

Configuration Manual

MSc Research Project
Masters In Computer Science Data Analytics

Mysura Reddy Polam
Student ID:X21143323

School of Computing
National College of Ireland

Supervisor: Catherine Mulwa

National College of Ireland
MSc Project Submission Sheet
School of Computing



Student Name: Mysura Reddy Polam

Student ID: X21143323

Programme: Msc in Data Analytics

Year: 2022

Module: Research Project

Lecturer: Catherine Mulwa

Submission

Due Date: 15/12/2022

Project Title: Sales and Logistics Analysis in E-Commerce using Machine Learning Models

Word Count: XXX

Page Count:

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Polam Mysura Reddy

Date: 15/12/2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only

Signature:

Date:

Penalty Applied (if applicable):

Sales and Logistics Analysis in E-commerce using Machine Learning models

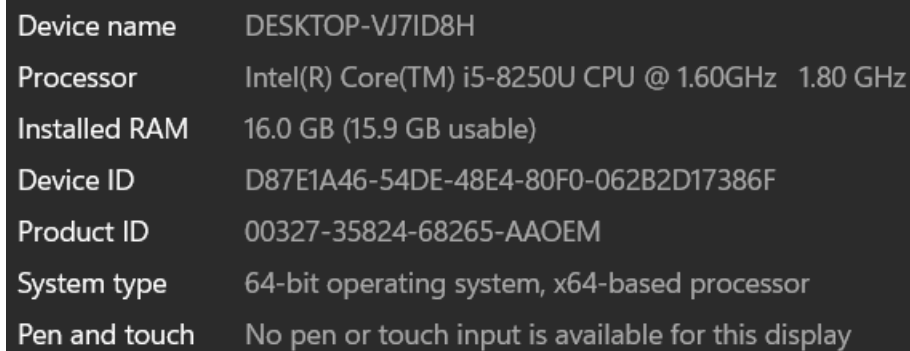
Mysura Reddy Polam
Student ID: X21143323

1 Overview

Your first section. Change the header and label to something appropriate.

2 System Introduction

2.1 Hardware



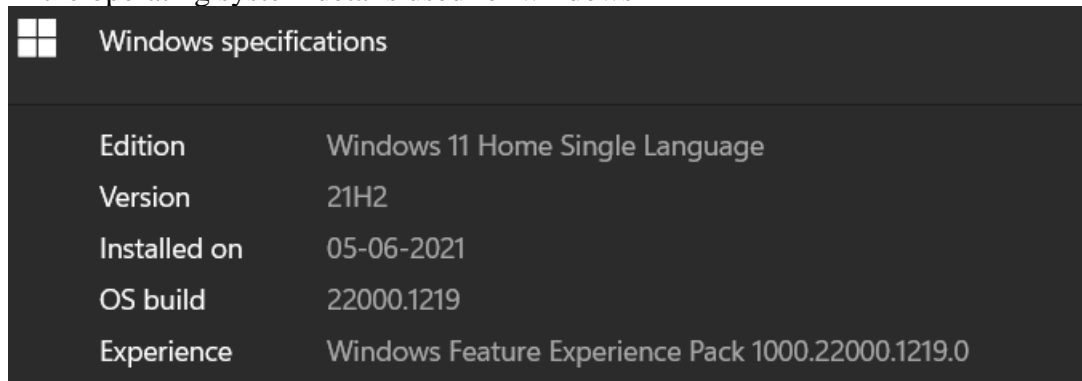
Device name	DESKTOP-VJ7ID8H
Processor	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80 GHz
Installed RAM	16.0 GB (15.9 GB usable)
Device ID	D87E1A46-54DE-48E4-80F0-062B2D17386F
Product ID	00327-35824-68265-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Figure 1: Hardware Details

2.2 Software Installed

The varies software requirements are details below.

- In the operating system details used for windows 11



Windows specifications	
Edition	Windows 11 Home Single Language
Version	21H2
Installed on	05-06-2021
OS build	22000.1219
Experience	Windows Feature Experience Pack 1000.22000.1219.0

Figure 2: Software Details

- Download and install Anaconda.



Figure 3: Download & Install Anaconda

- Downloading Power Bi

Microsoft Power BI Desktop

Microsoft Power BI Desktop is a companion product to app.powerbi.com.

Version: 2.111.590.0 64-bit (November, 2022)

User ID: 688e395c-0255-4366-9315-38df308da188

Session ID: 5ab14e10-be94-4edc-adf5-3772fe47fd51

Copy session diagnostics to clipboard

[Copy](#)

[Privacy Statement](#)

Figure 4: Power Bi Desktop Version Configurations

- Installing PostgreSQL and PG Admin

About pgAdmin 4



Version	6.13
Application Mode	Desktop
Current User	pgadmin4@pgadmin.org
NW.js Version	0.62.2
Browser	Chromium 99.0.4844.84
Operating System	Windows-10-10.0.22000-SP0
pgAdmin Database File	C:\Users\Mani\AppData\Roaming\pgadmin\pgadmin4.db
Log File	C:\Users\Mani\AppData\Roaming\pgadmin\pgadmin4.log

Server Configuration

[Copy](#)

