VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Machine Learning (23CS6PCMAL)

Submitted by

Ananya Agarwal (1BM22CS039)

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 **Ananya Agarwal (1BM22CS039),** 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.

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

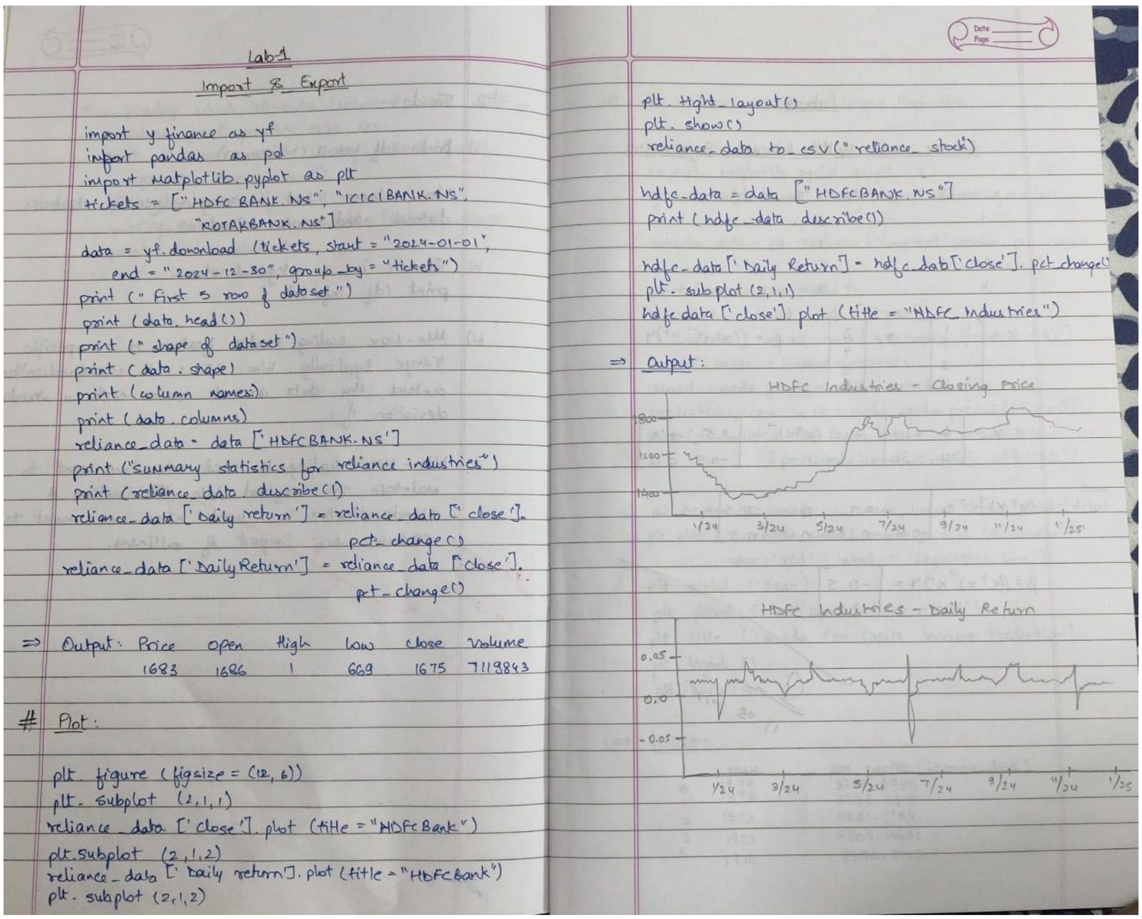
Index

Github Link: https://github.com/AnanyaCSE-039/ML-LAB

Program 1

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

Screenshot



Lab-1

Import & Export

```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
tickets = ["HDFCBANK.NS", "ICICIBANK.NS",
           "KOTAKBANK.NS"]
data = yf.download (tickets, start = "2024-01-01",
       end = "2024-12-30", group_by = "tickets")
print (" First 5 row of dataset ")
print (data.head())
print (" shape of dataset ")
print (data.shape)
print (column names)
print (data.columns)
reliance_data = data ['HDFCBANK.NS']
print ("summary statistics for reliance industries")
print (reliance_data.describe())
reliance_data ['Daily return'] = reliance_data ['close'].
                              pct_change()
reliance_data ['DailyReturn'] = reliance_data ['close'].
                              pct_change()
```

⇒ Output:

Price	open	High	low	close	volume
1683	1686	1	669	1675	7119843

\# Plot:

```
plt.figure (figsize = (12, 6))
plt.subplot (2,1,1)
reliance_data ['close'].plot (title = "HDFCBank")
plt.subplot (2,1,2)
reliance_data ['Daily return'].plot (title = "HDFCBank")
plt.subplot (2,1,2)
plt.tight_layout()
plt.show()
reliance_data.to_csv ("reliance_stock")

hdfc_data = data ["HDFCBANK.NS"]
print (hdfc_data.describe())

hdfc_data ['Daily Return'] = hdfc_data ['close'].pct_change()
plt.subplot (2,1,1)
hdfc_data ['close'].plot (title = "HDFC Industries")
```

⇒ Output:

HDFC Industries - Closing Price

HDFC Industries - Daily Return

Code:

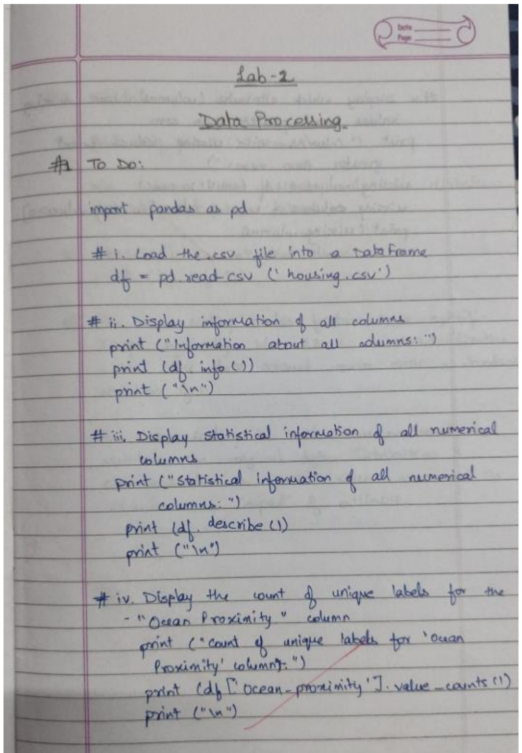
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from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

