CSCI303-B: Data Science Semester Project

Big Mart Sales Prediction Using Machine Learning

Auther: Ping Zhang

Major：Master in Computer Science（No Thesis）

Date: 02/25/2023

Abstract:

BigMart Sales data set includes the data for 1543 products with 11 different features. The data set contains the Train data set and the Test data set. Analyzing correlations between different features, and correlations between features and target to select different combinations of features for analyzing the data set. Then, using the features and the target variable Sales in the Train dataset, build a model for the Test dataset to predict the Sales of the products in the test dataset. At last, making evaluation of the built model. In these procedures, digging out how certain attributes play a key role to determine the sales of a product.

COLORADO SCHOOL OF MINES
EARTH • ENERGY • ENVIRONMENT

Semester Project Template

1. Title Page - title of project, author(s), date, and a brief (300 words or less) abstract.

2. Overview - provide an overview of the dataset, your motivation, and the problem you would like to solve (what questions do you hope to answer?). This section is somewhat speculative – you should be realistic about what is possible (e.g., "I will predict the stock market average for the next 10 years" is probably out of reach), but you shouldn't be afraid to reach for ambitious outcomes. What you write in this section is not a promise that I will hold you to, but a starting point for discussion of next steps.

Using BigMart Sales data set to build a model for predicting the Sales of the products. The data set contains the Train data set and the Test data set, includes the data for 1543 products with 11 different features.The key procedures to the problem are: (1) Data cleaning to get out the outline values of attributes in the data set. (2) Analyzing correlations between different features, and correlations between features and the target variable to select different combinations of features for analyzing the data set. (3) Then, using the features and the target variable Sales in the Train dataset, build a best-fit model for the Test dataset to predict the Sales of the products in the test dataset. (4) Checking for further machine learning techniques such as Clustering to improve the fit if necessary. (5) Making evaluation of the built model.

In the procedures above, digging out how certain attributes play a key role to determine the sales of a product. In these procedures, look at which features are playing a key role in determining the target variable Sales and built a regression model to get the best possible.

3. Related Work (graduate students only) - research and describe what others have done related to your project, and how your work will be unique. This may be something you are basing your work off of, or what others have attempted in the same problem space. Provide three or more references as appropriate.

I choose three research papers have done related to my project. My work will reference the research paper "Big Mart Sales Analysis" (References 1) to implement the basic procedures I mentioned in the overview. Then, I will try to use the core idea the In the research paper "A Two-Level Statistical Model for Big Mart Sales Prediction "(References 3) and combine it in my basic procedures to get a better-fit model for predicting the Sales of the market.

The research paper "Big Mart Sales Analysis "(References 1), applied four algorithms XGBoost, Random Forest, Linear Regression, and Decision Tree. From the results, we can conclude that among all the four algorithms XGBoost has the highest accuracy of 61.14% when distinguished together. Hence, said that XGBoost is the better algorithm for efficient sales analysis.

The research paper "Big Mart Sales Prediction using Machine Learning "(References 2), proposed a Grid Search Optimization (GSO) technique to optimize the parameters and select the best tuning hyperparameters, the further ensemble with Xgboost techniques for forecasting the future sales of a retail company such as Big Mart and we found our model produces the better result.

In the research paper "A Two-Level Statistical Model for Big Mart Sales Prediction "(References 3), the prediction of sales of a product from a particular outlet is performed via a two-level approach that produces better predictive performance compared to any of the popular single model predictive learning algorithms.

The approach is performed on Big Mart Sales data for the year 2013. Data exploration, data transformation, and feature engineering play a vital role in predicting accurate results. The result demonstrated that the two-level statistical approach performed better than a single model approach as the former provided more information that leads to better prediction.

References:

1. Vidya Chitre, Shruti Mahishi, Sharvari Mhatre, Shreya Bhagwat(2022,April).Big Mart Sales Analysis.2022 International Journal of Innovative Technology and Exploring Engineering (IJITEE)
2. Rohit Sav, Pratiksha Shinde, Saurabh Gaikwad (2021, June). Big Mart Sales Prediction using Machine Learning. 2021 International Journal of Research Thoughts (IJCRT).
3. Kumari Punam , Rajendra Pamula , Praphula Kumar Jain (2018, September 28-29). A Two-Level Statistical Model for Big Mart Sales Prediction. 2018 International conference on on Computing, Power and Communication Technologies.

4. **Data acquisition** - describe in some detail the dataset(s) you intend to work with. What data elements exist, how are they structured, what features you hope to extract, etc. This is also the place to explain where the data came from, and any limitations on your use/sharing of the data or your work on the data.

4.1 Describe the Data Structure

I got the BigMart Sales Data from Kaggle (https://www.kaggle.com/datasets/brijbhushannanda1979/bigmart-sales-data). The BigMart Sales Data is sales data from 10 different stores and 1559 different products. Use this to build a prediction model to determine the sales of each of the 10 stores. The data set contains the Train data set and the Test data set, includes the data for 1559 products with 12 columns. There are 11 different features and 1 target.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings # Ignores any warning
warnings.filterwarnings("ignore")

train = pd.read_csv("data/Train.csv")
test = pd.read_csv("data/Test.csv")
```

```
train.head()
```

	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
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.1380
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.2700
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store	732.3800
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052

```
train.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
```

```
train.describe()
```

	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
min	4.555000	0.000000	31.290000	1985.000000	33.290000
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

```
#Check for Unique ID
uniqueId = len(set(train.Item_Identifier))
totalID = train.shape[0]
print("There are " + str(uniqueId) + " Unique IDs for " + str(totalID) + " total entries")
```

```
There are 1559 Unique IDs for 8523 total entries
```

Data Structure Summarize:

- The train data set present 1559 unique ID for 8523 non-null values.
- The train data set has 12 features, 5 are numeric and 7 categorical.
- The target is the column Item_Outlet_Sales.
- Item_Weight and Outlet_Size has Null values.

Corporation of information above , we can identify the products and stores which play a key role in their sales. With using that information, the decision maker can take the correct measures to enhance the sales of products.

We classified all the features in two segment, product and store segment. My personal opinion from this first look at the data, the variables: Item_Visibility, Item_Type , Outlet_Size , Outlet_Location_Type, Outlet_Type will have higher impact on the target variable (Item_Outlet_Sales) .

	Name	Type	Description	segment	Features impact Expecation
1	Item_Identifier	object	product unique ID	product	Low Impact
2	Item_weight	float 64	product weight	product	Medium Impact
3	Item_Fat_Content	object	low fat or not	product	Medium Impact

4	Item_Visibility	float 64	% of total display area in store for this product	product	High Impact
5	Item_Type	object	product category	product	High Impact
6	Item_MRP	float 64	Maximum Retail Price	product	High Impact
7	Outlet_Identifier	object	Store uniqueID	store	Low Impact
8	Outlet_Establishment_Year	int 64	store established year	store	Low Impact
9	Outlet_Size	object	store size	store	Medium Impact
10	Outlet_Location_Type	object	store located Type of city	store	High Impact
11	Outlet_Type	object	Grocery store or some sort of supermarket	store	High Impact
12	Item_Outlet_Sales	float 64	Sales of product in particular store.	product	Target

Limitations:

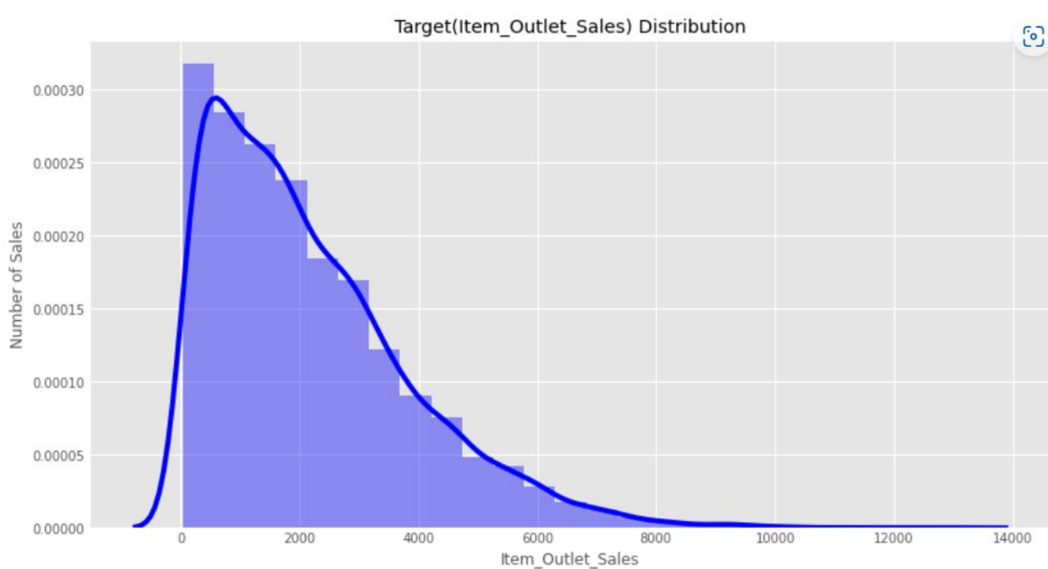
Item_Weight and Outlet_Size has Null values. Every item should have weight >0.0 and every store should have Outlet_Size>0.0. This absence of values has a significant meaning to target or not? How can we Look for these missing values in data processing stage?

4.2 Univariate analysis

4.2.1 Distribution of the target variable: Item_Outlet_Sales

According the distribution histogram of the target variable as below. We can see that higher concentration on lower sales. I make a hypothesis that most of food necessities, such as drinks and dairy, have low sales; but the fresh food, such as fruit and vegetable have high sales.

```
plt.style.use('ggplot')
plt.figure(figsize=(12,7))
sns.distplot(train.Item_Outlet_Sales, bins = 25, color = 'blue')
plt.ticklabel_format(style='plain', axis='x', scilimits=(0,1))
plt.xlabel("Item_Outlet_Sales")
plt.ylabel("Number of Sales")
plt.title("Target(Item_Outlet_Sales) Distribution")
```



4.2.2 Numerical feature analysis

We select all the numeric variables:Item_Weight, Item_Visibility, Item_MRP, Item_Outlet_Sales, Outlet_Establishment_Year.

	Name	Type	Description	segment	Features impact Expecation
1	Item_weight	float 64	product weight	product	Medium Impact
2	Item_Visibility	float 64	% of total display area in store for this product	product	High Impact
3	Item_MRP	float 64	Maximum Retail Price	product	High Impact
4	Outlet_Establishment_Year	int 64	store established year	store	Low Impact
5	Item_Outlet_Sales	float 64	Sales of product in particular store.	product	Target

Then we **analysis the** correlation between Numerical features and Target variable.

```
numeric_features.corr()
corr = numeric_features.corr()
#correlation matrix
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corr, vmax=.8, square=True,annot=True,cmap="crest")
ax.set_title('correlation between Numerical features and Target')
```

Analysis correlation map as below:

- **The Item_Visibility feature has the lowest correlation with Target variable.** At beginning, we made an assumptions the Item_Visibility was expected to have high impact in the Target. However, this is not an behaviour as we expected, so we should investigate.
- **The Item_Visibility feature has a negative correlation with all features.** I am curious that the Item_Visibility feature has a negative correlation with Item_MRP. It means a product has a higher price, if it is less visible in the store.
- **The Item_MRP feature has the most positive correlation with Target variable.** This was quite what we expected.

4.2.3 Non-numeric feature analysis

Bar charts and histograms are good ways of visualization for frequency counts. In this section, using the visualization method to find outliers for each Non-numeric feature. Then, analyze the reason of non-outlier values,and handle the outlier values later in the data cleaning phase.

	Name	Type	Description	segment	Features impact Expecation
1	Item_Identifier	object	product unique ID	product	Low Impact
2	Item_Fat_Content	object	low fat or not	product	Medium Impact
3	Item_Type	object	product category	product	High Impact
	Outlet Identifier	object	Store uniqueID		Low Impact
5	Outlet_Size	object	store size	store	Medium Impact
6	Outlet_Location_Type	object	store located Type of city	store	High Impact
7	Outlet_Type	object	Grocery store or some sort of supermarket	store	High Impact

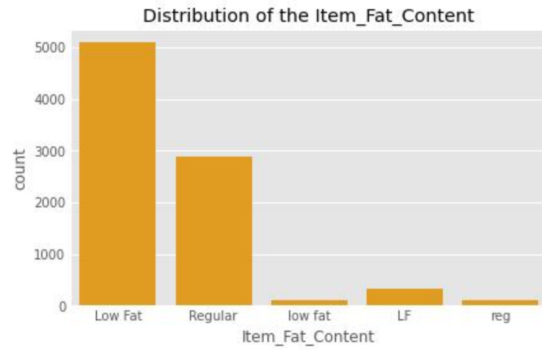
4.2.3.1 Distribution of the Item_Fat_Content

Item_Fat_Content feature has 5 different types: Low Fat, Regular, low fat, LF, reg. Actually, there are only two possible values : Low Fat and Regular. However, we have these two types written in different manners. This must be corrected in data processing phase in later.

Additionally, the number of Low Fat value is bigger than the Regular. This point should be digger to get more information on sales.

```
train.Item_Fat_Content.value_counts()
```

```
Low Fat     5089
Regular     2889
LF           316
reg          117
low fat      112
Name: Item_Fat_Content, dtype: int64
```



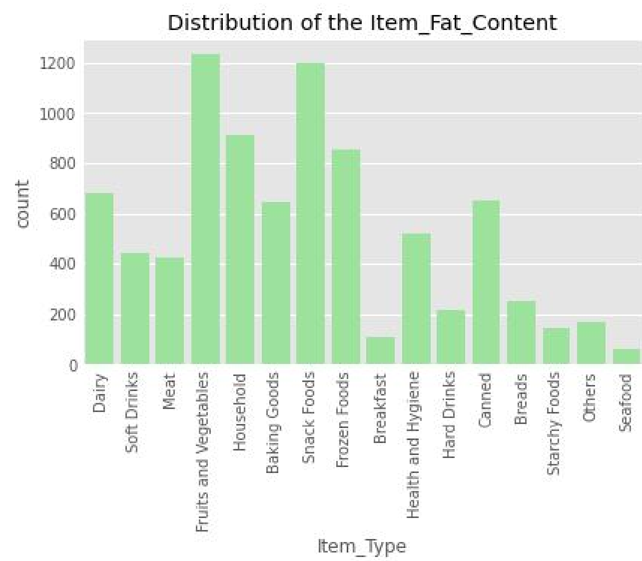

```
sns.countplot(train.Item_Fat_Content,color = 'orange').set(title='Distribution of the Item_Fat_Content')
```

4.2.3.2 Distribution of the Item_Type

Item_Type has 16 different types. The unique values for Item_Type is too much. So, we should reduce the number of unique values. Maybe, we can use 3 unique values: food, drink, other.

```
train.Item_Type.value_counts()
```

```
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
```



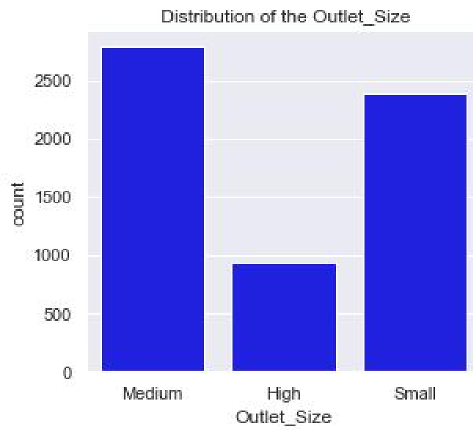

```
sns.countplot(train.Item_Type, color = 'lightgreen').set(title='Distribution of the Item_Fat_Content')
plt.xticks(rotation=90)
```

4.2.3.3 Distribution of the Outlet_Size

Outlet_Size has 3 different types. Most of the stores are Small size or Medium size, and only a small number of stores are high size.We expected the Outlet_Size has medium impact to the sales, because people are more like shopping in stores nearby home, rather than the big store.

```
train.Outlet_Size.value_counts()
```

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



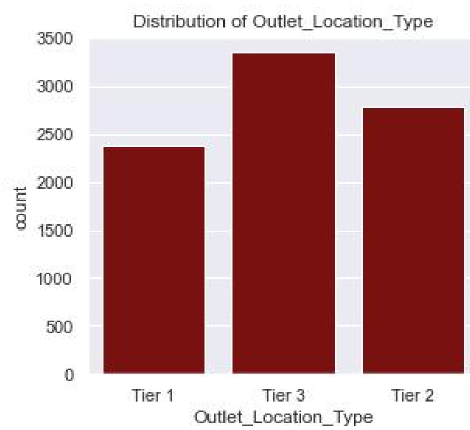

```
sns.countplot(train.Outlet_Size, color = 'blue').set(title=' Distribution of the Outlet_Size')
```

4.2.3.4 Distribution of the Outlet_Location_Type

Outlet_Location_Type has 3 unique types, it means store located at which type of city. All the stores appears to be a same supermarket brand, and most of them located in the small and medium cities, rather than the big cities. I am curious about the how the Outlet_Location_Type impact the sales.

```
train.Outlet_Location_Type.value_counts()
```

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



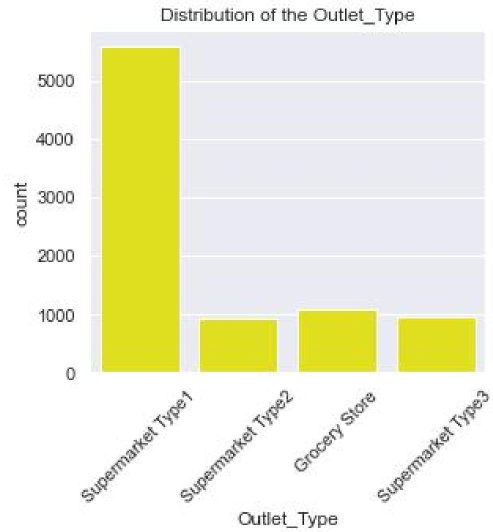

```
sns.countplot(train.Outlet_Location_Type, color = 'darkred').set(title='Distribution of Outlet_Location_Type')
```

4.2.3.5 Distribution of the Outlet_Type

Outlet_Type has 3 unique types. Most of the stores are Supermarket Type1 .As the the numbers of the other 3 Types are more lower than Supermarket Type1, we can merge Supermarket Type2 , Grocery Store and Supermarket Type3 into one type, if they don’t have much impact on the Item_Outlet_Sales.

```
train.Outlet_Type.value_counts()
```

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




```
sns.countplot(train.Outlet_Type, color = 'yellow').set(title='Distribution of the Outlet_Type')
plt.xticks(rotation=45)
```

4.3 Bivariate Analysis

Using bivariate analysis how each feature denoted to the Target, and what a relationship among different features.

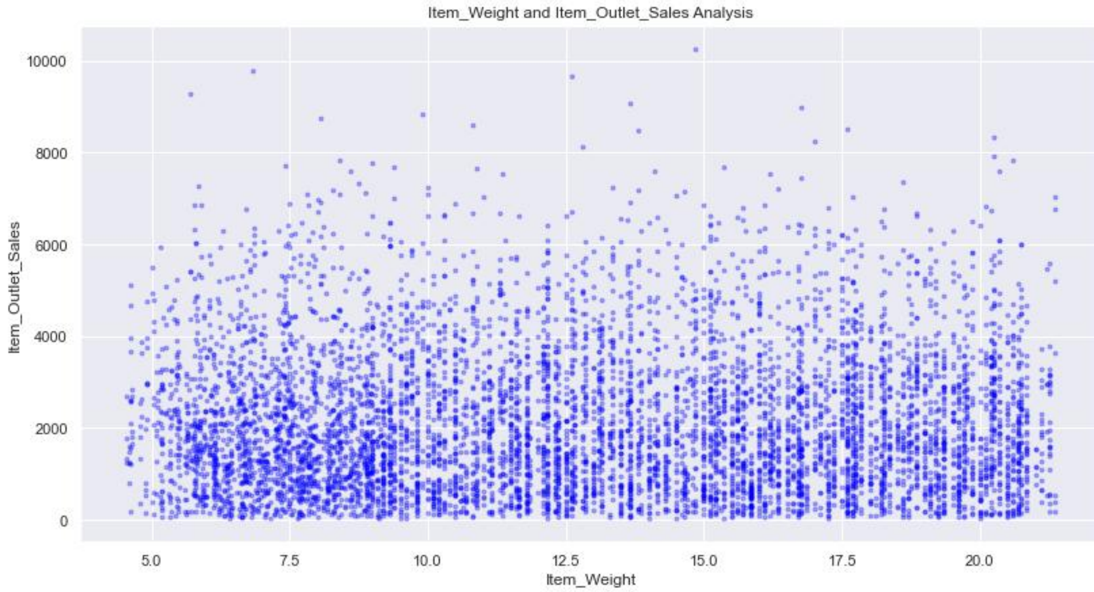
4.3.1 Bivariate Analysis for Numerical variable

	Name	Type	Description	segment	Features impact Expecation
1	Item_weight	float 64	product weight	product	Medium Impact
2	Item_Visibility	float 64	% of total display area in store for this product	product	High Impact
3	Item_MRP	float 64	Maximum Retail Price	product	High Impact
4	Outlet_Establishment_Year	int 64	store established year	store	Low Impact
5	Item_Outlet_Sales	float 64	Sales of product in particular store.	product	Target

4.3.1.1 Item_Weight VS Item_Outlet_Sales

In previously analysis we found that the feature Item_Weight had a low correlation with Target variable. Now we use plot visualization to see the relation between Item_Weight and Item_Outlet_Sales.

```
plt.figure(figsize=(12,7))
plt.xlabel("Item_Weight")
plt.ylabel("Item_Outlet_Sales")
plt.title("Item_Weight and Item_Outlet_Sales Analysis")
plt.plot(train.Item_Weight, train["Item_Outlet_Sales"],'.', alpha = 0.3,  color = 'blue')
```



4.3.1.2 Item_Visibility VS Item_Outlet_Sales

In previously analysis we found that the feature Item_Visibility had a negative correlation with Target variable. We expected Item_Visibility has high impact on Item_Outlet_Sales. Because, ones which are right at entrance will catch the eye of customer first rather than the ones in back. However, most sold products have lower visibility, it is even the negative impact as the image below shown.

Additionally, We find some a bunch of 0 values of Item_Visibility, and have to handle the zero values in later processing phase.

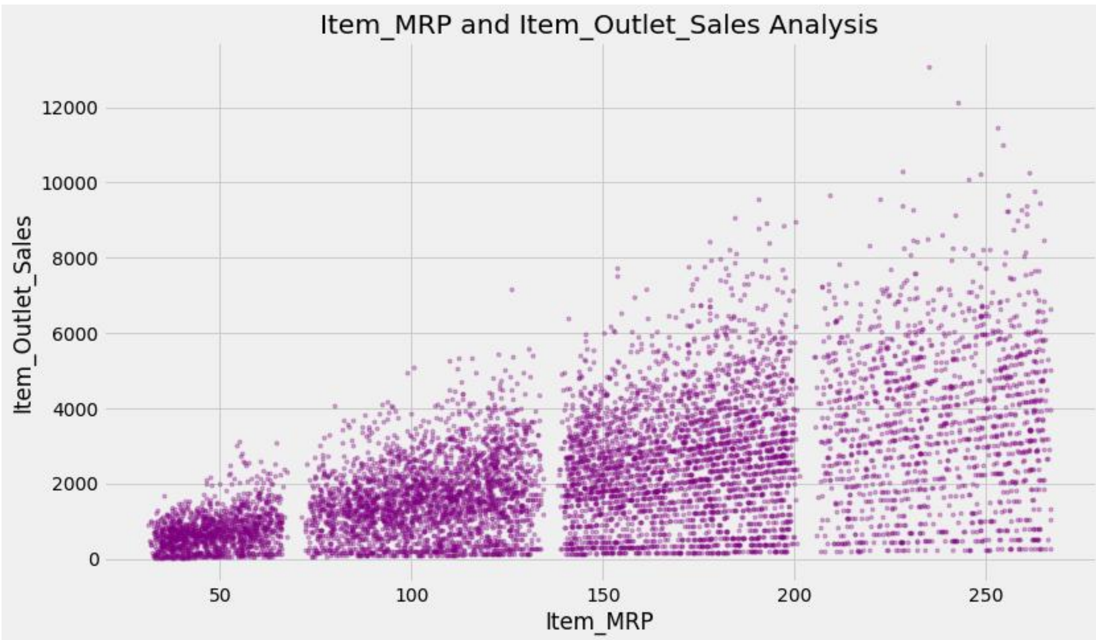
```
plt.figure(figsize=(12,7))
plt.xlabel("Item_Visibility")
plt.ylabel("Item_Outlet_Sales")
plt.title("Item_Visibility and Item_Outlet_Sales Analysis")
plt.plot(train.Item_Visibility, train["Item_Outlet_Sales"],'.', alpha = 0.3,color = 'green')
```

4.3.1.3 Item_MRP VS Item_Outlet_Sales

In previously analysis we found that the feature Item_MRP had positive correlation with Target variable. This is what we expect. I think the reason of this positive correlation is:

- Most of the items were sold When it first went to the stores, so high price match high sales.
- Part of the items were sold after the discount with the seasons reason or the stock reason.
- Small part of items were sold at lowest price when the stores want to clearance sale.

```
plt.figure(figsize=(12,7))
plt.xlabel("Item_MRP")
plt.ylabel("Item_Outlet_Sales")
plt.title("Item_MRP and Item_Outlet_Sales Analysis")
plt.plot(train.Item_MRP, train["Item_Outlet_Sales"],'.', alpha = 0.3, color = 'purple')
```

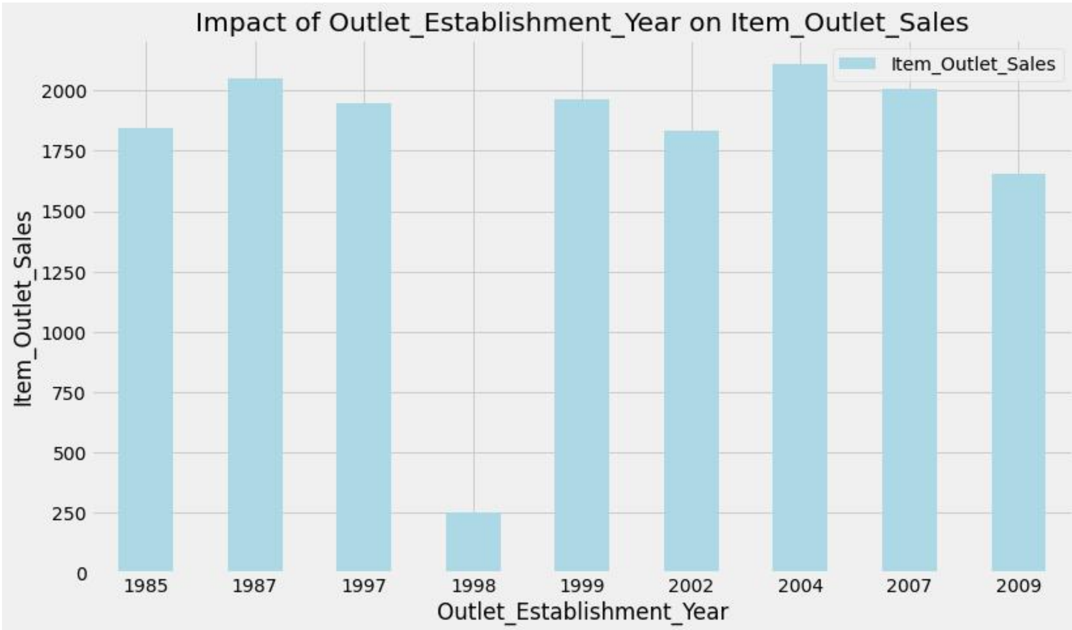


4.3.1.4 Outlet_Establishment_Year VS Item_Outlet_Sales

In previously analysis we found that the feature Item_MRP even had no correlation with Target variable. It also has no correlation with all of other features. This is what we expect.

It's important to note that 1998 has low values far away other values maybe as the reason of few stores opened in that year.

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

Outlet_Establishment_Year_pivot.plot(kind='bar', color='lightblue',figsize=(12,7))
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()
```



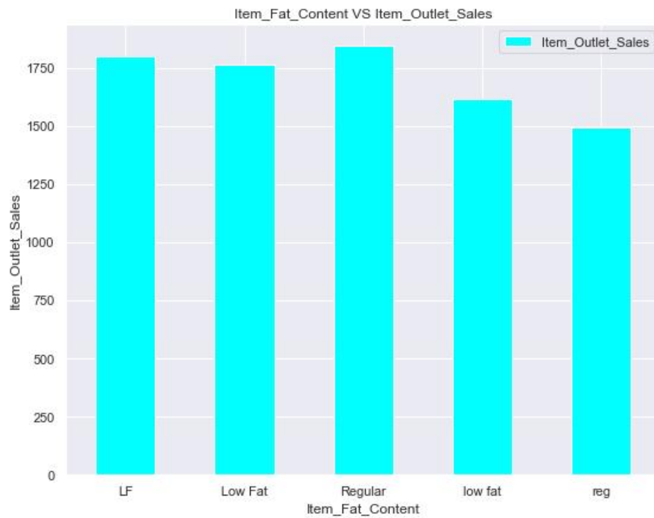
4.3.2 Bivariate Analysis for Non-numerical variable

	Name	Type	Description	segment	Features impact Expecation
1	Item_Identifier	object	product unique ID	product	Low Impact
2	ltem_Fat_Content	object	low fat or not	product	Medium Impact
3	Item_Type	object	product category	product	High Impact
	Outlet Identifier	object	Store uniqueID		Low Impact
5	Outlet_Size	object	store size	store	Medium Impact
6	Outlet_Location_Type	object	store located Type of city	store	High Impact
7	Outlet_Type	object	Grocery store or some sort of supermarket	store	High Impact

4.3.3.1 Item_Fat_Content VS Item_Outlet_Sales

Comparing to the regular products, low Fat products have higher sales values . This is what we expected in previous.

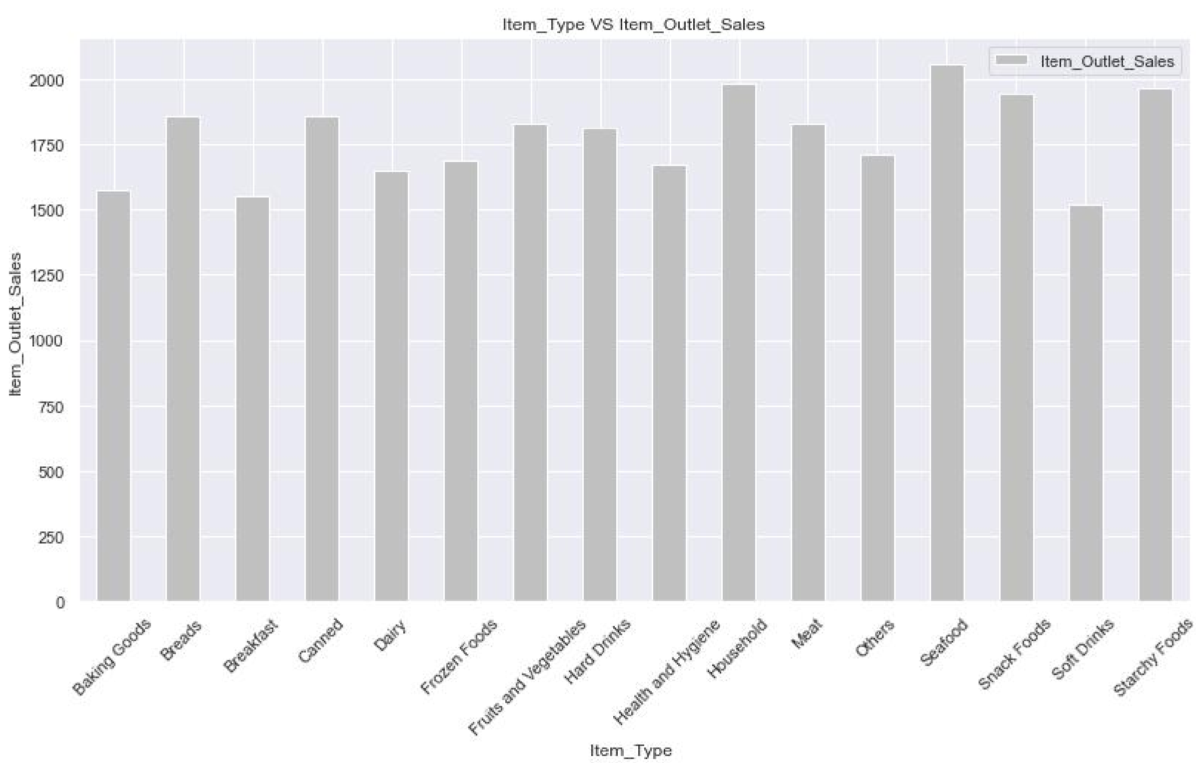
```
Item_Fat_Content_pivot = \
train.pivot_table(index='Item_Fat_Content', values="Item_Outlet_Sales", aggfunc=np.median)

Item_Fat_Content_pivot.plot(kind='bar', color='cyan',figsize=(8,7))
plt.xlabel("Item_Fat_Content")
plt.ylabel("Item_Outlet_Sales")
plt.title("Impact of Item_Fat_Content on Item_Outlet_Sales")
plt.xticks(rotation=0)
plt.show()
```



4.3.3.2 Item_Type VS Item_Outlet_Sales

There is not much difference in the sales of different type. So, the Item_Type is not an high impact on Item_Outlet_Sales. This is not what we expected.

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

Outlet_Identifier_pivot.plot(kind='bar', color='silver',figsize=(12,7))
plt.xlabel("Item_Type ")
plt.ylabel("Item_Outlet_Sales")
plt.title("Item_Type VS Item_Outlet_Sales")
plt.xticks(rotation=45)
plt.show()
```



4.3.3.3 Outlet_Identifier VS Item_Outlet_Sales

There are ten unique Outlet_Identifier, it means the total number of unique stores are 10 in the data set. These 10 stores have 3 unique size: High medium, small. Besides, These 10 stores have 4 unique type. As the picture below.

```
train.pivot_table(values='Outlet_Type', columns='Outlet_Identifier',aggfunc=lambda x:x.mode())
```

Outlet_Identifier	OUT010	OUT013	OUT017	OUT018	OUT019	OUT027	OUT035	OUT045	OUT046	OUT049
Outlet_Type	Grocery Store	Supermarket Type1	Supermarket Type1	Supermarket Type2	Grocery Store	Supermarket Type3	Supermarket Type1	Supermarket Type1	Supermarket Type1	Supermarket Type1

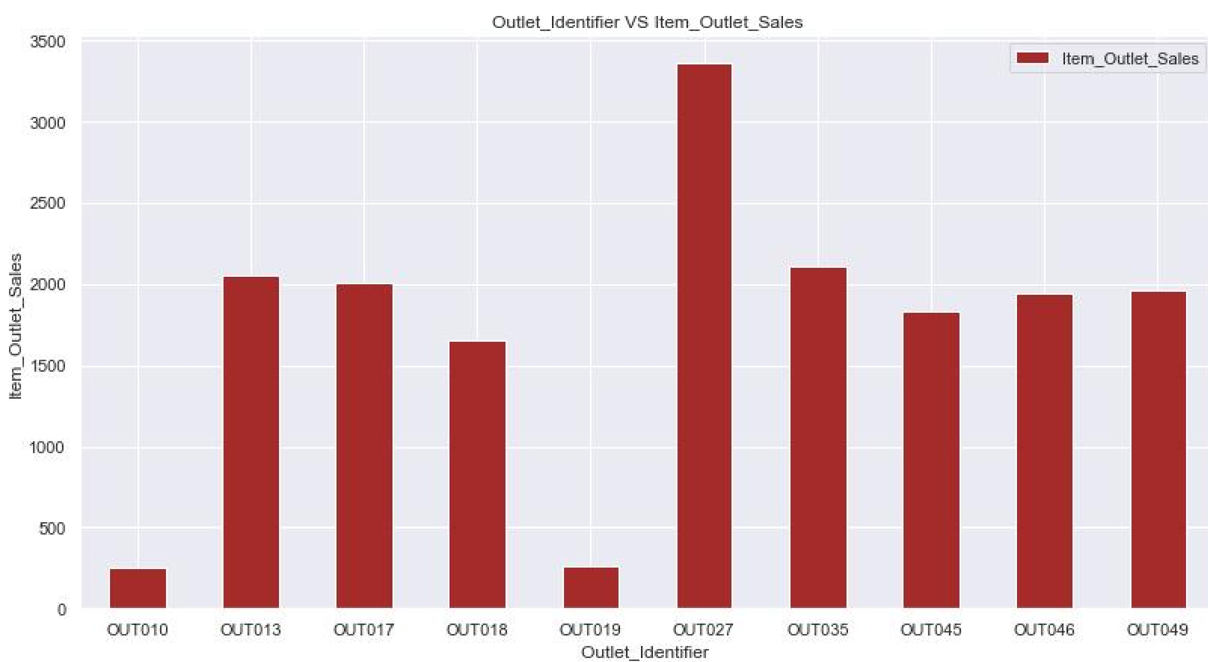
```
train.pivot_table(values='Outlet_Type', columns='Outlet_Size',aggfunc=lambda x:x.mode())
```

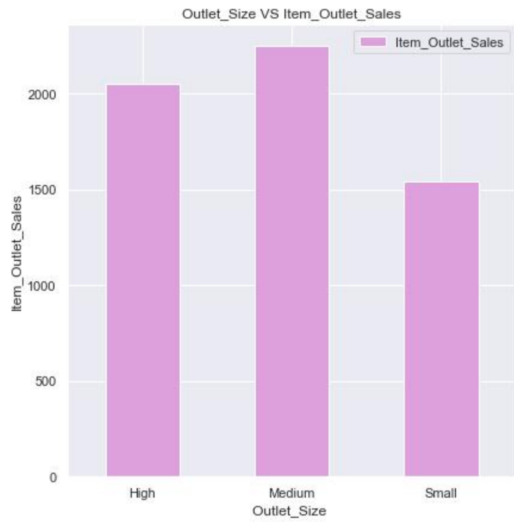
Outlet_Size	High	Medium	Small
Outlet_Type	Supermarket Type1	Supermarket Type3	Supermarket Type1

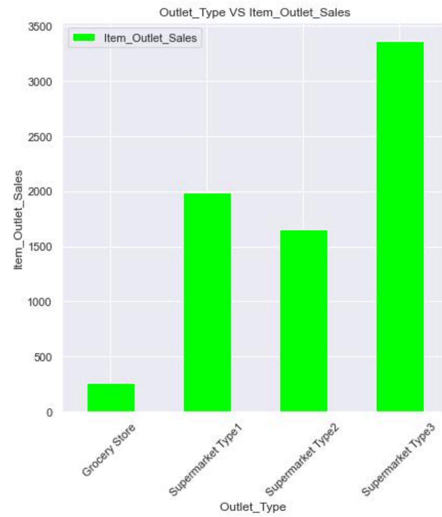
Combined the outlet_Size and outlet_type analysis with the bar chart shown as below:

- The Supermarket Type3, such as OUT027 has the highest sales.
- The groceries, such as OUT010 and OUT019, have the lowest sales.
- All the stores belonged to Supermarket Type 1 have similar sales , and the sales of Supermarket Type1 has no correlation to the outlet_Size.
- The outlet_Size of Medium has the highest sales, as the Supermarket Type3 belongs to the outlet_Size of Medium.

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

Outlet_Identifier_pivot.plot(kind='bar', color='brown',figsize=(12,7))
plt.xlabel("Outlet_Identifier ")
plt.ylabel("Item_Outlet_Sales")
plt.title("Outlet_Identifier VS Item_Outlet_Sales")
plt.xticks(rotation=0)
plt.show()
```







In conclusion:

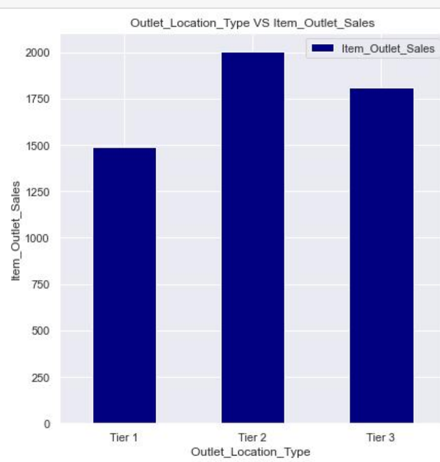
- The Outlet_Identifier has high impact on the Item_Outlet_Sales. This is not what we expected.
- The Outlet_Size has part impact on the Item_Outlet_Sales.This is what we expected.
- The Outlet_Type has high impact on the Item_Outlet_Sales.This is what we expected.

4.3.3.4 Outlet_Location_Type VS Item_Outlet_Sale

In previous part,we mentioned that we curious about the how the Outlet_Location_Type impact the sales. In general opinion, the Tier 1 city has big population, so the stores located in Tier 1 city should have highest sale.

However, stores located in Tier 1 city should have lowest sale in our data. The stores located in Tier 2 city have highest sale. This is not what we expected.

```
Outlet_Location_Type_pivot = \
train.pivot_table(index='Outlet_Location_Type', values="Item_Outlet_Sales", aggfunc=np.median)

Outlet_Location_Type_pivot.plot(kind='bar', color='navy',figsize=(6,7))
plt.xlabel("Outlet_Location_Type ")
plt.ylabel("Item_Outlet_Sales")
plt.title("Outlet_Location_Type VS Item_Outlet_Sales")
plt.xticks(rotation=0)
plt.show()
```



5. **Preprocessing** - this section includes your findings from initial data cleansing, exploratory statistics and visualizations, and additional information on reduced features selected through dimensionality reduction (if appropriate).

Our BigMart data come from Kaggle, and seperated as a train.csv and a test.csv. We decide to use 3 steps to handle the missing values. In this case, we do not have to go through the trouble of repeting twice the same code, for both datasets.

Step 1: Combine train.csv and a test.csv sets into one data set.

Step 2: Find the Nan valus and perform data cleaning and feature engineering in the data set.

Step 3: Divide the data set into train.csv and a test.csv sets again.

5.1 Dimensionality reduction

According our analysis in above section, we can classify all the feature by the impact on product's sale as below.

Features has high impact on product's sale:

- The Item_Fat Content has a high positive correlation as we expected.
- The Item MRP has a high positive correlation as we expected.
- The Outlet_Identifer has a high positive correlation as we expected.
- The Outlet_type has a high positive correlation as we expected.

Features has no high impact on product's sale:

- The Item_Visibility does not have a high positive correlation as we expected, quite the opposite
- The Item_Type does not have a high positive correlation as we expected,quite the opposite.

So, We decide to reduce two features, Item_Visibility and Item_Type.

5.2 Imputing for missing values

The Item_Weight and Outlet_Size seem to present NaN values, and Item_Visibility has some zero values needed to replace.

Now, we start with the step1: Combine train.csv and a test.csv sets into one data set.

```
# creat a new colum to Join Train and Test Dataset into a data set
train['source']='train'
test['source']='test'

data = pd.concat([train,test], ignore_index = True)
data.to_csv("data/data.csv",index=False)
# print(train.shape, test.shape, data.shape)
print("Train shape:")
print(train.shape)
print("Test shape:")
print(test.shape)
print("Joinded data.shape:")
print(data.shape)
```

```
Train shape:
(8523, 13)
Test shape:
(5681, 12)
Joinded data.shape:
(14204, 13)
```

Then we perform step 2: Find the Nan valus and perform data cleaning.

5.2.1 Find the Nan values and Imputing values for Item_weight feature.

Those missing the weight we can retrieve from this table the mean() weight of all products with the same.

```
def impute_Item_weight(cols):
    Weight = cols[0]
    Identifier = cols[1]

    if pd.isnull(Weight):
        return item_avg_weight['Item_Weight'][item_avg_weight.index == Identifier]
    else:
        return Weight
print ('Item_weight Orignal #missing: %d'%sum(data['Item_Weight'].isnull()))
```

```
Item_weight Orignal #missing: 2439
```

```
data['Item_Weight'] = data[['Item_Weight','Item_Identifier']].apply(impute_Item_weight,axis=1).astype(float)
print ('Item_weight Final #missing: %d'%sum(data['Item_Weight'].isnull()))
```

```
Item_weight Final #missing: 0
```

5.2.2 Find the Nan values and Imputing values for Outlet_size.

we will apply the same logic. In this case, instead of using the default code aggfunc = mean() for the pivot_table()we will use the mode

```
def impute_Outlet_size(cols):
    Size = cols[0]
    Type = cols[1]
    if pd.isnull(Size):
        return outlet_size_mode.loc['Outlet_Size'][outlet_size_mode.columns == Type][0]
    else:
        return Size
print ('Outlet_size Orignal #missing: %d'%sum(data['Outlet_Size'].isnull()))
```

```
Outlet_size Orignal #missing: 4016
```

```
data['Outlet_Size'] = data[['Outlet_Size','Outlet_Type']].apply(impute_Outlet_size,axis=1)
print ('Outlet_size Final #missing: %d'%sum(data['Outlet_Size'].isnull()))
```

```
Outlet_size Final #missing: 0
```

5.2.3 Find the zero values and Imputing values for Item_Visibility.

In our data exploration we saw that Item_Visibility had the minimum value 0, which makes no sense since every product must be visible to all clients. Let’s consider it as missing value and impute it with mean visibility of that product.

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

```
def impute_Item_visibility_mean(cols):
    visibility = cols[0]
    item = cols[1]
    if visibility == 0:
        return visibility_item_avg['Item_Visibility'][visibility_item_avg.index == item]
    else:
        return visibility

print ('Item_visibility Original #zeros: %d'%sum(data['Item_Visibility'] == 0))
```

```
Item_visibility Original #zeros: 879
```

```
data['Item_Visibility'] = data[['Item_Visibility','Item_Identifier']].apply(impute_visibility_mean,axis=1).astype(float)
print ('Item_visibility Final #zeros: %d'%sum(data['Item_Visibility'] == 0))
```

```
Item_visibility Final #zeros: 0
```

5.3 Feature Engineering

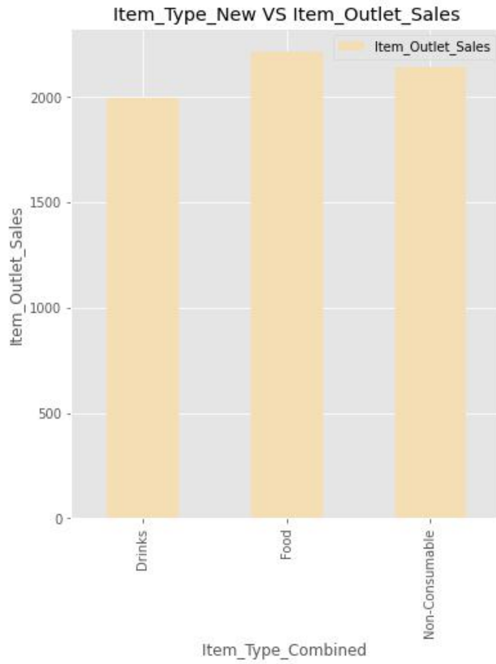
There are 4 Features we should Engineer:

- The Item_Identifer has different groups of letters per each product such as ‘FD’ (Food), ‘DR’(Drinks) and ‘NC’ (Non-Consumable). From this we can create a new variable. SoThe Item_Type we try to create a new feature that does not have 16 unique values.
- The Item_Fat_Content has vale “low fat” written in different manners.
- The Outlet_Establishment_Year values vary from 1985 to 2009 . It must be t must be converted to how old the store is to better see the impact on sales.
- Item Visibility Feature Transformations. Regarding Item_Visibility there are items with the value zero. This does not make lot of sense, since this is indicating those items are not visible on the store.

5.3.1 Create a new category of Item_Type

The Item_Identifier start with either “FD” (Food), “DR” (Drinks) or “NC” (Non-Consumables). So, we can group the Item_Type within these 3 categories.We can create a new variable. SoThe Item_Type we try to create a new feature that does not have 16 unique values.

```
#Get the first two characters of ID:
data['Item_Type_Combined'] = data['Item_Identifier'].apply(lambda x: x[0:2])
#Rename them to more intuitive categories:
data['Item_Type_Combined'] = data['Item_Type_Combined'].map({'FD':'Food',
                                                             'NC':'Non-Consumable',
                                                             'DR':'Drinks'})
# data['Item_Type_Combined'].value_counts()
pivoTable = \
data.pivot_table(index='Item_Type_Combined', values="Item_Outlet_Sales", aggfunc=np.mean)

pivoTable.plot(kind='bar', color='wheat',figsize=(6,7))
plt.xlabel("Item_Type_Combined ")
plt.ylabel("Item_Outlet_Sales")
plt.title("Item_Type_New VS Item_Outlet_Sales")
plt.xticks(rotation=90)
plt.show()
```

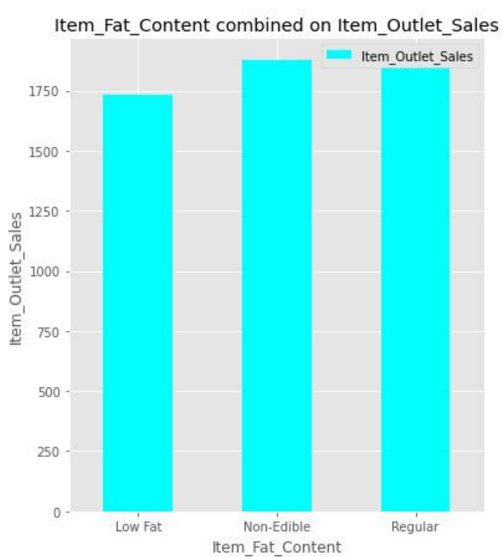


5.3.2 Modify Item_Type values

The Item_Fat_Content has vale “low fat” written in different manners. We combined the different low fat value to the type “low fat”.

But we also saw some non-consumables as well in previous section, and a fat-content should not be specified for them. So we can also create a separate category for such kind of observations.

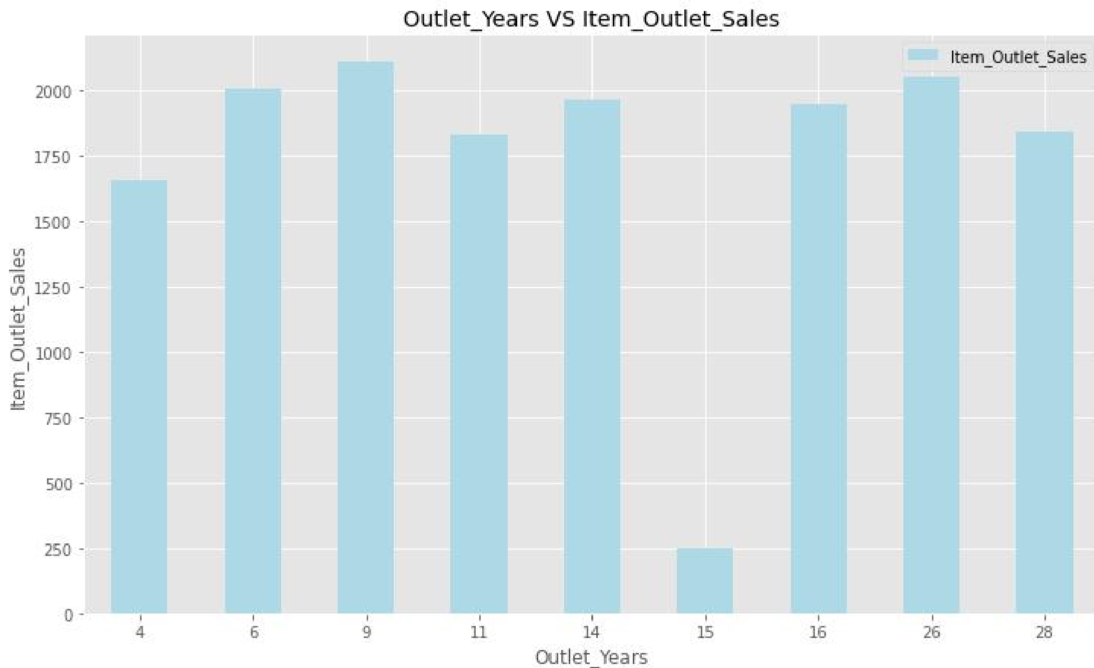
```
#Mark non-consumables as separate category in low_fat:
data.loc[data['Item_Type_Combined']=="Non-Consumable",'Item_Fat_Content'] = "Non-Edible"
data['Item_Fat_Content'].value_counts()

Item_Fat_Content_pivot = \
data.pivot_table(index='Item_Fat_Content', values="Item_Outlet_Sales", aggfunc=np.median)

Item_Fat_Content_pivot.plot(kind='bar', color='cyan',figsize=(6,7))
plt.xlabel("Item_Fat_Content")
plt.ylabel("Item_Outlet_Sales")
plt.title("Item_Fat_Content combined on Item_Outlet_Sales")
plt.xticks(rotation=0)
plt.show()
```



5.3.3 Modify Outlet_Establishment_Year

The Outlet_Establishment_Year values vary from 1985 to 2009 . It must be t must be converted to how old the store is to better see the impact on sales.

```
#Years:
data['Outlet_Years'] = 2013 - data['Outlet_Establishment_Year']
#data['Outlet_Years'].describe()

Outlet_Establishment_Year_pivot = \
data.pivot_table(index='Outlet_Years', values="Item_Outlet_Sales", aggfunc=np.median)

Outlet_Establishment_Year_pivot.plot(kind='bar', color='lightblue',figsize=(12,7))
plt.xlabel("Outlet_Years")
plt.ylabel("Item_Outlet_Sales")
plt.title("Outlet_Years VS Item_Outlet_Sales")
plt.xticks(rotation=0)
plt.show()
```



5.3.4 Transformations for Item Visibility

Item Visibility Feature Transformations. Regarding Item_Visibility there are items with the value zero. This does not make lot of sense, since this is indicating those items are not visible on the store.

```
Creating_MeanRatio = lambda x:x['Item_Visibility']/visibility_item_avg['Item_Visibility'][visibility_item_avg.index == x['Ite

data['Item_Visibility_MeanRatio'] = data.apply(Creating_MeanRatio,axis=1).astype(float)
data['Item_Visibility_MeanRatio'].describe()
```

```
count    14204.000000
mean         1.061884
std          0.235907
min          0.844563
25%          0.925131
50%          0.999070
75%          1.042007
max          3.010094
Name: Item_Visibility_MeanRatio, dtype: float64
```

5.3.5 Creating Dummy variables for Categorical Variables

	Name	Type	Description	segment	Features impact Expecation
1	Item_Identifier	object	product unique ID	product	Low Impact
2	Item_Fat_Content	object	low fat or not	product	Medium Impact
3	Item_Type	object	product category	product	High Impact
	Outlet Identifier	object	Store uniqueID		Low Impact
5	Outlet_Size	object	store size	store	Medium Impact
6	Outlet_Location_Type	object	store located Type of city	store	High Impact
7	Outlet_Type	object	Grocery store or some sort of supermarket	store	High Impact

we need to convert all categories of nominal variables into numeric types, since scikit-learn only accepts numerical variables. Using LabelEncoder() turn all categorical variables into numerical values (Encode labels with value between 0 and n_classes-1).

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

#New variable for outlet
var_Category = ['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','Item_Type_Combined','Outlet_Type','Outlet_Identifier
for i in var_Category:
    data[i] = LabelEncoder().fit_transform(data[i])
```

Then, we can use get_dummies function of Pandas to generate dummy variables from these numerical categorical variables. One-Hot-Coding refers to creating dummy variables, one for each category of a categorical variable. For example, the Item_Fat_Content has 3 categories: LowFat, Regular, Non-Edible. One hot coding will remove this variable and generate 3 new variables. Each will have binary numbers — 0 (if the category is not present) and 1(if category is present).

```
#Dummy Variables:
data = pd.get_dummies(data, columns=['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','Outlet_Type','Item_Type_Combine
data.info()
```

```
Data columns (total 36 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Identifier            14204 non-null  object
 1   Item_Weight                14204 non-null  float64
 2   Item_Visibility            14204 non-null  float64
 3   Item_Type                  14204 non-null  object
 4   Item_MRP                   14204 non-null  float64
 5   Outlet_Establishment_Year  14204 non-null  int64
 6   Item_Outlet_Sales          8523 non-null   float64
 7   source                     14204 non-null  object
 8   Outlet_Years               14204 non-null  int64
 9   Item_Visibility_MeanRatio  14204 non-null  float64
 10  Item_Fat_Content_0         14204 non-null  uint8
 11  Item_Fat_Content_1         14204 non-null  uint8
 12  Item_Fat_Content_2         14204 non-null  uint8
 13  Outlet_Location_Type_0     14204 non-null  uint8
 14  Outlet_Location_Type_1     14204 non-null  uint8
 15  Outlet_Location_Type_2     14204 non-null  uint8
 16  Outlet_Size_0              14204 non-null  uint8
 17  Outlet_Size_1              14204 non-null  uint8
 18  Outlet_Size_2              14204 non-null  uint8
 19  Outlet_Type_0              14204 non-null  uint8
 20  Outlet_Type_1              14204 non-null  uint8
 21  Outlet_Type_2              14204 non-null  uint8
 22  Outlet_Type_3              14204 non-null  uint8
 23  Item_Type_Combined_0       14204 non-null  uint8
 24  Item_Type_Combined_1       14204 non-null  uint8
 25  Item_Type_Combined_2       14204 non-null  uint8
 26  Outlet_Identifier_0        14204 non-null  uint8
 27  Outlet_Identifier_1        14204 non-null  uint8
 28  Outlet_Identifier_2        14204 non-null  uint8
 29  Outlet_Identifier_3        14204 non-null  uint8
 30  Outlet_Identifier_4        14204 non-null  uint8
 31  Outlet_Identifier_5        14204 non-null  uint8
 32  Outlet_Identifier_6        14204 non-null  uint8
 33  Outlet_Identifier_7        14204 non-null  uint8
 34  Outlet_Identifier_8        14204 non-null  uint8
 35  Outlet_Identifier_9        14204 non-null  uint8
dtypes: float64(5), int64(2), object(3), uint8(26)
```

We still have 3 object Categorical Variables: Item_Identifier, Item_type, and source. The Item_Identifier and source features will be removed before we fit the features. Besides, the Item_type and Outlet_Establishment_Year features will be will be removed before we fit the features, because they has been transferred into Item_Combined and Outlet_Years .

5.4 Exporting Data

Final step is Step 3: Divide the data set into train.csv and a test.csv sets again.

```
#Divide into test and train:
train = data.loc[data['source']=="train"]
test = data.loc[data['source']=="test"]

#Export files as modified versions:
train.to_csv("data/train_modified.csv",index=False)
test.to_csv("data/test_modified.csv",index=False)
```

6. <u>Model Selection</u> - identify and describe the algorithm(s) used and why; this can be using the techniques we discuss in class or other machine learning algorithm(s) of your choice.

We use four regression algorithm(s) as below and compare the R-squared scores to select the model.

- Linear Regression
- Ridge Regression
- Lasso Regression
- Elastic Net Regression
- Random Forest Model

To get best parameters of the each model, we use grid search with cross-validation. A common use of cross-validation is for tuning hyper parameters of a model.

Cross-validation: Split into train and test, and train multiple models by sampling the train set. Finally, just test once on the test set. So, we don't need to use train_test_split function for training set.

However, we have to do two steps as below before using the train set and test set.

- Remove 3 object Categorical Variables: Item_Identifier, Outlet_Establishment_Year features, Item_type,source, before using the train set and test set, because they have been transformed.
- Set the Item_Outlet_Sales as the target of the train set.

```
from sklearn.model_selection import train_test_split

train_df = pd.read_csv('data/train_modified.csv')
test_df = pd.read_csv('data/test_modified.csv')
#Drop unnecessary columns:
y_train = train_df.Item_Outlet_Sales
train_df.drop(['Item_Type','Outlet_Establishment_Year','source','Item_Identifier', 'Item_Outlet_Sales'], axis=1,inplace=True)
test_df.drop(['Item_Type','Outlet_Establishment_Year','source','Item_Identifier', 'Item_Outlet_Sales'],axis=1,inplace=True)

X_train = train_df
X_test = test_df
# X_train, X_test, y_train, y_test = train_test_split(input_train, target_train, test_size=0.5)

print("Train shape: ", X_train.shape, y_train.shape)
print("Test shape: ",X_test.shape)
```

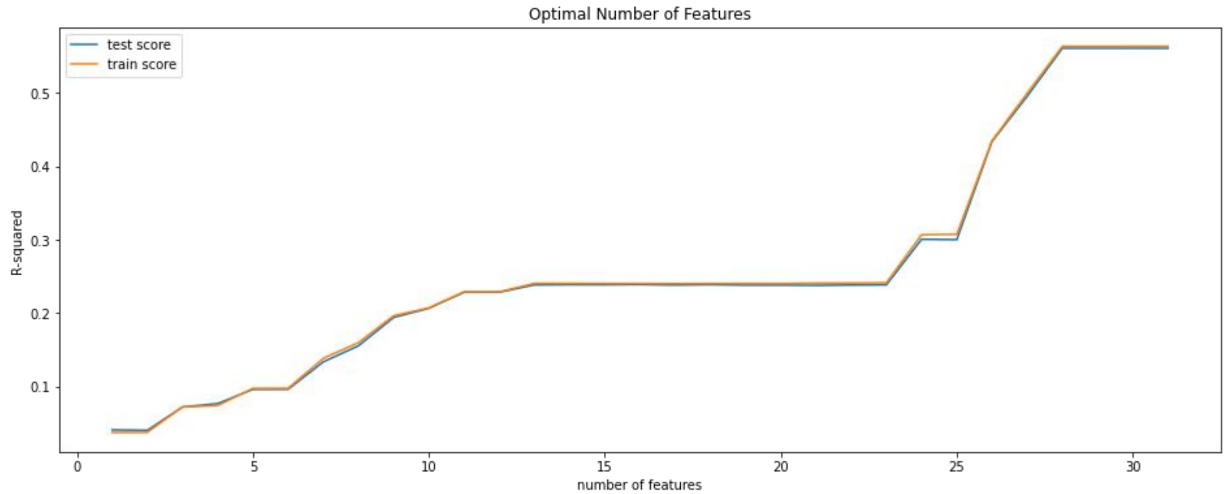
```
Train shape:  (8523, 31) (8523,)
Test shape:  (5681, 31)
```

6.1 Linear Regression

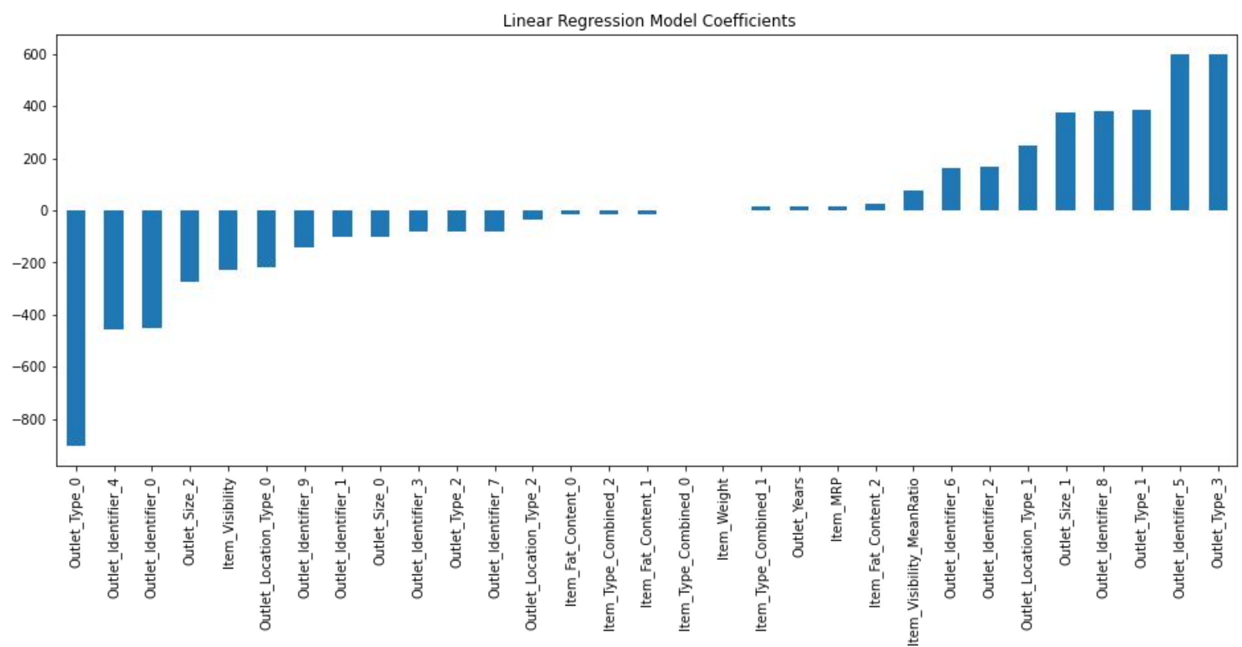
In Linear Regression we have Cost fucntion(ERROR) or Sum of Residual .ie (y-yhat)^2.

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.feature_selection import RFE
from sklearn.model_selection import GridSearchCV

# step-1: create a cross-validation scheme
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)

# step-2: specify range of hyperparameters to tune
hyper_params = [{'n_features_to_select': list(range(1, 32))}]

# step-3: perform grid search
# specify model
lin_Reg = LinearRegression()
lin_Reg.fit(X_train, y_train)
rfe = RFE(lin_Reg)

#call GridSearchCV()
model_cv = GridSearchCV(estimator = rfe,
                        param_grid = hyper_params,
                        scoring= 'r2',
                        cv = folds,
                        verbose = 1,
                        return_train_score=True)

# fit the model
model_cv.fit(X_train, y_train)

# cv results
cv_results = pd.DataFrame(model_cv.cv_results_)
```

```
# plotting cv results
plt.figure(figsize=(16,6))
plt.plot(cv_results["param_n_features_to_select"], cv_results["mean_test_score"])
plt.plot(cv_results["param_n_features_to_select"], cv_results["mean_train_score"])
plt.xlabel('number of features')
plt.ylabel('r-squared')
plt.title("Optimal Number of Features")
plt.legend(['test score', 'train score'], loc='upper left')
plt.show()

# checking best alpha from model_cv
print("The best number of features from model_cv", model_cv.best_params_)

# final model
n_features_optimal = 30

lin_Reg = LinearRegression()
lin_Reg.fit(X_train, y_train)

# plot coefficient
plt.figure(figsize=(16,6))
features =  X_train.columns
coef_lin_Reg = pd.Series(lin_Reg.coef_, features).sort_values()
coef_lin_Reg.plot(kind='bar', title=' Linear Regression Model Coefficients')
plt.show()

rfe = RFE(lin_Reg, n_features_to_select=n_features_optimal)
rfe = rfe.fit(X_train, y_train)

# R-suqared of the model score
r2_lin_Reg= lin_Reg.score(X_train, y_train)
print("The R-squared score of LinearRegression: ",r2_lin_Reg)

# predict sales of X_test
y_pred_linReg = lin_Reg.predict(X_test)
```



According to the result of grid search with cross-validation, the best number of the features for Linear Regression model is thirty. It means using all of the features can get the best predict. However, According to the feature selection plot as above, we can find the Linear Regression model will not have a significant enhance when the number of features > = 27.



Using the best number of the features for Linear Regression model by grid search with cross-validation, we refine our model and get the Linear Regression Model Coefficients as above. The R-squared score of the model is 0.5635.

Conclusion from the Linear Regression Model Coefficients:

- Outlet_Type has the highest impact for the sales, such as out Outlet_Type_0 has highest negative correlation coefficient and Outlet_Type_3 has highest positive correlation coefficient.

- Outlet_Identifier is the second in the influencing factors, such as Outlet_Identifier_4 has highest negative correlation coefficient and Outlet_Identifier_5 has highest positive correlation coefficient in all the ten different Outlet_Identifiers.

- Outlet_size has Obvious effect on sales, such as Outlet_size_2 has highest negative correlation coefficient and Outlet_size_1 has highest positive correlation coefficient in all the 3 different Outlet_size.

- Outlet_Location_Type has Obvious effect on sales, such as Outlet_Location_Type_0 has highest negative correlation coefficient and Outlet_Location_Type_1 has highest positive correlation coefficient in all the 3 different Outlet_Location_Type.

6.2 Ridge Regression Model

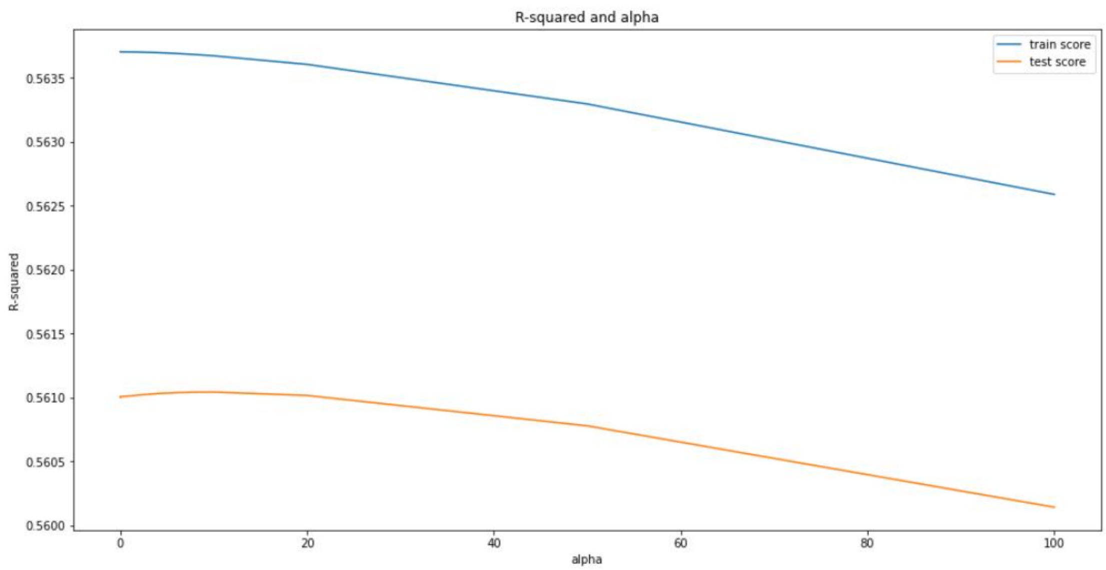
In Ridge Regression we update Cost Function(ERROR).

- Used to Reduce Over fitting Problem which can be understood as if you learn some topic in class say 100%(training set) and in class_test you only wrote say 50% of what you learned(test_set), now i can tell you you are Over fitting.
- It basically penalize features having higher slope, as a result Error reduces.
- for say more than one feature values of slope will be added and remaining formula remains the same i.e (slope_1^2 + slope_2^2).

```
from sklearn.linear_model import Ridge

params = {'alpha': [0.001, 0.005, 0.01, 0.1,0.5, 1.0, 2.0, 4.0, 6.0, 8.0, 10.0, 20, 50, 100]}

#initialising Ridge() function
ridge = Ridge()
# defining cross validation folds as 5
folds = 5

# Defining GridSearchCV
model_cv = GridSearchCV(estimator=ridge,
                        param_grid=params,
                        scoring='r2',
                        cv=folds,
                        return_train_score=True,
                        verbose=1)

# fiting GridSearchCV() with X_train and y_train
model_cv.fit(X_train,y_train)

 # Saving GridSearchCV results into a dataframe
cv_results = pd.DataFrame(model_cv.cv_results_)

# # filter cv_results with all param_alpha less than or equal to 200
# cv_results = cv_results[cv_results['param_alpha']<=200]

# cv_results head
cv_results.head()

# changing datatype of 'param_alpha' into int
cv_results['param_alpha'] = cv_results['param_alpha'].astype('int32')

# plotting
plt.figure(figsize=(16,8))
plt.plot(cv_results['param_alpha'],cv_results['mean_train_score'])
plt.plot(cv_results['param_alpha'],cv_results['mean_test_score'])
plt.title('R-squared and alpha')
plt.xlabel('alpha')
plt.ylabel('R-squared')
plt.legend(['train score','test score'],loc='upper right')
plt.show()

# checking best alpha from model_cv
print("The best alpha from model_cv", model_cv.best_params_)


#final model
alpha = 10
ridge =Ridge(alpha=alpha)
```

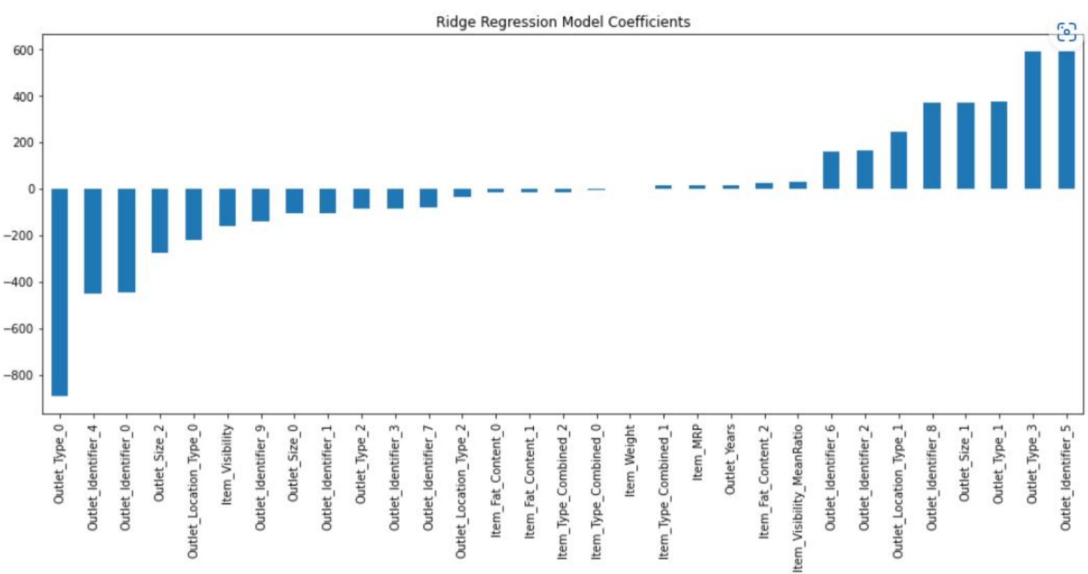
```
#fitting model
ridge.fit(X_train,y_train)

# plot coefficient
plt.figure(figsize=(16,6))
features =  X_train.columns
coef_ridge = pd.Series(ridge.coef_, features).sort_values()
coef_ridge.plot(kind='bar', title='Ridge Regression Model Coefficients')
plt.show()

# R-suqared of the model score
r2_ridge = ridge.score(X_train, y_train)
print("The R-squared score of Ridge Regression: ",r2_ridge)

# predict sales of X_test
y_pred_ridge = ridge.predict(X_test)
```

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
```




```
The best alpha from model_cv {'alpha': 10.0}
```

Observe that test and train scores start to become parallel to each other after apha crosses 10. It means apha =10 is the best parameter for Ridge Regression.




```
The R-squared score of Ridge Regression:  0.5634855875659633
```

Using the best parameter of the apha for Ridge Regression model by grid search with cross-validation, we refine our model and get the Ridge Regression Model Coefficients as above. The R-squared score of the model is 0.5634.

The refine Ridge Regression Model Coefficients is very closed to the Linear Regression Model. So, the Conclusion of Ridge Regression Model is same with the Linear Regression Model.

6. 3 Lasso Regression

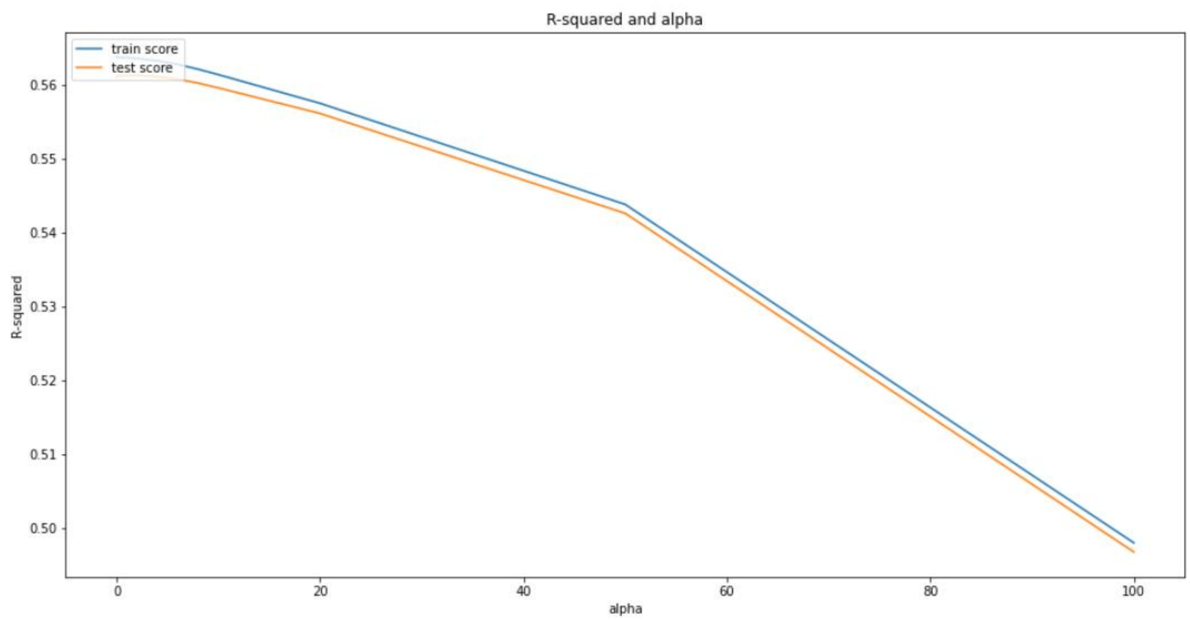
In Lasso Regression we update Cost Function(ERROR) or Sum of Residual .ie (y-yhat)^2 + lambda * |slope| => magnitude of slope not square fo slope

- it helps in reducing Over fitting fore sure.
- Also helps in Feature Selection.
- Feature having less slope value will be removed, that means those removed feature were not important for predicting best fit line.
- here slope moves toward 0 constantly and at some point becomes 0, so in the process feature are selected, in case of Ridge slope only shrinks but never become 0.
- Good choice when we have a large number of features but expect only a few to be important.

```
from sklearn.linear_model import Lasso

# Initialising Lasso()
lasso = Lasso()

#usig same attributes used for Ridge tuning except estimator here would be lasso
model_cv = GridSearchCV(estimator=lasso,
                        param_grid=params,
                        scoring='r2',
                        cv=folds,
                        return_train_score=True,
                        verbose=1)
#fiting model_cv
model_cv.fit(X_train,y_train)

# Saving model_cv results into a dataframe
cv_results = pd.DataFrame(model_cv.cv_results_)

# cv_results head
cv_results.head()

# changing param_alpha datatype to float
cv_results['param_alpha'] = cv_results['param_alpha'].astype('int32')

# plotting
plt.figure(figsize=(16,8))
plt.plot(cv_results['param_alpha'],cv_results['mean_train_score'])
plt.plot(cv_results['param_alpha'],cv_results['mean_test_score'])

plt.title('R-squared and alpha')
plt.xlabel('alpha')
plt.ylabel('R-squared')
plt.legend(['train score','test score'],loc='upper left')
plt.show()

# Checking best  alpha from model_cv
print("The best alpha from model_cv", model_cv.best_params_)

# final model
alpha =2.0
lasso =Lasso(alpha=alpha)

# fiting lasso
lasso.fit(X_train,y_train)

# plot coefficient
plt.figure(figsize=(16,6))
features =  X_train.columns
coef_ridge = pd.Series(lasso.coef_, features).sort_values()
coef_ridge.plot(kind='bar', title='Lasso Regression Model Coefficients')
plt.show()
```

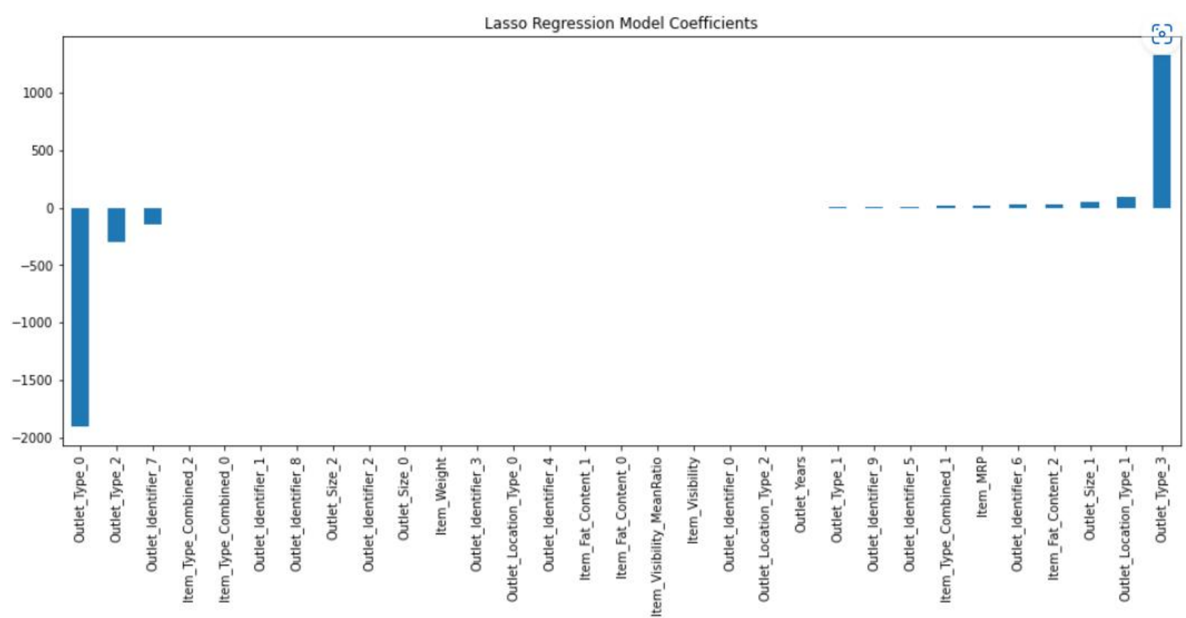
```python
# R-suqared of the model score
r2_lasso = lasso.score(X_train, y_train)
print("The R-squared score of Lasso Regression: ",r2_lasso)

# predict sales of X_test
y_pred_lasso = lasso.predict(X_test)
```

```
Fitting 5 folds for each of 14 candidates, totalling 70 fits
```




```
The best alpha from model_cv {'alpha': 2.0}
```

Observe that test and train scores start to become parallel to each other after apha crosses 2.0. It means apha =2.0 is the best parameter for Lasso Regression.




```
The R-squared score of Lasso Regression:  0.5633322802282524
```

Using the best parameter of the apha for Lasso Regression model by grid search with cross-validation, we refine our model and get the Lasso Regression Model Coefficients as above. The R-squared score of the model is 0.5633.

The features rank of impact for the sales, according to the Lasso Regression Model Coefficients:
1) Outlet_Type .
2) Outlet_Identifier .
3) Outlet_Location_Type_1.
4) Outlet_size_1.
5) Item_Fat_Content.
6) Item_MRP.
7) Outlet_Type_Combined_1.

The other Features having less slope value will be removed, that means those features were not important for predicting best fit line.

6.4 ElasticNet Regression

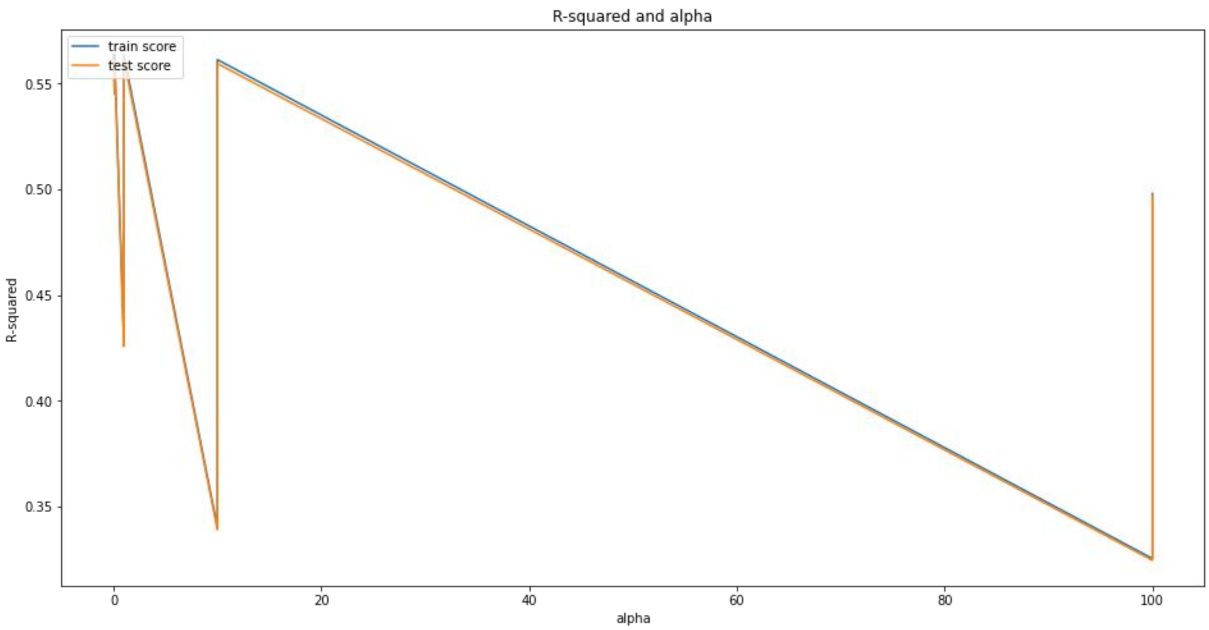
Elastic-Net Regression is a linear regression model that combines penalties of Lasso and Ridge

- l1_ratio parameter is used to control combination of L1 and L2 regularization
- l1_ratio = 0 we have L2 regularization (Ridge)
- l1_ratio = 1 we have L1 regularization (Lasso)
- Values between 0 and 1 give a combination of both L1 and L2 regularization

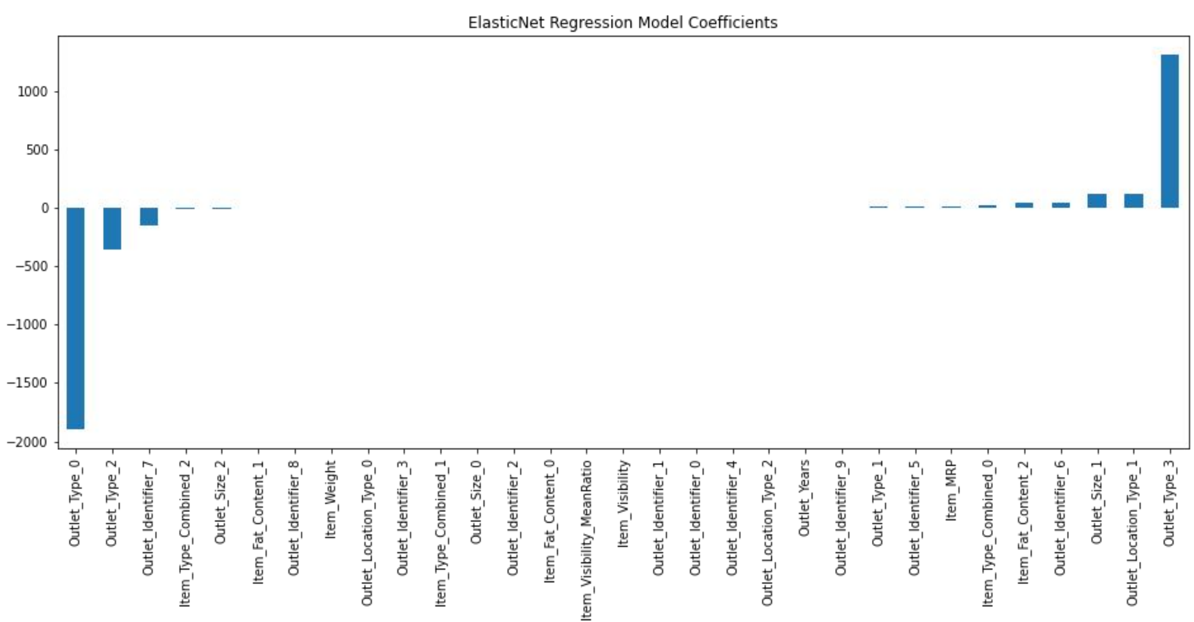
```
from sklearn.linear_model import ElasticNet

# Initialising ElasticNet()
elasticnet = ElasticNet()

params= {"alpha": [0.0001, 0.001, 0.01, 0.1, 1, 10, 100],"l1_ratio": [0.0, 0.2,0.4, 0.5,0.6, 0.8, 1.0]}

#using same attributes used for Ridge tuning except estimator here would be ElasticNet
model_cv = GridSearchCV(estimator=elasticnet,
                        param_grid=params,
                        scoring='r2',
                        cv=folds,
                        return_train_score=True,
                        verbose=1)
#fitingmodel_cv
model_cv.fit(X_train,y_train)

# Saving model_cv results into a dataframe
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results.head()

# Checking best alpha from model_cv
print("The best alpha and l1_ratio from model_cv", model_cv.best_params_)

# final model
alpha =1.0
l1_ratio =1.0
elasticnet = ElasticNet(alpha=alpha, l1_ratio =l1_ratio)
X_train_elasticnet, X_test_elasticnet, y_train_elasticnet, y_test_elasticnet = train_test_split(X_train, y_train, test_size=0

# fiting elastic net
elasticnet.fit(X_train_elasticnet,y_train_elasticnet)

# plot coefficient
plt.figure(figsize=(16,6))
features =  X_train.columns
coef_elasticnet = pd.Series(elasticnet.coef_, features).sort_values()
coef_elasticnet.plot(kind='bar', title='ElasticNet Regression Model Coefficients')
plt.show()
```

```
# predict sales of X_test_elasticnet
y_pred_elasticnet = elasticnet.predict(X_test_elasticnet)

# Eveluate the model
# R-suqared
r2_elasticnet = elasticnet.score(X_train_elasticnet, y_train_elasticnet)
print("The R-squared score of ElasticNet Regression: ",r2_elasticnet)
# RMSE
RMSE_elasticnet = metrics.mean_squared_error(y_test_elasticnet, y_pred_elasticnet)
print("The RMSE score of ElasticNet Regression: ", RMSE_elasticnet)
# MAE
MAE_elasticnet = metrics.mean_absolute_error(y_test_elasticnet, y_pred_elasticnet)
print("The MAE score of ElasticNet Regression: ", MAE_elasticnet)
```

```
Fitting 5 folds for each of 49 candidates, totalling 245 fits
The best alpha and l1_ratio from model_cv {'alpha': 1.0, 'l1_ratio': 1.0}
```



According to the result of grid search with cross-validation, the best for Elastic-Net Regression model is alpha = 1.0 and l1_ratio = 1.0.



The features rank of impact for the sales, according to the Elastic-Net Regression Model Coefficients:

1) Outlet_Type .
2) Outlet_Identifier .
3) Outlet_Location_Type_1.
4) Outlet_size_1.
5) Item_Fat_Content.
6) Item_MRP.
7) Outlet_Type_Combined_1.

```
The R-squared score of ElasticNet Regression:  0.5642650658456039
The RMSE score of ElasticNet Regression:  1263974.0581925448
The MAE score of ElasticNet Regression:  825.5552808167571
```

Using the best parameter of the apha for Elastic-Net Regression model by grid search with cross-validation, we refine our model and get the Elastic-Net Regression Model Coefficients as above. The R-squared score of the model is 0.5642.

The refine Elastic-Net Regression Model Coefficients is very closed to the Linear Regression Model. So, the Conclusion of Elastic-Net Regression Model is same with the Linear Regression Model.

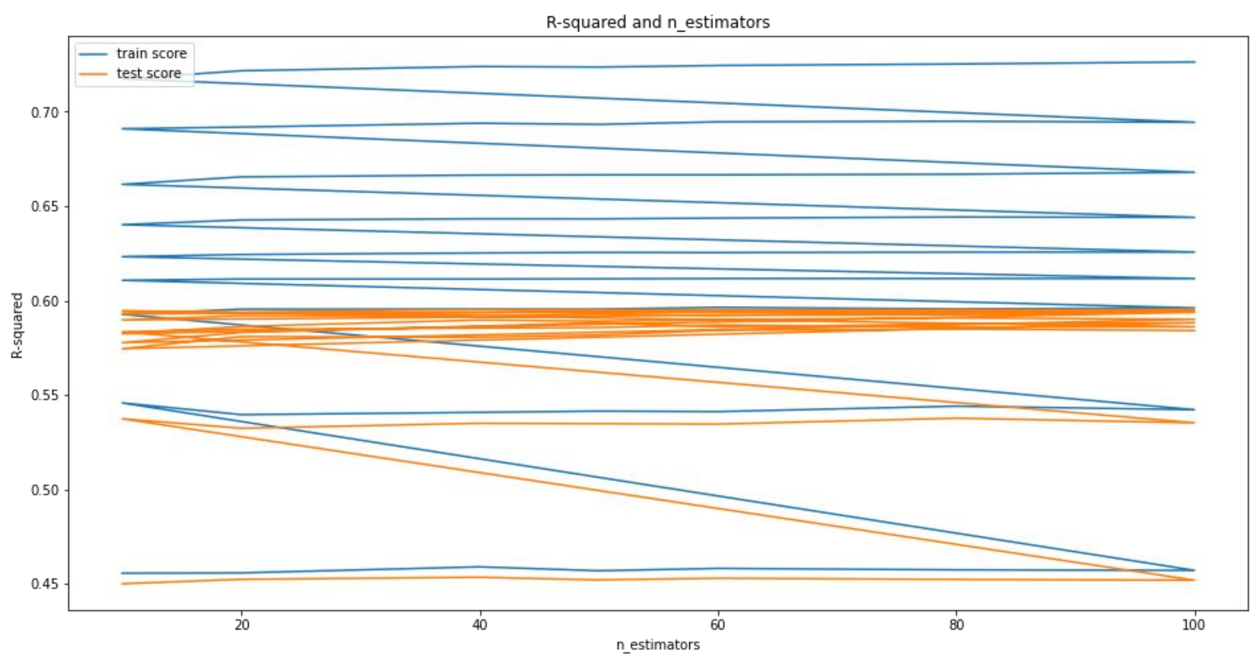
6. 5 Random Forest Regression Model

Random Forest is a Supervised learning algorithm that is based on the ensemble learning method and many Decision Trees. Random Forest is a Bagging technique, so all calculations are run in parallel and there is no interaction between the Decision Trees when building them. We tune the parameters of n_estimator and max_depth.

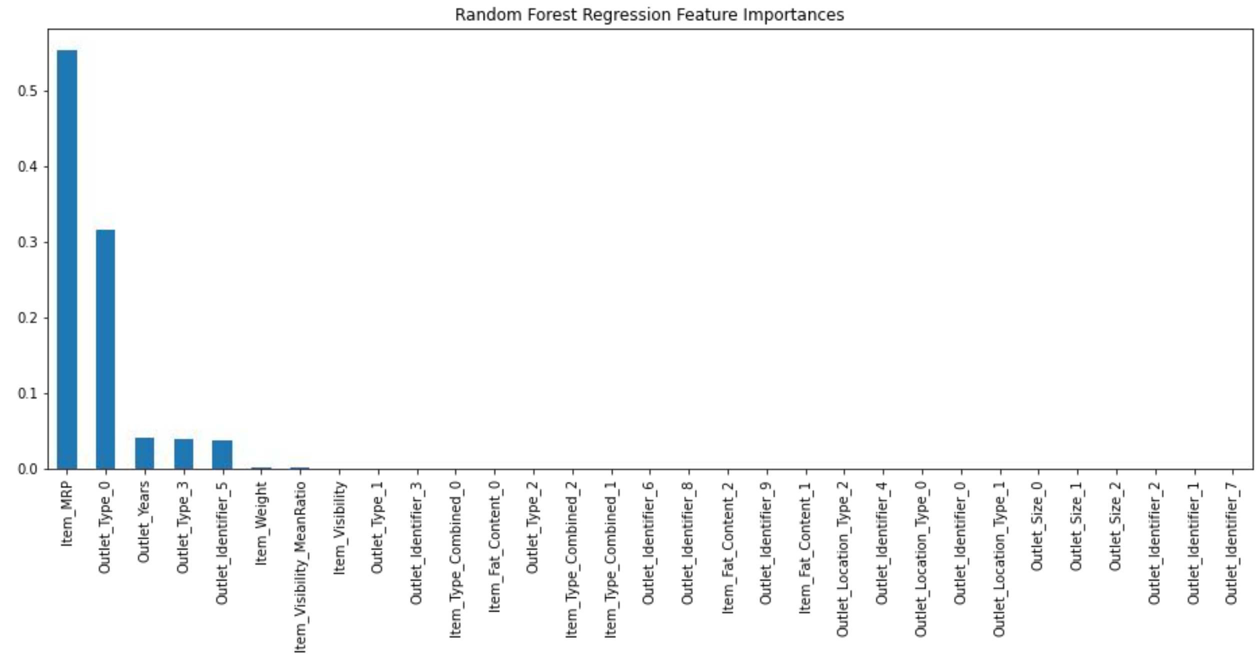
```
from sklearn.ensemble import RandomForestRegressor

# Initialising RandomForestRegressor()
randomForest = RandomForestRegressor()

#using same attributes used for randomForest tuning except estimator here would be RandomForestRegressor
params = {'n_estimators':[10,20,40,50,60,80,100], 'max_depth':[2,3,4,5,6,7,8,9,10]}
model_cv = GridSearchCV(estimator=randomForest,
                        param_grid=params,
                        scoring='r2',
                        cv=folds,
                        return_train_score=True,
                        verbose=1)

#fitingmodel_cv
model_cv.fit(X_train,y_train)

# Saving model_cv results into a dataframe
cv_results = pd.DataFrame(model_cv.cv_results_)
cv_results.head()

# change n_estimators datatype to int
cv_results['param_n_estimators'] = cv_results['param_n_estimators'].astype('int32')
cv_results['param_max_depth'] = cv_results['param_max_depth'].astype('int32')

# plotting
plt.figure(figsize=(16,8))
plt.plot(cv_results['param_n_estimators'],cv_results['mean_train_score'])
plt.plot(cv_results['param_n_estimators'],cv_results['mean_test_score'])
plt.title('R-squared and n_estimators')
plt.xlabel('n_estimators')
plt.ylabel('R-squared')
plt.legend(['train score','test score'],loc='upper left')
plt.show()
```

```
# Checking best alpha from model_cv
print("The best params_ from model_cv", model_cv.best_params_)

# final model
n_estimators=40
max_depth = 5
randomForest = RandomForestRegressor(n_estimators = n_estimators, max_depth = max_depth)
X_train_randomForest, X_test_randomForest, y_train_randomForest, y_test_randomForest = train_test_split(X_train, y_train, tes

# fiting randomForest
randomForest.fit(X_train_randomForest,y_train_randomForest)

# plot coefficient
plt.figure(figsize=(16,6))
features =  X_train.columns
coef_randomForest = pd.Series(randomForest.feature_importances_, features).sort_values(ascending=False)
coef_randomForest.plot(kind='bar', title='Random Forest Regression Feature Importances')
plt.show()

# predict sales of X_test_elasticnet
y_pred_randomForest = randomForest.predict(X_test_randomForest)

# Eveluate the model
# R-suqared
r2_randomForest = randomForest.score(X_train_randomForest, y_train_randomForest)
print("The R-squared score of RandomForest Regression: ",r2_randomForest)
# RMSE
RMSE_randomForest = metrics.mean_squared_error(y_test_randomForest, y_pred_randomForest)
print("The RMSE score of Random Forest Regression: ", RMSE_randomForest)
# MAE
MAE_randomForest = metrics.mean_absolute_error(y_test_randomForest, y_pred_randomForest)
print("The MAE score of RandomForest Regression: ", MAE_randomForest)
```

```
The best params_ from model_cv {'max_depth': 5, 'n_estimators': 40}
```



According to the result of grid search with cross-validation, the best for Random Forest Regression model is n_estimators = 40 and max_depth = 5.



The features rank of impact for the sales is very different from the previous regression model, according to the Random Forest Regression Model Coefficients:

(1) Item_MRP
(2) Outlet_Type
(3) Outlet_Years
(4) Outlet_Identifier
(5) Item_weight
(6) Item_Visibility_MeanRatio

```
The R-squared score of RandomForest Regression:  0.6084672521291847
The RMSE score of Random Forest Regression:  1090225.401678379
The MAE score of RandomForest Regression:  730.1572895385415
```

Using the best parameter of the n_estimators=40, and max_depth = 5 for Random Forest Regression model by grid search with cross-validation, we refine our model and get the Random Forest Regression Model Coefficients as above. The R-squared score of the model is 0.6084.

7 Results and Evaluation - report the conclusions and results of your analysis including validation metrics, techniques, and visualizations.

7.1 Analysis validation metrics of different techniques

We use R-Squared score, RMSE score and MAE score to analysis the five different regression techniques in section 6.

Note:

- Squared score higher is better in range[0,1] .
- RMSE lower is better.
- MAE lower is better.

```python
print("\nR-squared score of models:")
print("\nThe R-squared score of Linear Regression:", r2_linReg)
print("\nThe R-squared score of Ridge Regression:", r2_ridge)
print("\nThe R-squared score of Lasso Regression:", r2_lasso)
print("\nThe R-squared score of Elastic Net Regression:", r2_elasticnet)
print("\nThe R-squared score of RandomForest Net Regression:", r2_randomForest)

print("\nRMSE score of models:")
print("\nThe RMSE score of Linear Regression:", RMSE_linReg)
print("\nThe RMSE score of Ridge Regression:", RMSE_ridge)
print("\nThe RMSE score of Lasso Regression:", RMSE_lasso)
print("\nThe RMSE score of Elastic Net Regression:", RMSE_elasticnet)
print("\nThe RMSE score of Random Forest Regression:", RMSE_randomForest)


print("\nMAE score of models:")
print("\nThe MAE score of Linear Regression:", MAE_linReg)
print("\nThe MAE score of Ridge Regression:", MAE_ridge)
print("\nThe MAE score of Lasso Regression:", MAE_lasso)
print("\nThe MAE score of Elastic Net Regression:", MAE_elasticnet)
print("\nThe MAE score of Random Forest Regression:", MAE_randomForest)
```

```
R-squared score of models:

The R-squared score of Linear Regression: 0.5612789056697769

The R-squared score of Ridge Regression: 0.5631435022061182

The R-squared score of Lasso Regression: 0.5620884392425506

The R-squared score of Elastic Net Regression: 0.5642650658456039

The R-squared score of RandomForest Net Regression: 0.6084672521291847

RMSE score of models:

The RMSE score of Linear Regression: 1287910.3735835054

The RMSE score of Ridge Regression: 1313577.6261705693

The RMSE score of Lasso Regression: 1271537.9167466753

The RMSE score of Elastic Net Regression: 1263974.0581925448

The RMSE score of Random Forest Regression: 1090225.401678379

MAE score of models:

The MAE score of Linear Regression: 846.2458504993059

The MAE score of Ridge Regression: 836.4531788124508

The MAE score of Lasso Regression: 846.1259455488473

The MAE score of Elastic Net Regression: 825.5552808167571

The MAE score of Random Forest Regression: 730.1572895385415
```

Comparing the R-Squared score, RMSE score and MAE score from the five model with grid search cross validation, we can find the result:

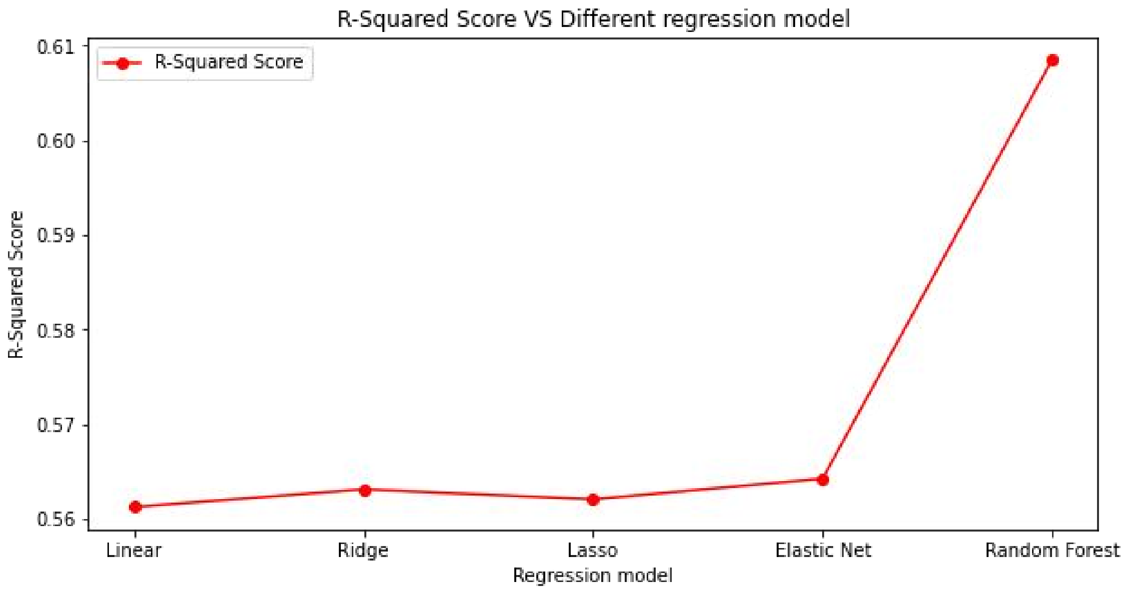
- Random Forest Regression has the best performance on the train set.
- The performance of four models, Linear Regression, Ridge Regression, Lasso Regression, and Elastic Net Regression have a very small difference on the train set.

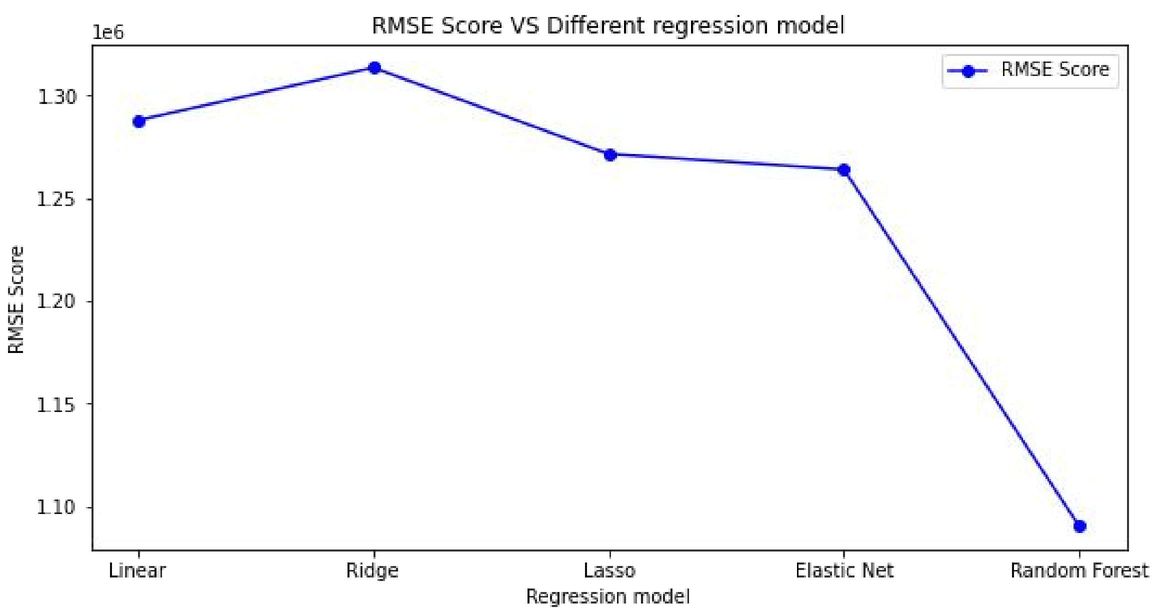
So, In this project, we selected Random Forest Regression as our model to predict the test set, because the amount of data is small, and it will not take too much time to find the best parameters with grid search and cross validation to refine the model to do better predict.

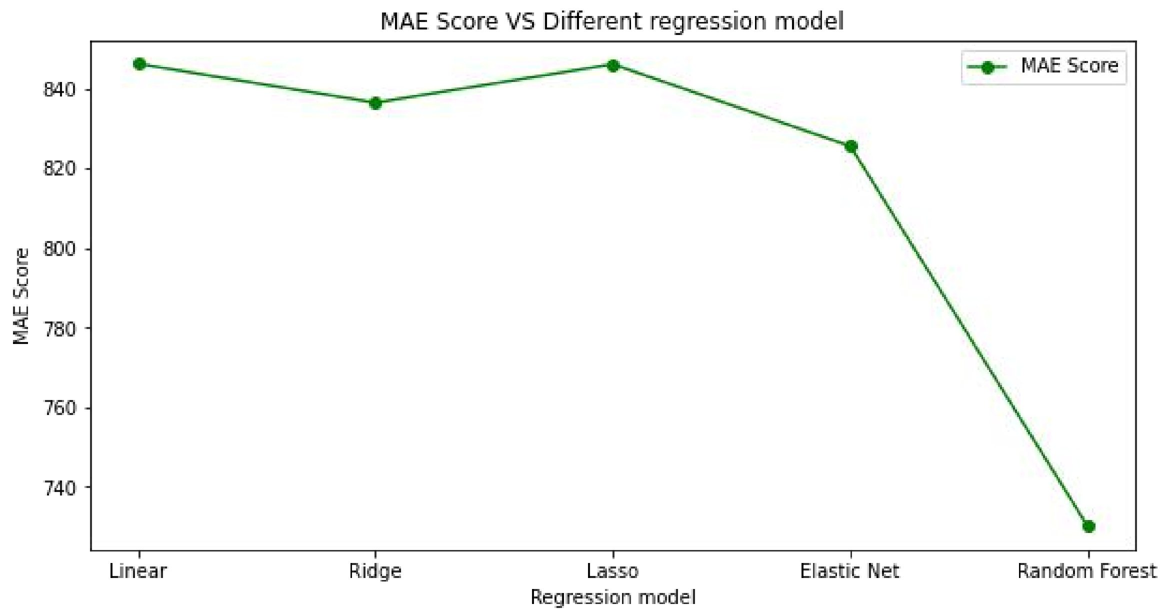
7.2 Visualization validation metrics of different techniques

Random Forest Regression has the best performance on the train set, because it has the highest R-squared score, the lowest RMSE score, and the lowest MAE score in the five different regression model.

Regression model	R-Squared score	RMSE score	MAE score
Linear Regression	0.561279	1287910	846
Ridge Regression	0.563144	1313577	836
Lasso Regression	0.562088	1271537	846
Elastic Net Regression	0.564265	1263974	825
Random Forest Regression	0.608467	1090225	730







8. Ethics (**graduate students only**) - explore ethical and societal considerations related to your project.

How can we make a determination for exploring the privacy of big mart data and analyze these two cases? What should and what should not do in these two privacy of big mart data and analysis? Perhaps the primary limitation is the lack of specificity that comes from applying an ethics framework that is more general than the use capabilities of a specific technology, such as big data.

8.1 Thoughts provoked by the Ted Talk

Ted Talk gave shocked my thinking. I have never touched any knowledge before. I was even relieved that I hadn't worked in data science before, and that biased data analysis results could have such a long and serious impact.

Algorithms are opinions embedded in code. Too tight a threshold for a big mart sale analyst could result may beyond ethical boundaries. For example,who likely is information of critical concern that deserves privacy and for which the release would not be ethical? So, The algorithm takes the biased of the person who designed it. In this way, an algorithm is not fully objective and not truly scientific. As the opinion in TED, a lot can go wrong when we put blind faith in big data.

However, algorithms can go run and even have deeply positive effects with good intentions and reasonable control. For example, using big mart data in the United States health care system indeed brought A favorable influence. This would suggest that the use of big data should not be aimed at causing harm and facilitating greater innovation.

8.2 My opinons

Comparing these two cases in inverse aspect. How can we use the big mart data and analyst properly? What rules should we keep? I think the content of the rules should be large and complex.

As an example of using an existing general ethical framework to generate and facilitate analysis of ethical issues in big data, David Ross laid out seven basic duties of right and wrong conduct. Two of those duties potentially relate to the analysis of big data:

- Duty 1: One ought to do what one can to improve a lot of others.

- Duty 2: One ought not to injure other people.

The key point is how can we determine what should we do for exploring the privacy of big mart data and analyze it? Data Science Association’s code of conduct provides more ethical rules that directly draw on knowledge from the specific discipline (see the ”Data Science Association Code of Conduct Rules’ sidebar). These organizations should monitor and implement developments in new big data of ethics.

As people working in the data science area, when we don't know what the specific guidelines are, we go to the data ethic agency to read the detailed manual. In my opinion, not only the data ethic but also the code ethic is important. Engaged in data analysis and program preparation of people, should obtain a data ethic test, and get a certificate. Because the effects of data ethics and code ethic are long-lasting, we should be in awe of this work.

9. Future Work (graduate students only) - explain what potential next steps or what other questions could be explored based on the results of your work, or given more time.

In this paper” A Two-Level Statistical Model for Big Mart Sales Prediction”, the ensemble of data mining predictive techniques via stacking is considered a two-level statistical approach. It is named as two-level because stacking is performed on two layers in which bottom layer consists of one or more than one learning algorithms and top layer consists of one learning algorithm. Stacking is also known as Stacked Generalization. It basically involves the training of the learning algorithm present in the top layer to combine the predictions made by the algorithms present in the bottom layer.

F. Model Building

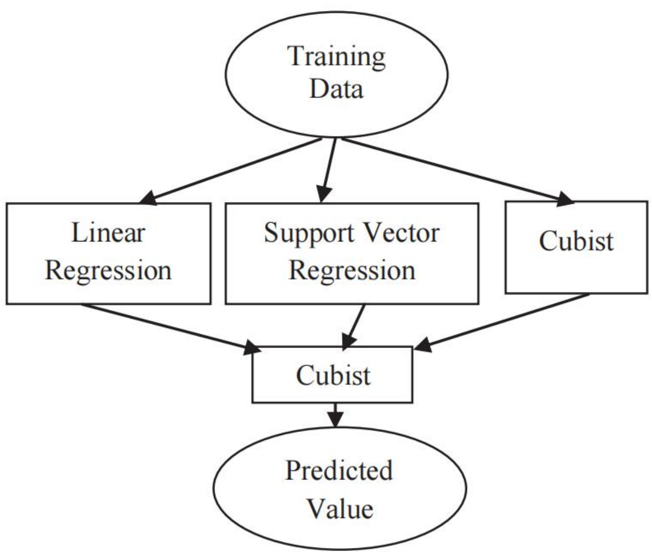


Fig. 3. A Two-level Statistical Model

After completing the phases of data exploration, data cleaning and feature engineering, the data set is ready and set to build predictive models. Model building is a process of building a model that best explains the relationship between predictor variable and response variable. In this paper, model building takes place in two stages. At the first stage, the single model of popular predictive techniques like linear regression, regression tree, cubist, and support vector regression and k- nearest neighbor is built. Then in the second stage, the two-level statistical model was built. The two-level statistical model consisted of machine learning techniques such as linear regression, support vector regression and cubist. These machine learning algorithms are combined and used to make the final prediction. Stacking is a type of ensemble method that is generally used to combine the machine learning techniques to improve the accuracy of predictive models. It is basically a combination of different models that are treated as a single unit. Stacking may have more than two layers, it increases the complexity of the model but it may be useful to make accurate predictions.

I want to try the two-level Statistic model and compare the result with my previous five regression result in section 6 in the future. Use different combinations of my five regression result to find a combination that can get the best performance.