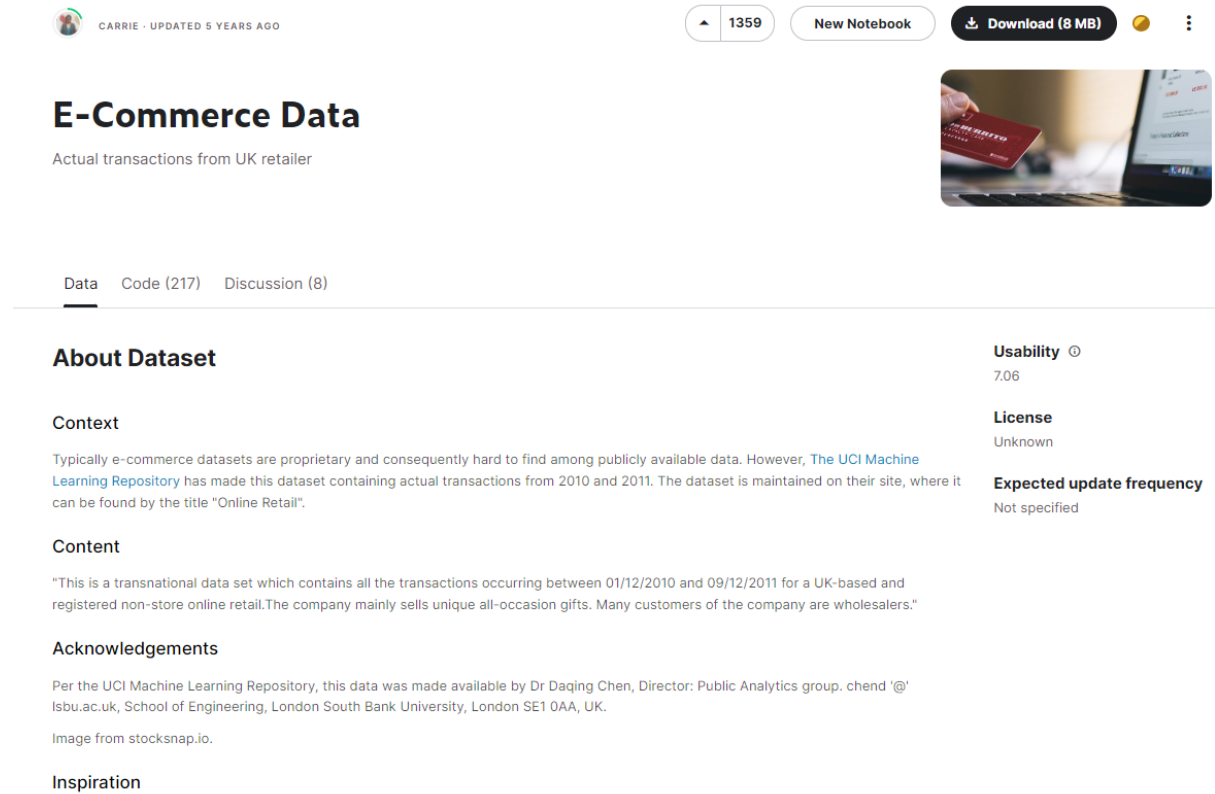
```
ALLOW_SAVE_PASSWORD = True
ALLOW_SAVE_TUNNEL_PASSWORD = False
APP_COPYRIGHT = "Copyright (C) 2013 - 2022, The pgAdmin Development Team"
APP_ICON = "pg-icon"
APP_NAME = "pgAdmin 4"
APP_RELEASE = 6
APP_REVISION = 13
APP_SUFFIX = ""
APP_VERSION = "6.13"
APP_VERSION_EXTN = ('.css', '.js', '.html', '.svg', '.png', '.gif', '.ico')
```

Figure 5: Pg Admin Configuration

3 Implementation and Results

3.1 Dataset

The data is downloaded from Kaggle



E-Commerce Data
Actual transactions from UK retailer

Download (8 MB)

About Dataset

Context
Typically e-commerce datasets are proprietary and consequently hard to find among publicly available data. However, [The UCI Machine Learning Repository](#) has made this dataset containing actual transactions from 2010 and 2011. The dataset is maintained on their site, where it can be found by the title "Online Retail".

Content
"This is a transnational data set which contains all the transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail. The company mainly sells unique all-occasion gifts. Many customers of the company are wholesalers."

Acknowledgements
Per the UCI Machine Learning Repository, this data was made available by Dr Daqing Chen, Director: Public Analytics group. chend '@' lsbu.ac.uk, School of Engineering, London South Bank University, London SE1 0AA, UK.
Image from stocksnap.io.

Usability 7.06

License Unknown

Expected update frequency Not specified

Figure 6: Kaggle dataset

3.2 Installing the Required Packages

Importing all the libraries in python as shown in

```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import pyplot as plt
import seaborn as sns
import warnings
from operator import attrgetter
import datetime as dt
import matplotlib.colors as mcolors
from IPython.display import display
from lazypredict.Supervised import LazyRegressor
from sklearn.utils import shuffle
from sklearn.cluster import KMeans
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
import psycpg2
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import mean_absolute_error

warnings.filterwarnings("ignore")
%matplotlib inline

```

Figure 7: Importing library

3.3 Load Dataset

- Loading dataset to sql after downloading from Kaggle
Data set from Kaggle is loaded into sql

Import/Export data - table 'data'

General Options Columns

Import/Export

Filename

Format

Encoding

Figure 8: Importing from kaggle

Import/Export data - table 'data'

GeneralOptionsColumns

Columns to import

InvoiceNo x

StockCode x

Description x

Quantity x

InvoiceDate x

UnitPrice x

CustomerID x

Country x

An optional list of columns to be copied. If no column list is specified, all columns of the table will be copied.

NOT NULL columns

Not null columns...

Do not match the specified column values against the null string. In the default case where the null string is empty, this means that empty values will be read as zero-length strings rather than nulls, even when they are not quoted. This option is allowed only in import, and only when using CSV format.

i

?

Close

Reset

OK

Figure 9: Loading data into sql

- Creating additional data in SQL

```

select a.*, trunc(random() * 6 + 0) as requested_days , trunc(random() * 10 + 0) as delivered_days from test2 a

select * from data

update test2
set delivered_days=trunc(random() * 9 + 0)
where "InvoiceNo" is not Null

alter table test2
rename to data

```

Figure 10: Adding more data using sql

- Loading data from SQL to python by giving username and password while creating database.

```

import psycopg2
hostname='localhost'
database='ECOM'
username='postgres'
pwd='123'
port_id=5432
conn=None
cur=None
try:
    conn=psycopg2.connect(
        host=hostname,
        dbname=database,
        user=username,
        password=pwd,
        port=port_id)
    cur=conn.cursor()
    cur.execute('SELECT * FROM data')
    cur.fetchall()
    conn.commit()

except Exception as error:
    print(error)

```

```

cur.execute('SELECT * FROM data')
cur.fetchall()
conn.commit()

```

```

data=pd.read_sql('SELECT * FROM DATA',conn)
data

```

Figure 11: Importing data from sql data

3.4 Execute Pre-processing

- Checking & removing null values

```

#1. Checking Null values
data.isnull().sum()

```

```

InvoiceNo      0
StockCode      0
Description    1454
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID    135080
Country        0
requested_days  0
delivered_days  0
dtype: int64

```

```

# deleting Description Name which is having nullvalues
data.dropna(subset=['Description'],how='any',inplace= True)

```

```

data['CustomerID'] = pd.to_numeric(data['CustomerID'], errors='coerce')
data["CustomerID"] = data["CustomerID"].fillna(value=data["CustomerID"].mean())

```

Figure 12: removing null values

- Checking datatypes


```
# checking the type of dataset
data.dtypes

InvoiceNo      object
StockCode      object
Description     object
Quantity       int64
InvoiceDate     object
UnitPrice      float64
CustomerID     float64
Country        object
requested_days  int64
delivered_days  int64
dtype: object
```

