Big Data Mart Sales Problem

Problem Statement:

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

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

In [1]:
```
#import the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
big_mart=pd.read_csv('bigmart_Train.csv')
big_mart.head()
```

Out[2]:

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

In [3]:
```
big_mart.tail()
```

Out[3]:

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

In [4]:
```
#check the row and columns Train data
big_mart.shape
```

Out[4]:
```
(8523, 12)
```

In [5]:
```
#show the columns only
big_mart.columns
```

Out[5]:
```
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 [6]:
```
#informaton about the data set
big_mart.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]:
```
#check the missing values
big_mart.isnull().sum()
```

Out[7]:
```
Item_Identifier               0
Item_Weight                1463
Item_Fat_Content              0
Item_Visibility               0
Item_Type                     0
Item_MRP                      0
Outlet_Identifier             0
Outlet_Establishment_Year     0
Outlet_Size                2410
Outlet_Location_Type          0
Outlet_Type                   0
Item_Outlet_Sales             0
dtype: int64
```

Categorical Features:

- Item_Identifier
- Item_Fat_Content
- Item_Type
- Outlet_Identifier
- Outlet_Size
- Outlet_Location_Type
- Outlet_Type

Handling Missing Values

- Mean --> average
- Mode --> more repeated value

In [8]:
```
# mean value of "Item_Weight" column
big_mart['Item_Weight'].mean()
```

Out[8]:
```
12.857645184136183
```

In [9]:
```
# filling the missing values in "Item_weight column" with "Mean" value
big_mart['Item_Weight'].fillna(big_mart['Item_Weight'].mean(), inplace=True)
```

In [10]:
```
# mode of "Outlet_Size" column
big_mart['Outlet_Size'].mode()
```

Out[10]:
```
0    Medium
Name: Outlet_Size, dtype: object
```

In [11]:
```
# filling the missing values in "Outlet_Size" column with Mode
mode_of_Outlet_size = big_mart.pivot_table(values='Outlet_Size', columns='Outlet_Type'
```

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

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

Outlet_Type Supermarket Type3
Outlet_Size            Medium
```

In [13]:
```
miss_values = big_mart['Outlet_Size'].isnull()
```

In [14]:
```
print(miss_values)
```

