VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

ARJUN PRABHAKARAN (1BM22CS053)

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 **ARJUN PRABHAKARAN (1BM22CS053),** 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.

Sarala D V Assistant Professor Department of CSE, BMSCE	Dr. Kavitha Sooda Professor & HOD Department of CSE, BMSCE

Index

Github Link: https://github.com/wombat-42/ML_LAB.git

Program 1

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

Screenshot

8/3/2025
Monday

LAB 1 - Data processing

import pandas as pd
import numpy as np

i) To load .csv file to dataframe
→ df = pd.read_csv("housing.csv")

ii) To display information of all columns
⇒ print(df)

iii) To display statistical information of all columns
⇒ df.describe()

iv) To display unique count of values for "ocean proximity" column.

v) To display which attributes (columns) in data set have missing values count greater than zero.
⇒ print(df.isnull().sum())

```
for i in df:
    if i.isnull().sum() > 0:
        print(i)
            or
missing_values = df.isnull().sum()
print(missing_values[missing_values > 0])
```

... have missing values? How did you handle them

ans. The columns did not have missing values, but I added them.

→ Diabetes.csv
Columns : TG, HDL
I handled them by using SimpleImputer method of Sklearn.
In the TG column, missing values were filled by ~~using~~ finding the mean of all the values present in the column TG.
For HDL also we found mean value of all values in the HDL column & filled it inplace of missing values.

→ Adult.csv :
Columns : capital-gain, hours-per-w...
Missing values in capital-gain was replaced by finding the mean of all values in capital-gain
& for hours-per-week, median ...

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

ans: Diabetes.csv
Categorical data : gender
The gender column was encoded by using ordinal encoder, which replaced M with 0.0 & F with 1.0

Adult.csv:
Categorical data : working-class
The workclass column was encoded by using OneHotEncoder, which ~~substitutes a~~ creates a column with all values of work class, places 1 in that column if the row belongs to that category

3. What ... scaling and standardization? When would you use one over the other?

Min max scaling is used to transform features to be on a similar scale.

$x\_new = (x - x\_min) / (x\_max - x\_min)$

Standardization : transformation of features by subtracting from mean & dividing by standard deviation

$x\_new = (x - mean) / std.$

Minmax scaling is used for uniform data. It is affected by outliers.
Standardisation is used when the data follows Gaussian distribution. It is not affected by outliers.

10/3/25

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(df.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(df.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

10/3/2020
Monday

LAB2

- Geographical features of the ~~ee~~
  - → longitude
  - → latitude
  - → ocean proximity
- Plot a graph to show features correlation with housing price. Which feature correlates to the maximum

ans: Median_income correlates ~~the most~~ to the maximum

- List the feature that could be combined to improve correlation & plot again to see if correlation has improved

ans Combine → total_rooms & total_bedrooms
population & households.

- ~~List feature~~ that could be combined to improve correlation ~~& plot~~
- List features that need to be cleaned & demonstrate the process of cleaning

ans: Total_rooms has some missing values & needs some cleaning.
You can remove the ~~null~~ null values by filling the null values with median no of bedrooms.

- Is there any categorical data that needs to be converted to numerical?

Yes, ocean_proximity.

- Discuss importance of feature scaling.

It ensures all features are on a similar scale, allowing for fair comparison between them & leading to improved model performance by preventing features with larger range from dominating the learning process.

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

17/3/2025
Monday

LAB 3

Q.

x_i (week)	y_i (sales in thousands)
1	2
2	4
3	5
4	9

Ans Matrix form

$y^T = [2 \quad 4 \quad 5 \quad 9]$

$x^T = [1 \quad 2 \quad 3 \quad 4]$

$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}$

$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$

$X^T X = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$

$(X^T X)^{-1} = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$

$(X^T X)^{-1} X^T = \begin{bmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{bmatrix}$

$\beta = ((X^T X)^{-1} X^T) y$

$\beta = \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix}$

$y = \beta_0 + \beta_1 x$

$\boxed{y = -0.5 + 2.2x}$

Normal form

$\beta_1 = \dfrac{(\overline{xy}) - (\bar{x})(\bar{y})}{(\overline{x^2}) - (\bar{x})^2}$

$\beta_0 = \bar{y} - \beta_1 \times \bar{x}$

x	y	x^2	xy
1	2	1	2
2	4	4	8
3	5	9	15
4	9	16	36

$\bar{x} = \frac{10}{4} = 2.5$ | $\overline{x^2} = 7.5$

$\bar{y} = \frac{20}{4} = 5$ | $\overline{xy} = 15.25$

$(\bar{x})^2 = 6.25$

$\beta_1 = \dfrac{(\overline{xy}) - (\bar{x})(\bar{y})}{(\overline{x^2}) - (\bar{x})^2}$

