A black background with a black square

Description automatically generated

FACULTY OF ENGINEERING AND TECHNOLOGY BACHELOR OF TECHNOLOGY

High Performance Computing (HPC) (203105430)

VI SEMESTER

Computer Science & Engineering Department





**CERTIFICATE**

*This is to certify that*

*Mr.* **VARIA DHRUV PRAFULBHAI** *with Enrollment No.* **210303105821** has *successfully completed his laboratory experiments in the subject* **Data Science (203105414)** *from the department of* **Computer Science and Engineering** *during the academic year* ***2023-2024.***



**Date of Submission …..…………… Staff In charge …..……………**

**Head of Department …..……………**

**TABLE OF CONTENT**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sr. No** | **Experiment Title** | **Page No** | | **Date of Start** | **Date of Completion** | **Sign** | **Marks (out of 10)** |
| **To** | **From** |
| 1. | House rent prediction using linear regression |  |  |  |  |  |  |
| 2. | Medical diagnosis for  disease spread pattern |  |  |  |  |  |  |
| 3. | Automate email classification and response |  |  |  |  |  |  |
| 4. | Customer segmentation in business model based on their demographic, psychographic and behavior data |  |  |  |  |  |  |
| 5. | Construct a recommendation system based on the customer transaction using  Association rule mining |  |  |  |  |  |  |
| 6. | Behavioral analysis of  customers for any online purchase model |  |  |  |  |  |  |
| 7. | Agricultural data analysis for yield prediction and crop selection on Indian terrain data set |  |  |  |  |  |  |
| 8. | Develop a business model to predict the trend in Investment and Funding |  |  |  |  |  |  |

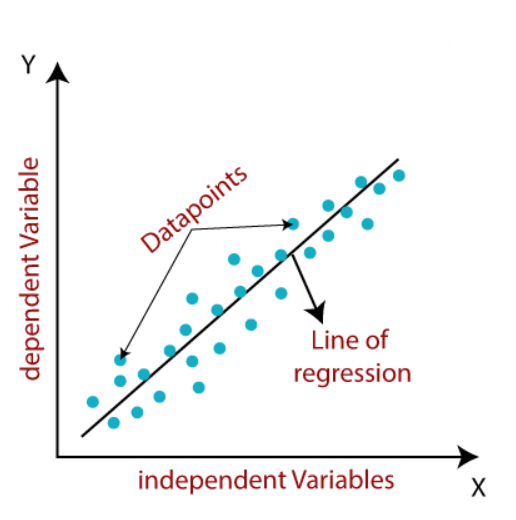
**Practical No – 01**

**Aim : House Rent prediction using linear regression.**

**Theory:**

Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable.

This form of analysis estimates the coefficients of the linear equation, involving one or more independent variables that best predict the value of the dependent variable. Linear regression fits a straight line or surface that minimizes the discrepancies between predicted and actual output values. There are simple linear regression calculators that use a “least squares” method to discover the best-fit line for a set of paired data. You then estimate the value of X (dependent variable) from Y (independent variable).



A diagram of a software development process

Description automatically generated with medium confidence

# package

import pandas as pd # data-set manipulation

import numpy as np # calcualting heavy mathematic and model data nd-array

import matplotlib.pyplot as plt # data visualization

import seaborn as sns # data visualization

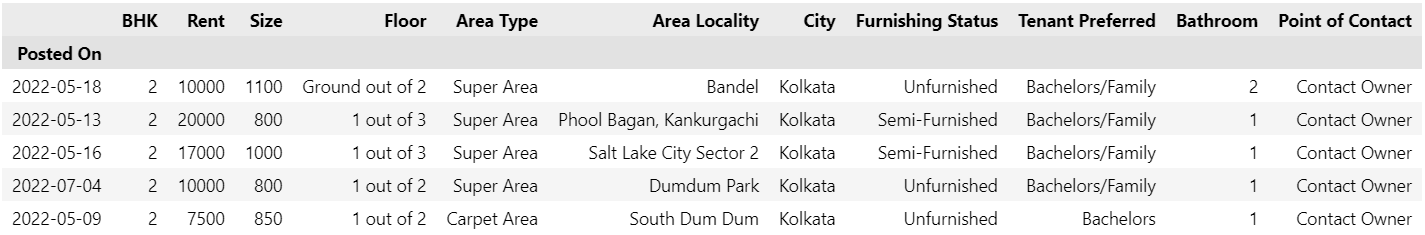
print("ok!!")

## Read Data

data\_path = "D:\programs\data\_science\prac\_1\House\_Rent\_Dataset.csv"

house\_data = pd.read\_csv(data\_path, index\_col="Posted On")

house\_data.head()



# dataset shape, dataset size (no\_of\_rows \* no\_of\_columns)

house\_data.shape , house\_data.size

((4746, 11), 52206)

# data info

house\_data.info()

<class 'pandas.core.frame.DataFrame'>

Index: 4746 entries, 2022-05-18 to 2022-05-04

Data columns (total 11 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 BHK 4746 non-null int64

1 Rent 4746 non-null int64

2 Size 4746 non-null int64

3 Floor 4746 non-null object

4 Area Type 4746 non-null object

5 Area Locality 4746 non-null object

6 City 4746 non-null object

7 Furnishing Status 4746 non-null object

8 Tenant Preferred 4746 non-null object

9 Bathroom 4746 non-null int64

10 Point of Contact 4746 non-null object

dtypes: int64(4), object(7)

memory usage: 444.9+ KB

# simple dataset summary

house\_data.describe()

A table with numbers and symbols

Description automatically generated

# lets visualize our data in scatter plot to see how our data correlate

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

plt.title("Rent vs Size")

plt.xlabel("Size(feet square)")

plt.ylabel("Rent")

# sns.scatterplot(x=house\_data['City'],y=house\_data['Rent'])

sns.scatterplot(x=house\_data['Size'],y=house\_data['Rent'],hue=house\_data['Furnishing Status'])

plt.show()

A graph of a chart

Description automatically generated with medium confidence

# Extract data

area = house\_data["Size"]

price = house\_data["Rent"]

# Calculate regression coefficients manually

mean\_area = np.mean(area)

mean\_price = np.mean(price)

numerator = np.sum((area - mean\_area) \* (price - mean\_price))

denominator = np.sum((area - mean\_area) \*\* 2)

slope = numerator / denominator

intercept = mean\_price - (slope \* mean\_area)

# Make predictions

predicted\_prices = slope \* area + intercept

# Calculate the cost/error

error = sum((predicted\_prices - price) \*\* 2) / (2 \* len(price))

# Scatter plot of the actual data points

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

plt.title("Rent vs Size")

plt.xlabel("Size (square feet)")

plt.ylabel("Rent")

sns.scatterplot(x=area, y=price, hue=house\_data['Furnishing Status'], label="Actual Data")

# Plot the regression line

plt.plot(area, predicted\_prices, color='red', label="Regression Line")

plt.legend()

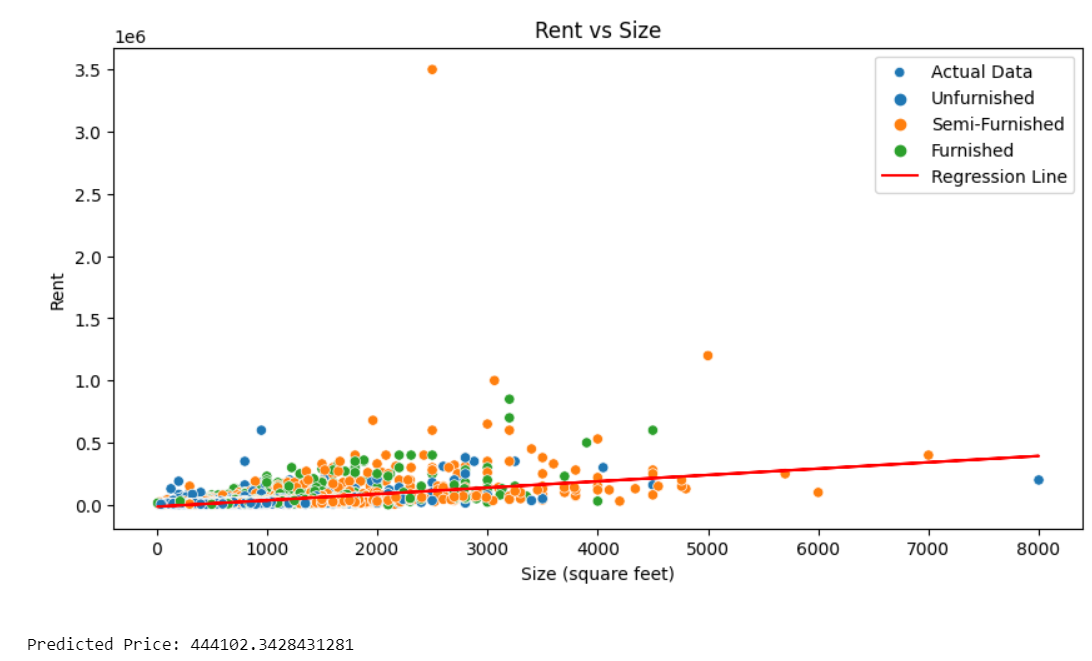
plt.show()

# Predict the house price for a new area

new\_area = 9000

predicted\_price = slope \* new\_area + intercept

print("Predicted Price:", predicted\_price)



**Practical No – 02**

**Aim –** Medical Diagnosis for disease spread pattern

**Theory** **-**

Using two datasets dataset-symptoms.csv which has the symptoms of diseases and symptom\_precaution.csv which has precaution measure for the diseases. Now for that using Linear Regression again to get the prediction based on the symptoms.

**A diagram of a data flow

Description automatically generated**

**Code –**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

# Load the symptom dataset

data = pd.read\_csv(r'D:\programs\data\_science\dataset\_symptom.csv')

df = pd.read\_csv(r'D:\programs\data\_science\symptom\_precaution.csv')

data.head()

**A screenshot of a computer

Description automatically generated**

df.head()

A screenshot of a computer

Description automatically generated

# Extract features (symptoms) and target (disease)

X = data[["Symptom\_1", "Symptom\_2", "Symptom\_3"]]

y = data["Disease"]

# Encode categorical features (one-hot encoding)

X\_encoded = pd.get\_dummies(X)

y = y.ravel()

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_encoded, y, test\_size=0.2, random\_state=42)

# Create and train the logistic regression model

model = LogisticRegression()

model.fit(X\_train, y\_train)

# Define new symptoms for prediction

new\_symptoms = [["itching", "skin\_rash", "nodal\_skin\_eruptions"]]

# Encode new symptoms

new\_symptoms\_encoded = pd.get\_dummies(pd.DataFrame(new\_symptoms, columns=["Symptom\_1", "Symptom\_2", "Symptom\_3"]))

# Ensure the new symptoms have the same columns as the training data

aligned\_features = pd.DataFrame(columns=X\_encoded.columns, data=new\_symptoms\_encoded)

new\_symptoms\_encoded = aligned\_features.fillna(0)

# Predict the disease based on new symptoms

predicted\_diseases = model.predict(new\_symptoms\_encoded)

print("Predicted Diseases:", predicted\_diseases)

dis = predicted\_diseases[0]

**A screenshot of a computer code

Description automatically generated**

# Retrieve precautions for the predicted disease

p1 = df[df['Disease'] == dis]['Precaution\_1']

p2 = df[df['Disease'] == dis]['Precaution\_2']

p3 = df[df['Disease'] == dis]['Precaution\_3']

p4 = df[df['Disease'] == dis]['Precaution\_4']

prec = [p1.values[0], p2.values[0], p3.values[0], p4.values[0]]

print("Precautions:", prec)

Precautions: ['consult nearest hospital', 'vaccination', 'eat healthy', 'medication']

**Practical No – 03**

**Aim –** Automate email classification and response.

**Theory:**

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.

Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.

A diagram of a data flow

Description automatically generated

**Code –**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.metrics import precision\_score, recall\_score, accuracy\_score, f1\_score, confusion\_matrix, ConfusionMatrixDisplay, PrecisionRecallDisplay, RocCurveDisplay

data = pd.read\_csv('D:\programs\data\_science\emails.csv')

data.head()

A screenshot of a computer

Description automatically generated

data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 5172 entries, 0 to 5171

Columns: 3002 entries, Email No. to Prediction

dtypes: int64(3001), object(1)

memory usage: 118.5+ MB

data.isna().sum()

Email No. 0

the 0

to 0

ect 0

and 0

..

military 0

allowing 0

ff 0

dry 0

Prediction 0

Length: 3002, dtype: int64

data = data.iloc[:, 1:] //HandleNumeric

data.shape

sns.countplot(data = data, x = "Prediction")

plt.show()

A graph with blue and orange squares

Description automatically generated

x = data.iloc[:, :3000].values

y = data.iloc[:, -1].values

(x.shape , y.shape)

x

array([[ 0, 0, 1, ..., 0, 0, 0],

[ 8, 13, 24, ..., 0, 1, 0],

[ 0, 0, 1, ..., 0, 0, 0],

...,

[ 0, 0, 1, ..., 0, 0, 0],

[ 2, 7, 1, ..., 0, 1, 0],

[22, 24, 5, ..., 0, 0, 0]], dtype=int64)

y

array([0, 0, 0, ..., 1, 1, 0], dtype=int64)

#Training \_ Testing data split

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.3, random\_state = 1)

def perform(y\_pred):

    print("Precision : ", precision\_score(y\_test, y\_pred))

    print("Recall : ", recall\_score(y\_test, y\_pred))

    print("Accuracy Score : ", accuracy\_score(y\_test, y\_pred))

    print("F1 Score : ", f1\_score(y\_test, y\_pred))

    print("\n", confusion\_matrix(y\_test, y\_pred))

    print("")

    cm\_display = ConfusionMatrixDisplay(confusion\_matrix = confusion\_matrix(y\_test, y\_pred), display\_labels=['Spam', 'Not Spam'] )

    cm\_display.plot()

    plt.show()

#Logistic regression

model\_lr = LogisticRegression(max\_iter = 700)

model\_lr.fit(x\_train, y\_train)

A close-up of a text

Description automatically generated

y\_pred\_lr = model\_lr.predict(x\_test)

perform(y\_pred\_lr)

Precision : 0.9488888888888889

Recall : 0.9405286343612335

Accuracy Score : 0.9677835051546392

F1 Score : 0.9446902654867257

[[1075 23]

A colorful squares with numbers

Description automatically generated [ 27 427]]

**Practical No – 04**

**Aim –** Customer segmentation in business model based on their demographic, psychographic and behavour data.

**Theory:**

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k- means clustering.

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

In Orange, we can see only one cluster but in python code we can segregate different clusters.

A diagram of a diagram

Description automatically generated

A graph showing a cluster of dots

Description automatically generated with medium confidence

**Code –**

import numpy as np

import pandas as pd

import datetime

import matplotlib

import matplotlib.pyplot as plt

from matplotlib import colors

import seaborn as sns

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from yellowbrick.cluster import KElbowVisualizer

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt, numpy as np

from mpl\_toolkits.mplot3d import Axes3D

from sklearn.cluster import AgglomerativeClustering

from matplotlib.colors import ListedColormap

from sklearn import metrics

#Loading the dataset

data = pd.read\_csv("D:\programs\data\_science\marketing\_campaign.csv" , sep=";")

print("Number of datapoints:", len(data))

data.head()

**A screenshot of a computer

Description automatically generated**

#Information on features

data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2240 entries, 0 to 2239

Data columns (total 29 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 ID 2240 non-null int64

1 Year\_Birth 2240 non-null int64

2 Education 2240 non-null object

3 Marital\_Status 2240 non-null object

4 Income 2216 non-null float64

5 Kidhome 2240 non-null int64

6 Teenhome 2240 non-null int64

7 Dt\_Customer 2240 non-null object

8 Recency 2240 non-null int64

9 MntWines 2240 non-null int64

10 MntFruits 2240 non-null int64

11 MntMeatProducts 2240 non-null int64

12 MntFishProducts 2240 non-null int64

13 MntSweetProducts 2240 non-null int64

14 MntGoldProds 2240 non-null int64

15 NumDealsPurchases 2240 non-null int64

16 NumWebPurchases 2240 non-null int64

17 NumCatalogPurchases 2240 non-null int64

18 NumStorePurchases 2240 non-null int64

19 NumWebVisitsMonth 2240 non-null int64

...

27 Z\_Revenue 2240 non-null int64

28 Response 2240 non-null int64

dtypes: float64(1), int64(25), object(3)

memory usage: 507.6+ KB

#To remove the NA values

data = data.dropna()

print("The total number of data-points after removing the rows with missing values are:", len(data))

The total number of data-points after removing the rows with missing values are: 2216

data["Dt\_Customer"] = pd.to\_datetime(data["Dt\_Customer"])

dates = []

for i in data["Dt\_Customer"]:

    i = i.date()

    dates.append(i)

#Dates of the newest and oldest recorded customer

print("The newest customer's enrolment date in therecords:",max(dates))

print("The oldest customer's enrolment date in the records:",min(dates))

The newest customer's enrolment date in therecords: 2014-06-29

The oldest customer's enrolment date in the records: 2012-07-30

#Created a feature "Customer\_For"

days = []

d1 = max(dates) #taking it to be the newest customer

for i in dates:

    delta = d1 - i

    days.append(delta)

data["Customer\_For"] = days

data["Customer\_For"] = pd.to\_numeric(data["Customer\_For"], errors="coerce")

# Calculate age

data["Age"] = 2021 - data["Year\_Birth"]

# Calculate total spending

data["Spent"] = data["MntWines"] + data["MntFruits"] + data["MntMeatProducts"] + data["MntFishProducts"] + data["MntSweetProducts"] + data["MntGoldProds"]

# Group living status

data["Living\_With"] = data["Marital\_Status"].replace({"Married": "Partner", "Together": "Partner", "Absurd": "Alone", "Widow": "Alone", "YOLO": "Alone", "Divorced": "Alone", "Single": "Alone"})

# Calculate family size

data["Children"] = data["Kidhome"] + data["Teenhome"]

data["Family\_Size"] = data["Living\_With"].replace({"Alone": 1, "Partner": 2}) + data["Children"]

# Determine if the customer is a parent

data["Is\_Parent"] = np.where(data.Children > 0, 1, 0)

# Update education levels

data["Education"] = data["Education"].replace({"Basic": "Undergraduate", "2n Cycle": "Undergraduate", "Graduation": "Graduate", "Master": "Postgraduate", "PhD": "Postgraduate"})

# Rename columns

data = data.rename(columns={"MntWines": "Wines", "MntFruits": "Fruits", "MntMeatProducts": "Meat", "MntFishProducts": "Fish", "MntSweetProducts": "Sweets", "MntGoldProds": "Gold"})

# Drop unnecessary columns

to\_drop = ["Marital\_Status", "Dt\_Customer", "Z\_CostContact", "Z\_Revenue", "Year\_Birth", "ID"]

data = data.drop(to\_drop, axis=1)

sns.set(rc={"axes.facecolor":"#FFF9ED","figure.facecolor":"#FFF9ED"})

pallet = ["#682F2F", "#9E726F", "#D6B2B1", "#B9C0C9", "#9F8A78", "#F3AB60"]

cmap = colors.ListedColormap(["#682F2F", "#9E726F", "#D6B2B1", "#B9C0C9", "#9F8A78", "#F3AB60"])

To\_Plot = [ "Income", "Recency", "Customer\_For", "Age", "Spent", "Is\_Parent"]

print("Reletive Plot Of Some Selected Features: A Data Subset")

plt.figure()

sns.pairplot(data[To\_Plot], hue= "Is\_Parent",palette= (["#682F2F","#F3AB60"]))

# Filter out outliers

data = data[(data["Age"] < 90)]

data = data[(data["Income"] < 600000)]

# Encode categorical columns

s = (data.dtypes == 'object')

object\_cols = list(s[s].index)

LE = LabelEncoder()

for i in object\_cols:

    data[i] = data[[i]].apply(LE.fit\_transform)

ds = data.copy()

cols\_del = ['AcceptedCmp3', 'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1', 'AcceptedCmp2', 'Complain', 'Response']

ds = ds.drop(cols\_del, axis=1)

scaler = StandardScaler()

scaler.fit(ds)

scaled\_ds = pd.DataFrame(scaler.transform(ds), columns=ds.columns)

pca = PCA(n\_components=3)

pca.fit(scaled\_ds)

PCA\_ds = pd.DataFrame(pca.transform(scaled\_ds), columns=(["col1", "col2", "col3"]))

PCA\_ds.describe().T

A screenshot of a calculator

Description automatically generated

#A 3D Projection Of Data In The Reduced Dimension

x =PCA\_ds["col1"]

y =PCA\_ds["col2"]

z =PCA\_ds["col3"]

#To plot

fig = plt.figure(figsize=(10,8))

ax = fig.add\_subplot(111, projection="3d")

ax.scatter(x,y,z, c="maroon", marker="o" )

ax.set\_title("A 3D Projection Of Data In The Reduced Dimension")

A graph of red dots

Description automatically generatedplt.show()

AC = AgglomerativeClustering(n\_clusters=4)

yhat\_AC = AC.fit\_predict(PCA\_ds)

PCA\_ds["Clusters"] = yhat\_AC

data["Clusters"] = yhat\_AC

#Plotting the clusters

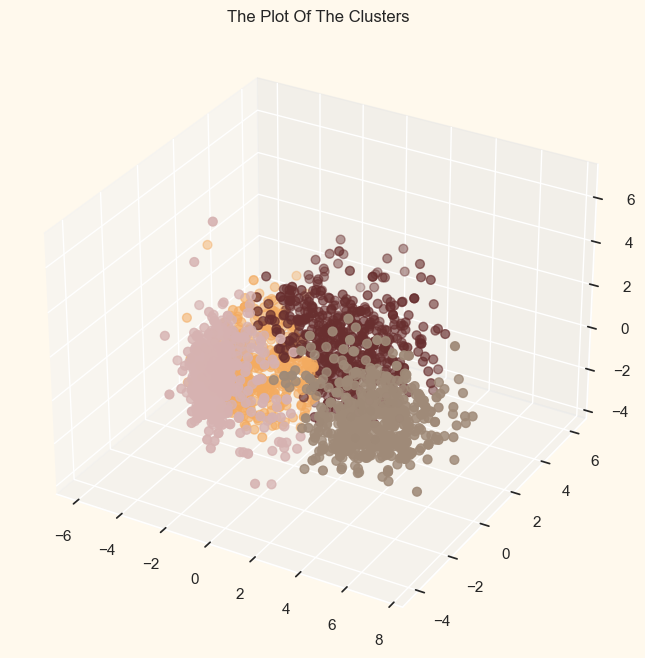
fig = plt.figure(figsize=(10,8))

ax = plt.subplot(111, projection='3d', label="bla")

ax.scatter(x, y, z, s=40, c=PCA\_ds["Clusters"], marker='o', cmap = cmap )

ax.set\_title("The Plot Of The Clusters")

plt.show()



pal = ["#682F2F", "#B9C0C9", "#9F8A78", "#F3AB60"]

pl = sns.countplot(x=data["Clusters"], palette=pal)

pl.set\_title("Distribution Of The Clusters")

plt.show()

A graph of a distribution of the clusters

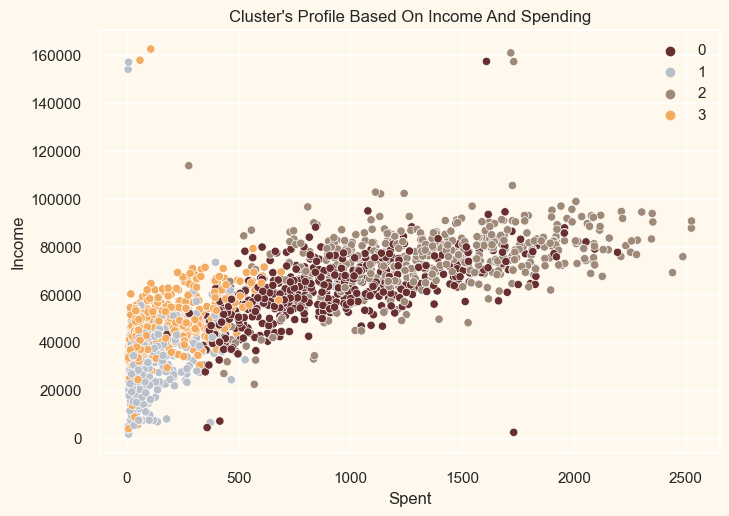
Description automatically generated

pl = sns.scatterplot(data=data, x=data["Spent"], y=data["Income"], hue=data["Clusters"], palette=pal)

pl.set\_title("Cluster's Profile Based On Income And Spending")

plt.legend()

plt.show()



**Practical No – 05**

**Aim -** Construct a recommendation system based on the customer transaction using Association rule mining.

**Theory:**

Recommendation systems aim to recommend content, products and services to users using some methods and algorithms. While content is plentiful, users’ interests tend to be more customized to the entire content set and differ from person to person. In order not to get lost in this abundant cluster and to reach the desired personalized service according to the field of interest, various filters should be made. These filters and algorithms appear as “Recommendation Systems”. It is used in many areas such as dating applications, e- commerce sites, social media channels etc.

**For calculation :**

Support(X, Y) = Freq(X, Y) / Total Transaction Confidence(X, Y) = Freq(X, Y) / Freq(X)

Lift = Support(X, Y) / (Support(X) \* Support(Y))

Association rule mining finds interesting associations and relationships among large sets of data items. This rule shows how frequently a itemset occurs in a transaction.

A diagram of a data flow

Description automatically generated

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

data\_movies = pd.read\_csv('D:\programs\data\_science\ml-latest-small\movies.csv')

data\_tages = pd.read\_csv('D:\programs\data\_science\ml-latest-small/tags.csv')

data\_ratings = pd.read\_csv('D:\programs\data\_science\ml-latest-small/ratings.csv')

data\_movies.head()

A screenshot of a computer

Description automatically generated

data\_tages

A screenshot of a computer

Description automatically generated

A screenshot of a table

Description automatically generateddata\_ratings

merge = data\_movies.merge(data\_tages,on = 'movieId',how = 'inner')

merge.drop(columns=['tag','timestamp','genres'],inplace=True)

merge

A screenshot of a screen shot

Description automatically generated

merge\_list = merge.groupby(by = ["userId"])["title"].apply(list).reset\_index()

merge\_list.head()

A screenshot of a computer

Description automatically generated

merge\_list = merge\_list["title"].tolist()

merge\_list[0:3]

[['Step Brothers (2008)',

'Step Brothers (2008)',

'Step Brothers (2008)',

'Warrior (2011)',

'Warrior (2011)',

'Warrior (2011)',

'Wolf of Wall Street, The (2013)',

'Wolf of Wall Street, The (2013)',

'Wolf of Wall Street, The (2013)'],

['Departed, The (2006)'],

["Carlito's Way (1993)",

"Carlito's Way (1993)",

"Carlito's Way (1993)",

'Godfather: Part II, The (1974)',

'Godfather: Part II, The (1974)',

'Pianist, The (2002)',

'Pianist, The (2002)',

'Lucky Number Slevin (2006)',

'Fracture (2007)',

'Fracture (2007)',

'Fracture (2007)',

'Upside Down: The Creation Records Story (2010)',

'Upside Down: The Creation Records Story (2010)',

'Upside Down: The Creation Records Story (2010)',

'Just Eat It: A Food Waste Story (2014)',

'Just Eat It: A Food Waste Story (2014)']]

from mlxtend.preprocessing import TransactionEncoder

te = TransactionEncoder()

te\_ary = te.fit(merge\_list).transform(merge\_list)

df = pd.DataFrame(te\_ary, columns=te.columns\_)

df.head()

A close-up of a document

Description automatically generated

df.shape

(58, 1572)

#apriori

from mlxtend.frequent\_patterns import apriori

%time

apriori\_frequent\_itemsets = apriori(df, min\_support=0.01,use\_colnames=True,max\_len=2)

apriori\_frequent\_itemsets['itemsets'].apply(lambda x: len(x)).value\_counts()

itemsets

2 774986

1 1572

Name: count, dtype: int64

#fpgrowth

from mlxtend.frequent\_patterns import fpgrowth

%time

fpgrowth\_frequent\_itemsets = fpgrowth(df, min\_support=0.01, use\_colnames=True,max\_len=2)

fpgrowth\_frequent\_itemsets.head()

A screenshot of a report

Description automatically generated

fpgrowth\_frequent\_itemsets['itemsets'].apply(lambda x: len(x)).value\_counts()

itemsets

2 774986

1 1572

Name: count, dtype: int64

#rules

from mlxtend.frequent\_patterns import association\_rules

rules = association\_rules(fpgrowth\_frequent\_itemsets,metric="lift",min\_threshold=0.01)

rules

A table with text on it

Description automatically generated

rules[rules["antecedents"].apply(lambda x: "Inception (2010)" in str(x))].sort\_values(ascending=False,by='lift')

#A less messy way to group the top 10 related movies to the selected one.

#We can see here the selected movie inception has top values of lift on related movie due to the same director in them (Dunkirk)

rules[rules["antecedents"].apply(lambda x: "Inception (2010)" in str(x))].groupby(

    ['antecedents', 'consequents'])[['lift']].max().sort\_values(ascending=False,by='lift').head(10)

A screenshot of a computer

Description automatically generated

rules[rules["antecedents"].apply(lambda x: "Inception (2010)" in str(x))].groupby(

    ['antecedents', 'consequents'])[['confidence']].max().sort\_values(ascending=False,

                                                                      by='confidence').head(10).plot(kind='bar').invert\_xaxis()

plt.title('Top movies that are likley to be watched with inception');

A blue and white striped graph

Description automatically generated

**Practical No – 06**

**Aim –** Behavioural Analysis of customer for any online purchase model.

**Theory:**

Customer behavior doesn’t describe who is shopping in your stores but how they’re shopping in your stores. It reviews factors like shopping frequency, product preferences, and how your marketing, sales, and service offers are perceived. Understanding these details helps businesses communicate with customers in a productive and delightful way. A customer behavior analysis is a qualitative and quantitative observation of how customers interact with your company.

A diagram of a diagram

Description automatically generated

**Python –**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read\_csv("D:\programs\data\_science\Social\_Network\_Ads.csv")

X = dataset.iloc[:, [2, 3]].values

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

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=0)

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(n\_estimators=10, criterion='entropy', random\_state=0)

classifier.fit(X\_train, y\_train)

y\_pred = classifier.predict(X\_test)

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

from matplotlib.colors import ListedColormap

# Create a colormap with color names

colors = ['red', 'green']

# Plotting the decision boundaries for the training set

X\_set, y\_set = X\_train, y\_train

X1, X2 = np.meshgrid(np.arange(start=X\_set[:, 0].min() - 1, stop=X\_set[:, 0].max() + 1, step=0.01),

                     np.arange(start=X\_set[:, 1].min() - 1, stop=X\_set[:, 1].max() + 1, step=0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha=0.75, cmap=ListedColormap(colors))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c=ListedColormap(colors)(i), label=j)

plt.title('Random Forest Classification (Training set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

A diagram of a training set

Description automatically generated

# Plotting the decision boundaries for the test set

X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start=X\_set[:, 0].min() - 1, stop=X\_set[:, 0].max() + 1, step=0.01),

                     np.arange(start=X\_set[:, 1].min() - 1, stop=X\_set[:, 1].max() + 1, step=0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

             alpha=0.75, cmap=ListedColormap(colors))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

    plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c=ListedColormap(colors)(i), label=j)

plt.title('Random Forest Classification (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

A green and red chart with red dots

Description automatically generated

**R Programming -**

# Load required libraries

library(randomForest)

library(ggplot2)

library(caret)

# Load the dataset

dataset <- read.csv("D:/programs/data\_science/Social\_Network\_Ads.csv")

# Split the dataset into features (X) and target variable (y)

X <- dataset[, c("Age", "EstimatedSalary")]

y <- dataset[, "Purchased"]

# Split the dataset into training and test sets

set.seed(0)  # For reproducibility

splitIndex <- createDataPartition(y, p = 0.75, list = FALSE)

X\_train <- X[splitIndex, ]

y\_train <- y[splitIndex]

X\_test <- X[-splitIndex, ]

y\_test <- y[-splitIndex]

# Standardize the features

X\_train <- scale(X\_train)

X\_test <- scale(X\_test)

# Create a random forest classifier

classifier <- randomForest(x = X\_train, y = y\_train, ntree = 10, mtry = 2, nodesize = 1)

# Predict on the test set

y\_pred <- predict(classifier, newdata = X\_test)

# Confusion Matrix

cm <- table(Actual = y\_test, Predicted = y\_pred)

# Plot the decision boundary for the test set

grid <- expand.grid(Age = seq(min(X\_test[, 1]) - 1, max(X\_test[, 1]) + 1, by = 0.01),

                    EstimatedSalary = seq(min(X\_test[, 2]) - 1, max(X\_test[, 2]) + 1, by = 0.01))

grid$y\_pred <- predict(classifier, newdata = grid)

ggplot() +

  geom\_contour(data = grid, aes(x = Age, y = EstimatedSalary, z = y\_pred),

               bins = 2, alpha = 0.75, fill = c("red", "green")) +

  geom\_point(data = dataset, aes(x = Age, y = EstimatedSalary, color = as.factor(Purchased))) +

  scale\_color\_manual(values = c("red", "green")) +

  xlim(min(X\_test[, 1]) - 1, max(X\_test[, 1]) + 1) +

  ylim(min(X\_test[, 2]) - 1, max(X\_test[, 2]) + 1) +

  labs(title = "Random Forest Classification (Test set)",

       x = "Age", y = "Estimated Salary") +

  theme\_minimal()

A screenshot of a computer

Description automatically generated

**Practical No – 07**

**Aim -** Agricultural data analysis for yield prediction and crop selection on Indian terrain data set.

**Theory:**

Agricultural data analysis is important for our country, a country which relies on farming a lot. It involves estimating the number of crops that will be produced in a given area based on various factors such as soil type, weather conditions, and crop management practices.

Agricultural data analysis helps predict crop yields for different regions and seasons. By analyzing historical data along with current weather and soil conditions, farmers can make informed decisions about planting and harvesting times. Therefore, we can develop a model. The model will be trained on historical data. It learns the relationships between input features (e.g., weather conditions, soil characteristics) and crop yields. The goal is to create a predictive model that captures these relationships.

**A diagram of a diagram

Description automatically generated**

**Python -**

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import MinMaxScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

# Load the dataset

df = pd.read\_csv(r'D:\programs\data\_science\Crop\_recommendation.csv')

# Display dataset summary statistics

df\_description = df.describe()

# Create a boxplot

sns.boxplot(y='label', x='ph', data=df)

# Convert the 'label' column to categorical

c = df['label'].astype('category')

# Create a mapping of category codes to labels

targets = dict(enumerate(c.cat.categories))

# Add a 'target' column with categorical codes

df['target'] = c.cat.codes

# Split the data into features (X) and target (y)

y = df['target']

X = df[['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall']]

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=1)

# Scale the features using Min-Max scaling

scaler = MinMaxScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# K-Nearest Neighbors Classifier

knn = KNeighborsClassifier()

knn.fit(X\_train\_scaled, y\_train)

knn\_score = knn.score(X\_test\_scaled, y\_test)

# K-Nearest Neighbors Classifier with different k values

k\_range = range(1, 11)

knn\_scores = []

for k in k\_range:

    knn = KNeighborsClassifier(n\_neighbors=k)

    knn.fit(X\_train\_scaled, y\_train)

    knn\_scores.append(knn.score(X\_test\_scaled, y\_test))

# Decision Tree Classifier

decision\_tree = DecisionTreeClassifier(random\_state=42)

decision\_tree.fit(X\_train, y\_train)

decision\_tree\_score = decision\_tree.score(X\_test, y\_test)

# Random Forest Classifier

random\_forest = RandomForestClassifier(max\_depth=4, n\_estimators=100, random\_state=42)

random\_forest.fit(X\_train, y\_train)

rf\_train\_accuracy = random\_forest.score(X\_train, y\_train)

rf\_test\_accuracy = random\_forest.score(X\_test, y\_test)

# Gradient Boosting Classifier

gradient\_boosting = GradientBoostingClassifier()

gradient\_boosting.fit(X\_train, y\_train)

gb\_accuracy = gradient\_boosting.score(X\_test, y\_test)

**A graph with different colored squares

Description automatically generated**

# Print results

print("Dataset Summary Statistics:")

print(df\_description)

print("\nK-Nearest Neighbors Classifier Accuracy:", knn\_score)

print("\nDecision Tree Classifier Accuracy:", decision\_tree\_score)

print("\nRandom Forest Classifier - Training Accuracy:", rf\_train\_accuracy)

print("Random Forest Classifier - Test Accuracy:", rf\_test\_accuracy)

print("\nGradient Boosting Classifier Accuracy:", gb\_accuracy)

Dataset Summary Statistics:

N P K temperature humidity \

count 2200.000000 2200.000000 2200.000000 2200.000000 2200.000000

mean 50.551818 53.362727 48.149091 25.616244 71.481779

std 36.917334 32.985883 50.647931 5.063749 22.263812

min 0.000000 5.000000 5.000000 8.825675 14.258040

25% 21.000000 28.000000 20.000000 22.769375 60.261953

50% 37.000000 51.000000 32.000000 25.598693 80.473146

75% 84.250000 68.000000 49.000000 28.561654 89.948771

max 140.000000 145.000000 205.000000 43.675493 99.981876

ph rainfall

count 2200.000000 2200.000000

mean 6.469480 103.463655

std 0.773938 54.958389

min 3.504752 20.211267

25% 5.971693 64.551686

50% 6.425045 94.867624

75% 6.923643 124.267508

max 9.935091 298.560117

K-Nearest Neighbors Classifier Accuracy: 0.9781818181818182

Decision Tree Classifier Accuracy: 0.9872727272727273

Random Forest Classifier - Training Accuracy: 0.9715151515151516

Random Forest Classifier - Test Accuracy: 0.9727272727272728

Gradient Boosting Classifier Accuracy: 0.9945454545454545

# Plot K-Nearest Neighbors Classifier Accuracy vs. k values

plt.xlabel('k')

plt.ylabel('Accuracy')

plt.scatter(k\_range, knn\_scores)

plt.vlines(k\_range, 0, knn\_scores, linestyle="dashed")

A graph of a number of points

Description automatically generatedplt.ylim(0.96, 0.99)

plt.xticks([i for i in range(1, 11)])

plt.show()

R Programming –

# Load required libraries

library(readr)

library(ggplot2)

library(caret)

library(randomForest)

library(e1071)  # Required for scaling

# Load the dataset

df <- read.csv("D:/programs/data\_science/Crop\_recommendation.csv")

# Display dataset summary statistics

df\_description <- summary(df)

# Create a boxplot

ggplot(df, aes(x = ph, y = label)) +

  geom\_boxplot() +

  labs(x = "pH", y = "Label") +

  theme\_minimal()

# Convert the 'label' column to categorical

df$label <- as.factor(df$label)

# Split the data into features (X) and target (y)

y <- df$label

X <- df[, c("N", "P", "K", "temperature", "humidity", "ph", "rainfall")]

# Split the data into training and test sets

set.seed(1)  # For reproducibility

splitIndex <- createDataPartition(y, p = 0.75, list = FALSE)

X\_train <- X[splitIndex, ]

y\_train <- y[splitIndex]

X\_test <- X[-splitIndex, ]

y\_test <- y[-splitIndex]

# Scale the features using Min-Max scaling

scaler <- preProcess(X\_train, method = c("range"))

X\_train\_scaled <- predict(scaler, X\_train)

X\_test\_scaled <- predict(scaler, X\_test)

# K-Nearest Neighbors Classifier

knn <- train(X\_train\_scaled, y\_train, method = "knn")

knn\_score <- knn$results$Accuracy[1]

# K-Nearest Neighbors Classifier with different k values

k\_range <- 1:10

knn\_scores <- numeric(length(k\_range))

for (k in k\_range) {

  knn\_model <- train(X\_train\_scaled, y\_train, method = "knn", tuneGrid = data.frame(k = k))

  knn\_scores[k] <- knn\_model$results$Accuracy

}

# Decision Tree Classifier

decision\_tree <- train(X\_train, y\_train, method = "rpart")

decision\_tree\_score <- decision\_tree$results$Accuracy[1]

# Random Forest Classifier

random\_forest <- randomForest(X\_train, y\_train, ntree = 100, mtry = 4)

rf\_train\_accuracy <- sum(random\_forest$confusion[1, ]) / sum(random\_forest$confusion)

rf\_test\_accuracy <- sum(predict(random\_forest, newdata = X\_test) == y\_test) / length(y\_test)

# Gradient Boosting Classifier

gradient\_boosting <- train(X\_train, y\_train, method = "gbm")

gb\_accuracy <- gradient\_boosting$results$Accuracy[1]

# Print results

cat("Dataset Summary Statistics:\n")

print(df\_description)

cat("\nK-Nearest Neighbors Classifier Accuracy:", knn\_score)

cat("\nDecision Tree Classifier Accuracy:", decision\_tree\_score)

cat("\nRandom Forest Classifier - Training Accuracy:", rf\_train\_accuracy)

cat("\nRandom Forest Classifier - Test Accuracy:", rf\_test\_accuracy)

cat("\nGradient Boosting Classifier Accuracy:", gb\_accuracy)

# Plot K-Nearest Neighbors Classifier Accuracy vs. k values

plot(k\_range, knn\_scores, type = "b", xlab = "k", ylab = "Accuracy", xlim = c(1, 10), ylim = c(0.96, 0.99))

abline(h = max(knn\_scores), col = "red", lty = 2)

A screenshot of a computer code

Description automatically generated

A diagram of a graph

Description automatically generated

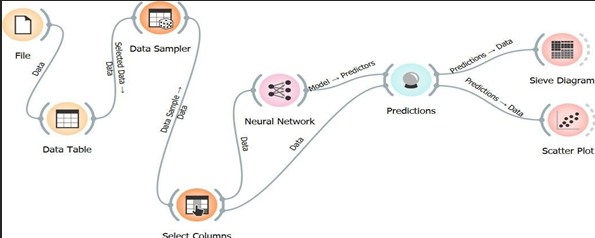
**Practical No - 08**

**Aim -** Develop a business model to predict the trend in Investment and Funding.

**Theory:**

Predicting trends in investment and funding using machine learning (ML) involves analyzing historical data and identifying patterns that can inform future investment decisions, if data includes time series information (e.g., funding rounds over time), perform time series analysis to identify trends, seasonality, and cyclical patterns. Time series decomposition and autocorrelation analysis can provide insights.

For time series forecasting, models like ARIMA, LSTM, or Prophet may be appropriate. For regression tasks (predicting funding amounts), consider linear regression, random forests, gradient boosting, or neural networks.



A screen shot of a computer

Description automatically generated

**Python –**

import numpy as np

import pandas as pd

import os

import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.layers import Dense, Dropout, LSTM

# Load the dataset

google\_stock\_data = pd.read\_csv('D:\programs\data\_science\google.csv')

google\_stock\_data = google\_stock\_data[['Date', 'Open', 'Close']]

google\_stock\_data['Date'] = pd.to\_datetime(google\_stock\_data['Date'].apply(lambda x: x.split()[0]))

google\_stock\_data.set\_index('Date', drop=True, inplace=True)

# Scale the data

MMS = MinMaxScaler()

google\_stock\_data[google\_stock\_data.columns] = MMS.fit\_transform(google\_stock\_data)

# Split the data into training and test sets

training\_size = round(len(google\_stock\_data) \* 0.80)

train\_data = google\_stock\_data[:training\_size]

test\_data = google\_stock\_data[training\_size:]

# Function to create sequences and labels

def create\_sequence(dataset):

    sequences = []

    labels = []

    start\_idx = 0

    for stop\_idx in range(50, len(dataset)):  # Selecting 50 rows at a time

        sequences.append(dataset.iloc[start\_idx:stop\_idx])

        labels.append(dataset.iloc[stop\_idx])

        start\_idx += 1

    return (np.array(sequences), np.array(labels))

# Create sequences and labels for training and test data

train\_seq, train\_label = create\_sequence(train\_data)

test\_seq, test\_label = create\_sequence(test\_data)

# Create the LSTM model

model = Sequential()

model.add(LSTM(units=50, return\_sequences=True, input\_shape=(train\_seq.shape[1], train\_seq.shape[2])))

model.add(Dropout(0.1))

model.add(LSTM(units=50))

model.add(Dense(2))

model.compile(loss='mean\_squared\_error', optimizer='adam', metrics=['mean\_absolute\_error'])

model.summary()

Model: "sequential\_2"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

lstm\_4 (LSTM) (None, 50, 50) 10600

dropout\_2 (Dropout) (None, 50, 50) 0

lstm\_5 (LSTM) (None, 50) 20200

dense\_2 (Dense) (None, 2) 102

=================================================================

Total params: 30902 (120.71 KB)

Trainable params: 30902 (120.71 KB)

Non-trainable params: 0 (0.00 Byte)

# Train the model

model.fit(train\_seq, train\_label, epochs=80, validation\_data=(test\_seq, test\_label), verbose=1)

Epoch 1/80

107/107 [==============================] - 7s 37ms/step - loss: 6.4582e-04 - mean\_absolute\_error: 0.0125 - val\_loss: 0.0045 - val\_mean\_absolute\_error: 0.0429

Epoch 2/80

107/107 [==============================] - 3s 29ms/step - loss: 6.7242e-05 - mean\_absolute\_error: 0.0058 - val\_loss: 0.0052 - val\_mean\_absolute\_error: 0.0463

Epoch 3/80

107/107 [==============================] - 3s 28ms/step - loss: 6.3580e-05 - mean\_absolute\_error: 0.0057 - val\_loss: 0.0054 - val\_mean\_absolute\_error: 0.0470

Epoch 4/80

107/107 [==============================] - 3s 29ms/step - loss: 5.8760e-05 - mean\_absolute\_error: 0.0054 - val\_loss: 0.0038 - val\_mean\_absolute\_error: 0.0381

Epoch 5/80

107/107 [==============================] - 3s 29ms/step - loss: 5.8066e-05 - mean\_absolute\_error: 0.0054 - val\_loss: 0.0045 - val\_mean\_absolute\_error: 0.0427

Epoch 6/80

107/107 [==============================] - 3s 29ms/step - loss: 5.6263e-05 - mean\_absolute\_error: 0.0054 - val\_loss: 0.0045 - val\_mean\_absolute\_error: 0.0439

Epoch 7/80

107/107 [==============================] - 3s 29ms/step - loss: 5.2187e-05 - mean\_absolute\_error: 0.0052 - val\_loss: 0.0040 - val\_mean\_absolute\_error: 0.0424

Epoch 8/80

107/107 [==============================] - 3s 30ms/step - loss: 5.3339e-05 - mean\_absolute\_error: 0.0051 - val\_loss: 0.0029 - val\_mean\_absolute\_error: 0.0352

Epoch 9/80

107/107 [==============================] - 3s 30ms/step - loss: 5.1563e-05 - mean\_absolute\_error: 0.0051 - val\_loss: 0.0033 - val\_mean\_absolute\_error: 0.0393

Epoch 10/80

107/107 [==============================] - 3s 30ms/step - loss: 4.7292e-05 - mean\_absolute\_error: 0.0048 - val\_loss: 0.0023 - val\_mean\_absolute\_error: 0.0301

Epoch 11/80

107/107 [==============================] - 3s 31ms/step - loss: 4.3797e-05 - mean\_absolute\_error: 0.0046 - val\_loss: 0.0027 - val\_mean\_absolute\_error: 0.0349

Epoch 12/80

107/107 [==============================] - 3s 32ms/step - loss: 3.6734e-05 - mean\_absolute\_error: 0.0042 - val\_loss: 0.0022 - val\_mean\_absolute\_error: 0.0307

Epoch 13/80

...

Epoch 79/80

107/107 [==============================] - 4s 33ms/step - loss: 1.1311e-05 - mean\_absolute\_error: 0.0024 - val\_loss: 0.0044 - val\_mean\_absolute\_error: 0.0341

Epoch 80/80

107/107 [==============================] - 4s 34ms/step - loss: 1.1177e-05 - mean\_absolute\_error: 0.0023 - val\_loss: 0.0060 - val\_mean\_absolute\_error: 0.0441

# Predict on the test set

test\_predicted = model.predict(test\_seq)

# Inverse transform the predictions

test\_inverse\_predicted = MMS.inverse\_transform(test\_predicted)

# Create a DataFrame with actual and predicted values

gs\_slic\_data = google\_stock\_data.iloc[-len(test\_inverse\_predicted):].copy()

gs\_slic\_data[['Open\_Predicted', 'Close\_Predicted']] = test\_inverse\_predicted

# Plot the actual vs. predicted open price

gs\_slic\_data[['Open', 'Open\_Predicted']].plot(figsize=(10, 6))

plt.xticks(rotation=45)

plt.xlabel('Date', size=15)

plt.ylabel('Stock Price', size=15)

plt.title('Actual vs Predicted for Open Price', size=15)

plt.show()

**A graph showing a line of orange and white

Description automatically generated with medium confidence**

# Plot the actual vs. predicted close price

gs\_slic\_data[['Close', 'Close\_Predicted']].plot(figsize=(10, 6))

plt.xticks(rotation=45)

plt.xlabel('Date', size=15)

plt.ylabel('Stock Price', size=15)

plt.title('Actual vs Predicted for Close Price', size=15)

plt.show()

A graph showing a line of orange and white

Description automatically generated with medium confidence