In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [2]:
```
df_test=pd.read_csv('https://raw.githubusercontent.com/dsrscientist/bigdatamart_rep/mast
```

In [3]:
```
df_test
```

Out[3]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Ou
0	FDW58	20.750	Low Fat	0.007565	Snack Foods	107.8622	OUT049	
1	FDW14	8.300	reg	0.038428	Dairy	87.3198	OUT017	
2	NCN55	14.600	Low Fat	0.099575	Others	241.7538	OUT010	
3	FDQ58	7.315	Low Fat	0.015388	Snack Foods	155.0340	OUT017	
4	FDY38	NaN	Regular	0.118599	Dairy	234.2300	OUT027	
...	...	...	...	...	...	...	...	
5676	FDB58	10.500	Regular	0.013496	Snack Foods	141.3154	OUT046	
5677	FDD47	7.600	Regular	0.142991	Starchy Foods	169.1448	OUT018	
5678	NCO17	10.000	Low Fat	0.073529	Health and Hygiene	118.7440	OUT045	
5679	FDJ26	15.300	Regular	0.000000	Canned	214.6218	OUT017	
5680	FDU37	9.500	Regular	0.104720	Canned	79.7960	OUT045	

5681 rows × 11 columns

In [4]:
```
df_train=pd.read_csv('https://raw.githubusercontent.com/dsrscientist/bigdatamart_rep/mas
```

In [5]:
```
df_train
```

Out[5]:

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

8523 rows × 12 columns

In [6]:
```
df_train.head()
```

Out[6]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet
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 [7]:
```
df_train.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
```

In [8]:
```
df_train.shape
```

Out[8]:
```
(8523, 12)
```

In [9]:
```
df_test.isnull().sum()
```

Out[9]:
```
Item_Identifier               0
Item_Weight                 976
Item_Fat_Content              0
Item_Visibility               0
Item_Type                     0
Item_MRP                      0
Outlet_Identifier             0
Outlet_Establishment_Year     0
Outlet_Size                1606
Outlet_Location_Type          0
Outlet_Type                   0
dtype: int64
```

In [10]:
```
df_test.shape
```

Out[10]:
```
(5681, 11)
```

- Both the train and test dataset has the null values for the columns Item Weight and Outlet Size has null values

In [11]:
```
df_train.info()
```

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

In [12]:
```
df_train.describe()
```

Out[12]:

	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

Item_Weight is numerical column so we fill it with Mean Imputation

In [13]:
```
df_train['Item_Weight'].describe()
```

Out[13]:
```
count    7060.000000
mean       12.857645
std         4.643456
min         4.555000
25%         8.773750
50%        12.600000
75%        16.850000
max        21.350000
Name: Item_Weight, dtype: float64
```

In [14]:
```
df_train['Item_Weight'].fillna(df_train['Item_Weight'].mean(),inplace=True)
df_test['Item_Weight'].fillna(df_test['Item_Weight'].mean(),inplace=True)
```

In [15]:
```
df_train.isnull().sum()
```

Out[15]:
```
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                  2410
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales               0
dtype: int64
```

In [16]:
```
df_train['Item_Weight'].describe()
```

Out[16]:
```
count    8523.000000
mean       12.857645
std         4.226124
min         4.555000
25%         9.310000
50%        12.857645
75%        16.000000
max        21.350000
Name: Item_Weight, dtype: float64
```

In [17]:
```
df_test['Item_Weight'].describe()
```

Out[17]:
```
count    5681.000000
mean       12.695633
std         4.245189
min         4.555000
25%         9.195000
50%        12.695633
75%        15.850000
max        21.350000
Name: Item_Weight, dtype: float64
```

Outlet_Size is catagorical column so filling it with Mode Imputation

In [18]:
```
df_train['Outlet_Size'].mode()
```

Out[18]:
```
0    Medium
Name: Outlet_Size, dtype: object
```

In [19]:
```
df_test['Outlet_Size'].mode()
```

Out[19]:
```
0    Medium
Name: Outlet_Size, dtype: object
```

In [20]:
```
df_train['Outlet_Size'].fillna(df_train['Outlet_Size'].mode()[0],inplace=True)
df_test['Outlet_Size'].fillna(df_test['Outlet_Size'].mode()[0],inplace=True)
```

