn as sns

import matplotlib.pyplot as plt

data = pd.read_csv('/content/iris (1) (1).csv')

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

y = data['species']

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

svm_rbf = SVC(kernel='rbf')

svm_rbf.fit(X_train, y_train)

y_pred_rbf = svm_rbf.predict(X_test)

accuracy_rbf = accuracy_score(y_test, y_pred_rbf)

cm_rbf = confusion_matrix(y_test, y_pred_rbf)

print("SVM with RBF Kernel:")

print("Accuracy:", accuracy_rbf)

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

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

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

            xticklabels=data['species'].unique(),

            yticklabels=data['species'].unique())

plt.xlabel('Predicted')

```

```

plt.ylabel('Actual')

plt.title('Confusion Matrix (RBF Kernel)')

plt.show()

svm_linear = SVC(kernel='linear')

svm_linear.fit(X_train, y_train)

y_pred_linear = svm_linear.predict(X_test)

accuracy_linear = accuracy_score(y_test, y_pred_linear)

cm_linear = confusion_matrix(y_test, y_pred_linear)

print("\nSVM with Linear Kernel:")

print("Accuracy:", accuracy_linear)

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

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

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

            xticklabels=data['species'].unique(),

            yticklabels=data['species'].unique())

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix (Linear Kernel)')

plt.show()

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

```

```
from sklearn.model_selection import train_test_split  
  
from sklearn.svm import SVC  
  
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, auc  
  
import seaborn as sns  
  
from sklearn.preprocessing import label_binarize  
  
from sklearn.multiclass import OneVsRestClassifier  
  
  
data = pd.read_csv('/content/letter-recognition.csv') # Replace with the correct path if necessary  
  
  
X = data.drop('letter', axis=1)  
y = data['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='rbf', probability=True) # probability=True is needed for ROC curve  
  
svm_classifier.fit(X_train, y_train)  
  
y_pred = svm_classifier.predict(X_test)  
  
  
accuracy = accuracy_score(y_test, y_pred)  
  
cm = confusion_matrix(y_test, y_pred)  
  
print("SVM Classifier:")  
  
print("Accuracy:", accuracy)  
  
print("Confusion Matrix:\n", cm)  
  
  
plt.figure(figsize=(10, 8))
```

```

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=np.unique(y),
            yticklabels=np.unique(y))

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

```

```

y_test_bin = label_binarize(y_test, classes=np.unique(y))

n_classes = y_test_bin.shape[1]

classifier = OneVsRestClassifier(SVC(kernel='rbf', probability=True))

classifier.fit(X_train, y_train)

y_score = classifier.predict_proba(X_test)

```

```

fpr = dict()

tpr = dict()

roc_auc = dict()

for i in range(n_classes):

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

    roc_auc[i] = auc(fpr[i], tpr[i])

```

```

fpr["micro"], tpr["micro"], _ = roc_curve(y_test_bin.ravel(), y_score.ravel())

roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

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

plt.plot(fpr["micro"], tpr["micro"],
         label='micro-average ROC curve (area = {0:0.2f})'.format(roc_auc["micro"]))

```

```
".format(roc_auc["micro"]))  
  
plt.plot([0, 1], [0, 1], 'k--')  
  
plt.xlim([0.0, 1.0])  
  
plt.ylim([0.0, 1.05])  
  
plt.xlabel('False Positive Rate')  
  
plt.ylabel('True Positive Rate')  
  
plt.title('Micro-averaged ROC Curve')  
  
plt.legend(loc="lower right")  
  
plt.show()  
  
print(f'Micro-averaged AUC: {roc_auc['micro']}')
```

Program 8

Implement Random forest ensemble method on a given dataset.

Screenshot

LAB 08: RANDOM FOREST :-

- * Decision Tree
 - 1. Single tree
 - 2. Use boosted
 - 3. Fast to train
 - 4. single prediction
- * Random Forest
 - multiple trees
 - more accurate
 - Slow to train
 - majority vote
- * Parameters of Random Forest:
 - in estimators
 - estimators
 - max_depth
 - min_samples_split
 - min_samples_leaf
 - oob_score
 - n_estimators
 - bootstrap
 - max_features
 - random_state
- * Algorithm:
 - (1) Input :- dataset \rightarrow with numbers \times 2 labels \downarrow
number of trees N
 - (2) For each tree ($i = 1 \dots N$)
 - Randomly select a bootstrap sample from the dataset
 - Train a decision tree on this sample
 - When splitting nodes:
 - Grow one more fully, or until a stopping condition is met
 - etc.

Code :-

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

iris = pd.read_csv('iris.csv')
X = iris[['petal.length', 'sep.width', 'petal.length', 'petal.width']]
y = iris['species']

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

estimators = [10, 50, 100, 200, 500]
scores = []

for n in est:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    print(f'For n = {n} in estimators')
    print(f'Accuracy = {accuracy_score(y_test, rf.predict(X_test))}')
    scores.append(accuracy_score(y_test, rf.predict(X_test)))
  