```
0       False
1       False
2       False
3        True
4       False
        ...
8518    False
8519     True
8520    False
8521    False
8522    False
Name: Outlet_Size, Length: 8523, dtype: bool
```

In [15]:
```
big_mart.loc[miss_values, 'Outlet_Size'] = big_mart.loc[miss_values,'Outlet_Type'].app
```

In [16]:
```
# checking for missing values
big_mart.isnull().sum()
```

Out[16]:
```
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
```

Here no missing values found

Data Analysis

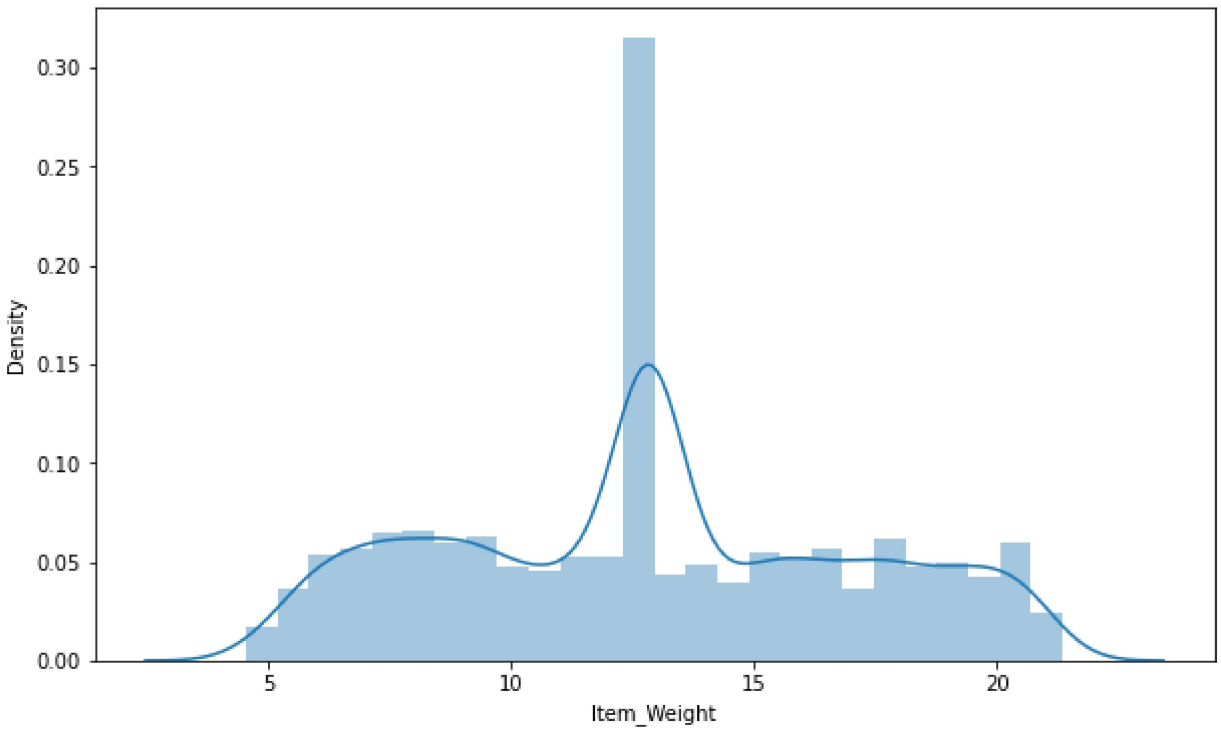
In [17]:
```
big_mart.describe()
```

Out[17]:

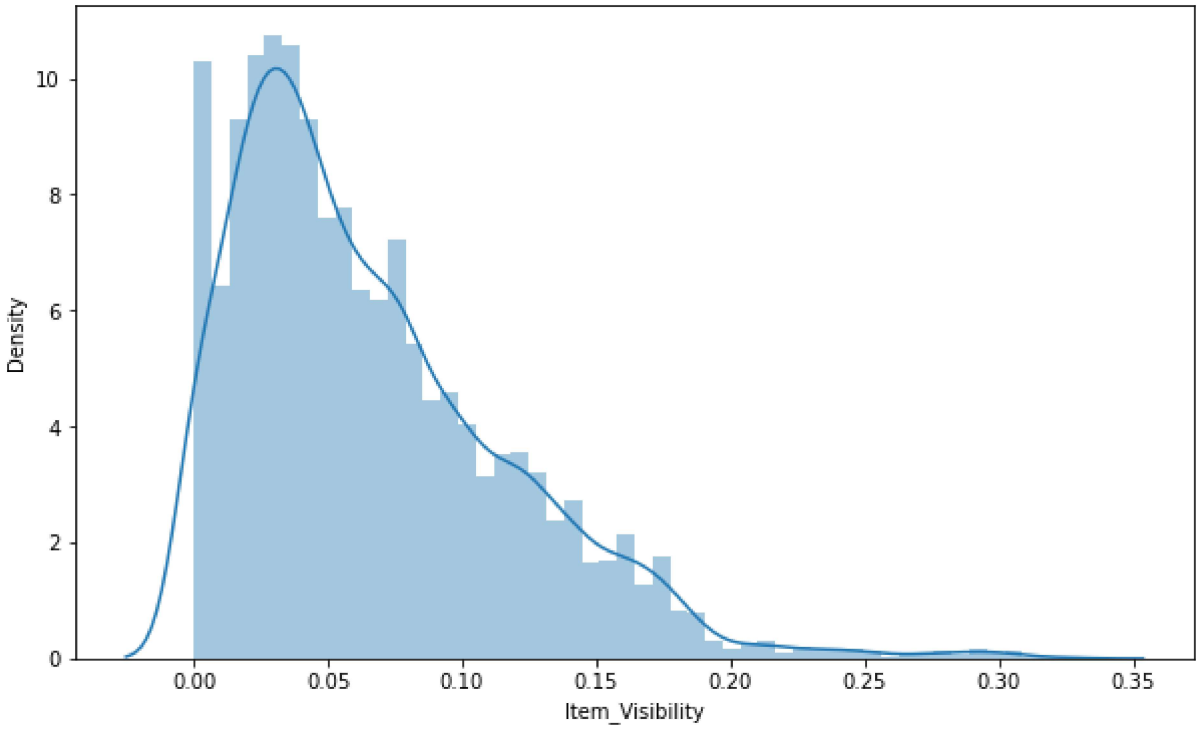
	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
count	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000
mean	12.857645	0.066132	140.992782	1997.831867	2181.288914
std	4.226124	0.051598	62.275067	8.371760	1706.499616
min	4.555000	0.000000	31.290000	1985.000000	33.290000
25%	9.310000	0.026989	93.826500	1987.000000	834.247400
50%	12.857645	0.053931	143.012800	1999.000000	1794.331000
75%	16.000000	0.094585	185.643700	2004.000000	3101.296400
max	21.350000	0.328391	266.888400	2009.000000	13086.964800

Data Visualization

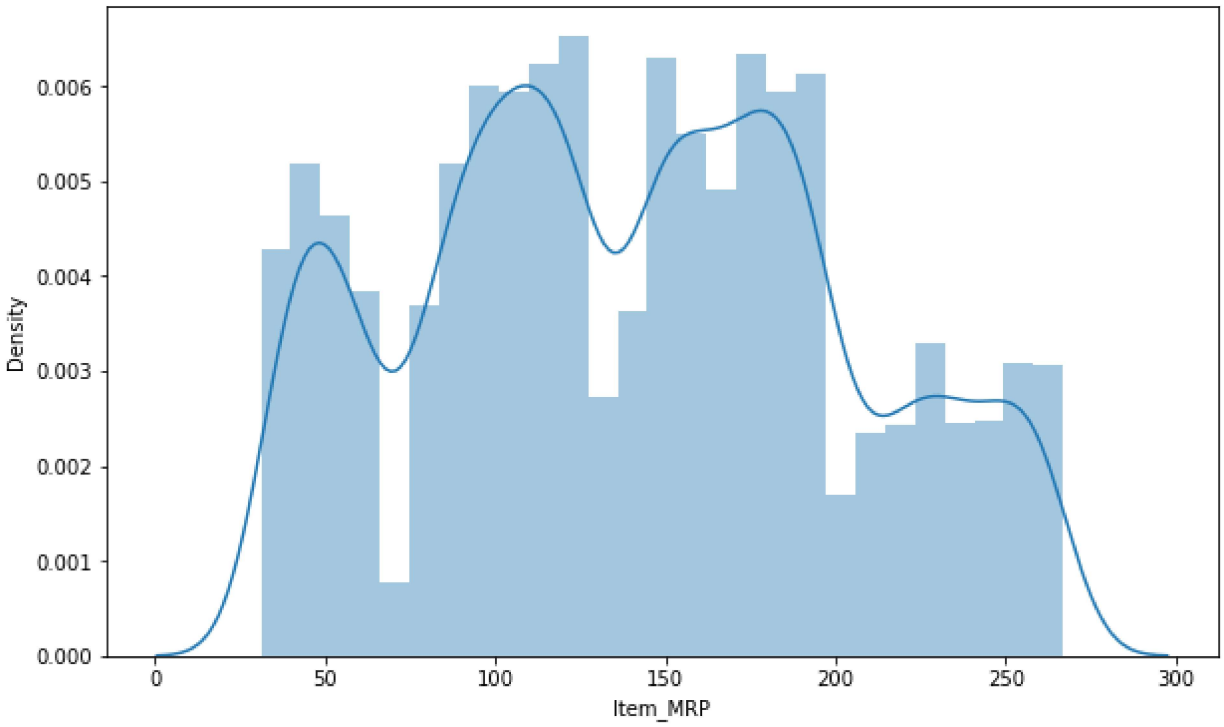
In [18]:
```
# Item_Weight distribution
plt.figure(figsize=(10,6))
sns.distplot(big_mart['Item_Weight'])
plt.show()
```



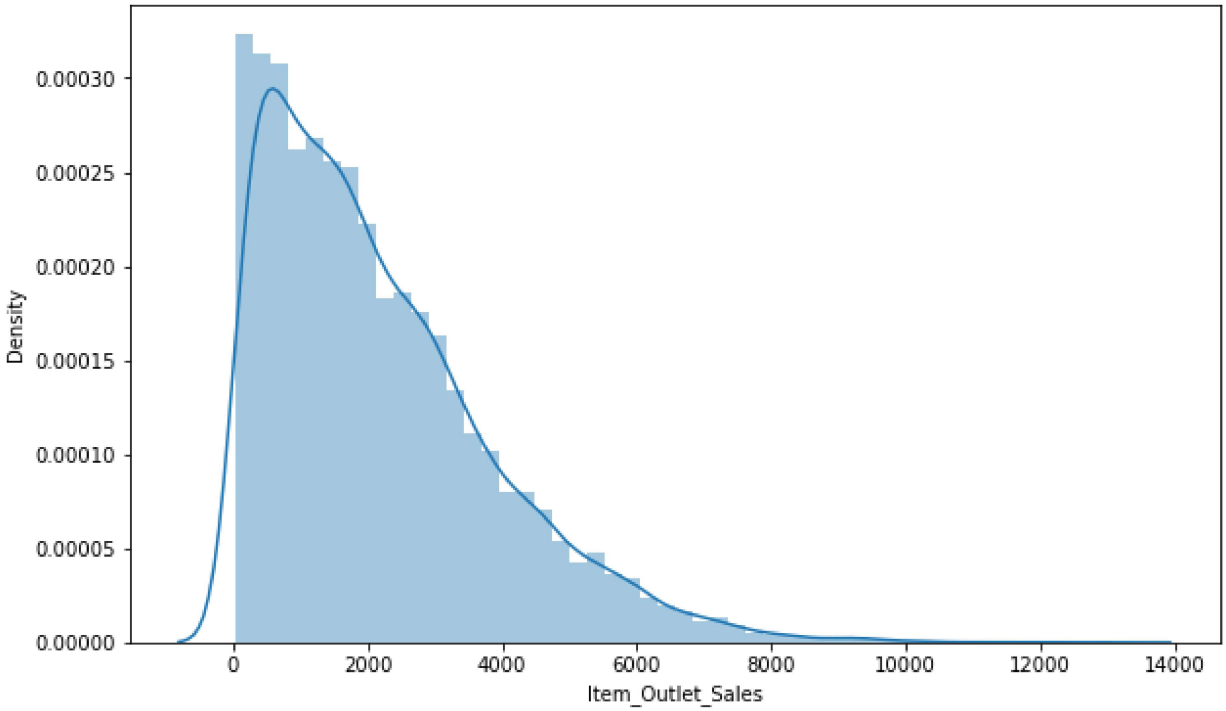
In [19]:
```
# Item Visibility distribution
plt.figure(figsize=(10,6))
sns.distplot(big_mart['Item_Visibility'])
plt.show()
```



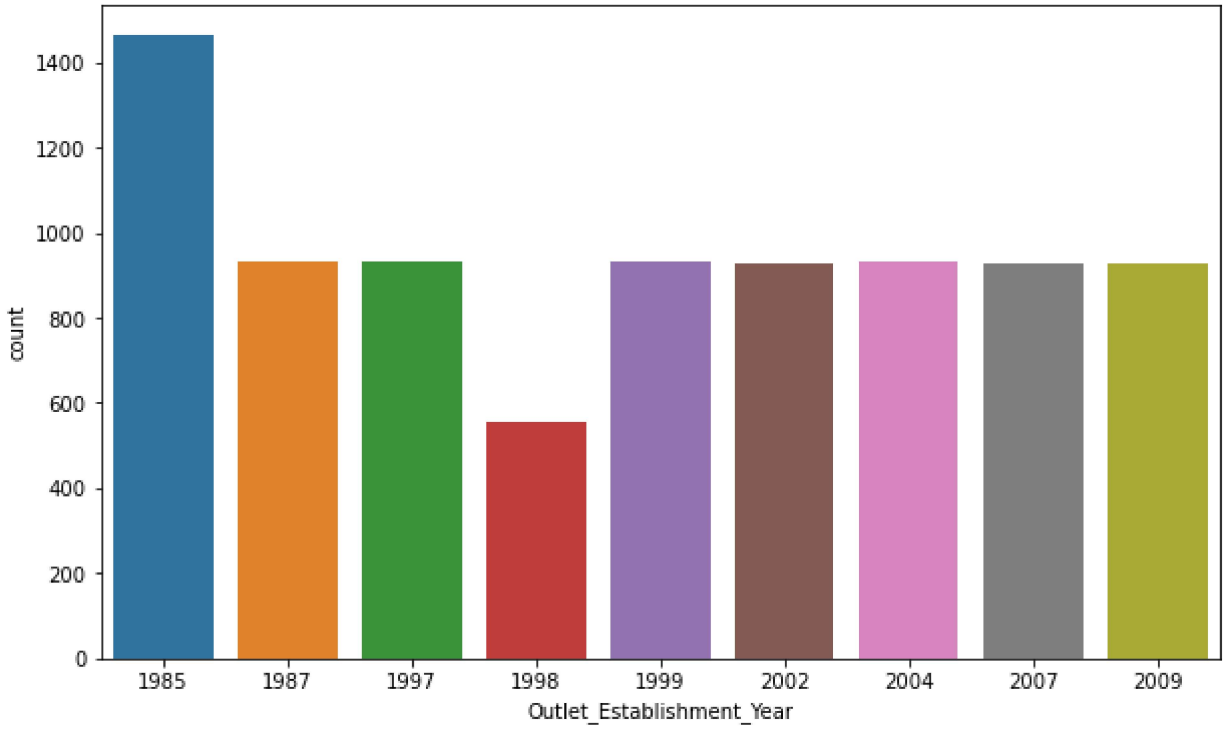
In [20]:
```
# Item MRP distribution
plt.figure(figsize=(10,6))
sns.distplot(big_mart['Item_MRP'])