In [21]:
```
df_train.isnull().sum()
```

Out[21]:
```
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
```

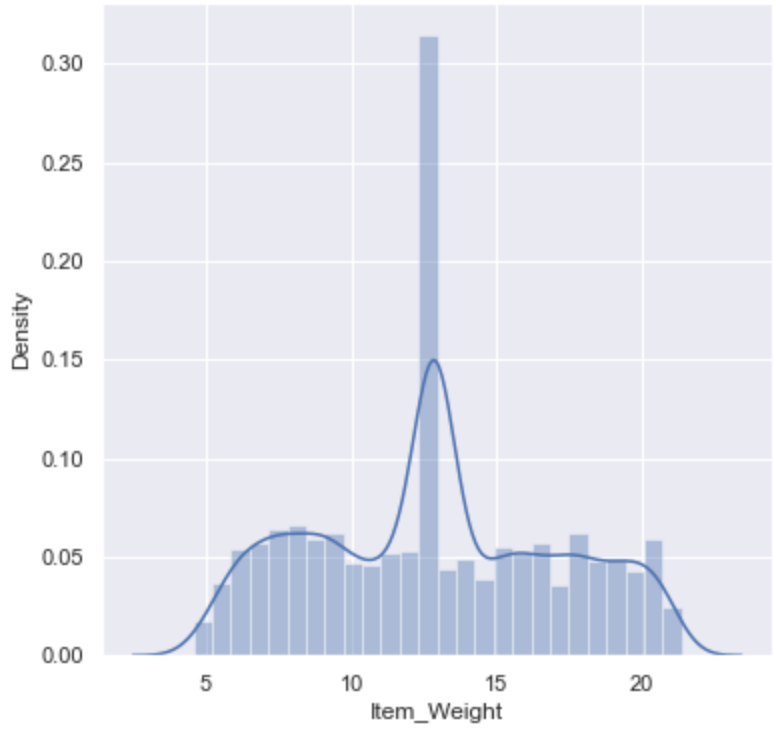
In [22]:
```
df_test.isnull().sum()
```

Out[22]:
```
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
dtype: int64
```

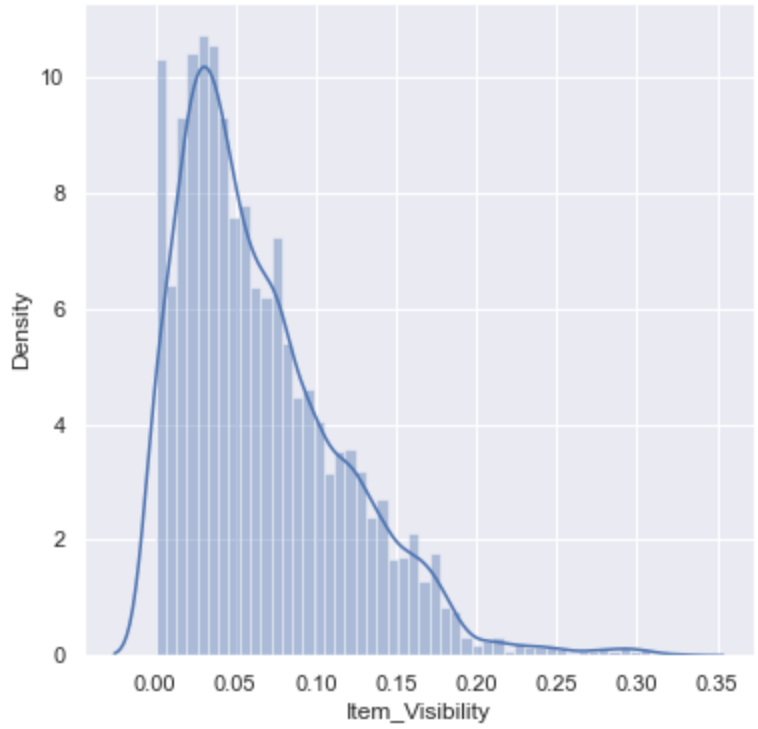
EDA for Training Data

In [23]:
```
sns.set()
# For Item_Weight
plt.figure(figsize=(6,6))
```

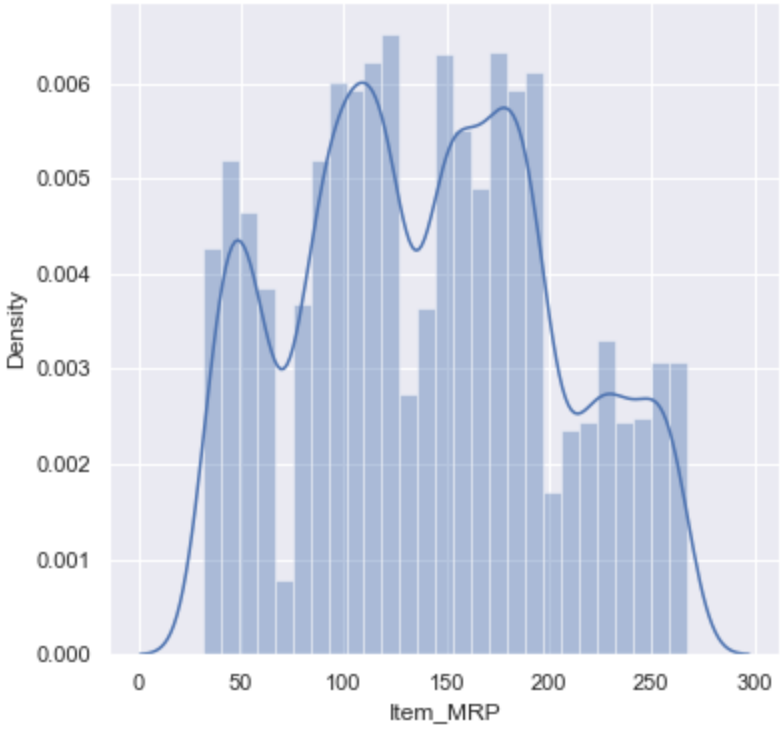
```
sns.distplot(df_train['Item_Weight'])
plt.show()
```