```

Code:

```
import pandas as pd
```

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

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import accuracy_score
```

```

import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('/content/iris (1).csv')

# Prepare features and target
X = df.drop(columns=['species']) # Assuming 'species' is the target column
y = df['species']

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Build Random Forest with default n_estimators (10)
rf_default = RandomForestClassifier(n_estimators=10, random_state=42)
rf_default.fit(X_train, y_train)
y_pred_default = rf_default.predict(X_test)

# Measure accuracy
default_score = accuracy_score(y_test, y_pred_default)
print(f"Default RF accuracy (n_estimators=10): {default_score:.4f}")

# Fine-tune the number of trees
scores = []
for n_estimators in range(1, 101):
    rf = RandomForestClassifier(n_estimators=n_estimators, random_state=42)
    rf.fit(X_train, y_train)
    score = accuracy_score(y_test, rf.predict(X_test))
    scores.append(score)

```

```

for n in n_range:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    score = accuracy_score(y_test, y_pred)
    scores.append(score)

# Find the best score and number of trees
best_score = max(scores)
best_n = n_range[scores.index(best_score)]
print(f'Best RF accuracy: {best_score:.4f} with n_estimators={best_n}')

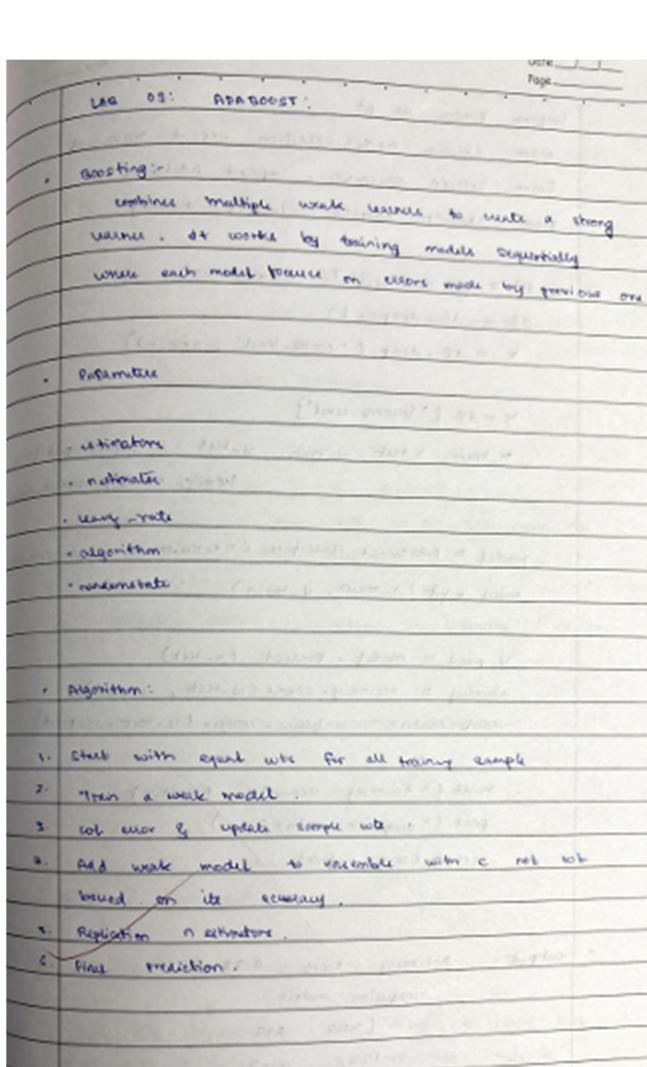
# Optional: Plot accuracy vs number of estimators
plt.figure(figsize=(10, 6))
plt.plot(n_range, scores, marker='o')
plt.title('Random Forest Accuracy vs Number of Trees')
plt.xlabel('Number of Trees (n_estimators)')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()

```

Program 9

Implement Boosting ensemble method on a given dataset.

Screenshot



