In [6]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
from sklearn import metrics
```

In [ ]:
```
# Importing Data from csv file
```

In [104…
```
data = pd.read_csv('/Train_sales_data.csv')
data
```

Out[104]:

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

8523 rows × 12 columns

In [9]:
```
data.head()
```

Out[9]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	

In [10]:
```
#Identifying the data size
data.shape
```

Out[10]:
```
(8523, 12)
```

In [11]:
```
#Getting information about the dataset
data.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]:
```
objectdatatypes=data.select_dtypes(include = 'object').columns
objectdatatypes
```

Out[12]:
```
Index(['Item_Identifier', 'Item_Fat_Content', 'Item_Type', 'Outlet_Identifier',
       'Outlet_Size', 'Outlet_Location_Type', 'Outlet_Type'],
      dtype='object')
```

There are 7 categorical columns

In [13]:
```
# checking for missing values
```

In [14]:
```
data.isnull().sum()
```

Out[14]:
```
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 [15]:
```
#Imputing null values of Item weight with mean of Item weight column
```

In [16]:
```
data['Item_Weight'].mean()
```

Out[16]:
```
12.857645184135976
```

In [17]:
```
data['Item_Weight'].fillna(data['Item_Weight'].mean(),inplace = True)
```

In [ ]:
```
#checking whether is there any null value in the Item_Weight column
```

In [18]:
```
data.isnull().sum()
```

Out[18]:
```
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 [19]:
```
#Since outlet_size is a categorical column-imputing the null value with mode function
```

In [20]:
```
data['Outlet_Size'].unique()
```

Out[20]:
```
array(['Medium', nan, 'High', 'Small'], dtype=object)
```

In [21]:
```
data['Outlet_Size'].mode()
```

Out[21]:
```
0    Medium
Name: Outlet_Size, dtype: object
```

In [26]:
```
mode_of_outlet_size = data.pivot_table(values = 'Outlet_Size',columns = 'Outlet_Type',aggfunc = (lambda x:x.mode()[0]))
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 [27]:
```
miss_values = data['Outlet_Size'].isnull()
miss_values
```

Out[27]:
```
0       False
1       False
2       False
3       False
4       False
        ...
8518    False
8519    False
8520    False
8521    False
8522    False
Name: Outlet_Size, Length: 8523, dtype: bool
```

In [28]:
```
data.loc[miss_values, 'Outlet_Size'] = data.loc[miss_values,'Outlet_Type'].apply(lambda x: mode_of_outlet_size[x])
```

In [29]:
```
data.isnull().sum()
```

Out[29]:
```
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
```

Data Analysis

In [30]:
```
data.describe()
```

Out[30]:

	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

In [31]:
```
# Numerical Features
```

In [32]:
```
#Item_weight distribution

plt.figure(figsize=(6,6))
```

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

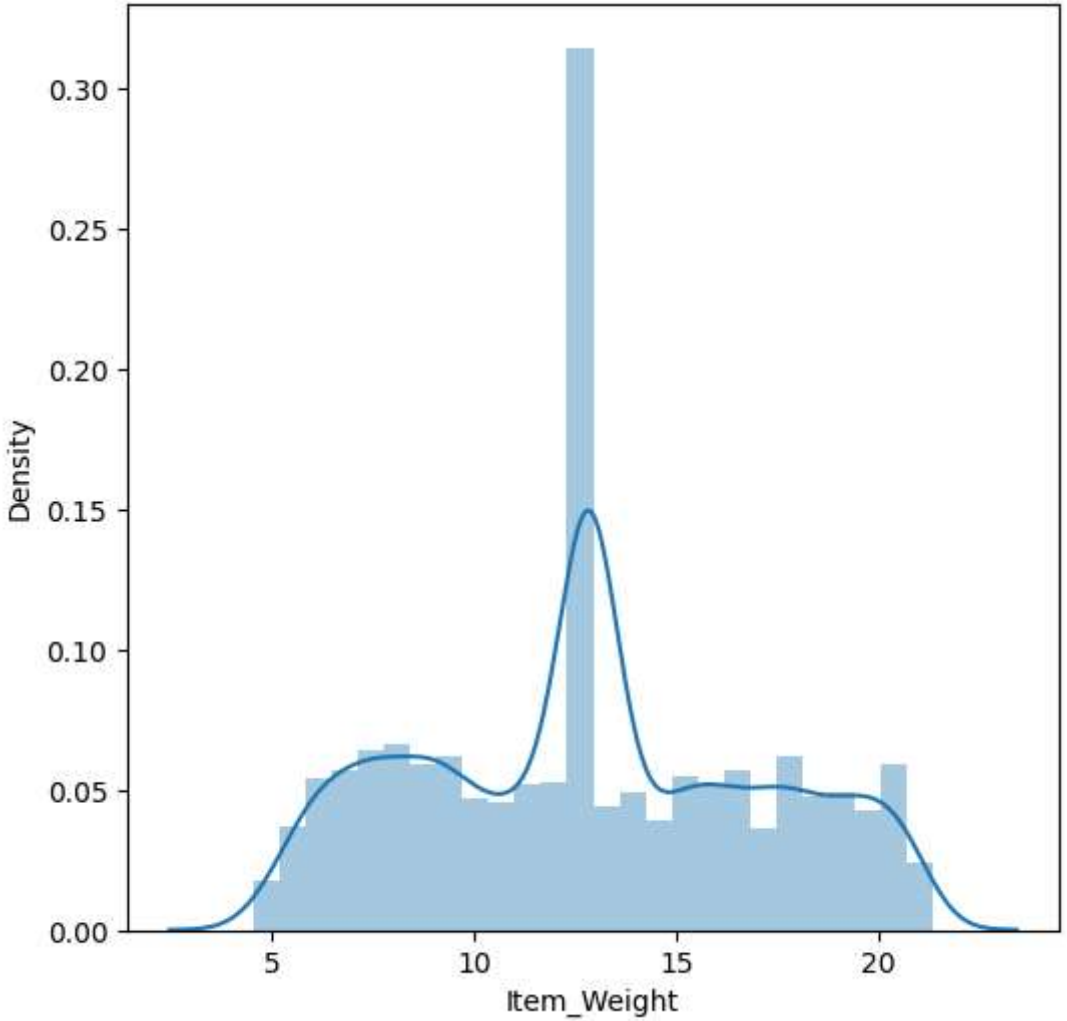
```
<ipython-input-32-f22ea175400e>:4: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['Item_Weight'])
```



In [ ]:
```
# Item Visibility distribution
```

In [33]:
```
plt.figure(figsize=(6,6))
sns.distplot(data['Item_Visibility'])
plt.show()
```

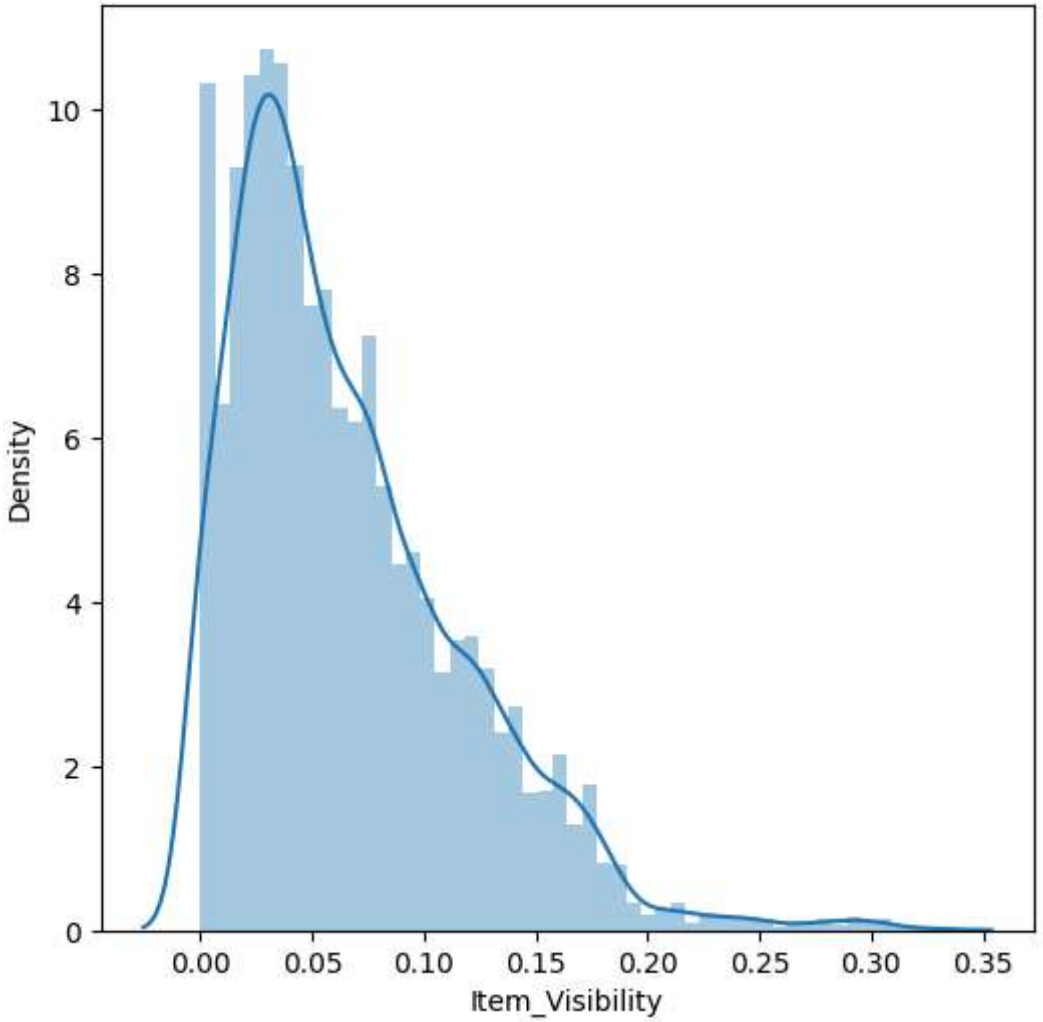
```
<ipython-input-33-9bc277e39cf4>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['Item_Visibility'])
```



In [34]:
```
plt.figure(figsize=(6,6))
sns.distplot(data['Item_MRP'])
plt.show()
```

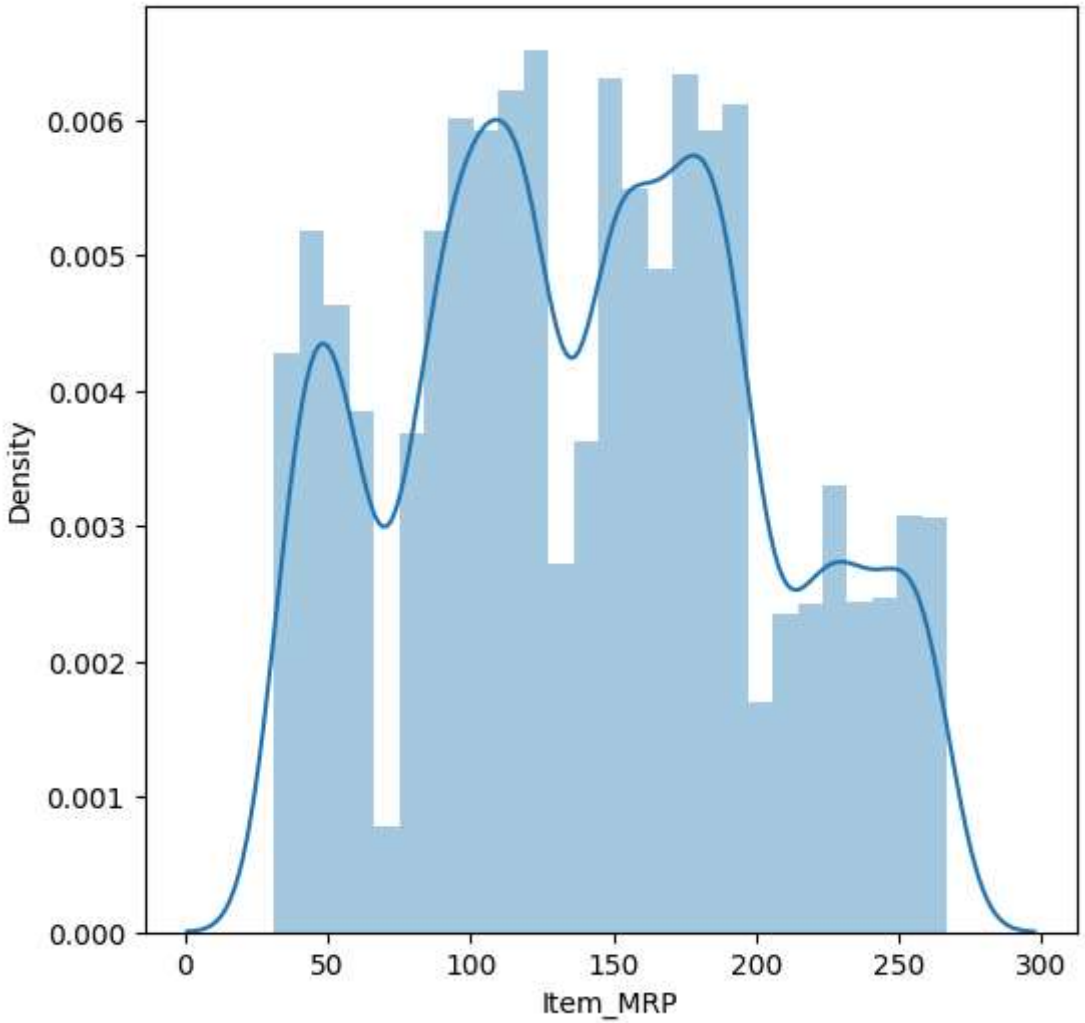
```
<ipython-input-34-9168d8271af4>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['Item_MRP'])
```



In [35]:
```
plt.figure(figsize=(6,6))
sns.distplot(data['Item_Outlet_Sales'])
plt.show()
```

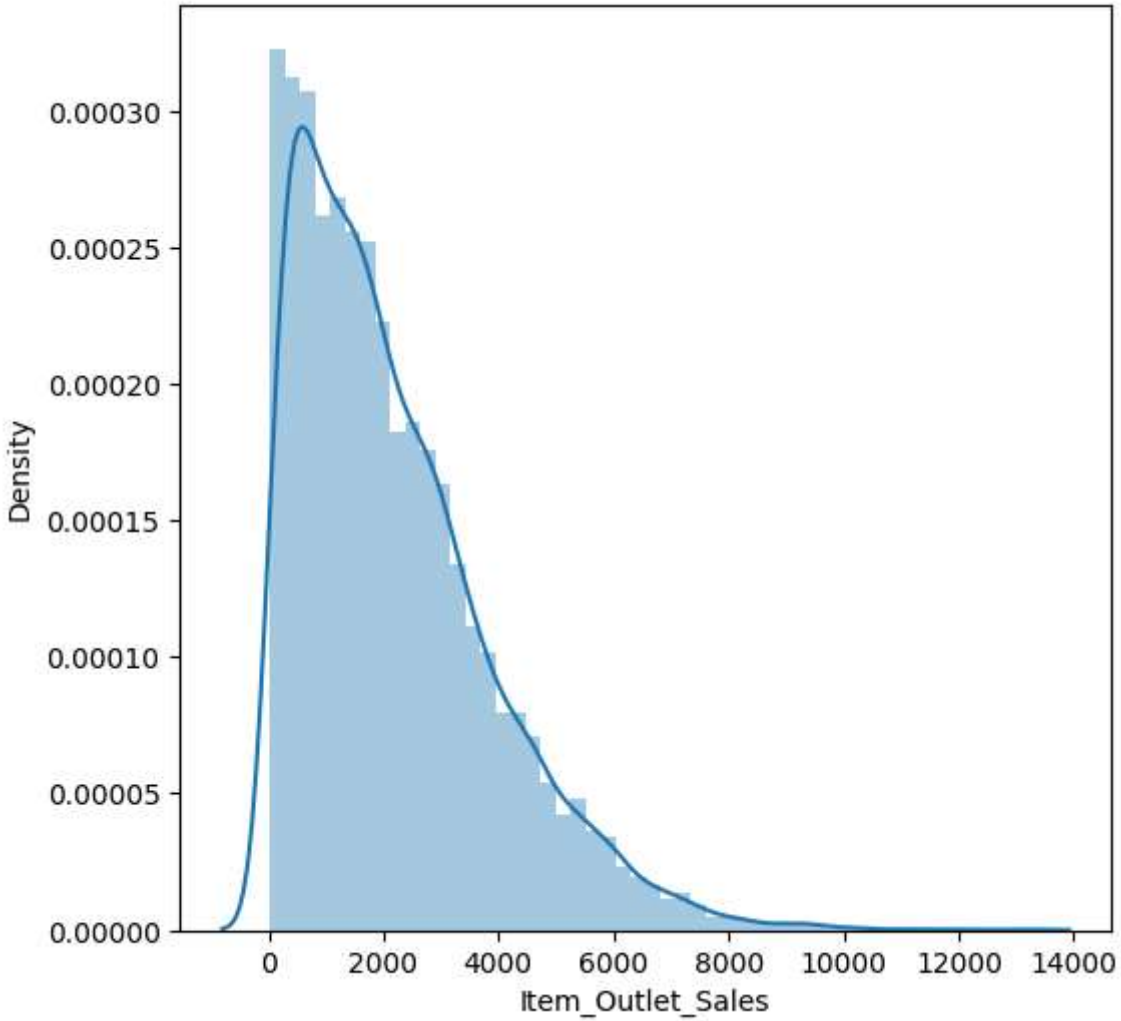
```
<ipython-input-35-39a234040911>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['Item_Outlet_Sales'])
```



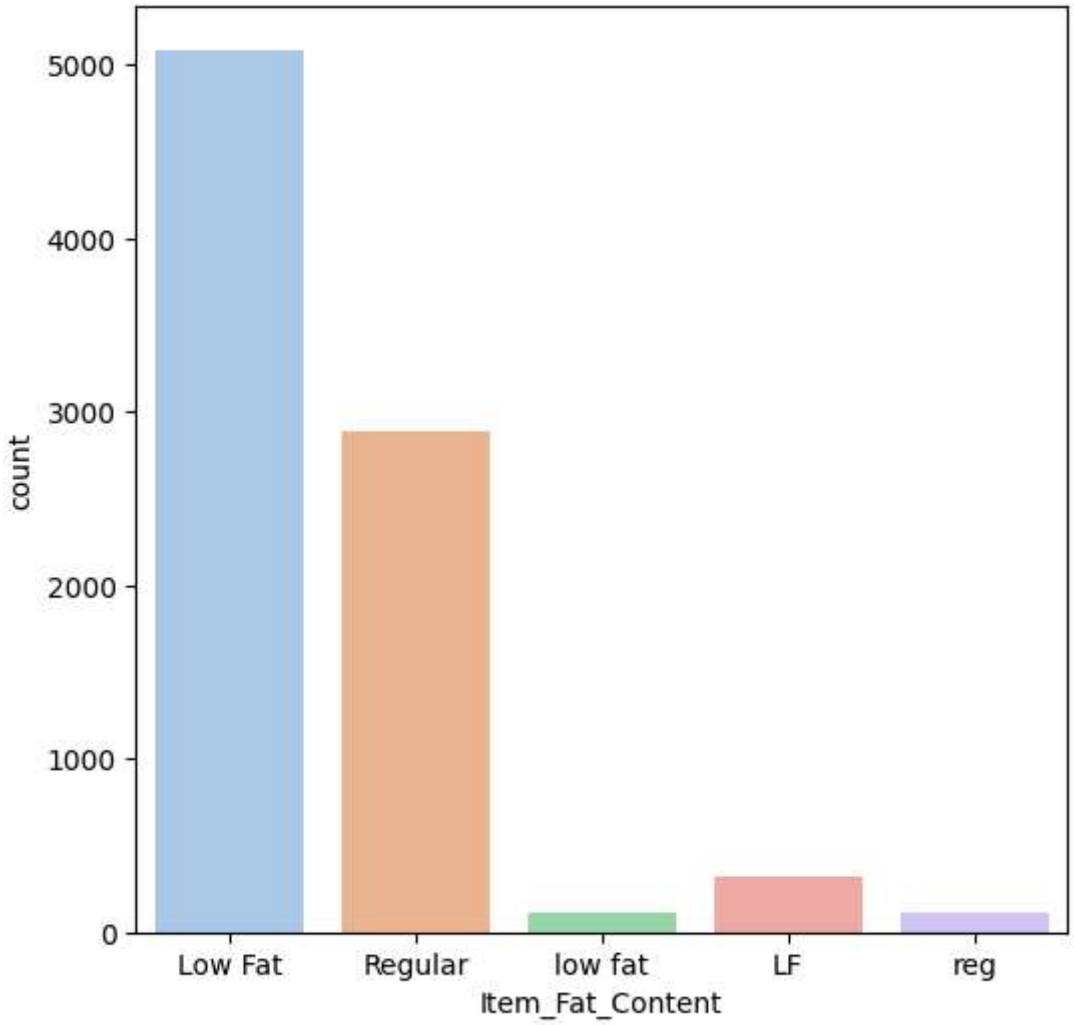
In [ ]:
```
# CATEGORICAL FEATURES
```

In [37]:
```
plt.figure(figsize=(6,6))
sns.countplot(x='Item_Fat_Content',data=data,palette = 'pastel')
plt.show()
```

```
<ipython-input-37-763f9b291851>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x='Item_Fat_Content',data=data,palette = 'pastel')
```

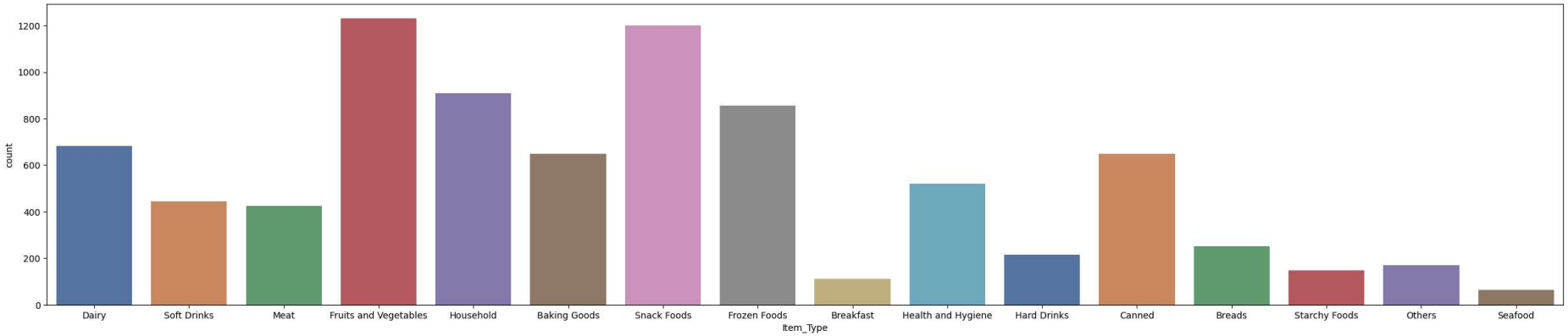


In [39]:
```
# Item_Type column
plt.figure(figsize=(30,6))
sns.countplot(x='Item_Type', data=data,palette = 'deep')
plt.show()
```

```
<ipython-input-39-19bb79d0e866>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x='Item_Type', data=data,palette = 'deep')
```

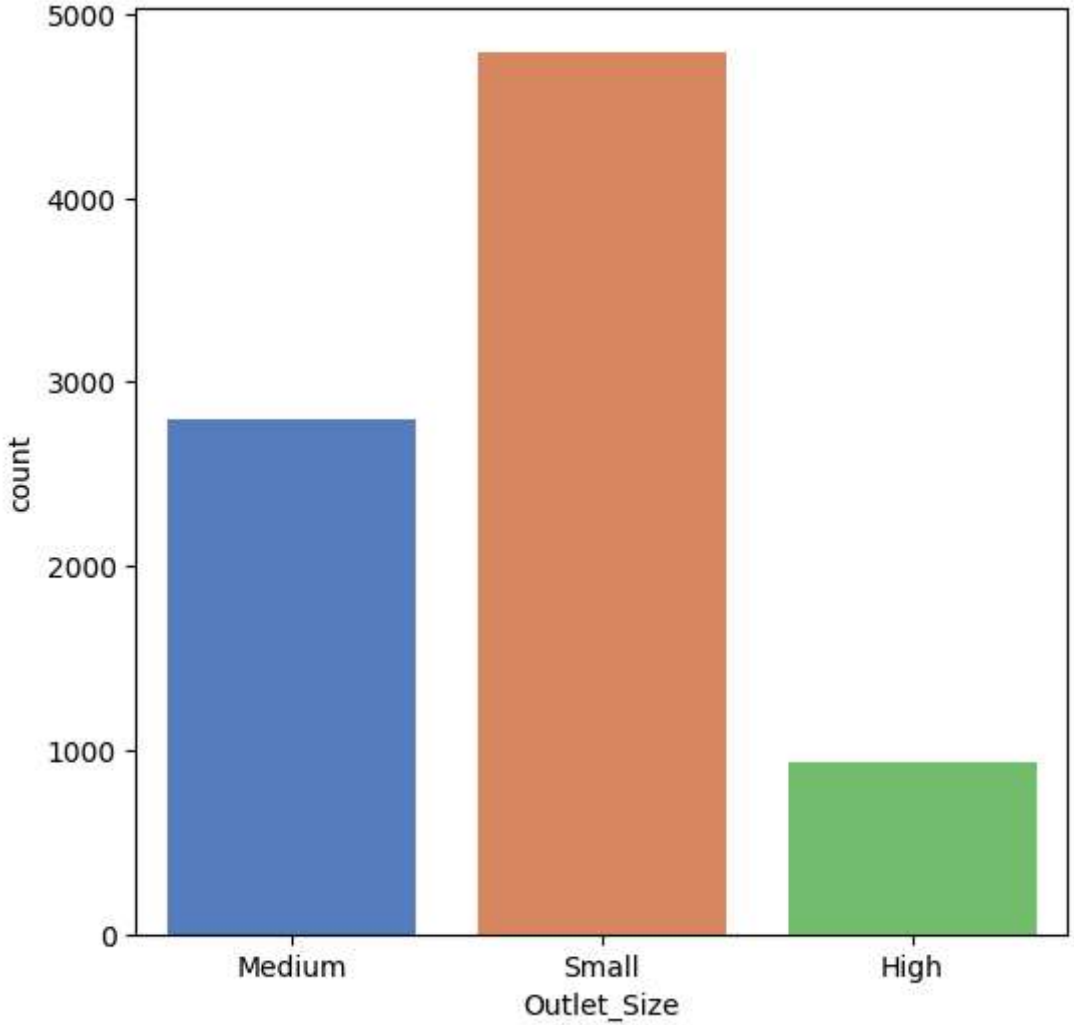


In [40]:
```
# Outlet_Size column
plt.figure(figsize=(6,6))
sns.countplot(x='Outlet_Size', data=data,palette ='muted')
plt.show()
```

```
<ipython-input-40-d4fdc312050f>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(x='Outlet_Size', data=data,palette ='muted')
```



Data Pre-Processing

In [41]:
```
data.head()
```

Out[41]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	Small	
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	

In [42]:
```
data['Item_Fat_Content'].value_counts()
```

Out[42]:
```
Item_Fat_Content
Low Fat    5089
Regular    2889
LF          316
reg         117
low fat     112
Name: count, dtype: int64
```

In [43]:
```
data.replace({'Item_Fat_Content': {'low fat':'Low Fat','LF':'Low Fat', 'reg':'Regular'}}, inplace=True)
```

In [44]:
```
data['Item_Fat_Content'].value_counts()
```

Out[44]:
```
Item_Fat_Content
Low Fat    5517
Regular    3006
Name: count, dtype: int64
```

Label Encoding

In [45]:
```
encoder = LabelEncoder()
```

In [48]:
```
data['Item_Identifier'] = encoder.fit_transform (data['Item_Identifier'])
data['Item_Fat_Content'] = encoder.fit_transform(data['Item_Fat_Content'])
data['Item_Type']=encoder.fit_transform(data['Item_Type'])
data['Outlet_Identifier'] = encoder.fit_transform(data['Outlet_Identifier'])
data['Outlet_Size']=encoder.fit_transform(data['Outlet_Size'])
data['Outlet_Location_Type']=encoder.fit_transform(data['Outlet_Location_Type'])
data['Outlet_Type'] = encoder.fit_transform(data['Outlet_Type'])
```

In [51]:
```
data.head()
```

Out[51]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_
0	156	9.30	0	0.016047	4	249.8092	9	1999	1	
1	8	5.92	1	0.019278	14	48.2692	3	2009	1	
2	662	17.50	0	0.016760	10	141.6180	9	1999	1	
3	1121	19.20	1	0.000000	6	182.0950	0	1998	2	
4	1297	8.93	0	0.000000	9	53.8614	1	1987	0	

In [52]:
```
encoder.classes_
```

Out[52]:
```
array(['Grocery Store', 'Supermarket Type1', 'Supermarket Type2',
       'Supermarket Type3'], dtype=object)
```

Splitting the data into Features and Target

In [58]:
```
x= data.drop(columns = ['Item_Outlet_Sales'],axis = 1)
y=data['Item_Outlet_Sales']
```

In [59]:
```
print(x)
```

```
      Item_Identifier  Item_Weight  Item_Fat_Content  Item_Visibility  \
0                 156        9.300                 0         0.016047
1                   8        5.920                 1         0.019278
2                 662       17.500                 0         0.016760
3                1121       19.200                 1         0.000000
4                1297        8.930                 0         0.000000
...               ...          ...               ...              ...
8518              370        6.865                 0         0.056783
8519              897        8.380                 1         0.046982
8520             1357       10.600                 0         0.035186
8521              681        7.210                 1         0.145221
8522               50       14.800                 0         0.044878

      Item_Type  Item_MRP  Outlet_Identifier  Outlet_Establishment_Year  \
0             4  249.8092                  9                       1999
1            14   48.2692                  3                       2009
2            10  141.6180                  9                       1999
3             6  182.0950                  0                       1998
4             9   53.8614                  1                       1987
...         ...       ...                ...                        ...
8518         13  214.5218                  1                       1987
8519          0  108.1570                  7                       2002
8520          8   85.1224                  6                       2004
8521         13  103.1332                  3                       2009
8522         14   75.4670                  8                       1997

      Outlet_Size  Outlet_Location_Type  Outlet_Type
0               1                     0            1
1               1                     2            2
2               1                     0            1
3               2                     2            0
4               0                     2            1
...           ...                   ...          ...
8518            0                     2            1
8519            2                     1            1
8520            2                     1            1
8521            1                     2            2
8522            2                     0            1

[8523 rows x 11 columns]
```

In [60]:
```
print(y)
```

```
0       3735.1380
1        443.4228
2       2097.2700
3        732.3800
4        994.7052
          ...
8518    2778.3834
8519     549.2850
8520    1193.1136
8521    1845.5976
8522     765.6700
Name: Item_Outlet_Sales, Length: 8523, dtype: float64
```

In [62]:
```
type(y)
```

Out[62]:
```
pandas.core.series.Series
def __init__(data=None, index=None, dtype: Dtype | None=None, name=None, copy: bool | None=None, fas
tpath: bool=False) -> None

One-dimensional ndarray with axis labels (including time series).

Labels need not be unique but must be a hashable type. The object
supports both integer- and label-based indexing and provides a host of
methods for performing operations involving the index. Statistical
methods from ndarray have been overridden to automatically exclude
```

In [63]:
```
type(x)
```

Out[63]:
```
pandas.core.frame.DataFrame
def __init__(data=None, index: Axes | None=None, columns: Axes | None=None, dtype: Dtype | None=Non
e, copy: bool | None=None) -> None

Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns).
Arithmetic operations align on both row and column labels. Can be
thought of as a dict-like container for Series objects. The primary
pandas data structure.
```

Splitting the data into Training and Testing Data

In [96]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size = 0.2,random_state=7)
```

In [89]:
```
print(x_train.shape,x_test.shape,y_train.shape,y_test.shape)
```

```
(6818, 11) (1705, 11) (6818,) (1705,)
```

Machine Learning and Model Training

In [97]:
```
regressor = XGBRegressor()
```

In [98]:
```
regressor.fit(x_train,y_train)
```

Out[98]:

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

Model Evaluation

In [99]:
```
#Prediction on Training Data
training_data_prediction = regressor.predict(x_train)
```

In [100…
```
#R squared value
r2_train = metrics.r2_score(training_data_prediction,y_train)
print("R squared value : ",r2_train)
```

```
R squared value :  0.8388098734577158
```

In [101…
```
#Prediction on Testing Data
test_data_prediction = regressor.predict(x_test)
```

In [103…
```
#R squared value
r2_test = metrics.r2_score(test_data_prediction,y_test)
print("R squared value : ",r2_test)
```

```
R squared value :  0.2663125659761737
```