



MACHINE LEARNING PROJECT REPORT

on

BIGMART SALES

Submitted by

LALITH KUMAR

Registration Number: 12009352

Program Name: B. tech Data Science (ML and AI)

Under the Guidance of

Mr. Ved Prakash Chaubey

School of Computer Science & Engineering

Lovely Professional University, Phagwara

I, **LALITH KUMAR**, certify that this project is my own work, based on my personal study and/or research and that I have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. I also certify that this project has not previously been submitted for assessment in any academic capacity, and that I have not copied in part or whole or otherwise plagiarised the work of other persons. I confirm that I have identified and declared all possible conflicts that I may have.

Date: 03/05/2023

Acknowledgement

I would like to express my gratitude to **Mr. Ved Prakash Chaubey** my project supervisor, for their guidance and support throughout the project. I would also like to thank **Lovely Professional University** for providing me with the necessary resources and infrastructure to complete this project.

Big Mart Sales

Platform: This dataset was chosen from Google Kaggle. A well-known website for all the data scientists.

Introduction: This is a Big Mart Sales dataset, a kind of a simulation where you perform extensive data analysis to deliver insights on how the company can increase its profits while minimizing the losses. The dataset provides the product details and the outlet information of the products purchased with their sales value split into a train set and test set.

Train set: It contains 8523 rows.

- It contains the item information with sales value in the Train dataset.

Test set: It contains 5681 rows.

- It contains the Item outlet combinations for which sales need to be forecasted.

Keywords: Big market Sales analysis, Machine Learning, Exploratory Data Analysis, Training the data, Testing the data, Prediction of the data, Regression Models.

Abstract: The Big Mart Sales contains the information of the products which have collected in the year of 2013 sales data for 1559 products across 10 stores in different cities.

Also, certain attributes of each product and store have been defined. And our aim is to build a predictive model and predict the sales of each product at a particular outlet. Using this model, the data will try to predict the output that gives the rise in their sales. The Train set contains 12 columns and Test set contains 11 columns.

It is commonly used for data analysis and visualization, as well as for machine learning and predictive modelling.

Description of the Data:

It is a Data set that contains the overview of the product. It 8523 rows and 12 columns -

- ProductID – Unique ID for the products.
- Weight – Weight of the products.
- FatContent – Fat content present in the product.
- Visibility - Percentage of total display area of all products in a store allocated to particular product.
- ProductType – Category of the product.
- MRP – Maximum Retail Price of the product.
- OutletID – Unique ID for the store.
- EstablishmentYear – Year of establishment of the outlets.
- OutletSize – The area covered of the store.
- LocationType – The type of city where the outlet is located.
- OutletType – The type of the outlet whether it is a grocery store or super market.
- OutletSales – Sales of the products in the outlet.

Methodology: We used a machine-learning approach to predict the price of used mobile phones. First, we performed data pre-processing by handling missing values, coding categorical variables, and scaling numeric variables. We then split the dataset into training and test sets. We trained several machine learning models, including linear regression, decision trees, random forests, and gradient boosting, and evaluated their performance using the mean square error (MSE), root mean square error (RMSE), and R-squared (R²) metrics.

Data Collection:

We have collected the data securely in accordance with an agreed methodology. The procedure for the collected data may differ from client to client and is dependent on the type, quantity, availability and need of data.

Data Cleaning and Processing:

The processed data is sent into a cleaning process so that to find the data is segregated properly or not.

And empty values in the data is filled with the aggregate numerical so that it will become easy for prediction.

Data Prediction:

Using XG boost regression and Linear regression and by finding the R Squared value we will calculate our predictive analysis.

And later they will be evaluated using the data.

Data Visualization Process:

Data Analysed is then further picturized to the customer providing insights of the data and how the variables are classified.

And in this we will see graphs like -

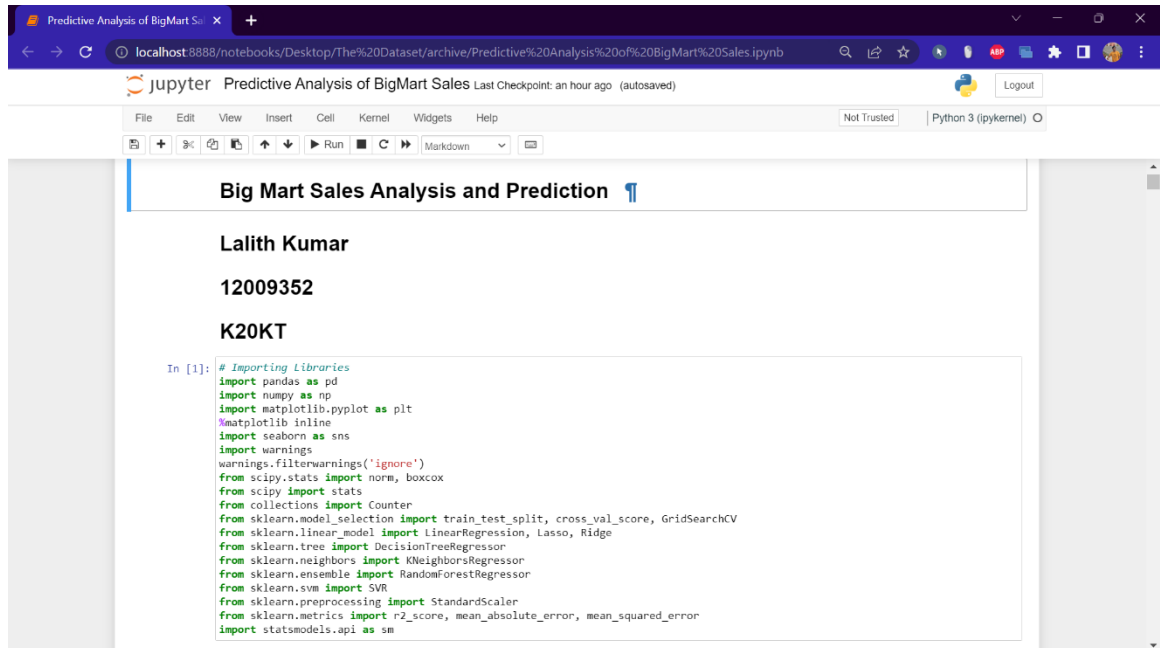
- Count Plot
- Scatter Plot
- Distplot

Corelation graphs which shows the insights of how each variable is connected to another variable.

Distplot contains the combination of both -

- Hist (Histogram)
- Dist (Density)

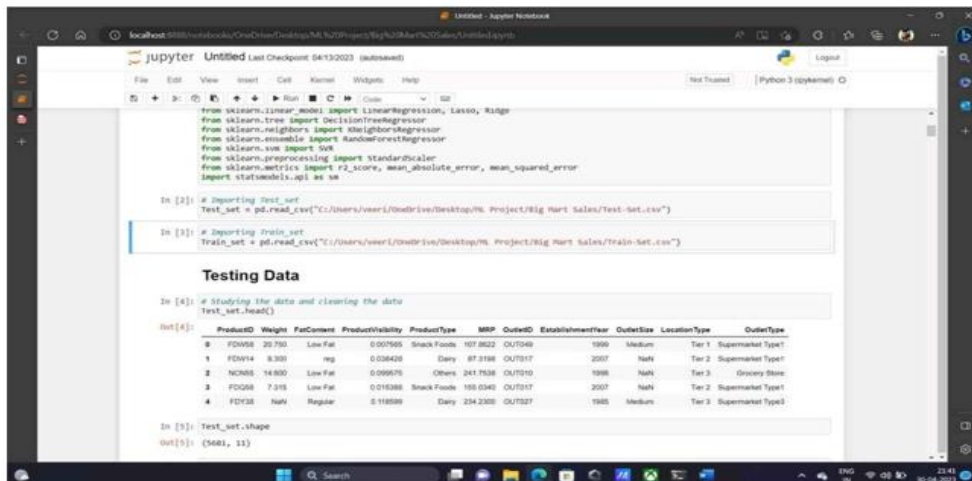
Importing The Libraries



```
In [1]: # Importing Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from scipy.stats import norm, boxcox
from scipy import stats
from collections import Counter
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import statsmodels.api as sm
```