```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoost
from sklearn.metrics import accuracy_score
df = pd.read_csv('income.csv')
df = df.dropna()
X = df.drop(['income-level'], axis=1)
y = df['income-level']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = AdaBoostClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
confusion_matrix = confusion_matrix(y_test, y_pred)
print("Accuracy Score: ", accuracy)
print("Confusion Matrix: ")
print(confusion_matrix)
output: Accuracy Score: 0.8238
confusion matrix:
[[1112 247]
 [1326 1019]]
```

Code:

```
import pandas as pd
```

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

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

```

from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

# Load dataset
df = pd.read_csv("/content/income.csv")

# Drop rows with missing values
df.dropna(inplace=True)

# Encode categorical columns
label_encoders = {}

for column in df.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Separate features and target
X = df.drop(columns=['income_level'], errors='ignore', axis=1)
y = df['income_level']

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# AdaBoost with 10 estimators
model_10 = AdaBoostClassifier(n_estimators=10, random_state=42)
model_10.fit(X_train, y_train)
y_pred_10 = model_10.predict(X_test)
score_10 = accuracy_score(y_test, y_pred_10)

```

```

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

# Fine-tune number of estimators

best_score = 0

best_n = 0

estimators_range = list(range(10, 201, 10))

scores = []

for n in estimators_range:

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

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    score = accuracy_score(y_test, y_pred)

    scores.append(score)

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

    if score > best_score:

        best_score = score

        best_n = n

print(f'\nBest Accuracy: {best_score:.4f} using {best_n} estimators')

# Plot accuracy vs number of estimators

plt.figure(figsize=(7, 4))

plt.plot(estimators_range, scores, marker='o', linestyle='-', color='blue')

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

plt.xlabel("Number of Estimators (Trees)")

plt.ylabel("Accuracy")

plt.grid(True)

```

```
plt.xticks(estimators_range)
```

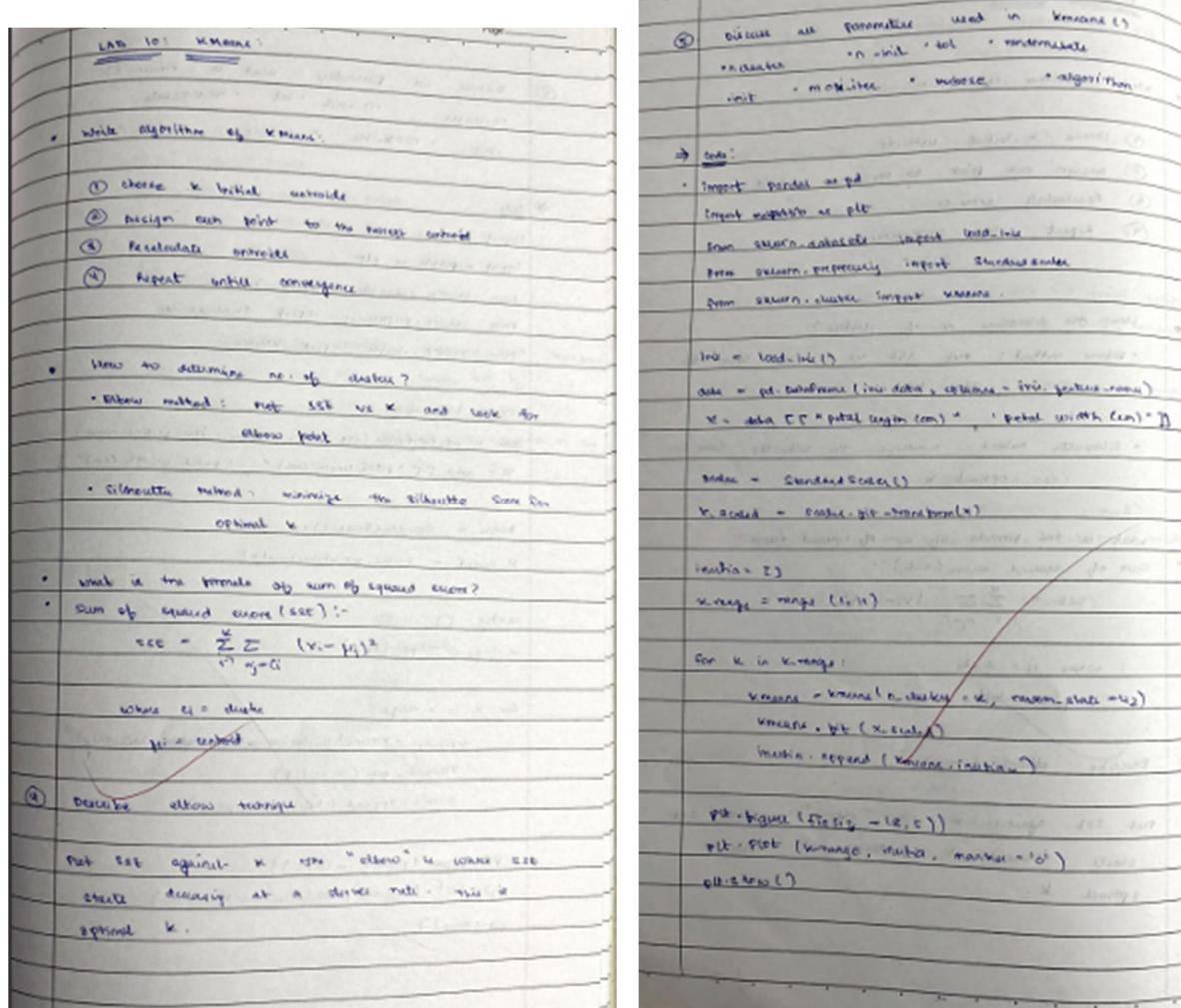
```
plt.tight_layout()
```

```
plt.show()
```

Program 10

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

Screenshot



Code:

```
import pandas as pd
```

```
import numpy as np
```

```

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans

from sklearn.metrics import accuracy_score

from scipy.stats import mode

import matplotlib.pyplot as plt

# Step 1: Generate sample data and save to CSV

np.random.seed(42)

names = [f"Person_{i}" for i in range(50)]

ages = np.random.randint(20, 60, 50)

income = np.random.randint(30000, 120000, 50)

df = pd.DataFrame({'Name': names, 'Age': ages, 'Income': income})

df.to_csv("income.csv", index=False)

# Step 2: Load the data

data = pd.read_csv("income.csv")

# Drop 'Name' and extract features

X = data[['Age', 'Income']]

# Step 3: Split the data

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

```

```

# Step 4: Perform scaling

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

# Step 5: Plot SSE vs number of clusters (Elbow method)

sse = []

k_range = range(1, 11)

for k in k_range:

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

    kmeans.fit(X_train_scaled)

    sse.append(kmeans.inertia_)

plt.figure(figsize=(8, 4))

plt.plot(k_range, sse, marker='o')

plt.xlabel('Number of clusters')

plt.ylabel('SSE (Inertia)')

plt.title('Elbow Method For Optimal k')

plt.grid(True)

plt.show()

# Step 6: Choose optimal number of clusters (say 3) and fit model

optimal_k = 3

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

```

```

kmeans.fit(X_train_scaled)

# Predict on test data
predictions = kmeans.predict(X_test_scaled)

# Note: There's no ground truth labels, but for demonstration,
# we can try assigning true clusters (via KMeans on full data)
# and see if predicted clusters align

# Fit on full data to assign pseudo-labels
full_kmeans = KMeans(n_clusters=optimal_k, random_state=42)
true_clusters = full_kmeans.fit_predict(scaler.fit_transform(X))

# Align predicted clusters using majority voting (only for demonstration)
# Match predicted labels to closest true labels

def map_clusters(true_labels, pred_labels):
    labels = np.zeros_like(pred_labels)
    for i in range(optimal_k):
        mask = (pred_labels == i)
        if np.sum(mask) == 0:
            continue
        labels[mask] = mode(true_labels[mask])[0]
    return labels

```

```

mapped_preds = map_clusters(true_clusters[X_test.index], predictions)

accuracy = accuracy_score(true_clusters[X_test.index], mapped_preds)

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

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load_iris

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette_score

# Step 1: Load Iris dataset

iris = load_iris()

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

df['target'] = iris.target

# Keep only petal length and petal width

X = df[['petal length (cm)', 'petal width (cm)']].values

# Step 2: Check impact of scaling

# Try without scaling

sse_unscaled = []

for k in range(1, 11):

```

```

kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(X)

sse_unscaled.append(kmeans.inertia_)

# Now scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

sse_scaled = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    sse_scaled.append(kmeans.inertia_)

# Step 3: Plot Elbow Comparison (Scaled vs Unscaled)
plt.figure(figsize=(10, 5))

plt.plot(range(1, 11), sse_unscaled, marker='o', label='Unscaled')
plt.plot(range(1, 11), sse_scaled, marker='s', label='Scaled')
plt.title('Elbow Method (Petal Features Only)')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('SSE (Inertia)')
plt.legend()
plt.grid(True)