Figure 13: checking datatypes

- Removal of special characters from numeric columns

```
data['InvoiceNo']=data.InvoiceNo.astype(str).str.replace('C', '')
data['InvoiceNo']=data.InvoiceNo.astype(str).str.replace('A', '')
data.drop(data.index[data['Country'] == 'Unspecified'], inplace=True)
data['InvoiceDate']=pd.to_datetime(data["InvoiceDate"])
data['InvoiceNo']=data['InvoiceNo'].astype(int)
data['CustomerID']=data['CustomerID'].astype(float)
data['Quantity']=data['Quantity'].astype(float)
data['Quantity'] = data[data['Quantity'] > 0]['Quantity']
data['Quantity'] = data['Quantity'].replace(np.nan, 0)

data.isnull().sum()
```

Figure 14: Removing special characters

- Creating additional column called total price

```
##adding one more column as total price
data['TotalPrice'] = data['UnitPrice'] * data['Quantity']

data.isnull().sum()
```

Figure 15: adding more data

- Checking for outliers

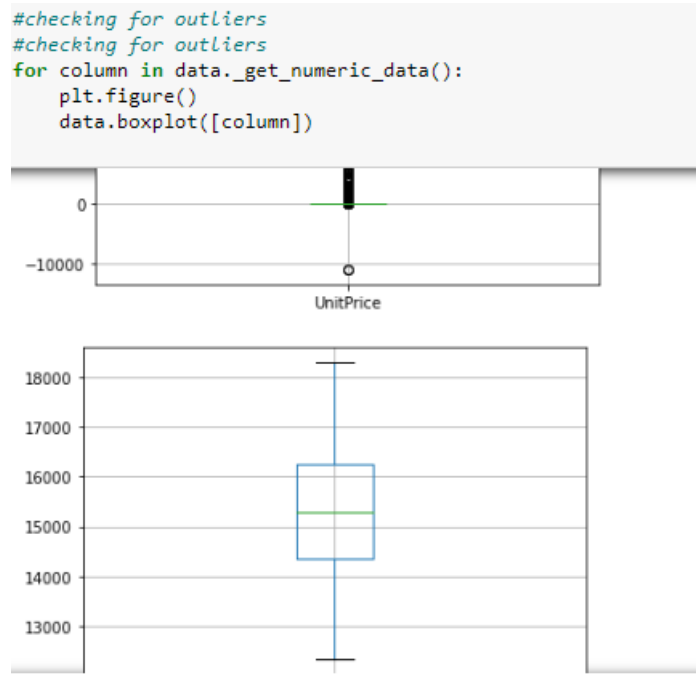


Figure 16: checking for outliers

- Creating additional data for logistics performance

```
data['logistic_performance_indicator']=data['requested_days']/data['delivered_days']*100
data.dropna(subset=['logistic_performance_indicator'],how='any',inplace=True)

# Replace infinite updated data with nan
data.replace([np.inf, -np.inf], np.nan, inplace=True)
# Drop rows with NaN
data.dropna(inplace=True)
(data)

data.isnull().sum()
```

Figure 17: Creating logistics performance column

3.5 EDA

- Data analysis in PowerBi



Figure 18: Dashboard for overall dataset

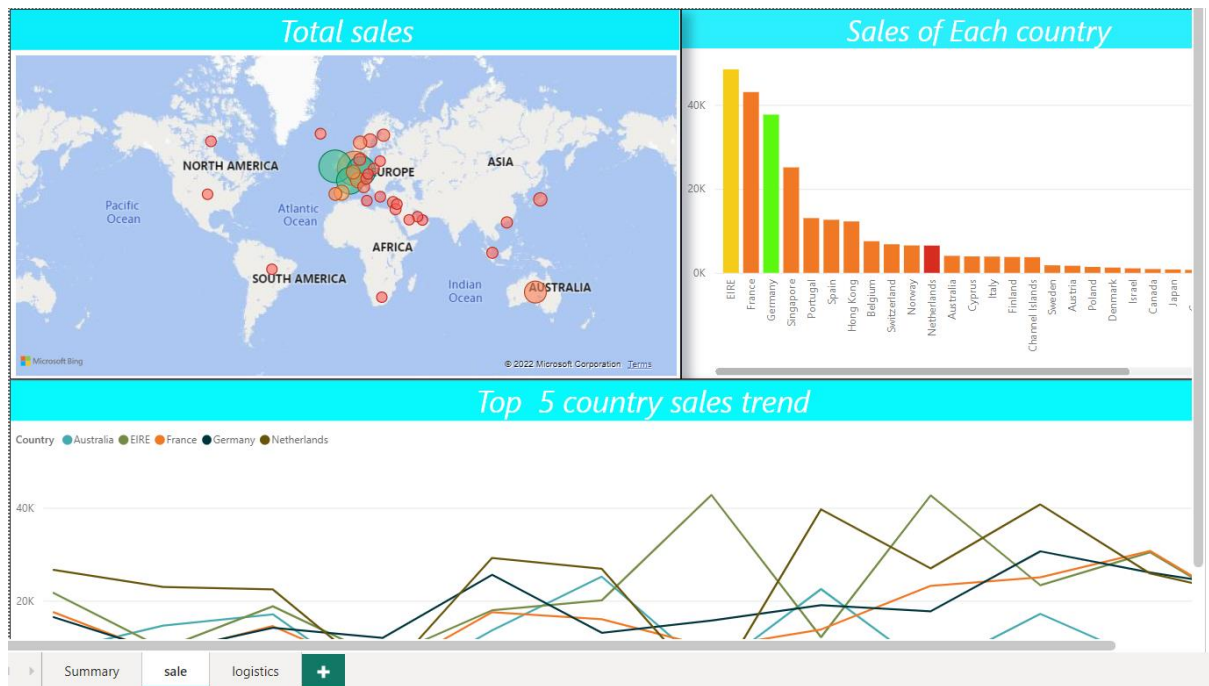


Figure 19: Dashboard for sales

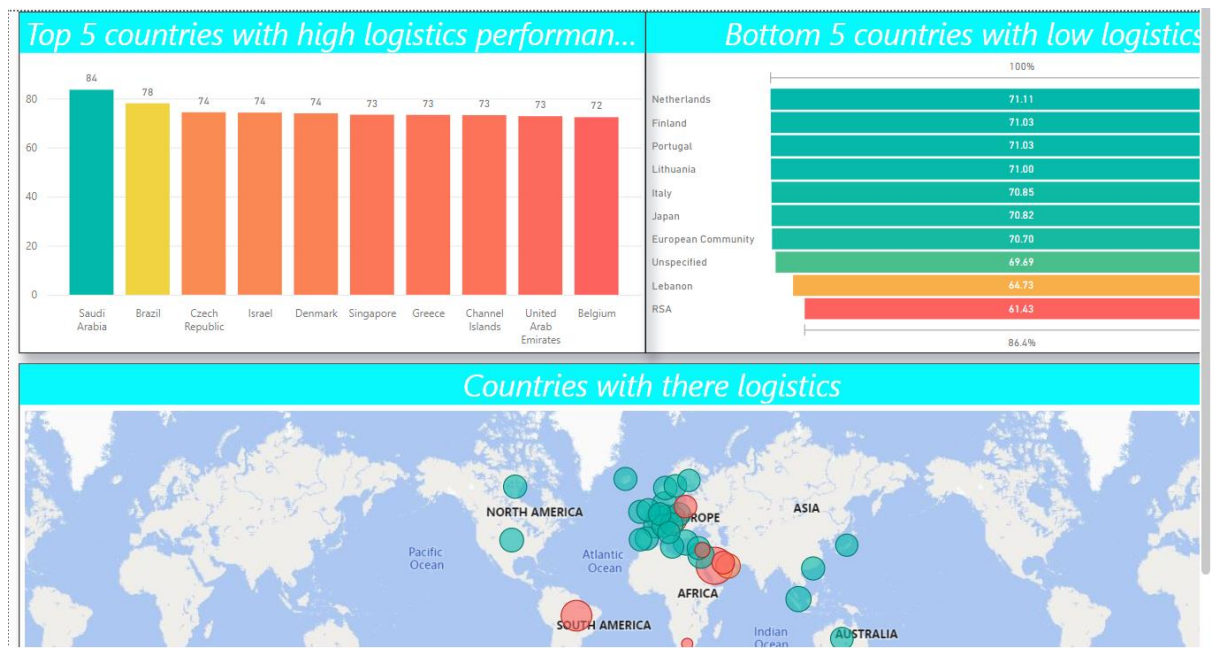


Figure 20: Dashboard for logistics

- Data Visualization in Python

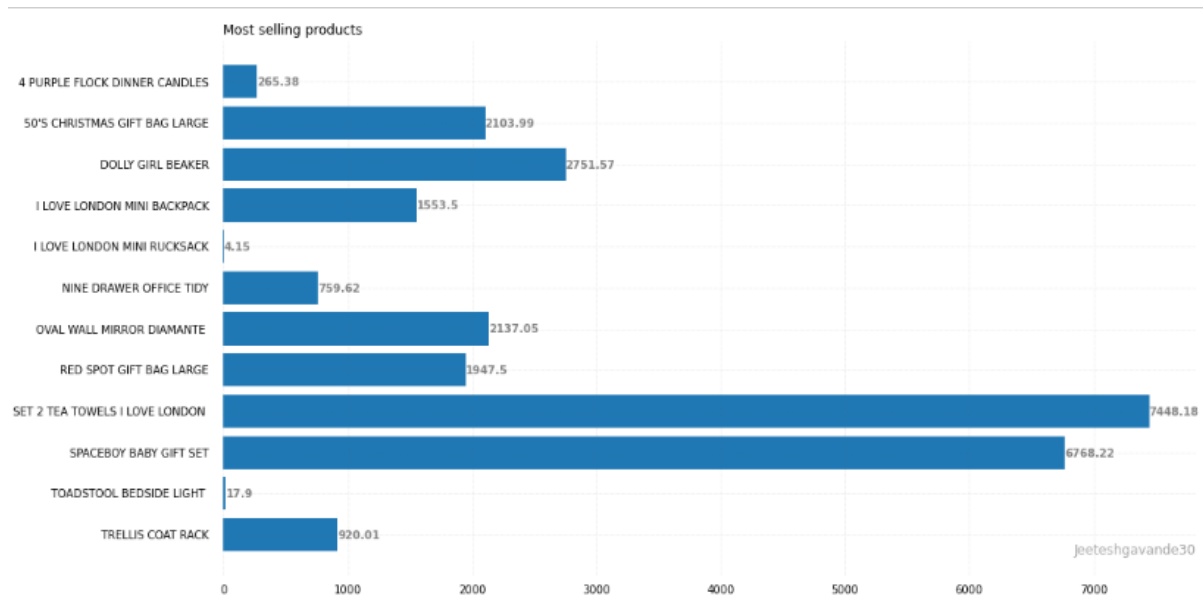


Figure 21: Product visualization in python

3.6 Feature Selection

Feature selections

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.model_selection import train_test_split
```

```
#as we can see in correlation table, experience & current job years columns are
#formula is (value-mean)/standard deviation
columns=["SALE", "Logistic_performance"]
dataPCA=df[columns]
transformedDF=(dataPCA-dataPCA.mean(axis=0))/dataPCA.std()
transformedDF
```

	SALE	Logistic_performance
0	-0.228914	-0.334861
1	-0.173347	-0.238240
2	-0.137530	-0.163825
3	-0.126587	-0.114485
4	-0.231383	2.569347
...
301	3.763611	0.028598
302	5.872086	-0.042857
303	6.176854	0.033693
304	8.787374	-0.002332
305	3.894773	-0.018214

306 rows × 2 columns

```
#applying PCA(principle component analysis) method to transformed DF
pca=PCA(n_components=2)
PC=pca.fit_transform(transformedDF)
principle=pd.DataFrame(data=PC, columns=['PC.SALE', 'PC.Logistic_performance'])
principle.head()
```

Figure 22: Principle component Analysis

```
#changing categorical values into numerical for training & test datasets by LabelEncoder
cat_col=['Country']
label_encoder=LabelEncoder()
for col in cat_col:
    df[col]=label_encoder.fit_transform(df[col])
df['Country']=df['Country']+1
df
```

	Country	Year	Month	Logistic_performance	SALE	PC.SALE	PC.Logistic_performance
0	1	2010	12	77.118347	965.350	-0.074916	-0.398649
1	1	2011	1	79.440901	8265.130	-0.045886	-0.291036
2	1	2011	2	81.229656	12970.380	-0.018593	-0.213090
3	1	2011	3	82.415675	14407.950	0.008558	-0.170464
4	1	2011	4	146.928571	641.000	1.980415	1.653190
...
301	37	2011	8	85.855034	525460.480	-2.641053	2.681496
302	37	2011	9	84.137439	802449.351	-4.182496	4.121887
303	37	2011	10	85.977507	842486.550	-4.343871	4.391520
304	37	2011	11	85.111569	1185428.600	-6.215260	6.211963
305	37	2011	12	84.729799	542691.120	-2.766899	2.741141

306 rows × 7 columns

Figure 23: Categorical changes

3.7 Splitting data

```
#Splitting data
X=df[["Country","Year","Month"]].values
y=df[["SALE"]].values

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

Figure 24: splitting data

```
#square root for sales
sqrt_x= np.sqrt(X+1)
sqrt_y = np.sqrt(y)

scalar = StandardScaler()

scalar.fit(sqrt_x)
scaled_data_sqrt = scalar.transform(sqrt_x)
X_train_sqrt, X_test_sqrt, y_train_sqrt, y_test_sqrt = train_test_split(scaled_data_sqrt , sqrt_y, test_size=0.3, random_state=2)
```

Figure 25: splitting data with squareroot transformation

3.8 Modelling

```
##SALES
from sklearn.linear_model import LinearRegression
from sklearn import metrics

LR=LinearRegression()
LR.fit(X_train,y_train)
y_prediction=LR.predict(X_test)#prediction using test sample

from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.metrics import mean_absolute_error
print('LinearRegression Model validation score for sales prediction:', LR.score(X_test,y_test)*1000)
print('R2 score of LinearRegression model for sales prediction: ', r2_score(y_test, y_prediction)*1000)
print("RMSE value of LinearRegression for sales prediction :",sqrt(mean_squared_error(y_test, y_prediction)))
print("MSE value of LinearRegression for sales prediction :",mean_squared_error(y_test, y_prediction))
print("Mean absolte error of LinearRegression for sales prediction :",mean_absolute_error(y_test, y_prediction))

results_df = pd.DataFrame(data=[["Linear Regression ", *evaluate(y_test, y_prediction) ]],
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])

results_df

#square root for linear regression
lm = LinearRegression()
lm.fit(X_train_sqrt,y_train_sqrt)

print('Training accuracy=',lm.score(X_train_sqrt,y_train_sqrt))
pred = lm.predict(X_test_sqrt)

mae = metrics.mean_absolute_error(y_test_sqrt, pred)
mse = metrics.mean_squared_error(y_test_sqrt, pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test_sqrt, pred))
r2_square = metrics.r2_score(y_test_sqrt, pred)

print('Prediction R-Square accuracy =',r2_square)
print('MAE:', mae)
print('MSE:', mse)
print('RMSE:',rmse)

results_df1 = pd.DataFrame(data=[["Linear Regression sqrt", *evaluate(y_test_sqrt, pred) ]],
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df1, ignore_index=True)
```

Figure 25: Linear regression

2. Decision Tree Regressor

```
##Decision Tree Regressor
from sklearn.tree import DecisionTreeRegressor
DTR=DecisionTreeRegressor()

DTR.fit(X_train,y_train)

y_prediction=DTR.predict(X_test)#prediction using test sample
print('Decision TreeRegression Model validation score for sales prediction:', DTR.score(X_train,y_train))
print('R2 score of Decision TreeRegression model for sales prediction: ', r2_score(y_test, y_prediction))
print("RMSE value of Decision Tree Regression for sales prediction :",sqrt(mean_squared_error(y_test, y_prediction)))
print("MSE value of Decision Tree Regression for sales prediction:",mean_squared_error(y_test, y_prediction))
print("Mean absolute error of Decision Tree Regression for sales prediction",mean_absolute_error(y_test, y_prediction))
results_df2 = pd.DataFrame(data=[["Decision TreeRegression ", *evaluate(y_test, y_prediction) ]],
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])

results_df = results_df.append(results_df2, ignore_index=True)
```

Decision TreeRegression Model validation score for sales prediction: 1.0
R2 score of Decision TreeRegression model for sales prediction: 0.9762917983127251
RMSE value of Decision Tree Regression for sales prediction : 17766.162045842753
MSE value of Decision Tree Regression for sales prediction: 315636513.8391435
Mean absolute error of Decision Tree Regression for sales prediction 6092.688209677419

```
#square root for Decision Tree
DTR = DecisionTreeRegressor()
DTR.fit(X_train_sqrt,y_train_sqrt)

print('Training accuracy=',DTR.score(X_train_sqrt,y_train_sqrt)*100)
pred = DTR.predict(X_test_sqrt)

mae = metrics.mean_absolute_error(y_test_sqrt, pred)
mse = metrics.mean_squared_error(y_test_sqrt, pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test_sqrt, pred))
r2_square = metrics.r2_score(y_test_sqrt, pred)

print('Prediction R-Square accuracy =',r2_square)
print('MAE:', mae)
print('MSE:', mse)
print('RMSE:',rmse)

results_df3 = pd.DataFrame(data=[["DecisionTreeRegressor sqrt", *evaluate(y_test_sqrt, pred) ]],
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df3, ignore_index=True)
D=results_df2.append(results_df3, ignore_index=True)
```

Training accuracy= 100.0
Prediction R-Square accuracy = 0.913302445412232

Figure 26: Decision Tree Regression

3. GradientBoostingRegressor

```
: from sklearn.ensemble import GradientBoostingRegressor

GBR=GradientBoostingRegressor()
GBR.fit(X_train,y_train)
y_prediction=GBR.predict(X_test)#prediction using test sample

print('GradientBoostingRegressor Model validation score for sales prediction:', GBR.score(X_train,y_train))

print('R2 score of GradientBoostingRegressor: ', r2_score(y_test, y_prediction))
print("RMSE value of GradientBoostingRegressor :",sqrt(mean_squared_error(y_test, y_prediction)))
print("MSE value of GradientBoostingRegressor():",mean_squared_error(y_test, y_prediction))
print("Mean absolute error of GradientBoostingRegressor()",mean_absolute_error(y_test, y_prediction))
results_df4 = pd.DataFrame(data=[[" GradientBoostingRegressor ", *evaluate(y_test, y_prediction) ]],
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])

results_df = results_df.append(results_df4, ignore_index=True)
```

GradientBoostingRegressor Model validation score for sales prediction: 0.9993683369663092
R2 score of GradientBoostingRegressor: 0.977344666161835
RMSE value of GradientBoostingRegressor : 17367.189596962122
MSE value of GradientBoostingRegressor(): 301619274.4968294
Mean absolute error of GradientBoostingRegressor() 5742.457655402642

```
: #square root for Decision Tree
GBR=GradientBoostingRegressor()
GBR.fit(X_train_sqrt,y_train_sqrt)

print('Training accuracy=',GBR.score(X_train_sqrt,y_train_sqrt)*100)
pred = GBR.predict(X_test_sqrt)

mae = metrics.mean_absolute_error(y_test_sqrt, pred)
mse = metrics.mean_squared_error(y_test_sqrt, pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test_sqrt, pred))
r2_square = metrics.r2_score(y_test_sqrt, pred)

print('Prediction R-Square accuracy =',r2_square)
print('MAE:', mae)
print('MSE:', mse)
print('RMSE:',rmse)

results_df5 = pd.DataFrame(data=[["GradientBoostingRegressor sqrt", *evaluate(y_test_sqrt, pred) ]],
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df5, ignore_index=True)
G=results_df4.append(results_df5, ignore_index=True)
```

Training accuracy= 98.00139106080262
Prediction R-Square accuracy = 0.9415665957243657

Figure 27: Gradient Boosting Regression

4. BaggingRegressor

```
from sklearn.ensemble import BaggingRegressor

BR=BaggingRegressor()

BR.fit(X_train,y_train)
y_prediction=BR.predict(X_test)#prediction using test sample

print('BaggingRegressor Model validation score for sales prediction:', BR.score(X_train,y_train))

print('R2 score of BaggingRegressor: ', r2_score(y_test, y_prediction))
print("RMSE value of BaggingRegressor :",sqrt(mean_squared_error(y_test, y_prediction)))
print("MSE value of BaggingRegressor:",mean_squared_error(y_test, y_prediction))
print("Mean absolute error of BaggingRegressor",mean_absolute_error(y_test, y_prediction))
results_df6 = pd.DataFrame(data=[[" BaggingRegressor ", *evaluate(y_test, y_prediction) ]],
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df6, ignore_index=True)
```

BaggingRegressor Model validation score for sales prediction: 0.9777153703257783
R2 score of BaggingRegressor: 0.9706351045137956
RMSE value of BaggingRegressor : 19772.36150245107
MSE value of BaggingRegressor: 390946279.3836091
Mean absolute error of BaggingRegressor 6745.297124193548

```
#square root for BaggingRegressor
BR=BaggingRegressor()
BR.fit(X_train_sqrt,y_train_sqrt)

print('Training accuracy=',BR.score(X_train_sqrt,y_train_sqrt)*100)
pred = BR.predict(X_test_sqrt)

mae = metrics.mean_absolute_error(y_test_sqrt, pred)
mse = metrics.mean_squared_error(y_test_sqrt, pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test_sqrt, pred))
r2_square = metrics.r2_score(y_test_sqrt, pred)

print('Prediction R-Square accuracy =',r2_square)
print('MAE:', mae)
print('MSE:', mse)
print('RMSE:',rmse)

results_df7 = pd.DataFrame(data=[["BaggingRegressor sqrt", *evaluate(y_test_sqrt, pred) ]],
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df7, ignore_index=True)
B=results_df6.append(results_df7, ignore_index=True)
```

Training accuracy= 97.11191101651141

Figure 28: Bagging Regression

5. AdaBoostRegressor

```
from sklearn.ensemble import AdaBoostRegressor

ABR=AdaBoostRegressor()

ABR.fit(X_train,y_train)
y_prediction=ABR.predict(X_test)#prediction using test sample

print('AdaBoostRegressor Model validation score for sales prediction:', ABR.score(X_train,y_train))

print('R2 score of AdaBoostRegressor: ', r2_score(y_test, y_prediction))
print("RMSE value of AdaBoostRegressor :",sqrt(mean_squared_error(y_test, y_prediction)))
print("MSE value of AdaBoostRegressor:",mean_squared_error(y_test, y_prediction))
print("Mean absolute error of AdaBoostRegressor",mean_absolute_error(y_test, y_prediction))
results_df8 = pd.DataFrame(data=[[" AdaBoostRegressor ", *evaluate(y_test, y_prediction) ]],
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df8, ignore_index=True)
```

AdaBoostRegressor Model validation score for sales prediction: 0.9965653073285126
R2 score of AdaBoostRegressor: 0.980560006350952
RMSE value of AdaBoostRegressor : 16087.64120316435
MSE value of AdaBoostRegressor: 258812199.48175132
Mean absolute error of AdaBoostRegressor 6596.151258913114

```
#square root for AdaRegressor
ABR=AdaBoostRegressor()
ABR.fit(X_train_sqrt,y_train_sqrt)

print('Training accuracy=',ABR.score(X_train_sqrt,y_train_sqrt)*100)
pred = ABR.predict(X_test_sqrt)

mae = metrics.mean_absolute_error(y_test_sqrt, pred)
mse = metrics.mean_squared_error(y_test_sqrt, pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test_sqrt, pred))
r2_square = metrics.r2_score(y_test_sqrt, pred)

print('Prediction R-Square accuracy =',r2_square)
print('MAE:', mae)
print('MSE:', mse)
print('RMSE:',rmse)

results_df9 = pd.DataFrame(data=[["AdaBoostRegressor sqrt", *evaluate(y_test_sqrt, pred) ]],
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df9, ignore_index=True)
A=results_df8.append(results_df9, ignore_index=True)
```

Training accuracy= 91.93542598031743
Prediction R-Square accuracy = 0.919303610076128
MAE: 36.007000000000000

Figure 29: AdaBoost Regression

6.KNeighborsRegressor

```
from sklearn.neighbors import KNeighborsRegressor

KNN=KNeighborsRegressor()

KNN.fit(X_train,y_train)
y_prediction=KNN.predict(X_test)#prediction using test sample

print('KNeighborsRegressor Model validation score for sales prediction:', KNN.score(X_train,y_train))

print('R2 score of KNeighborsRegressor: ', r2_score(y_test, y_prediction))
print("RMSE value of KNeighborsRegressor :",sqrt(mean_squared_error(y_test, y_prediction)))
print("MSE value of KNeighborsRegressor:",mean_squared_error(y_test, y_prediction))
print("Mean absolute error of KNeighborsRegressor",mean_absolute_error(y_test, y_prediction))
results_df10 = pd.DataFrame(data=[[" KNeighborsRegressor ", *evaluate(y_test, y_prediction) ]],
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df10, ignore_index=True)
```

```
KNeighborsRegressor Model validation score for sales prediction: 0.8630708271996068
R2 score of KNeighborsRegressor: 0.8051057372897185
RMSE value of KNeighborsRegressor : 50938.2287886249
MSE value of KNeighborsRegressor: 2594703152.122295
Mean absolute error of KNeighborsRegressor 13634.617774193552
```

```
#square root for KNN
KNN=KNeighborsRegressor()
KNN.fit(X_train_sqrt,y_train_sqrt)

print('Training accuracy=',KNN.score(X_train_sqrt,y_train_sqrt)*100)
pred = KNN.predict(X_test_sqrt)

mae = metrics.mean_absolute_error(y_test_sqrt, pred)
mse = metrics.mean_squared_error(y_test_sqrt, pred)
rmse = np.sqrt(metrics.mean_squared_error(y_test_sqrt, pred))
r2_square = metrics.r2_score(y_test_sqrt, pred)

print('Prediction R-Square accuracy =',r2_square)
print('MAE:', mae)
print('MSE:', mse)
print('RMSE:',rmse)

results_df11 = pd.DataFrame(data=[["KNeighborsRegressor sqrt", *evaluate(y_test_sqrt, pred) ]],
                           columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df = results_df.append(results_df11, ignore_index=True)
K=results_df10.append(results_df11, ignore_index=True)
```

```
Training accuracy= 38.398242521133184
```

Figure 30: KNeighbor Regression

3.9 Results

results_df

	Model	MAE	MSE	RMSE	R2 Square
0	Linear Regression	59784.77	12525174522.17	111915.93	0.08
1	Linear Regression sqrt	94.85	36310.94	190.55	0.05
2	Decision TreeRegression	6092.69	315636513.84	17766.16	0.98
3	DecisionTreeRegressor sqrt	35.59	3323.79	57.65	0.91
4	GradientBoostingRegressor	5742.46	301619274.50	17367.19	0.98
5	GradientBoostingRegressor sqrt	26.22	2240.21	47.33	0.94
6	BaggingRegressor	6745.30	390946279.38	19772.36	0.97
7	BaggingRegressor sqrt	31.84	2896.76	53.82	0.92
8	AdaBoostRegressor	6596.15	258812199.48	16087.64	0.98
9	AdaBoostRegressor sqrt	37.79	3146.33	56.09	0.92
10	KNeighborsRegressor	13634.62	2594703152.12	50938.23	0.81
11	KNeighborsRegressor sqrt	84.88	28436.85	168.63	0.26
12	AdaBoostRegressor sqrt	36.90	3093.72	55.62	0.92

```
import seaborn as sns
fig = plt.figure(figsize =(15, 5))

sns.barplot(x="R2 Square", y="Model", data=results_df)
plt.xlabel('Accuracy')
```

Text(0.5, 0, 'Accuracy')

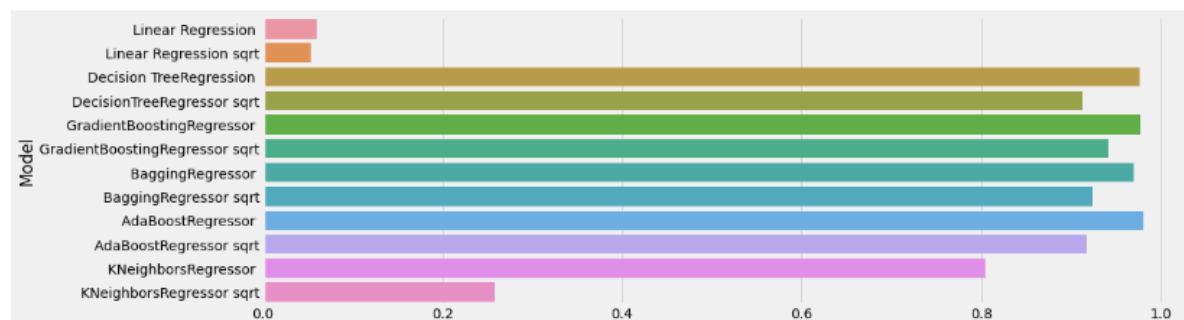


Figure 31: Results for sales prediction

```

##Logistics
A=df[["Country","Year","Month","PC.SALE"]].values
b=df[["Logistic_performance"]].values
A_train, A_test, b_train, b_test = train_test_split( A, b, test_size=0.2, random_state=42)
LR1=LinearRegression()
LR1.fit(A_train,b_train)
b_prediction=LR1.predict(A_test)#prediction using test sample

print('LinearRegression Model validation score for logistic performance prediction:', LR1.score(A_test,b_test)*100)
print('R2 score of LinearRegression for logistic performance prediction: ', r2_score(b_test, b_prediction)*100)
print("RMSE value of LinearRegression for logistic performance prediction:",sqrt(mean_squared_error(b_test, b_prediction)))
print("MSE value of LinearRegression for logistic performance prediction:",mean_squared_error(b_test, b_prediction))
print("Mean absolute error of LinearRegression for logistic performance prediction:",mean_absolute_error(b_test, b_prediction))
results_df_logis = pd.DataFrame(data=[["Linear Regression_logistics ", *evaluate(b_test, b_prediction) ]],
                                columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df_logis

LinearRegression Model validation score for logistic performance prediction: 47.80524354062721
R2 score of LinearRegression for logistic performance prediction: 47.80524354062721
RMSE value of LinearRegression for logistic performance prediction: 15.283022865835143
MSE value of LinearRegression for logistic performance prediction: 233.57078791763982
Mean absolute error of LinearRegression for logistic performance prediction: 10.064666093957092

```

	Model	MAE	MSE	RMSE	R2 Square
0	Linear Regression_logistics	10.06	233.57	15.28	0.48

```

##decision tree
DTR1=DecisionTreeRegressor()
DTR1.fit(A_train,b_train)
b_prediction=DTR1.predict(A_test)#prediction using test sample
print('Decision TreeRegression Model validation score for logistic performance prediction:',DTR1.score(A_train,b_train))
print('R2 score of Decision TreeRegression model for logistic performance prediction: ', r2_score(b_test, b_prediction))
print("RMSE value of Decision TreeRegression model for logistic performance prediction:",sqrt(mean_squared_error(b_test, b_prediction)))
print("MSE value of Decision TreeRegression model for logistic performance prediction:",mean_squared_error(b_test, b_prediction))
print("Mean absolute error of Decision TreeRegression model for logistic performance prediction:",mean_absolute_error(b_test, b_prediction))
results_df_logis1 = pd.DataFrame(data=[["Decision TreeRegression_logistics ", *evaluate(b_test, b_prediction) ]],
                                columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df_logis = results_df_logis.append(results_df_logis1, ignore_index=True)

```

```

Decision TreeRegression Model validation score for logistic performance prediction: 1.0
R2 score of Decision TreeRegression model for logistic performance prediction: 0.9905614680446949
RMSE value of Decision TreeRegression model for logistic performance prediction: 2.0551714131606165
MSE value of Decision TreeRegression model for logistic performance prediction: 4.223729537472605
Mean absolute error of Decision TreeRegression model for logistic performance prediction: 1.2770580041826773

```

Figure 32: Linear and Decision Tree Regression

```

#gradient
GBR1=GradientBoostingRegressor()
GBR1.fit(A_train,b_train)
b_prediction=GBR1.predict(A_test)#prediction using test sample
print('GradientBoostingRegressor Model validation score for logistic performance prediction:',GBR1.score(A_train,b_train))
print('R2 score of GradientBoostingRegressor model for logistic performance prediction: ', r2_score(b_test, b_prediction))
print("RMSE value of GradientBoostingRegressor model for logistic performance prediction:",sqrt(mean_squared_error(b_test, b_prediction)))
print("MSE value of GradientBoostingRegressor model for logistic performance prediction:",mean_squared_error(b_test, b_prediction))
print("Mean absolute error of GradientBoostingRegressor model for logistic performance prediction:",mean_absolute_error(b_test, b_prediction))
results_df_logis2 = pd.DataFrame(data=[["GradientBoostingRegressor_logistics ", *evaluate(b_test, b_prediction) ]],
                                columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df_logis = results_df_logis.append(results_df_logis2, ignore_index=True)

```

```

GradientBoostingRegressor Model validation score for logistic performance prediction: 0.9991969663892237
R2 score of GradientBoostingRegressor model for logistic performance prediction: 0.9925437823812469
RMSE value of GradientBoostingRegressor model for logistic performance prediction: 1.8266490384895775
MSE value of GradientBoostingRegressor model for logistic performance prediction: 3.336646709814898
Mean absolute error of GradientBoostingRegressor model for logistic performance prediction: 1.1448978522695836

```

```

#bagging
BR1=BaggingRegressor()
BR1.fit(A_train,b_train)
b_prediction=BR1.predict(A_test)#prediction using test sample
print('BaggingRegressor Model validation score for logistic performance prediction:',BR1.score(A_train,b_train))
print('R2 score of BaggingRegressor model for logistic performance prediction: ', r2_score(b_test, b_prediction))
print("RMSE value of BaggingRegressor model for logistic performance prediction:",sqrt(mean_squared_error(b_test, b_prediction)))
print("MSE value of BaggingRegressor model for logistic performance prediction:",mean_squared_error(b_test, b_prediction))
print("Mean absolute error of BaggingRegressor model for logistic performance prediction:",mean_absolute_error(b_test, b_prediction))
results_df_logis3 = pd.DataFrame(data=[[" BaggingRegressor ", *evaluate(b_test, b_prediction) ]],
                                columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df_logis = results_df_logis.append(results_df_logis3, ignore_index=True)

```

```

BaggingRegressor Model validation score for logistic performance prediction: 0.9686813408591127
R2 score of BaggingRegressor model for logistic performance prediction: 0.978852080133003
RMSE value of BaggingRegressor model for logistic performance prediction: 3.076306854719316
MSE value of BaggingRegressor model for logistic performance prediction: 9.46366386439305
Mean absolute error of BaggingRegressor model for logistic performance prediction: 1.5129388544086515

```

Figure 33: Gradient & Bagging Regression

```

from sklearn.ensemble import AdaBoostRegressor

ABR=AdaBoostRegressor()

ABR.fit(A_train,b_train)
b_prediction=ABR.predict(A_test)#prediction using test sample

print('AdaBoostRegressor Model validation score for logistic performance prediction:', ABR.score(A_train,b_train))

print('R2 score of AdaBoostRegressor: ', r2_score(b_test, b_prediction))
print("RMSE value of AdaBoostRegressor :",sqrt(mean_squared_error(b_test, b_prediction)))
print("MSE value of AdaBoostRegressor:",mean_squared_error(b_test, b_prediction))
print("Mean absolute error of AdaBoostRegressor",mean_absolute_error(b_test, b_prediction))
results_df_logis4 = pd.DataFrame(data=[[" AdaBoostRegressor ", *evaluate(b_test, b_prediction) ]],
                                columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df_logis = results_df_logis.append(results_df_logis4, ignore_index=True)

```

AdaBoostRegressor Model validation score for logistic performance prediction: 0.9910356486483552
 R2 score of AdaBoostRegressor: 0.9865746315601962
 RMSE value of AdaBoostRegressor : 2.4510881576544343
 MSE value of AdaBoostRegressor: 6.007833156593809
 Mean absolute error of AdaBoostRegressor 1.9521591541677739

```

from sklearn.neighbors import KNeighborsRegressor

KNN=KNeighborsRegressor()

KNN.fit(A_train,b_train)
b_prediction=KNN.predict(A_test)#prediction using test sample

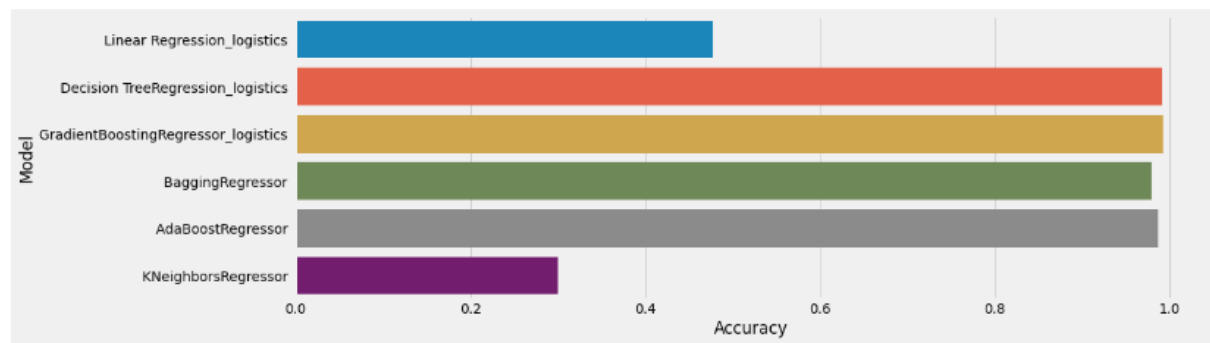
print('KNeighborsRegressor Model validation score for logistic performance prediction:', KNN.score(A_train,b_train))

print('R2 score of KNeighborsRegressor: ', r2_score(b_test, b_prediction))
print("RMSE value of KNeighborsRegressor :",sqrt(mean_squared_error(b_test, b_prediction)))
print("MSE value of KNeighborsRegressor:",mean_squared_error(b_test, b_prediction))
print("Mean absolute error of KNeighborsRegressor",mean_absolute_error(b_test, b_prediction))
results_df_logis5 = pd.DataFrame(data=[[" KNeighborsRegressor ", *evaluate(b_test, b_prediction) ]],
                                columns=['Model', 'MAE', 'MSE', 'RMSE', 'R2 Square'])
results_df_logis = results_df_logis.append(results_df_logis5, ignore_index=True)

```

KNeighborsRegressor Model validation score for logistic performance prediction: 0.4702490179600902
 R2 score of KNeighborsRegressor: 0.29981561519449584
 RMSE value of KNeighborsRegressor : 17.701172554168792
 MSE value of KNeighborsRegressor: 313.3315097924585
 Mean absolute error of KNeighborsRegressor 12.140655699935738

Figure 34: AdaBoost & KNeighbor Regression



results_df_logis

	Model	MAE	MSE	RMSE	R2 Square
0	Linear Regression_logistics	10.06	233.57	15.28	0.48
1	Decision TreeRegression_logistics	1.28	4.22	2.06	0.99
2	GradientBoostingRegressor_logistics	1.14	3.34	1.83	0.99
3	BaggingRegressor	1.51	9.46	3.08	0.98
4	AdaBoostRegressor	1.95	6.01	2.45	0.99
5	KNeighborsRegressor	12.14	313.33	17.70	0.30

Figure 35: Results for logistics performance