$= \dfrac{15.25 - (2.5)(5)}{7.5 - 6.25}$

$\beta_1 = \dfrac{2.75}{1.25}$

$\boxed{\beta_1 = 2.2}$

$\beta_0 = (\bar{y}) - \beta_1 \times \bar{x}$

$= 5 - 2.2(2.5)$

$\boxed{\beta_0 = -0.5}$

$y = \beta_0 + \beta_1 x_1$

$\boxed{y = -0.5 + 2.2x}$

Q

Diameter in inch (x)	Price in \$ (y)
8	10
10	13
12	16

Matrix method:

$X = \begin{bmatrix} 1 & 8 \\ 1 & 10 \\ 1 & 12 \end{bmatrix}, \quad y = \begin{bmatrix} 10 \\ 13 \\ 16 \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$

$\beta = ((X^T X)^{-1} X^T) y = \begin{bmatrix} -2 \\ 1.5 \end{bmatrix} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$

$\beta_0 = -2$

$\beta_1 = 1.5$

$y = \beta_0 + \beta_1 x$

$y = -2 + 1.5x$

$y = ?$ when $x = 20$

$y = -2 + 1.5(20)$

$= -2 + 30$

$\boxed{y = 28}$

Normal method

x	y	xy	x^2
8	10	80	64
10	13	130	100
12	16	192	144

$\bar{x} = 10$ | $\overline{x^2} = 308/3$

$\bar{y} = 13$ | $\overline{xy} = 402/3 = 134$

$\beta_1 = \dfrac{\overline{xy} - (\bar{x})(\bar{y})}{(\overline{x^2}) - (\bar{x})^2} = \dfrac{134 - 130}{\frac{308}{3} - \frac{300}{3}}$

$= \dfrac{3(4)}{8}$

$\beta_1 = 1.5$

$\beta_0 = (\bar{y}) - \beta_1(\bar{x})$

$= 13 - 1.5(10)$

$\boxed{\beta_0 = -2}$

$\therefore y = \beta_0 + \beta_1 x$

$y = -2 + 1.5x$

Q. Did you perform any data processing steps?

Yes, Handled missing values by filling them with the column mean. Also applied label encoding to categorical (like = "state") & scaled numerical features for 1000-companies.csv to normalize the data.

Q. Did you visualize the regression line for canada-per-capita.csv?

Yes the regression line was plotted. It shows a strong linear r/p b/w year & per capita income, meaning that as the year increases per capita also rises.

Q. Predict salary for 12 years, 10 test score, 10 interview score

~~The predicted~~

Q. Did you encode categorical variables for 1000-companies.csv?

Yes.

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"""

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
```

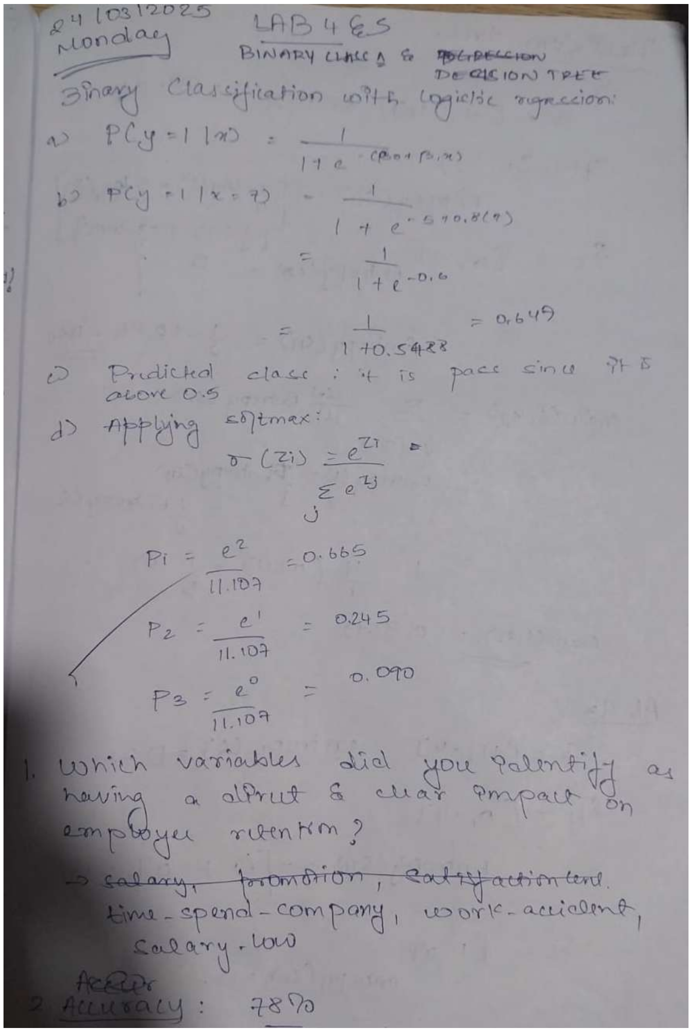
```
# Convert 'experience' column to numeric, replacing non-numeric with NaN
df['experience'] = pd.to_numeric(df['experience'], errors='coerce')

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



24/03/2025
Monday

LAB 4 & 5

BINARY CLASS & ~~REGRESSION~~ DECISION TREE

Binary Classification with logistic regression:

a) $P(y=1|x) = \frac{1}{1+e^{-(\beta_0+\beta_1 x)}}$

b) $P(y=1|x=7) = \frac{1}{1+e^{-5+0.8(7)}}$

$= \frac{1}{1+e^{-0.6}}$

$= \frac{1}{1+0.5488} = 0.649$

c) Predicted class : it is pass since it is above 0.5

d) Applying softmax:

$\sigma(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$

$P_1 = \frac{e^2}{11.107} = 0.665$

$P_2 = \frac{e^1}{11.107} = 0.245$

$P_3 = \frac{e^0}{11.107} = 0.090$

1. Which variables did you identify as having a direct & clear impact on employee retention?

→ ~~salary, promotion, satisfaction level.~~ time-spend-company, work-accident, salary.low

2. ~~Accur~~ Accuracy : 78%

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

Decision tree:

a_2

$S = [+1, -4]$ Entropy(S) = 0.72

$S_H = [+1, -3]$

Entropy$(S_H) = -\left[\frac{1}{4}\log_2\left(\frac{1}{4}\right) + \frac{3}{4}\log_2\left(\frac{3}{4}\right)\right]$

$= -\left[\frac{1}{4}(-2) + \frac{3}{4}(0.415)\right]$

$S_C = [0, -1]$ Entropy$(S_C) = 0$

Entropy$(S_H) = \frac{1}{2} + 0.311 = 0.811$

Gain$(S, a_2) = \sum_{v \in [T, F]} \frac{|S_v|}{|S|}$ Entropy(S_v)

$=$ Entropy(S) $- \frac{4}{5}$ Entropy$(S_H) - \frac{1}{5}$ Entropy(S_C)

$= 1 - \frac{4}{5}(0.811) - \frac{1}{5}(0)$

Gain$(S, a_2) = 0.3512$

At a_3

$S = [+1, -4]$ entropy(S) = 0.72

$S_H = [0, -4]$

Entropy$(S_H) = -[0 + 0] = 0$

$S_C = [1, 0]$

entropy$(S_C) = 0$

Gain$(S, a_3) = 1$

Branching at a_3

Entropy(S) $= -\left(\frac{1}{4}\log_2\left(\frac{1}{5}\right) + \frac{4}{5}\log_2\left(\frac{1}{2}\right)\right)$

$= 0.72$

1\. 1'

Accuracy for iris dataset => 98%

Confusion matrix ->

10	0	0
0	9	0
0	0	11

The confusion matrix shows no miscalculation.

2\. For petrol_consumption.csv

regression tree predicts based on features.

It predicts for continuous values.

MAE : 90.60

MSE : 16851.60

RMSE : 129.81

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)
```

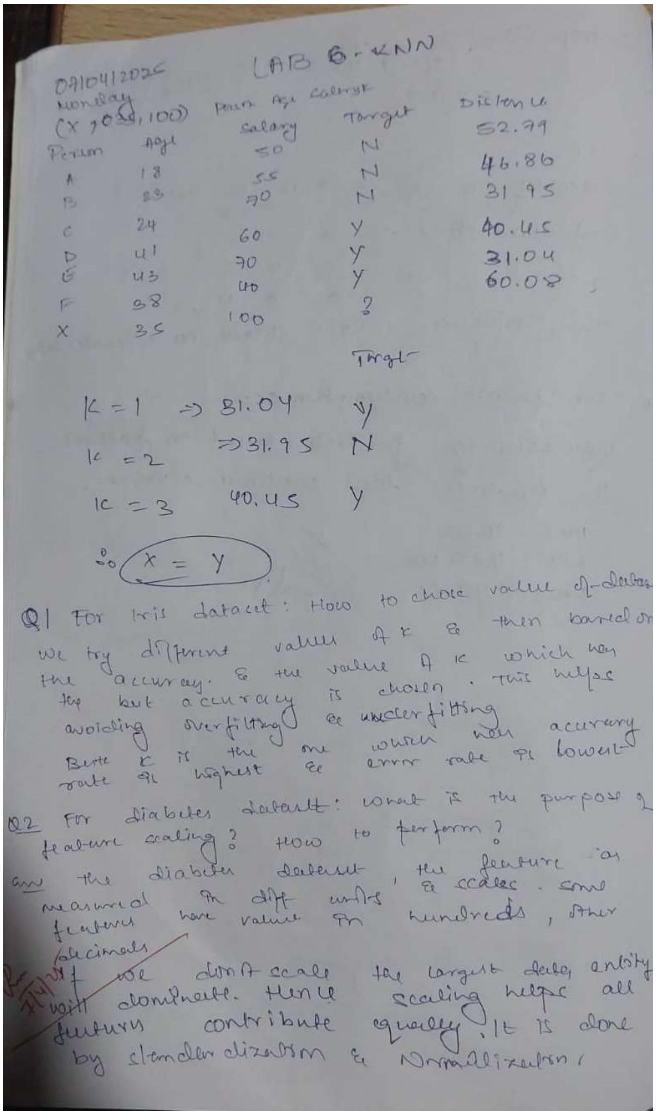
```
print("Confusion Matrix:\n", conf_matrix)



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



07/04/2025 Monday

LAB 6 - KNN

(X, 35, 100) Person Age Salary Target

Person	Age	Salary	Target	Distance
A	18	50	N	52.79
B	23	55	N	46.86
C	24	70	N	31.95
D	41	60	Y	40.45
E	43	70	Y	31.04
F	38	40	Y	60.08
X	35	100	?	

Target

K = 1 => 31.04 Y

K = 2 => 31.95 N

K = 3 40.45 Y

∴ X = Y

Q1 For Iris dataset: How to chose value of K

We try different values of K & then based on the accuracy & the value of K which has the best accuracy is chosen. This helps avoiding overfitting & underfitting. Best K is the one which has accuracy rate is highest & error rate is lowest.

Q2 For diabetes dataset: what is the purpose of feature scaling? How to perform?

Ans The diabetes dataset, the features are measured in diff units & scales. Some features have values in hundreds, other decimals. If we don't scale the largest data entity will dominate. Hence scaling helps all features contribute equally. It is done by standardization & Normalization.

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)
```

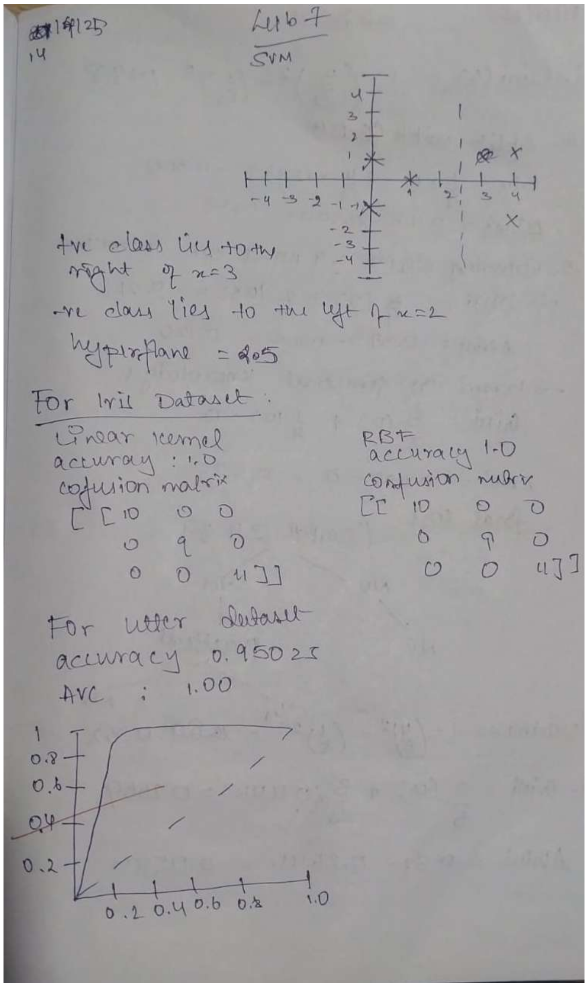
```
# Evaluate the model
print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

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



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))
```

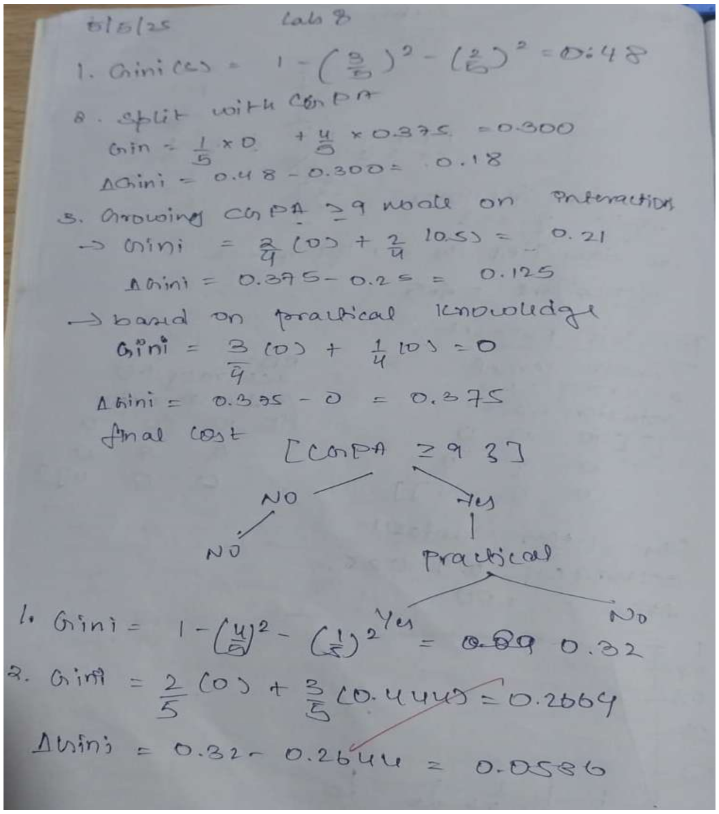
```
sns.heatmap(cm_linear, annot=True, fmt='d', cmap='Blues',
            xticklabels=data['species'].unique(),
            yticklabels=data['species'].unique())
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix (Linear Kernel)')
plt.show()
import numpy as np
import matplotlib.pyplot 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"]))
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



5/5/25 Lab 8

1. Gini (S) = $1 - (\frac{3}{5})^2 - (\frac{2}{5})^2 = 0.48$

2. split with CGPA

Gini = $\frac{1}{5} \times 0 + \frac{4}{5} \times 0.375 = 0.300$

ΔGini = 0.48 − 0.300 = 0.18

3. Growing CGPA ≥ 9 node on interaction

→ Gini = $\frac{2}{4}(0) + \frac{2}{4}(0.5) = 0.25$

ΔGini = 0.375 − 0.25 = 0.125

→ based on practical knowledge

Gini = $\frac{3}{4}(0) + \frac{1}{4}(0) = 0$

ΔGini = 0.375 − 0 = 0.375

final cost

[CGPA ≥ 9 ?]

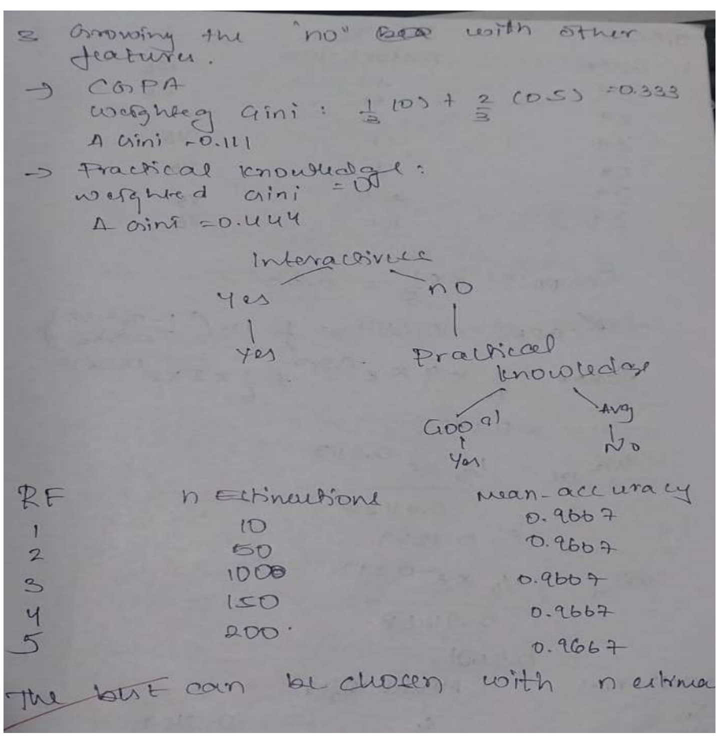
NO → NO

Yes → Practical → Yes / No

1. Gini = $1 - (\frac{4}{5})^2 - (\frac{1}{5})^2 = \cancel{0.89}\ 0.32$

2. Gini = $\frac{2}{5}(0) + \frac{3}{5}(0.444) = 0.2664$

ΔGini = 0.32 − 0.2644 = 0.0556



3. Growing the "no" ~~node~~ with other features.

→ CGPA

weighted Gini : $\frac{1}{3}(0) + \frac{2}{3}(0.5) = 0.333$

ΔGini = 0.111

→ Practical knowledge :

weighted Gini = 0

ΔGini = 0.444

Interactive

Yes → Yes

no → Practical knowledge → Good → Yes; Avg → No

RF	n Estimators	mean-accuracy
1	10	0.9667
2	50	0.9667
3	100	0.9667
4	150	0.9667
5	200	0.9667

The best can be chosen with n estima

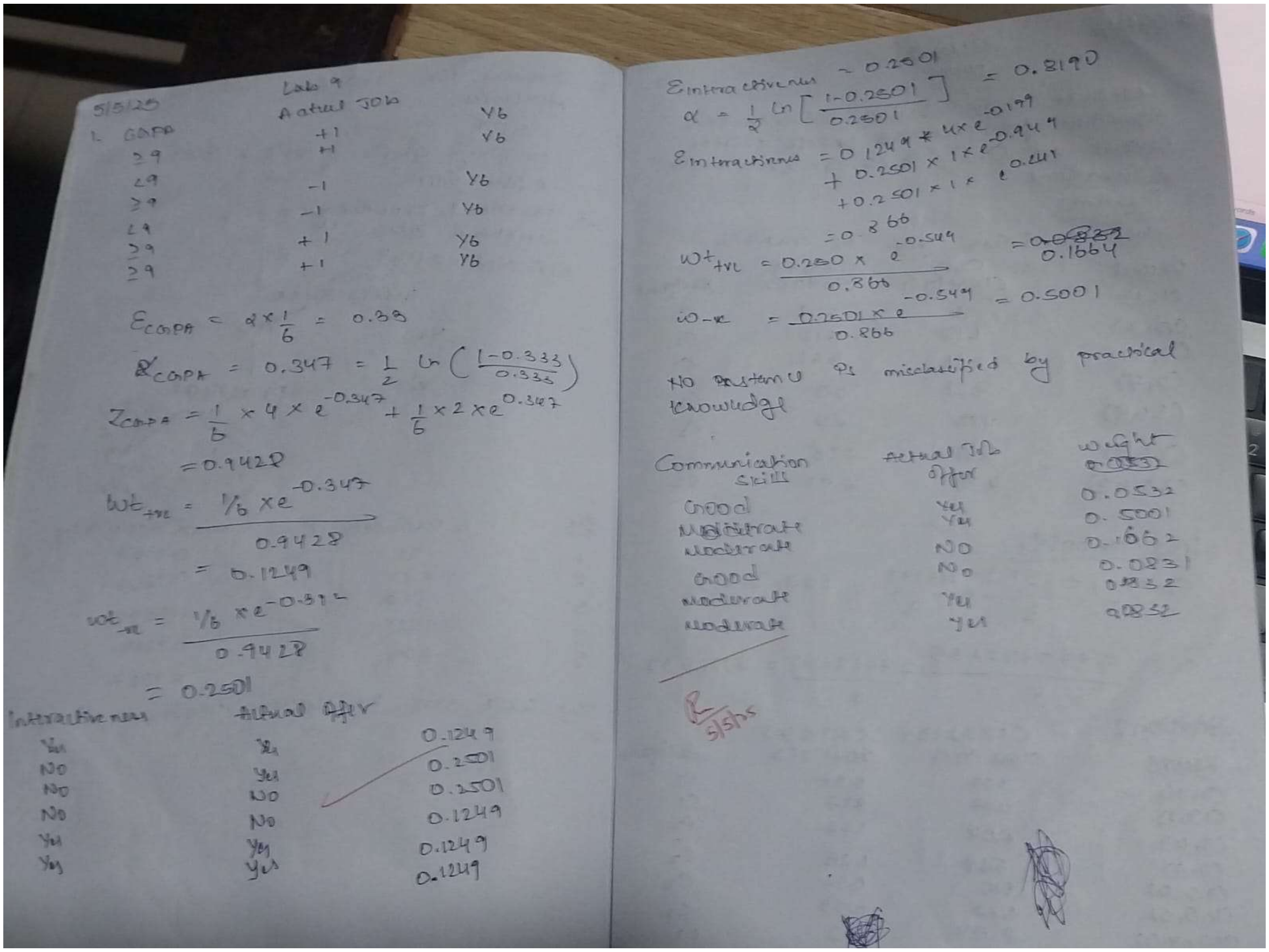
Code

```
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 = []
n_range = range(1, 101)


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


5/5/25 Lab 9

CGPA	Actual Job	
≥9	+1	Y6
<9	+1	Y6
≥9	-1	Y6
<9	-1	Y6
≥9	+1	Y6
≥9	+1	Y6

$\varepsilon_{CGPA} = 2 \times \frac{1}{6} = 0.33$

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

$Z_{CGPA} = \frac{1}{6} \times 4 \times e^{-0.347} + \frac{1}{6} \times 2 \times e^{0.347}$

$= 0.9428$

$wt_{+ve} = \frac{1/6 \times e^{-0.347}}{0.9428} = 0.1249$

$wt_{-ve} = \frac{1/6 \times e^{-0.312}}{0.9428} = 0.2501$

Interactiveness	Actual Offer	
Yes	Yes	0.1249
No	Yes	0.2501
No	No	0.2501
No	No	0.1249
Yes	Yes	0.1249
Yes	Yes	0.1249

$\varepsilon_{Interactiveness} = 0.2501$

$\alpha = \frac{1}{2} \ln\left[\frac{1-0.2501}{0.2501}\right] = 0.8190$

$Z_{Interactiveness} = 0.1249 \times 4 \times e^{-0.199} + 0.2501 \times 1 \times e^{-0.944} + 0.2501 \times 1 \times e^{0.441}$

$= 0.866$

$wt_{+ve} = \frac{0.250 \times e^{-0.544}}{0.866} = 0.1664$

$w_{-ve} = \frac{0.2501 \times e^{-0.544}}{0.866} = 0.5001$

No problem is misclassified by practical knowledge

Communication Skills	Actual Job Offer	weight
Good	Yes	0.0532
Moderate	Yes	0.5001
Moderate	No	0.1662
Good	No	0.0831
Moderate	Yes	0.0832
Moderate	Yes	0.0832

5/5/25

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

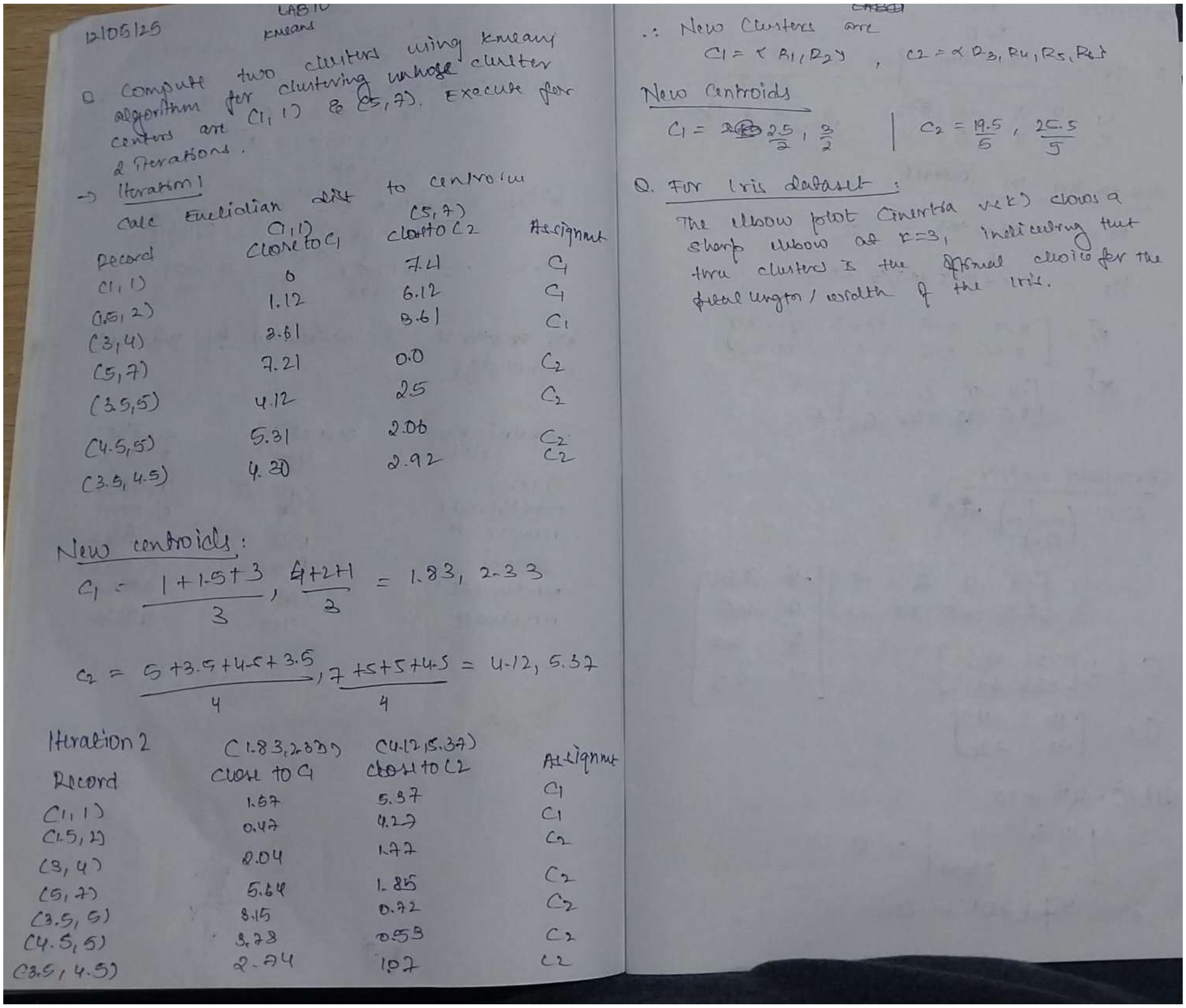
from sklearn.tree import DecisionTreeClassifier

```
# 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



12/05/25 LAB 10 Kmeans

Q. Compute two clusters using kmeans algorithm for clustering whose cluster centers are (1, 1) & (5, 7). Execute for 2 iterations.

→ Iteration 1

Calc Euclidian dist to centroids

Record	(1,1) Close to C1	(5,7) Close to C2	Assignment
(1, 1)	0	7.21	C1
(1.5, 2)	1.12	6.12	C1
(3, 4)	3.61	3.61	C1
(5, 7)	7.21	0.0	C2
(3.5, 5)	4.72	2.5	C2
(4.5, 5)	5.31	2.06	C2
(3.5, 4.5)	4.30	2.92	C2

New centroids:

$C_1 = \frac{1+1.5+3}{3}, \frac{4+2+1}{3} = 1.83, 2.33$

$C_2 = \frac{5+3.5+4.5+3.5}{4}, \frac{7+5+5+4.5}{4} = 4.12, 5.37$

Iteration 2

Record	(1.83, 2.33) Close to C1	(4.12, 5.37) Close to C2	Assignment
(1, 1)	1.57	5.37	C1
(1.5, 2)	0.47	4.27	C1
(3, 4)	2.04	1.72	C2
(5, 7)	5.64	1.85	C2
(3.5, 5)	3.15	0.72	C2
(4.5, 5)	3.73	0.53	C2
(3.5, 4.5)	2.74	1.07	C2

∴ New Clusters are

C1 = {R1, R2}, C2 = {R3, R4, R5, R6}

New Centroids

$C_1 = \frac{2.5}{2}, \frac{3}{2}$ | $C_2 = \frac{19.5}{5}, \frac{26.5}{5}$

Q. For Iris dataset:

The elbow plot (inertia vs k) shows a sharp elbow at k=3, indicating that three clusters is the optimal choice for the petal length / width of the Iris.

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
```

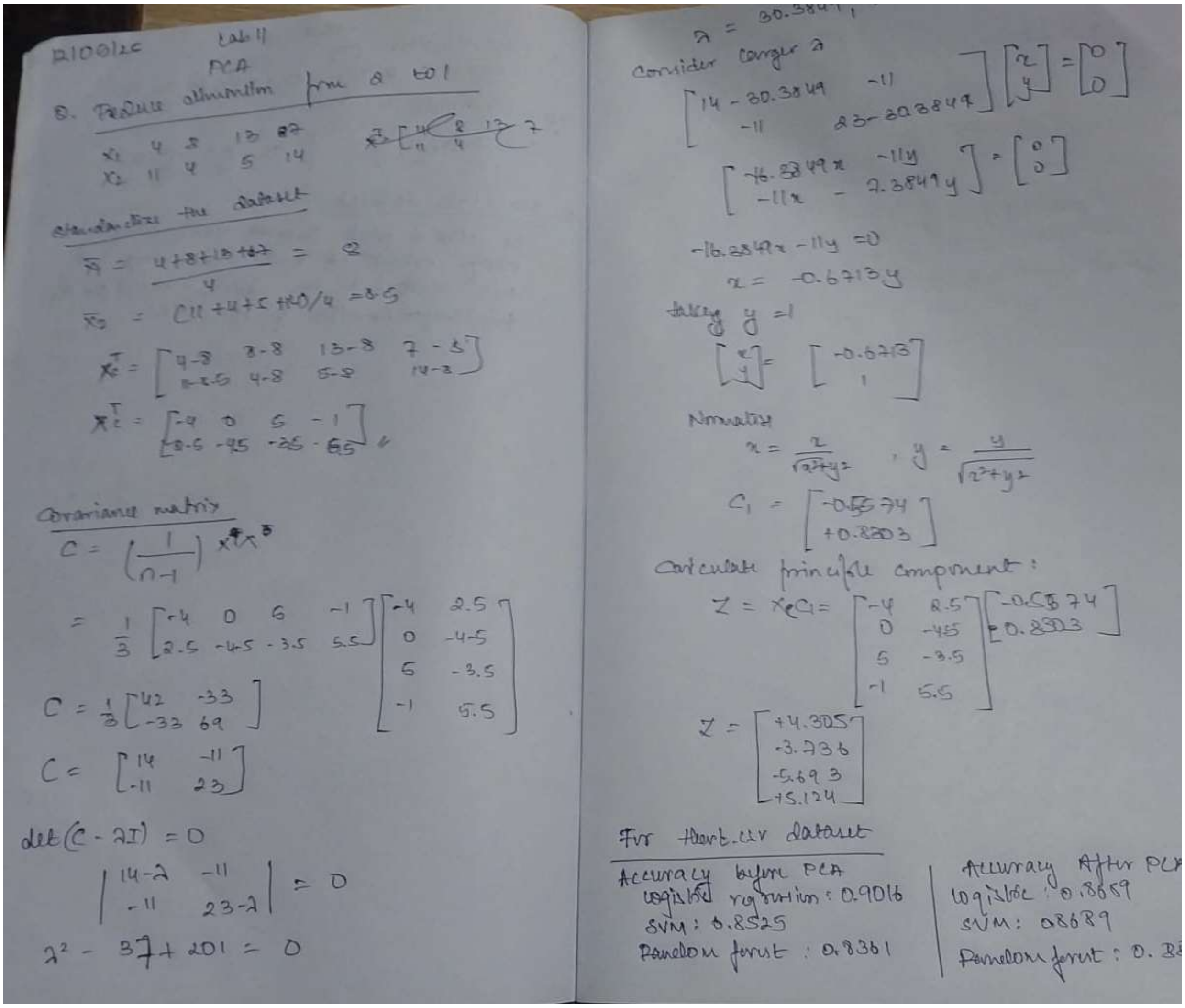
```python
predictions = kmeans.predict(X_test_scaled)

# 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}")
```