- Importing different python libraries which are used for the prediction of our dataset

Importing the Test and Train dataset:



```
In [2]: # Importing Test set
Test_set = pd.read_csv("C:/Users/veeri/OneDrive/Desktop/ML Project/Big Mart Sales/Test-set.csv")

In [3]: # Importing Train set
Train_set = pd.read_csv("C:/Users/veeri/OneDrive/Desktop/ML Project/Big Mart Sales/train-set.csv")

Testing Data

In [4]: # Studying the data and cleaning the data
Test_set.head()
```

	ProductID	Weight	FatContent	ProductVelicity	ProductType	MRP	OutletID	EstablishmentYear	OutletSize	LocationType	OutletType
0	FD0058	20.750	Low Fat	0.007565	Snack Foods	107.8622	OUT049	1999	Medium	Tier 1	Supermarket Type1
1	FD0014	8.300	reg	0.038420	Dairy	87.3198	OUT017	2007	NaN	Tier 2	Supermarket Type1
2	NCD086	14.850	Low Fat	0.088676	Others	241.7538	OUT010	1998	NaN	Tier 3	Grocery Store
3	FD0068	7.216	Low Fat	0.016386	Snack Foods	168.0340	OUT017	2007	NaN	Tier 2	Supermarket Type1
4	FD0738	NaN	Regular	0.116589	Dairy	234.2300	OUT027	1985	Medium	Tier 3	Supermarket Type3

```
In [5]: Test_set.shape
Out[5]: (5081, 11)
```

- Importing test and train model datasets into our jupyter notebook so we can analyse our data using python libraries.

Studying and Analysing our Test Dataset:

The screenshot shows a Jupyter Notebook titled 'Untitled - Jupyter Notebook' with the following code and output:

```
In [4]: # Studying the data and cleaning the data
Test_set.head()
```

```
Out[4]:
```

	ProductID	Weight	FatContent	ProductVisibility	ProductType	MRP	OutletID	EstablishmentYear	OutletSize	LocationType	OutletType
0	FDW58	20.750	Low Fat	0.007565	Snack Foods	107.8622	OUT049	1999	Medium	Tier 1	Supermarket Type1
1	FDW14	8.300	reg	0.038428	Dairy	87.3198	OUT017	2007	NaN	Tier 2	Supermarket Type1
2	NCN65	14.600	Low Fat	0.096575	Others	241.7538	OUT010	1998	NaN	Tier 3	Grocery Store
3	FDQ58	7.315	Low Fat	0.015388	Snack Foods	155.0340	OUT017	2007	NaN	Tier 2	Supermarket Type1
4	FDY38	NaN	Regular	0.118599	Dairy	234.2300	OUT027	1985	Medium	Tier 3	Supermarket Type3

```
In [5]: Test_set.shape
Out[5]: (5681, 11)
```

```
In [6]: Test_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5681 entries, 0 to 5680
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  -
0   ProductID             5681 non-null   object
1   Weight                4705 non-null   float64
2   FatContent            5681 non-null   object
3   ProductVisibility      5681 non-null   float64
4   ProductType           5681 non-null   object
5   MRP                   5681 non-null   float64
6   OutletID              5681 non-null   object
7   EstablishmentYear      5681 non-null   int64
8   OutletSize            4875 non-null   object
9   LocationType          4684 non-null   object
10  OutletType             5681 non-null   object
```

The screenshot shows the continuation of the Jupyter Notebook with the following code and output:

```
In [7]: Test_set.describe()
```

```
Out[7]:
```

	Weight	ProductVisibility	MRP	EstablishmentYear
count	4705.000000	5681.000000	5681.000000	5681.000000
mean	12.695633	0.065684	141.023273	1997.823903
std	4.664849	0.051252	61.809091	8.372256
min	4.555000	0.000000	31.960000	1985.000000
25%	8.845000	0.027047	94.412000	1967.000000
50%	12.500000	0.054154	141.415400	1999.000000
75%	16.700000	0.093463	186.026600	2004.000000
max	21.350000	0.323637	266.588400	2009.000000

```
In [8]: Test_set.isnull().sum()
```

```
Out[8]:
```

ProductID	0
Weight	976
FatContent	0
ProductVisibility	0
ProductType	0
MRP	0
OutletID	0
EstablishmentYear	0
OutletSize	1606
LocationType	0
OutletType	0
dtype: int64	

```
In [9]: Test_set["Weight"].fillna(Test_set["Weight"].mean(), inplace = True)
Test_set.isnull().sum()
```

```
Out[9]:
```

ProductID	0
-----------	---

The screenshot shows a Jupyter Notebook with the following code and output:

```
In [9]: Test_set["weight"].fillna(test_set["weight"].mean(), inplace = True)
Test_set.isnull().sum()
```

```
Out[9]: ProductID      0
Weight      0
FatContent    0
ProductVisibility  0
ProductType    0
MRP      0
OutletID      0
EstablishmentYear  0
OutletSize    1606
LocationType    0
OutletType    0
dtype: int64
```

```
In [10]: Test_set["outletsize"].fillna(test_set["outletsize"].mode(), inplace = True)
Test_set.isnull().sum()
```

```
Out[10]: ProductID      0
Weight      0
FatContent    0
ProductVisibility  0
ProductType    0
MRP      0
OutletID      0
EstablishmentYear  0
OutletSize    1606
LocationType    0
OutletType    0
dtype: int64
```

```
In [11]: from scipy.stats import mode
mode_of_outlet_size = Test_set.pivot_table(values = 'outletsize', index = 'outlettype', aggfunc=(lambda x : mode(x.dropna()).mode))
```

The screenshot shows a Jupyter Notebook with the following code and output:

```
In [11]: from scipy.stats import mode
mode_of_outlet_size = Test_set.pivot_table(values = 'outletsize', index = 'outlettype', aggfunc=(lambda x : mode(x.dropna()).mode))
```

```
In [12]: print(mode_of_outlet_size)
```

```
Outlettype      OutletSize
Grocery Store      Small
Supermarket Type1  Small
Supermarket Type2  Medium
Supermarket Type3  Medium
```

```
In [13]: Test_set["FatContent"].value_counts()
```

```
Out[13]: Low Fat    3396
Regular    1935
LF         206
reg        78
Low Fat     66
Name: FatContent, dtype: int64
```

```
In [14]: Test_set.replace({'FatContent': {'low fat': 'Low Fat', 'LF': 'Low Fat', 'reg': 'Regular'}}, inplace=True)
Test_set["FatContent"].value_counts()
```

```
Out[14]: Low Fat    3668
Regular    2013
Name: FatContent, dtype: int64
```

```
In [15]: Test_set.head()
```

```
Out[15]:
```

	ProductID	Weight	FatContent	ProductVisibility	ProductType	MRP	OutletID	EstablishmentYear	OutletSize	LocationType	OutletType
0	FDV58	20.750000	Low Fat	0.007595	Snack Foods	107.8822	OUT049	1996	Medium	Tier 1	Supermarket Type1
1	FDV14	8.300000	Regular	0.038428	Dairy	87.3198	OUT017	2007	NaN	Tier 2	Supermarket Type1

- We analysed the data and studied the whole dataset is about.
- We performed data cleaning method which includes removing null values, removing unwanted columns, removing duplicate values from the dataset.
- And we will perform that the columns which contains string value has changed to numerical value.


```

In [15]: Test_set.head()
Out[15]:
  ProductID  Weight FatContent ProductVisibility ProductType  MRP  OutletID  EstablishmentYear  OutletSize  LocationType  OutletType
0    FDW58  20.750000   Low Fat    0.007565    Snack Foods  107.8922  OUT049      1999      Medium      Tier 1    Supermarket Type1
1    FDW14   8.300000   Regular    0.038428         Dairy   87.3198  OUT017      2007         NaN      Tier 2    Supermarket Type1
2    NCH55  14.600000   Low Fat    0.099575         Others  241.7538  OUT010      1998         NaN      Tier 3     Grocery Store
3    FDQ58   7.315000   Low Fat    0.015388    Snack Foods  155.0340  OUT017      2007         NaN      Tier 2    Supermarket Type1
4    FDY38  12.695633   Regular    0.118599         Dairy  234.2300  OUT027      1985      Medium      Tier 3    Supermarket Type3

In [16]: from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()

In [17]: Test_set["ProductID"] = encoder.fit_transform(Test_set["ProductID"])
Test_set["FatContent"] = encoder.fit_transform(Test_set["FatContent"])
Test_set["ProductType"] = encoder.fit_transform(Test_set["ProductType"])
Test_set["OutletID"] = encoder.fit_transform(Test_set["OutletID"])
Test_set["OutletSize"] = encoder.fit_transform(Test_set["OutletSize"])
Test_set["LocationType"] = encoder.fit_transform(Test_set["LocationType"])
Test_set["OutletType"] = encoder.fit_transform(Test_set["OutletType"])

In [18]: Test_set.head()
Out[18]:
  ProductID  Weight FatContent ProductVisibility ProductType  MRP  OutletID  EstablishmentYear  OutletSize  LocationType  OutletType
0    1103  20.750000    1103    0.007565         13  107.8922         9      1999         1         0         1
1     1067   8.300000    1067    0.038428          4   87.3198         2      2007         3         1         1
2     1406  14.600000    1406    0.099575        11  241.7538         0      1998         3         2         0
3      809   7.315000     809    0.015388        13  155.0340         2      2007         3         1         1
4    1184  12.695633    1184    0.118599          4  234.2300         5      1985         1         2         3

```

- We can see that the string values in the column has changed to numerical values so that we can perform our analysis clearly and based on that we can do the prediction.

Studying and Analysing our Train Dataset:

```

In [19]: Train_set.head()
Out[19]:
  ProductID  Weight FatContent ProductVisibility ProductType  MRP  OutletID  EstablishmentYear  OutletSize  LocationType  OutletType  OutletSales
0    FDA15   9.30   Low Fat    0.019047         Dairy  249.8092  OUT049      1999      Medium      Tier 1    Supermarket Type1    3735.1380
1    DRC01   5.90   Regular    0.019278    Soft Drinks  48.2892  OUT018      2009      Medium      Tier 3    Supermarket Type2    443.4228
2    FDN15  17.50   Low Fat    0.016760         Meat  141.6180  OUT049      1999      Medium      Tier 1    Supermarket Type1    2097.2700
3    FDQ07  19.20   Regular    0.000000    Fruits and Vegetables  182.0950  OUT010      1998         NaN      Tier 3     Grocery Store    732.3800
4    NCD19   8.90   Low Fat    0.000000    Household   53.8614  OUT013      1987         High      Tier 3    Supermarket Type1    994.7052

In [20]: Train_set.shape
Out[20]: (8523, 12)

In [21]: Train_set.info()
Out[21]:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  --
 0   ProductID           8523 non-null   object
 1   Weight              7068 non-null   float64
 2   FatContent          8523 non-null   object
 3   ProductVisibility   8523 non-null   float64
 4   ProductType         8523 non-null   object
 5   MRP                 8523 non-null   float64

```

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [22]: Train_set.describe()
```

```
Out[22]:
```

	Weight	ProductVisibility	MRP	EstablishmentYear	OutletSales
count	7090.000000	8523.000000	8523.000000	8523.000000	8523.000000
mean	12.857945	0.066132	140.962782	1997.831967	2181.288914
std	4.843456	0.051598	62.275067	8.371760	1706.499616
min	4.555000	0.000000	31.290000	1985.000000	33.290000
25%	8.773750	0.029889	93.829500	1987.000000	834.247400
50%	12.600000	0.053931	143.012800	1996.000000	1794.331000
75%	16.856000	0.084585	185.943700	2004.000000	3101.296400
max	21.350000	0.328391	266.868400	2009.000000	13086.964800

```
In [23]: Train_set.isnull().sum()
```

```
Out[23]:
```

ProductID	0
Weight	1853
FatContent	0
ProductVisibility	0
ProductType	0
MRP	0
OutletID	0
EstablishmentYear	0
OutletSize	2410
LocationType	0
OutletType	0
OutletSales	0

dtype: int64

```
In [24]: Train_set["weight"].fillna(Train_set["weight"].mean(),inplace = True)
```

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [24]: Train_set["weight"].fillna(Train_set["weight"].mean(),inplace = True)
```

```
In [25]: Train_set["OutletSize"].fillna(Train_set["OutletSize"].mode(),inplace = True)
```

```
In [26]: Train_set.isnull().sum()
```

```
Out[26]:
```

ProductID	0
Weight	0
FatContent	0
ProductVisibility	0
ProductType	0
MRP	0
OutletID	0
EstablishmentYear	0
OutletSize	2410
LocationType	0
OutletType	0
OutletSales	0

dtype: int64

```
In [27]: Train_set["FatContent"].value_counts()
```

```
Out[27]:
```

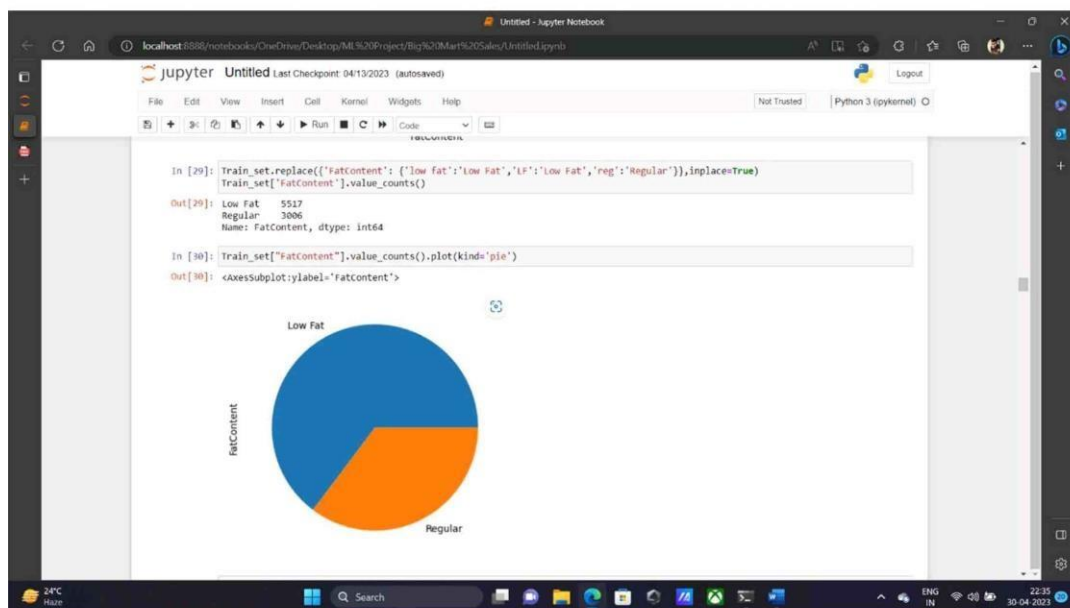
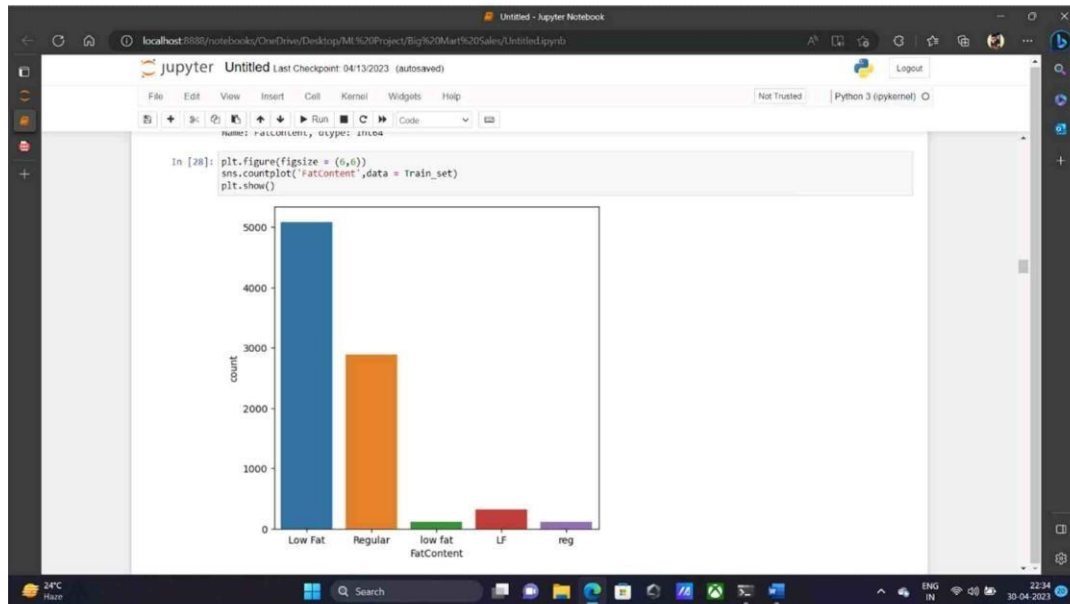
Low Fat	5089
Regular	2889
LF	316
reg	117
low fat	112

Name: FatContent, dtype: int64

```
In [28]: plt.figure(figsize = (6,6))  
sns.countplot('FatContent',data = Train_set)  
plt.show()
```

- We performed the same analytical statistics which we have performed for our test dataset.
- And now we will perform the count plot for Fat Content variable so that we can clearly observe that how many of them are duplicated values.

- And after that we will remove duplicate values and combine them into one value and visualize it using pie chart.



- We can see that the Fat Content variable values in our train data are different from that of test data.

The screenshot shows a Jupyter Notebook with the following code and output:

```
In [31]: encoder=LabelEncoder()
In [32]: Train_set["ProductID"]=encoder.fit_transform(Train_set["ProductID"])
Train_set["FatContent"]=encoder.fit_transform(Train_set["ProductID"])
Train_set["ProductType"]=encoder.fit_transform(Train_set["ProductType"])
Train_set["OutletID"]=encoder.fit_transform(Train_set["OutletID"])
Train_set["OutletSize"]=encoder.fit_transform(Train_set["OutletSize"])
Train_set["LocationType"]=encoder.fit_transform(Train_set["LocationType"])
Train_set["OutletType"]=encoder.fit_transform(Train_set["OutletType"])
In [33]: Train_set.head()
Out[33]:
```

	ProductID	FatContent	ProductVisibility	ProductType	MRP	OutletID	EstablishmentYear	OutletSize	LocationType	OutletType	OutletSales	
0	158	9.30	158	0.018047	4	240.8062	9	1999	1	0	1	3735.1380
1	8	5.92	8	0.019278	14	48.2892	3	2009	1	2	2	443.4228
2	662	17.50	662	0.018760	10	141.6180	9	1999	1	0	1	2097.2700
3	1121	19.20	1121	0.000000	6	182.0950	0	1998	3	2	0	732.3800
4	1297	6.93	1297	0.000000	9	53.8614	1	1987	0	2	1	994.7052

```
In [34]: y=Train_set["OutletSales"]
x=Train_set.drop(["OutletSales"],inplace=True,axis=1)
In [35]: y
Out[35]:
```

	OutletSales
0	3735.1380
1	443.4228
2	2097.2700
3	732.3800
4	994.7052
...	...
8518	2778.3834
8519	540.2830

- Converted the string values into numerical values for our predictive analysis.
- And removed the outlet sales value and stored it aside so that we can calculate our R-Squared value which is used for predictive models.

Predicting R-Squared value:

The screenshot shows a Jupyter Notebook with the following code and output:

```
In [38]: from sklearn.model_selection import train_test_split
In [39]: y_train,y_test,x_train,x_test=train_test_split(y,x,test_size = 0.3,random_state=2)
print(x.shape,x_train.shape,x_test.shape)
(8523, 11) (5966, 11) (2557, 11)
In [40]: import xgboost
from xgboost import XGBRegressor
model = XGBRegressor(n_estimators = 750,learning_rate = 0.007)
model.fit(x_train,y_train)
Out[40]: XGBRegressor(base_score=None, booster=None, callbacks=None,
colsample_bylevel=None, colsample_bynode=None,
colsample_bytree=None, early_stopping_rounds=None,
enable_categorical=False, eval_metric=None, feature_types=None,
gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
interaction_constraints=None, learning_rate=0.007, max_bin=None,
max_cat_threshold=None, max_cat_to_onehot=None,
max_delta_step=None, max_depth=None, max_leaves=None,
min_child_weight=None, missing=None, monotone_constraints=None,
n_estimators=750, n_jobs=None, num_parallel_tree=None,
predictor=None, random_state=None, ...)
```

```
In [41]: train_prediction = model.predict(x_train)
In [42]: from sklearn import metrics
r2_train = metrics.r2_score(y_train,train_prediction)
print("R Squared Value : ",r2_train)
R Squared Value : 0.6935136962191357
```

- By Using the above data cleaning and data prediction for train set and test set we found the R-Squared value.
- With R-Squared value we can define that it is a goodness-of-fit measure for linear regression models. And from this value we can see the best fit line for our linear regression model which is used for the prediction analysis.

Performing Linear Regression model on Test data:

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
In [50]: asTest_set["FatContent"]
         bTest_set["MRP"]

In [51]: from sklearn.model_selection import train_test_split
         a_train, a_test, b_train, b_test = train_test_split(a, b, train_size = 0.7, test_size = 0.3, random_state = 100)

In [52]: a_train.head()
Out[52]: 1827    1105
         1953     753
         2937     842
         3696    1304
         1195     521
         Name: FatContent, dtype: int64

In [53]: b_train.head()
Out[53]: 1827    176.0370
         1953    52.4324
         2937    183.2292
         3696    50.2034
         1195    124.2730
         Name: MRP, dtype: float64

In [54]: a_test.head()
Out[54]: 888     330
         1835    535
         2201    692
         3232    235
         610     803
         Name: FatContent, dtype: int64

In [55]: b_test.head()
```

- For our prediction in Linear Regression model, we will use our Fat Content variable and MRP Sales of the dataset.
- By this we can see the best fit line for our Linear Regression model which is used for predictive analysis.
- And we will be seeing T-Test value, P-Value, F-Score value and all other statistical values for our Linear Regression model.


```

In [55]: b_test.head()
Out[55]:
888    144.2812
1835    100.4700
2201    194.9452
3232    195.5794
610     158.1630
Name: HBP, dtype: float64

In [56]: import statsmodels.api as sm

In [57]: a_train_sm = sm.add_constant(a_train)
lr=sm.OLS(b_train,a_train_sm).fit()

In [58]: lr.params
Out[58]:
const      140.820338
FatContent   -0.000649
dtype: float64

In [59]: print(lr.summary())

OLS Regression Results
=====
Dep. Variable:          HBP      R-squared:            0.000
Model:                OLS      Adj. R-squared:        -0.000
Method:               Least Squares      F-statistic:         0.00750
Date:                Wed, 26 Apr 2023      Prob (F-statistic):      0.767
Time:                19:47:25      Log-Likelihood:        -22034.
No. Observations:      3976      AIC:                4.407e+04
Df Residuals:          3974      BIC:                4.408e+04
Df Model:               1
Covariance Type:       nonrobust
=====
coef    std err          t      Pr>|t|      [0.025    0.975]
-----
const    140.8203    1.963    71.748    0.000    136.972    144.668
FatContent -0.0006    0.002   -0.296    0.767    -0.005    0.004
=====
Omnibus:            483.929   Durbin-Watson:        2.042
Prob(Omnibus):      0.000   Jarque-Bera (JB):      141.389
Skew:               0.154   Prob(JB):              1.99e-31
Kurtosis:           2.129   Cond. No.              1.79e+03
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.79e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

```

```

In [59]: print(lr.summary())

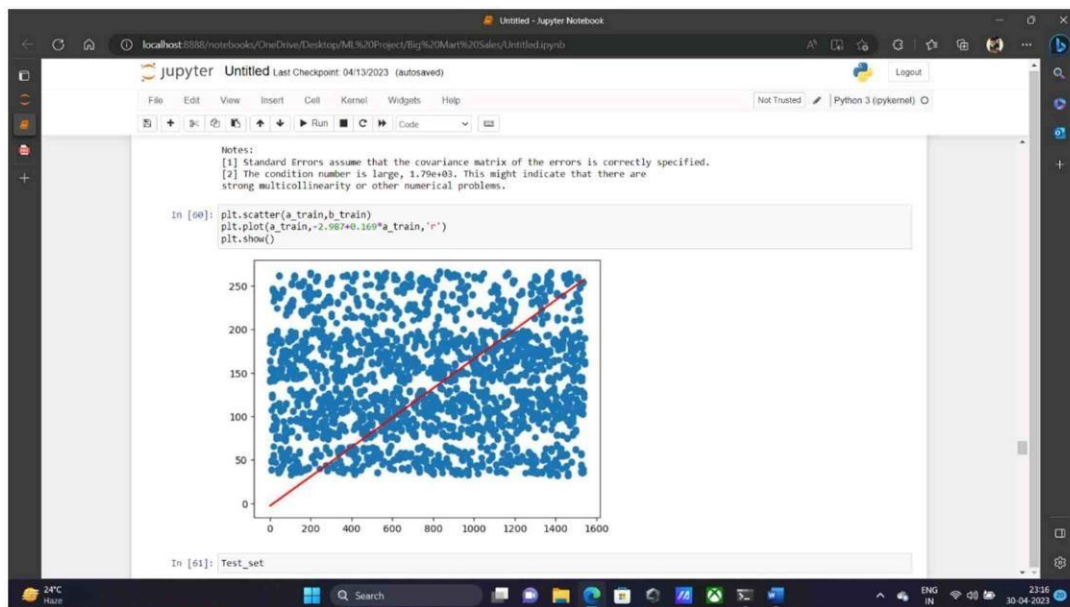
OLS Regression Results
=====
Dep. Variable:          HBP      R-squared:            0.000
Model:                OLS      Adj. R-squared:        -0.000
Method:               Least Squares      F-statistic:         0.00750
Date:                Wed, 26 Apr 2023      Prob (F-statistic):      0.767
Time:                19:47:25      Log-Likelihood:        -22034.
No. Observations:      3976      AIC:                4.407e+04
Df Residuals:          3974      BIC:                4.408e+04
Df Model:               2
Covariance Type:       nonrobust
=====
coef    std err          t      Pr>|t|      [0.025    0.975]
-----
const    140.8203    1.963    71.748    0.000    136.972    144.668
FatContent -0.0006    0.002   -0.296    0.767    -0.005    0.004
=====
Omnibus:            483.929   Durbin-Watson:        2.042
Prob(Omnibus):      0.000   Jarque-Bera (JB):      141.389
Skew:               0.154   Prob(JB):              1.99e-31
Kurtosis:           2.129   Cond. No.              1.79e+03
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.79e+03. This might indicate that there are
strong multicollinearity or other numerical problems.

In [60]: plt.scatter(a_train,b_train)
plt.plot(a_train,-2.987+0.169*a_train,'r')
plt.show()

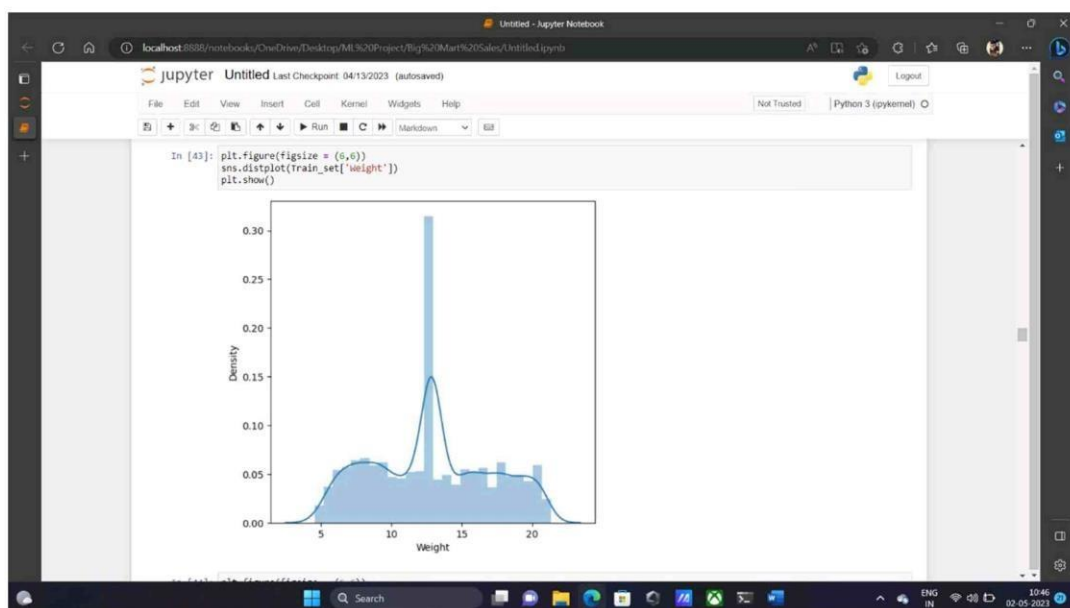
```

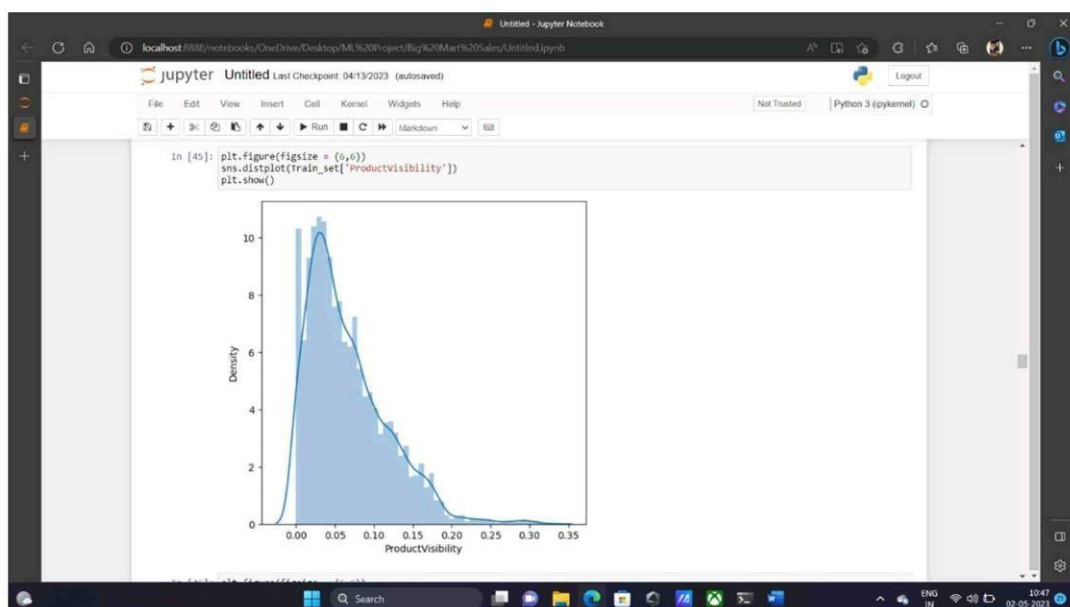
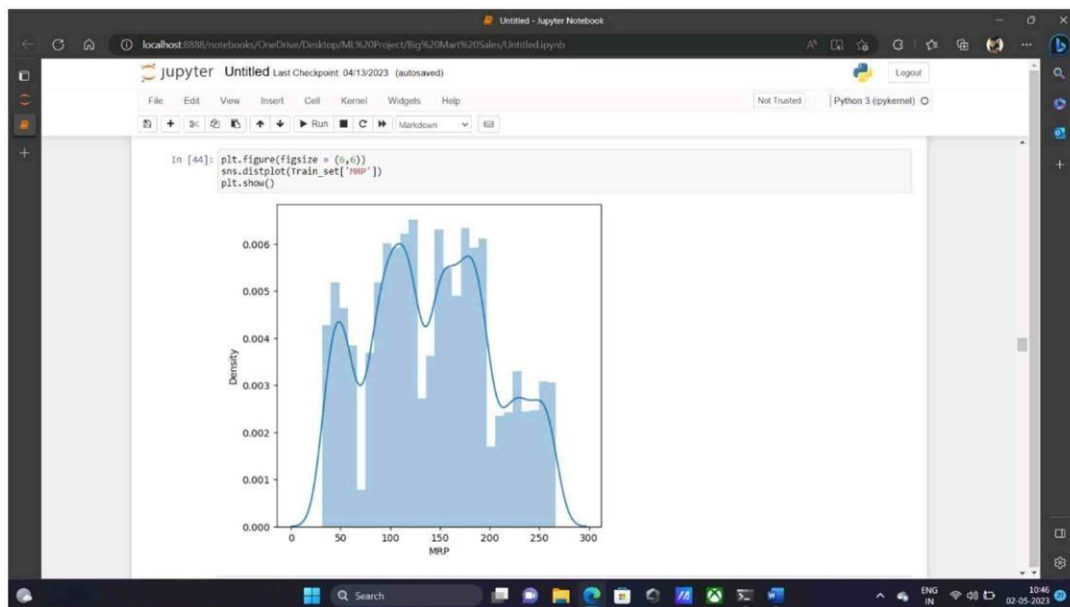
- From the above we can see all the statistical values which are used for Hypothesis Testing –
- If the P-value is greater than 0.05 then we will accept the null hypothesis.
- If the P-value is less than 0.05 then we will accept the alternate hypothesis.



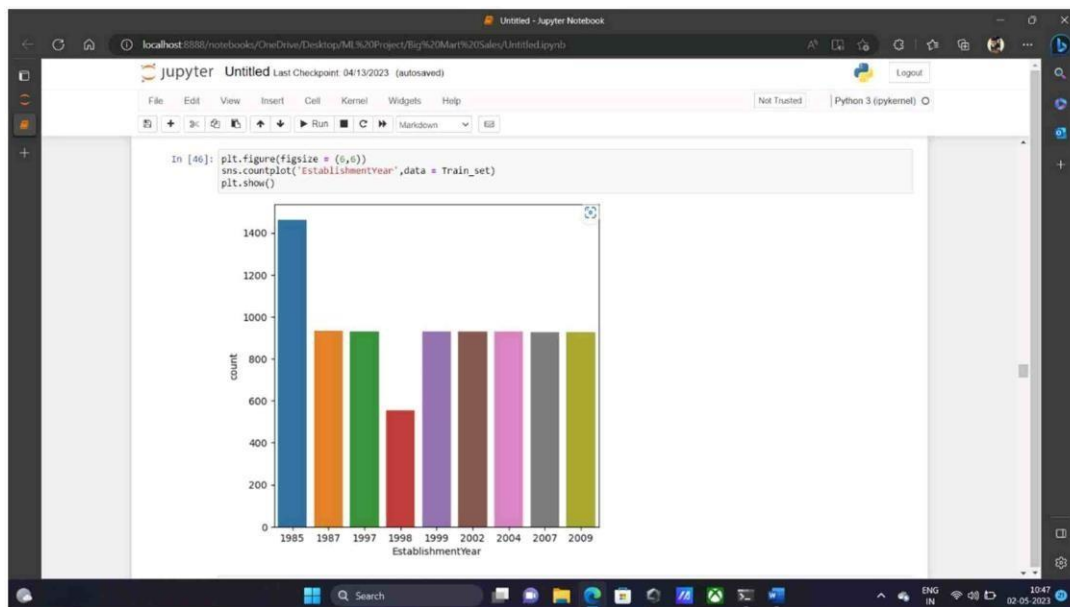
- We can see that best fit line for our Linear Regression model.
- From that we can say that the MRP Sales increases if there is decrease in Fat Content in the products.

Visualization of the dataset:

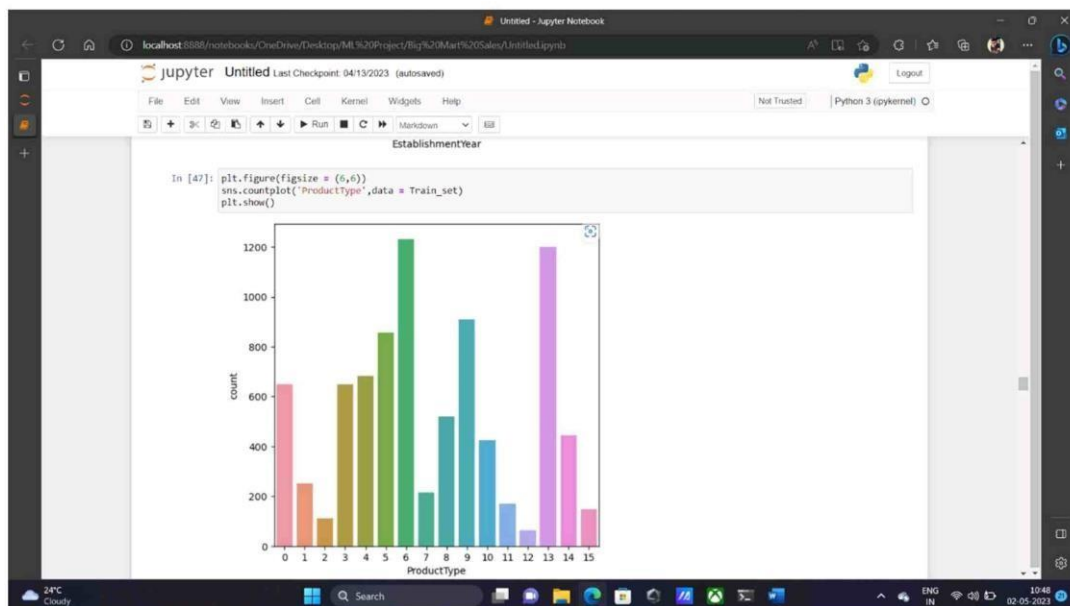




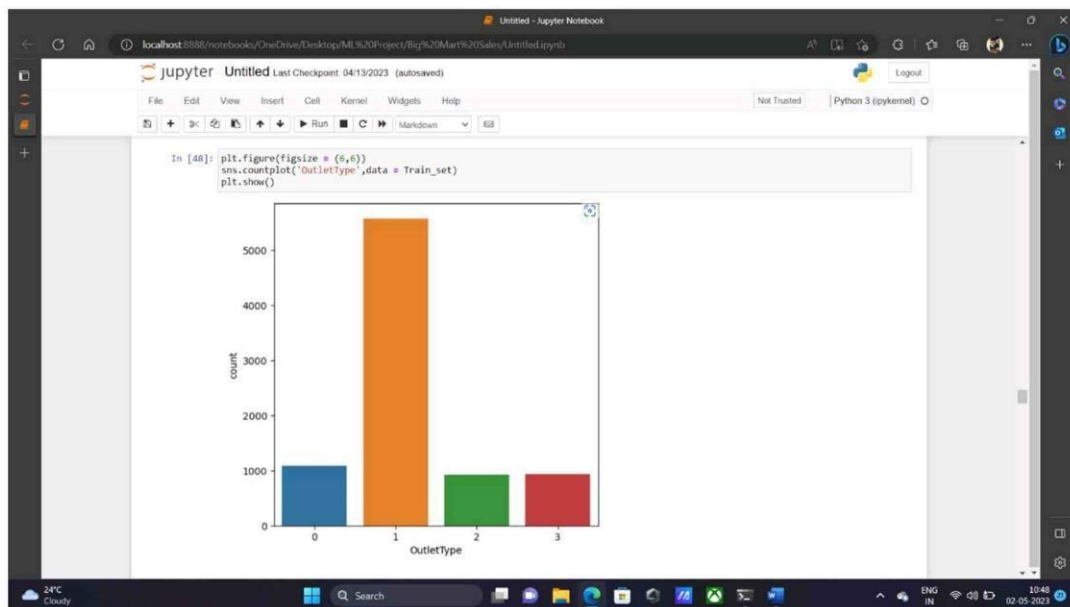
- These three graphs are the distplot which represents the density and hist of the given variable.
- And here we checked how the density is distributed throughout the data for the Weight, MRP, Product Visibility.
- We can observe that MRP is fluctuating here and when we check for product visibility it is decreasing rapidly.



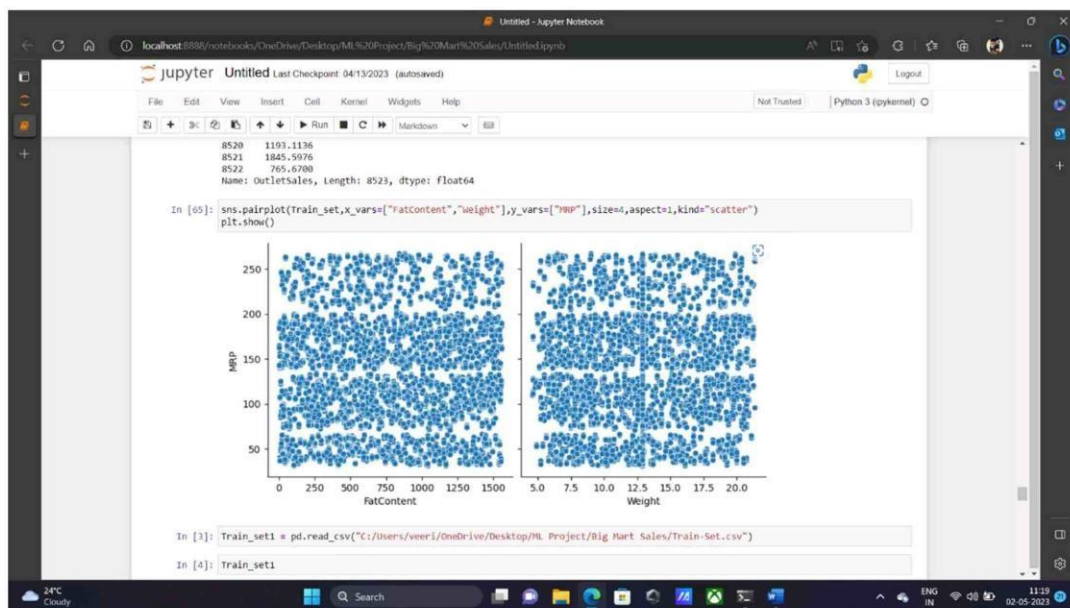
- From the count plot of the establishment year we can see that the most of the outlets are established in the year of 1985 and from preceding year they are establishing equal number of outlets except in the year of 1998.



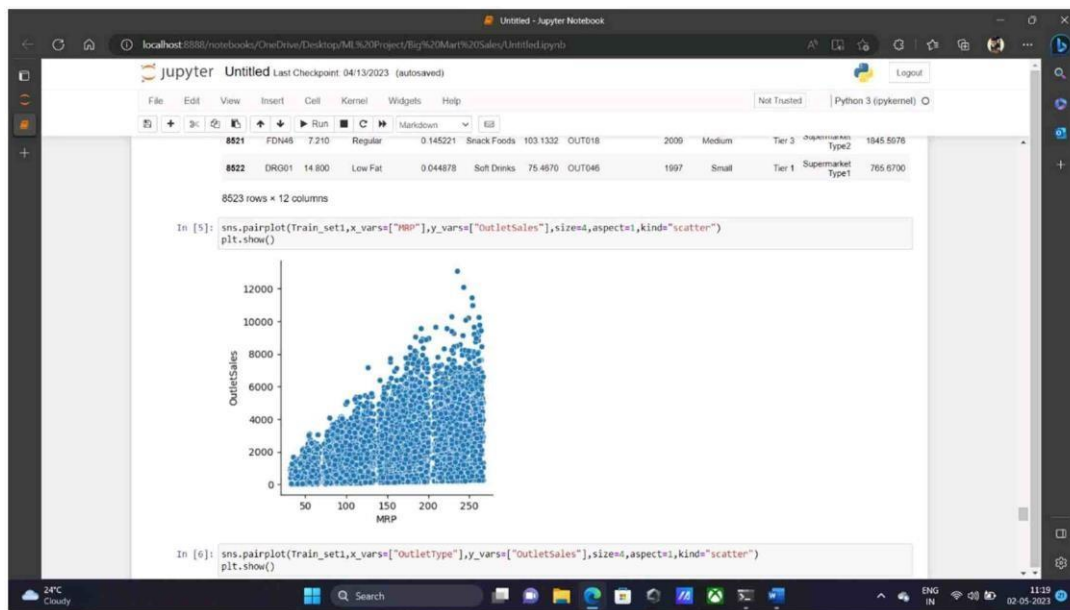
- From the count plot of product type, we can see that the 6, 13 are having the greater number of sales as compared to others.



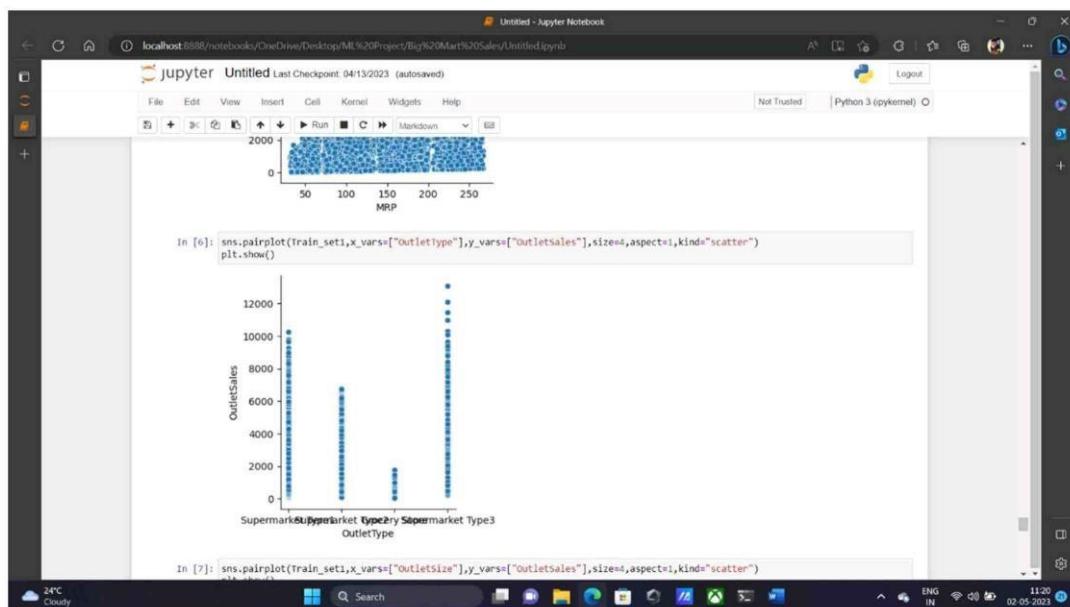
- From the count plot of outlet type we can see that the type 1 is greater than other outlets which means supermarkets are selling more than the grocery stores.

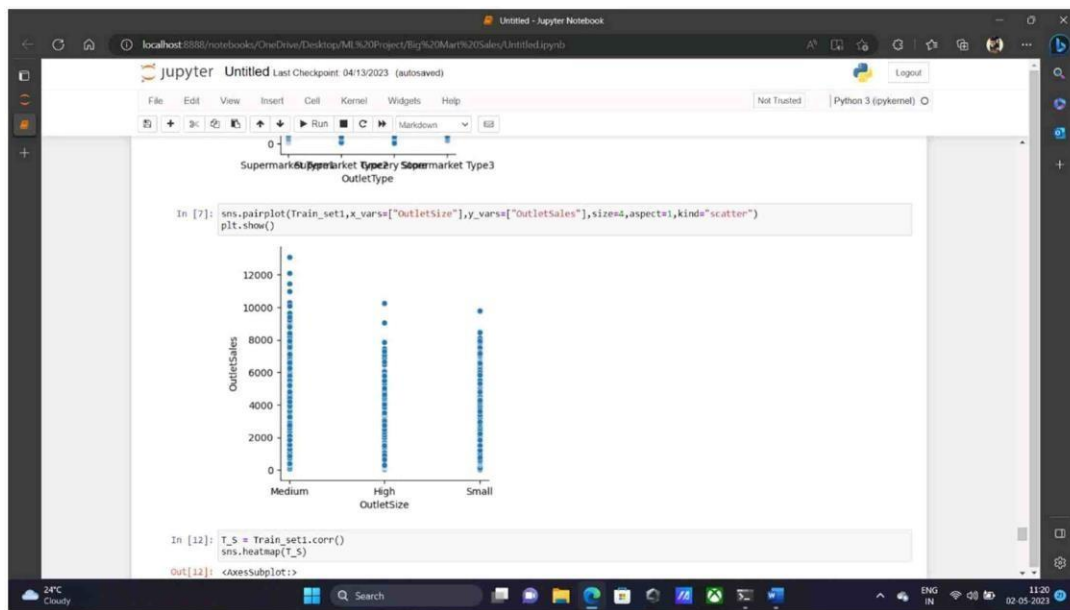


- Scatter plot for Fat Content and weight for train data depending upon the MRP sales and from this we can see it is also same as the test data.

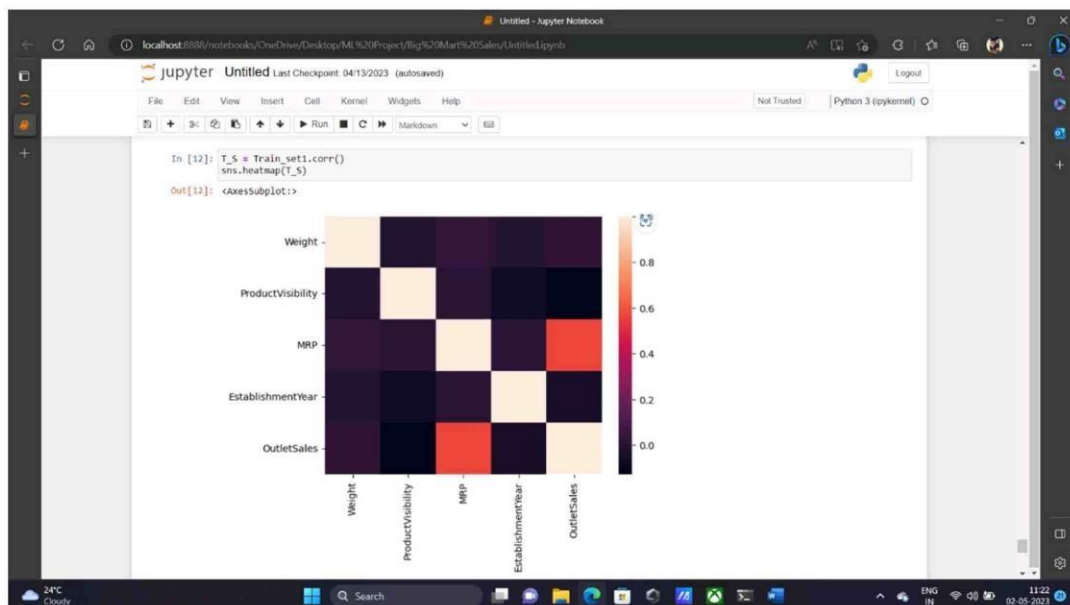


- From the above scatter plot we have checked the outlet sales depending upon the MRP sales and we can observe that the outlet sales are increasing rapidly in increase of the MRP sales.





- We can see here that the outlet sales do depend on the size of the outlet and here medium outlet size has a greater number of sales than high and small outlet size and we can say it does not depend whether the size is big or small it depends on the location.



Conclusion:

We have applied four algorithms XGBoost, Linear Regression and Decision Tree. From the results, we can conclude that among all the four algorithms XGBoost has the highest accuracy of 61.14% when distinguished together. Hence, we can say that XGBoost is the better algorithm for efficient sales analysis. This methodology is primarily used by shopping marts, groceries, Brand outlets etc. The data analysis applied to the predictive machine learning models provides a very effective way to manage sales, it also generously contributes to better decisions and plan strategies based on future demands. This approach is very much encouraged in today's world since it aids many companies, enterprises, researchers and brands for outcomes that lead to management of their profits, sales, inventory management, data research and customer demand.

From the above Statistical Analysis, Virtualization, Linear Regression Model we can conclude that our data is normally distributed.

And the null hypothesis is accepted.

We can also conclude that the outlet sales for the train data depends upon the Location, Outlet Type, Fat Content and MRP price of the product

We can clearly see that sales are high for the products which has low fat content.

And outlet sales do not depend upon the outlet size but it will depend upon the outlet type.

The big mart with medium size has more sales.

The big mart with small size has more size than big outlet size.

From the Linear Regression graph we can clearly see that the MRP increases in the future when there is increase in products which has low Fat Content.

Due to increase in MRP the Outlet Sales will be automatically increased.

References:

- [Find Open Datasets and Machine Learning Projects | Kaggle](#)
- Gopal Behere, Neeta Nain (2019). Grid Search Optimization (GSO) Based Future Sales Prediction for Big Mart. 2019 International Conference on Signal-Image Technology & Internet-Based Systems (SITIS).
- Bohdan M. Pavlyshenko (2018, August 25). Rainfall Predictive Approach for La Trinidad, Benguet using Machine Learning Classification. 2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP).
- A. Chandel, A. Dubey, S. Dhawale and M. Ghuge,,Sales Prediction System using Machine Learning; International Journal of Scientific Research and Engineering Development, vol. 2, no. 2, pp. 1-4, 2019. [Accessed 27 January 2020].
- Heramb Kadam, Rahul Shevade, Prof. Deven Ketkar , Mr. Sufiyan Rajguru (2018). A Forecast for Big Mart Sales Based on Random Forests and Multiple Linear Regression. (IJEDR).

THANK YOU