In [1]:

```
import pandas as pd
import numpy as np
bm=pd.read_csv('bigdatamart.csv')
```

In [2]:

```
bm
```

Out[2]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	1999
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	1999
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998
4	NCD19	8.930	Low Fat	0.000000	Household	53.8614	OUT013	1987
...	...	...	...	...	...	...	...	...
8518	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5218	OUT013	1987
8519	FDS36	8.380	Regular	0.046982	Baking Goods	108.1570	OUT045	2002
8520	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	OUT035	2004
8521	FDN46	7.210	Regular	0.145221	Snack Foods	103.1332	OUT018	2009
8522	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.4670	OUT046	1997

8523 rows × 12 columns

In [3]:

```
bm.dtypes
```

Out[3]:

```
Item_Identifier               object
Item_Weight                  float64
Item_Fat_Content              object
Item_Visibility              float64
Item_Type                     object
Item_MRP                     float64
Outlet_Identifier             object
Outlet_Establishment_Year      int64
Outlet_Size                   object
Outlet_Location_Type          object
Outlet_Type                   object
Item_Outlet_Sales            float64
dtype: object
```

In [4]:

```
bm.columns
```

Out[4]:

```
Index(['Item_Identifier', 'Item_Weight', 'Item_Fat_Content', 'Item_Visibility',
       'Item_Type', 'Item_MRP', 'Outlet_Identifier',
```

```
       'Outlet_Establishment_Year', 'Outlet_Size', 'Outlet_Location_Type',
       'Outlet_Type', 'Item_Outlet_Sales'],
      dtype='object')
```

In [5]:

```
bm.describe()
```

Out[5]:

	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

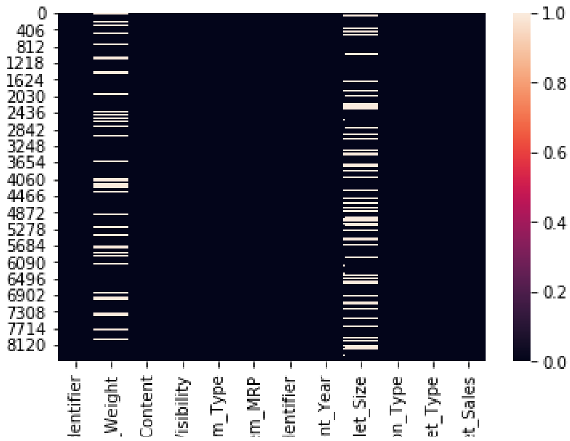
In [6]:

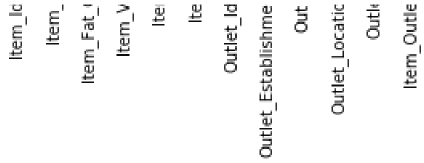
```
bm.info()
```

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

In [7]:

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(bm.isnull())
plt.show()
```





In [8]:

```
bm.Item_Weight=bm.Item_Weight.fillna(bm.Item_Weight.mean())
```

In [9]:

```
bm['Outlet_Size'].value_counts()
```

Out[9]:

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

In [10]:

```
bm.Outlet_Size=bm.Outlet_Size.fillna(bm.Outlet_Size.fillna('Medium'))
```

In [11]:

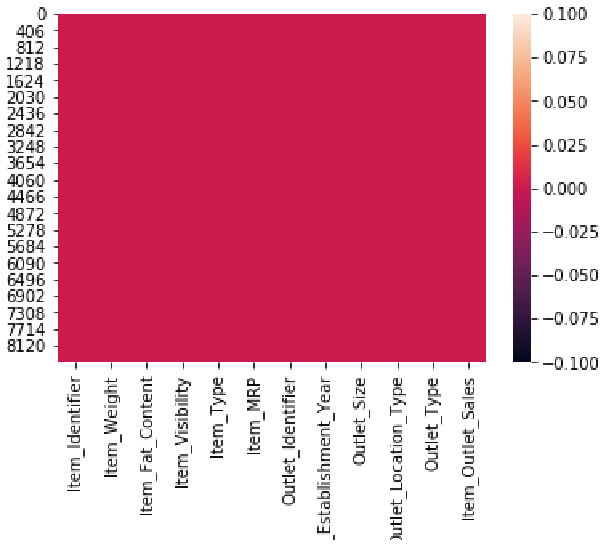
```
bm.isnull().sum()
```

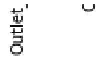
Out[11]:

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

In [12]:

```
sns.heatmap(bm.isnull())
plt.show()
```





In [13]:

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
list1=['Item_Fat_Content','Item_Type','Outlet_Size','Outlet_Type','Outlet_Location_Type']
for val in list1:
    bm[val]=le.fit_transform(bm[val].astype(str))
```

In [14]:

```
bm.dtypes
```

Out[14]:

```
Item_Identifier               object
Item_Weight                  float64
Item_Fat_Content               int32
Item_Visibility              float64
Item_Type                      int32
Item_MRP                     float64
Outlet_Identifier             object
Outlet_Establishment_Year      int64
Outlet_Size                    int32
Outlet_Location_Type           int32
Outlet_Type                    int32
Item_Outlet_Sales            float64
dtype: object
```

In [15]:

```
bm1=bm.drop(['Item_Identifier','Outlet_Identifier','Outlet_Establishment_Year'],axis=1)
```

In [16]:

```
bm1.shape
```

Out[16]:

```
(8523, 9)
```

In [17]:

```
bm1.skew()
```

Out[17]:

```
Item_Weight             0.090561
Item_Fat_Content        0.994824
Item_Visibility         1.167091
Item_Type               0.101655
Item_MRP                0.127202
Outlet_Size            -0.087072
Outlet_Location_Type   -0.209093
Outlet_Type             0.927438
Item_Outlet_Sales       1.177531
dtype: float64
```

In [34]:

```
for col in bm1.columns:
    if bm1[col].skew()>0.55:
        bm1[col]=np.log1p(bm1[col])
```

In [35]:

```
bm1.skew()
```

Out[35]:

```
Item_Weight                0.090561
Item_Fat_Content          -0.332843
Item_Visibility            1.015334
Item_Type                  0.101655
Item_MRP                   0.127202
Outlet_Size               -0.087072
Outlet_Location_Type      -0.209093
Outlet_Type               -0.236040
Item_Outlet_Sales         -0.882266
dtype: float64
```

In [36]:

```
sns.pairplot(bm1)
```

Out[36]:

```
<seaborn.axisgrid.PairGrid at 0x20895dfc688>
```

In [37]:

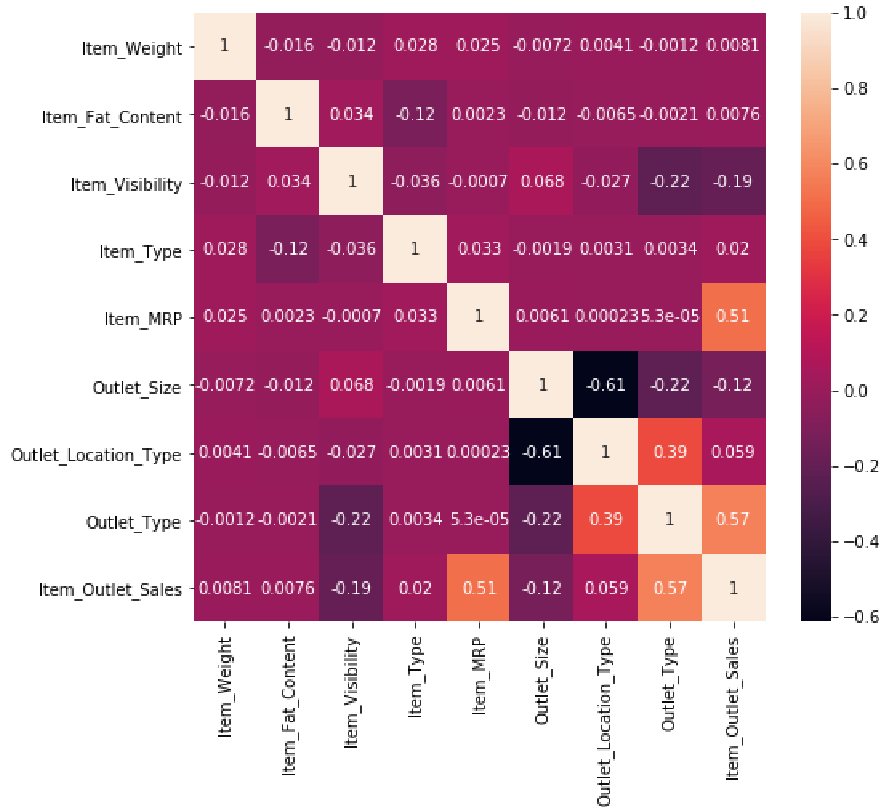
```
bm1.corr()
```

Out[37]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Size	Outlet_Location_Type	Outlet_
Item_Weight	1.000000	-0.015940	-0.012044	0.028015	0.024756	-0.007225	0.004088	-0.00
Item_Fat_Content	-0.015940	1.000000	0.033649	-0.115934	0.002278	-0.011713	-0.006528	-0.00
Item_Visibility	-0.012044	0.033649	1.000000	-0.035995	-0.000701	0.067534	-0.027210	-0.22
Item_Type	0.028015	-0.115934	-0.035995	1.000000	0.032651	-0.001859	0.003084	0.00
Item_MRP	0.024756	0.002278	-0.000701	0.032651	1.000000	0.006059	0.000232	0.00
Outlet_Size	-0.007225	-0.011713	0.067534	-0.001859	0.006059	1.000000	-0.614311	-0.22
Outlet_Location_Type	0.004088	-0.006528	-0.027210	0.003084	0.000232	-0.614311	1.000000	0.38
Outlet_Type	-0.001187	-0.002072	-0.220345	0.003380	0.000053	-0.223204	0.389361	1.00
Item_Outlet_Sales	0.008059	0.007620	-0.188500	0.019914	0.509886	-0.122951	0.059030	0.57

In [38]:

```
corr_hmap=bm1.corr()
plt.figure(figsize=(8,7))
sns.heatmap(corr_hmap,annot=True)
plt.show()
```



In [ ]:

```

```

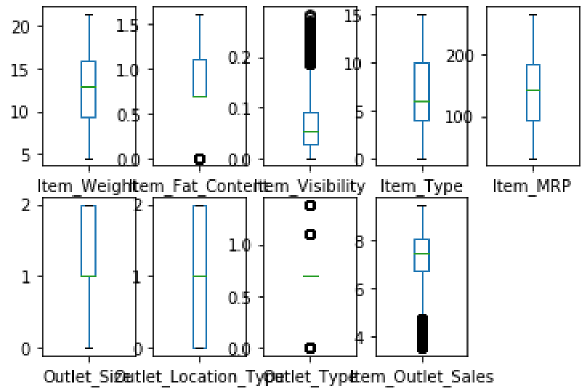
In [39]:

```
bm1.plot(kind='box',subplots=True,layout=(2,5))
```

Out[39]:

```
Item_Weight            AxesSubplot(0.125,0.536818;0.133621x0.343182)
Item_Fat_Content    AxesSubplot(0.285345,0.536818;0.133621x0.343182)
Item_Visibility      AxesSubplot(0.44569,0.536818;0.133621x0.343182)
Item_Type           AxesSubplot(0.606034,0.536818;0.133621x0.343182)
```

```
Item_MRP                  AxesSubplot(0.766379,0.536818;0.133621x0.343182)
Outlet_Size                     AxesSubplot(0.125,0.125;0.133621x0.343182)
Outlet_Location_Type         AxesSubplot(0.285345,0.125;0.133621x0.343182)
Outlet_Type                   AxesSubplot(0.44569,0.125;0.133621x0.343182)
Item_Outlet_Sales            AxesSubplot(0.606034,0.125;0.133621x0.343182)
dtype: object
```



In [40]:

```
bm1.shape
```

Out[40]:

```
(8523, 9)
```

In [41]:

```
bm1.head()
```

Out[41]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_S
0	9.30	0.693147	0.015920	4	249.8092	1	0	0.693147	8.225
1	5.92	1.098612	0.019095	14	48.2692	1	2	1.098612	6.096
2	17.50	0.693147	0.016621	10	141.6180	1	0	0.693147	7.648
3	19.20	1.098612	0.000000	6	182.0950	1	2	0.000000	6.597
4	8.93	0.693147	0.000000	9	53.8614	0	2	0.693147	6.903

In [42]:

```
#Removing outliers
from scipy.stats import zscore
z_score=abs(zscore(bm1))
print(bm1.shape)
bmr=bm1.loc[(z_score<3).all(axis=1)]
print(bmr.shape)
```

```
(8523, 9)
(8075, 9)
```

In [43]:

```
x1=bmr.iloc[:,0:-1]
x1.head()
```

Out[43]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Size	Outlet_Location_Type	Outlet_Type
0	9.30	0.693147	0.015920	4	249.8092	1	0	0.693147
1	5.92	1.098612	0.019095	14	48.2692	1	2	1.098612

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Size	Outlet_Location_Type	Outlet_Type
2	17.50	0.693147	0.016621	10	141.6180	1	0	0.693147
3	19.20	1.098612	0.000000	6	182.0950	1	2	0.000000
4	8.93	0.693147	0.000000	9	53.8614	0	2	0.693147

In [44]:

```
y=bmr.iloc[:,-1]
y.head()
```

Out[44]:

```
0    8.225808
1    6.096776
2    7.648868
3    6.597664
4    6.903451
Name: Item_Outlet_Sales, dtype: float64
```

In [45]:

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x=sc.fit_transform(x1)
x=pd.DataFrame(x,columns=x1.columns)
```

In [46]:

```
x.shape
```

Out[46]:

```
(8075, 8)
```

In [47]:

```
y.shape
```

Out[47]:

```
(8075,)
```

In [48]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
max_r_score=0
for r_state in range (40,3000):
    x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=r_state,test_size=.20)
    regr=linear_model.LinearRegression()
    regr.fit(x_train,y_train)
    y_pred=regr.predict(x_test)
    r_scr=r2_score(y_test,y_pred)
    if r_scr>max_r_score:
        max_r_score=r_scr
        final_r_state=r_state
print("max r2 score corresponding to ",final_r_state,"is",max_r_score)
```

```
max r2 score corresponding to  1007 is 0.6635535994155548
```

In [49]:

```
from sklearn.linear_model import LinearRegression,Lasso,Ridge,ElasticNet
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
```

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings('ignore')
```

In [51]:

```
model=[LinearRegression(),DecisionTreeRegressor(),KNeighborsRegressor(),SVR(),Lasso(),Ridge(),Elast
icNet()]
for m in model:
    x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=1007,test_size=.20)
    m.fit(x_train,y_train)
    print('Score of',m,'is:',m.score(x_train,y_train))
    predm=m.predict(x_test)
    print('Error:')
    print('Mean Absolute Error :',mean_absolute_error(y_test,predm))
    print('Mean Squared Error :',mean_squared_error(y_test,predm))
    print('r2_score',r2_score(y_test,predm))

print('*************************************************************************************************
*******')
    print('\n')
```

```
Score of LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False) is: 0.600
214607780897
Error:
Mean Absolute Error : 0.4508819407140201
Mean Squared Error : 0.3247586052585833
r2_score 0.6635535994155548
*******************************************************************************************************



Score of DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      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, presort='deprecated',
                      random_state=None, splitter='best') is: 0.9999999999632032
Error:
Mean Absolute Error : 0.5884126471575293
Mean Squared Error : 0.5707892595472069
r2_score 0.4086685040600848
*******************************************************************************************************



Score of KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform') is: 0.7533811194825271
Error:
Mean Absolute Error : 0.43385052978471444
Mean Squared Error : 0.31178630884849357
r2_score 0.6769927581131789
*******************************************************************************************************



Score of SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False) is: 0.7232454766131324
Error:
Mean Absolute Error : 0.3854120441409365
Mean Squared Error : 0.26050879263308246
r2_score 0.730115709998777
*******************************************************************************************************



Score of Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False) is: 0.0
Error:
Mean Absolute Error : 0.7948631810569567
```

```
Mean Squared Error : 0.9656995350682439
r2_score -0.00045426775091050864
************************************************************************************************



Score of Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001) is: 0.6002145775289626
Error:
Mean Absolute Error : 0.4508975355779774
Mean Squared Error : 0.3247723336567663
r2_score 0.6635393769436028
************************************************************************************************



Score of ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False) is: 0.0136490887641
72033
Error:
Mean Absolute Error : 0.7901526790960364
Mean Squared Error : 0.9522511621278656
r2_score 0.013478101080101279
************************************************************************************************
```

In [50]:

```python
from sklearn.model_selection import cross_val_score
for m in model:
    score=cross_val_score(m,x,y,cv=5,scoring='r2')
    print('Score of',m,'is:',score)
    print('Mean score:',score.mean())
    print('Standard deviation:',score.std())

print('*************************************************************************************************
*******')
    print('\n')
```

```
Score of LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False) is: [0.61
996276 0.61721137 0.5959433  0.6043708  0.62295103]
Mean score: 0.6120878533559326
Standard deviation: 0.010261312117869494
************************************************************************************************



Score of DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=None,
                      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, presort='deprecated',
                      random_state=None, splitter='best') is: [0.39766214 0.38597254 0.43708902 0.3
6506107 0.46837009]
Mean score: 0.41083097007516045
Standard deviation: 0.037111936346064814
************************************************************************************************



Score of KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform') is: [0.62508417 0.62123307 0.63672728 0.6221901  0.66677253]
Mean score: 0.6344014303310026
Standard deviation: 0.01710339692739742
************************************************************************************************



Score of SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False) is: [0.69155112
0.69128264 0.69045974 0.6902548  0.7212101 ]
```

```
Mean score: 0.6969516795613064
Standard deviation: 0.012138934896551306
*****************************************************************************************************



Score of Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False) is: [-6.48365130e-05 -1.14887371e-03 -1.54
616990e-03 -1.39153574e-04
 -5.70222568e-04]
Mean score: -0.0006938512524821139
Standard deviation: 0.0005748261686497952
*****************************************************************************************************



Score of Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001) is: [0.61995312 0.61721265
0.59595595 0.60437558 0.62294499]
Mean score: 0.6120884577627396
Standard deviation: 0.010253982393208405
*****************************************************************************************************



Score of ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=False,
           random_state=None, selection='cyclic', tol=0.0001, warm_start=False) is: [0.02085523 0.0
1936004 0.02182611 0.01882532 0.0085976 ]
Mean score: 0.017892861016483685
Standard deviation: 0.004768018566463891
*****************************************************************************************************
```

In [52]:

```
import joblib
joblib.dump(DecisionTreeRegressor,'bigmartdata.pkl')
```

Out[52]:

```
['bigmartdata.pkl']
```

In [ ]:

```

```