In [ ]:

```
Big_mart Sales prediction using Regression algorithms
```

In [ ]:

```
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 project is to build a predictive model and find ou
t the sales of each product at a particular store.

Using this model, BigMart will try to understand the properties of products and stores w
hich play a key role in increasing sales.

The data has missing values as some stores do not report all the data due to technical g
litches. Hence, it will be required to treat them accordingly.
```

In [ ]:

```
#Every project must go through the data science life cycle they are:
1.identify the business statement and think of big picture.
2.Get the data.[Data collection]
3.Exploratory analysis.[statistics of data]
4.Data cleaning[dat wrangling]
5.Select a model and tarin it.[identify the robust algorithm]
6.Fine-tune your model.[hyper parameter tuning]
7.Present the result.[insights and trends about the data]
8.deploy and provide maintainance
```

In [ ]:

```
the project follows sequence of steps to derive the insights from the data
1.Understanding the problem Statement
2.Importing the Dataset and performing basic EDA
3.Checking for the null values and describing the variables
4.Imputation of the Null-Values using pivot tables
5.Feature Engineering/ Creating New features
6.Using seaborn to understand the contribution of the categorical values on target varia
bles
7.Using boxplot for identifying outliers
8.Fixing categorical variables using Label and One hot encoding
9.Applying Linear, Bayesian Regression models
10.Applying ensemble bagging models like Random Forest and Bagging models
11.Applying boosting models like Gradient Boosting Tree and XGboost
12.Applying Neural Network model MLPRegressor
13.Making function for On spot-checking and selecting the best for hyperparameter tuning
14.Defining function for HyperParameter tuning
15.Standardization and effect of Standardization
16.Understanding Robust Scaler and Normalization
17.Implementing Robust Scaler and Normalization
18.Concluding the final model and predicting for the test data set
19.Saving the model using Joblib
```

In [ ]:

```
#we will start the project by importing the necessary librarys which are essential for an
alysis of data #here we are importing below librarys and aliasing them to hide the comple
xcity[increases Readbilty]

pds=[data manipulation]
npy=[mathematical operations on arrays]
sea=[data visualization in 3d & attractive visualization]
pt=[data visualization in 2d]
```

In [ ]:

```

```

In [6]:

```
#Importing Necessary Libraries
#Matplot and seaborn for making graphs
%matplotlib notebook
from sklearn.linear_model import Ridge
from sklearn.model_selection import KFold, cross_val_score

import numpy as npy
import pandas as pds
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from scipy import stats
import matplotlib.pyplot as pt
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [7]:

```
#Imorting the datasets
Tr =pds.read_csv(r"C:\Users\R411996\Desktop\data-science\Big mart\train_kOBLwZA.csv")
Te=pds.read_csv(r"C:\Users\R411996\Desktop\data-science\Big mart\test_t02dQwI.csv")
```

In [9]:

```
#one can find out the number of rows and columns by using shape function
print(Tr.shape,Te.shape)
```

```
(8523, 12) (5681, 11)
```

In [10]:

```
#here we are creating function named combine which is used to combine both train and test
dataset
def combine(X,Y):
    tt= pds.concat([X,Y],ignore_index=True)
    return tt
```

In [11]:

```
##one can find out the number of rows and columns by using shape function
tt=combine(Tr,Te)
print(tt.shape)
```

```
(14204, 12)
```

In [12]:

```
#head() function displays the top 5 rows of dataset
tt.head()
```

Out[12]:

	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 [13]:

```
#we can specify the limit of the number
tt.head(10)
```

Out[13]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	
5	FDP36	10.395	Regular	0.000000	Baking Goods	51.4008	OUT018	
6	FDO10	13.650	Regular	0.012741	Snack Foods	57.6588	OUT013	
7	FDP10	NaN	Low Fat	0.127470	Snack Foods	107.7622	OUT027	
8	FDH17	16.200	Regular	0.016687	Frozen Foods	96.9726	OUT045	
9	FDU28	19.200	Regular	0.094450	Frozen Foods	187.8214	OUT017	

In [14]:

```
#tail() function displays the bottom 5 rows of dataset
tt.tail()
```

Out[14]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishn
14199	FDB58	10.5	Regular	0.013496	Snack Foods	141.3154	OUT046	
14200	FDD47	7.6	Regular	0.142991	Starchy Foods	169.1448	OUT018	
14201	NCO17	10.0	Low Fat	0.073529	Health and Hygiene	118.7440	OUT045	
14202	FDJ26	15.3	Regular	0.000000	Canned	214.6218	OUT017	
14203	FDU37	9.5	Regular	0.104720	Canned	79.7960	OUT045	

In [15]:

```
#we can specify the limit of the number
tt.tail(15)
```

Out[15]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establish

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishi
14189	FDF34	9.300	Regular	0.014019	Snack Foods	196.5084	OUT046	
14190	FDZ22	9.395	Low Fat	0.045270	Snack Foods	82.1250	OUT046	
14191	FDC44	15.600	Low Fat	0.288892	Fruits and Vegetables	115.1518	OUT010	
14192	FDN31	NaN	Low Fat	0.072529	Fruits and Vegetables	188.0530	OUT027	
14193	FDO03	10.395	Regular	0.037092	Meat	229.4352	OUT017	
14194	FDA01	15.000	reg	0.054463	Canned	59.5904	OUT049	
14195	NCH42	6.860	Low Fat	0.036594	Household	231.1010	OUT049	
14196	FDF46	7.070	Low Fat	0.094053	Snack Foods	116.0834	OUT018	
14197	DRL35	15.700	Low Fat	0.030704	Hard Drinks	43.2770	OUT046	
14198	FDW46	13.000	Regular	0.070411	Snack Foods	63.4484	OUT049	
14199	FDB58	10.500	Regular	0.013496	Snack Foods	141.3154	OUT046	
14200	FDD47	7.600	Regular	0.142991	Starchy Foods	169.1448	OUT018	
14201	NCO17	10.000	Low Fat	0.073529	Health and Hygiene	118.7440	OUT045	
14202	FDJ26	15.300	Regular	0.000000	Canned	214.6218	OUT017	
14203	FDU37	9.500	Regular	0.104720	Canned	79.7960	OUT045	

In [16]:

```
#the describe function gives the statistics about the data
tt.describe()
```

Out[16]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
count	11765.000000	14204.000000	14204.000000	14204.000000	8523.000000
mean	12.792854	0.065953	141.004977	1997.830681	2181.288914
std	4.652502	0.051459	62.086938	8.371664	1706.499616
min	4.555000	0.000000	31.290000	1985.000000	33.290000
25%	8.710000	0.027036	94.012000	1987.000000	834.247400
50%	12.600000	0.054021	142.247000	1999.000000	1794.331000
75%	16.750000	0.094037	185.855600	2004.000000	3101.296400
max	21.350000	0.328391	266.888400	2009.000000	13086.964800

In [17]:

```
#the info() function enumerates over the column and gives details about the data types
tt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14204 entries, 0 to 14203
Data columns (total 12 columns):
 #   Column                     Non-Null Count  Dtype
```

```
---  ------                     --------------  -----
 0   Item_Identifier            14204 non-null  object
 1   Item_Weight                11765 non-null  float64
 2   Item_Fat_Content           14204 non-null  object
 3   Item_Visibility            14204 non-null  float64
 4   Item_Type                  14204 non-null  object
 5   Item_MRP                   14204 non-null  float64
 6   Outlet_Identifier          14204 non-null  object
 7   Outlet_Establishment_Year  14204 non-null  int64
 8   Outlet_Size                10188 non-null  object
 9   Outlet_Location_Type       14204 non-null  object
 10  Outlet_Type                14204 non-null  object
 11  Item_Outlet_Sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(7)
memory usage: 1.3+ MB
```

In [7]:

```
#one can have a count of missing values in dta by using isnull.sum() function
tt.isnull().sum()
#Checks number of null values for all the variables
#Item_Weight has 2439 null values
#Outlet Size has 4016 null values
```

Out[7]:

```
Item_Fat_Content                0
Item_Identifier                 0
Item_MRP                        0
Item_Outlet_Sales            5681
Item_Type                       0
Item_Visibility                 0
Item_Weight                  2439
Outlet_Establishment_Year       0
Outlet_Identifier               0
Outlet_Location_Type            0
Outlet_Size                  4016
Outlet_Type                     0
dtype: int64
```

In [ ]:

```
#the missing values should be filled with appropriate techniques
```

In [8]:

```
#obtaining the unique values in dataset and removing the irrevalant coloumns which doesnt contribute while predicting the target value
tt.apply(lambda x: len(x.unique()))
#Checks the number of unique entries correspnding to each variable
```

Out[8]:

```
Item_Fat_Content                 5
Item_Identifier               1559
Item_MRP                      8052
Item_Outlet_Sales             3494
Item_Type                       16
Item_Visibility              13006
Item_Weight                    416
Outlet_Establishment_Year        9
Outlet_Identifier               10
Outlet_Location_Type             3
Outlet_Size                      4
Outlet_Type                      4
dtype: int64
```

In [9]:

```
#defining a function which counts the occurence of each entry
#frequency of unique entries in each columns with their names
```

```
def countofitem(X,Y):
    for i in Y:
        print("frequency of each category for",i)
        print(X[i].value_counts())
```

In [9]:

```
#defining a function which counts the occurence of each entry

#frequency of unique entries in each columns with their names
category=['Item_Fat_Content','Item_Type','Outlet_Location_Type','Outlet_Size','Outlet_Typ
e']
countofitem(tt,category)
```

```
frequency of each category for Item_Fat_Content
Low Fat    8485
Regular    4824
LF          522
reg         195
low fat     178
Name: Item_Fat_Content, dtype: int64
frequency of each category for Item_Type
Fruits and Vegetables    2013
Snack Foods              1989
Household                1548
Frozen Foods             1426
Dairy                    1136
Baking Goods             1086
Canned                   1084
Health and Hygiene        858
Meat                      736
Soft Drinks               726
Breads                    416
Hard Drinks               362
Others                    280
Starchy Foods             269
Breakfast                 186
Seafood                    89
Name: Item_Type, dtype: int64
frequency of each category for Outlet_Location_Type
Tier 3    5583
Tier 2    4641
Tier 1    3980
Name: Outlet_Location_Type, dtype: int64
frequency of each category for Outlet_Size
Medium    4655
Small     3980
High      1553
Name: Outlet_Size, dtype: int64
frequency of each category for Outlet_Type
Supermarket Type1    9294
Grocery Store        1805
Supermarket Type3    1559
Supermarket Type2    1546
Name: Outlet_Type, dtype: int64
```

In [10]:

```
mode_Outlet_Size=tt.pivot_table(values='Outlet_Size', index='Outlet_Type',aggfunc=(lambd
a x: stats.mode(x)[0]))
print(mode_Outlet_Size)
bool2=tt['Outlet_Size'].isnull()
tt['Outlet_Size'][bool2]=tt['Outlet_Type'][bool2].apply(lambda x : mode_Outlet_Size.loc[
x]).values
sum(tt['Outlet_Size'].isnull())
```

```
                  Outlet_Size
Outlet_Type
Grocery Store           Small
Supermarket Type1       Small
Supermarket Type2      Medium
Supermarket Type3      Medium
```

Out[10]:

0

In [11]:

```
# Correcting the mis-written datas
tt['Item_Fat_Content'].replace(to_replace =['low fat','reg','LF'],
                   value =['Low Fat','Regular','Low Fat'],inplace=True)
tt['Item_Fat_Content'].value_counts()
tt.head()
```

Out[11]:

	Item_Fat_Content	Item_Identifier	Item_MRP	Item_Outlet_Sales	Item_Type	Item_Visibility	Item_Weight	Outlet_Establishme
0	Low Fat	FDA15	249.8092	3735.1380	Dairy	0.016047	9.30	
1	Regular	DRC01	48.2692	443.4228	Soft Drinks	0.019278	5.92	
2	Low Fat	FDN15	141.6180	2097.2700	Meat	0.016760	17.50	
3	Regular	FDX07	182.0950	732.3800	Fruits and Vegetables	0.000000	19.20	
4	Low Fat	NCD19	53.8614	994.7052	Household	0.000000	8.93	

In [12]:

```
avg_item_weight=tt.pivot_table(values='Item_Weight', index='Item_Identifier',aggfunc=[np
y.mean])
print(avg_item_weight)
bool=tt['Item_Weight'].isnull()
tt['Item_Weight'][bool]=tt['Item_Identifier'][bool].apply(lambda x :avg_item_weight.loc[
x]).values
sum(tt['Item_Weight'].isnull())
```

```
                       mean
                Item_Weight
Item_Identifier
DRA12                11.600
DRA24                19.350
DRA59                 8.270
DRB01                 7.390
DRB13                 6.115
DRB24                 8.785
DRB25                12.300
DRB48                16.750
DRC01                 5.920
DRC12                17.850
DRC13                 8.260
DRC24                17.850
DRC25                 5.730
DRC27                13.800
DRC36                13.000
DRC49                 8.670
DRD01                12.100
DRD12                 6.960
DRD13                15.000
DRD15                10.600
DRD24                13.850
DRD25                 6.135
DRD27                18.750
DRD37                 9.800
DRD49                 9.895
DRD60                15.700
DRE01                10.100
DRE03                19.600
```

```
DRE12                4.590
DRE13                6.280
...                    ...
NCX05               15.200
NCX06               17.600
NCX17               21.250
NCX18               14.150
NCX29               10.000
NCX30               16.700
NCX41               19.000
NCX42                6.360
NCX53               20.100
NCX54                9.195
NCY05               13.500
NCY06               15.250
NCY17               18.200
NCY18                7.285
NCY29               13.650
NCY30               20.250
NCY41               16.750
NCY42                6.380
NCY53               20.000
NCY54                8.430
NCZ05                8.485
NCZ06               19.600
NCZ17               12.150
NCZ18                7.825
NCZ29               15.000
NCZ30                6.590
NCZ41               19.850
NCZ42               10.500
NCZ53                9.600
NCZ54               14.650

[1559 rows x 1 columns]
```

Out[12]:

```
0
```

In [13]:

```
#Reducing food category to only 3 types with the help of the first 2 alphabets of the Ite
m_Identifier column

tt['Item_Type_combined']=tt['Item_Identifier'].apply(lambda x : x[0:2])
tt['Item_Type_combined'].replace(to_replace =['FD','DR','NC'],
                  value =['Food','Drinks','Non_consumable'],inplace=True)
 #dropping the redundant column
tt=tt.drop(columns=['Item_Type'])
tt.head()
```

Out[13]:

	Item_Fat_Content	Item_Identifier	Item_MRP	Item_Outlet_Sales	Item_Visibility	Item_Weight	Outlet_Establishment_Year	Out
0	Low Fat	FDA15	249.8092	3735.1380	0.016047	9.30	1999	
1	Regular	DRC01	48.2692	443.4228	0.019278	5.92	2009	
2	Low Fat	FDN15	141.6180	2097.2700	0.016760	17.50	1999	
3	Regular	FDX07	182.0950	732.3800	0.000000	19.20	1998	
4	Low Fat	NCD19	53.8614	994.7052	0.000000	8.93	1987	

In [14]:

```
#Calculating number of Item_fat_contents that are also non_consumable

bool3=tt['Item_Type_combined']=='Non_consumable'
tt['Item_Fat_Content'][bool3]='Non_edible'
tt['Item_Fat_Content'].value_counts()
```

Out[14]:

```
Low Fat       6499
Regular       5019
Non_edible    2686
Name: Item_Fat_Content, dtype: int64
```

In [15]:

```
#Using feature Engineering and adding new column
tt['yearsold']=2013-tt['Outlet_Establishment_Year']
tt=tt.drop(columns=['Outlet_Establishment_Year'])
tt.head()
```

Out[15]:

	Item_Fat_Content	Item_Identifier	Item_MRP	Item_Outlet_Sales	Item_Visibility	Item_Weight	Outlet_Identifier	Outlet_Locatic
0	Low Fat	FDA15	249.8092	3735.1380	0.016047	9.30	OUT049	
1	Regular	DRC01	48.2692	443.4228	0.019278	5.92	OUT018	
2	Low Fat	FDN15	141.6180	2097.2700	0.016760	17.50	OUT049	
3	Regular	FDX07	182.0950	732.3800	0.000000	19.20	OUT010	
4	Non_edible	NCD19	53.8614	994.7052	0.000000	8.93	OUT013	

In [16]:

```
# Converting all the zero values to mean in the visibility column
Item_Visibility_mean=tt.pivot_table(index='Item_Identifier',values='Item_Visibility',agg
func=[npy.mean])
print(Item_Visibility_mean)
bool4=tt['Item_Visibility']==0
tt['Item_Visibility'][bool4]=tt['Item_Identifier'][bool4].apply(lambda x:Item_Visibility
_mean.loc[x] ).values
tt.head()
```

```
                           mean
                Item_Visibility
Item_Identifier
DRA12                  0.034938
DRA24                  0.045646
DRA59                  0.133384
DRB01                  0.079736
DRB13                  0.006799
DRB24                  0.020596
DRB25                  0.079407
DRB48                  0.023973
DRC01                  0.020653
DRC12                  0.037862
DRC13                  0.028408
DRC24                  0.026913
DRC25                  0.047354
DRC27                  0.066423
DRC36                  0.046932
DRC49                  0.070950
DRD01                  0.066330
DRD12                  0.074150
DRD13                  0.049125
DRD15                  0.064930
```

```
DRD24                 0.035205
DRD25                 0.082385
DRD27                 0.020545
DRD37                 0.013352
DRD49                 0.167987
DRD60                 0.040369
DRE01                 0.179808
DRE03                 0.026061
DRE12                 0.061981
DRE13                 0.031673
...                        ...
NCX05                 0.110962
NCX06                 0.017934
NCX17                 0.113709
NCX18                 0.008293
NCX29                 0.101920
NCX30                 0.025977
NCX41                 0.017291
NCX42                 0.006482
NCX53                 0.014409
NCX54                 0.051698
NCY05                 0.059645
NCY06                 0.065816
NCY17                 0.126951
NCY18                 0.033510
NCY29                 0.088295
NCY30                 0.028140
NCY41                 0.086582
NCY42                 0.016440
NCY53                 0.056916
NCY54                 0.191145
NCZ05                 0.063030
NCZ06                 0.102096
NCZ17                 0.076568
NCZ18                 0.180954
NCZ29                 0.076774
NCZ30                 0.027302
NCZ41                 0.056396
NCZ42                 0.011015
NCZ53                 0.026330
NCZ54                 0.081345

[1559 rows x 1 columns]
```

Out[16]:

	Item_Fat_Content	Item_Identifier	Item_MRP	Item_Outlet_Sales	Item_Visibility	Item_Weight	Outlet_Identifier	Outlet_Locatic
0	Low Fat	FDA15	249.8092	3735.1380	0.016047	9.30	OUT049	
1	Regular	DRC01	48.2692	443.4228	0.019278	5.92	OUT018	
2	Low Fat	FDN15	141.6180	2097.2700	0.016760	17.50	OUT049	
3	Regular	FDX07	182.0950	732.3800	0.017834	19.20	OUT010	
4	Non_edible	NCD19	53.8614	994.7052	0.009780	8.93	OUT013	

In [17]:

```
#Checks for correation between different numerical columns
tt.corr()
```

Out[17]:

	Item_MRP	Item_Outlet_Sales	Item_Visibility	Item_Weight	yearsold

	Item_MRP	Item_Outlet_Sales	Item_Visibility	Item_Weight	yearsold
Item_MRP	1.000000	0.567574	-0.007550	0.035751	-0.000141
Item_Outlet_Sales	0.567574	1.000000	-0.128453	0.013261	0.049135
Item_Visibility	-0.007550	-0.128453	1.000000	-0.022028	0.084481
Item_Weight	0.035751	0.013261	-0.022028	1.000000	-0.000247
yearsold	-0.000141	0.049135	0.084481	-0.000247	1.000000

Identifying outliers and fixing them

In [18]:

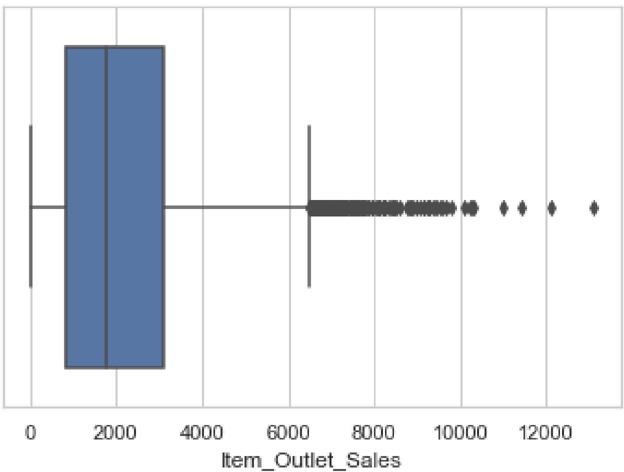
```
tt.describe()
```

Out[18]:

	Item_MRP	Item_Outlet_Sales	Item_Visibility	Item_Weight	yearsold
count	14204.000000	8523.000000	14204.000000	14204.000000	14204.000000
mean	141.004977	2181.288914	0.069710	12.793380	15.169319
std	62.086938	1706.499616	0.049728	4.651716	8.371664
min	31.290000	33.290000	0.003575	4.555000	4.000000
25%	94.012000	834.247400	0.031145	8.710000	9.000000
50%	142.247000	1794.331000	0.057194	12.600000	14.000000
75%	185.855600	3101.296400	0.096930	16.750000	26.000000
max	266.888400	13086.964800	0.328391	21.350000	28.000000

In [19]:

```
sns.set(style="whitegrid")
ax = sns.boxplot(x=tt["Item_Outlet_Sales"])
```



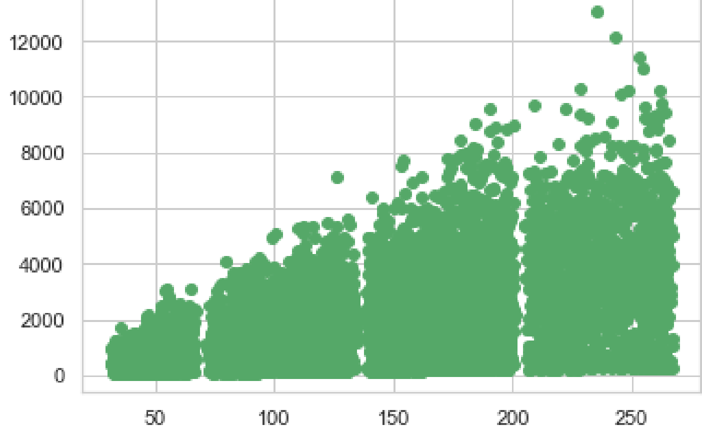
In [20]:

```
#As we know only Item_Outlet_Sales have outliers we can fix them but fixing them will inc
rease our RMSE score
#to a large extent
```

Plotting Graphs for more Analysis

In [21]:

```
#value of sales increases for the increase in MRP of the item
pt.scatter(tt.Item_MRP,df.Item_Outlet_Sales,c='g')
pt.show()
```

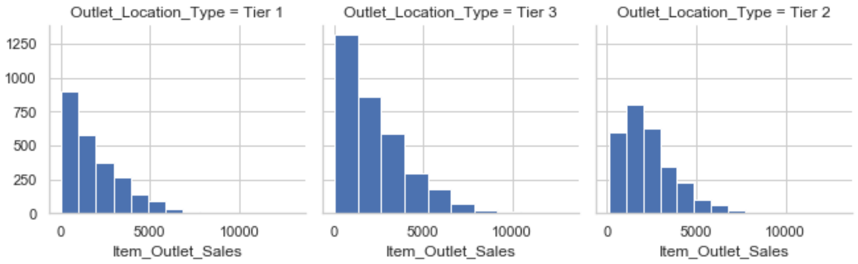


In [22]:

```
sns.FacetGrid(tt, col='Item_Type_combined', size=3, col_wrap=5) \
    .map(plt.hist, 'Item_Outlet_Sales') \
    .add_legend();
# Maximum contribution to outlet sales is from Items that are food type and least is from drinks
```

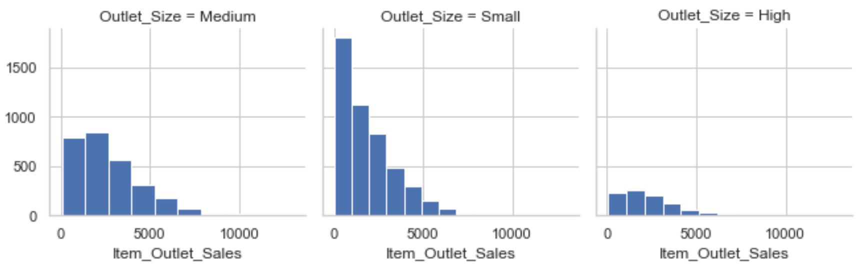
In [23]:

```
sns.FacetGrid(tt, col='Outlet_Location_Type', size=3, col_wrap=5) \
    .map(plt.hist, 'Item_Outlet_Sales') \
    .add_legend();
#Tier3 type of outlet location provides for the maximum sales and other two provides the least sales
```



In [24]:

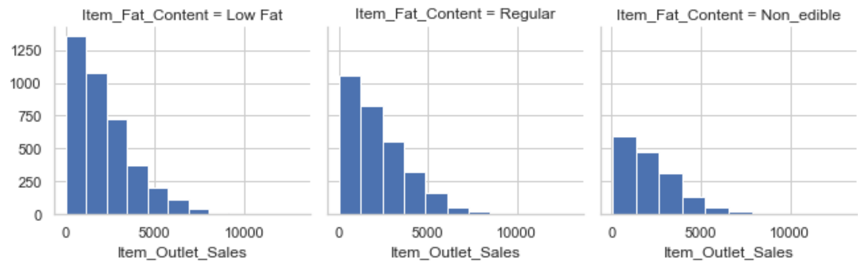
```
sns.FacetGrid(tt, col='Outlet_Size', size=3, col_wrap=5) \
    .map(plt.hist, 'Item_Outlet_Sales') \
    .add_legend();
#Small sized Outlets are providing the maximum sales whereas large sized outlets
# are contributing the least
```



In [25]:

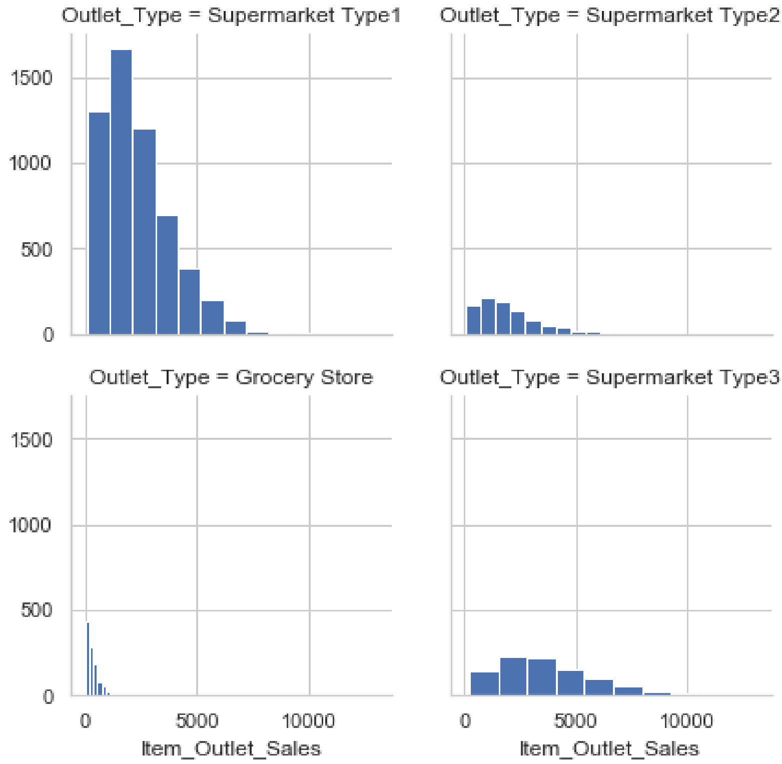
```
sns.FacetGrid(tt, col='Item_Fat_Content', size=3, col_wrap=5) \
```

```
    .map(plt.hist, 'Item_Outlet_Sales') \
    .add_legend();
# people are prefering items with lowest fat content the most
```



In [26]:

```
sns.FacetGrid(tt, col='Outlet_Type', size=3, col_wrap=2) \
    .map(plt.hist, 'Item_Outlet_Sales') \
    .add_legend();
#Maximum of the high sales margin is from Supermarket Type1
#Grocery store has the least sales
```



In [27]:

```
#Label Encoding all the columns with text entries and dropping Item_identifier
le=LabelEncoder()
list=['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','Outlet_Type','Item_Type_co
mbined',
       'Outlet_Size']
for i in list:
    le.fit(tt[i])
    tt[i]=le.transform(tt[i])
tt_new=df.drop(columns='Item_Identifier')
tt_new= pd.get_dummies(tt_new,columns=['Outlet_Identifier'])
tt_new.head()
```

Out[27]:

	Item_Fat_Content	Item_MRP	Item_Outlet_Sales	Item_Visibility	Item_Weight	Outlet_Location_Type	Outlet_Size	Outlet_Type
0	0	249.8092	3735.1380	0.016047	9.30	0	1	1
1	2	48.2692	443.4228	0.019278	5.92	2	1	2
2	0	141.6180	2097.2700	0.016760	17.50	0	1	1
3	2	182.0950	732.3800	0.017834	19.20	2	2	0

4	Item_Fat_Content 1	Item_MRP 53.8614	Item_Outlet_Sales 994.7052	Item_Visibility 0.000780	Item_Weight 8.93	Outlet_Location_Type 2	Outlet_Size 0	Outlet_Type

In [28]:

```
#Separating test and train set
tt_new_train=df_new.iloc[:8523,:]
tt_new_test=df_new.iloc[8523:,:]
tt_new_test=df_new_test.drop(columns=['Item_Outlet_Sales'])
```

In [29]:

```
Y_train=df_new_train['Item_Outlet_Sales']
tt_train_test=tt_new_train.drop(columns=['Item_Outlet_Sales'])
```

In [30]:

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import ElasticNet
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import KFold, cross_val_score
from xgboost import XGBRegressor
import xgboost as xgb
```

In [31]:

```
models = [('lr',LinearRegression()),('ridge',Ridge()),('rfr',RandomForestRegressor()),('
etr',ExtraTreesRegressor()),
         ('br',BaggingRegressor()),('gbr',GradientBoostingRegressor()),('en',ElasticNet(
)),('mlp',MLPRegressor())]
```

In [34]:

```
#Making function for making best 2 models for further hyperparameter tuning
def m_selection(x,y,cross_folds,model):
    scores=[]
    names = []
    for i , j in model:
        cv_scores = cross_val_score(j, x, y, cv=cross_folds,n_jobs=5)
        scores.append(cv_scores)
        names.append(i)
    for k in range(len(scores)):
        print(names[k],scores[k].mean())
```

In [35]:

```
m_selection(tt_train_test,Y_train,4,models)
```

```
lr 0.5600167514366813
ridge 0.5600211200777783
rfr 0.5259810264637599
etr 0.49389106307163183
br 0.5248672168702679
gbr 0.5924393177295112
en 0.47782907311746925
mlp 0.5661736156299884
```

In [36]:

```
#Average score for XGBoost matrix
# define data_dmatrix
data_dmatrix = xgb.DMatrix(data=tt_train_test,label=Y_train)
# import XGBRegressor
xgb1 = XGBRegressor()
cv_score = cross_val_score(xgb1, tt_train_test, Y_train, cv=4,n_jobs=5)
```

```
print(cv_score.mean())
```

```
0.5951594627612504
```

Gradient Boost Regression and XGBoost Regression will be used for further hyperparameter tuning

In [37]:

```
def model_parameter_tuning(x,y,model,parameters,cross_folds):
    model_grid = GridSearchCV(model,
                    parameters,
                    cv = cross_folds,
                    n_jobs = 5,
                    verbose=True)
    model_grid.fit(x,y)
    y_predicted = model_grid.predict(x)
    print(model_grid.score)
    print(model_grid.best_params_)
    print("The RMSE score is",np.sqrt(np.mean((y-y_predicted)**2)))

#defining function for hyper parameter tuning and using RMSE as my metric
```

In [50]:

```
parameters_xgb = {'nthread':[3,4],
              'learning_rate':[0.02,0.03], #so called `eta` value
              'max_depth': [3,2,4],
              'min_child_weight':[3,4,5],
              'silent': [1],
              'subsample': [0.5],
              'colsample_bytree': [0.7],
              'n_estimators': [300,320]
             }
parameters_gbr={'loss':['ls','lad'],
               'learning_rate':[0.3],
               'n_estimators':[300],
               'min_samples_split':[3,4],
               'max_depth':[3,4],
               'min_samples_leaf':[3,4,2],
               'max_features':['auto','log2','sqrt']
              }

# Defining the useful parameters for parameter tuning
# to get the optimum output
```

In [39]:

```
model_parameter_tuning(tt_train_test,Y_train,xgb1,parameters_xgb,4)
```

```
Fitting 4 folds for each of 72 candidates, totalling 288 fits
```

```
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  40 tasks      | elapsed:   22.8s
[Parallel(n_jobs=5)]: Done 190 tasks      | elapsed:  1.8min
[Parallel(n_jobs=5)]: Done 288 out of 288 | elapsed:  2.7min finished
```

```
<bound method BaseSearchCV.score of GridSearchCV(cv=4, error_score='raise-deprecating',
             estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                    colsample_bylevel=1, colsample_bynode=1,
                                    colsample_bytree=1, gamma=0,
                                    importance_type='gain', learning_rate=0.1,
                                    max_delta_step=0, max_depth=3,
                                    min_child_weight=1, missing=None,
                                    n_estimators=100, n_jobs=1, nthread=None,
                                    objective='reg:linear', random_state=0,...
                                    scale_pos_weight=1, seed=None, silent=None,
                                    subsample=1, verbosity=1),
             iid='warn', n_jobs=5,
             param_grid={'colsample_bytree': [0.7],
```

```
                         'learning_rate': [0.02, 0.03], 'max_depth': [3, 2, 4],
                         'min_child_weight': [3, 4, 5],
                         'n_estimators': [300, 320], 'nthread': [3, 4],
                         'silent': [1], 'subsample': [0.5]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=True)>
{'colsample_bytree': 0.7, 'learning_rate': 0.02, 'max_depth': 3, 'min_child_weight': 3, '
n_estimators': 300, 'nthread': 3, 'silent': 1, 'subsample': 0.5}
The RMSE score is 1055.85632573498
```

In [40]:

```
gbr=GradientBoostingRegressor()
model_parameter_tuning(tt_train_test,Y_train,gbr,parameters_gbr,4)
```

```
Fitting 4 folds for each of 324 candidates, totalling 1296 fits
```

```
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  40 tasks      | elapsed:   17.1s
[Parallel(n_jobs=5)]: Done 190 tasks      | elapsed:  1.0min
[Parallel(n_jobs=5)]: Done 440 tasks      | elapsed:  2.6min
[Parallel(n_jobs=5)]: Done 790 tasks      | elapsed:  4.9min
[Parallel(n_jobs=5)]: Done 1240 tasks      | elapsed:  7.7min
[Parallel(n_jobs=5)]: Done 1296 out of 1296 | elapsed:  8.1min finished
```

```
<bound method BaseSearchCV.score of GridSearchCV(cv=4, error_score='raise-deprecating',
             estimator=GradientBoostingRegressor(alpha=0.9,
                                                 criterion='friedman_mse',
                                                 init=None, learning_rate=0.1,
                                                 loss='ls', max_depth=3,
                                                 max_features=None,
                                                 max_leaf_nodes=None,
                                                 min_impurity_decrease=0.0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_leaf=0.0,
                                                 n_estimators=100,
                                                 n_iter...
                                                 validation_fraction=0.1,
                                                 verbose=0, warm_start=False),
             iid='warn', n_jobs=5,
             param_grid={'learning_rate': [0.3, 0.6], 'loss': ['ls', 'lad'],
                         'max_depth': [3, 4, 5],
                         'max_features': ['auto', 'log2', 'sqrt'],
                         'min_samples_leaf': [3, 4, 2],
                         'min_samples_split': [3, 4, 2],
                         'n_estimators': [300]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=True)>
{'learning_rate': 0.3, 'loss': 'lad', 'max_depth': 3, 'max_features': 'auto', 'min_sample
s_leaf': 2, 'min_samples_split': 3, 'n_estimators': 300}
The RMSE score is 1051.766720606019
```

In [41]:

```
from sklearn.neural_network import MLPRegressor
mp=MLPRegressor()
parameters_mp = {'hidden_layer_sizes':[300,400,500],
              'activation':['relu','tanh'],
              'learning_rate':['adaptive'],
              'learning_rate_init':[0.001,0.004],
              'solver':['adam'],
              'max_iter':[200,300]
             }
```

In [42]:

```
model_parameter_tuning(df_train_test,Y_train,mlp,parameters_mlp,4)
```

```
Fitting 4 folds for each of 24 candidates, totalling 96 fits
```

```
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  40 tasks      | elapsed:  3.1min
[Parallel(n_jobs=5)]: Done  96 out of  96 | elapsed:  8.6min finished
```

```
<bound method BaseSearchCV.score of GridSearchCV(cv=4, error_score='raise-deprecating',
             estimator=MLPRegressor(activation='relu', alpha=0.0001,
                                    batch_size='auto', beta_1=0.9, beta_2=0.999,
                                    early_stopping=False, epsilon=1e-08,
                                    hidden_layer_sizes=(100,),
                                    learning_rate='constant',
                                    learning_rate_init=0.001, max_iter=200,
                                    momentum=0.9, n_iter_no_change=10,
                                    nesterovs_momentum=True, power_t=0.5,
                                    random_stat...
                                    solver='adam', tol=0.0001,
                                    validation_fraction=0.1, verbose=False,
                                    warm_start=False),
             iid='warn', n_jobs=5,
             param_grid={'activation': ['relu', 'tanh'],
                         'hidden_layer_sizes': [300, 400, 500],
                         'learning_rate': ['adaptive'],
                         'learning_rate_init': [0.001, 0.004],
                         'max_iter': [200, 300], 'solver': ['adam']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=True)>
{'activation': 'relu', 'hidden_layer_sizes': 400, 'learning_rate': 'adaptive', 'learning_
rate_init': 0.004, 'max_iter': 300, 'solver': 'adam'}
The RMSE score is 1071.4634280469263
```

Standardization of the model before training

In [43]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
standard=scaler.fit_transform(tt_train_test)
column_names = tt_train_test.columns
tt_standard = pds.DataFrame(data=standard,columns=column_names)
tt_standard.head()
```

Out[43]:

	Item_Fat_Content	Item_MRP	Item_Visibility	Item_Weight	Outlet_Location_Type	Outlet_Size	Outlet_Type	Item_Type_combi
0	-0.997813	1.747454	-1.081039	-0.769246	-1.369334	-0.664080	-0.252658	-0.179
1	1.236942	-1.489023	-1.016230	-1.496813	1.091569	-0.664080	1.002972	-2.095
2	-0.997813	0.010040	-1.066741	0.995858	-1.369334	-0.664080	-0.252658	-0.179
3	1.236942	0.660050	-1.045193	1.361794	1.091569	0.799954	-1.508289	-0.179
4	0.119565	-1.399220	-1.206757	-0.848890	1.091569	-2.128115	-0.252658	1.735

In [44]:

```
m_selection(tt_standard,Y_train,4,models)
```

```
lr 0.5599682122990035
ridge 0.5600174793091026
rfr 0.5165207793760711
etr 0.4956682768018107
br 0.5171096007508749
gbr 0.5925857985119254
en 0.5116677567108185
mlp 0.5952554309685527
```

In [45]:

```
#Average score for XGBoost matrix
# define data_dmatrix
data_dmatrix = xgb.DMatrix(data=tt_standard,label=Y_train)
# import XGBRegressor
xgb1 = XGBRegressor()
cv_score = cross_val_score(xgb1, tt_standard, Y_train, cv=4,n_jobs=5)
print(cv_score.mean())
```

```
0.5951547716500721
```

The Models for hyperparameter tuning are same XGBoost and GradientBoostingRegression

In [46]:

```
model_parameter_tuning(df_standard,Y_train,xgb1,parameters_xgb,4)
```

```
Fitting 4 folds for each of 72 candidates, totalling 288 fits
```

```
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  40 tasks      | elapsed:   27.9s
[Parallel(n_jobs=5)]: Done 190 tasks      | elapsed:  2.1min
[Parallel(n_jobs=5)]: Done 288 out of 288 | elapsed:  3.1min finished
```

```
<bound method BaseSearchCV.score of GridSearchCV(cv=4, error_score='raise-deprecating',
             estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                    colsample_bylevel=1, colsample_bynode=1,
                                    colsample_bytree=1, gamma=0,
                                    importance_type='gain', learning_rate=0.1,
                                    max_delta_step=0, max_depth=3,
                                    min_child_weight=1, missing=None,
                                    n_estimators=100, n_jobs=1, nthread=None,
                                    objective='reg:linear', random_state=0,...
                                    scale_pos_weight=1, seed=None, silent=None,
                                    subsample=1, verbosity=1),
             iid='warn', n_jobs=5,
             param_grid={'colsample_bytree': [0.7],
                         'learning_rate': [0.02, 0.03], 'max_depth': [3, 2, 4],
                         'min_child_weight': [3, 4, 5],
                         'n_estimators': [300, 320], 'nthread': [3, 4],
                         'silent': [1], 'subsample': [0.5]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=True)>
{'colsample_bytree': 0.7, 'learning_rate': 0.02, 'max_depth': 3, 'min_child_weight': 3, '
n_estimators': 300, 'nthread': 3, 'silent': 1, 'subsample': 0.5}
The RMSE score is 1055.8552017069048
```

In [47]:

```
model_parameter_tuning(tt_standard,Y_train,gbr,parameters_gbr,4)
```

```
Fitting 4 folds for each of 324 candidates, totalling 1296 fits
```

```
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  40 tasks      | elapsed:   15.6s
[Parallel(n_jobs=5)]: Done 190 tasks      | elapsed:   56.4s
[Parallel(n_jobs=5)]: Done 440 tasks      | elapsed:  2.6min
[Parallel(n_jobs=5)]: Done 790 tasks      | elapsed:  5.2min
[Parallel(n_jobs=5)]: Done 1240 tasks      | elapsed:  8.3min
[Parallel(n_jobs=5)]: Done 1296 out of 1296 | elapsed:  8.7min finished
```

```
<bound method BaseSearchCV.score of GridSearchCV(cv=4, error_score='raise-deprecating',
             estimator=GradientBoostingRegressor(alpha=0.9,
                                                 criterion='friedman_mse',
                                                 init=None, learning_rate=0.1,
                                                 loss='ls', max_depth=3,
                                                 max_features=None,
                                                 max_leaf_nodes=None,
```

```
                                                  min_impurity_decrease=0.0,
                                                  min_impurity_split=None,
                                                  min_samples_leaf=1,
                                                  min_samples_split=2,
                                                  min_weight_fraction_leaf=0.0,
                                                  n_estimators=100,
                                                  n_iter...
                                                  validation_fraction=0.1,
                                                  verbose=0, warm_start=False),
             iid='warn', n_jobs=5,
             param_grid={'learning_rate': [0.3, 0.6], 'loss': ['ls', 'lad'],
                         'max_depth': [3, 4, 5],
                         'max_features': ['auto', 'log2', 'sqrt'],
                         'min_samples_leaf': [3, 4, 2],
                         'min_samples_split': [3, 4, 2],
                         'n_estimators': [300]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=True)>
{'learning_rate': 0.3, 'loss': 'lad', 'max_depth': 3, 'max_features': 'auto', 'min_sample
s_leaf': 2, 'min_samples_split': 2, 'n_estimators': 300}
The RMSE score is 1054.081191236635
```

In [49]:

```
tt_train_test.head()
```

Out[49]:

	Item_Fat_Content	Item_MRP	Item_Visibility	Item_Weight	Outlet_Location_Type	Outlet_Size	Outlet_Type	Item_Type_combi
0	0	249.8092	0.016047	9.30	0	1	1	
1	2	48.2692	0.019278	5.92	2	1	2	
2	0	141.6180	0.016760	17.50	0	1	1	
3	2	182.0950	0.017834	19.20	2	2	0	
4	1	53.8614	0.009780	8.93	2	0	1	

Using Robust Scaler

My dataset having outliers make it more prone to mistakes

Robust Scaler handles the outliers as well

It scales according to the quartile range

In [51]:

```
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import MinMaxScaler

normalizedata = MinMaxScaler()
robust = RobustScaler(quantile_range = (0.1,0.8)) #range of inerquartile is one of the pa
rameters
robust_stan = robust.fit_transform(tt_train_test)
robust_stan_normalize = normalizedata.fit_transform(robust_stan)
# also normalized the dataset using MinMaxScaler i.e has bought the data set between (0,1
)
tt_robust_normalize = pds.DataFrame(robust_stan_normalize,columns=column_names)
tt_robust_normalize.head()
```

Out[51]:

	Item_Fat_Content	Item_MRP	Item_Visibility	Item_Weight	Outlet_Location_Type	Outlet_Size	Outlet_Type	Item_Type_combi

	Item_Fat_Content	Item_MRP	Item_Visibility	Item_Weight	Outlet_Location_Type	Outlet_Size	Outlet_Type	Item_Type_combi
0	0.0	0.927507	0.038399	0.982525	0.0	0.5	0.333333	
1	1.0	0.072068	0.048346	0.081274	1.0	0.5	0.666667	
2	0.0	0.468288	0.040593	0.770765	0.0	0.5	0.333333	
3	1.0	0.640093	0.043901	0.871986	1.0	1.0	0.000000	
4	0.5	0.095805	0.019104	0.260494	1.0	0.0	0.333333	

In [52]:

```
m_selection(tt_robust_normalize,Y_train,4,models)
```

```
lr 0.5599279983880873
ridge 0.5600244473901529
rfr 0.5165027968093021
etr 0.4962660499002653
br 0.5162076976506412
gbr 0.5925765424070908
en 0.16451782722500888
mlp 0.494242396352599
```

In [53]:

```
crv_score = cross_val_score(xgb1, tt_robust_normalize, Y_train, cv=4,n_jobs=5)
print(cv_score.mean())
```

```
0.5951547716500721
```

In [54]:

```
model_parameter_tuning(tt_robust_normalize,Y_train,xgb1,parameters_xgb,4)
```

```
Fitting 4 folds for each of 72 candidates, totalling 288 fits
```

```
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  40 tasks      | elapsed:   22.3s
[Parallel(n_jobs=5)]: Done 190 tasks      | elapsed:  1.8min
[Parallel(n_jobs=5)]: Done 288 out of 288 | elapsed:  2.7min finished
```

```
<bound method BaseSearchCV.score of GridSearchCV(cv=4, error_score='raise-deprecating',
             estimator=XGBRegressor(base_score=0.5, booster='gbtree',
                                    colsample_bylevel=1, colsample_bynode=1,
                                    colsample_bytree=1, gamma=0,
                                    importance_type='gain', learning_rate=0.1,
                                    max_delta_step=0, max_depth=3,
                                    min_child_weight=1, missing=None,
                                    n_estimators=100, n_jobs=1, nthread=None,
                                    objective='reg:linear', random_state=0,...
                                    scale_pos_weight=1, seed=None, silent=None,
                                    subsample=1, verbosity=1),
             iid='warn', n_jobs=5,
             param_grid={'colsample_bytree': [0.7],
                         'learning_rate': [0.02, 0.03], 'max_depth': [3, 2, 4],
                         'min_child_weight': [3, 4, 5],
                         'n_estimators': [300, 320], 'nthread': [3, 4],
                         'silent': [1], 'subsample': [0.5]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=True)>
{'colsample_bytree': 0.7, 'learning_rate': 0.02, 'max_depth': 3, 'min_child_weight': 3, '
n_estimators': 300, 'nthread': 3, 'silent': 1, 'subsample': 0.5}
The RMSE score is 1055.8687294077038
```

In [55]:

```
model_parameter_tuning(tt_robust_normalize,Y_train,gbr,parameters_gbr,4)
```

```
Fitting 4 folds for each of 72 candidates, totalling 288 fits
```

```
[Parallel(n_jobs=5)]: Using backend LokyBackend with 5 concurrent workers.
[Parallel(n_jobs=5)]: Done  40 tasks      | elapsed:   11.5s
[Parallel(n_jobs=5)]: Done 190 tasks      | elapsed:   51.9s
```

```
[Parallel(n_jobs=5)]: Done 288 out of 288 | elapsed:  1.4min finished
```

```
<bound method BaseSearchCV.score of GridSearchCV(cv=4, error_score='raise-deprecating',
             estimator=GradientBoostingRegressor(alpha=0.9,
                                                 criterion='friedman_mse',
                                                 init=None, learning_rate=0.1,
                                                 loss='ls', max_depth=3,
                                                 max_features=None,
                                                 max_leaf_nodes=None,
                                                 min_impurity_decrease=0.0,
                                                 min_impurity_split=None,
                                                 min_samples_leaf=1,
                                                 min_samples_split=2,
                                                 min_weight_fraction_leaf=0.0,
                                                 n_estimators=100,
                                                 n_iter...
                                                 subsample=1.0, tol=0.0001,
                                                 validation_fraction=0.1,
                                                 verbose=0, warm_start=False),
             iid='warn', n_jobs=5,
             param_grid={'learning_rate': [0.3], 'loss': ['ls', 'lad'],
                         'max_depth': [3, 4],
                         'max_features': ['auto', 'log2', 'sqrt'],
                         'min_samples_leaf': [3, 4, 2],
                         'min_samples_split': [3, 4], 'n_estimators': [300]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=True)>
{'learning_rate': 0.3, 'loss': 'lad', 'max_depth': 3, 'max_features': 'auto', 'min_sample
s_leaf': 2, 'min_samples_split': 3, 'n_estimators': 300}
The RMSE score is 1049.14085875651
```

Best Model

Comparing all models using RMSE score

Gradient Boosting Method is the best method when implemented using Robust Scaler and MinMaxScaler normalization

PARAMETERS AND RMSE RESPECTIVELY {'learning_rate': 0.3, 'loss': 'lad', 'max_depth': 3, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 300} The RMSE score is 1049.14085875651

In [56]:

```
robust_test = robust.fit_transform(tt_new_test)
robust_normalize_test = normalize.fit_transform(robust_test)
tt_test_robust_normalize = pds.DataFrame(robust_normalize_test,columns=column_names)
```

In [59]:

```
gbr = GradientBoostingRegressor(learning_rate= 0.3, loss= 'lad',max_depth= 3,min_samples_
leaf=2,min_samples_split=3
                                ,n_estimators= 300)
# Defining my final model that I will use for prediction
```

In [60]:

```
gbr.fit(tt_robust_normalize,Y_train)
```

Out[60]:

```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                          learning_rate=0.3, loss='lad', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=2, min_samples_split=3,
```

```
                              min_weight_fraction_leaf=0.0, n_estimators=300,
                              n_iter_no_change=None, presort='auto',
                              random_state=None, subsample=1.0, tol=0.0001,
                              validation_fraction=0.1, verbose=0, warm_start=False)
```

In [61]:

```
fi_pr=gbr.predict(df_test_robust_normalize) #Predicting the outlet sales
```

In [65]:

```
#the prediction is in the form of numpy array
# Converting into Dataframe
tt_final_prediction = pds.DataFrame(fi_pr,columns=['Item_Outlet_Sales'])
```

In [66]:

```
tt_final_prediction.head()
```

Out[66]:

	Item_Outlet_Sales
0	1621.189785
1	1285.430878
2	531.413666
3	2569.549126
4	5662.989576

Saving the final model using Joblib

In [62]:

```
import joblib as jb
filename = 'final_model.sav' # Name of the model
jb.dump(gbr, filename) # it is saved in your current working directory
```

Out[62]:

```
['final_model.sav']
```

In [67]:

```
# This command loads the model once again
L_model = jb.load(filename)
```