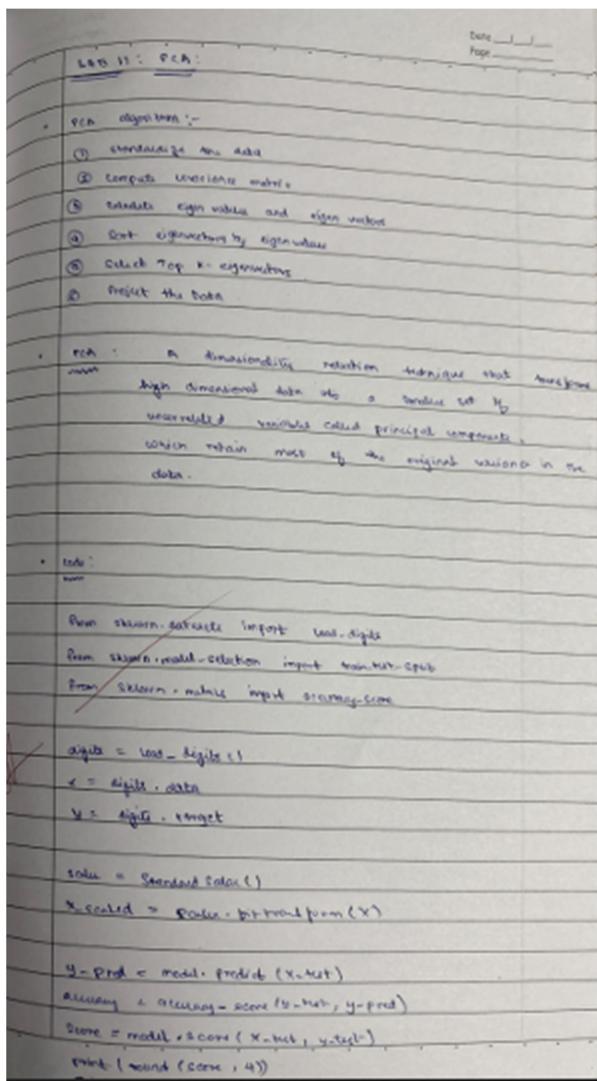
```

```
plt.show()
```

Program 11

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

Screenshot



Code:

```
import pandas as pd

from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline
```

```

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

from sklearn.decomposition import PCA

from sklearn.metrics import accuracy_score


# 1. Load data

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


# 2. Label-encode binary text columns

le = LabelEncoder()

for col in ["Sex", "ExerciseAngina"]:

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


# 3. Separate features and target

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

y = df["HeartDisease"]


# 4. Build preprocessing pipeline:

# - One-hot for multi-category columns (using sparse_output=False)

# - passthrough the rest

# - then scale everything

cat_cols = ["ChestPainType", "RestingECG", "ST_Slope"]

```

```
preprocessor = Pipeline([
    ("onehot", ColumnTransformer([
        ("ohe", OneHotEncoder(sparse_output=False, drop="first"), cat_cols)
    ], remainder="passthrough")),
    ("scaler", StandardScaler())
])
```

```
# 5. Apply preprocessing
```

```
X_proc = preprocessor.fit_transform(X)
```

```
# 6. Train/test split
```

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

```
# 7. Define models
```

```
models = {
    "SVM": SVC(random_state=42),
    "LogisticRegression": LogisticRegression(max_iter=1000, random_state=42),
    "RandomForest": RandomForestClassifier(random_state=42)
}
```

```
# 8. Train & evaluate before PCA
```

```
print("==== Accuracies BEFORE PCA ===")
```

```

scores_before = {}

for name, clf in models.items():

    clf.fit(X_train, y_train)

    preds = clf.predict(X_test)

    acc = accuracy_score(y_test, preds)

    scores_before[name] = acc

    print(f'{name:17s}: {acc:.4f}')


# 9. Apply PCA (retain 95% variance)

pca = PCA(n_components=0.95, random_state=42)

X_train_pca = pca.fit_transform(X_train)

X_test_pca = pca.transform(X_test)

print(f'\nPCA retained {pca.n_components_} components,'

      f'explained variance = {pca.explained_variance_ratio_.sum():.4f}\n')

# 10. Train & evaluate after PCA

print("==== Accuracies AFTER PCA ====")

scores_after = {}

for name, clf in models.items():

    clf.fit(X_train_pca, y_train)

    preds = clf.predict(X_test_pca)

    acc = accuracy_score(y_test, preds)

    scores_after[name] = acc

    print(f'{name:17s}: {acc:.4f}')

```