plt.show()
```



In [21]:
```
# Item_Outlet_Sales distribution
plt.figure(figsize=(10,6))
sns.distplot(big_mart['Item_Outlet_Sales'])
plt.show()
```

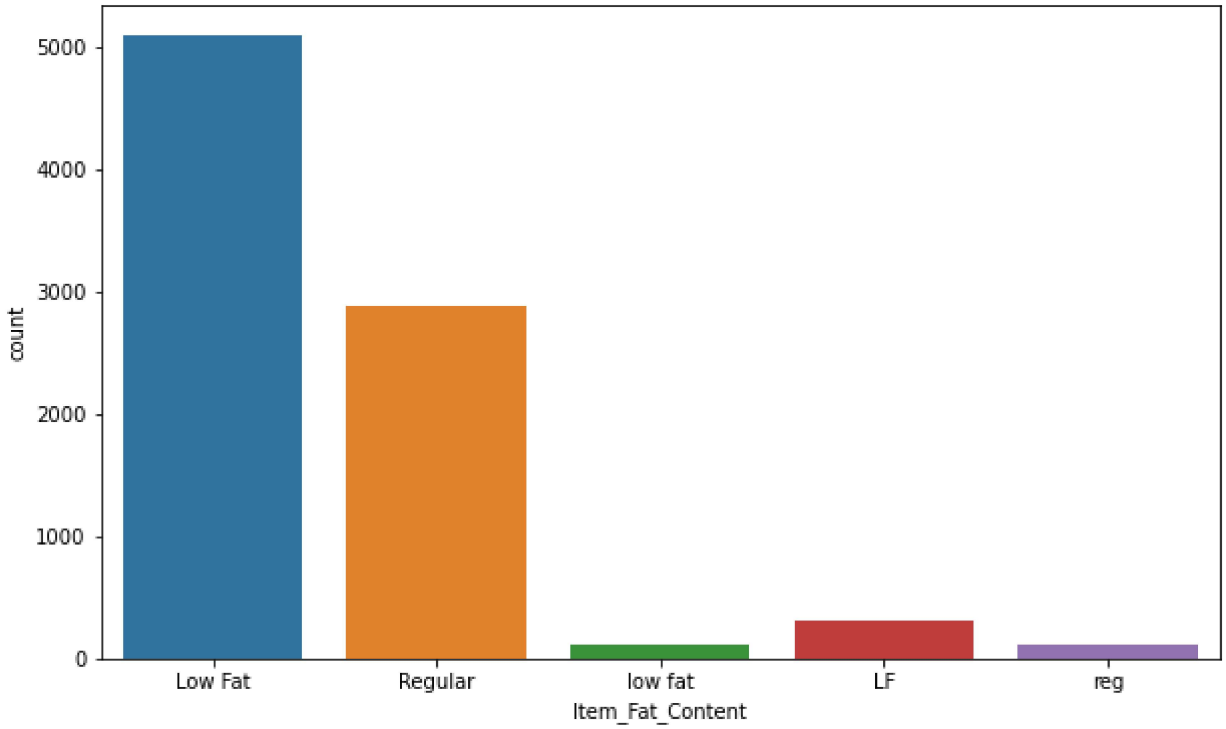


In [22]:
```
# Outlet_Establishment_Year column
plt.figure(figsize=(10,6))
sns.countplot(x='Outlet_Establishment_Year', data=big_mart)
plt.show()
```

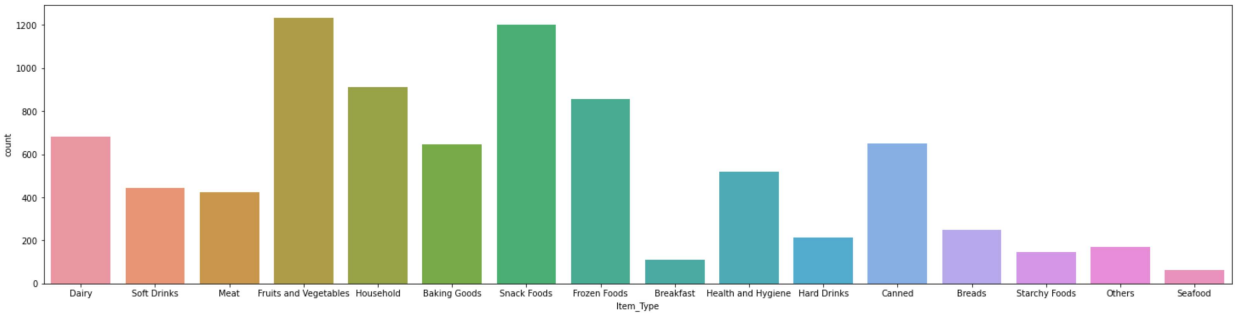


Categorical Features

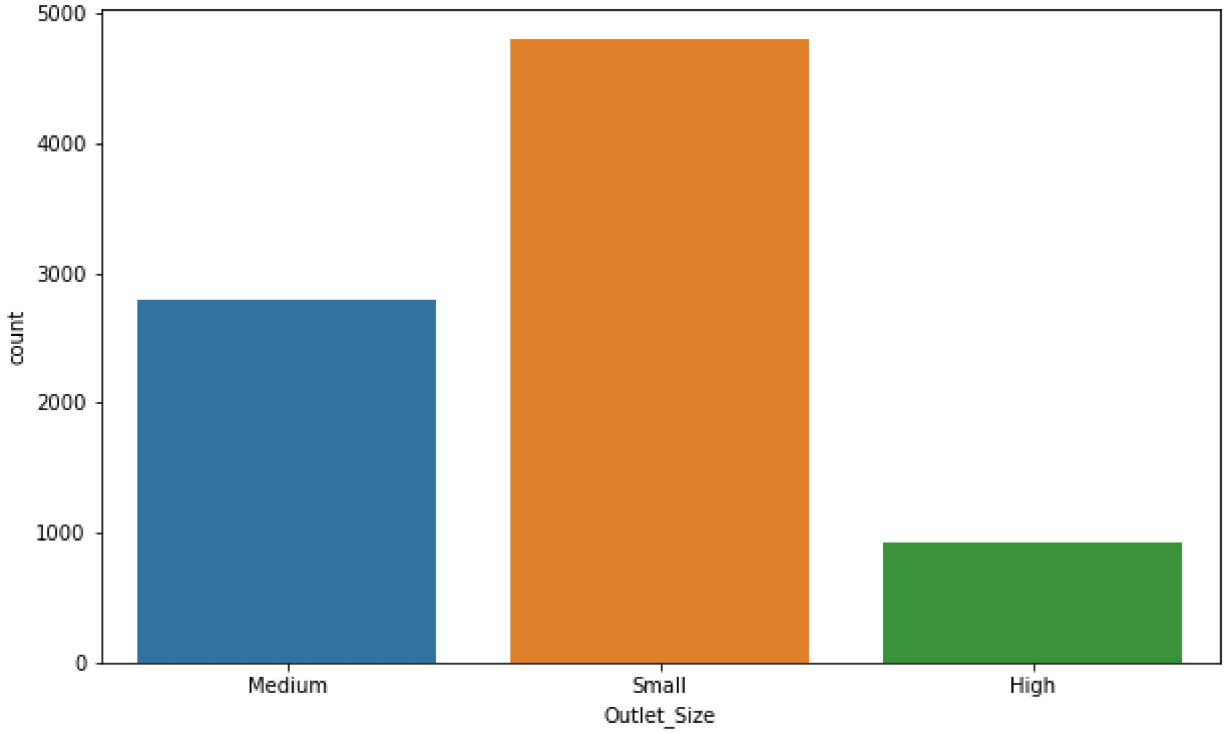
In [23]:
```
# Item_Fat_Content column
plt.figure(figsize=(10,6))
sns.countplot(x='Item_Fat_Content', data=big_mart)
plt.show()
```



In [24]:
```
# Item_Type column
plt.figure(figsize=(25,6))
sns.countplot(x='Item_Type', data=big_mart)
plt.show()
```



In [25]:
```
# Outlet_Size column
plt.figure(figsize=(10,6))
sns.countplot(x='Outlet_Size', data=big_mart)
plt.show()
```



Data Pre-Processing

In [26]:
```
big_mart.head()
```

Out[26]:

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

In [27]:
```
big_mart['Item_Fat_Content'].value_counts()
```

Out[27]:
```
Low Fat    5089
Regular    2889
LF          316
reg         117
low fat     112
Name: Item_Fat_Content, dtype: int64
```

In [28]:
```
#here replace the values
big_mart.replace({'Item_Fat_Content': {'low fat':'Low Fat','LF':'Low Fat', 'reg':'Regu
```

In [29]:
```
#show the replace values
big_mart['Item_Fat_Content'].value_counts()
```

Out[29]:
```
Low Fat    5517
Regular    3006
Name: Item_Fat_Content, dtype: int64
```

Label Encoding

In [30]:
```
encoder = LabelEncoder()
```

In [31]:
```
big_mart['Item_Identifier'] = encoder.fit_transform(big_mart['Item_Identifier'])

big_mart['Item_Fat_Content'] = encoder.fit_transform(big_mart['Item_Fat_Content'])

big_mart['Item_Type'] = encoder.fit_transform(big_mart['Item_Type'])

big_mart['Outlet_Identifier'] = encoder.fit_transform(big_mart['Outlet_Identifier'])

big_mart['Outlet_Size'] = encoder.fit_transform(big_mart['Outlet_Size'])

big_mart['Outlet_Location_Type'] = encoder.fit_transform(big_mart['Outlet_Location_Typ

big_mart['Outlet_Type'] = encoder.fit_transform(big_mart['Outlet_Type'])
```

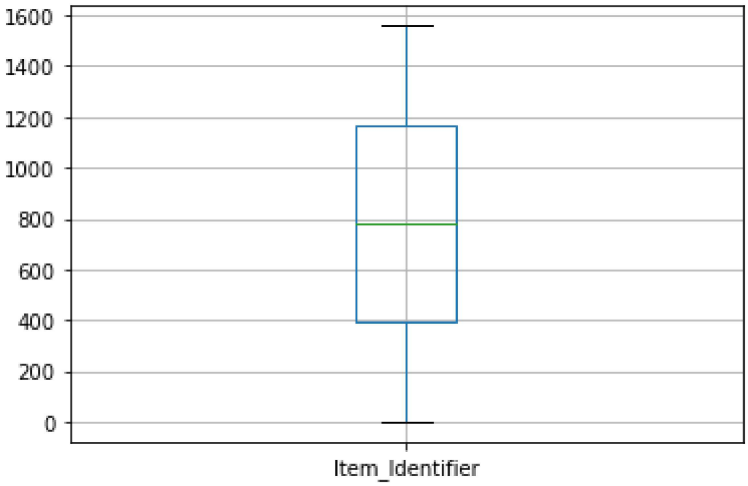
In [32]:
```
big_mart.head()
```

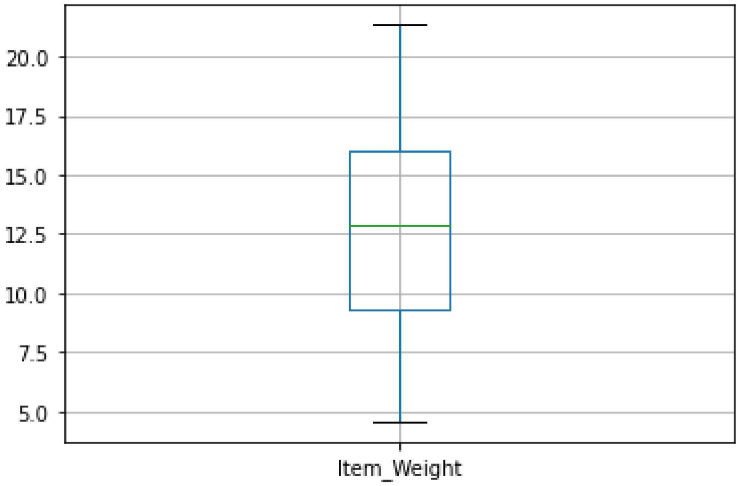
Out[32]:

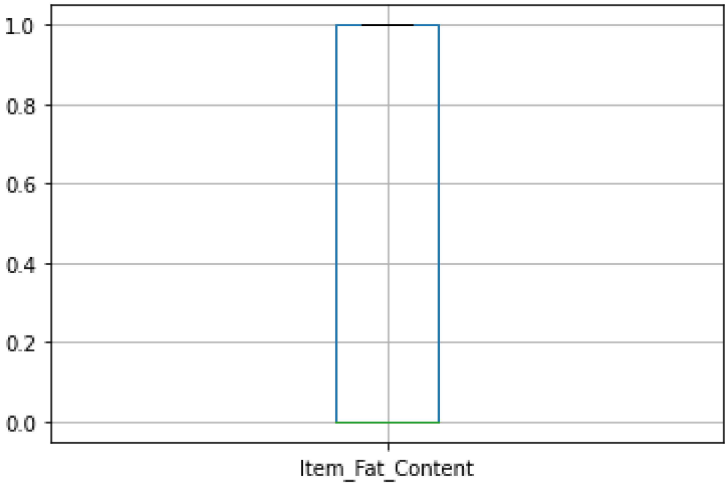
	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Ident
0	156	9.30	0	0.016047	4	249.8092	
1	8	5.92	1	0.019278	14	48.2692	
2	662	17.50	0	0.016760	10	141.6180	
3	1121	19.20	1	0.000000	6	182.0950	
4	1297	8.93	0	0.000000	9	53.8614	

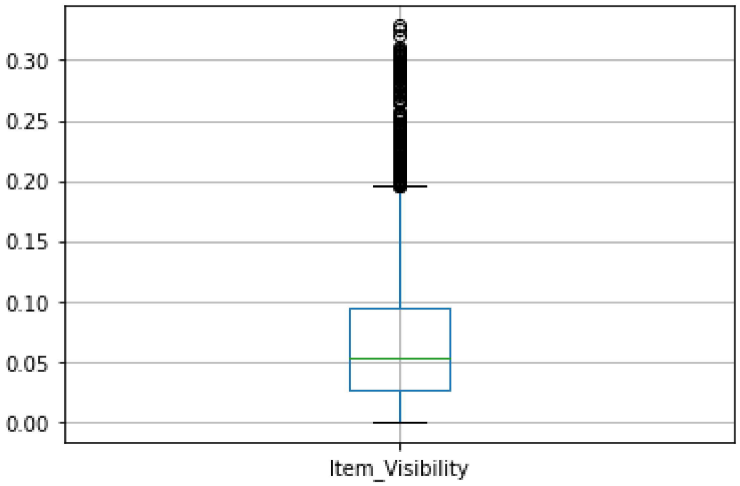
Outliers Detection

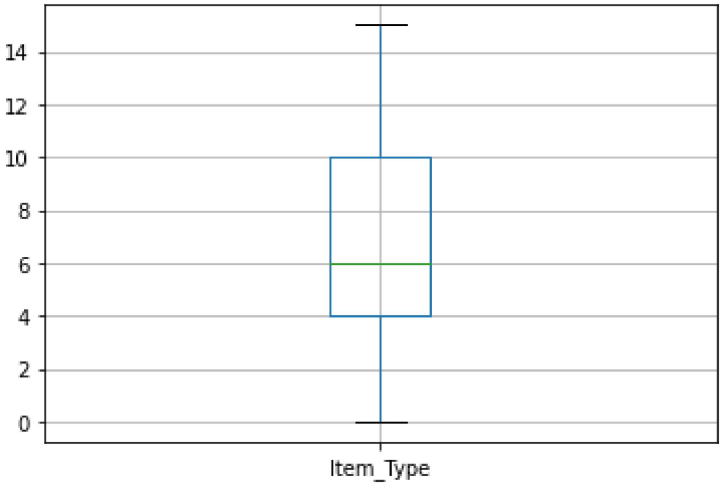
In [33]:
```
for column in big_mart:
    plt.figure()
    big_mart.boxplot([column])
```

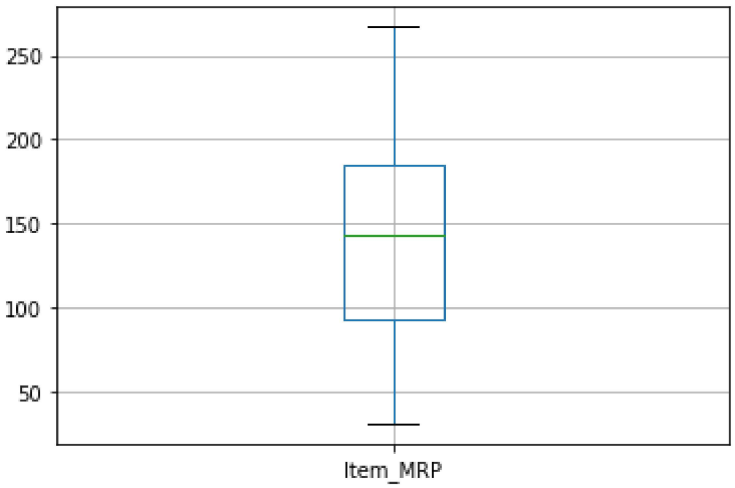

1600
1400
1200
1000
800
600
400
200
0
Item_Identifier

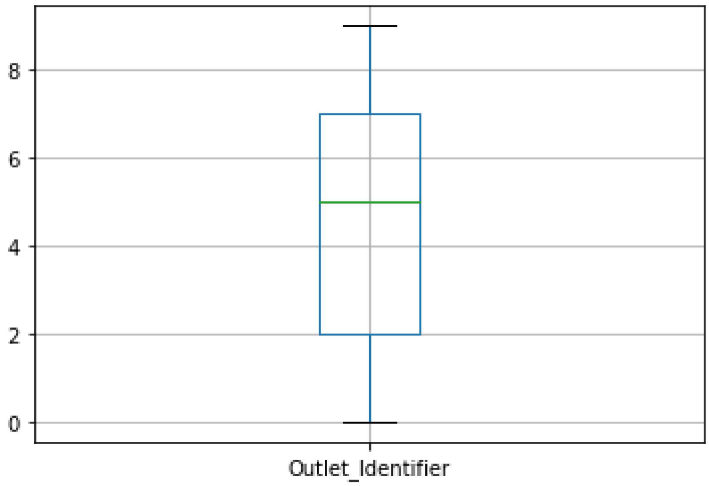

20.0
17.5
15.0
12.5
10.0
7.5
5.0
Item_Weight

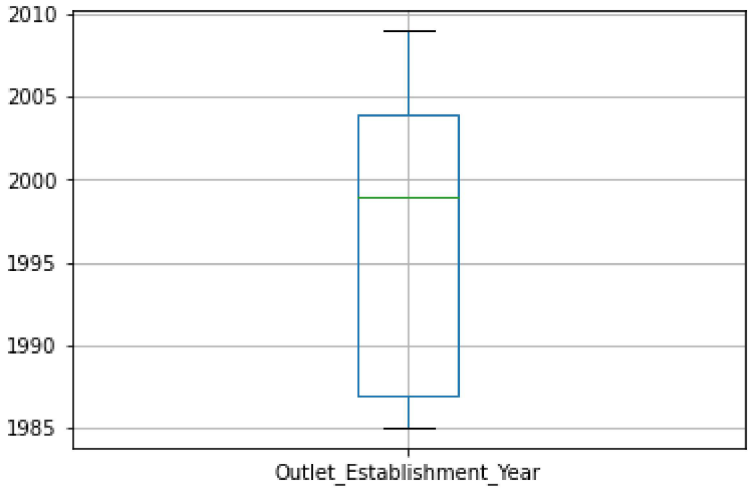

1.0
0.8
0.6
0.4
0.2
0.0
Item_Fat_Content

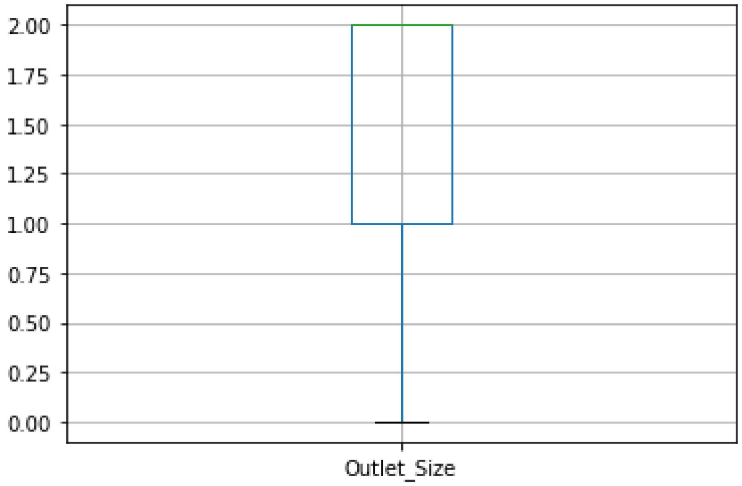

0.30
0.25
0.20
0.15
0.10
0.05
0.00
Item_Visibility

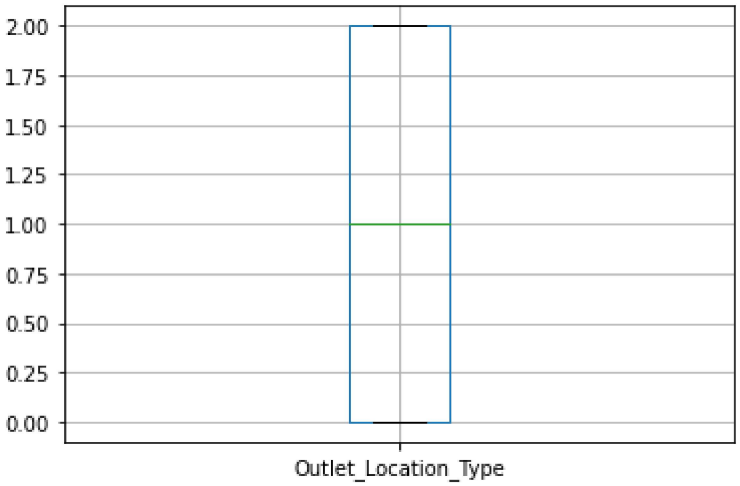

14
12
10
8
6
4
2
0
Item_Type

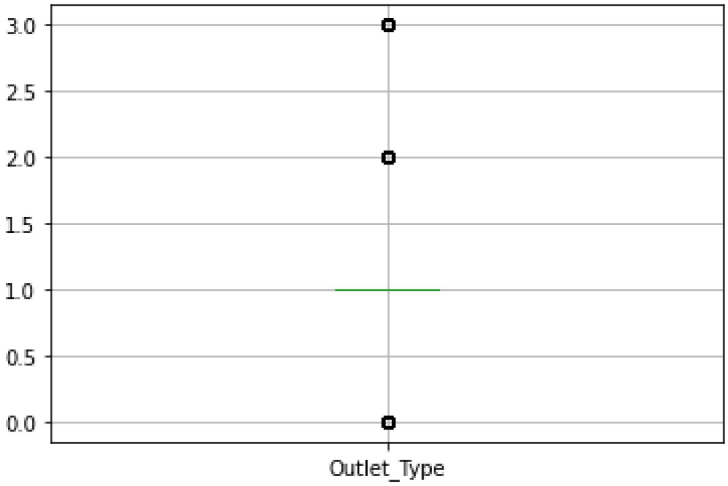

250
200
150
100
50
Item_MRP

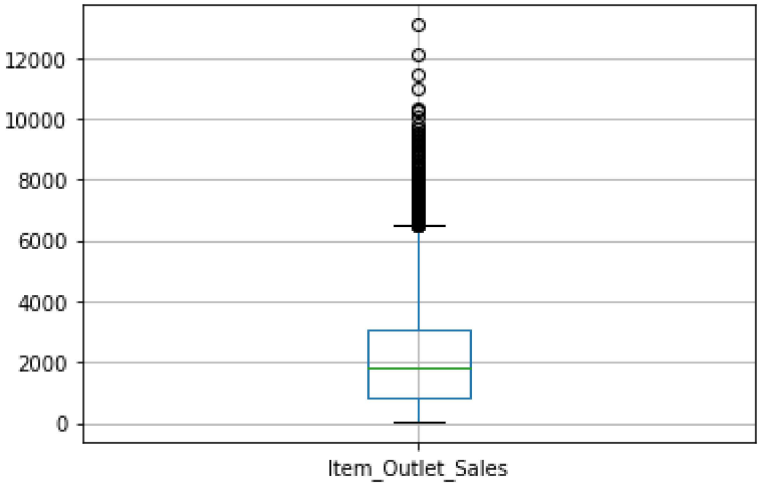

8
6
4
2
0
Outlet_Identifier


2010
2005
2000
1995
1990
1985
Outlet_Establishment_Year


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Outlet_Size







Removal Outliers

In [34]:
```
from scipy.stats import zscore
z=np.abs(zscore(big_mart))
z.head()
```

Out[34]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Ident
0	1.388514	0.841872	0.738147	0.970732	0.766479	1.747454	1.507
1	1.717991	1.641706	1.354743	0.908111	1.608963	1.489023	0.607
2	0.262057	1.098554	0.738147	0.956917	0.658786	0.010040	1.507
3	0.759769	1.500838	1.354743	1.281758	0.291391	0.660050	1.664
4	1.151580	0.929428	0.738147	1.281758	0.421242	1.399220	1.312

In [35]:
```
threshold=3
print(np.where(z>3))
```

```
(array([  49,   83,  108,  130,  145,  174,  276,  304,  333,  373,  434,
        456,  472,  502,  521,  532,  661,  694,  753,  809,  854,  935,
        991,  997, 1009, 1138, 1159, 1254, 1272, 1291, 1311, 1324, 1434,
       1450, 1496, 1501, 1506, 1560, 1575, 1644, 1651, 1805, 1827, 1841,
       1859, 1998, 2088, 2109, 2122, 2216, 2238, 2251, 2293, 2324, 2439,
       2445, 2516, 2535, 2586, 2613, 2728, 2776, 2855, 2895, 3017, 3083,
       3087, 3098, 3154, 3206, 3234, 3286, 3399, 3407, 3458, 3474, 3488,
       3497, 3540, 3667, 3709, 3720, 3750, 3757, 3767, 3884, 3993, 4006,
       4219, 4289, 4346, 4349, 4382, 4408, 4530, 4656, 4674, 4676, 4681,
       4772, 4789, 4888, 4941, 4961, 4991, 5031, 5065, 5150, 5177, 5185,
       5223, 5287, 5315, 5374, 5500, 5550, 5619, 5732, 5734, 5752, 5795,
       5799, 5812, 5837, 5880, 5891, 5946, 5977, 6000, 6008, 6012, 6134,
       6179, 6334, 6409, 6536, 6537, 6541, 6576, 6606, 6622, 6674, 6786,
       6886, 6903, 6993, 7005, 7016, 7076, 7088, 7107, 7121, 7188, 7191,
       7215, 7236, 7250, 7256, 7278, 7302, 7343, 7388, 7499, 7558, 7560,
       7562, 7603, 7609, 7690, 7744, 7752, 7816, 7826, 7866, 7930, 7949,
       8039, 8194, 8201, 8215, 8245, 8273, 8345, 8432, 8447], dtype=int64), array([
3,  3,  3, 11, 11,  3, 11, 11, 11, 11,  3, 11, 11,  3,  3,  3, 11,
       11, 11, 11,  3, 11, 11, 11, 11, 11,  3, 11,  3,  3,  3,  3,  3, 11,
        3,  3, 11,  3,  3,  3,  3,  3,  3,  3,  3, 11, 11,  3, 11,  3, 11, 11,
        3, 11,  3,  3,  3, 11, 11,  3,  3,  3, 11,  3,  3,  3, 11, 11, 11,
       11,  3, 11, 11,  3, 11,  3,  3,  3,  3,  3, 11, 11, 11,  3, 11,  3,
        3,  3,  3,  3, 11, 11, 11,  3,  3,  3,  3,  3, 11, 11, 11,  3, 11,
        3, 11, 11,  3, 11,  3, 11, 11, 11,  3, 11,  3, 11, 11, 11,  3,  3,
       11,  3, 11, 11,  3,  3,  3,  3, 11, 11,  3,  3, 11,  3, 11, 11,  3,
       11, 11,  3, 11,  3,  3,  3, 11,  3, 11,  3, 11, 11,  3,  3,  3, 11,
       11,  3, 11,  3, 11,  3, 11, 11,  3,  3,  3, 11, 11,  3, 11, 11,  3,
       11, 11, 11,  3, 11,  3, 11,  3, 11,  3, 11,  3,  3,  3, 11],
      dtype=int64))
```

In [36]:
```
Q1=big_mart.quantile(0.25)
Q2=big_mart.quantile(0.75)
```

In [37]:
```
Q1
```

Out[37]:
```
Item_Identifier               395.500000
Item_Weight                     9.310000
Item_Fat_Content                0.000000
Item_Visibility                 0.026989
Item_Type                       4.000000
Item_MRP                       93.826500
Outlet_Identifier               2.000000
Outlet_Establishment_Year    1987.000000
Outlet_Size                     1.000000
Outlet_Location_Type            0.000000
Outlet_Type                     1.000000
Item_Outlet_Sales             834.247400
Name: 0.25, dtype: float64
```

In [38]:
```
Q2
```

Out[38]:
```
Item_Identifier              1167.000000
Item_Weight                    16.000000
Item_Fat_Content                1.000000
Item_Visibility                 0.094585
Item_Type                      10.000000
Item_MRP                      185.643700
Outlet_Identifier               7.000000
Outlet_Establishment_Year    2004.000000
Outlet_Size                     2.000000
Outlet_Location_Type            2.000000
Outlet_Type                     1.000000
Item_Outlet_Sales            3101.296400
Name: 0.75, dtype: float64
```

In [39]:
```
IQR=Q2-Q1
IQR
```

Out[39]:
```
Item_Identifier               771.500000
Item_Weight                     6.690000
Item_Fat_Content                1.000000
Item_Visibility                 0.067596
Item_Type                       6.000000
Item_MRP                       91.817200
Outlet_Identifier               5.000000
Outlet_Establishment_Year      17.000000
Outlet_Size                     1.000000
Outlet_Location_Type            2.000000
Outlet_Type                     0.000000
Item_Outlet_Sales            2267.049000
dtype: float64
```

In [40]:
```
#Here remove the outliers
new_big_mart=big_mart[(z<3).all(axis=1)]
new_big_mart.head()
```

Out[40]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Ident
0	156	9.30	0	0.016047	4	249.8092	
1	8	5.92	1	0.019278	14	48.2692	
2	662	17.50	0	0.016760	10	141.6180	
3	1121	19.20	1	0.000000	6	182.0950	
4	1297	8.93	0	0.000000	9	53.8614	

In [41]:
```
#Check the old dataset
big_mart.shape
```

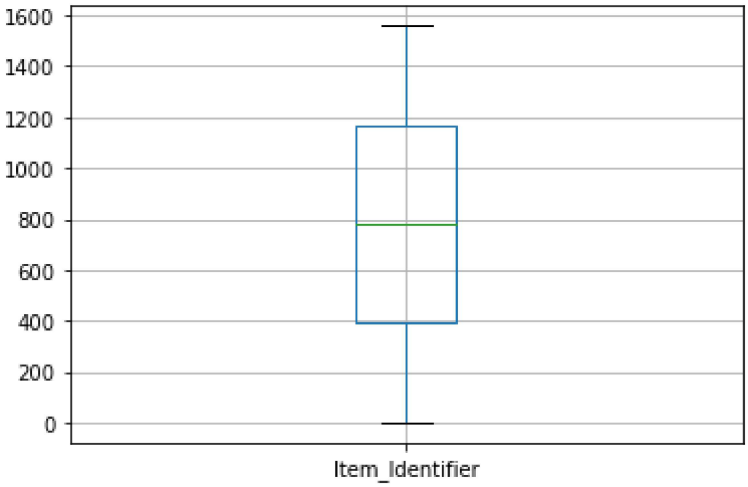
Out[41]:
```
(8523, 12)
```

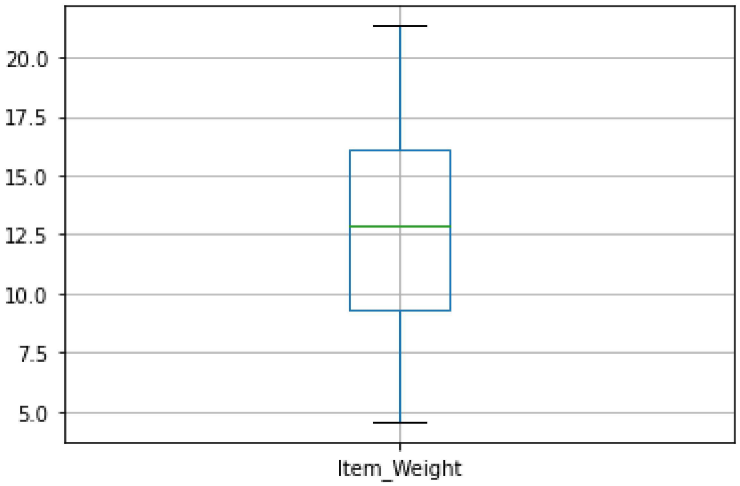
In [42]:
```
#After outliers removal datasets
new_big_mart.shape
```

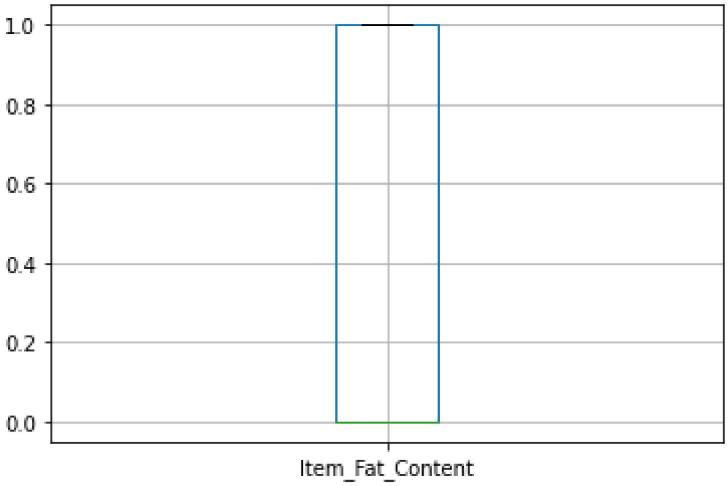
Out[42]:
```
(8338, 12)
```

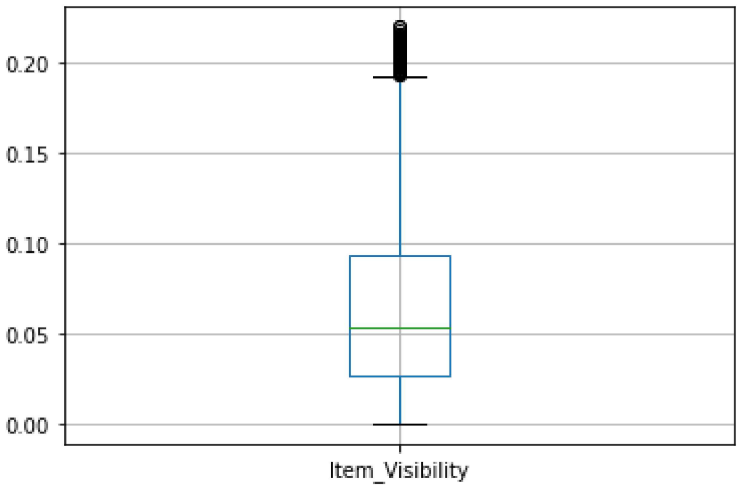
After removal outliers it will check again with help of boxplot

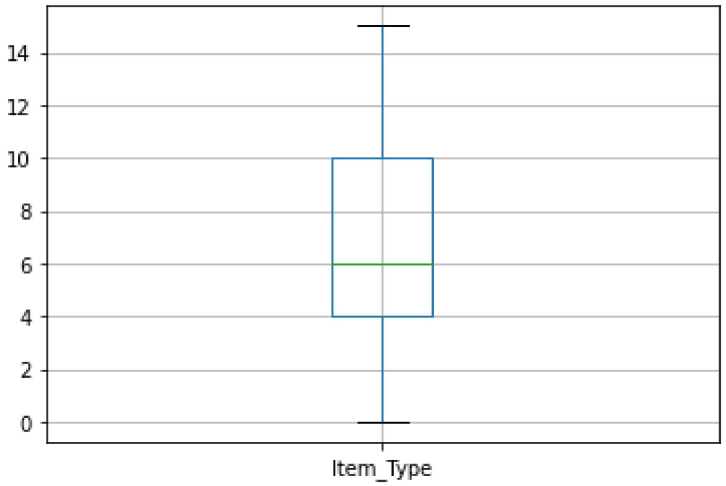
In [43]:
```
for column in new_big_mart:
    plt.figure()
    new_big_mart.boxplot([column])
```

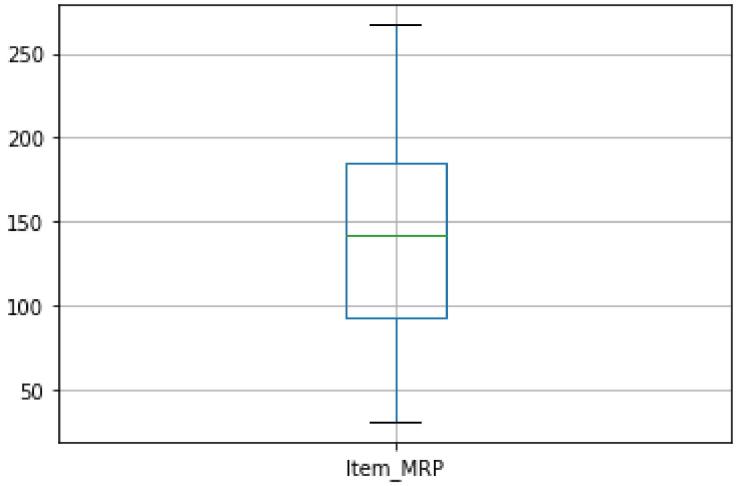


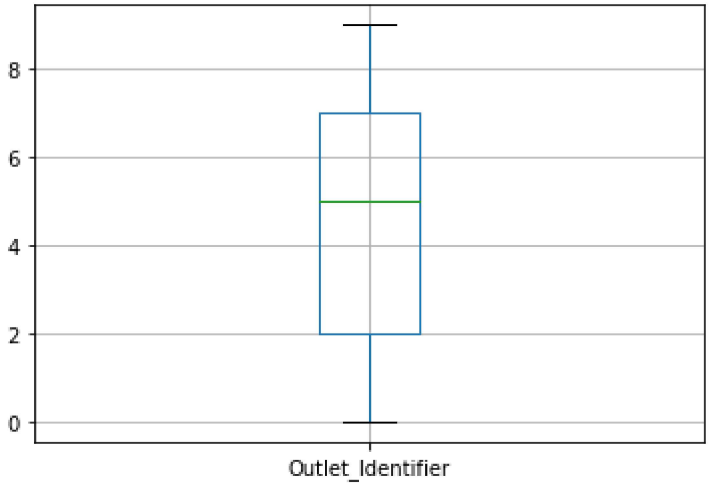

20.0
17.5
15.0
12.5
10.0
7.5
5.0
Item_Weight

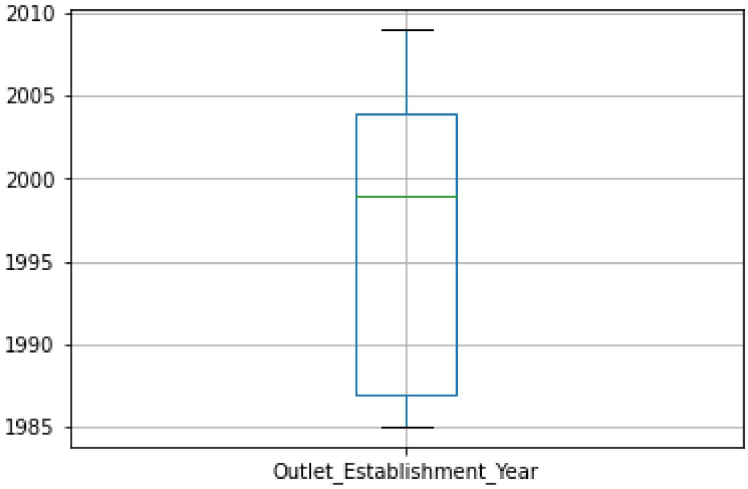

1.0
0.8
0.6
0.4
0.2
0.0
Item_Fat_Content

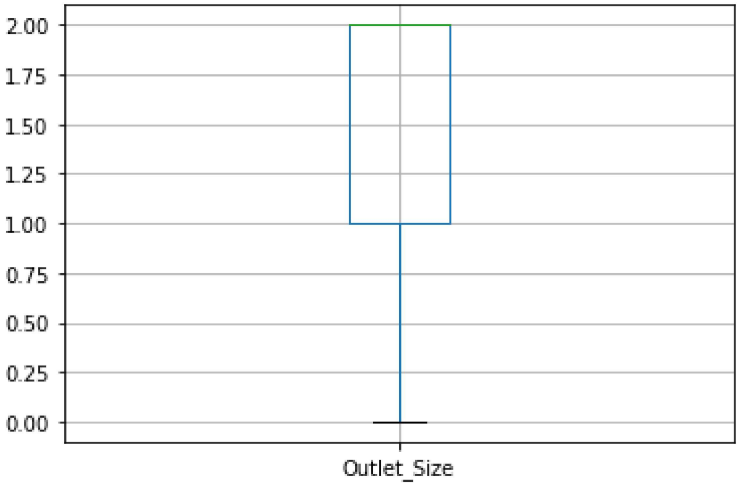

0.20
0.15
0.10
0.05
0.00
Item_Visibility

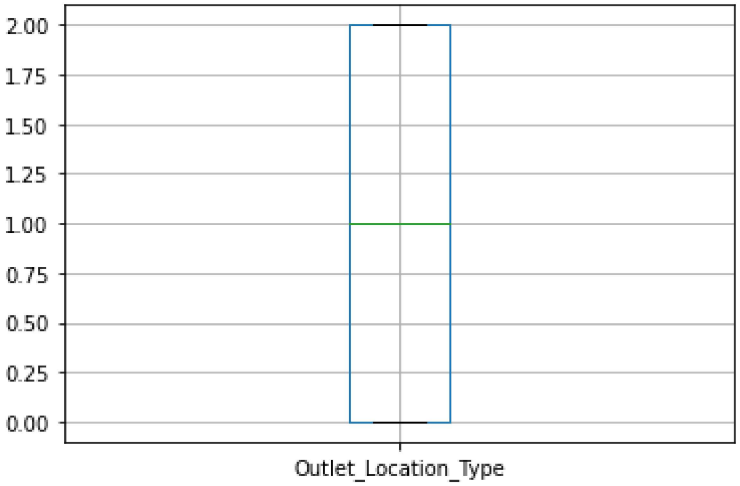

14
12
10
8
6
4
2
0
Item_Type

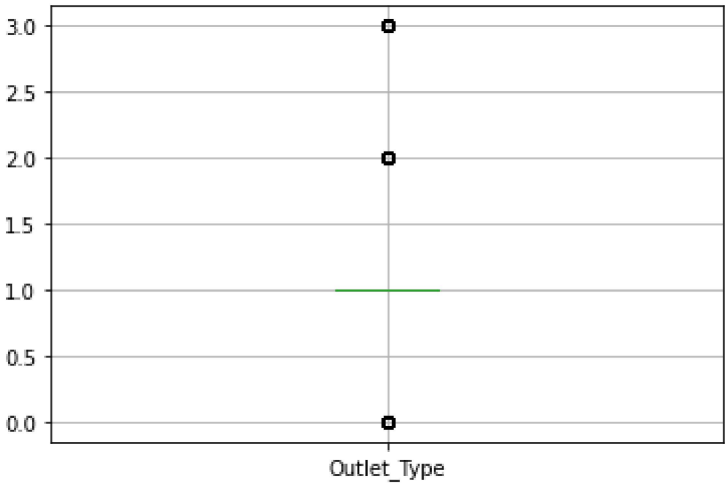

250
200
150
100
50
Item_MRP

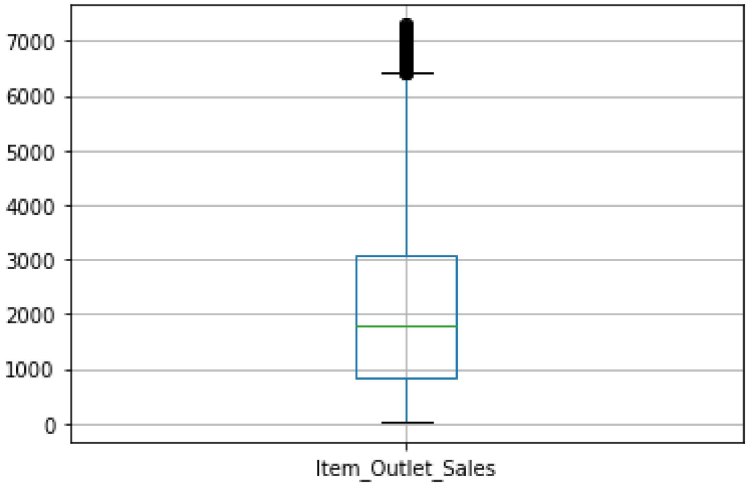

8
6
4
2
0
Outlet_Identifier


2010
2005
2000
1995
1990
1985
Outlet_Establishment_Year


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Outlet_Size


2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Outlet_Location_Type

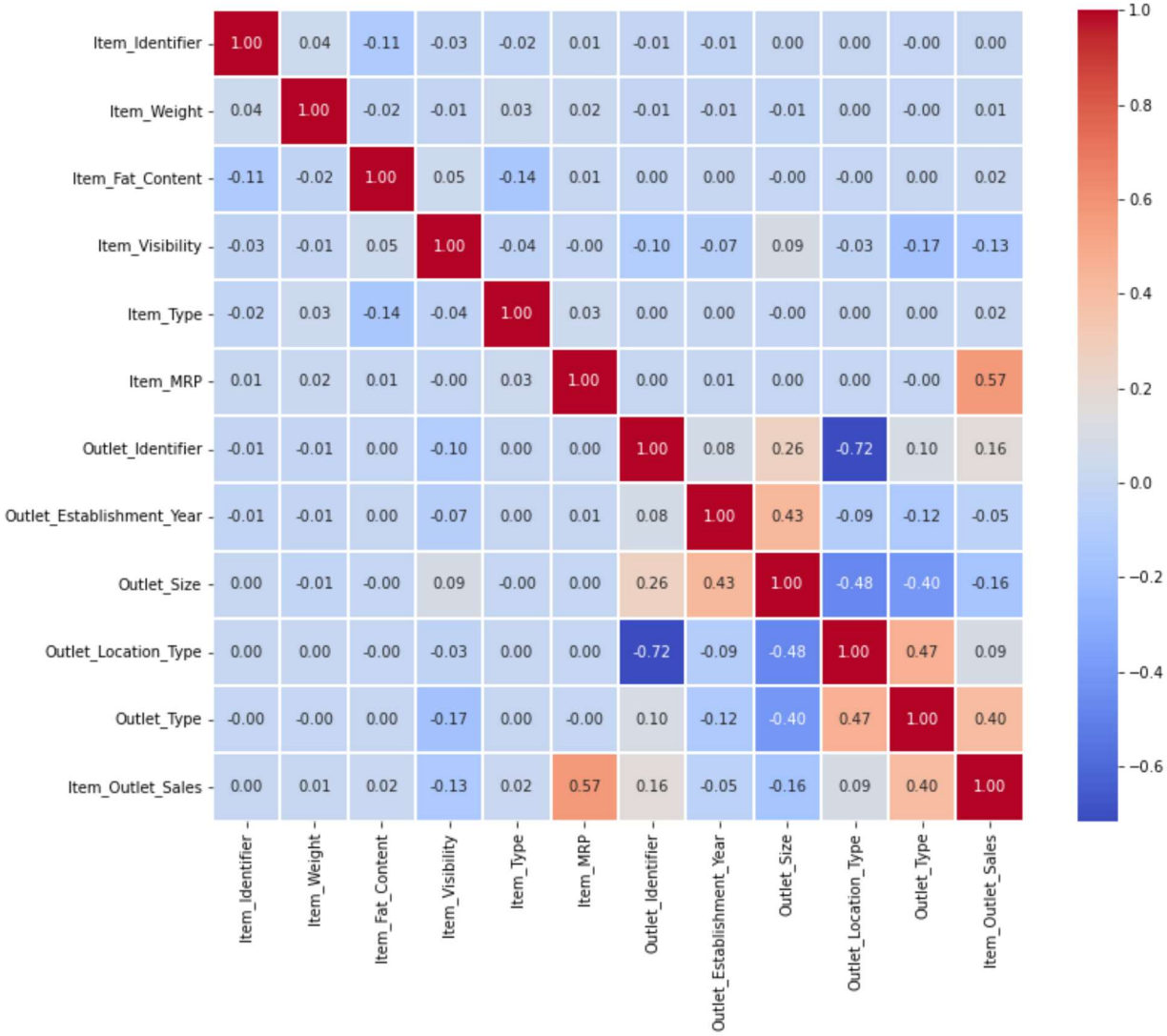




Correlation Matrix

In [44]:
```
plt.figure(figsize=(13,10))
sns.heatmap(big_mart.corr(),cbar=True,square=True,fmt='.2f',linewidths=0.1,annot=True,
```

Out[44]:
```
<AxesSubplot:>
```



Splitting features and Target

In [45]:
```
x = new_big_mart.drop(columns='Item_Outlet_Sales', axis=1)
y = new_big_mart['Item_Outlet_Sales']
```

In [46]:
```
print(x.shape)
print(y.shape)
```

```
(8338, 11)
(8338,)
```

Splitting the data into Training data & Testing Data

In [47]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=51)

print('shape of x_train =', x_train.shape)
print('shape of y_train =', y_train.shape)
print('shape of x_test =', x_test.shape)
print('shape of y_test =', y_test.shape)
```

```
shape of x_train = (6670, 11)
shape of y_train = (6670,)
shape of x_test = (1668, 11)
shape of y_test = (1668,)
```

Model Training

In [48]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x_train,y_train)
```

Out[48]:
```
LinearRegression()
```

Model Evaluation

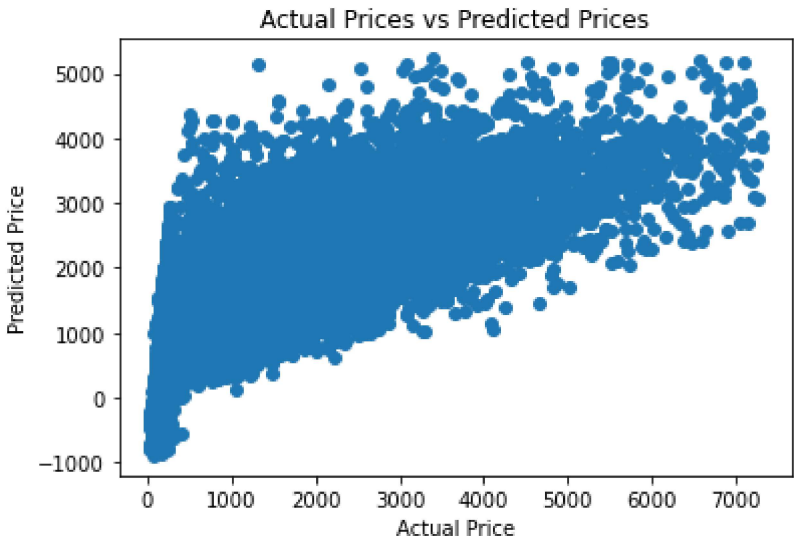
In [49]:
```
#Prediction training data
training_data_prediction=lr.predict(x_train)
```

In [50]:
```
#R squared Error
error_score=metrics.r2_score(y_train,training_data_prediction)
print("R squared Error:",error_score)
```

```
R squared Error: 0.4889683422090866
```

Visualize the actual price and predicted price

In [51]:
```
plt.scatter(y_train,training_data_prediction)
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual Prices vs Predicted Prices')
plt.show()
```



Lasso Regression

In [52]:
```
#Loading the Lasso Regression model
lss_reg_model=Lasso()
lss_reg_model.fit(x_train,y_train)
lss_reg_model.score(x_test,y_test)
```

Out[52]:
```
0.5031769927954819
```

conclusion

In [53]:
```
a=np.array(y_test)
predicted=np.array(lr.predict(x_test))
big_mart.com=pd.DataFrame({"original":a,"predicted":predicted},index=range(len(a)))
big_mart.com
```

Out[53]:

	original	predicted
0	2621.9204	2186.211944
1	978.7260	2811.944591
2	2021.3688	2924.683438
3	3142.5760	3218.805983
4	1651.1840	2830.749851
...	...	...
1663	2067.3090	3333.064071
1664	3124.5994	3388.136840
1665	2796.3600	2100.773140
1666	1286.3256	1129.123984
1667	905.4880	639.711215

1668 rows × 2 columns

In [ ]:

In [ ]: