Importing the Libraries

In [1]:
```
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
```

Data Collection and Processing

In [2]:
```
# loading the data from csv file to Pandas DataFrame
big_mart_data = pd.read_csv('/kaggle/input/bigmart-sales-data/Train
```

In [3]:
```
# first 5 rows of the dataframe
big_mart_data.head()
```

Out[3]:

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

In [4]:
```
# number of data points & number of features
big_mart_data.shape
```

Out[4]: (8523, 12)

In [5]:
```
# getting some information about thye dataset
big_mart_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
```

Categorical Features:

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

In [6]:
```
# checking for missing values
big_mart_data.isnull().sum()
```

Out[6]:
```
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
```

Handling Missing Values

Mean - average

Mode - more repeated value

In [7]:
```
# mean value of "Item_Weight" column
big_mart_data['Item_Weight'].mean()
```

Out[7]:
```
12.857645184135976
```

In [8]:
```
# filling the missing values in "Item_weight column" with "Mean" va
big_mart_data['Item_Weight'].fillna(big_mart_data['Item_Weight'].me
```

```
/tmp/ipykernel_33/2509980927.py:2: FutureWarning: A value is tryin
g to be set on a copy of a DataFrame or Series through chained ass
ignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will n
ever work because the intermediate object on which we are setting
values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] = df[co
l].method(value) instead, to perform the operation inplace on the
original object.


  big_mart_data['Item_Weight'].fillna(big_mart_data['Item_Weigh
t'].mean(), inplace=True)
```

Replaceing the missing values in "Outlet_Size" with mode

In [9]:
```
# mode of "Outlet_Size" column
big_mart_data['Outlet_Size'].mode()
```

Out[9]:
```
0    Medium
Name: Outlet_Size, dtype: object
```

In [10]:
```
# filling the missing values in "Outlet_Size" column with Mode
mode_of_Outlet_size = big_mart_data.pivot_table(values='Outlet_Size
```

In [11]:
```
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 [12]:
```
miss_values = big_mart_data['Outlet_Size'].isnull()
```

In [13]:
```
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 [14]:
```
big_mart_data.loc[miss_values, 'Outlet_Size'] = big_mart_data.loc[m
```

In [15]:
```
# checking for missing values
big_mart_data.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                   0
Outlet_Location_Type          0
Outlet_Type                   0
Item_Outlet_Sales             0
dtype: int64
```

Data Analysis

In [16]:
```
big_mart_data.describe()
```

Out[16]:

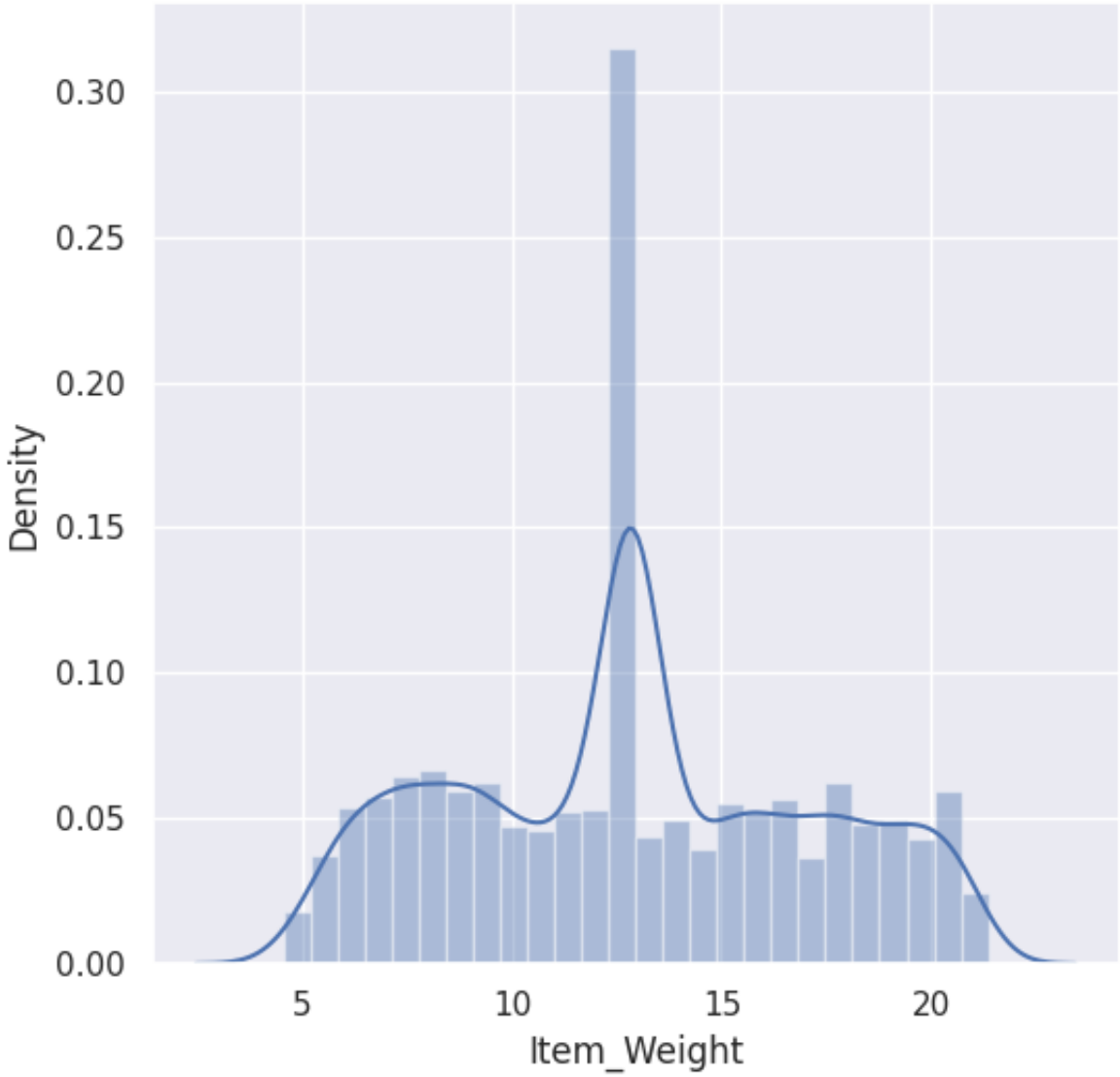
	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sale
count	8523.000000	8523.000000	8523.000000	8523.000000	8523.00000
mean	12.857645	0.066132	140.992782	1997.831867	2181.28891
std	4.226124	0.051598	62.275067	8.371760	1706.49961
min	4.555000	0.000000	31.290000	1985.000000	33.29000
25%	9.310000	0.026989	93.826500	1987.000000	834.24740
50%	12.857645	0.053931	143.012800	1999.000000	1794.33100
75%	16.000000	0.094585	185.643700	2004.000000	3101.29640
max	21.350000	0.328391	266.888400	2009.000000	13086.96480

Numerical Features

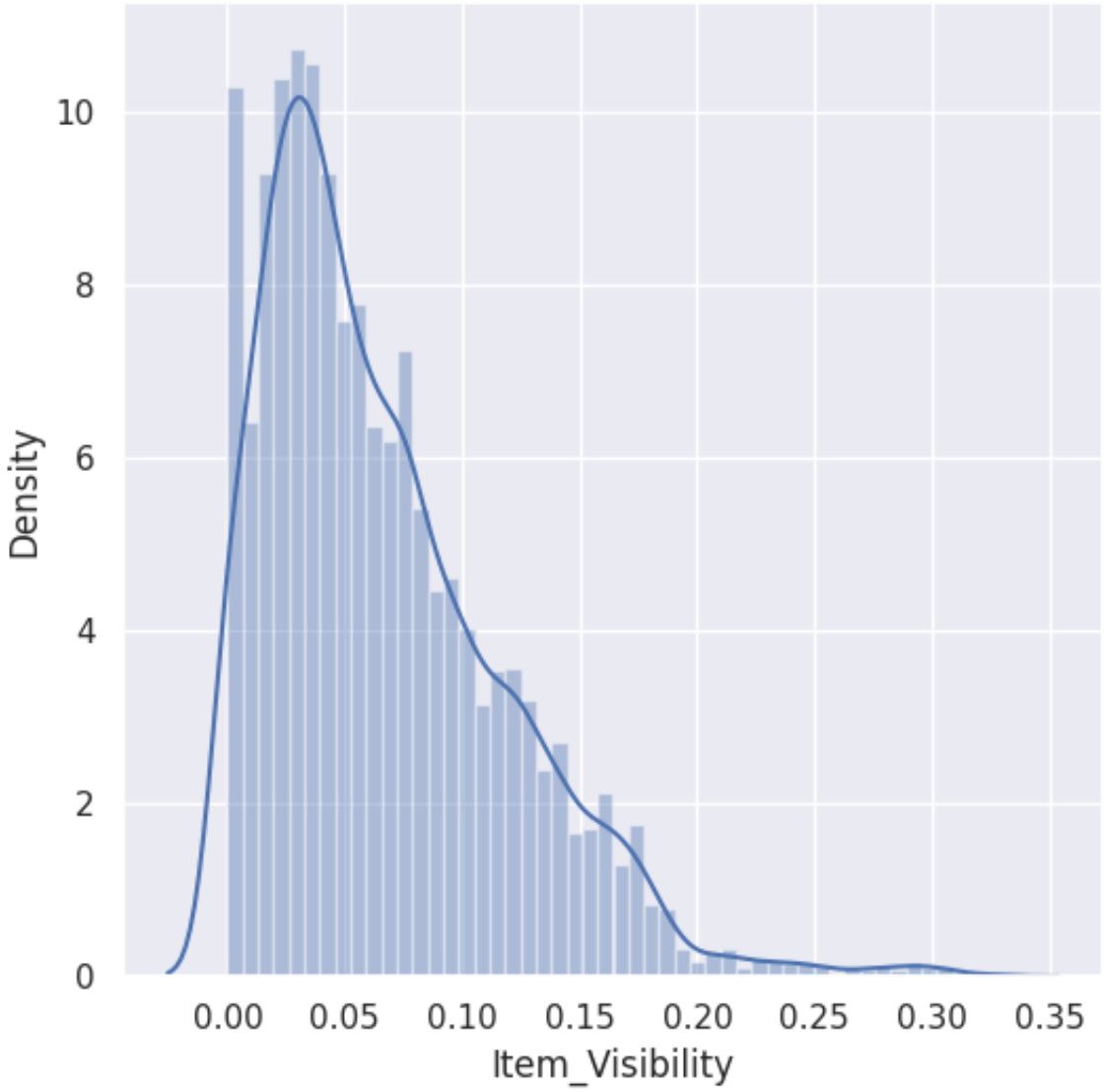
In [17]:
```
sns.set()
```

In [18]:
```
import warnings
warnings.filterwarnings('ignore')
```

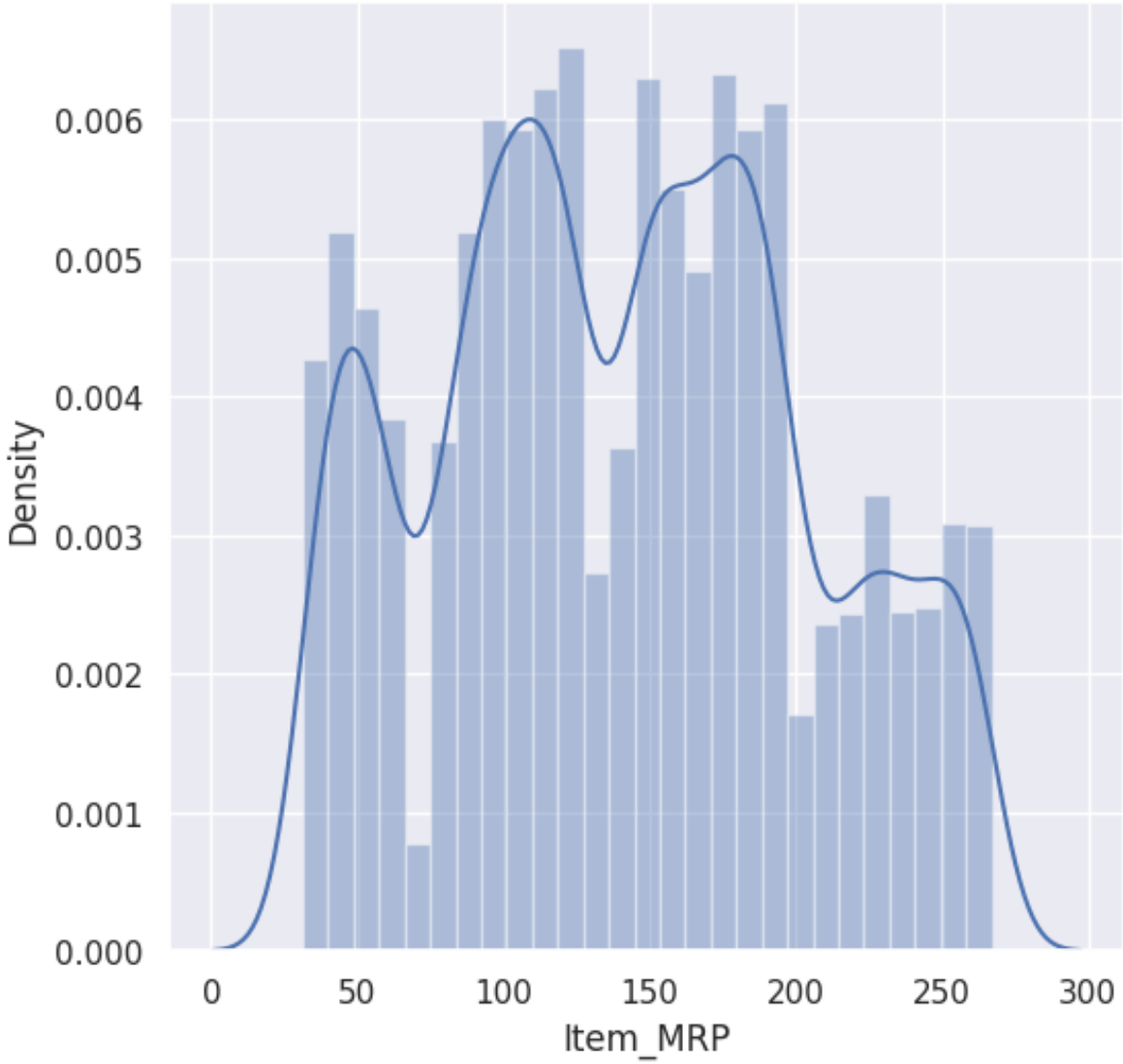
In [19]:
```
# Item_Weight distribution
plt.figure(figsize=(6,6))
sns.distplot(big_mart_data['Item_Weight'])
plt.show()
```



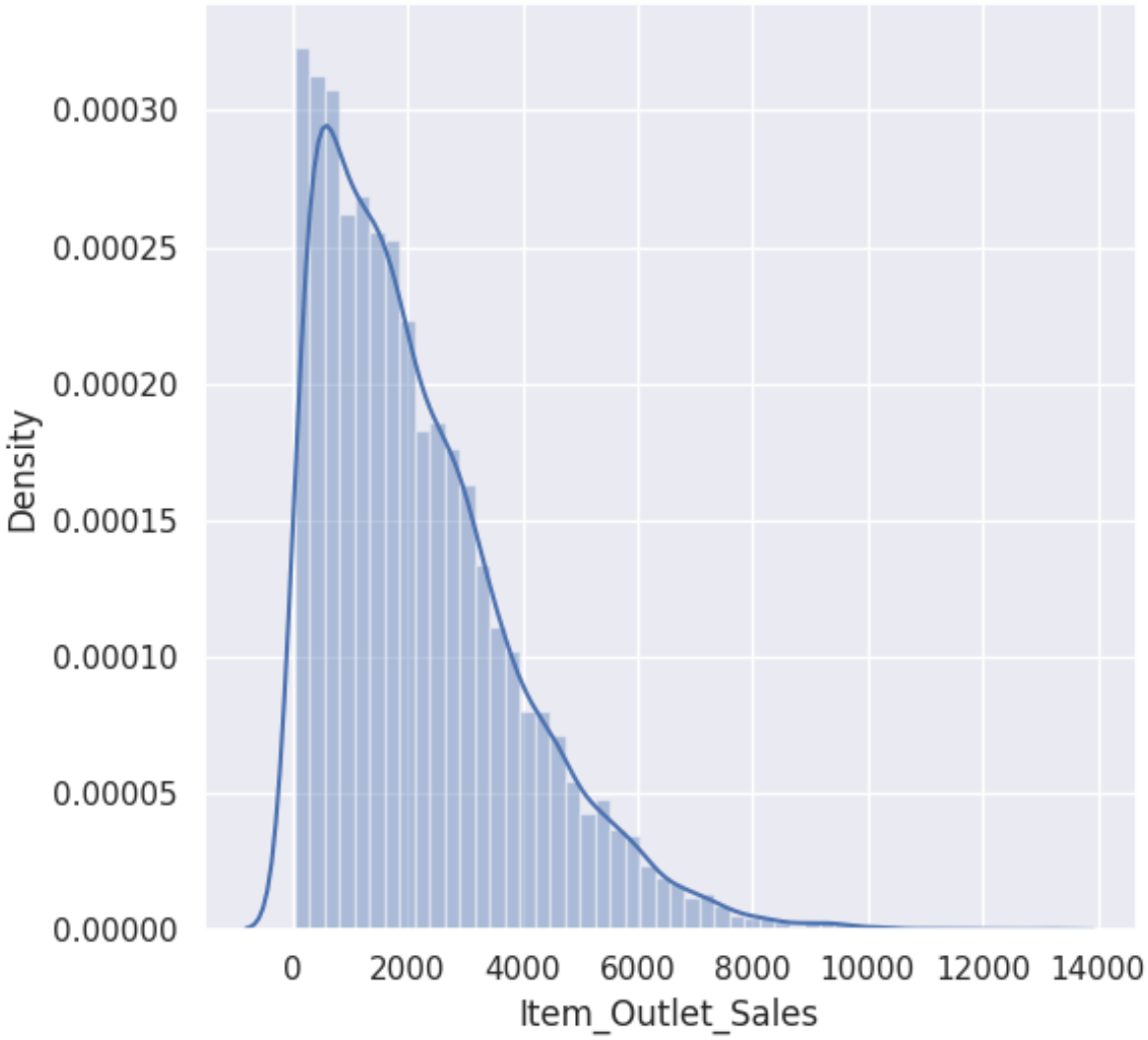
In [20]:
```
# Item Visibility distribution
plt.figure(figsize=(6,6))
sns.distplot(big_mart_data['Item_Visibility'])
plt.show()
```



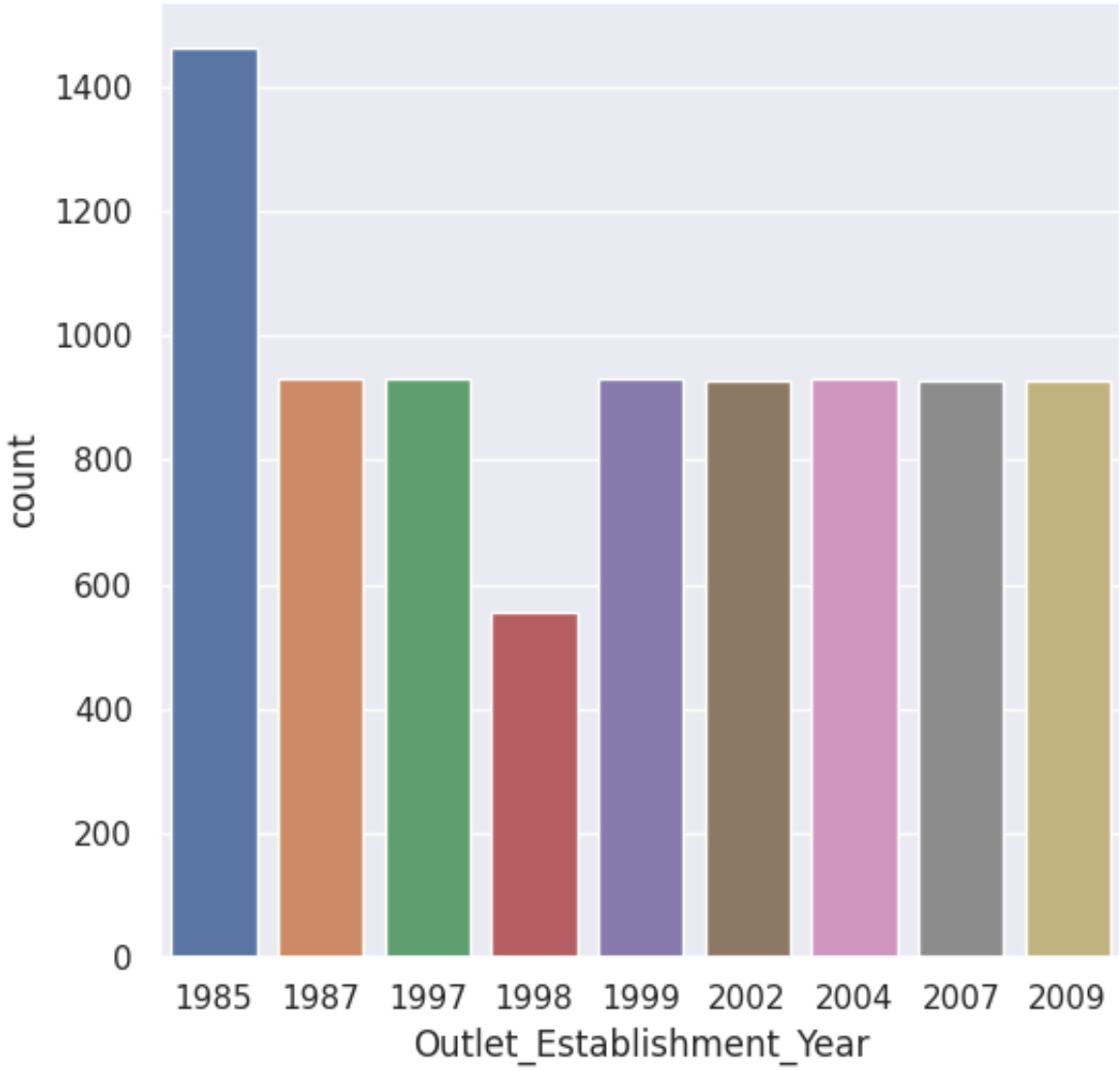
In [21]:
```
# Item MRP distribution
plt.figure(figsize=(6,6))
sns.distplot(big_mart_data['Item_MRP'])
plt.show()
```



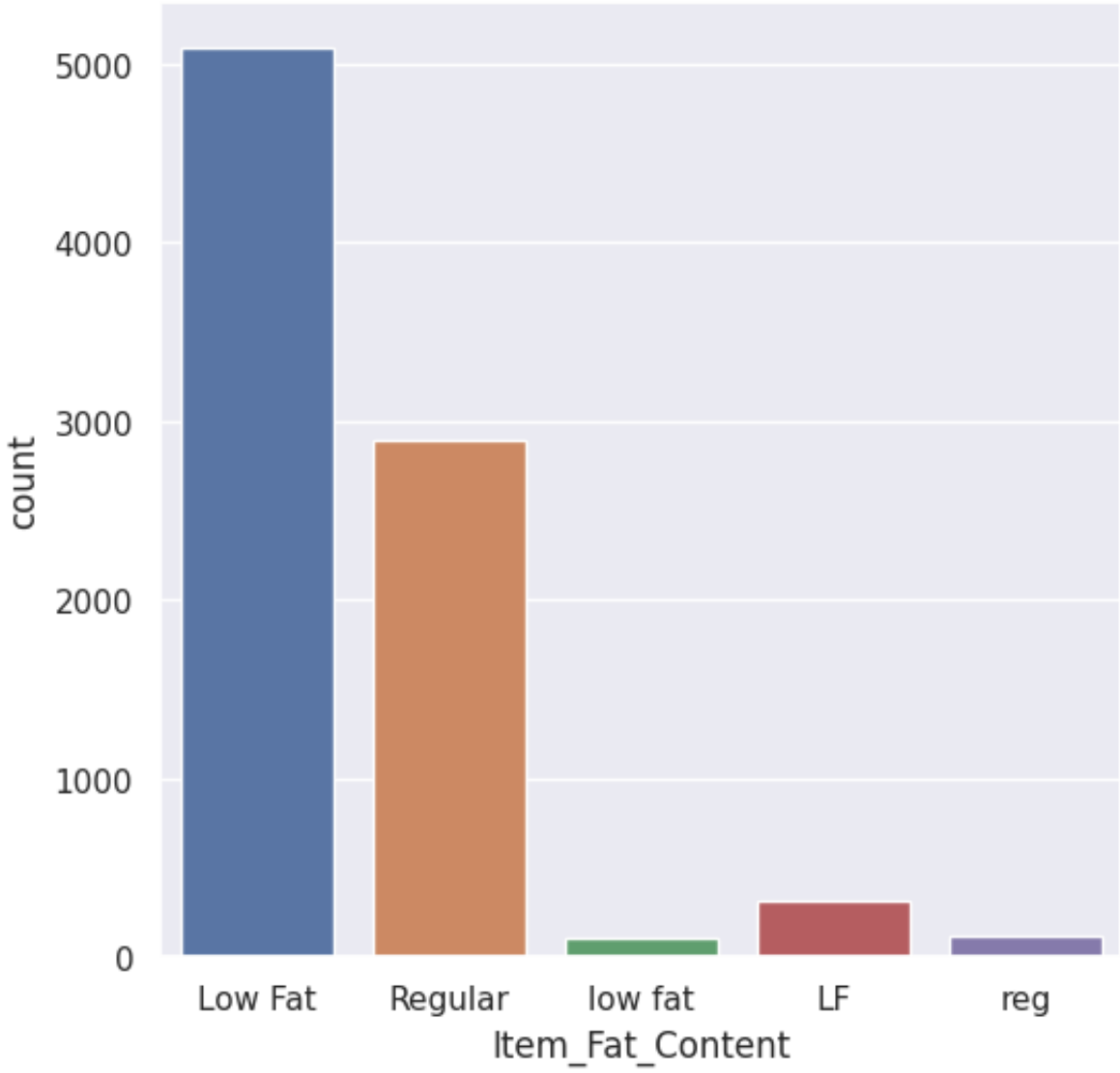
In [22]:
```
# Item_Outlet_Sales distribution
plt.figure(figsize=(6,6))
sns.distplot(big_mart_data['Item_Outlet_Sales'])
plt.show()
```



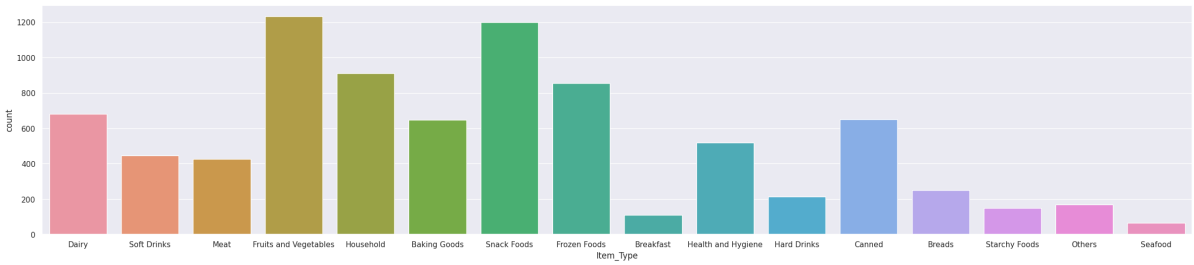
In [23]:
```
# Outlet_Establishment_Year column
plt.figure(figsize=(6,6))
sns.countplot(x='Outlet_Establishment_Year', data=big_mart_data)
plt.show()
```



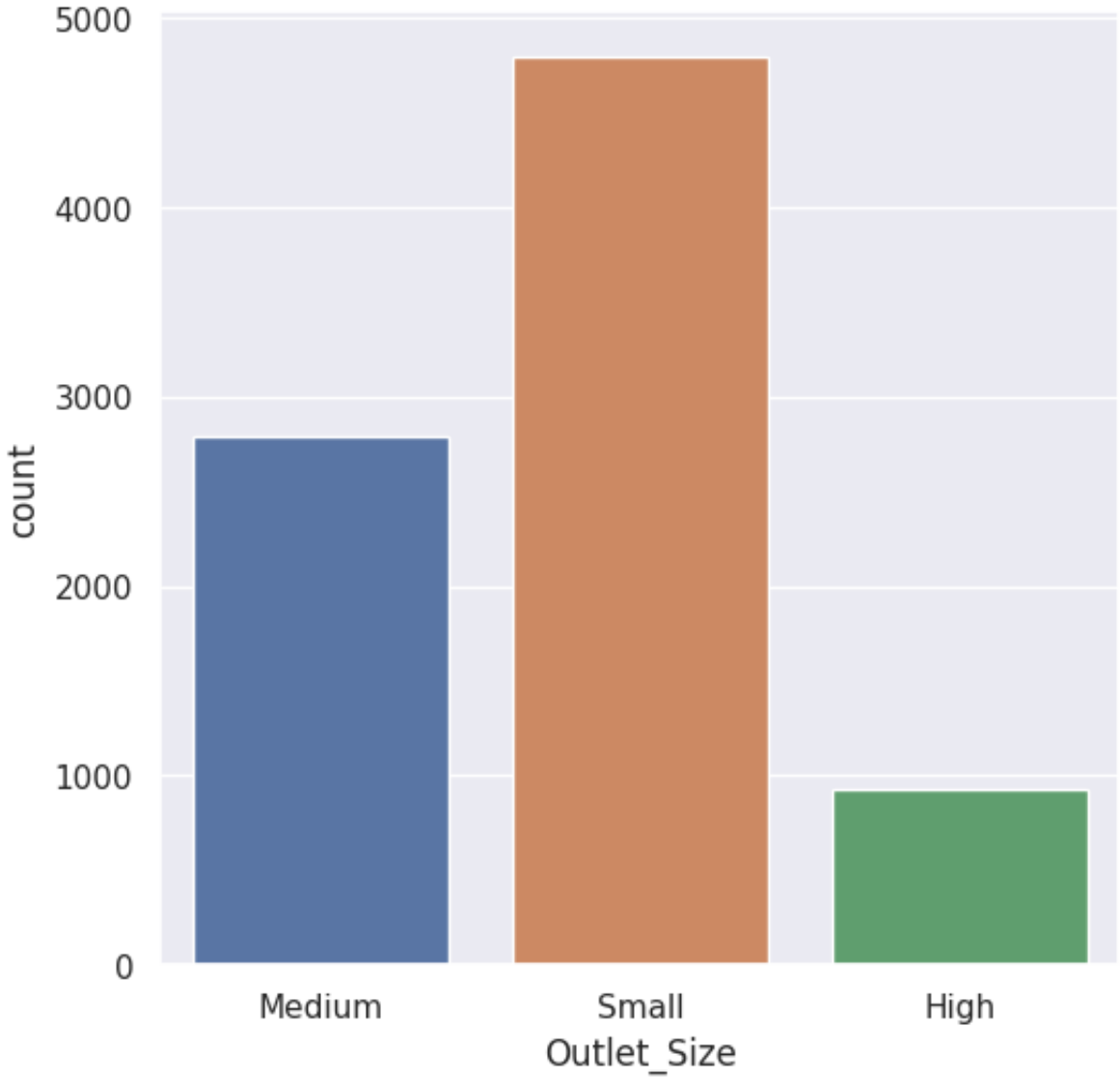
In [24]:
```
# Item_Fat_Content column
plt.figure(figsize=(6,6))
sns.countplot(x='Item_Fat_Content', data=big_mart_data)
plt.show()
```



In [25]:
```
# Item_Type column
plt.figure(figsize=(30,6))
sns.countplot(x='Item_Type', data=big_mart_data)
plt.show()
```



In [26]:
```
# Outlet_Size column
plt.figure(figsize=(6,6))
sns.countplot(x='Outlet_Size', data=big_mart_data)
plt.show()
```



Data Pre-Processing

In [27]:
```
big_mart_data.head()
```

Out[27]:

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

In [28]:
```
big_mart_data['Item_Fat_Content'].value_counts()
```

Out[28]:
```
Item_Fat_Content
Low Fat    5089
Regular    2889
LF          316
reg         117
low fat     112
Name: count, dtype: int64
```

In [29]:
```
big_mart_data.replace({'Item_Fat_Content': {'low fat':'Low Fat','LF
```

In [30]:
```
big_mart_data['Item_Fat_Content'].value_counts()
```

Out[30]:
```
Item_Fat_Content
Low Fat    5517
Regular    3006
Name: count, dtype: int64
```

Label Encoding

In [31]:
```
encoder = LabelEncoder()
```

In [32]:
```
big_mart_data['Item_Identifier'] = encoder.fit_transform(big_mart_d
big_mart_data['Item_Fat_Content'] = encoder.fit_transform(big_mart_
big_mart_data['Item_Type'] = encoder.fit_transform(big_mart_data['I
big_mart_data['Outlet_Identifier'] = encoder.fit_transform(big_mart
big_mart_data['Outlet_Size'] = encoder.fit_transform(big_mart_data[
big_mart_data['Outlet_Location_Type'] = encoder.fit_transform(big_m
big_mart_data['Outlet_Type'] = encoder.fit_transform(big_mart_data[
```

In [33]:
```
big_mart_data.head()
```

Out[33]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Out
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	

Splitting features and Target

In [34]:
```
X = big_mart_data.drop(columns='Item_Outlet_Sales', axis=1)
Y = big_mart_data['Item_Outlet_Sales']
```

In [35]:
```
print(X)
```

```
      Item_Identifier  Item_Weight  Item_Fat_Content  Item_Visibil
ity  \
0                 156        9.300                 0         0.016
047
1                   8        5.920                 1         0.019
278
2                 662       17.500                 0         0.016
760
3                1121       19.200                 1         0.000
000
4                1297        8.930                 0         0.000
000
...               ...          ...               ...
...
8518              370        6.865                 0         0.056
783
```

```
8519              897        8.380                 1         0.046
982
8520             1357       10.600                 0         0.035
186
8521              681        7.210                 1         0.145
221
8522               50       14.800                 0         0.044
878

      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 [36]:
```
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
```

Splitting the data into Training data & Testing Data

In [37]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size
```

In [38]:
```
print(X.shape, X_train.shape, X_test.shape)
```

```
(8523, 11) (6818, 11) (1705, 11)
```

Machine Learning Model Training

XGBoost Regressor

In [39]:
```
regressor = XGBRegressor()
```

In [40]:
```
regressor.fit(X_train, Y_train)
```

Out[40]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_ro
unds=None,
             enable_categorical=False, eval_metric=None, feature_t
ypes=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_constrai
nts=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Evaluation

In [41]:
```
# prediction on training data
training_data_prediction = regressor.predict(X_train)
```

In [42]:
```
# R squared Value
r2_train = metrics.r2_score(Y_train, training_data_prediction)
```

In [43]:
```
print('R Squared value = ', r2_train)
```

```
R Squared value =  0.8762174618111388
```

In [44]:
```
# prediction on test data
test_data_prediction = regressor.predict(X_test)
```

In [45]:
```
# R squared Value
r2_test = metrics.r2_score(Y_test, test_data_prediction)
```

In [46]:
```
print('R Squared value = ', r2_test)
```

```
R Squared value =  0.5017253991620692
```

Making a predictive system

In [49]:
```
import numpy as np

# Example input data for prediction
# The input data should match the order and types of features used
# Let's assume the features in your model are in the following orde
# (Item_Identifier, Item_Weight, Item_Fat_Content, Item_Visibility,
#  Outlet_Identifier, Outlet_Establishment_Year, Outlet_Size, Outle

input_data = (1121, 19.20, 1, 0.000000, 6, 182.0950, 0, 1998, 2, 2,

# Convert the input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the data as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1, -1)

# Make prediction using the trained model
prediction = regressor.predict(input_data_reshaped)

print(f'Predicted Item Outlet Sales: {prediction[0]:.2f}')
```

```
Predicted Item Outlet Sales: 712.16
```

In [51]:
```
# Extract the values at index 3 and convert them to a list
index_list = big_mart_data.iloc[2].tolist()

# Print the list
print(index_list)
```

```
[662.0, 17.5, 0.0, 0.016760075, 10.0, 141.618, 9.0, 1999.0, 1.0,
0.0, 1.0, 2097.27]
```

In [52]:
```
import numpy as np

# Example input data for prediction
# The input data should match the order and types of features used
# Let's assume the features in your model are in the following orde
# (Item_Identifier, Item_Weight, Item_Fat_Content, Item_Visibility,
#  Outlet_Identifier, Outlet_Establishment_Year, Outlet_Size, Outle

input_data = (662.0, 17.5, 0.0, 0.016760075, 10.0, 141.618, 9.0, 19

# Convert the input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# Reshape the data as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1, -1)

# Make prediction using the trained model
prediction = regressor.predict(input_data_reshaped)

print(f'Predicted Item Outlet Sales: {prediction[0]:.2f}')
```

```
Predicted Item Outlet Sales: 2342.50
```

Save the trained model

In [53]:
```
import pickle
# Saving the model
model_filename = 'big_mart_sales_model.pkl'
with open(model_filename, 'wb') as file:
    pickle.dump(regressor, file)

print(f"Model saved to {model_filename}")
```

```
Model saved to big_mart_sales_model.pkl
```