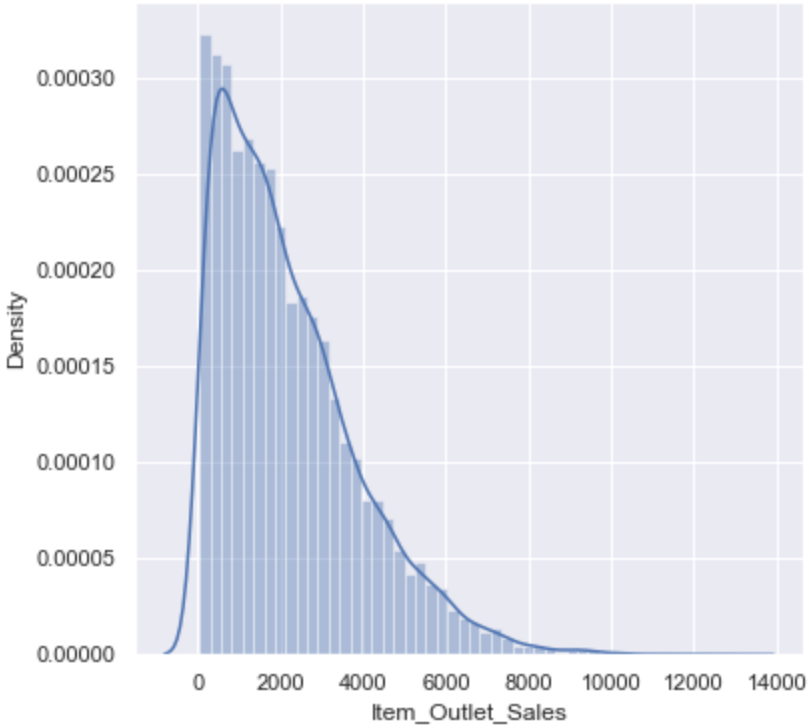
In [24]:
```
# Item Visibility
plt.figure(figsize=(6,6))
sns.distplot(df_train['Item_Visibility'])
plt.show()
```



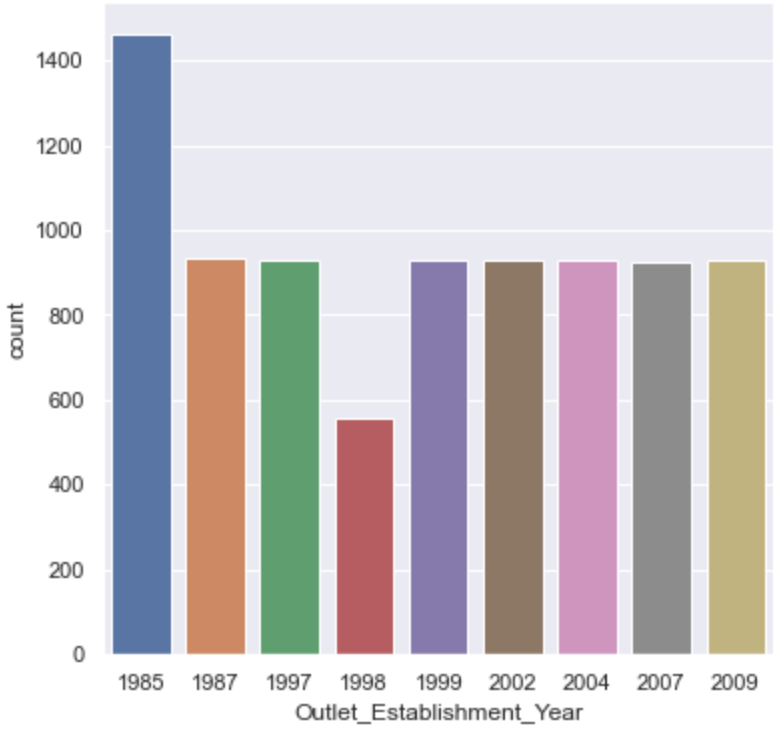
In [25]:
```
# Item MRP
plt.figure(figsize=(6,6))
sns.distplot(df_train['Item_MRP'])
plt.show()
```



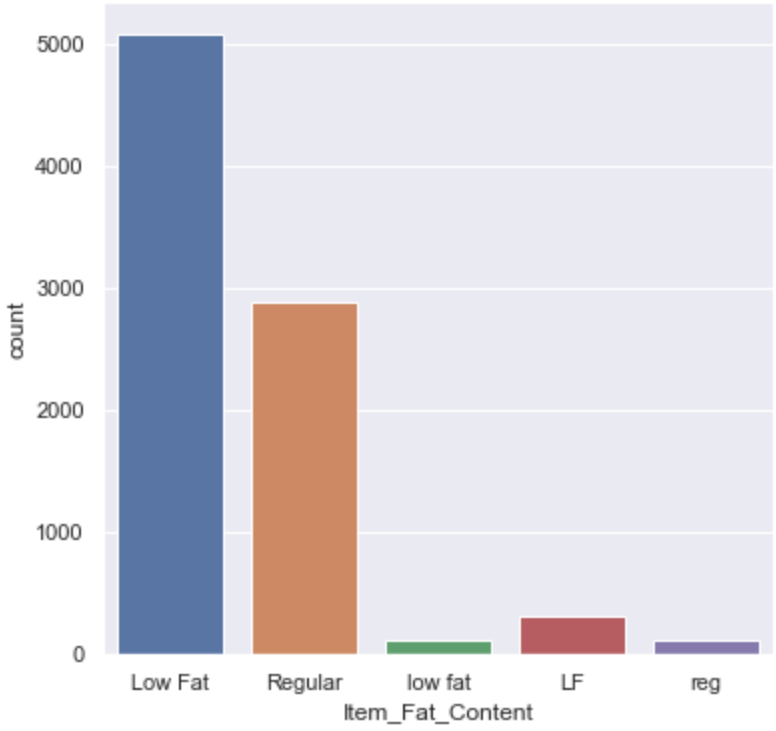
In [26]:
```
# Item_Outlet_Sales
plt.figure(figsize=(6,6))
sns.distplot(df_train['Item_Outlet_Sales'])
plt.show()
```



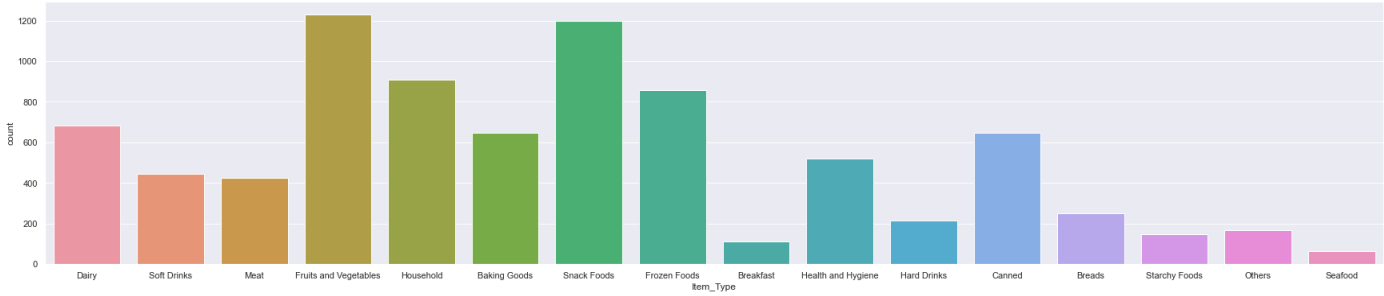
In [27]:
```
# Outlet_Establishment_Year
plt.figure(figsize=(6,6))
sns.countplot(x='Outlet_Establishment_Year', data=df_train)
plt.show()
```



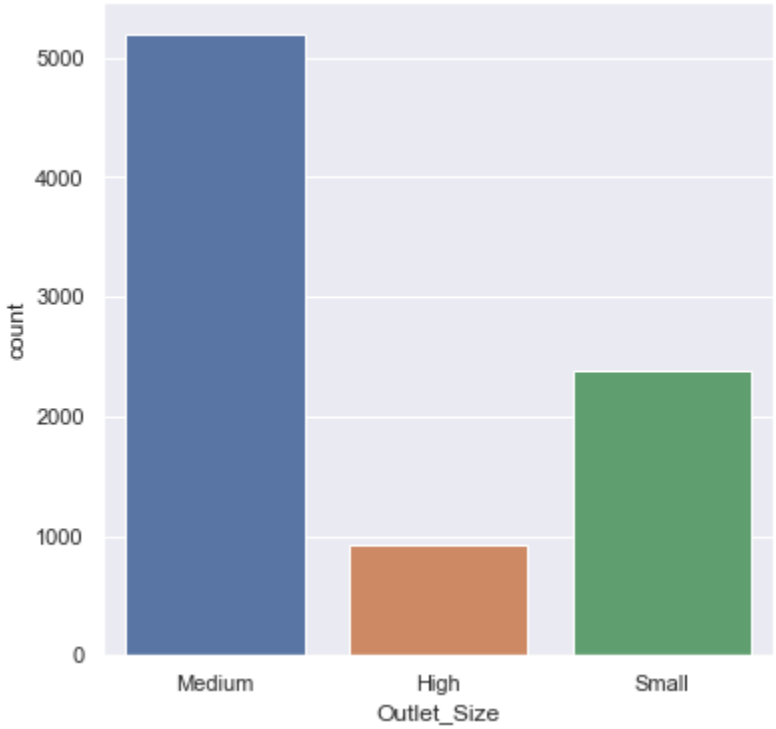
In [28]:
```
# Item_Fat_Content
plt.figure(figsize=(6,6))
sns.countplot(x='Item_Fat_Content', data=df_train)
plt.show()
```



In [29]:
```
# Item_Type
plt.figure(figsize=(30,6))
sns.countplot(x='Item_Type', data=df_train)
plt.show()
```



In [30]:
```
# Outlet_Size
plt.figure(figsize=(6,6))
sns.countplot(x='Outlet_Size', data=df_train)
plt.show()
```



In [31]:
```
df_train
```

Out[31]:

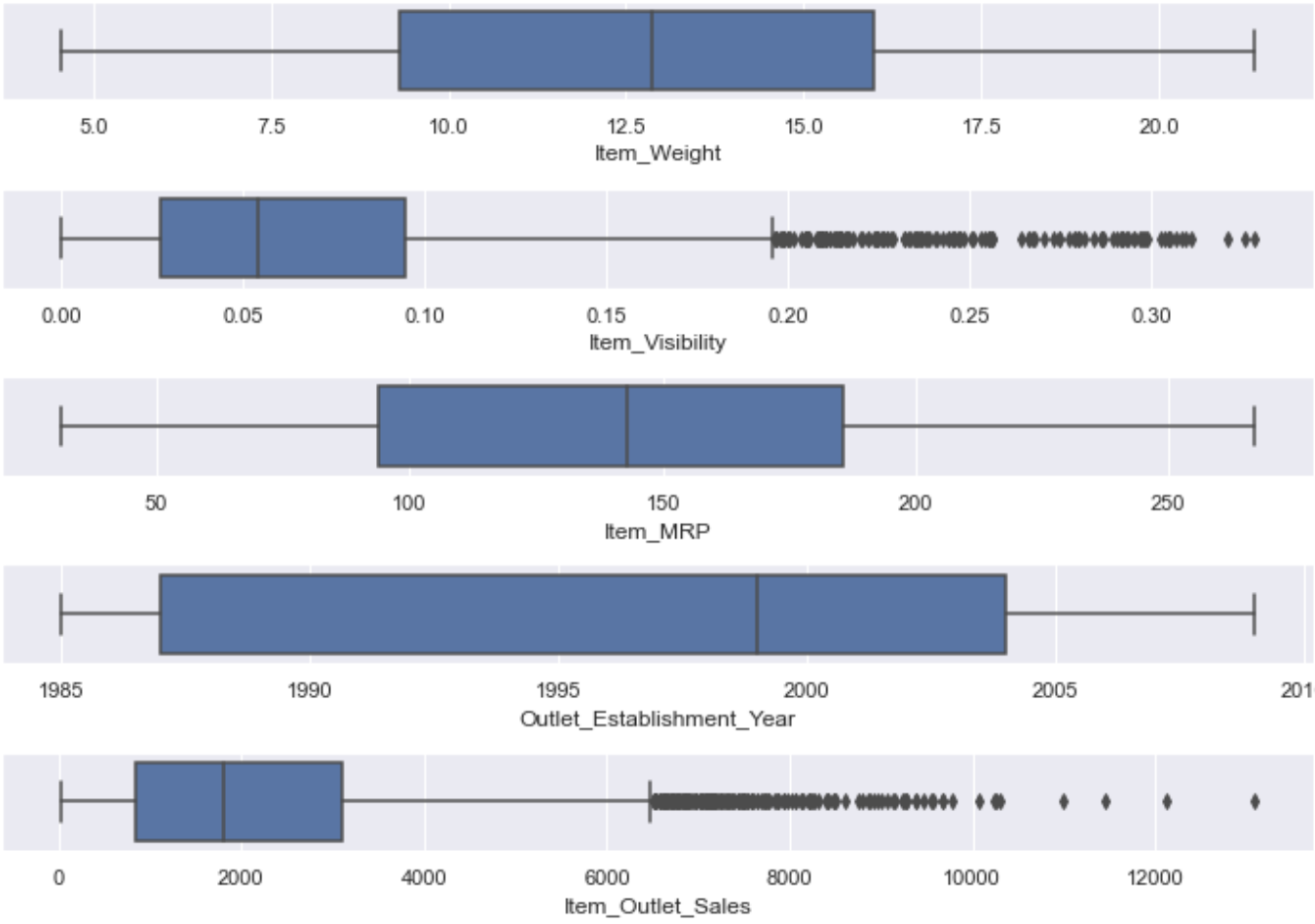
	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Ou
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8092	OUT049	
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.2692	OUT018	
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6180	OUT049	
3	FDX07	19.200	Regular	0.000000	Fruits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

8523 rows × 12 columns

Outliers Checking

In [35]:

```
fig, axs = plt.subplots(5, figsize = (10,7))
pt1 = sns.boxplot(df_train['Item_Weight'], ax = axs[0])
pt2 = sns.boxplot(df_train['Item_Visibility'], ax = axs[1])
pt3 = sns.boxplot(df_train['Item_MRP'], ax = axs[2])
pt4 = sns.boxplot(df_train['Outlet_Establishment_Year'], ax = axs[3])
pt5 = sns.boxplot(df_train['Item_Outlet_Sales'], ax = axs[4])
plt.tight_layout()
```



- no outliers to remove

Correlation

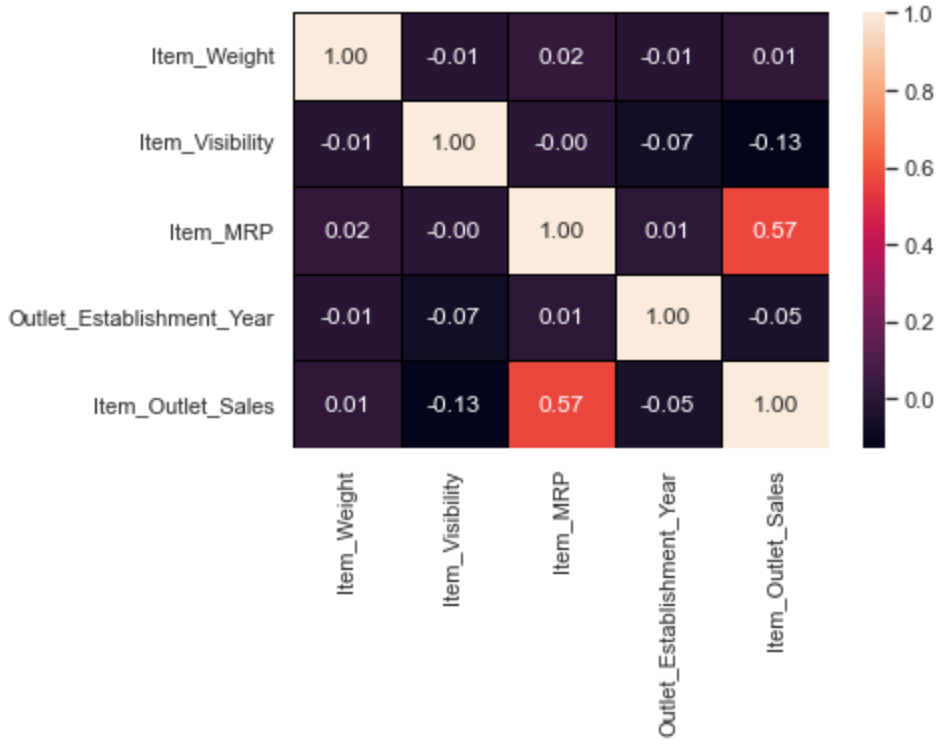
In [32]:
```
# Calculate correlations
corr = df_train.corr()
corr
```

Out[32]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
Item_Weight	1.000000	-0.012049	0.024756	-0.008301	0.011550
Item_Visibility	-0.012049	1.000000	-0.001315	-0.074834	-0.128625
Item_MRP	0.024756	-0.001315	1.000000	0.005020	0.567574
Outlet_Establishment_Year	-0.008301	-0.074834	0.005020	1.000000	-0.049135
Item_Outlet_Sales	0.011550	-0.128625	0.567574	-0.049135	1.000000

In [33]:
```
# Heatmap
sns.heatmap(corr,  annot=True,linewidths=0.5,linecolor="black", fmt=".2f")
```

Out[33]:
```
<AxesSubplot:>
```



Preprocessing

In [37]:
```
df_train['Item_Fat_Content'].value_counts()
```

Out[37]:
```
Low Fat    5089
Regular    2889
LF          316
reg         117
low fat     112
Name: Item_Fat_Content, dtype: int64
```

In [38]:
```
df_train.replace({'Item_Fat_Content': {'low fat':'Low Fat','LF':'Low Fat', 'reg':'Regula
df_test.replace({'Item_Fat_Content': {'low fat':'Low Fat','LF':'Low Fat', 'reg':'Regular
```

In [39]:
```
df_train['Item_Fat_Content'].value_counts()
```

Out[39]:
```
Low Fat    5517
Regular    3006
Name: Item_Fat_Content, dtype: int64
```

Removing the columns which not needed from both training & testing Data

In [40]:
```
df_train.drop(['Item_Identifier','Outlet_Identifier'],axis=1,inplace=True)
df_test.drop(['Item_Identifier','Outlet_Identifier'],axis=1,inplace=True)
```

In [41]:
```
df_train
```

Out[41]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_S
0	9.300	Low Fat	0.016047	Dairy	249.8092	1999	Med
1	5.920	Regular	0.019278	Soft Drinks	48.2692	2009	Med
2	17.500	Low Fat	0.016760	Meat	141.6180	1999	Med
3	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	1998	Med
4	8.930	Low Fat	0.000000	Household	53.8614	1987	H
...	...	...	...	...	...	...	
8518	6.865	Low Fat	0.056783	Snack Foods	214.5218	1987	H
8519	8.380	Regular	0.046982	Baking Goods	108.1570	2002	Med
8520	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	2004	Sr
8521	7.210	Regular	0.145221	Snack Foods	103.1332	2009	Med
8522	14.800	Low Fat	0.044878	Soft Drinks	75.4670	1997	Sr

8523 rows × 10 columns

In [42]:
```
df_test
```

Out[42]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_S
0	20.750000	Low Fat	0.007565	Snack Foods	107.8622	1999	Medi
1	8.300000	Regular	0.038428	Dairy	87.3198	2007	Medi
2	14.600000	Low Fat	0.099575	Others	241.7538	1998	Medi
3	7.315000	Low Fat	0.015388	Snack Foods	155.0340	2007	Medi
4	12.695633	Regular	0.118599	Dairy	234.2300	1985	Medi
...	...	...	...	...	...	...	
5676	10.500000	Regular	0.013496	Snack Foods	141.3154	1997	Sr
5677	7.600000	Regular	0.142991	Starchy Foods	169.1448	2009	Medi
5678	10.000000	Low Fat	0.073529	Health and Hygiene	118.7440	2002	Medi
5679	15.300000	Regular	0.000000	Canned	214.6218	2007	Medi
5680	9.500000	Regular	0.104720	Canned	79.7960	2002	Medi

5681 rows × 9 columns

Label Encoding

In [43]:
```
encoder = LabelEncoder()
df_train['Item_Fat_Content'] = encoder.fit_transform(df_train['Item_Fat_Content'])

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

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

df_train['Outlet_Location_Type'] = encoder.fit_transform(df_train['Outlet_Location_Type'

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

In [44]:
```
df_train
```

Out[44]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_S
0	9.300	0	0.016047	4	249.8092	1999	
1	5.920	1	0.019278	14	48.2692	2009	
2	17.500	0	0.016760	10	141.6180	1999	
3	19.200	1	0.000000	6	182.0950	1998	
4	8.930	0	0.000000	9	53.8614	1987	
...	...	...	...	...	...	...	
8518	6.865	0	0.056783	13	214.5218	1987	
8519	8.380	1	0.046982	0	108.1570	2002	
8520	10.600	0	0.035186	8	85.1224	2004	
8521	7.210	1	0.145221	13	103.1332	2009	
8522	14.800	0	0.044878	14	75.4670	1997	

8523 rows × 10 columns

Train Test Splitting

In [46]:
```
X=df_train.drop('Item_Outlet_Sales',axis=1)
```

In [47]:
```
Y=df_train['Item_Outlet_Sales']
```

In [48]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, random_state=101, test_size=0.2
```

In [49]:
```
X.describe()
```

Out[49]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Ou
count	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	852
mean	12.857645	0.352693	0.066132	7.226681	140.992782	1997.831867	
std	4.226124	0.477836	0.051598	4.209990	62.275067	8.371760	
min	4.555000	0.000000	0.000000	0.000000	31.290000	1985.000000	
25%	9.310000	0.000000	0.026989	4.000000	93.826500	1987.000000	
50%	12.857645	0.000000	0.053931	6.000000	143.012800	1999.000000	
75%	16.000000	1.000000	0.094585	10.000000	185.643700	2004.000000	
max	21.350000	1.000000	0.328391	15.000000	266.888400	2009.000000	

In [50]:
```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
```

In [51]:
```
X_train_std= sc.fit_transform(X_train)
```

In [52]:
```
X_test_std= sc.transform(X_test)
```

In [53]:
```
X_train_std
```

Out[53]:
```
array([[ 1.52290029, -0.74155088,  0.68469729, ..., -1.95699503,
         1.08786619, -0.25964107],
       [-1.23985603, -0.74155088, -0.09514748, ..., -0.28872895,
        -0.13870429, -0.25964107],
       [ 1.54667616,  1.34852514, -0.00838589, ..., -0.28872895,
        -0.13870429, -0.25964107],
       ...,
       [-0.08197107, -0.74155088, -0.9191623 , ...,  1.37953713,
        -1.36527477, -0.25964107],
       [-0.74888428,  1.34852514,  1.21363058, ..., -0.28872895,
        -0.13870429, -0.25964107],
       [ 0.67885683, -0.74155088,  1.83915356, ..., -0.28872895,
         1.08786619,  0.98524841]])
```

In [54]:
```
X_test_std
```

Out[54]:
```
array([[-0.43860915, -0.74155088, -0.21609255, ..., -0.28872895,
         1.08786619,  0.98524841],
       [ 1.22570189, -0.74155088, -0.52943461, ..., -1.95699503,
         1.08786619, -0.25964107],
       [-1.21845775,  1.34852514,  0.16277342, ...,  1.37953713,
        -1.36527477, -0.25964107],
       ...,
       [ 0.65508096, -0.74155088,  0.87824237, ..., -0.28872895,
         1.08786619, -1.50453056],
       [ 1.01171904, -0.74155088, -1.28409256, ..., -0.28872895,
         1.08786619,  0.98524841],
       [-1.56558548,  1.34852514, -1.09265374, ..., -0.28872895,
        -0.13870429, -0.25964107]])
```

In [55]:
```
Y_train
```

Out[55]:
```
3684     163.7868
1935    1607.2412
5142    1510.0344
4978    1784.3440
2299    3558.0352
          ...
599     5502.8370
5695    1436.7964
8006    2167.8448
1361    2700.4848
1547     829.5868
Name: Item_Outlet_Sales, Length: 6818, dtype: float64
```

In [56]:
```
Y_test
```

Out[56]:
```
8179     904.8222
8355    2795.6942
3411    1947.4650
7089     872.8638
6954    2450.1440
          ...
1317    1721.0930
4996     914.8092
531      370.1848
3891    1358.2320
6629    2418.1856
Name: Item_Outlet_Sales, Length: 1705, dtype: float64
```

Building Model

LinearRegression

In [70]:
```
from sklearn.linear_model import LinearRegression
lr= LinearRegression()
```

In [71]:
```
lr.fit(X_train_std,Y_train)
```

Out[71]:
```
LinearRegression()
```

In [72]:
```
X_test.head()
```

Out[72]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_S
8179	11.00	0	0.055163	8	100.3358	2009	
8355	18.00	0	0.038979	13	148.6418	1987	
3411	7.72	1	0.074731	1	77.5986	1997	
7089	20.70	0	0.049035	6	39.9506	2007	
6954	7.55	0	0.027225	3	152.9340	2002	

In [73]:
```
Y_pred_lr=lr.predict(X_test_std)
```

In [74]:
```
from sklearn.metrics import r2_score
print(r2_score(Y_test,Y_pred_lr))
```

```
0.5040717488447088
```

In [75]:
```
from sklearn.model_selection import cross_val_score
lrscore = cross_val_score(lr, X, Y, cv=5)
```

```
lrc = lrscore.mean()
print('Cross val score :', lrc*100 )
```

```
Cross val score : 50.59368501934476
```

Random Forest Regressor

In [76]:
```
from sklearn.ensemble import RandomForestRegressor
rf= RandomForestRegressor(n_estimators=1000)
```

In [77]:
```
rf.fit(X_train_std,Y_train)
```

Out[77]:
```
RandomForestRegressor(n_estimators=1000)
```

In [78]:
```
Y_pred_rf= rf.predict(X_test_std)
```

In [79]:
```
print(r2_score(Y_test,Y_pred_rf))
```

```
0.5493673287022895
```

In [81]:
```
rfscore = cross_val_score(rf, X, Y, cv=5)
rfc = rfscore.mean()
print('Cross val score :', rfc*100 )
```

```
Cross val score : 55.30370437390253
```

In [84]:
```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV

parameters = {'alpha':[.0001, .001, .01, .1, 1, 10],
              'random_state':list(range(0,10))}
ls = Lasso()
clf = GridSearchCV(ls,parameters)
clf.fit(X_train,Y_train)
print(clf.best_params_)
```

```
{'alpha': 0.1, 'random_state': 0}
```

In [87]:
```
#model training
ls = Lasso(alpha=0.01,random_state=0)
ls.fit(X_train,Y_train)
ls_score_training = ls.score(X_train,Y_train)
pred_ls = ls.predict(X_test)
ls_score_training*100
```

Out[87]:
```
50.85057617143938
```

In [89]:
```
cv_score=cross_val_score(ls,X,Y,cv=5)
cv_mean=cv_score.mean()
cv_mean*100
```

Out[89]:
```
50.593680554025624
```

In [90]:
```
from sklearn.model_selection import RepeatedStratifiedKFold

# define models and parameters
model = RandomForestRegressor()
n_estimators = [10, 100, 1000]
max_depth=range(1,31)
min_samples_leaf=np.linspace(0.1, 1.0)
max_features=["auto", "sqrt", "log2"]
min_samples_split=np.linspace(0.1, 1.0, 10)
```

```
# define grid search
grid = dict(n_estimators=n_estimators)

#cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=101)

grid_search_forest = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                           scoring='r2',error_score=0,verbose=2,cv=2)

grid_search_forest.fit(X_train_std, Y_train)
```

Out[90]:
```
Fitting 2 folds for each of 3 candidates, totalling 6 fits
GridSearchCV(cv=2, error_score=0, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid={'n_estimators': [10, 100, 1000]}, scoring='r2',
             verbose=2)
```

In [91]:
```
# summarize results
print(f"Best: {grid_search_forest.best_score_:.3f} using {grid_search_forest.best_params
means = grid_search_forest.cv_results_['mean_test_score']
stds = grid_search_forest.cv_results_['std_test_score']
params = grid_search_forest.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print(f"{mean:.3f} ({stdev:.3f}) with: {param}")
```

```
Best: 0.548 using {'n_estimators': 1000}
0.501 (0.006) with: {'n_estimators': 10}
0.547 (0.005) with: {'n_estimators': 100}
0.548 (0.005) with: {'n_estimators': 1000}
```

In [92]:
```
grid_search_forest.best_params_
```

Out[92]:
```
{'n_estimators': 1000}
```

In [93]:
```
grid_search_forest.best_score_
```

Out[93]:
```
0.5484005161682183
```

In [94]:
```
Y_pred_rf_grid=grid_search_forest.predict(X_test_std)
```

In [95]:
```
r2_score(Y_test,Y_pred_rf_grid)
```

Out[95]:
```
0.5487320716540314
```

In [97]:
```
import pickle
filename = 'outletsales.pkl'
pickle.dump(grid_search_forest,open(filename, 'wb'))
```

In [ ]:

In [ ]: