A PROJECT REPORT ON

BIGMART SALES PREDICTION

Submitted in partial fulfilment for the requirement of the award of

TRAINING

IN

Data Analytics, Machine Learning and AI using Python

Submitted By

ARGHYA BISWAS

(MTech 2nd Year Student at Indian Institute of Technology, Roorkee).

Under the guidance of

Mr Bandenawaz Bagwan

ACKNOWLEDGEMENT

My sincere gratitude and appreciations towards my project paper guide Mr Bandenawaz Bagwan. It was only with his help and support that I could complete the report. He provided me with proper assistance. He has my utmost gratitude. Last but not the least, I acknowledge parents for being such a nice source of encouragement and moral support that helped me tremendously in this aspect. I also declare to the best of my knowledge and belief that the Project Work has not been submitted anywhere else.

ABSTRACT

To implement an outlet sales prediction model for BIGMART, based on the dataset given by them. It's a Machine Learning project which integrates Data Science and Web Development. The major focus of this project is on predicting outlet sales using genuine factors as considered from the dataset that is given. It is intended to base an evaluation on all basic norm that is taken into account when establishing the sales. This project is a part of my training and internship in Data Analytics, Machine Learning and AI using Python at Diginique TechLabs.

INTRODUCTION

Context

This project is the final deliverable for the training and internship at Diginique TechLabs for Data Science Machine Learning and AI using Python. My supervisor was Mr Bandenawaz Bagwan, and I had one month to finish the task. We had two weeks of training sessions where we studied everything from the ground up.

Objective

To make it as detailed as possible by dealing with all the stages of the machine learning process and trying to understand it well.

The data scientists at BigMart have collected 2013 sales data for 1559 products across 10 stores in different cities. Also, certain attributes of each product and store have been defined. The aim is to build a predictive model and find out the sales of each product at a particular store. Create a model by which Big Mart can analyze and predict the outlet production sales.

I have completed the project following a proper structure which is as follows:

Problem Statement
Hypothesis Generation
Loading Packages and Data
Data Structure and Content
Exploratory Data Analysis- Univariate Analysis
Bivariate Analysis
Missing Value Treatment
Feature Engineering
Encoding Categorical Variables
Label Encoding
One Hot Encoding
Pre-Processing Data
Modeling
Linear Regression
Regularized Linear Regression
Random Forest
XGBoost

METHODOLOGY

Data Collection

BigMart has collected sales data from the year 2013, for 1559 products across 10 stores in different cities. Where the dataset consists of 12 attributes like Item Fat, Item Type, Item MRP, Outlet Type, Item Visibility, Item Weight, Outlet Identifier, Outlet Size, Outlet Establishment Year, Outlet Location Type, Item Identifier and Item Outlet Sales. Out of these attributes response variable is the Item Outlet Sales attribute and remaining attributes are used as the predictor variables.

The data-set is also based on hypotheses of store level and product level. Where store level involves attributes like: city, population density, store capacity, location, etc and the product level hypotheses involves attributes like: brand, advertisement, promotional offer, etc.

Linear Regression

Linear regression is a supervised learning technique. It is responsible for predicting the value of a dependent variable (Y) based on a given independent variable (X). It is the connection between the input (X) and the output (Y). It is one of the most well-known and well-understood machine learning algorithms. Simple linear regression, ordinary least squares, Gradient Descent, and Regularization are the linear regression models.

Decision Tree Regression

It is an object that trains a tree-structured model to predict data in the future in order to provide meaningful continuous output. The core principles of decision trees, Maximizing Information Gain, Classification trees, and Regression trees are the processes involved in decision tree regression. The essential notion of decision trees is that they are built via recursive partitioning. Each node can be divided into child nodes, beginning with the root node, which is known as the parent node. These nodes have the potential to become the parent nodes of their resulting offspring nodes. The nodes at the informative features are specified as the maximizing information gain, to establish an objective function that is to optimize the tree learning method.

Classification Trees

Classification trees are used to forecast the object into classes of a categorical dependent variable based on one or more predictor variables.

Regression Trees

It supports both continuous and categorical input variables. Regression trees are regarded as research with various machine algorithms for the regression issue, with the Decision Tree approach providing the lowest loss. The R-Squared value for the Decision Tree is 0.998, indicating that it is an excellent model. The Decision Tree was used to complete the web development.

Random Forest Regression

It is an essential learning approach for classification and regression to create a large number of decision trees. Preliminaries of decision trees are common approaches for a variety of machine learning problems. Tree learning is required for serving n off the self-produce for data mining since it is invariant despite scaling and several other changes. The trees are grown very deep in order to learn a high regular pattern. Random forest is a method of averaging several deep decision trees trained on various portions of the same training set. This comes at the price of a slight increase in bias and some interoperability.

XGBoost Regression

Extreme Gradient Boosting (XGBoost) is an open-source library that provides an efficient and effective implementation of the gradient boosting algorithm.
Shortly after its development and initial release, XGBoost became the go-to method and often the key component in winning solutions for a range of problems in machine learning competitions.
Regression predictive modeling problems involve predicting a numerical value such as a dollar amount or a height. XGBoost can be used directly for regression predictive modeling.

PROJECT

Problem Statement

Create a model by which Big Mart can analyze and predict the outlet production sales.

Data

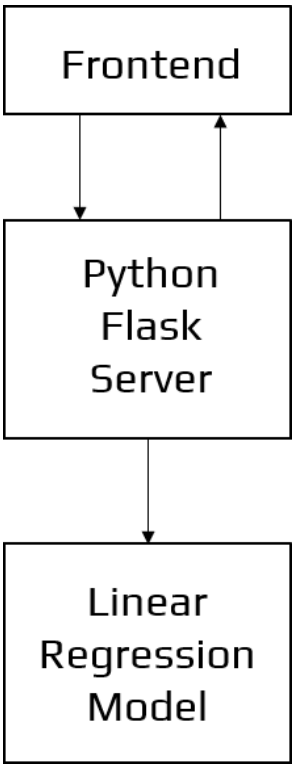
The data is the most important aspect of a machine learning assignment, to which special attention should be paid. Indeed, the data will heavily affect the findings depending on where we found them, how they are presented, if they are consistent, if there is an outlier, and so on. Many questions must be addressed at this stage to ensure that the learning algorithm is efficient and correct.

To obtain, clean, and convert the data, many steps are required. I'll go through each one to illustrate how they've been used in my project and why they're helpful for the machine learning section.

Dataset

Dataset: https://www.kaggle.com/aakash2016/big-mart-sales-dataset/download

Project Architecture



Data Science

The first stage is standard data science work, in which we take a data set named 'BigMart Sales Prediction'. We will do significant data cleaning on it to guarantee that it provides reliable predictions throughout prediction.

This Jupyter notebook, 'BigMart-Sales-Prediction-Model.ipynb,' is where we do all of our data science work, since the Jupyter notebook is self-explanatory. In terms of data cleansing, our dataset needs a significant amount of effort. In fact, 70% of the notebook is dedicated to data cleaning, in which we eliminate empty rows and remove superfluous columns that will not aid in prediction.

The process of obtaining valuable and significant information from a dataset that will contribute the most to a successful prediction is the next stage.

The final stage is to deal with outliers. Outliers are abnormalities that do massive damage to data and prediction. There is a lot to comprehend conceptually from the dataset in order to discover and eliminate these outliers.

Finally, the original dataset of over 8500 rows and 12 columns is reduced to much less.

Machine Learning

The resulting data is fed into a machine learning model. To find the optimal procedure and parameters for the model, we will mostly employ K-fold Cross-Validation and the GridSearchCV approach.

It turns out that the linear regression model produces the best results for our data, with a score of more than 80%, which is not terrible.

Now, we need to export our model as a pickle file (bigmartsales.pkl), which transforms Python objects into a character stream. Also, in order to interact with the locations(columns) from the frontend, we must export them into a JSON (columns.json) file.

Frontend

The front end is built up of straightforward HTML. To receive an estimated pricing, the user may fill-up the form with the proper details of the product the outlet sale of which is to be predicted. and click on the 'Predict' button. The JavaScript file is in charge of connecting with the backend flask server routes as well as the frontend HTML. It takes the form data entered by the user and executes the function, which employs the prediction model to calculate the projected outlet sales of the specific product and displays it in rupees after navigating to another html file that was linked to this file. The file that displays the prediction result will have a navigation link leading back to the first page.

MODEL

Steps to create the model are:

- Hypotheses and Data Exploration
- Handling Missing Values
- Univariate and Bivariate Analysis
- Feature Engineering
- Fitting a Linear Regression Model
- Decision Tree Regression Model
- Random Forest Regression Model
- XGBoost Regression Model

The steps are well defined in the following pages in the order given above.

Hypotheses and Data Exploration

Project Description: The data scientists at BigMart have collected 2013 sales data for 1559 products across 10 stores in different cities. Also, certain attributes of each product and store have been defined. The aim of this data science projectis to build a predictive model and find out the sales of each product at a particular store. Using this model, BigMart will try to understand the properties of products and stores which play a key role in increasing sales. The data has missing values as some stores do not report all the data due to technical glitches. Hence, it will berequired to treat them accordingly.

Store Level Hypotheses:

City type: Stores located in urban or Tier 1 cities should have higher sales because of the higher incomelevels of people there.
Population Density: Stores located in densely populated areas should have higher sales because of more demand.
Store Capacity: Stores which are very big in size should have higher sales as they act like one-stop-shopsand people would prefer getting everything from one place
Competitors: Stores having similar establishments nearby should have less sales because of more competition.
Marketing: Stores which have a good marketing division should have higher sales as it will be able to attractcustomers through the right offers and advertising.
Location: Stores located within popular marketplaces should have higher sales because of better access tocustomers.
Customer Behavior: Stores keeping the right set of products to meet the local needs of customers will havehigher sales.
Ambiance: Stores which are well-maintained and managed by polite and humble people are expected tohave higher footfall and thus higher sales.

Product Level Hypotheses:

Brand: Branded products should have higher sales because of higher trust in the customer.
Packaging: Products with good packaging can attract customers and sell more.
Utility: Daily use products should have a higher tendency to sell as compared to the specific use products. Display Area: Products which are given bigger shelves in the store are likely to catch attention first and sellmore.
Visibility in Store: The location of product in a store will impact sales. Ones which are right at entrance willcatch the eye of customer first rather than the ones in back.
Advertising: Better advertising of products in the store will should higher sales in most cases. Promotional Offers: Products accompanied with attractive offers and discounts will sell more.

Import Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns
```

In [2]:

```
#Read files:
df = pd.read_csv("train.csv")
```

In [3]:

```
df.head()
```

Out[3]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	

In [4]:

```
df.tail()
```

Out[4]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishm
8518	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5218	OUT013	
8519	FDS36	8.380	Regular	0.046982	Baking Goods	108.1570	OUT045	
8520	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	OUT035	
8521	FDN46	7.210	Regular	0.145221	Snack Foods	103.1332	OUT018	
8522	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.4670	OUT046	

In [5]:

```
df.shape
```

Out[5]:

```
(8523, 12)
```

In [6]:

```
df.describe()
```

Out[6]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
count	7060.000000	8523.000000	8523.000000	8523.000000	8523.000000
mean	12.857645	0.066132	140.992782	1997.831867	2181.288914
std	4.643456	0.051598	62.275067	8.371760	1706.499616

25%	8.773750	0.026989	93.826500	1987.000000	834.247400
50%	12.600000	0.053931	143.012800	1999.000000	1794.331000
75%	16.850000	0.094585	185.643700	2004.000000	3101.296400
max	21.350000	0.328391	266.888400	2009.000000	13086.964800

In [7]:

```
df.columns
```

Out[7]:

```
Index(['Item_Identifier', 'Item_Weight', 'Item_Fat_Content', 'Item_Visibility',
       'Item_Type', 'Item_MRP', 'Outlet_Identifier',
       'Outlet_Establishment_Year', 'Outlet_Size', 'Outlet_Location_Type',
       'Outlet_Type', 'Item_Outlet_Sales'],
      dtype='object')
```

Exploratory Data Analysis

As we can observe, all the column names are listed in the output.

Item_Identifier column is referenced as Unique Product ID.The remaining columns appear to be our features, with the last column being the response variable.

- **Item_Weight: Represents the weight of the product in floating values.**
- **Item_Fat_Content: Categorical Data which tells whether the product is low fat or not.**
- **Item_Visibility: The percentage of total display area all the products in a store.**
- **Item_Type: The Category to which the product belongs.**
- **Item_MRP: MRP of the products.**
- **Outlet_Identifier:Unique store ID.**
- **Outlet_Establishment_Year: Year in which the store was established.**
- **Outlet_Size: Size of the Store.**
- **Outlet_Location_Type: Type of City the store is located.**
- **Outlet_Type: Whether the store is a grocery sotre or a super market.**
- **Item_Outlet_Sales: The Sales of the product in a particular store.**

In [8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Item_Identifier            8523 non-null   object 
 1   Item_Weight                7060 non-null   float64
 2   Item_Fat_Content           8523 non-null   object 
 3   Item_Visibility            8523 non-null   float64
 4   Item_Type                  8523 non-null   object 
 5   Item_MRP                   8523 non-null   float64
 6   Outlet_Identifier          8523 non-null   object 
 7   Outlet_Establishment_Year  8523 non-null   int64  
 8   Outlet_Size                6113 non-null   object 
 9   Outlet_Location_Type       8523 non-null   object 
 10  Outlet_Type                8523 non-null   object 
 11  Item_Outlet_Sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 799.2+ KB
```

In [9]:

```
# Finding out the missing values for all the columns
```

```
df.isnull().sum()
```

Out[9]:

```
Item_Identifier                 0
Item_Weight                  1463
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  2410
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales               0
dtype: int64
```

In [10]:

```
# Displaying the Unique data for each column
df.nunique()
```

Out[10]:

```
Item_Identifier              1559
Item_Weight                   415
Item_Fat_Content                5
Item_Visibility              7880
Item_Type                      16
Item_MRP                     5938
Outlet_Identifier              10
Outlet_Establishment_Year       9
Outlet_Size                     3
Outlet_Location_Type            3
Outlet_Type                     4
Item_Outlet_Sales            3493
dtype: int64
```

Evaluating the Categorical Features

In [11]:

```
print('Frequency of Categories for varible Item ID')
id_counts = df['Item_Identifier'].value_counts()
id_counts.head()
```

```
Frequency of Categories for varible Item ID
```

Out[11]:

```
FDW13    10
FDG33    10
FDO19     9
FDF56     9
NCL31     9
Name: Item_Identifier, dtype: int64
```

In [12]:

```
print('Frequency of Categories for varible Item_Fat_Content')
df['Item_Fat_Content'].value_counts()
```

```
Frequency of Categories for varible Item_Fat_Content
```

Out[12]:

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

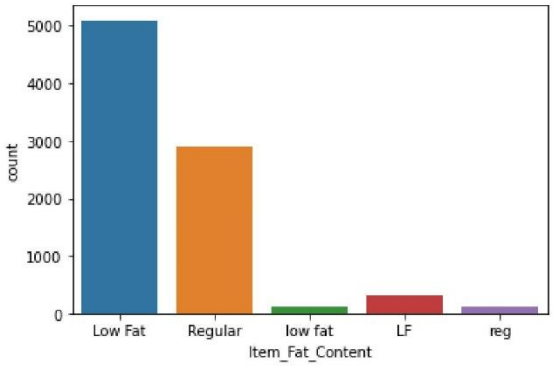
```
Name: Item_Fat_Content, dtype: int64
```

In [13]:

```
sns.countplot(df.Item_Fat_Content)
```

Out[13]:

```
<AxesSubplot:xlabel='Item_Fat_Content', ylabel='count'>
```



In [14]:

```
print('Frequency of Categories for varible Item_Type')
df['Item_Type'].value_counts()
```

```
Frequency of Categories for varible Item_Type
```

Out[14]:

```
Fruits and Vegetables    1232
Snack Foods              1200
Household                 910
Frozen Foods              856
Dairy                     682
Canned                    649
Baking Goods              648
Health and Hygiene        520
Soft Drinks               445
Meat                      425
Breads                    251
Hard Drinks               214
Others                    169
Starchy Foods             148
Breakfast                 110
Seafood                    64
Name: Item_Type, dtype: int64
```

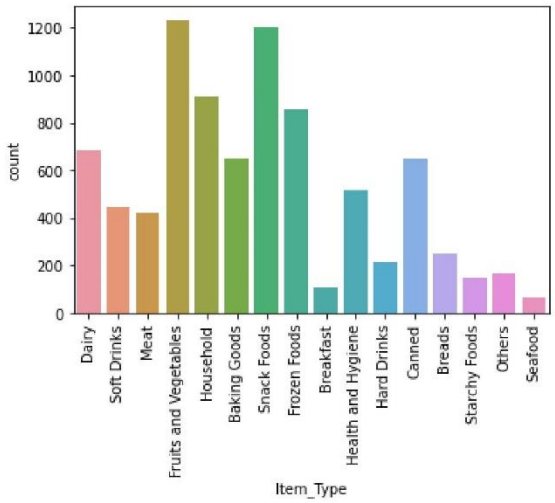
In [15]:

```
sns.countplot(df.Item_Type)
plt.xticks(rotation=90)
```

Out[15]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 [Text(0, 0, 'Dairy'),
  Text(1, 0, 'Soft Drinks'),
  Text(2, 0, 'Meat'),
  Text(3, 0, 'Fruits and Vegetables'),
  Text(4, 0, 'Household'),
  Text(5, 0, 'Baking Goods'),
  Text(6, 0, 'Snack Foods'),
  Text(7, 0, 'Frozen Foods'),
  Text(8, 0, 'Breakfast'),
  Text(9, 0, 'Health and Hygiene'),
  Text(10, 0, 'Hard Drinks'),
  Text(11, 0, 'Canned'),
```

```
Text(12, 0, 'Breads'),
Text(13, 0, 'Starchy Foods'),
Text(14, 0, 'Others'),
Text(15, 0, 'Seafood')])
```



In [16]:

```
Item_Type_and_fat_counts = df.groupby(["Item_Fat_Content","Item_Type"]).size()
print(Item_Type_and_fat_counts)
```

```
Item_Fat_Content  Item_Type
LF                Baking Goods             20
                  Breads                    8
                  Breakfast                 2
                  Canned                   17
                  Dairy                    24
                                           ..
reg               Fruits and Vegetables    25
                  Meat                      7
                  Snack Foods              23
                  Soft Drinks               1
                  Starchy Foods             7
Length: 70, dtype: int64
```

In [17]:

```
print('Frequency of Categories for varible Outlet ID')
outlet_id_counts = df['Outlet_Identifier'].value_counts()
outlet_id_counts
```

```
Frequency of Categories for varible Outlet ID
```

Out[17]:

```
OUT027    935
OUT013    932
OUT046    930
OUT035    930
OUT049    930
OUT045    929
OUT018    928
OUT017    926
OUT010    555
OUT019    528
Name: Outlet_Identifier, dtype: int64
```

In [18]:

```
print('Frequency of Categories for varible Outlet_Location_Type')
df['Outlet_Location_Type'].value_counts()
```

Frequency of Categories for varible Outlet_Location_Type

Out[18]:

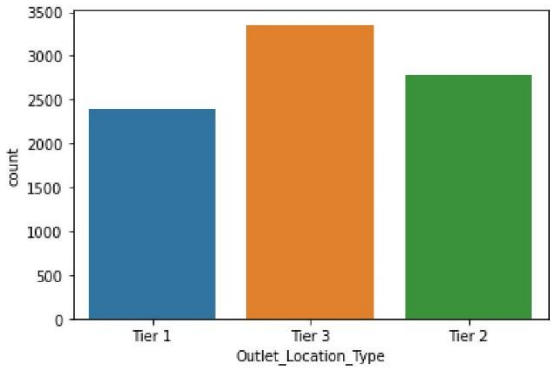
```
Tier 3    3350
Tier 2    2785
Tier 1    2388
Name: Outlet_Location_Type, dtype: int64
```

In [19]:

```
sns.countplot(df.Outlet_Location_Type)
```

Out[19]:

```
<AxesSubplot:xlabel='Outlet_Location_Type', ylabel='count'>
```



In [20]:

```
print('Frequency of Categories for varible Outlet_Size')
df['Outlet_Size'].value_counts()
```

Frequency of Categories for varible Outlet_Size

Out[20]:

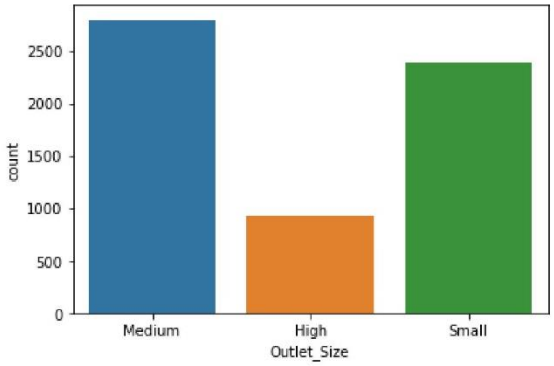
```
Medium    2793
Small     2388
High       932
Name: Outlet_Size, dtype: int64
```

In [21]:

```
sns.countplot(df.Outlet_Size)
```

Out[21]:

```
<AxesSubplot:xlabel='Outlet_Size', ylabel='count'>
```



In [22]:

```
print('Frequency of Categories for varible Outlet_Type')
df['Outlet_Type'].value_counts()
```

```
Frequency of Categories for varible Outlet_Type
```

Out[22]:

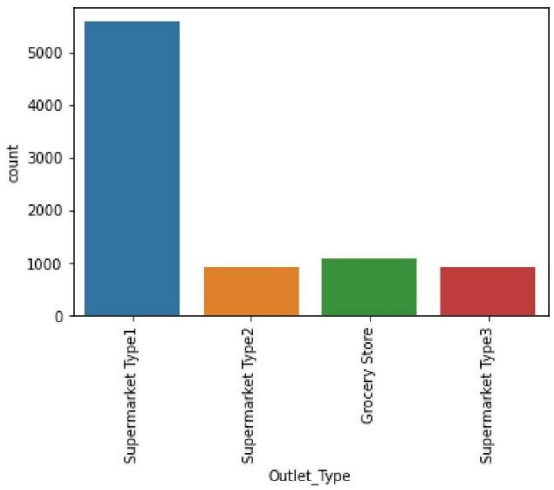
```
Supermarket Type1    5577
Grocery Store        1083
Supermarket Type3     935
Supermarket Type2     928
Name: Outlet_Type, dtype: int64
```

In [23]:

```
sns.countplot(df.Outlet_Type)
plt.xticks(rotation=90)
```

Out[23]:

```
(array([0, 1, 2, 3]),
 [Text(0, 0, 'Supermarket Type1'),
  Text(1, 0, 'Supermarket Type2'),
  Text(2, 0, 'Grocery Store'),
  Text(3, 0, 'Supermarket Type3')])
```



In [24]:

```
Outlet_Location_Type_counts = df.groupby(["Outlet_Location_Type","Outlet_Identifier","Ou
tlet_Type"]).size()
print(Outlet_Location_Type_counts)
```

```
Outlet_Location_Type  Outlet_Identifier  Outlet_Type
Tier 1                OUT019             Grocery Store        528
                      OUT046             Supermarket Type1    930
                      OUT049             Supermarket Type1    930
Tier 2                OUT017             Supermarket Type1    926
                      OUT035             Supermarket Type1    930
                      OUT045             Supermarket Type1    929
Tier 3                OUT010             Grocery Store        555
                      OUT013             Supermarket Type1    932
                      OUT018             Supermarket Type2    928
                      OUT027             Supermarket Type3    935
dtype: int64
```

In [25]:

```
Outlet_Location_Type_counts = df.groupby(["Outlet_Type","Outlet_Size"]).size()
print(Outlet_Location_Type_counts)
```

```
Outlet_Type         Outlet_Size
Grocery Store       Small           528
Supermarket Type1   High            932
                    Medium          930
                    Small          1860
Supermarket Type2   Medium          928
Supermarket Type3   Medium          935
dtype: int64
```

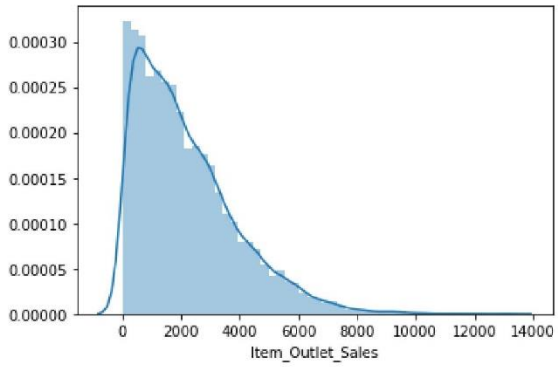
Visualizing the Relationship between Features and Response

In [26]:

```
sns.distplot(a = df['Item_Outlet_Sales'])
```

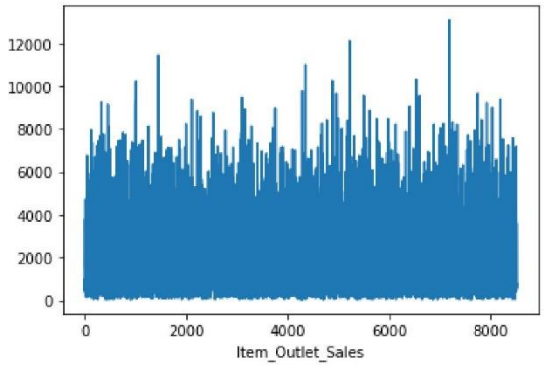
Out[26]:

```
<AxesSubplot:xlabel='Item_Outlet_Sales'>
```



In [27]:

```
#This is a line plot to show the variation of the target variable across the dataset
import seaborn as sns
sns.lineplot(data = df['Item_Outlet_Sales'])
plt.xlabel('Item_Outlet_Sales')
plt.show()
```

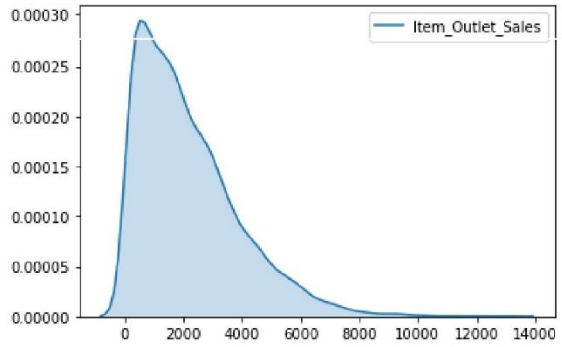


In [28]:

```
sns.kdeplot(data = df['Item_Outlet_Sales'], shade = True)
```

Out[28]:

<AxesSubplot:>



In [29]:

```
print ("Skew is:", df.Item_Outlet_Sales.skew())
print("Kurtosis: %f" % df.Item_Outlet_Sales.kurt())
```

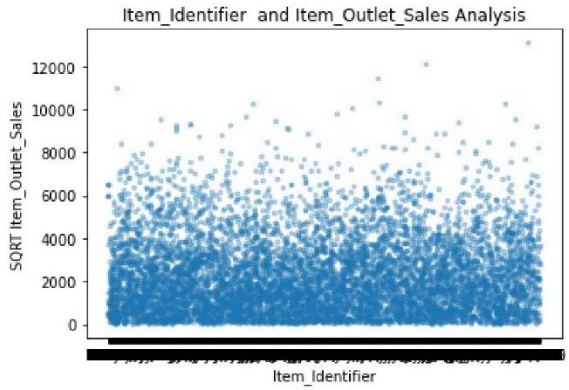
```
Skew is: 1.1775306028542798
Kurtosis: 1.615877
```

In [30]:

```
plt.xlabel("Item_Identifier")
plt.ylabel("SQRT Item_Outlet_Sales")
plt.title("Item_Identifier  and Item_Outlet_Sales Analysis")
plt.plot(df.Item_Identifier , df["Item_Outlet_Sales"],'.', alpha = 0.3)
```

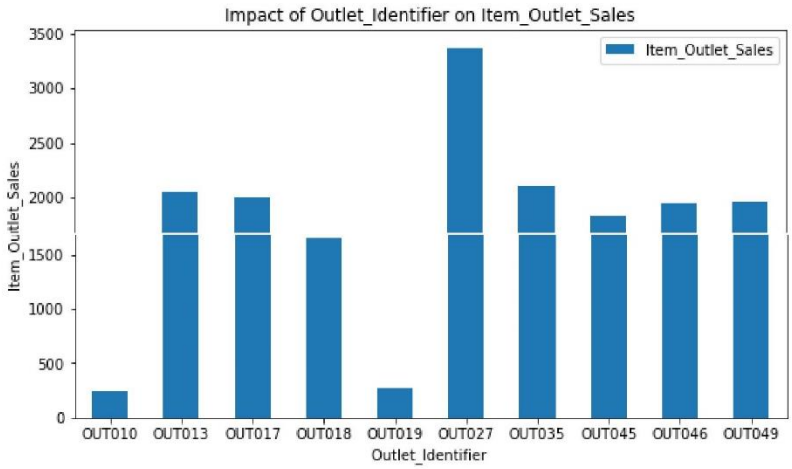
Out[30]:

```
[<matplotlib.lines.Line2D at 0x205cd503f28>]
```



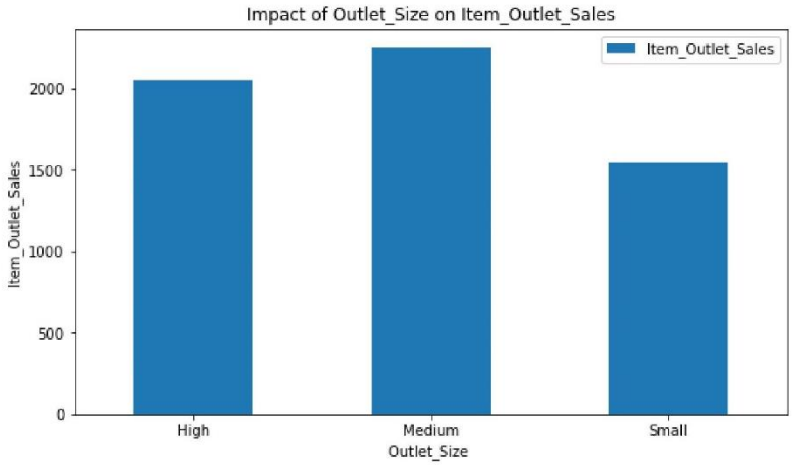
In [31]:

```
Outlet_Identifier_pivot = \
df.pivot_table(index='Outlet_Identifier', values="Item_Outlet_Sales", aggfunc=np.median)

Outlet_Identifier_pivot.plot(kind='bar',figsize=(9,5))
plt.xlabel("Outlet_Identifier ")
plt.ylabel("Item_Outlet_Sales")
plt.title("Impact of Outlet_Identifier on Item_Outlet_Sales")
plt.xticks(rotation=0)
plt.show()
```



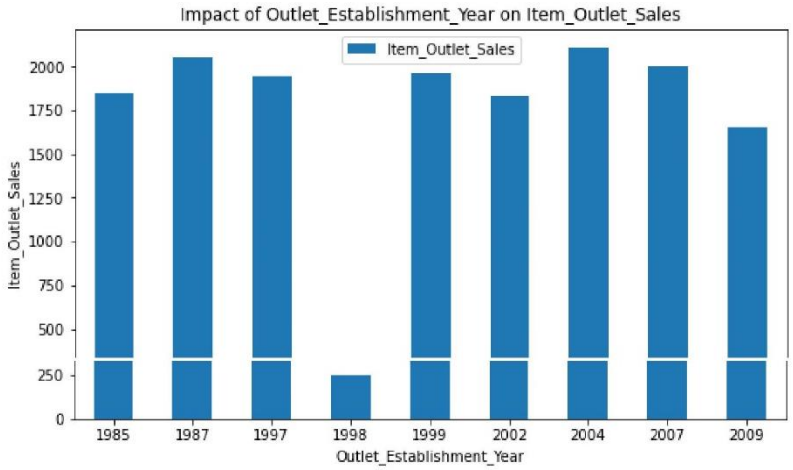
In [32]:

```
Outlet_Size_pivot = \
df.pivot_table(index='Outlet_Size', values="Item_Outlet_Sales", aggfunc=np.median)

Outlet_Size_pivot.plot(kind='bar',figsize=(9,5))
plt.xlabel("Outlet_Size ")
plt.ylabel("Item_Outlet_Sales")
plt.title("Impact of Outlet_Size on Item_Outlet_Sales")
plt.xticks(rotation=0)
plt.show()
```



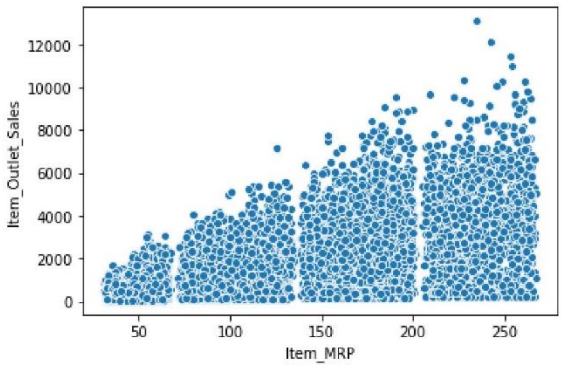
In [33]:

```
Outlet_Establishment_Year_pivot = \
df.pivot_table(index='Outlet_Establishment_Year', values="Item_Outlet_Sales", aggfunc=np
.median)

Outlet_Establishment_Year_pivot.plot(kind='bar',figsize=(9,5))
plt.xlabel("Outlet_Establishment_Year")
plt.ylabel("Item_Outlet_Sales")
plt.title("Impact of Outlet_Establishment_Year on Item_Outlet_Sales")
plt.xticks(rotation=0)
plt.show()
```



In [34]:

```
#This scatter plot show the variation of Item MRP vs Item Sales using scatter plot
sns.scatterplot(x = df['Item_MRP'], y = df['Item_Outlet_Sales'])
# plt.xlabel('Item_MRP')
# plt.ylabel('Item_Outlet_Sales')
plt.show()
```

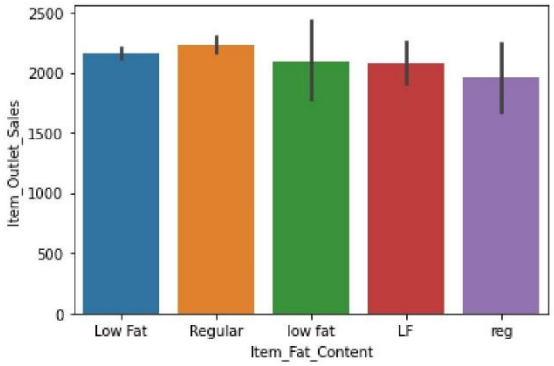


In [35]:

```
#This shows the variation of Item Fat Content vs Item Sales using bar plot
sns.barplot(x = df['Item_Fat_Content'],y = df['Item_Outlet_Sales'])
```

Out[35]:

```
<AxesSubplot:xlabel='Item_Fat_Content', ylabel='Item_Outlet_Sales'>
```

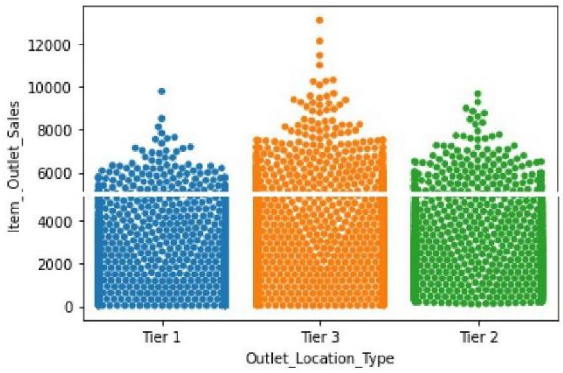


In [36]:

```
#This is a variation of outletlocation type vs sales using a swarm plot
sns.swarmplot(x = df['Outlet_Location_Type'],y = df['Item_Outlet_Sales'])
```

Out[36]:

```
<AxesSubplot:xlabel='Outlet_Location_Type', ylabel='Item_Outlet_Sales'>
```

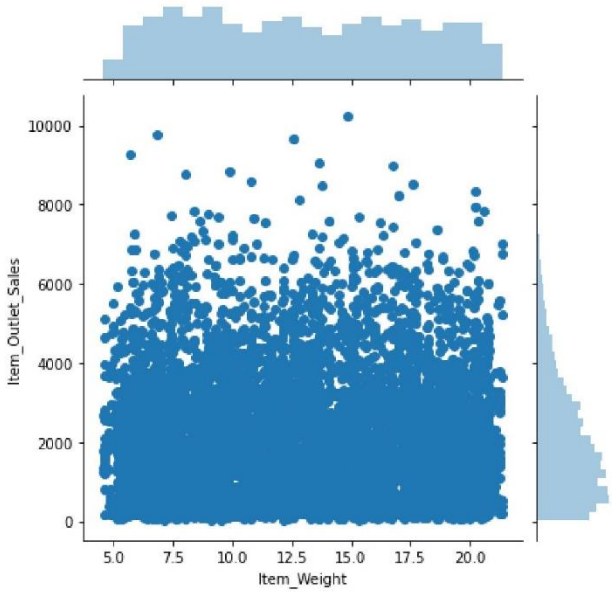


In [37]:

```
#A simple joint plot to visualize item MRP and outlet sales
sns.jointplot(x = df['Item_Weight'],y = df['Item_Outlet_Sales'])
```

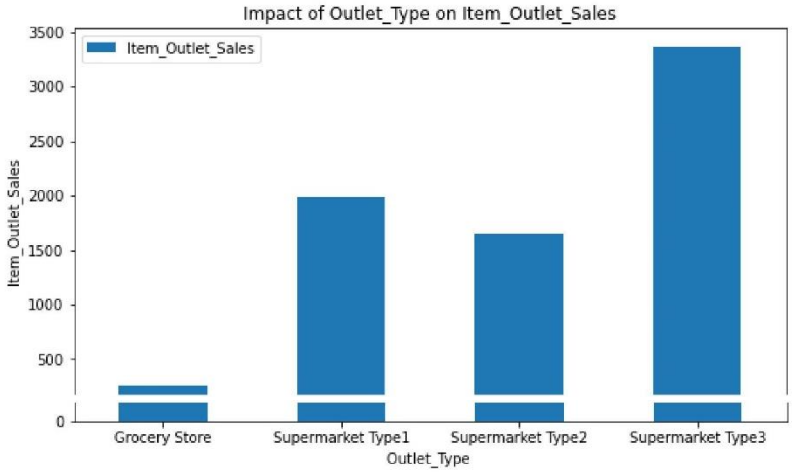
Out[37]:

```
<seaborn.axisgrid.JointGrid at 0x205cf48d2b0>
```



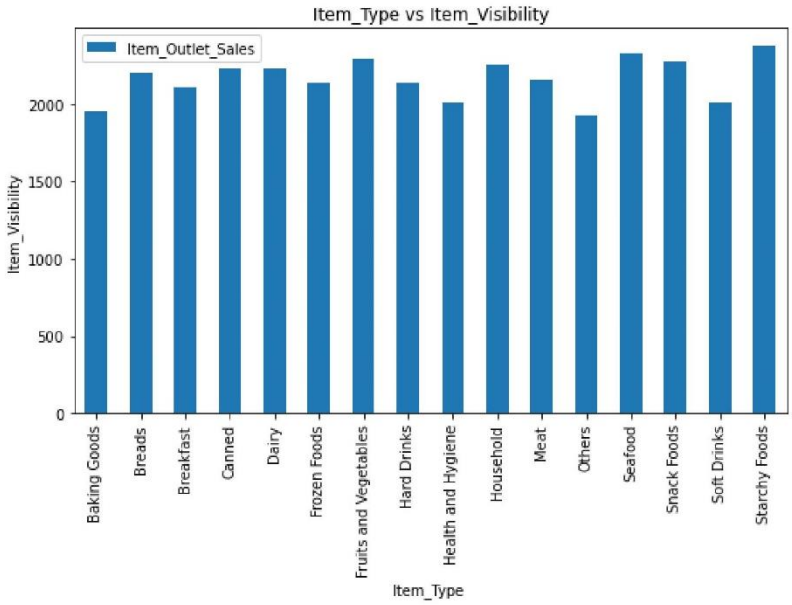
In [38]:

```
Outlet_Type_pivot = \
df.pivot_table(index='Outlet_Type', values="Item_Outlet_Sales", aggfunc=np.median)

Outlet_Type_pivot.plot(kind='bar',figsize=(9,5))
plt.xlabel("Outlet_Type ")
plt.ylabel("Item_Outlet_Sales")
plt.title("Impact of Outlet_Type on Item_Outlet_Sales")
plt.xticks(rotation=0)
plt.show()
```



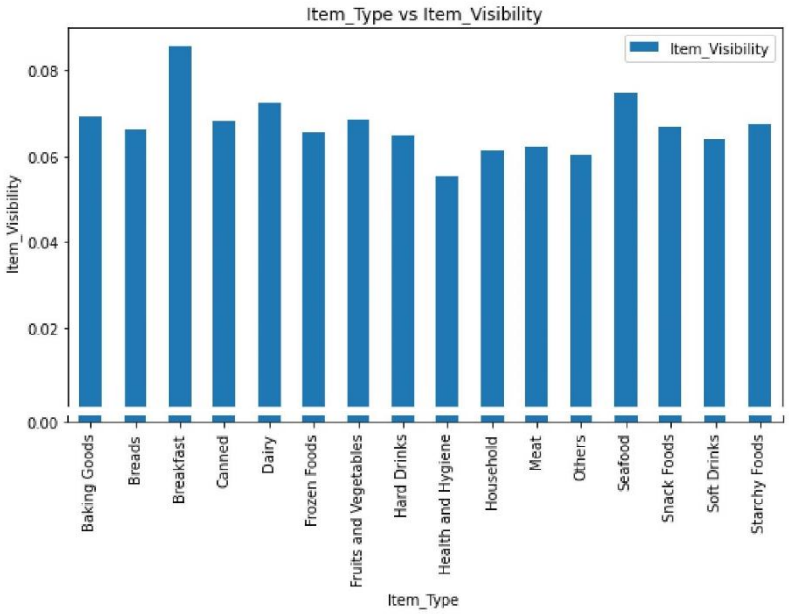
In [39]:

```
pivoTable = \
df.pivot_table(index='Item_Type', values="Item_Outlet_Sales", aggfunc=np.mean)

pivoTable.plot(kind='bar',figsize=(9,5))
plt.xlabel("Item_Type ")
plt.ylabel("Item_Visibility")
plt.title("Item_Type vs Item_Visibility")
plt.xticks(rotation=90)
plt.show()
```



In [40]:

```
pivoTable = \
df.pivot_table(index='Item_Type', values="Item_Visibility", aggfunc=np.mean)

pivoTable.plot(kind='bar',figsize=(9,5))
plt.xlabel("Item_Type ")
plt.ylabel("Item_Visibility")
plt.title("Item_Type vs Item_Visibility")
plt.xticks(rotation=90)
plt.show()
```

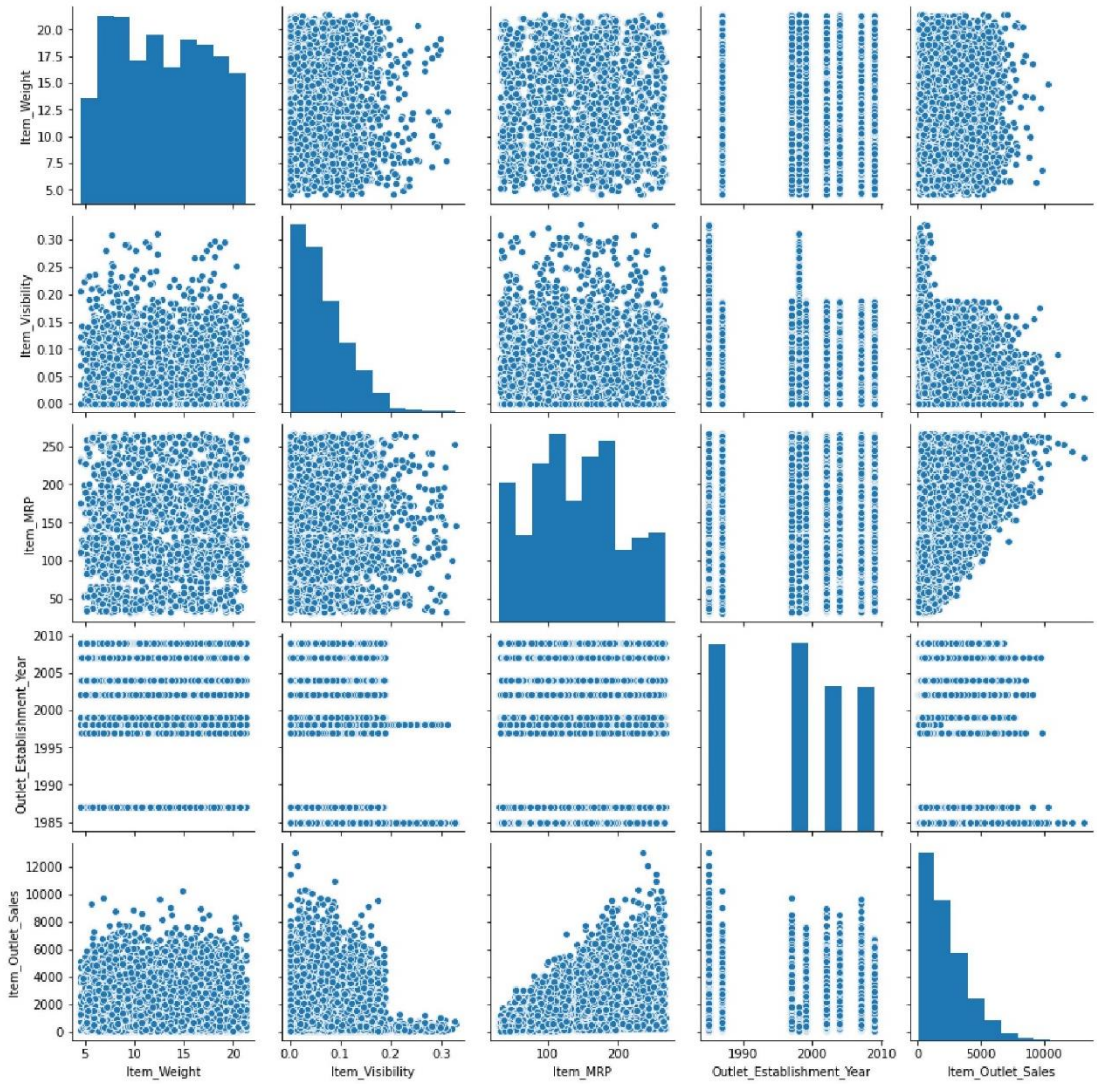

Item_Type vs Item_Visibility
Item_Visibility
0.08
0.06
0.04
0.02
0.00
Baking Goods
Breads
Breakfast
Canned
Dairy
Frozen Foods
Fruits and Vegetables
Hard Drinks
Health and Hygiene
Household
Meat
Others
Seafood
Snack Foods
Soft Drinks
Starchy Foods
Item_Type

In [41]:

```
sns.pairplot(data = df)
```

Out[41]:

```
<seaborn.axisgrid.PairGrid at 0x205d061aef0>
```



Handling Missing Values

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns
```

In [2]:

```
#Read files:
df = pd.read_csv("train.csv")
df.head()
```

Out[2]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            8523 non-null   object
 1   Item_Weight                7060 non-null   float64
 2   Item_Fat_Content           8523 non-null   object
 3   Item_Visibility            8523 non-null   float64
 4   Item_Type                  8523 non-null   object
 5   Item_MRP                   8523 non-null   float64
 6   Outlet_Identifier          8523 non-null   object
 7   Outlet_Establishment_Year  8523 non-null   int64
 8   Outlet_Size                6113 non-null   object
 9   Outlet_Location_Type       8523 non-null   object
 10  Outlet_Type                8523 non-null   object
 11  Item_Outlet_Sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 799.2+ KB
```

In [4]:

```
df.nunique()
```

Out[4]:

```
Item_Identifier              1559
Item_Weight                   415
Item_Fat_Content                5
Item_Visibility              7880
Item_Type                      16
Item_MRP                     5938
Outlet_Identifier              10
```

```
Outlet_Establishment_Year       9
Outlet_Size                     3
Outlet_Location_Type            3
Outlet_Type                     4
Item_Outlet_Sales            3493
dtype: int64
```

In [5]:

```
df.isnull().sum()
```

Out[5]:

```
Item_Identifier               0
Item_Weight                1463
Item_Fat_Content               0
Item_Visibility                0
Item_Type                      0
Item_MRP                       0
Outlet_Identifier              0
Outlet_Establishment_Year      0
Outlet_Size                 2410
Outlet_Location_Type           0
Outlet_Type                    0
Item_Outlet_Sales              0
dtype: int64
```

Handling Missing values

We found two variables with missing values – Item_Weight and Outlet_Size

In [6]:

```
# #Determine the average weight per item:
item_avg_weight = df.groupby(["Item_Identifier"])["Item_Weight"].mean()
item_avg_weight
```

Out[6]:

```
Item_Identifier
DRA12    11.600
DRA24    19.350
DRA59     8.270
DRB01     7.390
DRB13     6.115
          ...
NCZ30     6.590
NCZ41    19.850
NCZ42    10.500
NCZ53     9.600
NCZ54    14.650
Name: Item_Weight, Length: 1559, dtype: float64
```

In [7]:

```
#Get a boolean variable specifying missing Item_Weight values
miss_bool = df['Item_Weight'].isnull()
miss_bool
```

Out[7]:

```
0       False
1       False
2       False
3       False
4       False
        ...
8518    False
8519    False
8520    False
8521    False
```

```
8522    False
Name: Item_Weight, Length: 8523, dtype: bool
```

In [8]:

```
miss_bool.sum()
```

Out[8]:

```
1463
```

In [9]:

```
df.loc[miss_bool,'Item_Identifier']
```

Out[9]:

```
7       FDP10
18      DRI11
21      FDW12
23      FDC37
29      FDC14
        ...
8485    DRK37
8487    DRG13
8488    NCN14
8490    FDU44
8504    NCN18
Name: Item_Identifier, Length: 1463, dtype: object
```

Replacing the missing values of weight with the average weight of the same product

In [10]:

```
def impute_Item_Weight(df):
    # #Determine the average weight per item:
    item_avg_weight = df.groupby(["Item_Identifier"])["Item_Weight"].mean()
    item_avg_weight

    #Get a boolean variable specifying missing Item_Weight values
    miss_bool = df['Item_Weight'].isnull()

    #Impute data and check #missing values before and after imputation to confirm
    print('Orignal #missing: %d'% sum(miss_bool))
    df.loc[miss_bool,'Item_Weight'] = df.loc[miss_bool,'Item_Identifier'].apply(lambda x
: item_avg_weight.loc[x])
    print('Final #missing: %d'% sum(df['Item_Weight'].isnull()))
```

In [11]:

```
impute_Item_Weight(df)
```

```
Orignal #missing: 1463
Final #missing: 4
```

Even after replacing the weight values, the weight of 4 products is still missing. As we were replacing the values with the average weight of same product, 4 product are unique and don't have the any reference

In [12]:

```
#Get a boolean variable specifying missing Item_Weight values
miss_after_bool = df['Item_Weight'].isnull()
miss_after_bool.sum()
```

Out[12]:

```
4
```

These 4 Products are unique and we don't know the product weight.

In [13]:

```
miss_af = df.loc[miss_after_bool,'Item_Identifier']
miss_af
```

Out[13]:

```
927     FDN52
1922    FDK57
4187    FDE52
5022    FDQ60
Name: Item_Identifier, dtype: object
```

In [14]:

```
# Displaying the Product having missing weight
df.loc[miss_after_bool,:]
```

Out[14]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishm
927	FDN52	NaN	Regular	0.130933	Frozen Foods	86.9198	OUT027	
1922	FDK57	NaN	Low Fat	0.079904	Snack Foods	120.0440	OUT027	
4187	FDE52	NaN	Regular	0.029742	Dairy	88.9514	OUT027	
5022	FDQ60	NaN	Regular	0.191501	Baking Goods	121.2098	OUT019	

In [15]:

```
df.shape
```

Out[15]:

```
(8523, 12)
```

To deal with these missing values we simply ignore and delete the observation

In [16]:

```
df = df.loc[~miss_after_bool,:]
```

In [17]:

```
df.shape
```

Out[17]:

```
(8519, 12)
```

Lets impute Outlet_Size with the mode of the Outlet_Size for the particular type of outlet.

In [18]:

```
#Import mode function:
from scipy.stats import mode

def impute_Outlet_size(df):
    #Determing the mode for each
    outlet_size_mode = df.pivot_table(values='Outlet_Size', columns='Outlet_Type',aggfun
c=(lambda x:mode(x).mode[0]) )
    print('Mode for each Outlet_Type:')
    print(outlet_size_mode)

    #Get a boolean variable specifying missing Item_Weight values
```

```
    miss_bool = df['Outlet_Size'].isnull()

    #Impute data and check #missing values before and after imputation to confirm
    print('\nOrignal #missing: %d'% sum(miss_bool))
    df.loc[miss_bool,'Outlet_Size'] = df.loc[miss_bool,'Outlet_Type'].apply(lambda x: ou
tlet_size_mode[x])
    print('\nFinal #missing: %d'%sum(df['Outlet_Size'].isnull()))
```

In [19]:

```
impute_Outlet_size(df)
```

```
Mode for each Outlet_Type:
Outlet_Type Grocery Store Supermarket Type1 Supermarket Type2  \
Outlet_Size         Small             Small            Medium

Outlet_Type Supermarket Type3
Outlet_Size            Medium

Orignal #missing: 2410

Final #missing: 0
```

In [20]:

```
df.to_csv("clean_1_train.csv",index = False)
```

Univariate and Bivariate Analysis

In [1]:

```
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
#Read files:
df = pd.read_csv("clean_1_train.csv")
df.head(3)
```

Out[2]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	

In [3]:

```
df.shape
```

Out[3]:

```
(8519, 12)
```

In [4]:

```
df.columns
```

Out[4]:

```
Index(['Item_Identifier', 'Item_Weight', 'Item_Fat_Content', 'Item_Visibility',
       'Item_Type', 'Item_MRP', 'Outlet_Identifier',
       'Outlet_Establishment_Year', 'Outlet_Size', 'Outlet_Location_Type',
       'Outlet_Type', 'Item_Outlet_Sales'],
      dtype='object')
```

In [5]:

```
overall_sales_rate = df['Item_Outlet_Sales'].mean()
overall_sales_rate
```

Out[5]:

```
2181.188779387252
```

In [6]:

```
df.groupby("Outlet_Identifier")["Item_Identifier"].count()
```

Out[6]:

```
Outlet_Identifier
OUT010    555
OUT013    932
OUT017    926
OUT018    928
OUT019    527
OUT027    932
OUT035    930
```

```
OUT045    929
OUT046    930
OUT049    930
Name: Item_Identifier, dtype: int64
```

Feature Correlation

In [7]:

```
corr_matrix=df.corr()
corr_matrix
```

Out[7]:

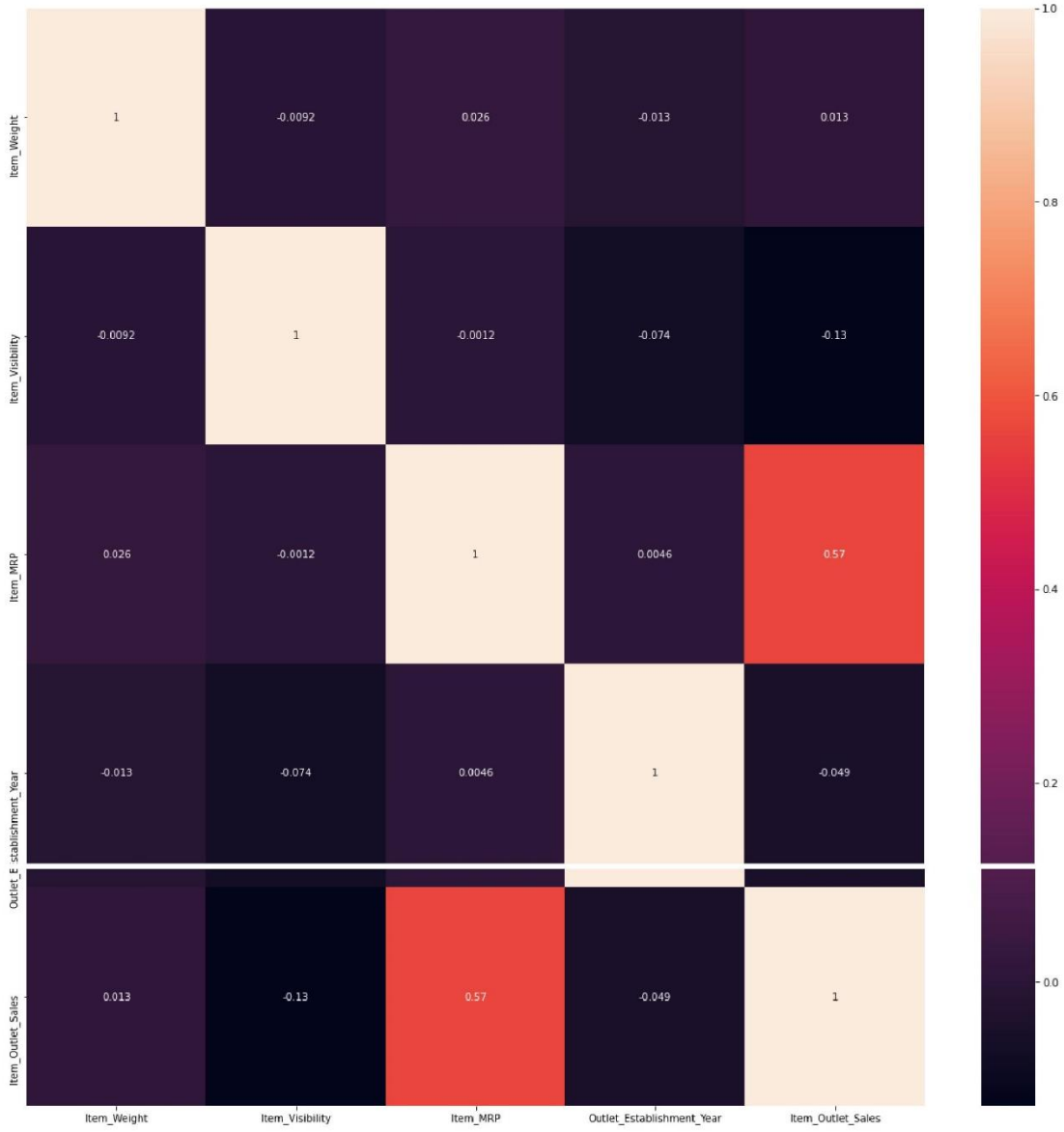
	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
Item_Weight	1.000000	-0.009173	0.025975	-0.013426	0.013168
Item_Visibility	-0.009173	1.000000	-0.001155	-0.074325	-0.128297
Item_MRP	0.025975	-0.001155	1.000000	0.004599	0.567803
Outlet_Establishment_Year	-0.013426	-0.074325	0.004599	1.000000	-0.049083
Item_Outlet_Sales	0.013168	-0.128297	0.567803	-0.049083	1.000000

In [8]:

```
plt.figure(figsize = (20,20))
sns.heatmap(df.corr(),annot = True)
```

Out[8]:

<AxesSubplot:>



We see that the Item_MRP is highly correlated with Item_outlet_Sales. Therefore,we want to compute the association of the target with each feature while adjusting for the effect of the remaining features.

This can be done using Univariante and Bivariante linear regression.

Univariante Analysis

In [9]:

```
X=df['Item_MRP'].values.reshape(-1,1)
y=df['Item_Outlet_Sales']
```

In [10]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=24)
```

In [11]:

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(6815, 1)
(1704, 1)
(6815,)
(1704,)
```

In [12]:

```
from sklearn.linear_model import LinearRegression
lr_model = LinearRegression()
lr_model.fit(X_train,y_train)
```

Out[12]:

```
LinearRegression()
```

In [13]:

```
y_pred = lr_model.predict(X_train)
y_pred
```

Out[13]:

```
array([1638.06631253, 1827.84394572, 2657.62078823, ..., 2740.73188653,
       3565.88699524, 3359.76999251])
```

In [14]:

```
from sklearn.metrics import r2_score
```

```
score = r2_score(y_train,y_pred)
print("Score of Training:",100*score)
```

```
Score of Training: 32.031452064357325
```

In [15]:

```
from sklearn.metrics import mean_squared_error,make_scorer,mean_absolute_error
from sklearn.model_selection import cross_val_score

#Perform cross-validation:
cv_score = cross_val_score(lr_model,X_train, y_train, cv=20, scoring = make_scorer(mean_
squared_error))
cv_score = np.sqrt(np.abs(cv_score))

#Print model report:
print("\nModel Report")
print("MAE : %.4g" % np.sqrt(mean_absolute_error(y_train,y_pred)))
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y_train,y_pred)))
print("CV Score : Mean - %.4g | Std - %.4g | Min - %.4g | Max - %.4g" % (np.mean(cv_scor
e),np.std(cv_score),np.min(cv_score),np.max(cv_score)))
```

```
Model Report
MAE : 32.08
RMSE : 1400
CV Score : Mean - 1398 | Std - 79.86 | Min - 1229 | Max - 1494
```

Polynomial Regression

Create interaction features for the case study data using scikit-learn's PolynomialFeatures.

In [16]:

```
# importing libraries for polynomial transform
from sklearn.preprocessing import PolynomialFeatures
# for creating pipeline
from sklearn.pipeline import Pipeline
# creating pipeline and fitting it on data
Input=[('polynomial',PolynomialFeatures(degree=2)),('modal',LinearRegression())]
pipe=Pipeline(Input)
pipe.fit(X.reshape(-1,1),y)
```

Out[16]:

```
Pipeline(steps=[('polynomial', PolynomialFeatures()),
                ('modal', LinearRegression())])
```

In [17]:

```
poly_pred=pipe.predict(X.reshape(-1,1))
```

In [18]:

```
from sklearn.metrics import r2_score

score = r2_score(y,poly_pred)
print(100*score)
```

```
32.24684888585043
```

In [19]:

```
from sklearn.metrics import mean_squared_error,make_scorer,mean_absolute_error
from sklearn.model_selection import cross_val_score

#Perform cross-validation:
cv_score = cross_val_score(lr_model,X_train, y_train, cv=20, scoring = make_scorer(mean_
squared_error))
```

```
cv_score = np.sqrt(np.abs(cv_score))

#Print model report:
print("\nModel Report")
print("MAE : %.4g" % np.sqrt(mean_absolute_error(y_train,y_pred)))
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y_train,y_pred)))
print("CV Score : Mean - %.4g | Std - %.4g | Min - %.4g | Max - %.4g" % (np.mean(cv_scor
e),np.std(cv_score),np.min(cv_score),np.max(cv_score)))
```

```
Model Report
MAE : 32.08
RMSE : 1400
CV Score : Mean - 1398 | Std - 79.86 | Min - 1229 | Max - 1494
```

Bivariante Analysis

In [20]:

```
response_features = ['Item_Visibility','Item_MRP']
```

In [21]:

```
X=df[response_features].values
y=df['Item_Outlet_Sales']
```

In [22]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=24)
```

In [23]:

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(6815, 2)
(1704, 2)
(6815,)
(1704,)
```

In [24]:

```
from sklearn.linear_model import LinearRegression
lr_model = LinearRegression()
lr_model.fit(X_train,y_train)
```

Out[24]:

```
LinearRegression()
```

In [25]:

```
y_pred = lr_model.predict(X_train)
y_pred
```

Out[25]:

```
array([1708.35906422, 1819.13824543, 2819.8002026 , ..., 2674.40040624,
```

```
       3600.21174098, 3568.80878111])
```

In [26]:

```
from sklearn.metrics import r2_score

score = r2_score(y_train,y_pred)
print(100*score)
```

```
33.468271103932814
```

In [27]:

```
from sklearn.metrics import mean_squared_error,make_scorer,mean_absolute_error
from sklearn.model_selection import cross_val_score

#Perform cross-validation:
cv_score = cross_val_score(lr_model,X_train, y_train, cv=20, scoring = make_scorer(mean_
squared_error))
cv_score = np.sqrt(np.abs(cv_score))

#Print model report:
print("\nModel Report")
print("MAE : %.4g" % np.sqrt(mean_absolute_error(y_train,y_pred)))
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y_train,y_pred)))
print("CV Score : Mean - %.4g | Std - %.4g | Min - %.4g | Max - %.4g" % (np.mean(cv_scor
e),np.std(cv_score),np.min(cv_score),np.max(cv_score)))
```

```
Model Report
MAE : 31.97
RMSE : 1385
CV Score : Mean - 1383 | Std - 79.89 | Min - 1220 | Max - 1495
```

Polynomial Regression

Create interaction features for the case study data using scikit-learn's PolynomialFeatures.

In [28]:

```
# importing libraries for polynomial transform
from sklearn.preprocessing import PolynomialFeatures
# for creating pipeline
from sklearn.pipeline import Pipeline
# creating pipeline and fitting it on data
Input=[('polynomial',PolynomialFeatures(degree=2)),('model',LinearRegression())]
pipe=Pipeline(Input)
pipe.fit(X,y)
```

Out[28]:

```
Pipeline(steps=[('polynomial', PolynomialFeatures()),
                ('model', LinearRegression())])
```

In [29]:

```
poly_pred=pipe.predict(X)
```

In [30]:

```
from sklearn.metrics import r2_score

score = r2_score(y,poly_pred)
print(100*score)
```

```
34.95958877773293
```

In [31]:

```
from sklearn.metrics import mean_squared_error,make_scorer,mean_absolute_error
```

```
from sklearn.model_selection import cross_val_score

#Perform cross-validation:
cv_score = cross_val_score(lr_model,X_train, y_train, cv=20, scoring = make_scorer(mean_
squared_error))
cv_score = np.sqrt(np.abs(cv_score))

#Print model report:
print("\nModel Report")
print("MAE : %.4g" % np.sqrt(mean_absolute_error(y_train,y_pred)))
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y_train,y_pred)))
print("CV Score : Mean - %.4g | Std - %.4g | Min - %.4g | Max - %.4g" % (np.mean(cv_scor
e),np.std(cv_score),np.min(cv_score),np.max(cv_score)))
```

```
Model Report
MAE : 31.97
RMSE : 1385
CV Score : Mean - 1383 | Std - 79.89 | Min - 1220 | Max - 1495
```

Feature Engineering

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns
```

In [2]:

```
#Read files:
df = pd.read_csv("clean_1_train.csv")
df.head()
```

Out[2]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8519 entries, 0 to 8518
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            8519 non-null   object
 1   Item_Weight                8519 non-null   float64
 2   Item_Fat_Content           8519 non-null   object
 3   Item_Visibility            8519 non-null   float64
 4   Item_Type                  8519 non-null   object
 5   Item_MRP                   8519 non-null   float64
 6   Outlet_Identifier          8519 non-null   object
 7   Outlet_Establishment_Year  8519 non-null   int64
 8   Outlet_Size                8519 non-null   object
 9   Outlet_Location_Type       8519 non-null   object
 10  Outlet_Type                8519 non-null   object
 11  Item_Outlet_Sales          8519 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 798.8+ KB
```

In [4]:

```
df.isnull().sum()
```

Out[4]:

```
Item_Identifier              0
Item_Weight                  0
Item_Fat_Content             0
Item_Visibility              0
Item_Type                    0
Item_MRP                     0
Outlet_Identifier            0
```

```
Outlet_Establishment_Year    0
Outlet_Size                  0
Outlet_Location_Type         0
Outlet_Type                  0
Item_Outlet_Sales            0
dtype: int64
```

In [5]:

```
df.nunique()
```

Out[5]:

```
Item_Identifier              1555
Item_Weight                   474
Item_Fat_Content                5
Item_Visibility              7876
Item_Type                      16
Item_MRP                     5936
Outlet_Identifier              10
Outlet_Establishment_Year       9
Outlet_Size                     3
Outlet_Location_Type            3
Outlet_Type                     4
Item_Outlet_Sales            3493
dtype: int64
```

In [6]:

```
# Displaying the mean sales by type of store.
df.pivot_table(values='Item_Outlet_Sales',index='Outlet_Type')
```

Out[6]:

Outlet_Type	Item_Outlet_Sales
Grocery Store	340.031198
Supermarket Type1	2316.181148
Supermarket Type2	1995.498739
Supermarket Type3	3695.781505

Modify Item_Visibility

We noticed that the minimum value here is 0, which makes no practical sense.

In [7]:

```
def modify_item_visibility(df):
    #Determine average visibility of a product
    visibility_avg = df.pivot_table(values='Item_Visibility', index='Item_Identifier')

    #Impute 0 values with mean visibility of that product:
    miss_bool = (df['Item_Visibility'] == 0)

    print('Number of 0 values initially: %d'%sum(miss_bool))
    df.loc[miss_bool,'Item_Visibility'] = df.loc[miss_bool,'Item_Identifier'].apply(lamb
da x: visibility_avg.loc[x])
    print('Number of 0 values after modification: %d'%sum(df['Item_Visibility'] == 0))
```

In [8]:

```
modify_item_visibility(df)
```

```
Number of 0 values initially: 526
Number of 0 values after modification: 0
```

In [9]:

In [9]:

```
df["Item_Visibility"].describe()
```

Out[9]:

```
count    8519.000000
mean        0.069652
std         0.049798
min         0.003575
25%         0.031114
50%         0.056919
75%         0.097132
max         0.328391
Name: Item_Visibility, dtype: float64
```

In [10]:

```
df['Item_Identifier']
```

Out[10]:

```
0       FDA15
1       DRC01
2       FDN15
3       FDX07
4       NCD19
        ...
8514    FDF22
8515    FDS36
8516    NCJ29
8517    FDN46
8518    DRG01
Name: Item_Identifier, Length: 8519, dtype: object
```

In [ ]:

```

```

In [11]:

```
print('Frequency of Categories for varible Item Type')
df['Item_Type'].value_counts()
```

```
Frequency of Categories for varible Item Type
```

Out[11]:

```
Fruits and Vegetables    1232
Snack Foods              1199
Household                 910
Frozen Foods              855
Dairy                     681
Canned                    649
Baking Goods              647
Health and Hygiene        520
Soft Drinks               445
Meat                      425
Breads                    251
Hard Drinks               214
Others                    169
Starchy Foods             148
Breakfast                 110
Seafood                    64
Name: Item_Type, dtype: int64
```

In [12]:

```
df.groupby(["Item_Identifier","Item_Type"]).size()
```

Out[12]:

```
Item_Identifier  Item_Type
DRA12            Soft Drinks          6
```

```
DRA24            Soft Drinks           7
DRA59            Soft Drinks           8
DRB01            Soft Drinks           3
DRB13            Soft Drinks           5
                                      ..
NCZ30            Household             7
NCZ41            Health and Hygiene    5
NCZ42            Household             5
NCZ53            Health and Hygiene    5
NCZ54            Household             7
Length: 1555, dtype: int64
```

We can clearly observe that the First 2 characters of the Item ID is same for the One kind of Item Type. Example: DR is the code for Soft Drinks, NC is the code of Non- Consumable Products and FD is for Food products

Create a broad category of Type of Item

In [13]:

```
def broad_item_type(df):
    #Get the first two characters of ID:
    df['Item_Type_Combined'] = df['Item_Identifier'].apply(lambda x: x[0:2])
    #Rename them to more intuitive categories:
    df['Item_Type_Combined'] = df['Item_Type_Combined'].map({'FD':'Food',
                                                             'NC':'Non-Consumable',
                                                             'DR':'Drinks'})
    print(df['Item_Type_Combined'].value_counts())
```

In [14]:

```
broad_item_type(df)
```

```
Food              6121
Non-Consumable    1599
Drinks             799
Name: Item_Type_Combined, dtype: int64
```

Determine the years of operation of a store

In [15]:

```
df["Outlet_Establishment_Year"].value_counts()
```

Out[15]:

```
1985    1459
1987     932
1999     930
1997     930
2004     930
2002     929
2009     928
2007     926
1998     555
Name: Outlet_Establishment_Year, dtype: int64
```

In [16]:

```
df.groupby(["Outlet_Establishment_Year","Outlet_Identifier","Outlet_Type","Outlet_Locatio
n_Type"])["Item_Outlet_Sales"].mean()
```

Out[16]:

```
Outlet_Establishment_Year  Outlet_Identifier  Outlet_Type        Outlet_Location_Type
1985                       OUT019             Grocery Store      Tier 1
340.746838
                           OUT027             Supermarket Type3  Tier 3
3695.781505
1987                       OUT013             Supermarket Type1  Tier 3
```

```
2298.995256
1997                        OUT046             Supermarket Type1  Tier 1
2277.844267
1998                        OUT010             Grocery Store      Tier 3
339.351662
1999                        OUT049             Supermarket Type1  Tier 1
2348.354635
2002                        OUT045             Supermarket Type1  Tier 2
2192.384798
2004                        OUT035             Supermarket Type1  Tier 2
2438.841866
2007                        OUT017             Supermarket Type1  Tier 2
2340.675263
2009                        OUT018             Supermarket Type2  Tier 3
1995.498739
Name: Item_Outlet_Sales, dtype: float64
```

In [17]:

```
def cal_outlet_year(df):
    #Years:
    df['Outlet_Years'] = 2013 - df['Outlet_Establishment_Year']
    print(df['Outlet_Years'].describe())
```

In [18]:

```
cal_outlet_year(df)
```

```
count    8519.000000
mean       15.162108
std         8.369105
min         4.000000
25%         9.000000
50%        14.000000
75%        26.000000
max        28.000000
Name: Outlet_Years, dtype: float64
```

Modify categories of Item_Fat_Content

We found typos and difference in representation in categories of Item_Fat_Content variable.

In [19]:

```
def modify_item_fat_content(data):
    #Change categories of low fat:
    print('Original Categories:')
    print(data['Item_Fat_Content'].value_counts())

    print('\nModified Categories:')
    data['Item_Fat_Content'] = data['Item_Fat_Content'].replace({'LF':'Low Fat',
                                                                 'reg':'Regular',
                                                                 'low fat':'Low Fat'})
    print(data['Item_Fat_Content'].value_counts())
```

In [20]:

```
modify_item_fat_content(df)
```

```
Original Categories:
Low Fat    5088
Regular    2886
LF          316
reg         117
low fat     112
Name: Item_Fat_Content, dtype: int64

Modified Categories:
Low Fat    5516
Regular    3003
Name: Item_Fat_Content, dtype: int64
```

We have corrected the typos and the difference in representation but some non-consumables as well and a fat-content should not be specified for them. So we can also create a separate category for such kind of observations.

In [21]:

```
def non_consumable_category(data):
    #Mark non-consumables as separate category in low_fat:
    data.loc[data['Item_Type_Combined']=="Non-Consumable",'Item_Fat_Content'] = "Non-Edib
le"
    print(data['Item_Fat_Content'].value_counts())
```

In [22]:

```
non_consumable_category(df)
```

```
Low Fat       3917
Regular       3003
Non-Edible    1599
Name: Item_Fat_Content, dtype: int64
```

In [23]:

```
def Item_Visibility_MeanRatio(data):
    #Get all Item_Visibility mean values for respective Item_Identifier
    visibility_item_avg = data.pivot_table(values='Item_Visibility',index='Item_Identifi
er')

    func = lambda x: x['Item_Visibility']/visibility_item_avg['Item_Visibility'][visibil
ity_item_avg.index == x['Item_Identifier']][0]
    data['Item_Visibility'] = data.apply(func,axis=1).astype(float)
    data['Item_Visibility'].describe()
```

In [24]:

```
Item_Visibility_MeanRatio(df)
```

In [25]:

```
df.head()
```

Out[25]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment
0	FDA15	9.30	Low Fat	0.922960	Dairy	249.8092	OUT049	
1	DRC01	5.92	Regular	1.003057	Soft Drinks	48.2692	OUT018	
2	FDN15	17.50	Low Fat	0.831990	Meat	141.6180	OUT049	
3	FDX07	19.20	Regular	0.750000	Fruits and Vegetables	182.0950	OUT010	
4	NCD19	8.93	Non-Edible	0.666667	Household	53.8614	OUT013	

In [26]:

```
df.shape
```

Out[26]:

```
(8519, 14)
```

Numerical and One-Hot Coding of Categorical variables

In [27]:

```
df.nunique()
```

Out[27]:

```
Item_Identifier              1555
Item_Weight                   474
Item_Fat_Content                3
Item_Visibility              8032
Item_Type                      16
Item_MRP                     5936
Outlet_Identifier              10
Outlet_Establishment_Year       9
Outlet_Size                     3
Outlet_Location_Type            3
Outlet_Type                     4
Item_Outlet_Sales            3493
Item_Type_Combined              3
Outlet_Years                    9
dtype: int64
```

Label Encoding

In [28]:

```
#Import library:
from sklearn.preprocessing import LabelEncoder

def label_encoding(df):
    le = LabelEncoder()
    #New variable for outlet
    df['Outlet'] = le.fit_transform(df['Outlet_Identifier'])
    df['Outlet']
    var_mod = ['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','Item_Type_Combine
d','Outlet_Type','Outlet']
    le = LabelEncoder()
    for i in var_mod:
        df[i] = le.fit_transform(df[i])
```

In [29]:

```
label_encoding(df)
```

In [30]:

```
df.head()
```

Out[30]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment
0	FDA15	9.30	0	0.922960	Dairy	249.8092	OUT049	
1	DRC01	5.92	2	1.003057	Soft Drinks	48.2692	OUT018	
2	FDN15	17.50	0	0.831990	Meat	141.6180	OUT049	
3	FDX07	19.20	2	0.750000	Fruits and Vegetables	182.0950	OUT010	
4	NCD19	8.93	1	0.666667	Household	53.8614	OUT013	

In [31]:

```
df.shape
```

(8519, 15)

In [32]:

```
def One_hot_encoding(df):
    #One Hot Coding:
    df = pd.get_dummies(df, columns=['Item_Fat_Content','Outlet_Location_Type','Outlet_S
ize','Outlet_Type','Item_Type_Combined','Outlet'],drop_first = True)

    return df
```

In [33]:

```
df = One_hot_encoding(df)
```

In [34]:

```
df.head()
```

Out[34]:

	Item_Identifier	Item_Weight	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Item_Outlet
0	FDA15	9.30	0.922960	Dairy	249.8092	OUT049	1999	373
1	DRC01	5.92	1.003057	Soft Drinks	48.2692	OUT018	2009	44
2	FDN15	17.50	0.831990	Meat	141.6180	OUT049	1999	209
3	FDX07	19.20	0.750000	Fruits and Vegetables	182.0950	OUT010	1998	73
4	NCD19	8.93	0.666667	Household	53.8614	OUT013	1987	99

5 rows × 29 columns

In [35]:

```
df.shape
```

Out[35]:

(8519, 29)

In [36]:

```
df.dtypes
```

Out[36]:

```
Item_Identifier               object
Item_Weight                  float64
Item_Visibility              float64
Item_Type                     object
Item_MRP                     float64
Outlet_Identifier             object
Outlet_Establishment_Year      int64
Item_Outlet_Sales            float64
Outlet_Years                   int64
Item_Fat_Content_1             uint8
Item_Fat_Content_2             uint8
Outlet_Location_Type_1         uint8
Outlet_Location_Type_2         uint8
Outlet_Size_1                  uint8
Outlet_Size_2                  uint8
Outlet_Type_1                  uint8
Outlet_Type_2                  uint8
Outlet_Type_3                  uint8
Item_Type_Combined_1           uint8
Item_Type_Combined_2           uint8
Outlet_1                       uint8
Outlet_2                       uint8
```

```
Outlet_3                       uint8
Outlet_4                       uint8
Outlet_5                       uint8
Outlet_6                       uint8
Outlet_7                       uint8
Outlet_8                       uint8
Outlet_9                       uint8
dtype: object
```

In [37]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8519 entries, 0 to 8518
Data columns (total 29 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            8519 non-null   object
 1   Item_Weight                8519 non-null   float64
 2   Item_Visibility            8519 non-null   float64
 3   Item_Type                  8519 non-null   object
 4   Item_MRP                   8519 non-null   float64
 5   Outlet_Identifier          8519 non-null   object
 6   Outlet_Establishment_Year  8519 non-null   int64
 7   Item_Outlet_Sales          8519 non-null   float64
 8   Outlet_Years               8519 non-null   int64
 9   Item_Fat_Content_1         8519 non-null   uint8
 10  Item_Fat_Content_2         8519 non-null   uint8
 11  Outlet_Location_Type_1     8519 non-null   uint8
 12  Outlet_Location_Type_2     8519 non-null   uint8
 13  Outlet_Size_1              8519 non-null   uint8
 14  Outlet_Size_2              8519 non-null   uint8
 15  Outlet_Type_1              8519 non-null   uint8
 16  Outlet_Type_2              8519 non-null   uint8
 17  Outlet_Type_3              8519 non-null   uint8
 18  Item_Type_Combined_1       8519 non-null   uint8
 19  Item_Type_Combined_2       8519 non-null   uint8
 20  Outlet_1                   8519 non-null   uint8
 21  Outlet_2                   8519 non-null   uint8
 22  Outlet_3                   8519 non-null   uint8
 23  Outlet_4                   8519 non-null   uint8
 24  Outlet_5                   8519 non-null   uint8
 25  Outlet_6                   8519 non-null   uint8
 26  Outlet_7                   8519 non-null   uint8
 27  Outlet_8                   8519 non-null   uint8
 28  Outlet_9                   8519 non-null   uint8
dtypes: float64(4), int64(2), object(3), uint8(20)
memory usage: 765.5+ KB
```

In [38]:

```
df.to_csv("clean_2_train.csv",index = False)
```

Fitting a Linear Regression Model

Import the Libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
import seaborn as sns
```

Import Dataset

In [2]:

```
#Read files:
df = pd.read_csv("clean_2_train.csv")
df.head()
```

Out[2]:

	Item_Identifier	Item_Weight	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Item_Outlet
0	FDA15	9.30	0.922960	Dairy	249.8092	OUT049	1999	373
1	DRC01	5.92	1.003057	Soft Drinks	48.2692	OUT018	2009	44
2	FDN15	17.50	0.831990	Meat	141.6180	OUT049	1999	209
3	FDX07	19.20	0.750000	Fruits and Vegetables	182.0950	OUT010	1998	73
4	NCD19	8.93	0.666667	Household	53.8614	OUT013	1987	99

5 rows × 29 columns

In [3]:

```
df.shape
```

Out[3]:

```
(8519, 29)
```

In [4]:

```
df.describe()
```

Out[4]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales	Outlet_Years	Item_Fat_Content
count	8519.000000	8519.000000	8519.000000	8519.000000	8519.000000	8519.000000	8519.0000
mean	12.875420	1.000000	141.010019	1997.837892	2181.188779	15.162108	0.1876
std	4.646098	0.196805	62.283594	8.369105	1706.511093	8.369105	0.3904
min	4.555000	0.636364	31.290000	1985.000000	33.290000	4.000000	0.0000
25%	8.785000	0.888335	93.844900	1987.000000	834.247400	9.000000	0.0000
50%	12.650000	0.943167	143.047000	1999.000000	1794.331000	14.000000	0.0000
75%	16.850000	1.003298	185.676600	2004.000000	3100.630600	26.000000	0.0000
max	21.350000	1.819614	266.888400	2009.000000	13086.964800	28.000000	1.0000

8 rows × 26 columns

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8519 entries, 0 to 8518
Data columns (total 29 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            8519 non-null   object
 1   Item_Weight                8519 non-null   float64
 2   Item_Visibility            8519 non-null   float64
 3   Item_Type                  8519 non-null   object
 4   Item_MRP                   8519 non-null   float64
 5   Outlet_Identifier          8519 non-null   object
 6   Outlet_Establishment_Year  8519 non-null   int64
 7   Item_Outlet_Sales          8519 non-null   float64
 8   Outlet_Years               8519 non-null   int64
 9   Item_Fat_Content_1         8519 non-null   int64
 10  Item_Fat_Content_2         8519 non-null   int64
 11  Outlet_Location_Type_1     8519 non-null   int64
 12  Outlet_Location_Type_2     8519 non-null   int64
 13  Outlet_Size_1              8519 non-null   int64
 14  Outlet_Size_2              8519 non-null   int64
 15  Outlet_Type_1              8519 non-null   int64
 16  Outlet_Type_2              8519 non-null   int64
 17  Outlet_Type_3              8519 non-null   int64
 18  Item_Type_Combined_1       8519 non-null   int64
 19  Item_Type_Combined_2       8519 non-null   int64
 20  Outlet_1                   8519 non-null   int64
 21  Outlet_2                   8519 non-null   int64
 22  Outlet_3                   8519 non-null   int64
 23  Outlet_4                   8519 non-null   int64
 24  Outlet_5                   8519 non-null   int64
 25  Outlet_6                   8519 non-null   int64
 26  Outlet_7                   8519 non-null   int64
 27  Outlet_8                   8519 non-null   int64
 28  Outlet_9                   8519 non-null   int64
dtypes: float64(4), int64(22), object(3)
memory usage: 1.9+ MB
```

Created a list remove_cols to remove those columns which doesn't required for Model Building

In [6]:

```
remove_cols = [
    'Item_Identifier',
    'Item_Type',
    'Outlet_Identifier',
    'Outlet_Establishment_Year'
]
df = df.drop(remove_cols,axis =1)
df.head()
```

Out[6]:

	Item_Weight	Item_Visibility	Item_MRP	Item_Outlet_Sales	Outlet_Years	Item_Fat_Content_1	Item_Fat_Content_2	Outlet_Lo
0	9.30	0.922960	249.8092	3735.1380	14	0	0	
1	5.92	1.003057	48.2692	443.4228	4	0	1	
2	17.50	0.831990	141.6180	2097.2700	14	0	0	
3	19.20	0.750000	182.0950	732.3800	15	0	1	
4	8.93	0.666667	53.8614	994.7052	26	1	0	

5 rows × 25 columns

In [7]:

```
df.shape
```

Out[7]:

```
(8519, 25)
```

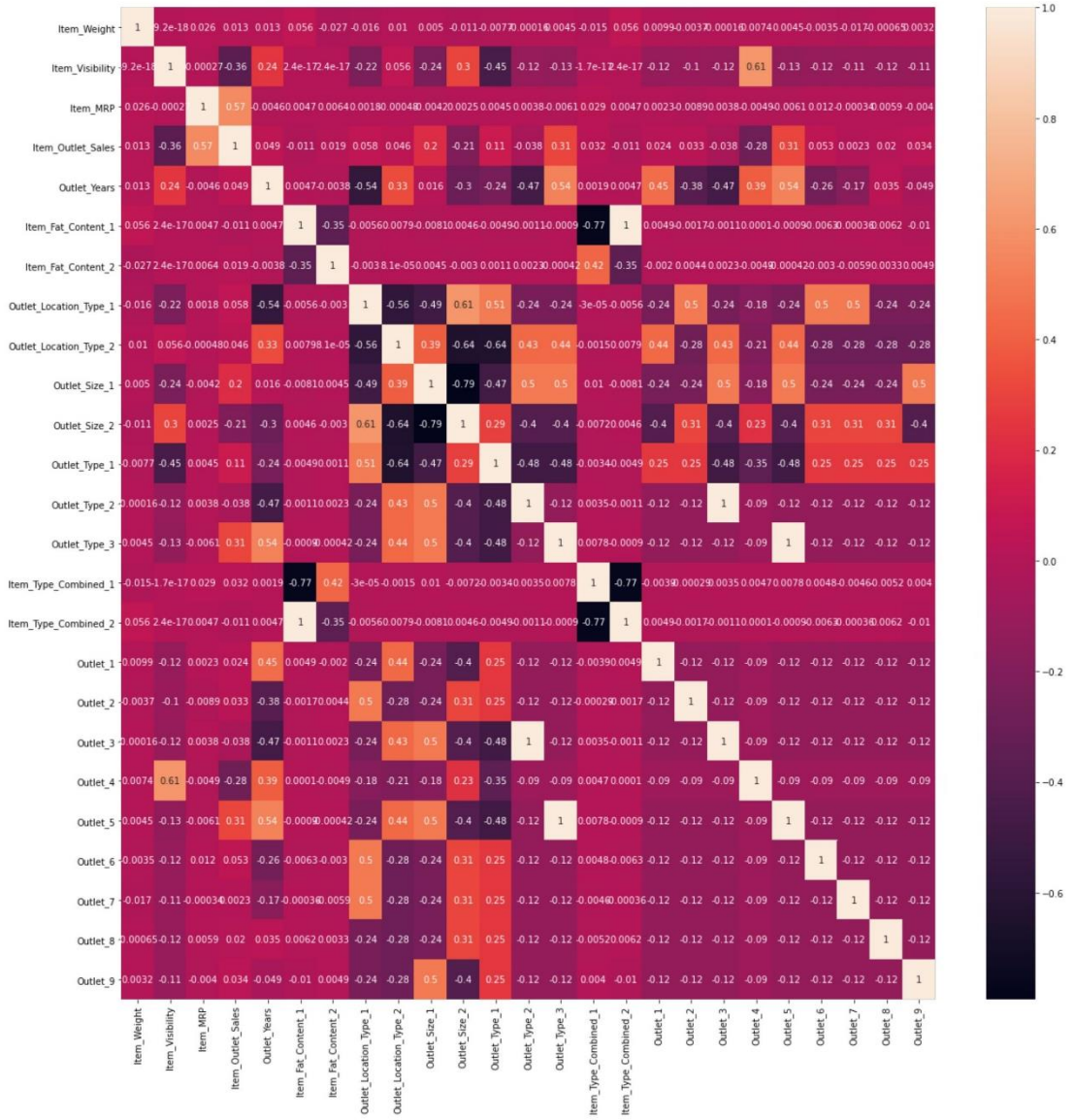
Correlation Matrix

In [8]:

```
plt.figure(figsize = (20,20))
sns.heatmap(df.corr(),annot = True)
```

Out[8]:

```
<AxesSubplot:>
```



Data Preprocessing

In [9]:

```
y = df.Item_Outlet_Sales.values
X = df.drop('Item_Outlet_Sales',axis = 1)
```

In [10]:

```
print(X.shape,y.shape)
```

```
(8519, 24) (8519,)
```

In [20]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [21]:

```
cols = [
    'Item_Weight',
    'Item_Visibility',
    'Item_MRP',
    'Outlet_Years'
]
X[cols]
```

Out[21]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Years
0	9.300	0.922960	249.8092	14
1	5.920	1.003057	48.2692	4
2	17.500	0.831990	141.6180	14
3	19.200	0.750000	182.0950	15
4	8.930	0.666667	53.8614	26
...	...	...	...	...
8514	6.865	0.920247	214.5218	26
8515	8.380	1.000657	108.1570	11
8516	10.600	0.999512	85.1224	9
8517	7.210	1.031393	103.1332	4
8518	14.800	0.870321	75.4670	16

8519 rows × 4 columns

In [22]:

```
X[cols]  =  sc.fit_transform(X[cols])
```

In [23]:

```
X.head()
```

Out[23]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Years	Item_Fat_Content_1	Item_Fat_Content_2	Outlet_Location_Type_1	Out
0	-0.769598	-0.391478	1.746938	-0.138865	0	0	0	
1	-1.497133	0.015532	-1.489096	-1.333806	0	1	0	
2	0.995427	-0.853739	0.009762	-0.138865	0	0	0	
3	1.361347	-1.270366	0.659682	-0.019371	0	1	0	

Trained a Linear Regression model

In [24]:

```
from sklearn.linear_model import LinearRegression
#Training the Simple Linear Regression model on the Training set
model = LinearRegression(normalize=True)
```

Now, train on the training data and obtain predicted classes, as well as class probabilities, using the testing data.

In [25]:

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

Out[25]:

```
LinearRegression(normalize=True)
```

In [26]:

```
#Predicting the Test set results
y_pred = model.predict(X)
```

In [27]:

```
from sklearn.metrics import r2_score,mean_squared_error

score = r2_score(y,y_pred)
print("Score of Training:",100*score)
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y,y_pred)))
```

```
Score of Training: 56.33267069746055
RMSE : 1128
```

In [28]:

```
from sklearn.metrics import mean_squared_error,make_scorer,mean_absolute_error
from sklearn.model_selection import cross_val_score

#Perform cross-validation:
cv_score = cross_val_score(model,X, y, cv=20, scoring = make_scorer(mean_squared_error))
cv_score = np.sqrt(np.abs(cv_score))

#Print model report:
print("\nModel Report")
print("MAE : %.4g" % np.sqrt(mean_absolute_error(y,y_pred)))
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y,y_pred)))
print("CV Score : Mean - %.4g | Std - %.4g | Min - %.4g | Max - %.4g" % (np.mean(cv_scor
e),np.std(cv_score),np.min(cv_score),np.max(cv_score)))
```

```
Model Report
MAE : 28.92
RMSE : 1128
CV Score : Mean - 1129 | Std - 42.62 | Min - 1075 | Max - 1208
```

```
model.coef_
```

Out[29]:

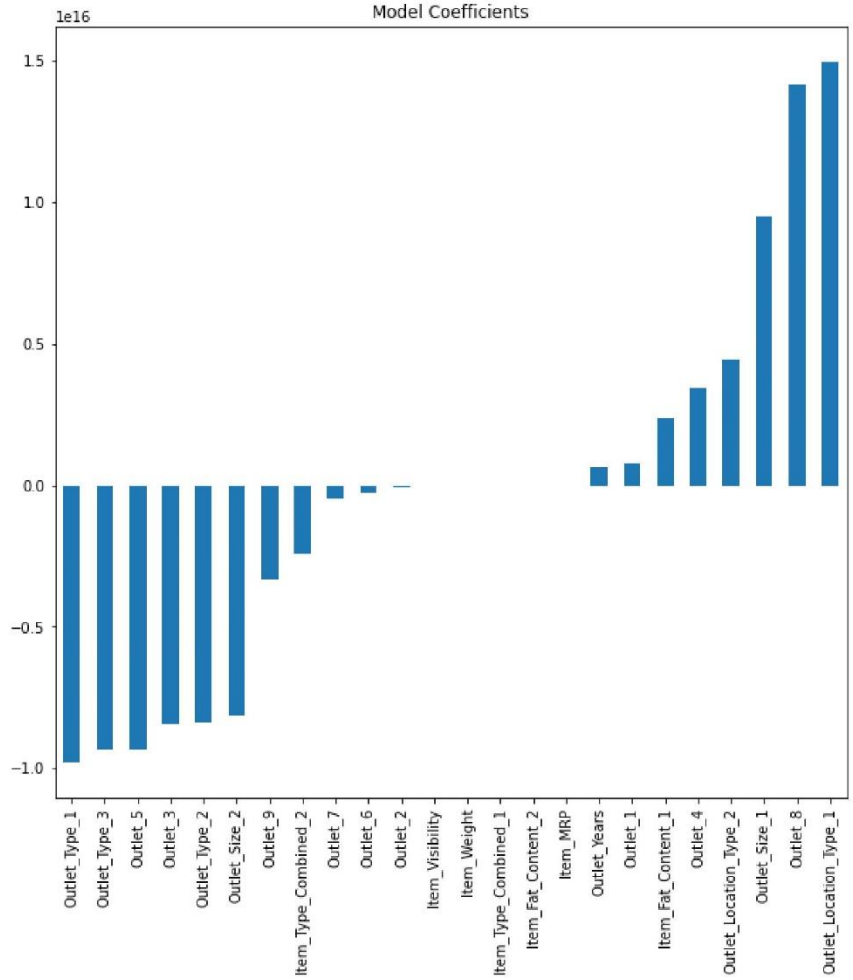
```
array([ 7.59621340e-01, -1.63013748e+00,  9.68752740e+02,  6.42432625e+14,
        2.39868917e+15,  3.88740975e+01,  1.49769646e+16,  4.44032299e+15,
        9.50808600e+15, -8.13284509e+15, -9.78551229e+15, -8.37176495e+15,
       -9.32521304e+15,  1.87205558e+01, -2.39868917e+15,  8.08231234e+14,
       -6.02271719e+13, -8.42473018e+15,  3.44235321e+15, -9.31368782e+15,
       -2.90527890e+14, -4.44061703e+14,  1.41490684e+16, -3.33832890e+15])
```

In [30]:

```
coef2 = pd.Series(model.coef_ ,X.columns).sort_values()
plt.figure(figsize = (10,10))
coef2.plot(kind='bar', title='Model Coefficients')
```

Out[30]:

```
<AxesSubplot:title={'center':'Model Coefficients'}>
```



Regularized Linear Regression

Ridge Regression

In [31]:

```
from sklearn.linear_model import Ridge
model_ridge = Ridge(alpha=0.05,normalize=True)
```

In [32]:

```
model_ridge.fit(X, y)
```

Out[32]:

```
Ridge(alpha=0.05, normalize=True)
```

In [33]:

```
#Predicting the Test set results
y_pred = model_ridge.predict(X)
```

In [34]:

```
score = r2_score(y,y_pred)
print("Score of Training:",100*score)
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y,y_pred)))
```

```
Score of Training: 56.09227723010813
RMSE : 1131
```

In [35]:

```
from sklearn.metrics import mean_squared_error,make_scorer,mean_absolute_error
from sklearn.model_selection import cross_val_score

#Perform cross-validation:
cv_score = cross_val_score(model,X, y, cv=20, scoring = make_scorer(mean_squared_error))
cv_score = np.sqrt(np.abs(cv_score))

#Print model report:
print("\nModel Report")
print("MAE : %.4g" % np.sqrt(mean_absolute_error(y,y_pred)))
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y,y_pred)))
print("CV Score : Mean - %.4g | Std - %.4g | Min - %.4g | Max - %.4g" % (np.mean(cv_scor
e),np.std(cv_score),np.min(cv_score),np.max(cv_score)))
```

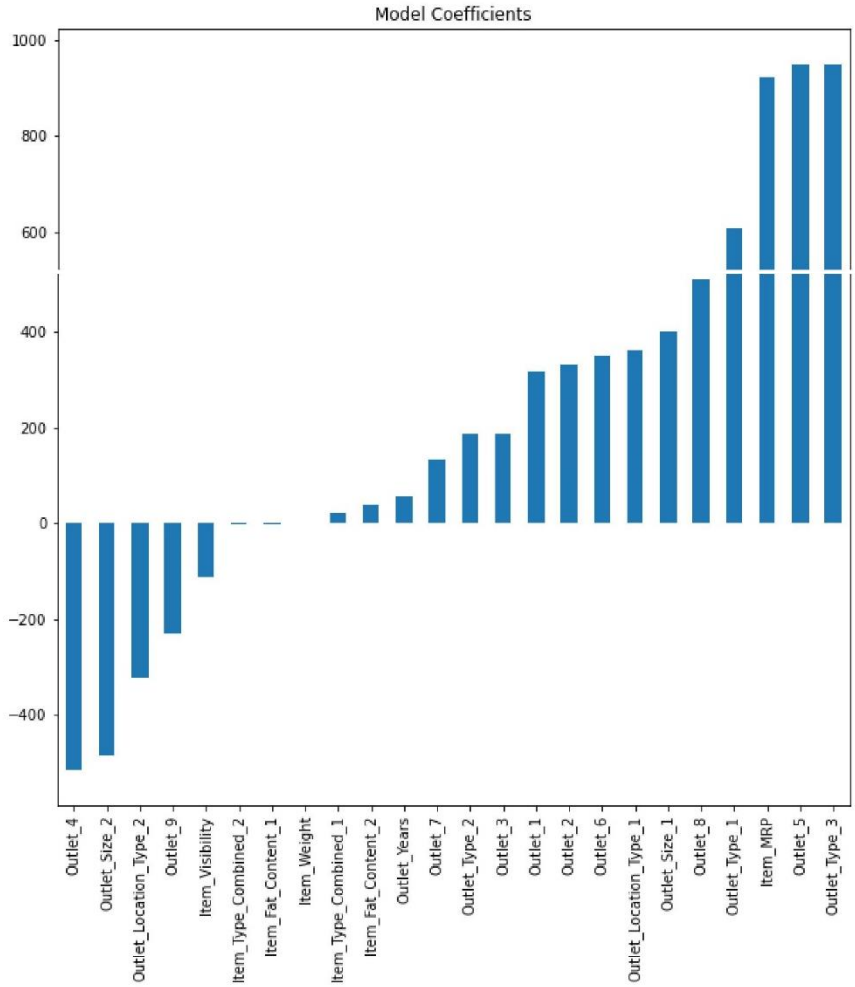
```
Model Report
MAE : 28.93
RMSE : 1131
CV Score : Mean - 1129 | Std - 42.62 | Min - 1075 | Max - 1208
```

In [36]:

```
coef2 = pd.Series(model_ridge.coef_ ,X.columns).sort_values()
plt.figure(figsize = (10,10))
coef2.plot(kind='bar', title='Model Coefficients')
```

Out[36]:

```
<AxesSubplot:title={'center':'Model Coefficients'}>
```



Decision Tree Regression Mod

Import Libraries

In [1]:

```
import numpy as np #numerical computation
import pandas as pd #data wrangling
import matplotlib.pyplot as plt #plotting package
```

Import Dataset

In [2]:

```
df = pd.read_csv('clean_2_train.csv')
df.head()
```

Out[2]:

	Item_Identifier	Item_Weight	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Item_Outlet
0	FDA15	9.30	0.922960	Dairy	249.8092	OUT049	1999	373
1	DRC01	5.92	1.003057	Soft Drinks	48.2692	OUT018	2009	44
2	FDN15	17.50	0.831990	Meat	141.6180	OUT049	1999	209
3	FDX07	19.20	0.750000	Fruits and Vegetables	182.0950	OUT010	1998	73
4	NCD19	8.93	0.666667	Household	53.8614	OUT013	1987	99

5 rows × 29 columns

Created a list remove_cols to remove those columns which doesn't required for Model Building

In [3]:

```
remove_cols = [
    'Item_Identifier',
    'Item_Type',
    'Outlet_Identifier',
    'Outlet_Establishment_Year'
]
df = df.drop(remove_cols,axis =1)
df.head()
```

Out[3]:

	Item_Weight	Item_Visibility	Item_MRP	Item_Outlet_Sales	Outlet_Years	Item_Fat_Content_1	Item_Fat_Content_2	Outlet_Lo
0	9.30	0.922960	249.8092	3735.1380	14	0	0	
1	5.92	1.003057	48.2692	443.4228	4	0	1	
2	17.50	0.831990	141.6180	2097.2700	14	0	0	
3	19.20	0.750000	182.0950	732.3800	15	0	1	
4	8.93	0.666667	53.8614	994.7052	26	1	0	

5 rows × 25 columns

In [4]:

```
df.shape
```

Out[4]:

```
(8519, 25)
```

Data Preprocessing

In [5]:

```
y = df.Item_Outlet_Sales.values
X = df.drop('Item_Outlet_Sales',axis = 1)
```

In [6]:

```
print(X.shape,y.shape)
```

```
(8519, 24) (8519,)
```

Trained a Random Forest model

In [7]:

```
from sklearn.tree import DecisionTreeRegressor
DT = DecisionTreeRegressor()
```

List of parameters for hyperparameter tuning

In [8]:

```
param = {
    'max_depth':[6,9,12,15],
    'min_samples_leaf':[10,50,100,150]
}
```

Hyperparameter optimization using RandomizedSearchCV

In [11]:

```
from sklearn.metrics import mean_squared_error,make_scorer
from sklearn.model_selection import RandomizedSearchCV
random_search=RandomizedSearchCV(DT,param_distributions=param,n_iter=5,scoring=make_score
r(mean_squared_error),n_jobs=-1,cv=5,verbose=3)
```

In [12]:

```
random_search.fit(X,y)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  19 out of  25 | elapsed:    3.0s remaining:    0.9s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    3.1s finished
```

Out[12]:

```
RandomizedSearchCV(cv=5, estimator=DecisionTreeRegressor(), n_iter=5, n_jobs=-1,
                   param_distributions={'max_depth': [6, 9, 12, 15],
                                        'min_samples_leaf': [10, 50, 100, 150]},
                   scoring=make_scorer(mean_squared_error), verbose=3)
```

In [13]:

```
means = random_search.cv_results_['mean_test_score']
params = random_search.cv_results_['params']
for mean, param in zip(means, params):
    print("%f with: %r" % (mean, param))
    if mean == min(means):
```

```
        print('Best parameters with the minimum Mean Square Error are:',param)
```

```
1218784.768530 with: {'min_samples_leaf': 100, 'max_depth': 6}
Best parameters with the minimum Mean Square Error are: {'min_samples_leaf': 100, 'max_de
pth': 6}
1234745.474520 with: {'min_samples_leaf': 100, 'max_depth': 15}
1253119.899713 with: {'min_samples_leaf': 50, 'max_depth': 12}
1238998.459008 with: {'min_samples_leaf': 10, 'max_depth': 6}
1243248.046485 with: {'min_samples_leaf': 50, 'max_depth': 9}
```

Evaluating the model for Train and Test set

In [14]:

```
DT = DecisionTreeRegressor(min_samples_leaf=100, max_depth=6)
```

In [15]:

```
DT.fit(X,y)
```

Out[15]:

```
DecisionTreeRegressor(max_depth=6, min_samples_leaf=100)
```

In [16]:

```
y_pred = DT.predict(X)
```

In [17]:

```
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error

score = r2_score(y,y_pred)
print("Score:",100*score)
print("MAE : %.4g" % np.sqrt(mean_absolute_error(y,y_pred)))
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y,y_pred)))
```

```
Score: 59.532325752312374
MAE : 27.6
RMSE : 1086
```

In [18]:

```
from sklearn.metrics import mean_squared_error,make_scorer,mean_absolute_error
from sklearn.model_selection import cross_val_score

#Perform cross-validation:
cv_score = cross_val_score(DT,X, y, cv=20, scoring = make_scorer(mean_squared_error))
cv_score = np.sqrt(np.abs(cv_score))

#Print model report:
print("\nModel Report")
print("MAE : %.4g" % np.sqrt(mean_absolute_error(y,y_pred)))
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y,y_pred)))
print("CV Score : Mean - %.4g | Std - %.4g | Min - %.4g | Max - %.4g" % (np.mean(cv_scor
e),np.std(cv_score),np.min(cv_score),np.max(cv_score)))
```

```
Model Report
MAE : 27.6
RMSE : 1086
CV Score : Mean - 1097 | Std - 45.76 | Min - 1032 | Max - 1211
```

Random Forest Regression Model

Import Libraries

In [1]:

```
import numpy as np #numerical computation
import pandas as pd #data wrangling
import matplotlib.pyplot as plt #plotting package
```

Import Dataset

In [2]:

```
df = pd.read_csv('clean_2_train.csv')
df.head()
```

Out[2]:

	Item_Identifier	Item_Weight	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Item_Outlet
0	FDA15	9.30	0.922960	Dairy	249.8092	OUT049	1999	373
1	DRC01	5.92	1.003057	Soft Drinks	48.2692	OUT018	2009	44
2	FDN15	17.50	0.831990	Meat	141.6180	OUT049	1999	209
3	FDX07	19.20	0.750000	Fruits and Vegetables	182.0950	OUT010	1998	73
4	NCD19	8.93	0.666667	Household	53.8614	OUT013	1987	99

5 rows × 29 columns

Created a list remove_cols to remove those columns which doesn't required for Model Building

In [3]:

```
remove_cols = [
    'Item_Identifier',
    'Item_Type',
    'Outlet_Identifier',
    'Outlet_Establishment_Year'
]
df = df.drop(remove_cols,axis =1)
df.head()
```

Out[3]:

	Item_Weight	Item_Visibility	Item_MRP	Item_Outlet_Sales	Outlet_Years	Item_Fat_Content_1	Item_Fat_Content_2	Outlet_Lo
0	9.30	0.922960	249.8092	3735.1380	14	0	0	
1	5.92	1.003057	48.2692	443.4228	4	0	1	
2	17.50	0.831990	141.6180	2097.2700	14	0	0	
3	19.20	0.750000	182.0950	732.3800	15	0	1	
4	8.93	0.666667	53.8614	994.7052	26	1	0	

5 rows × 25 columns

In [4]:

```
df.shape
```

Out[4]:

```
(8519, 25)
```

Data Preprocessing

In [5]:

```
y = df.Item_Outlet_Sales.values
X = df.drop('Item_Outlet_Sales',axis = 1)
```

In [6]:

```
print(X.shape,y.shape)
```

```
(8519, 24) (8519,)
```

Trained a Random Forest model

In [7]:

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()
```

List of parameters for hyperparameter tuning

In [8]:

```
param = {
    'max_depth':[3,6,9,12],
    'n_estimators' : [10,50,100,200]
}
```

Hyperparameter optimization using RandomizedSearchCV

In [9]:

```
from sklearn.metrics import mean_squared_error,make_scorer
from sklearn.model_selection import RandomizedSearchCV
random_search=RandomizedSearchCV(rf,param_distributions=param,n_iter=5,scoring=make_score
r(mean_squared_error),n_jobs=-1,cv=5,verbose=3)
```

In [10]:

```
random_search.fit(X,y)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  19 out of  25 | elapsed:   11.9s remaining:    3.7s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:   14.2s finished
```

Out[10]:

```
RandomizedSearchCV(cv=5, estimator=RandomForestRegressor(), n_iter=5, n_jobs=-1,
                   param_distributions={'max_depth': [3, 6, 9, 12],
                                        'n_estimators': [10, 50, 100, 200]},
                   scoring=make_scorer(mean_squared_error), verbose=3)
```

In [11]:

```
means = random_search.cv_results_['mean_test_score']
params = random_search.cv_results_['params']
for mean, param in zip(means, params):
    print("%f with: %r" % (mean, param))
    if mean == min(means):
```

```
        print('Best parameters with the minimum Mean Square Error are:',param)
```

```
1212171.965558 with: {'n_estimators': 50, 'max_depth': 9}
1203943.597910 with: {'n_estimators': 100, 'max_depth': 9}
Best parameters with the minimum Mean Square Error are: {'n_estimators': 100, 'max_depth'
: 9}
1242554.264999 with: {'n_estimators': 50, 'max_depth': 12}
1387927.715629 with: {'n_estimators': 200, 'max_depth': 3}
1204071.376814 with: {'n_estimators': 10, 'max_depth': 6}
```

Evaluating the model for Train and Test set

In [12]:

```
rf = RandomForestRegressor(
    n_estimators=100, max_depth=6,
)
```

In [13]:

```
rf.fit(X,y)
```

Out[13]:

```
RandomForestRegressor(max_depth=6)
```

In [14]:

```
y_pred = rf.predict(X)
```

In [17]:

```
from sklearn.metrics import r2_score,mean_squared_error

score = r2_score(y,y_pred)
print("Score:",100*score)
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y,y_pred)))
```

```
Score: 61.53361077164543
RMSE : 1058
```

In [18]:

```
from sklearn.metrics import mean_squared_error,make_scorer,mean_absolute_error
from sklearn.model_selection import cross_val_score

#Perform cross-validation:
cv_score = cross_val_score(rf,X, y, cv=20, scoring = make_scorer(mean_squared_error))
cv_score = np.sqrt(np.abs(cv_score))

#Print model report:
print("\nModel Report")
print("MAE : %.4g" % np.sqrt(mean_absolute_error(y,y_pred)))
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y,y_pred)))
print("CV Score : Mean - %.4g | Std - %.4g | Min - %.4g | Max - %.4g" % (np.mean(cv_scor
e),np.std(cv_score),np.min(cv_score),np.max(cv_score)))
```

```
Model Report
MAE : 27.3
RMSE : 1058
CV Score : Mean - 1090 | Std - 47.1 | Min - 1021 | Max - 1210
```

XGBoost Regression Model

Import Libraries

In [1]:

```
import numpy as np #numerical computation
import pandas as pd #data wrangling
import matplotlib.pyplot as plt #plotting package
```

Import Dataset

In [2]:

```
df = pd.read_csv('clean_2_train.csv')
df.head()
```

Out[2]:

	Item_Identifier	Item_Weight	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Item_Outlet
0	FDA15	9.30	0.922960	Dairy	249.8092	OUT049	1999	373
1	DRC01	5.92	1.003057	Soft Drinks	48.2692	OUT018	2009	44
2	FDN15	17.50	0.831990	Meat	141.6180	OUT049	1999	209
3	FDX07	19.20	0.750000	Fruits and Vegetables	182.0950	OUT010	1998	73
4	NCD19	8.93	0.666667	Household	53.8614	OUT013	1987	99

5 rows × 29 columns

Created a list remove_cols to remove those columns which doesn't required for Model Building

In [3]:

```
remove_cols = [
    'Item_Identifier',
    'Item_Type',
    'Outlet_Identifier',
    'Outlet_Establishment_Year'
]
df = df.drop(remove_cols,axis =1)
df.head()
```

Out[3]:

	Item_Weight	Item_Visibility	Item_MRP	Item_Outlet_Sales	Outlet_Years	Item_Fat_Content_1	Item_Fat_Content_2	Outlet_Lo
0	9.30	0.922960	249.8092	3735.1380	14	0	0	
1	5.92	1.003057	48.2692	443.4228	4	0	1	
2	17.50	0.831990	141.6180	2097.2700	14	0	0	
3	19.20	0.750000	182.0950	732.3800	15	0	1	
4	8.93	0.666667	53.8614	994.7052	26	1	0	

5 rows × 25 columns

In [4]:

```
df.shape
```

Out[4]:

```
(8519, 25)
```

Data Preprocessing

In [5]:

```
y = df.Item_Outlet_Sales.values
X = df.drop('Item_Outlet_Sales',axis = 1)
```

In [6]:

```
print(X.shape,y.shape)
```

```
(8519, 24) (8519,)
```

In [7]:

```
X.head()
```

Out[7]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Years	Item_Fat_Content_1	Item_Fat_Content_2	Outlet_Location_Type_1	Out
0	9.30	0.922960	249.8092	14	0	0	0	
1	5.92	1.003057	48.2692	4	0	1	0	
2	17.50	0.831990	141.6180	14	0	0	0	
3	19.20	0.750000	182.0950	15	0	1	0	
4	8.93	0.666667	53.8614	26	1	0	0	

5 rows × 24 columns

Feature Scaling

Scaling the features using Standard Scaler

In [8]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
```

In [9]:

```
cols = [
    'Item_Weight',
    'Item_Visibility',
    'Item_MRP',
    'Outlet_Years'
]
X[cols]
```

Out[9]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Years
0	9.300	0.922960	249.8092	14
1	5.920	1.003057	48.2692	4
2	17.500	0.831990	141.6180	14
3	19.200	0.750000	182.0950	15
4	8.930	0.666667	53.8614	26

8514	6.865	0.920247	214.5218	26
8515	8.380	1.000657	108.1570	11
8516	10.600	0.999512	85.1224	9
8517	7.210	1.031393	103.1332	4
8518	14.800	0.870321	75.4670	16

8519 rows × 4 columns

In [69]:

```
X[cols]  =  sc.fit_transform(X[cols])
```

In [70]:

```
X.head()
```

Out[70]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Years	Item_Fat_Content_1	Item_Fat_Content_2	Outlet_Location_Type_1	Out
0	-0.769598	-0.391478	1.746938	-0.138865	0	0	0	
1	-1.497133	0.015532	-1.489096	-1.333806	0	1	0	
2	0.995427	-0.853739	0.009762	-0.138865	0	0	0	
3	1.361347	-1.270366	0.659682	-0.019371	0	1	0	
4	-0.849240	-1.693822	-1.399305	1.295064	1	0	0	

5 rows × 24 columns

Hyperparameter Tuning

In [71]:

```
## Hyper Parameter Optimization

params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]

}
```

In [72]:

```
## Hyperparameter optimization using RandomizedSearchCV
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
import xgboost
```

In [73]:

```
model = xgboost.XGBRegressor()
```

In [74]:

```
def timer(start_time=None):
    if not start_time:
        start_time = datetime.now()
        return start_time
    elif start_time:
        thour, temp_sec = divmod((datetime.now() - start_time).total_seconds(), 3600)
```

```
        tmin, tsec = divmod(temp_sec, 60)
        print('\n Time taken: %i hours %i minutes and %s seconds.' % (thour, tmin, round
(tsec, 2)))
```

Hyperparameter optimization using RandomizedSearchCV

In [75]:

```
from sklearn.metrics import mean_squared_error,make_scorer
random_search=RandomizedSearchCV(model,param_distributions=params,n_iter=5,scoring='neg_m
ean_squared_error',n_jobs=-1,cv=5,verbose=3)
```

In [76]:

```
from datetime import datetime
# Here we go
start_time = timer(None) # timing starts from this point for "start_time" variable
random_search.fit(X,y)
timer(start_time) # timing ends here for "start_time" variable
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  19 out of  25 | elapsed:    9.8s remaining:    3.0s
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:   11.5s finished

 Time taken: 0 hours 0 minutes and 12.21 seconds.
```

In [77]:

```
random_search
```

Out[77]:

```
RandomizedSearchCV(cv=5,
                   estimator=XGBRegressor(base_score=None, booster=None,
                                          colsample_bylevel=None,
                                          colsample_bynode=None,
                                          colsample_bytree=None, gamma=None,
                                          gpu_id=None, importance_type='gain',
                                          interaction_constraints=None,
                                          learning_rate=None,
                                          max_delta_step=None, max_depth=None,
                                          min_child_weight=None, missing=nan,
                                          monotone_constraints=None,
                                          n_estimators=100, n...
                                          scale_pos_weight=None, subsample=None,
                                          tree_method=None,
                                          validate_parameters=None,
                                          verbosity=None),
                   n_iter=5, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.3, 0.4, 0.5,
                                                             0.7],
                                        'gamma': [0.0, 0.1, 0.2, 0.3, 0.4],
                                        'learning_rate': [0.05, 0.1, 0.15, 0.2,
                                                          0.25, 0.3],
                                        'max_depth': [3, 4, 5, 6, 8, 10, 12,
                                                      15],
                                        'min_child_weight': [1, 3, 5, 7]},
                   scoring='neg_mean_squared_error', verbose=3)
```

In [78]:

```
means = random_search.cv_results_['mean_test_score']
params = random_search.cv_results_['params']
for mean, param in zip(means, params):
    print("%f with: %r" % (mean, param))
    if mean == min(means):
        print('Best parameters with the minimum Mean Square Error are:',param)
```

```
-1218153.413715 with: {'min_child_weight': 7, 'max_depth': 5, 'learning_rate': 0.15, 'gam
```

```
ma': 0.2, 'colsample_bytree': 0.4}
-1216276.986630 with: {'min_child_weight': 1, 'max_depth': 4, 'learning_rate': 0.15, 'gam
ma': 0.4, 'colsample_bytree': 0.7}
-1331299.184173 with: {'min_child_weight': 5, 'max_depth': 15, 'learning_rate': 0.05, 'ga
mma': 0.0, 'colsample_bytree': 0.7}
Best parameters with the minimum Mean Square Error are: {'min_child_weight': 5, 'max_dept
h': 15, 'learning_rate': 0.05, 'gamma': 0.0, 'colsample_bytree': 0.7}
-1265622.936871 with: {'min_child_weight': 7, 'max_depth': 10, 'learning_rate': 0.05, 'ga
mma': 0.2, 'colsample_bytree': 0.5}
-1254538.414191 with: {'min_child_weight': 5, 'max_depth': 4, 'learning_rate': 0.3, 'gamm
a': 0.1, 'colsample_bytree': 0.5}
```

In [79]:

```
random_search
```

Out[79]:

```
RandomizedSearchCV(cv=5,
                   estimator=XGBRegressor(base_score=None, booster=None,
                                          colsample_bylevel=None,
                                          colsample_bynode=None,
                                          colsample_bytree=None, gamma=None,
                                          gpu_id=None, importance_type='gain',
                                          interaction_constraints=None,
                                          learning_rate=None,
                                          max_delta_step=None, max_depth=None,
                                          min_child_weight=None, missing=nan,
                                          monotone_constraints=None,
                                          n_estimators=100, n...
                                          scale_pos_weight=None, subsample=None,
                                          tree_method=None,
                                          validate_parameters=None,
                                          verbosity=None),
                   n_iter=5, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.3, 0.4, 0.5,
                                                             0.7],
                                        'gamma': [0.0, 0.1, 0.2, 0.3, 0.4],
                                        'learning_rate': [0.05, 0.1, 0.15, 0.2,
                                                          0.25, 0.3],
                                        'max_depth': [3, 4, 5, 6, 8, 10, 12,
                                                      15],
                                        'min_child_weight': [1, 3, 5, 7]},
                   scoring='neg_mean_squared_error', verbose=3)
```

In [80]:

```
random_search.best_params_
```

Out[80]:

```
{'min_child_weight': 1,
 'max_depth': 4,
 'learning_rate': 0.15,
 'gamma': 0.4,
 'colsample_bytree': 0.7}
```

Best parameters obtain from Hyperparameter optimization with the minimum Mean Square Error are:

{'min_child_weight': 7, 'max_depth': 3, 'learning_rate': 0.1, 'gamma': 0.4, 'colsample_bytree': 0.4}

In [34]:

```
model = xgboost.XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0.6, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.4, max_delta_step=0, max_depth=15,
             min_child_weight=1, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
```

```
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [35]:

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

Out[35]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0.6, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.4, max_delta_step=0, max_depth=15,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [36]:

```
#Predicting the Test set results
y_pred = model.predict(X)
```

Evaluation using R2 Score and Mean Square Error

In [37]:

```
from sklearn.metrics import r2_score,mean_squared_error,mean_absolute_error

score = r2_score(y,y_pred)
print("Score of Training:",100*score)
print("MAE : %.4g" % np.sqrt(mean_absolute_error(y,y_pred)))
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y,y_pred)))
```

```
Score of Training: 99.99755323383201
MAE : 2.243
RMSE : 8.441
```

In [41]:

```
from sklearn.metrics import mean_squared_error,make_scorer
from sklearn.model_selection import cross_val_score

#Perform cross-validation:
cv_score = cross_val_score(model,X, y, cv=20, scoring='neg_mean_squared_error')
cv_score = np.sqrt(np.abs(cv_score))

#Print model report:
print("\nModel Report")
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y,y_pred)))
print("CV Score : Mean - %.4g | Std - %.4g | Min - %.4g | Max - %.4g" % (np.mean(cv_scor
e),np.std(cv_score),np.min(cv_score),np.max(cv_score)))
```

```
Model Report
RMSE : 8.441
CV Score : Mean - 1246 | Std - 59.53 | Min - 1145 | Max - 1351
```

Evaluating the model for Train and Test set

In [27]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y ,test_size = 0.2,random_state = 2 )
```

In [28]:

```
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
```

```
(6815, 30) (1704, 30) (6815,) (1704,)
```

In [29]:

```
#Predicting the Test set results
y_pred = model.predict(X_train)
```

In [30]:

```
from sklearn.metrics import r2_score,mean_squared_error

score = r2_score(y_train,y_pred)
print("Score of Training:",100*score)
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y_train,y_pred)))
```

```
Score of Training: 99.3195540115969
RMSE : 140.9
```

In [31]:

```
from sklearn.metrics import mean_squared_error,make_scorer
from sklearn.model_selection import cross_val_score

#Perform cross-validation:
cv_score = cross_val_score(model,X_train, y_train, cv=20, scoring = make_scorer(mean_squ
ared_error))
cv_score = np.sqrt(np.abs(cv_score))

#Print model report:
print("\nModel Report")
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y_train,y_pred)))
print("CV Score : Mean - %.4g | Std - %.4g | Min - %.4g | Max - %.4g" % (np.mean(cv_scor
e),np.std(cv_score),np.min(cv_score),np.max(cv_score)))
```

```
Model Report
RMSE : 140.9
CV Score : Mean - 1237 | Std - 62.73 | Min - 1157 | Max - 1342
```

In [32]:

```
y_test_pred = model.predict(X_test)
```

In [33]:

```
score = r2_score(y_test,y_test_pred)
print("Score of Testing:",100*score)
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y_test,y_test_pred)))
```

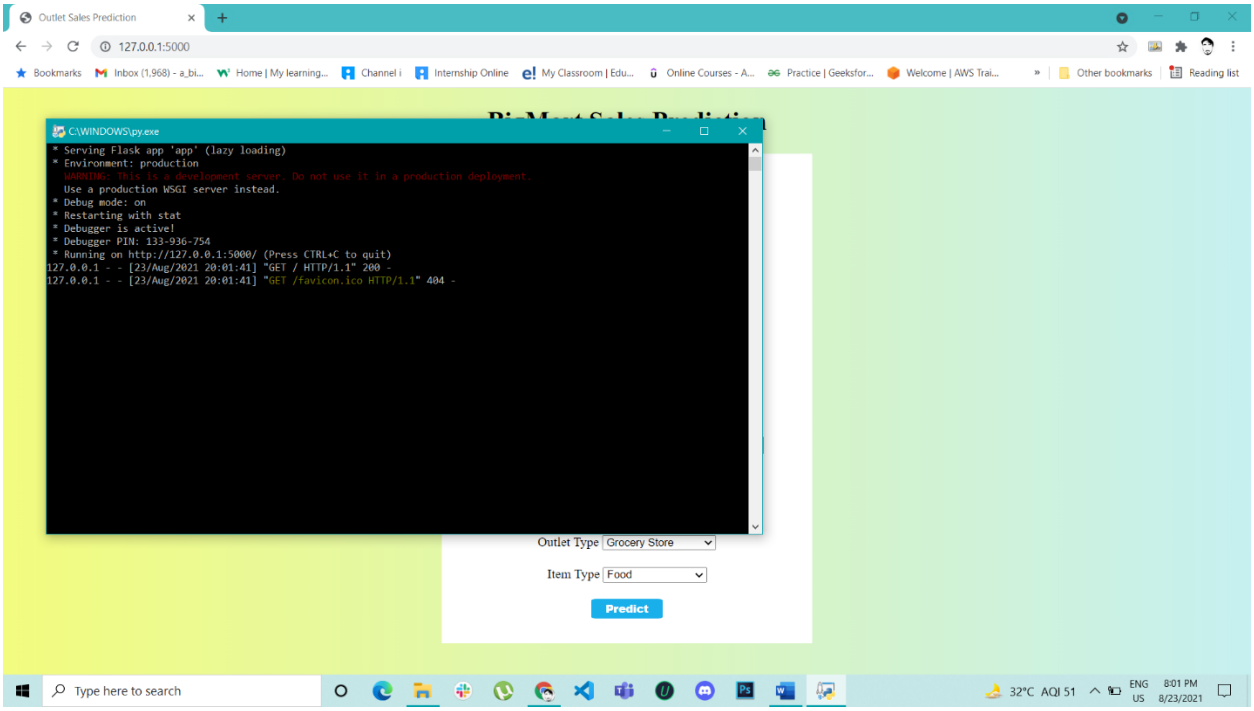
```
Score of Testing: 99.33178445289097
RMSE : 138.8
```

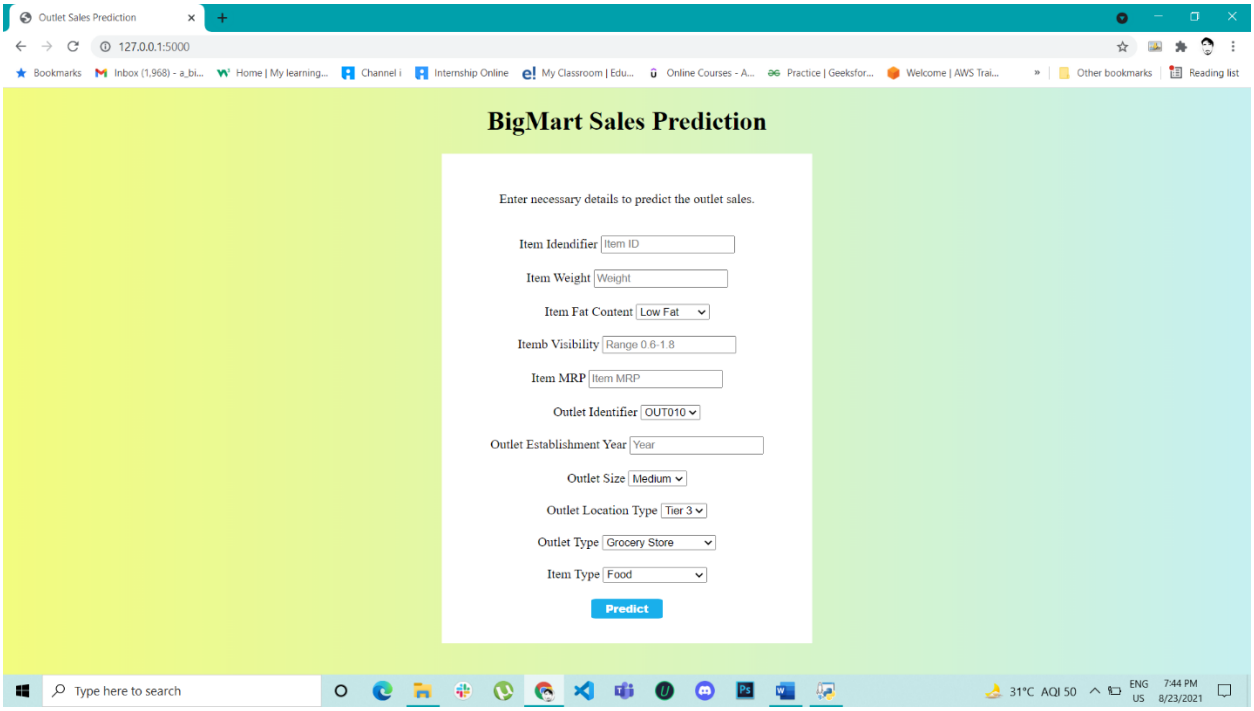
In [34]:

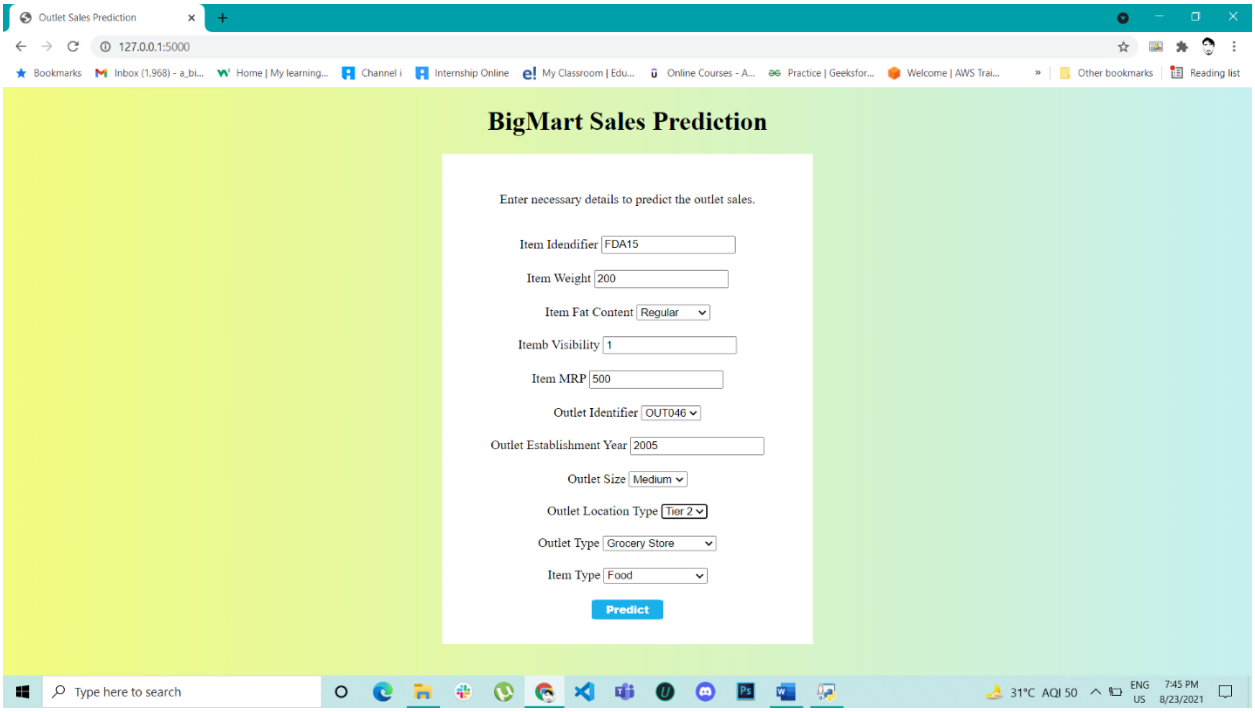
```
#Perform cross-validation:
cv_score = cross_val_score(model,X_test, y_test, cv=20, scoring = make_scorer(mean_squar
ed_error))
cv_score = np.sqrt(np.abs(cv_score))

#Print model report:
print("\nModel Report")
print("RMSE : %.4g" % np.sqrt(mean_squared_error(y_test,y_test_pred)))
print("CV Score : Mean - %.4g | Std - %.4g | Min - %.4g | Max - %.4g" % (np.mean(cv_scor
e),np.std(cv_score),np.min(cv_score),np.max(cv_score)))
```

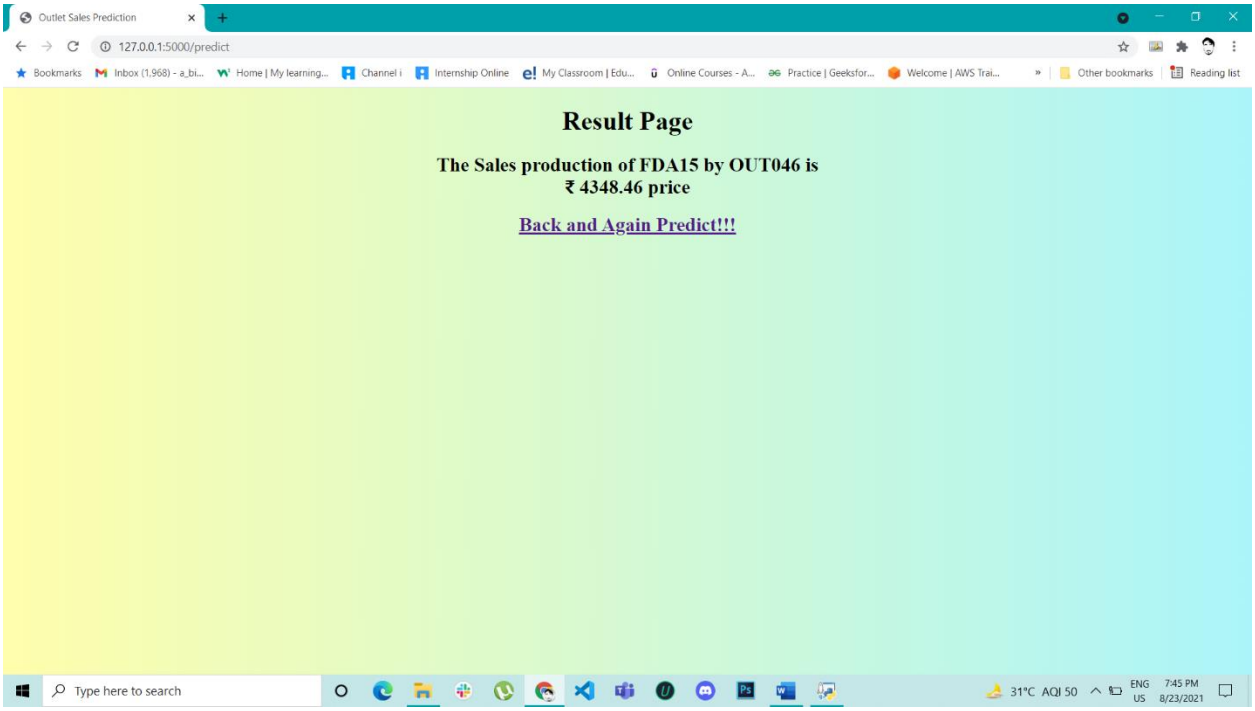
```
Model Report
RMSE : 138.8
CV Score : Mean - 1223 | Std - 120.4 | Min - 1043 | Max - 1464
```

RESULTS


Outlet Sales Prediction
127.0.0.1:5000
C:\WINDOWS\py.exe
* Serving Flask app 'app' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 133-936-754
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [23/Aug/2021 20:01:41] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [23/Aug/2021 20:01:41] "GET /favicon.ico HTTP/1.1" 404 -
Outlet Type Grocery Store
Item Type Food
Predict
Type here to search
32°C AQI 51
ENG US
8:01 PM 8/23/2021


Outlet Sales Prediction
127.0.0.1:5000
BigMart Sales Prediction
Enter necessary details to predict the outlet sales.
Item Idendifier Item ID
Item Weight Weight
Item Fat Content Low Fat
Itemb Visibility Range 0.6-1.8
Item MRP Item MRP
Outlet Identifier OUT010
Outlet Establishment Year Year
Outlet Size Medium
Outlet Location Type Tier 3
Outlet Type Grocery Store
Item Type Food
Predict
Type here to search
31°C AQI 50
ENG US
7:44 PM 8/23/2021


Outlet Sales Prediction
127.0.0.1:5000
BigMart Sales Prediction
Enter necessary details to predict the outlet sales.
Item Idendifier FDA15
Item Weight 200
Item Fat Content Regular
Itemb Visibility 1
Item MRP 500
Outlet Identifier OUT046
Outlet Establishment Year 2005
Outlet Size Medium
Outlet Location Type Tier 2
Outlet Type Grocery Store
Item Type Food
Predict
Type here to search


Outlet Sales Prediction
127.0.0.1:5000/predict
Result Page
The Sales production of FDA15 by OUT046 is
₹ 4348.46 price
Back and Again Predict!!!
Type here to search