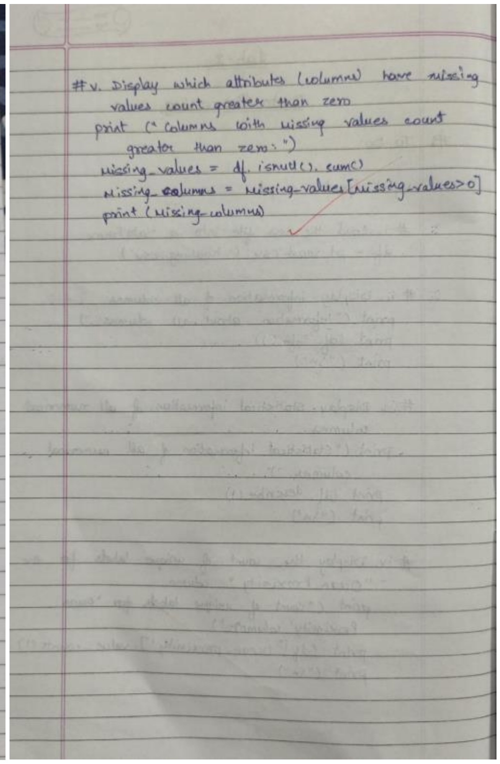


Lab-2

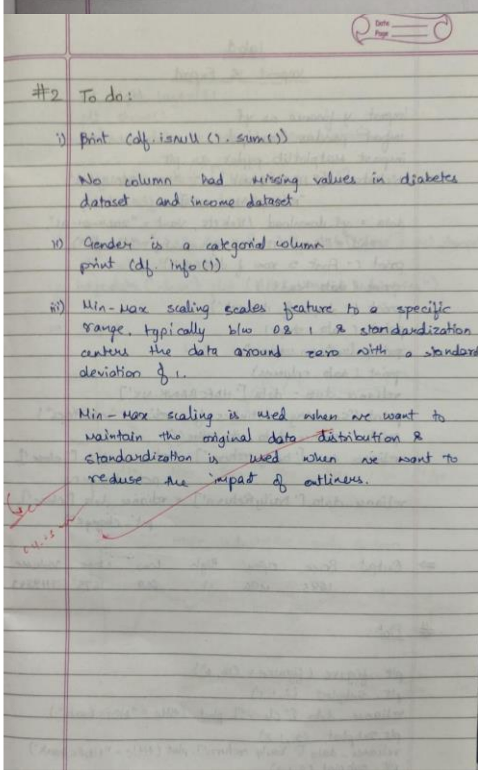
Data Processing

#1 To Do:

```
import pandas as pd

# i. Load the .csv file into a DataFrame
df = pd.read_csv ('housing.csv')

# ii. Display information of all columns
print ("Information about all columns:")
print (df.info ())
print ("\n")

# iii. Display statistical information of all numerical columns
print ("Statistical information of all numerical columns:")
print (df.describe ())
print ("\n")

# iv. Display the count of unique labels for the "Ocean Proximity" column
print ("Count of unique labels for 'Ocean Proximity' column:")
print (df['Ocean_proximity'].value_counts ())
print ("\n")
```



```
#v. Display which attributes (columns) have missing values count greater than zero
print ("Columns with missing values count greater than zero:")
missing_values = df.isnull().sum()
missing_columns = missing_values[missing_values>0]
print (missing_columns)
```



#2 To do:

i) Print (df.isnull ().sum())

No column had missing values in diabetes dataset and income dataset

ii) Gender is a categorial column
print (df.info())

iii) Min-Max scaling scales feature to a specific range, typically b/w 0 & 1 & standardization centers the data around zero with a standard deviation of 1.

Min-Max scaling is used when we want to maintain the original data distribution & standardization is used when we want to reduse the impact of outliners.

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):
```

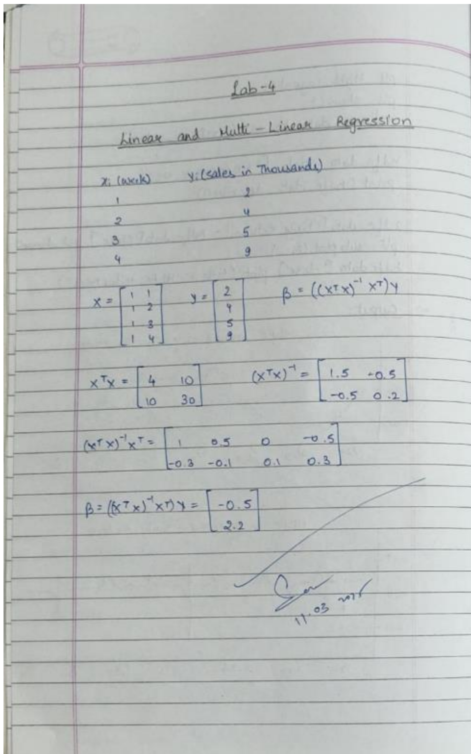
```
        # Assumes X is a NumPy array with the following columns:
        # total_rooms (index 3), total_bedrooms (index 2), population (index 4), households (index 5)
        rooms_per_household = X[:, 3] / X[:, 5]
        population_per_household = X[:, 4] / X[:, 5]
        if self.add_bedrooms_per_room:
            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



Lab-4

Linear and Multi-Linear Regression

x: (week)	y: (sales in Thousands)
1	2
2	4
3	5
4	9

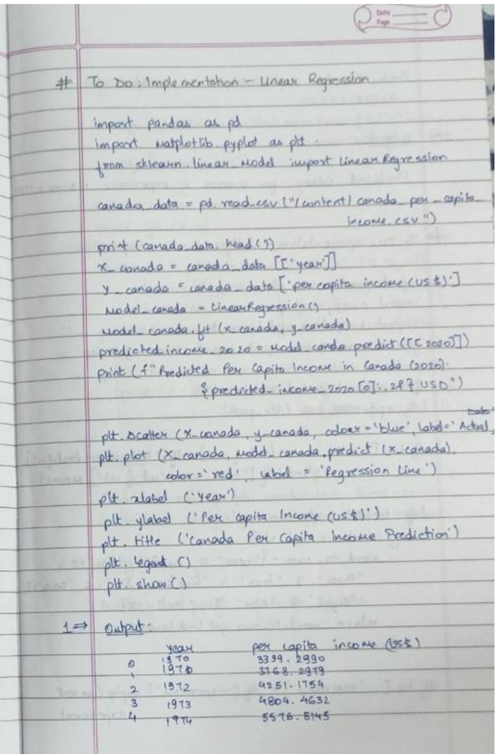
$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{bmatrix}$ $y = \begin{bmatrix} 2 \\ 4 \\ 5 \\ 9 \end{bmatrix}$ $\beta = ((X^TX)^{-1} X^T)y$

$X^TX = \begin{bmatrix} 4 & 10 \\ 10 & 30 \end{bmatrix}$ $(X^TX)^{-1} = \begin{bmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{bmatrix}$

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

$\beta = ((X^TX)^{-1}X^T)y = \begin{bmatrix} -0.5 \\ 2.2 \end{bmatrix}$

11.03

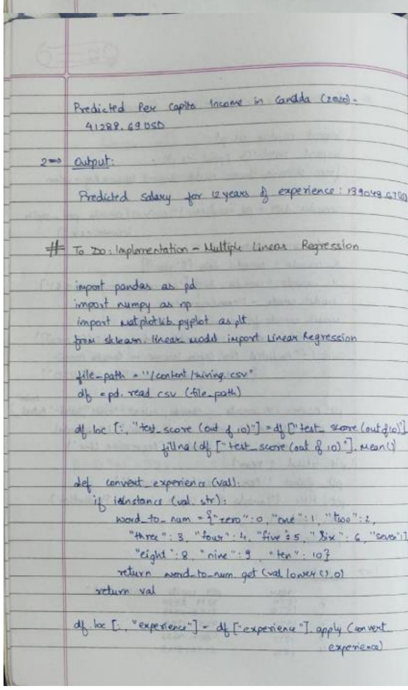


\# To Do: Implementation - Linear Regression

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

canada_data = pd.read_csv("/content/canada_per_capita_income.csv")

print(canada_data.head())
x_canada = canada_data[['year']]
y_canada = canada_data['per capita income (US$)']
model_canada = LinearRegression()
model_canada.fit(x_canada, y_canada)
predicted_income_2020 = model_canada.predict([[2020]])
print(f"Predicted Per Capita Income in Canada (2020):
      {predicted_income_2020[0]:.2f} USD")

plt.scatter(x_canada, y_canada, color='blue', label='Actual Data')
plt.plot(x_canada, model_canada.predict(x_canada),
      color='red', label='Regression Line')
plt.xlabel('year')
plt.ylabel('Per capita Income (US$)')
plt.title('Canada Per Capita Income Prediction')
plt.legend()
plt.show()
```

1 ⇒ Output:

	year	per capita income (US$)
0	1970	3399.2990
1	1971	3768.2979
2	1972	4251.1754
3	1973	4804.4632
4	1974	5576.5145



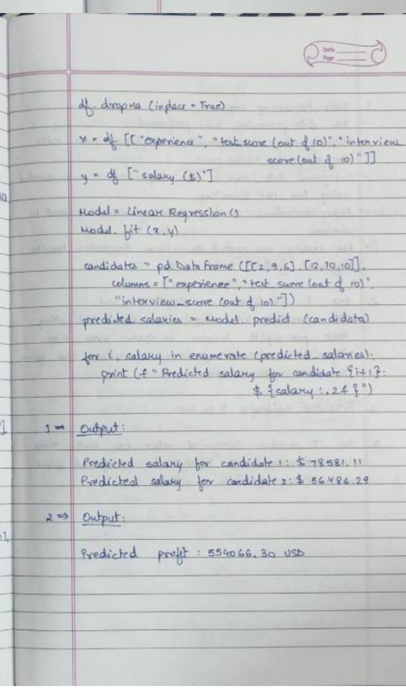
Predicted Per Capita Income in Canada (2020): 41288.69 USD

2 ⇒ Output:

Predicted salary for 12 years of experience: 139049.6790

\# To Do: Implementation - Multiple Linear Regression

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

file_path = "/content/hiring.csv"
df = pd.read_csv(file_path)

df.loc[:, "test_score (out of 10)"] = df["test_score (out of 10)"].
          fillna(df["test_score (out of 10)"].mean())

def convert_experience(val):
    if isinstance(val, str):
        word_to_num = {"zero": 0, "one": 1, "two": 2,
          "three": 3, "four": 4, "five": 5, "six": 6, "seven": 7,
          "eight": 8, "nine": 9, "ten": 10}
        return word_to_num.get(val.lower(), 0)
    return val

df.loc[:, "experience"] = df["experience"].apply(convert_
                                        experience)
```



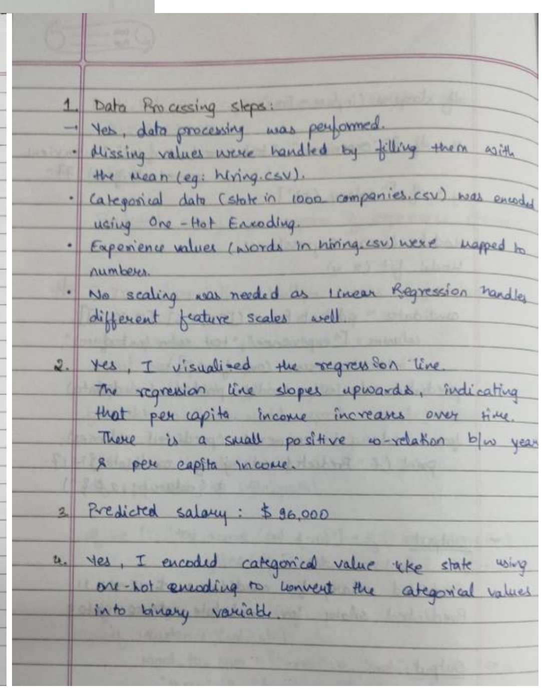
```
df.dropna(inplace=True)

x = df[["experience", "test_score (out of 10)", "interview_
                                    score(out of 10)"]]
y = df["salary ($)"]

Model = LinearRegression()
Model.fit(x, y)

candidates = pd.DataFrame([[2, 9, 6], [12, 10, 10]],
     columns = ["experience", "test_score (out of 10)",
     "interview_score (out of 10)"])
predicted_salaries = model.predict(candidata)

for i, salary in enumerate(predicted_salaries):
    print(f"Predicted salary for candidate {i+1}:
                              ${salary:.2f}")
```

1 ⇒ Output:

Predicted salary for candidate 1: $ 78581.11
Predicted salary for candidate 2: $ 56486.29

2 ⇒ Output:

Predicted profit : 554066.30 USD



1. Data Processing steps:
→ Yes, data processing was performed.
- Missing values were handled by filling them with the mean (eg: hiring.csv).
- Categorical data (state in 1000 companies.csv) was encoded using One-Hot Encoding.
- Experience values (words in hiring.csv) were mapped to numbers.
- No scaling was needed as Linear Regression handles different feature scales well.

2. Yes, I visualized the regression line.
The regression line slopes upwards, indicating that per capita income increases over time. There is a small positive co-relation b/w year & per capita income.

3. Predicted salary : $ 96,000

4. Yes, I encoded categorical value like state using one-hot encoding to convert the categorical values into binary variable.

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
```

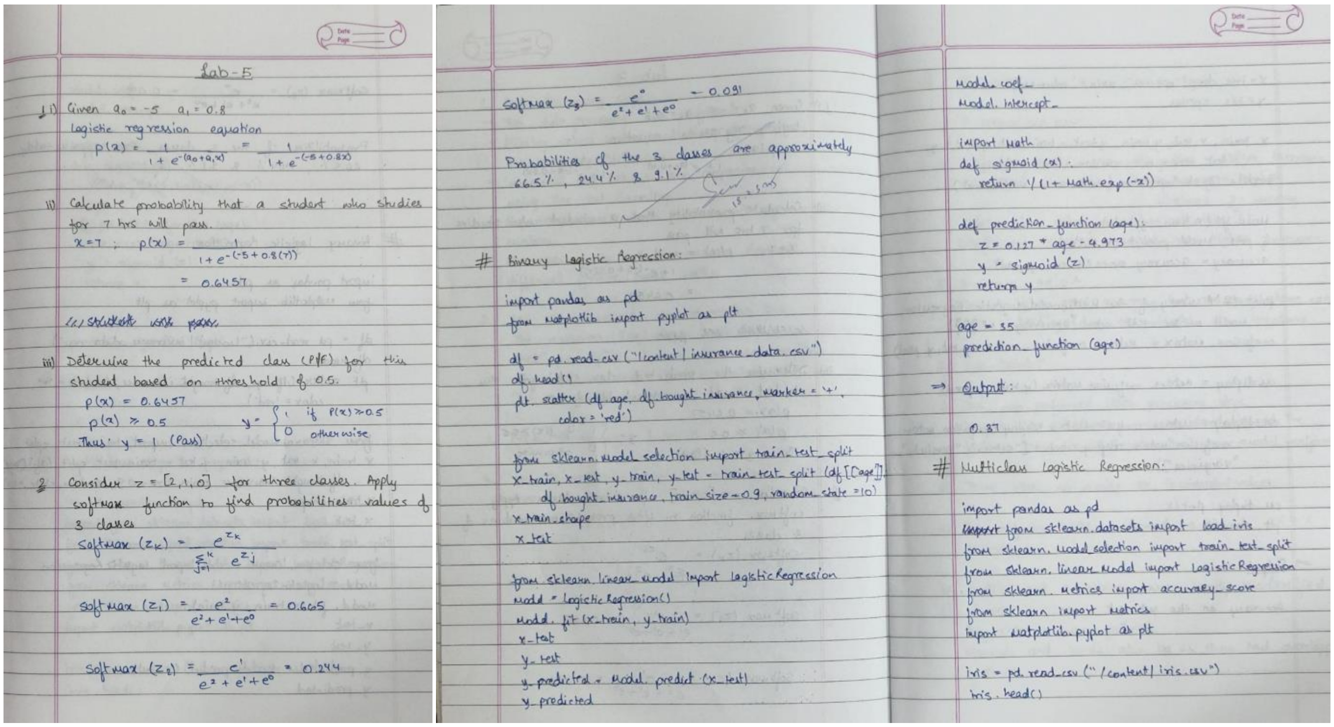
```
model = LinearRegression()
model.fit(X, y)

# 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



Lab-5

1 i) Given $a_0 = -5$ $a_1 = 0.8$

Logistic regression equation

$$p(x) = \frac{1}{1+e^{-(a_0+a_1x)}} = \frac{1}{1+e^{-(-5+0.8x)}}$$

ii) Calculate probability that a student who studies for 7 hrs will pass.

$$x=7 \; ; \; p(x) = \frac{1}{1+e^{-(-5+0.8(7))}} = 0.6457$$

~~∴ student will pass~~

iii) Determine the predicted class (P/F) for this student based on threshold of 0.5.

$p(x) = 0.6457$

$p(x) \geq 0.5$

$$y = \begin{cases} 1 & \text{if } P(x) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Thus: $y = 1$ (Pass)

2 Consider $z = [2,1,0]$ for three classes. Apply softmax function to find probabilities values of 3 classes

$$\text{softmax}(z_k) = \frac{e^{z_k}}{\sum_{j=1}^{k} e^{z_j}}$$

$$\text{softmax}(z_1) = \frac{e^2}{e^2+e^1+e^0} = 0.665$$

$$\text{softmax}(z_2) = \frac{e^1}{e^2+e^1+e^0} = 0.244$$

$$\text{softmax}(z_3) = \frac{e^0}{e^2+e^1+e^0} = 0.091$$

Probabilities of the 3 classes are approximately 66.5%, 24.4% & 9.1%

Binary Logistic Regression:

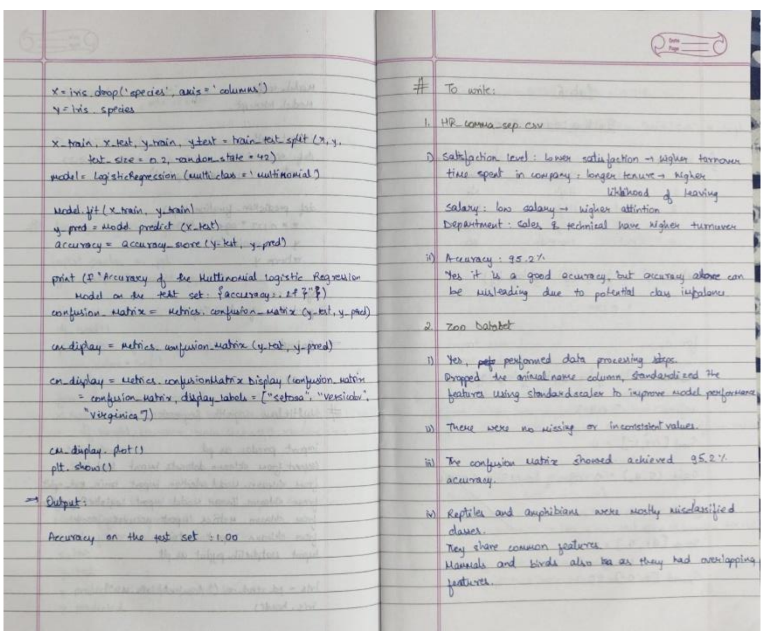
```
import pandas as pd
from matplotlib import pyplot as plt

df = pd.read_csv("/content/insurance_data.csv")
df.head()
plt.scatter(df.age, df.bought_insurance, marker = '+',
    color = 'red')

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df[['age']],
    df.bought_insurance, train_size = 0.9, random_state = 10)
X_train.shape
X_test

from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
X_test
y_test
y_predicted = model.predict(X_test)
y_predicted

model.coef_
model.intercept_

import math
def sigmoid(x):
    return 1/(1+math.exp(-x))

def prediction_function(age):
    z = 0.127 * age - 4.973
    y = sigmoid(z)
    return y

age = 35
prediction_function(age)
```

⇒ Output:

0.37

Multiclass Logistic Regression:

```
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("/content/iris.csv")
iris.head()
```



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

X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size = 0.2, 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(f'Accuracy of the Multinomial Logistic Regression
    Model on the test set: {accuracy:.2f}')
confusion_matrix = metrics.confusion_matrix(y_test, y_pred)

cm_display = metrics.confusion_matrix(y_test, y_pred)

cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix
    = confusion_matrix, display_labels = ["setosa", "versicolor",
    "virginica"])

cm_display.plot()
plt.show()
```

→ Output:

Accuracy on the test set : 1.00

To write:

1. HR_comma_sep.csv

i) Satisfaction level: lower satisfaction → higher turnover
time spent in company: longer tenure → higher likelihood of leaving
salary: low salary → higher attrition
Department: sales & technical have higher turnover

ii) Accuracy: 95.2%
Yes it is a good accuracy, but accuracy alone can be misleading due to potential class imbalance

2. Zoo Dataset

i) Yes, performed data processing steps.
Dropped the animal name column, standardized the features using StandardScaler to improve model performance

ii) There were no missing or inconsistent values.

iii) The confusion matrix showed achieved 95.2% accuracy.

iv) Reptiles and amphibians were mostly misclassified classes.
They share common features.
Mammals and birds also as they had overlapping features.

Code:

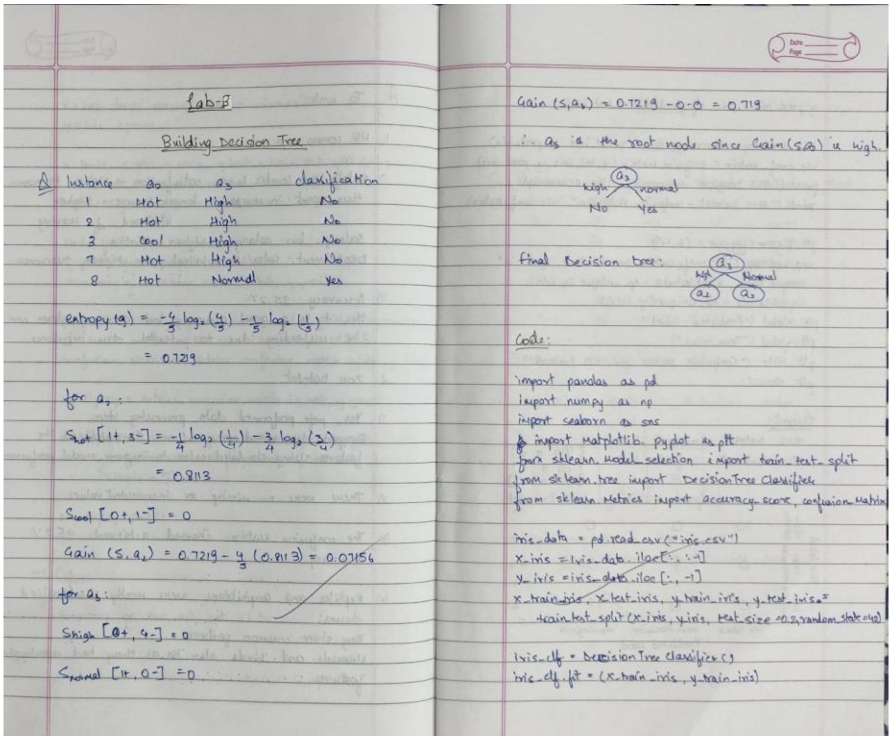
```
import pandas as pd

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



Lab-B

Building Decision Tree

Instance	a_2	a_3	classification
1	Hot	High	No
2	Hot	High	No
3	Cool	High	No
7	Hot	High	No
8	Hot	Normal	Yes

entropy (a) $= -\frac{4}{5}\log_2\left(\frac{4}{5}\right) - \frac{1}{5}\log_2\left(\frac{1}{5}\right)$

$= 0.7219$

for a_2:

$S_{hot}[1+, 3-] = -\frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right)$

$= 0.8113$

$S_{cool}[0+, 1-] = 0$

Gain $(S, a_2) = 0.7219 - \frac{4}{5}(0.8113) = 0.07156$

for a_3:

$S_{high}[0+, 4-] = 0$

$S_{normal}[1+, 0-] = 0$

Gain $(S, a_3) = 0.7219 - 0 - 0 = 0.719$

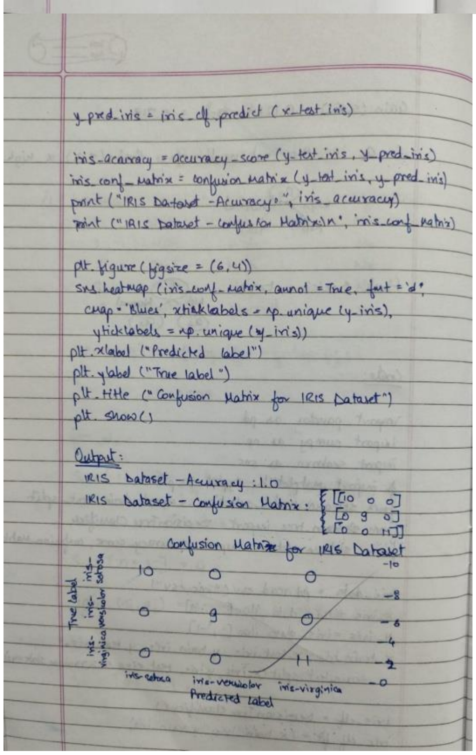
$\therefore a_3$ is the root node since Gain(S, a_3) is high.

a_3: high → No, normal → Yes

Final Decision tree: a_3: High → a_2, Normal → a_3

Code:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

iris_data = pd.read_csv("iris.csv")
x_iris = iris_data.iloc[:, :-1]
y_iris = iris_data.iloc[:, -1]
x_train_iris, x_test_iris, y_train_iris, y_test_iris =
    train_test_split(x_iris, y_iris, test_size=0.2, random_state=42)

iris_clf = DecisionTreeClassifier()
iris_clf.fit = (x_train_iris, y_train_iris)
```



```
y_pred_iris = iris_clf.predict (x_test_iris)

iris_accuracy = accuracy_score (y_test_iris, y_pred_iris)
iris_conf_matrix = confusion_matrix (y_test_iris, y_pred_iris)
print ("IRIS Dataset - Accuracy:", iris_accuracy)
print ("IRIS Dataset - Confusion Matrix:\n", iris_conf_matrix)

plt.figure(figsize = (6,4))
sns.heatmap (iris_conf_matrix, annot = True, fmt = 'd',
  cmap = 'Blues', xticklabels = np.unique (y_iris),
  yticklabels = np.unique (y_iris))
plt.xlabel ("Predicted label")
plt.ylabel ("True label")
plt.title ("Confusion Matrix for IRIS Dataset")
plt.show()
```

Output:

IRIS Dataset - Accuracy : 1.0

IRIS Dataset - Confusion Matrix:

```
[[10  0  0]
 [ 0  9  0]
 [ 0  0 11]]
```

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)
```

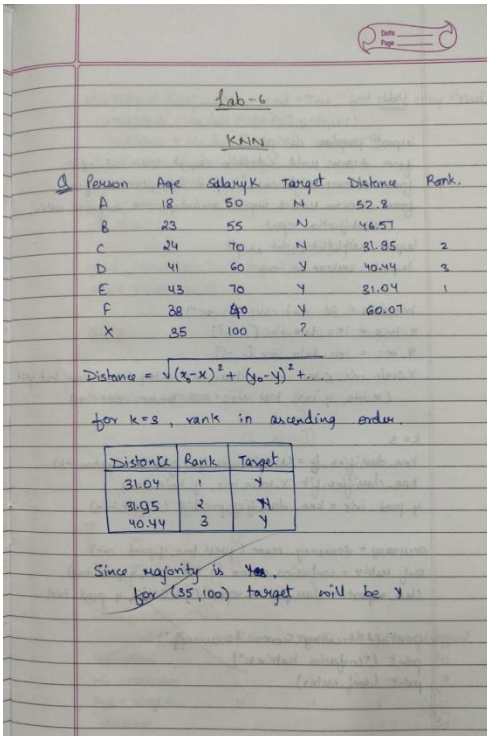
```
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
plt.figure(figsize=(12, 8))
tree.plot_tree(clf, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.show()


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



Lab-6

KNN

Q

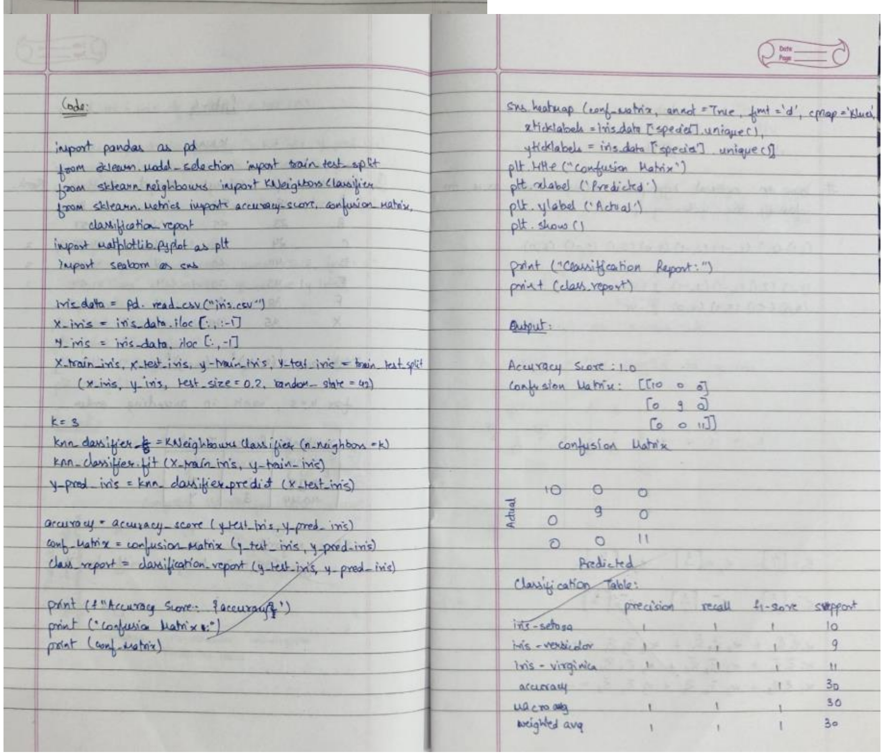
Person	Age	Salary K	Target	Distance	Rank.
A	18	50	N	52.8	
B	23	55	N	46.57	
C	24	70	N	31.95	2
D	41	60	Y	40.44	3
E	43	70	Y	31.04	1
F	38	40	Y	60.07	
X	35	100	?		

$$\text{Distance} = \sqrt{(x_0-x)^2 + (y_0-y)^2 + \ldots}$$

for k=3, rank in ascending order.

Distance	Rank	Target
31.04	1	Y
31.95	2	N
40.44	3	Y

Since majority is Y,
for (35,100) target will be Y



Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbours import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
    classification_report
import matplotlib.pyplot as plt
import seaborn as sns

iris_data = pd.read_csv("iris.csv")
x_iris = iris_data.iloc[:, :-1]
y_iris = iris_data.iloc[:, -1]
x_train_iris, x_test_iris, y_train_iris, y_test_iris = train_test_split
    (x_iris, y_iris, test_size=0.2, random_state=42)

k=3
knn_classifier = KNeighboursClassifier(n_neighbors=k)
knn_classifier.fit(x_train_iris, y_train_iris)
y_pred_iris = knn_classifier.predict(x_test_iris)

accuracy = accuracy_score(y_test_iris, y_pred_iris)
conf_matrix = confusion_matrix(y_test_iris, y_pred_iris)
class_report = classification_report(y_test_iris, y_pred_iris)

print(f"Accuracy Score: {accuracy}")
print("Confusion Matrix:")
print(conf_matrix)

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
    xticklabels=iris_data['species'].unique(),
    yticklabels=iris_data['species'].unique())
plt.title("Confusion Matrix")
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

print("Classification Report:")
print(class_report)
```

Output:

Accuracy Score: 1.0
Confusion Matrix: [[10 0 0]
[0 9 0]
[0 0 11]]

confusion Matrix

$$\begin{array}{c|ccc} & 10 & 0 & 0 \\ \text{Actual} & 0 & 9 & 0 \\ & 0 & 0 & 11 \\ \hline & & \text{Predicted} & \end{array}$$

Classification Table:

	precision	recall	f1-score	support
iris-setosa	1	1	1	10
iris-versicolor	1	1	1	9
iris-virginica	1	1	1	11
accuracy			1	30
macro avg	1	1	1	30
weighted avg	1	1	1	30

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
```

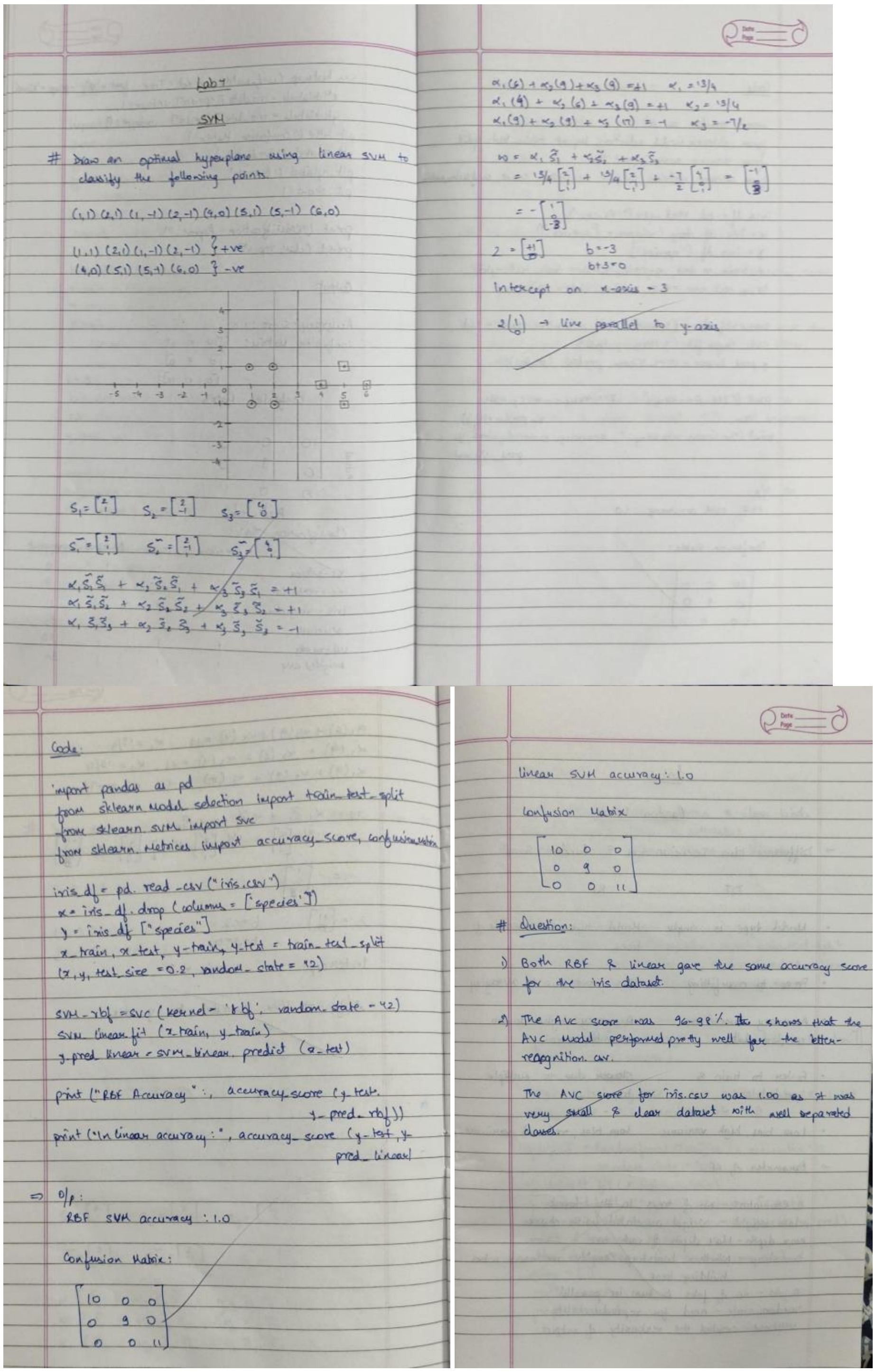
```
# Load data
df = pd.read_csv('/content/iris (1).csv')
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

# 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



Lab 7

SVM

Draw an optimal hyperplane using linear SVM to classify the following points.

(1,1) (2,1) (1,-1) (2,-1) (4,0) (5,1) (5,-1) (6,0)

(1,1) (2,1) (1,-1) (2,-1) } +ve
(4,0) (5,1) (5,-1) (6,0) } -ve

$S_1 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$ $S_2 = \begin{bmatrix} 2 \\ -1 \end{bmatrix}$ $S_3 = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$

$\tilde{S}_1 = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$ $\tilde{S}_2 = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$ $\tilde{S}_3 = \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix}$

$\alpha_1 \tilde{S}_1 \tilde{S}_1 + \alpha_2 \tilde{S}_2 \tilde{S}_1 + \alpha_3 \tilde{S}_3 \tilde{S}_1 = +1$
$\alpha_1 \tilde{S}_1 \tilde{S}_2 + \alpha_2 \tilde{S}_2 \tilde{S}_2 + \alpha_3 \tilde{S}_3 \tilde{S}_2 = +1$
$\alpha_1 \tilde{S}_1 \tilde{S}_3 + \alpha_2 \tilde{S}_2 \tilde{S}_3 + \alpha_3 \tilde{S}_3 \tilde{S}_3 = -1$

$\alpha_1(6) + \alpha_2(4) + \alpha_3(9) = +1$ $\alpha_1 = 13/4$
$\alpha_1(4) + \alpha_2(6) + \alpha_3(9) = +1$ $\alpha_2 = 13/4$
$\alpha_1(9) + \alpha_2(9) + \alpha_3(17) = -1$ $\alpha_3 = -7/2$

$w = \alpha_1 \tilde{S}_1 + \alpha_2 \tilde{S}_2 + \alpha_3 \tilde{S}_3$
$= 13/4 \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} + 13/4 \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix} + \frac{-7}{2} \begin{bmatrix} 4 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 3 \end{bmatrix}$
$= - \begin{bmatrix} 1 \\ 0 \\ -3 \end{bmatrix}$

$2 = \begin{bmatrix} +1 \\ 0 \end{bmatrix}$ $b = -3$
$b + 3 = 0$

Intercept on x-axis = 3

$2\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ → line parallel to y-axis

Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrices import accuracy_score, confusion_matrix

iris_df = pd.read_csv("iris.csv")
x = iris_df.drop(columns = ['species'])
y = iris_df["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', random_state = 42)
svm_linear.fit(x_train, y_train)
y_pred_linear = svm_linear.predict(x_test)

print("RBF Accuracy":, accuracy_score(y_test,
                                     y_pred_rbf))
print("\n linear accuracy:", accuracy_score(y_test, y_
                                     pred_linear)
```

⇒ O/p:
RBF SVM accuracy : 1.0

Confusion Matrix :

$\begin{bmatrix} 10 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 11 \end{bmatrix}$

linear SVM accuracy : 1.0

Confusion Matrix

$\begin{bmatrix} 10 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 11 \end{bmatrix}$

Question:

1) Both RBF & linear gave the same accuracy score for the iris dataset.

2) The AUC score was 96-98%. This shows that the AUC model performed pretty well for the letter-recognition.csv.

The AUC score for iris.csv was 1.00 as it was very small & clear dataset with well separated classes.

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']
```

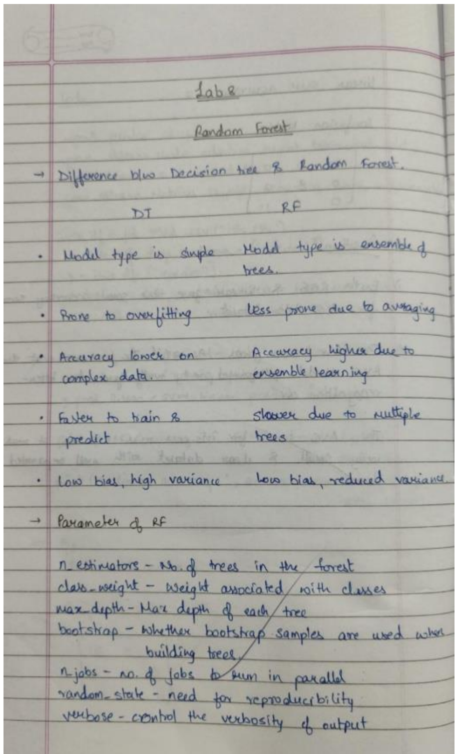
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

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



Lab 8

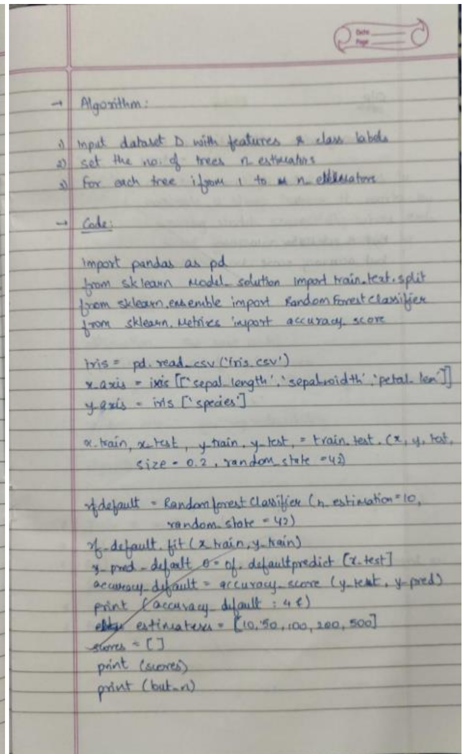
Random Forest

→ Difference b/w Decision tree & Random Forest.

DT	RF
Model type is simple	Model type is ensemble of trees.
Prone to overfitting	less prone due to averaging
Accuracy lower on complex data.	Accuracy higher due to ensemble learning
Faster to train & predict	slower due to multiple trees
Low bias, high variance	Low bias, reduced variance.

→ Parameter of RF

n_estimators - No. of trees in the forest
class_weight - weight associated with classes
max_depth - Max depth of each tree
bootstrap - whether bootstrap samples are used when building trees.
n_jobs - no. of jobs to run in parallel
random_state - need for reproducibility
verbose - control the verbosity of output

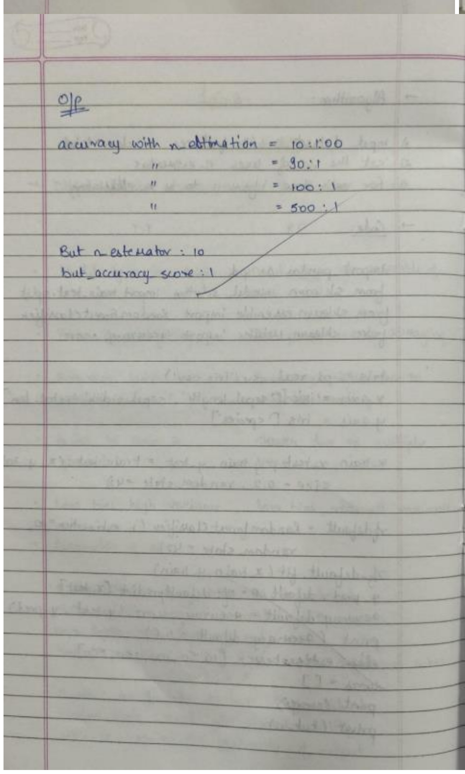


→ Algorithm:

1) input dataset D with features & class labels
2) set the no. of trees n_estimators
3) For each tree i from 1 to n_estimators

→ Code:

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

iris = pd.read_csv ('iris.csv')
x_axis = iris [['sepal_length', 'sepal_width', 'petal_len']]
y_axis = iris ['species']

x_train, x_test, y_train, y_test, = train_test_(x, y, test_
        size = 0.2, random_state = 42)

rf_default = RandomForestClassifier (n_estimation = 10,
        random_state = 42)
rf_default.fit (x_train, y_train)
y_pred_default = rf_defaultpredict [x_test]
accuracy_default = accuracy_score (y_test, y_pred)
print (accuracy_default : 4 f)
estimaters = [10, 50, 100, 200, 500]
scores = []
print (scores)
print (but_n)
```



O/P

accuracy with n_estimation = 10 : 1.00
" = 50 : 1
" = 100 : 1
" = 500 : 1

But n_estemator : 10
but_accuracy score : 1

Code:

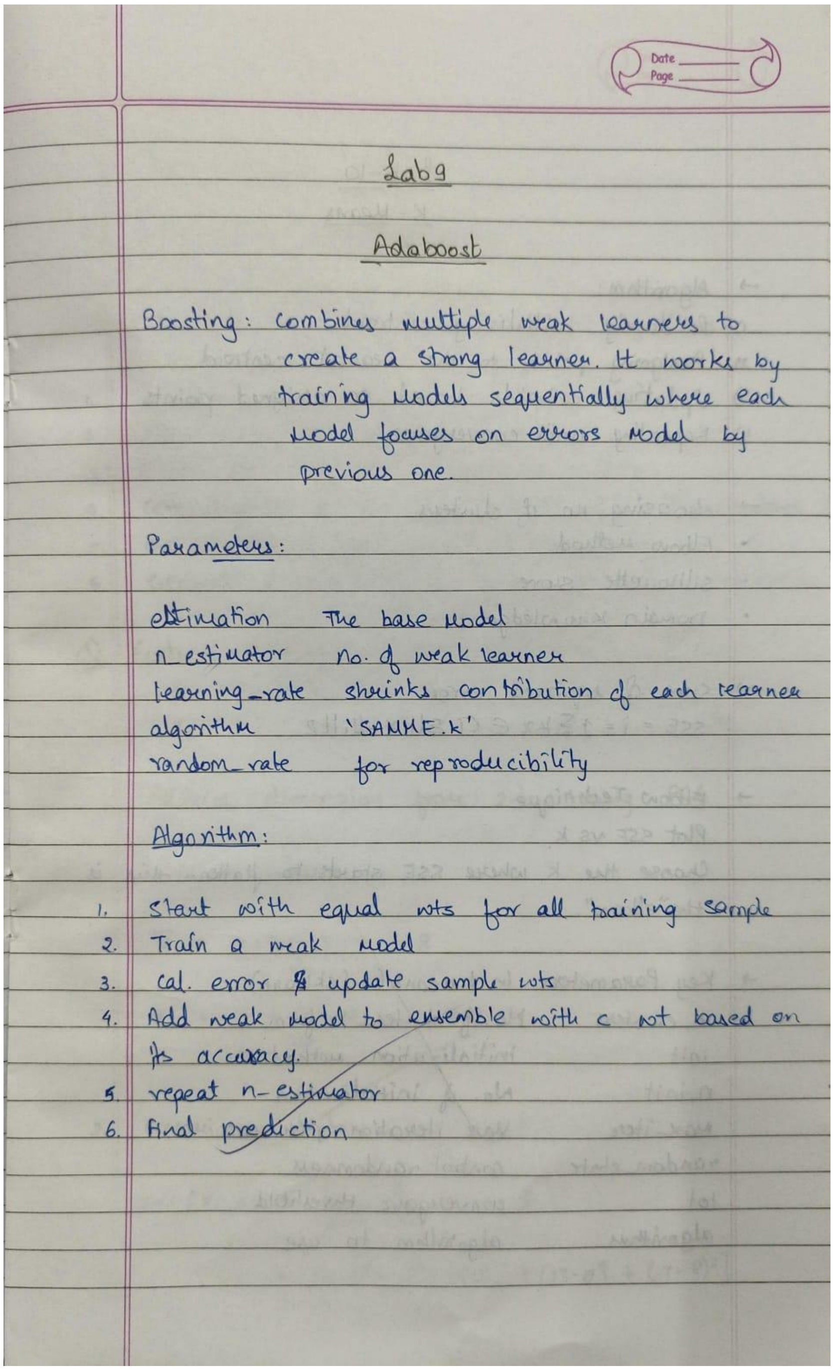
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# 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



Lab9

Adaboost

Boosting: combines multiple weak learners to create a strong learner. It works by training models sequentially where each model focuses on errors model by previous one.

Parameters:

estimation The base model
n_estimator no. of weak learner
learning_rate shrinks contribution of each learner
algorithm 'SAMME.R'
random_rate for reproducibility

Algorithm:

1. Start with equal wts for all training sample
2. Train a weak model
3. cal. error & update sample wts
4. Add weak model to ensemble with c wt based on its accuracy.
5. repeat n-estimator
6. Final prediction

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

Lab-10

K-Means

→ Algorithm:
1) Randomly initializing centroids
2) Assigning points to the nearest centroid
3) Updating centroids based on assigned points
4) Repeating until convergence.

→ choosing no. of clusters
- Elbow method
- silhouette score
- Domain knowledge

→ Sum of squared error

$SSE = i = 1 \sum k x \in C_i \sum \| x - M_i \| 2$

→ Elbow Technique

Plot SSE vs k

Choose the k where SSE starts to flatten - this is the "elbow".

→ Key Parameters in kmeans() (sklearn)

n_clusters	No. of clusters to form
init	initialization method
n_init	No. of initialization
max_iter	Max iteration per run
random_state	control randomness
tol	convergence threshold
algorithm	algorithm to use.

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

Lab11

PCA

Principal component Analysis (PCA) algorithm.

1. Calculate Mean
2. Calculation of covariance matrix
3. Eigenvalues of the covariance matrix
4. computation of the eigenvectors - Unit eigenvectors
5. Computation of first principal components
6. Geometrical meaning of first principal components.

Q

Feature	Ex 1	Ex 2	Ex 3	Ex 4
x_1	4	8	13	7
x_2	11	4	5	14

Reduce dimension from 2 to 1 using PCA

1. Calculate Mean:

$$\bar{x}_1 = \frac{4+8+13+7}{4} = 8$$

$$\bar{x}_2 = \frac{11+4+5+14}{4} = 8.5$$

2. Covariance Matrix:

$$cov(x_1, x_1) = \frac{1}{N-1}\sum_{k=1}^{N}(x_{1k} - \bar{x}_1)^2$$
$$= 1/3\left[(4-8)^2 + (8-8)^2 + (13-8)^2 + (7-8)^2\right]$$
$$= 14$$

$$cov(x_1, x_2) = \frac{1}{N-1}\sum_{k=1}^{N}(x_{1k} - \bar{x}_1)(x_{2k} - \bar{x}_2)$$
$$= 1/3\left[(4-8)(11-8.5) + (8-8)(4-8.5) + (13-8)(5-8.5) + (7-8)(14-8.5)\right]$$
$$= -11$$

$$cov(x_2, x_1) = cov(x_1, x_2)$$
$$= -11$$

$$cov(x_2, x_2) = \frac{1}{N-1}\sum_{k=1}^{N}(x_{2k} - \bar{x}_2)^2$$
$$= 1/3\left[(11-8.5)^2 + (4-8.5)^2 + (5-8.5)^2 + (14-8.5)^2\right]$$
$$= 23$$

Covariance Matrix:

$$S = \begin{bmatrix} cov(x_1, x_1) & cov(x_1, x_2) \\ cov(x_2, x_1) & cov(x_2, x_2) \end{bmatrix} = \begin{bmatrix} 14 & -11 \\ -11 & 23 \end{bmatrix}$$

3. Eigen values:

Characteristic eqn of covariance matrix:

$$0 = \det(S - \lambda I)$$
$$= \begin{bmatrix} 14-\lambda & -11 \\ -11 & 23-\lambda \end{bmatrix}$$
$$= (14-\lambda)(23-\lambda) - (-11)(-11)$$
$$= 322 - 23\lambda - 14\lambda + \lambda^2 - 121$$
$$= \lambda^2 - 37\lambda + 201$$

$$\lambda = \frac{1}{2}(37 \pm \sqrt{565})$$
$$= 30.3849, 6.6151$$
$$= \lambda_1, \lambda_2$$

4. computation of eigenvectors:

$$U = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} = (S - \lambda_1 I)X$$
$$= \begin{bmatrix} 14-\lambda_1 & -11 \\ -11 & 23-\lambda_1 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$
$$= \begin{bmatrix} (14-\lambda_1)u_1 & -11u_2 \\ -11u_1 & (23-\lambda_1)u_2 \end{bmatrix}$$

$$(14-\lambda_1)u_1 - 11u_2 = 0 \quad -①$$
$$-11u_1 + (23-\lambda_1)u_2 = 0$$

$$\frac{u_1}{11} = \frac{u_2}{14-\lambda_1} = t \qquad \text{from eq. ①}$$

$$u_1 = 11t \qquad u_2 = (14-\lambda_1)t$$

taking $t = 1$,

$$U_1 = \begin{bmatrix} 11 \\ 14-\lambda_1 \end{bmatrix}$$

Compute the length of x_1:

$$\|U_1\| = \sqrt{11^2 + (14-\lambda_1)^2}$$
$$= \sqrt{11^2 + (14-30.3849)^2}$$
$$= 19.7348$$

$\therefore$ unit eigenvector corresponding to λ_1 is:

$$e_1 = \begin{bmatrix} 11/\|U_1\| \\ (14-\lambda_1)/\|U_1\| \end{bmatrix} = \begin{bmatrix} 11/19.7348 \\ (14-30.3849)/19.7348 \end{bmatrix} = \begin{bmatrix} 0.5574 \\ -0.8303 \end{bmatrix}$$

After similar computations, the unit eigenvector e_2 corresponding to eigenvalue λ_2 can be shown as

$$e_2 = \begin{bmatrix} 0.8303 \\ 0.5574 \end{bmatrix}$$

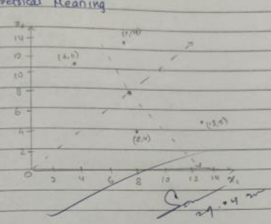
5. Computation of first principal components:

let, $\begin{bmatrix} x_{1k} \\ x_{2k} \end{bmatrix}$

$$e_1^T\begin{bmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - \bar{x}_2 \end{bmatrix} = \begin{bmatrix} 0.5574 & -0.8303 \end{bmatrix}\begin{bmatrix} x_{1k} - \bar{x}_1 \\ x_{2k} - \bar{x}_2 \end{bmatrix}$$

$$= 0.5574(x_{1k} - \bar{x}_1) - 0.8303(x_{2k} - \bar{x}_2)$$

6. Geometrical Meaning



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}")
```