In [1]:
```
# Importing pandas for data manipulation
import pandas as pd

# Importing numpy for numerical operations
import numpy as np


# Importing matplotlib for data visualization
import matplotlib.pyplot as plt

# Importing seaborn for advanced data visualization
import seaborn as sns
```

In [2]:
```
# Loading the test dataset from a CSV file
test = pd.read_csv('Test.csv')

# Loading the train dataset from a CSV file
train = pd.read_csv('Train.csv')
```

In [3]:
```
#view the head of test dataset
test.head()
```

Out[3]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Iden
0	FDW58	20.750	Low Fat	0.007565	Snack Foods	107.8622	OU
1	FDW14	8.300	reg	0.038428	Dairy	87.3198	OU
2	NCN55	14.600	Low Fat	0.099575	Others	241.7538	OU
3	FDQ58	7.315	Low Fat	0.015388	Snack Foods	155.0340	OU
4	FDY38	NaN	Regular	0.118599	Dairy	234.2300	OU

In [4]:
```
#view the head of train dataset
train.head()
```

Out[4]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Iden
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 [5]:
```
#check the shape of both test and train datasets
train.shape, test.shape
```

Out[5]:
```
((8523, 12), (5681, 11))
```

In [6]:
```
#View the columns in both dataset
test.columns, train.columns
```

Out[6]:
```
(Index(['Item_Identifier', 'Item_Weight', 'Item_Fat_Content', 'Item_Visibilit
y',
        'Item_Type', 'Item_MRP', 'Outlet_Identifier',
        'Outlet_Establishment_Year', 'Outlet_Size', 'Outlet_Location_Type',
        'Outlet_Type'],
       dtype='object'),
 Index(['Item_Identifier', 'Item_Weight', 'Item_Fat_Content', 'Item_Visibilit
y',
        'Item_Type', 'Item_MRP', 'Outlet_Identifier',
        'Outlet_Establishment_Year', 'Outlet_Size', 'Outlet_Location_Type',
        'Outlet_Type', 'Item_Outlet_Sales'],
       dtype='object'))
```

In [7]:
```
train.describe()
```

Out[7]:

	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 [8]:
```
test.describe()
```

Out[8]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year
count	4705.000000	5681.000000	5681.000000	5681.000000
mean	12.695633	0.065684	141.023273	1997.828903
std	4.664849	0.051252	61.809091	8.372256
min	4.555000	0.000000	31.990000	1985.000000
25%	8.645000	0.027047	94.412000	1987.000000
50%	12.500000	0.054154	141.415400	1999.000000
75%	16.700000	0.093463	186.026600	2004.000000
max	21.350000	0.323637	266.588400	2009.000000

In [9]:
```
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
```

The dataset has 4 columns with float data type, 1 column with int data type and 7 columns with object data type. The columns "Item_Weight" and "Outlet_Size" have missing values.

In [10]:
```
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5681 entries, 0 to 5680
Data columns (total 11 columns):
 #   Column                     Non-Null Count  Dtype  
---  ------                     --------------  -----  
 0   Item_Identifier            5681 non-null   object 
 1   Item_Weight                4705 non-null   float64
 2   Item_Fat_Content           5681 non-null   object 
 3   Item_Visibility            5681 non-null   float64
 4   Item_Type                  5681 non-null   object 
 5   Item_MRP                   5681 non-null   float64
 6   Outlet_Identifier          5681 non-null   object 
 7   Outlet_Establishment_Year  5681 non-null   int64  
 8   Outlet_Size                4075 non-null   object 
 9   Outlet_Location_Type       5681 non-null   object 
 10  Outlet_Type                5681 non-null   object 
dtypes: float64(3), int64(1), object(7)
memory usage: 488.3+ KB
```

This is a summary of the test dataset which has 5681 entries and 11 columns. The columns are the same as the training dataset except for the Item_Outlet_Sales column which is absent, as it is the target variable that we need to predict using our model. The dataset contains both numerical and categorical features. There are missing values in the Item_Weight, Outlet_Size columns, which need to be handled before modeling.

In [11]:
```
train.isnull().sum()
```

Out[11]:
```
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
```

This output shows the number of null values in each column of the training dataset. There are 1463 null values in the Item_Weight column and 2410 null values in the Outlet_Size column.

In [12]:
```
test.isnull().sum()
```

Out[12]:
```
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
```

This output shows the number of null values in each column of the testing dataset. there are 976 null values in the Item_Weight column and 1606 null values in the Outlet_Size column.

In [13]:
```
# Fill null values in 'Item_Weight' column with the median value of the colu
train['Item_Weight'].fillna(train['Item_Weight'].median(), inplace=True)

# Fill null values in 'Outlet_Size' column with the mode value of the column
train['Outlet_Size'].fillna(train['Outlet_Size'].mode()[0], inplace=True)
```

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

Out[14]:
```
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
```

This output indicates that 'Item_Weight' & 'Outlet_size' columns in the dataset have been imputed with median & mode, resulting in no null values present in any column

In [15]:
```
# Fill null values in 'Item_Weight' column with the median value of the colu
test['Item_Weight'].fillna(test['Item_Weight'].median(), inplace=True)

# Fill null values in 'Outlet_Size' column with the mode value of the column
test['Outlet_Size'].fillna(test['Outlet_Size'].mode()[0], inplace=True)
```

In [16]:
```
test.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
dtype: int64
```

This output indicates that 'Item_Weight' & 'Outlet_size' columns in the dataset have been imputed with median & mode, resulting in no null values present in any column

In [17]:
```
train.Item_Fat_Content.value_counts()
```

Out[17]:
```
Low Fat    5089
Regular    2889
LF          316
reg         117
low fat     112
Name: Item_Fat_Content, dtype: int64
```

In [18]:
```
test.Item_Fat_Content.value_counts()
```

Out[18]:
```
Low Fat    3396
Regular    1935
LF          206
reg          78
low fat      66
Name: Item_Fat_Content, dtype: int64
```

We see there are some irregularities in the column and it is needed to fix them

In [19]:
```
train['Item_Fat_Content'].replace(['low fat', 'LF', 'reg'],['Low Fat', 'Low
test['Item_Fat_Content'].replace(['low fat', 'LF', 'reg'],['Low Fat', 'Low F
```

In [20]:
```
train.Item_Fat_Content.value_counts()
```

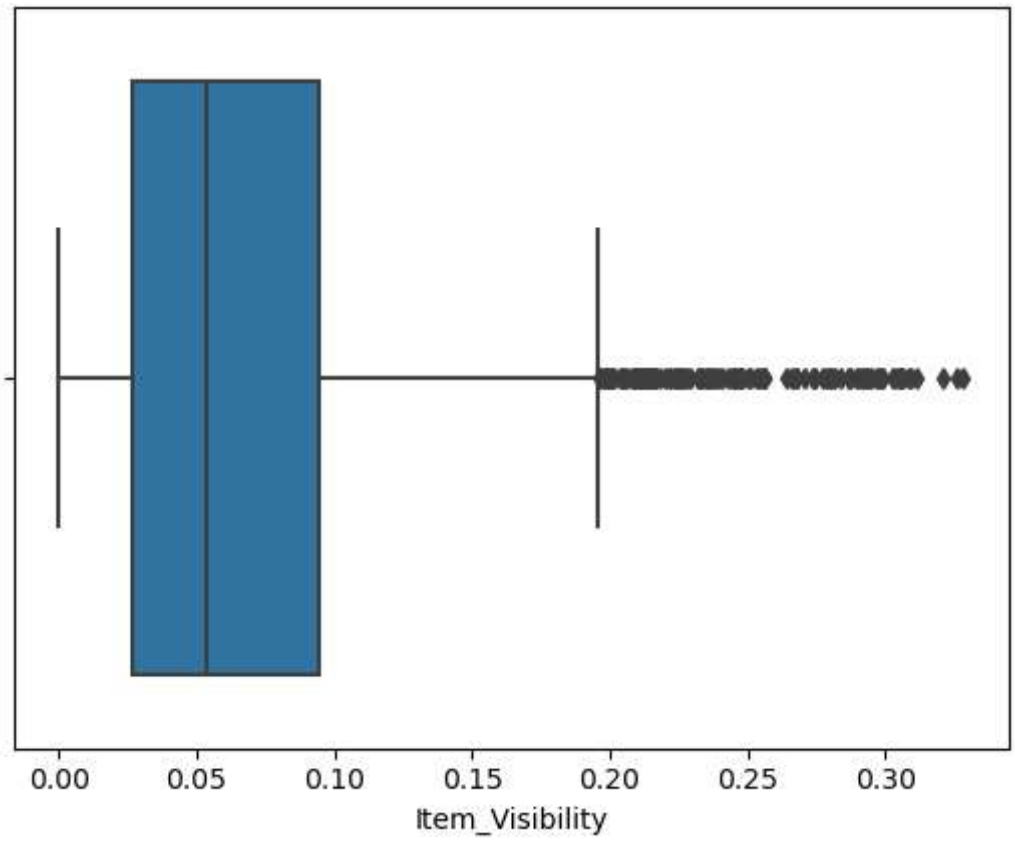
Out[20]:
```
Low Fat    5517
Regular    3006
Name: Item_Fat_Content, dtype: int64
```

In [21]:
```
test.Item_Fat_Content.value_counts()
```

Out[21]:
```
Low Fat    3668
Regular    2013
Name: Item_Fat_Content, dtype: int64
```

In [22]:
```
sns.boxplot(data=train, x='Item_Visibility')
```

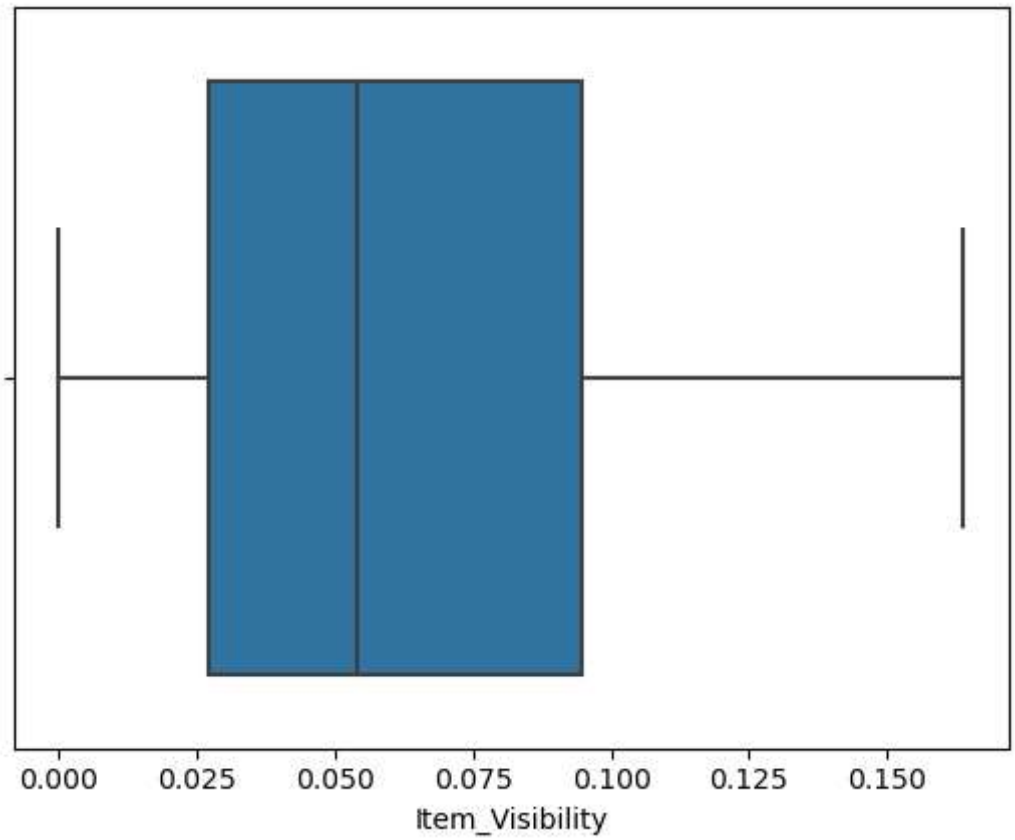
Out[22]: <AxesSubplot:xlabel='Item_Visibility'>



In [23]:
```
from scipy.stats.mstats import winsorize

# Winsorize 'Item_Visibility' column to remove outliers
train['Item_Visibility'] = winsorize(train['Item_Visibility'], limits=(0.05,
```

In [24]:
```
sns.boxplot(data=train, x='Item_Visibility')
```

Out[24]:
```
<AxesSubplot:xlabel='Item_Visibility'>
```



In [25]:
```
train.shape, test.shape
```

Out[25]:
```
((8523, 12), (5681, 11))
```

EDA

In [26]:
```
# Select all columns in "train" that have a data type of "object"
categorical = train.select_dtypes(include =[object])

# Print the number of categorical features and their names
print(categorical.shape[1], "Categorical Features in Train Set are:")
print('\t' + '\n\t'.join(categorical.columns) + "\n")
```

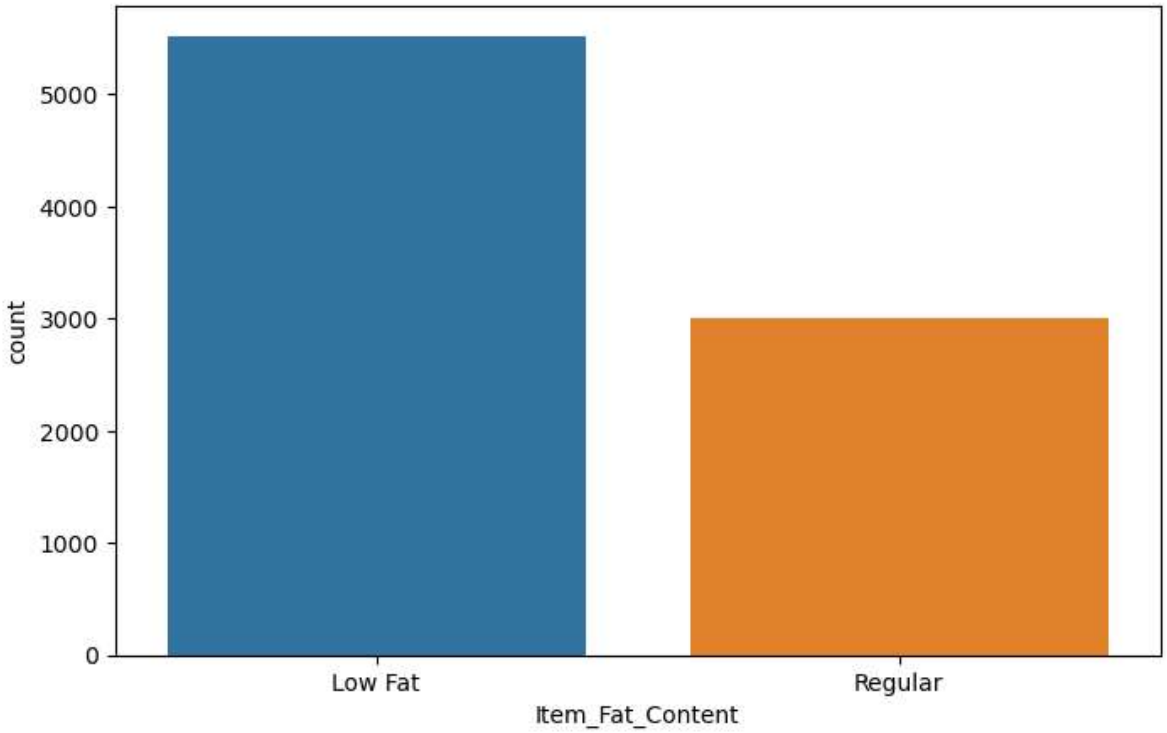
```
7 Categorical Features in Train Set are:
	Item_Identifier
	Item_Fat_Content
	Item_Type
	Outlet_Identifier
	Outlet_Size
	Outlet_Location_Type
	Outlet_Type
```

Categorical Analysis

1.Item_Fat_Content

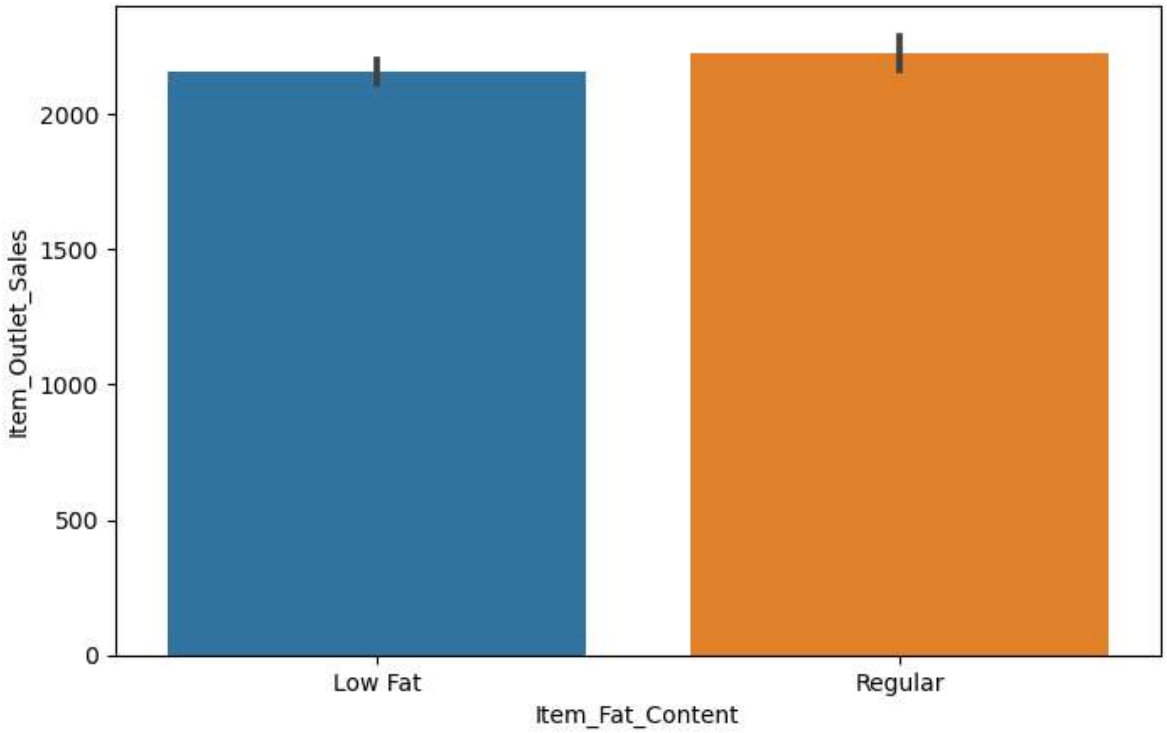
In [27]:
```
plt.figure(figsize=(8,5))
sns.countplot(data=train, x='Item_Fat_Content')
```

Out[27]: <AxesSubplot:xlabel='Item_Fat_Content', ylabel='count'>



In [28]:
```
plt.figure(figsize=(8,5))
sns.barplot(data=train, x='Item_Fat_Content', y='Item_Outlet_Sales')
```

Out[28]: <AxesSubplot:xlabel='Item_Fat_Content', ylabel='Item_Outlet_Sales'>



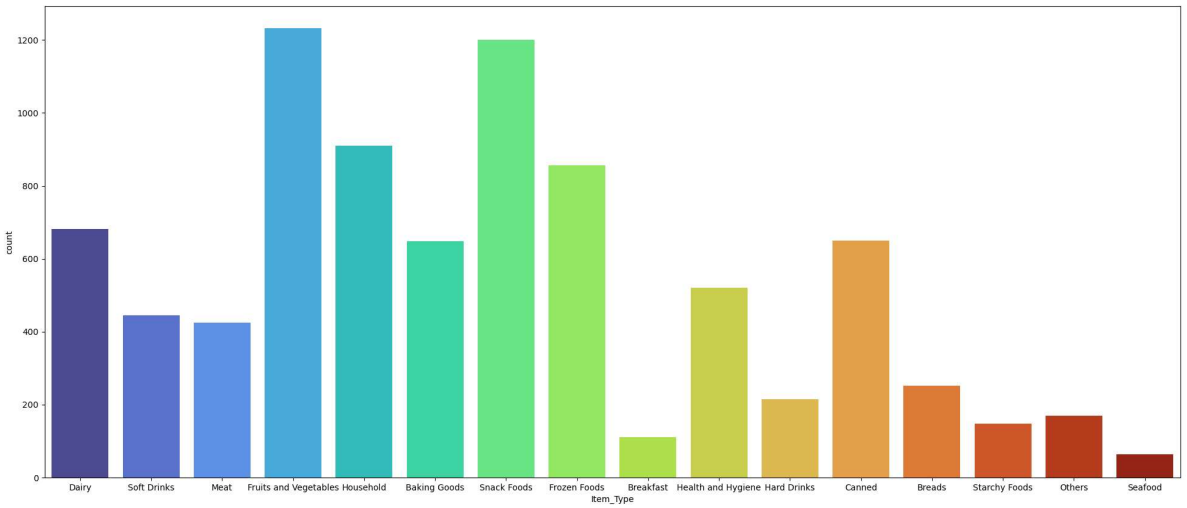
Observations:

The Items bought are more of Low Fat.

But Item Outles sales are almost same for both Low Fat and Regular Item Content

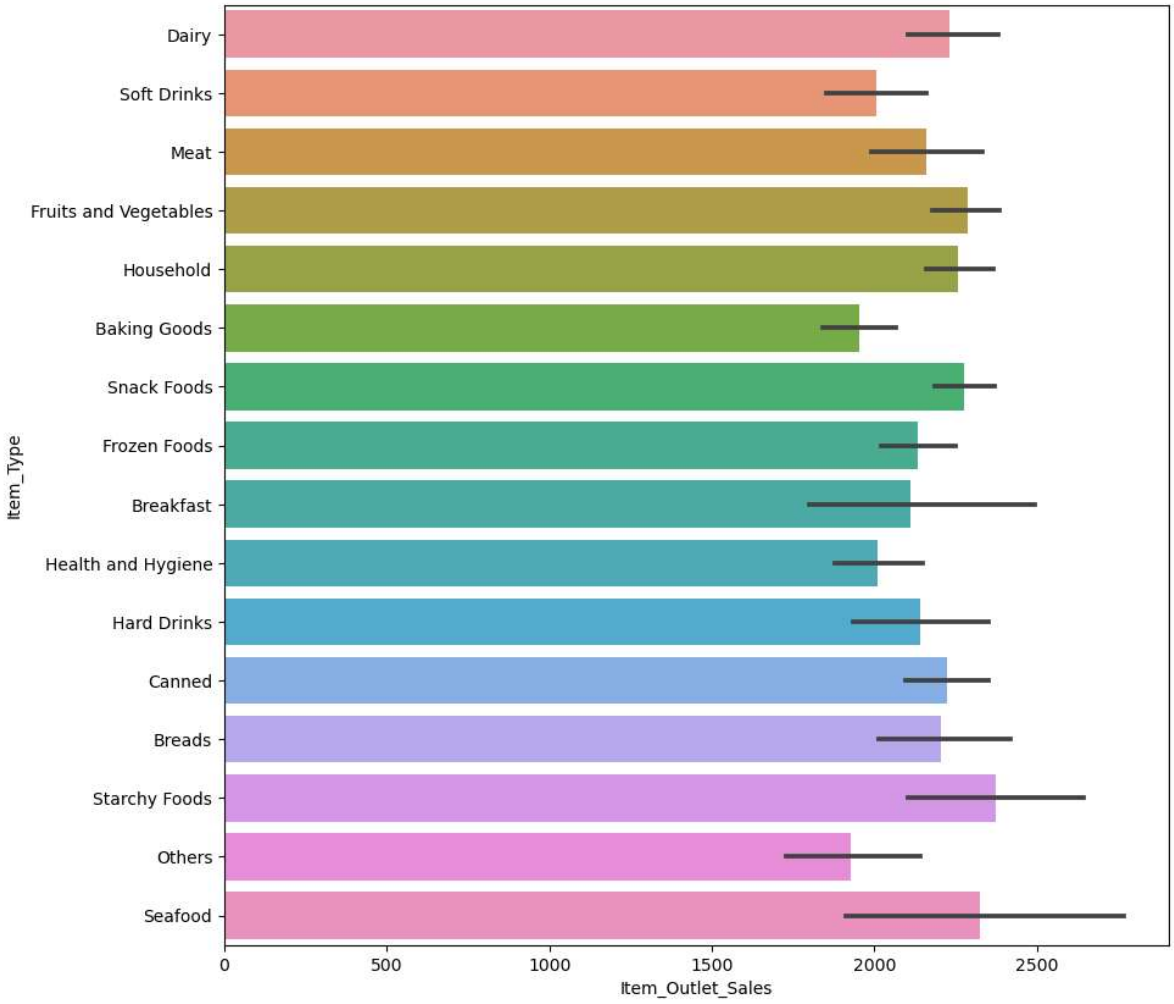
2. Item_Type

In [29]:
```
# Set the figure size to 20x8
plt.figure(figsize=(24,10))

# Create a count plot using Seaborn, and rotate the x-axis labels by 35 degr
sns.countplot(data=train, x='Item_Type', palette='turbo', orient="h")

# Display the plot using Matplotlib
plt.show()
```



In [30]:
```
plt.figure(figsize=(10,10))
sns.barplot(data=train, y='Item_Type', x='Item_Outlet_Sales')
```

Out[30]: <AxesSubplot:xlabel='Item_Outlet_Sales', ylabel='Item_Type'>



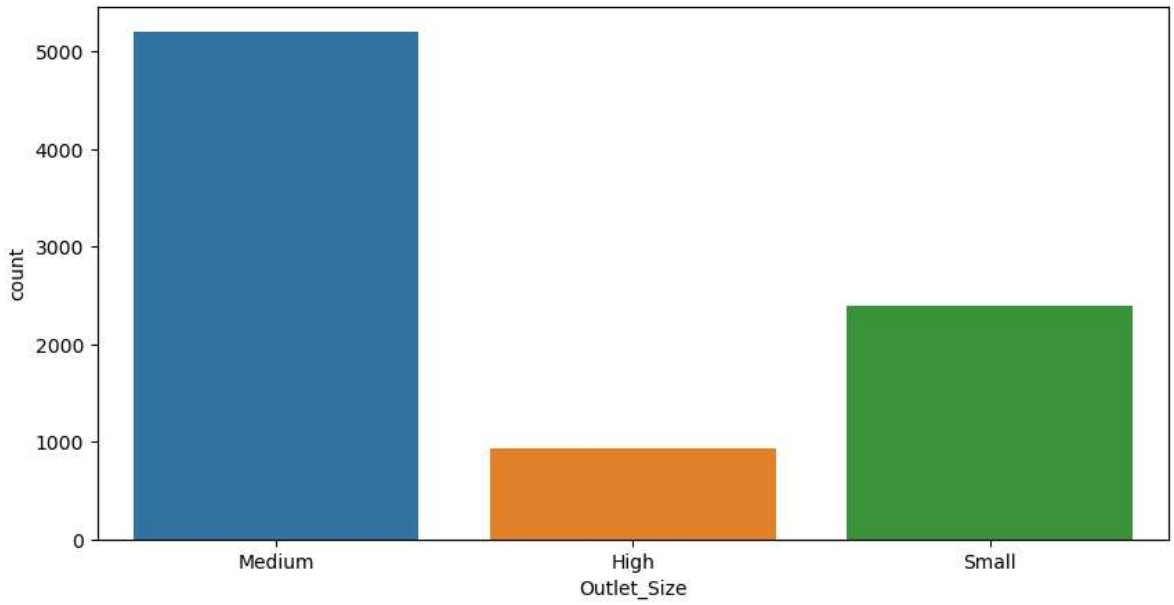
Observations :

The products available were Fruits-Veggies and Snack Foods but the sales of Seafood and Starchy Foods seems higher and hence the sales can be improved with having stock of products that are most bought by customers

3.Outlet_Size

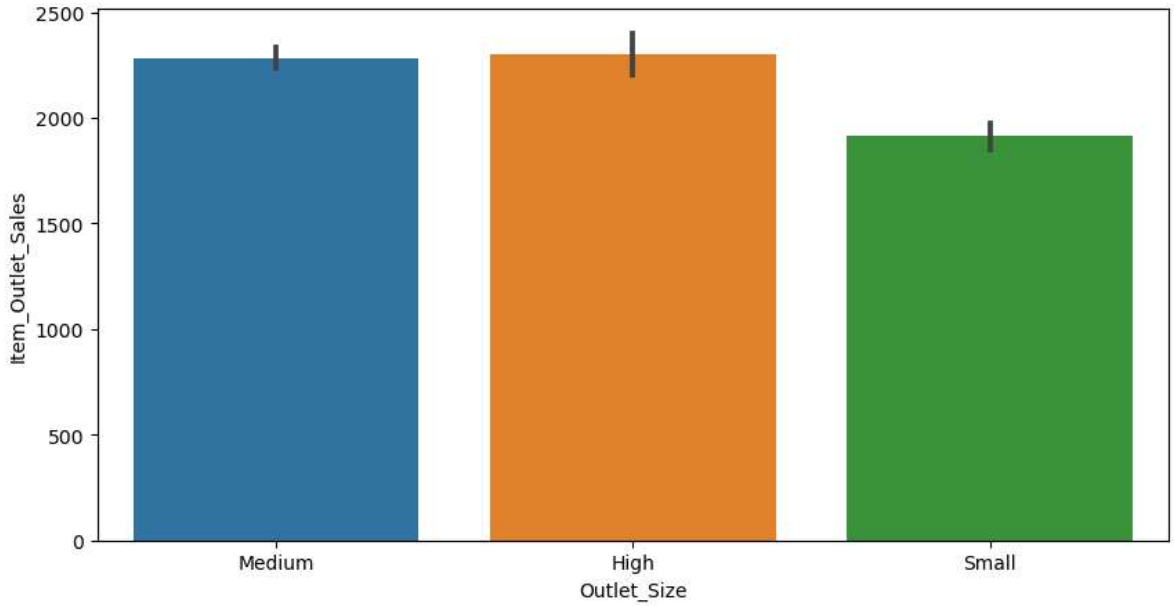
In [31]:
```
plt.figure(figsize=(10,5))
sns.countplot(data=train, x='Outlet_Size')
```

Out[31]: <AxesSubplot:xlabel='Outlet_Size', ylabel='count'>



In [32]:
```
plt.figure(figsize=(10,5))
sns.barplot(data=train, x='Outlet_Size', y='Item_Outlet_Sales')
```

Out[32]: <AxesSubplot:xlabel='Outlet_Size', ylabel='Item_Outlet_Sales'>



Observations:

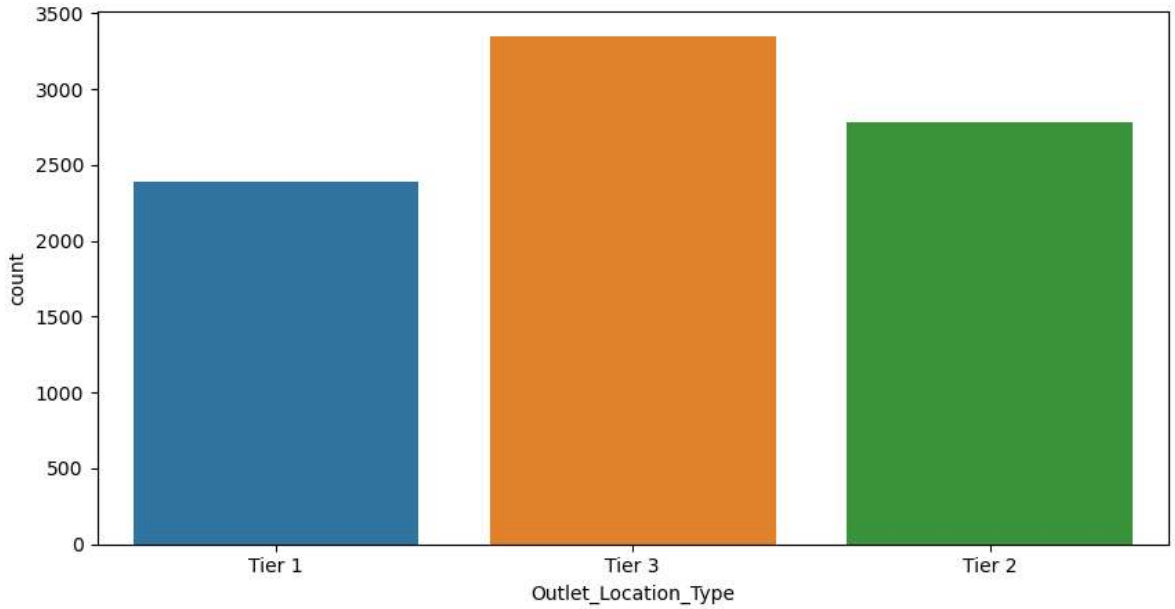
The Outlets are more of Medium Size

But Outlet Sales is maximum for Medium and High sized Outlets so may be with High size Outlets can improve the Outlet Sales.

4. Outlet_Location_Type

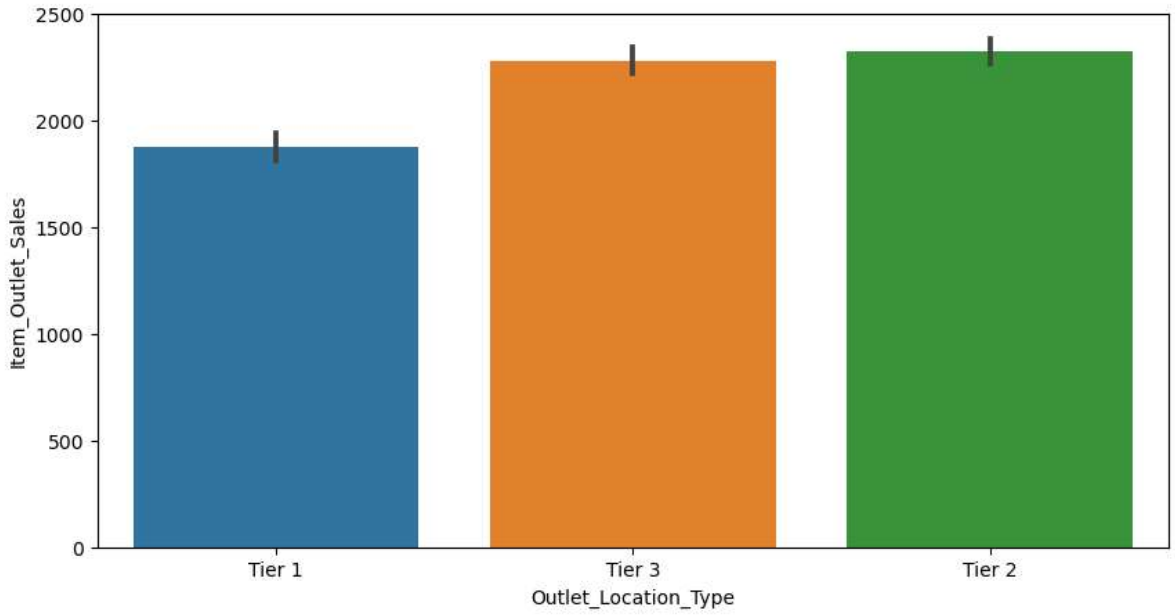
In [33]:
```
plt.figure(figsize=(10,5))
sns.countplot(data=train, x='Outlet_Location_Type')
```

Out[33]:
```
<AxesSubplot:xlabel='Outlet_Location_Type', ylabel='count'>
```



In [34]:
```
plt.figure(figsize=(10,5))
sns.barplot(data=train, x='Outlet_Location_Type', y='Item_Outlet_Sales')
```

Out[34]: <AxesSubplot:xlabel='Outlet_Location_Type', ylabel='Item_Outlet_Sales'>



Observations:

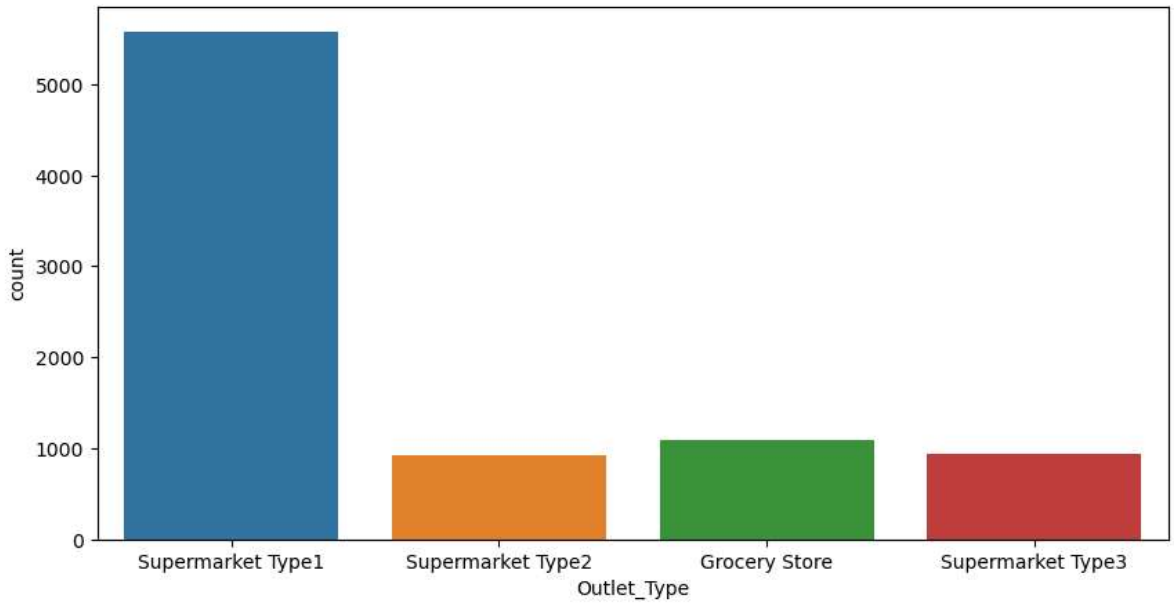
The Outlet Sales tend to be high for Tier3 and Tier 2 location types

But we have only Tier3 locations maximum Outlets

5. Outlet_Type

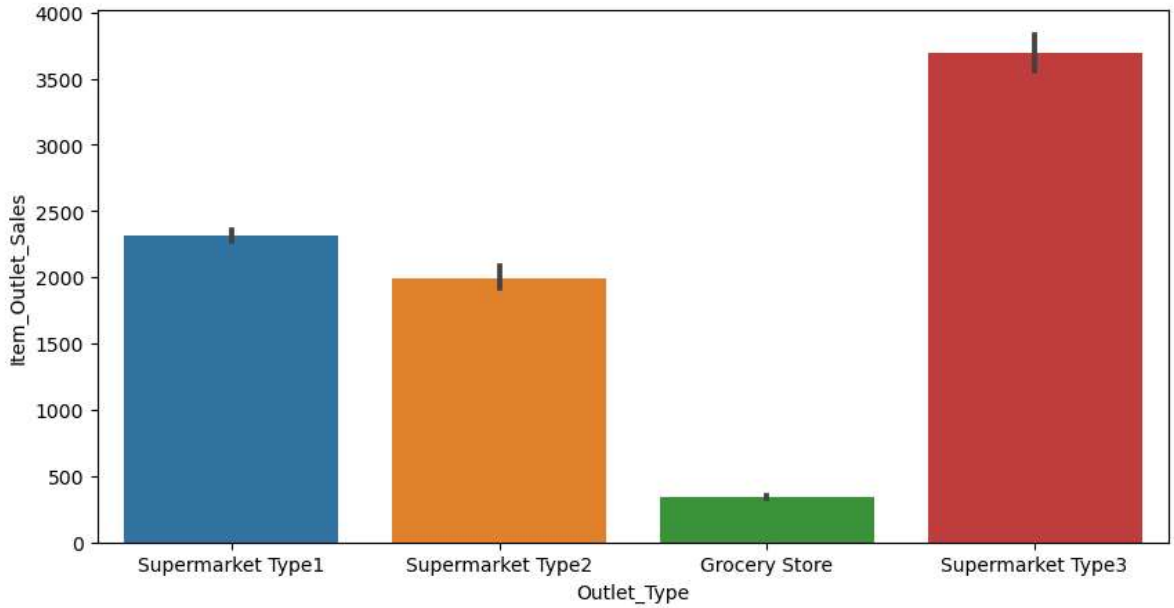
In [35]:
```
plt.figure(figsize=(10,5))
sns.countplot(data=train, x='Outlet_Type')
```

Out[35]: <AxesSubplot:xlabel='Outlet_Type', ylabel='count'>



In [36]:
```
plt.figure(figsize=(10,5))
sns.barplot(data=train, x='Outlet_Type', y='Item_Outlet_Sales')
```

Out[36]: <AxesSubplot:xlabel='Outlet_Type', ylabel='Item_Outlet_Sales'>

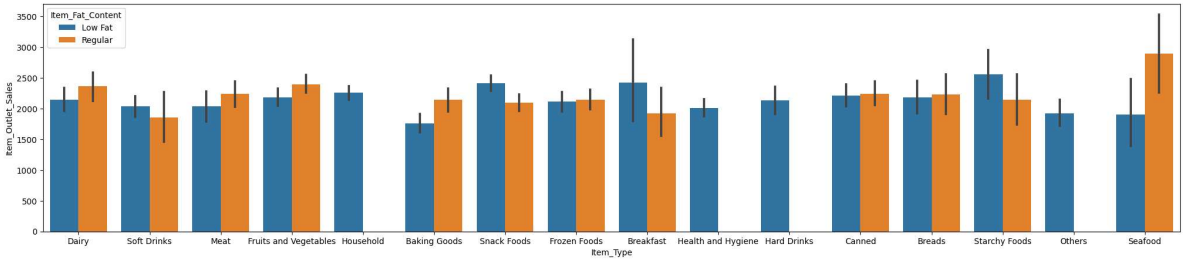


Observations:

The Outlets are more of Supermarket Type1.

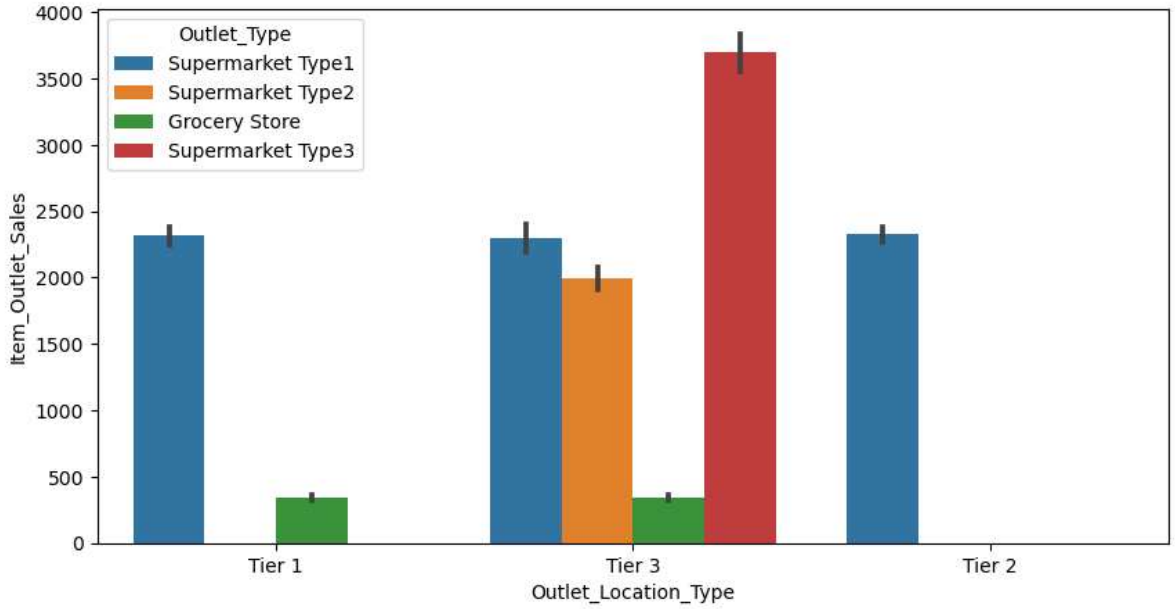
But sales are more on Type 3

In [37]:
```
plt.figure(figsize=(25,5))
sns.barplot(x='Item_Type',y='Item_Outlet_Sales',hue='Item_Fat_Content',data=
plt.show()
```



In [38]:
```
plt.figure(figsize=(10,5))
sns.barplot(x='Outlet_Location_Type', y='Item_Outlet_Sales',hue='Outlet_Type
```

Out[38]: <AxesSubplot:xlabel='Outlet_Location_Type', ylabel='Item_Outlet_Sales'>



Observations:

The Tier-3 location type has all types of Outlet type and has high sales margin

Analysis on Numerical Features

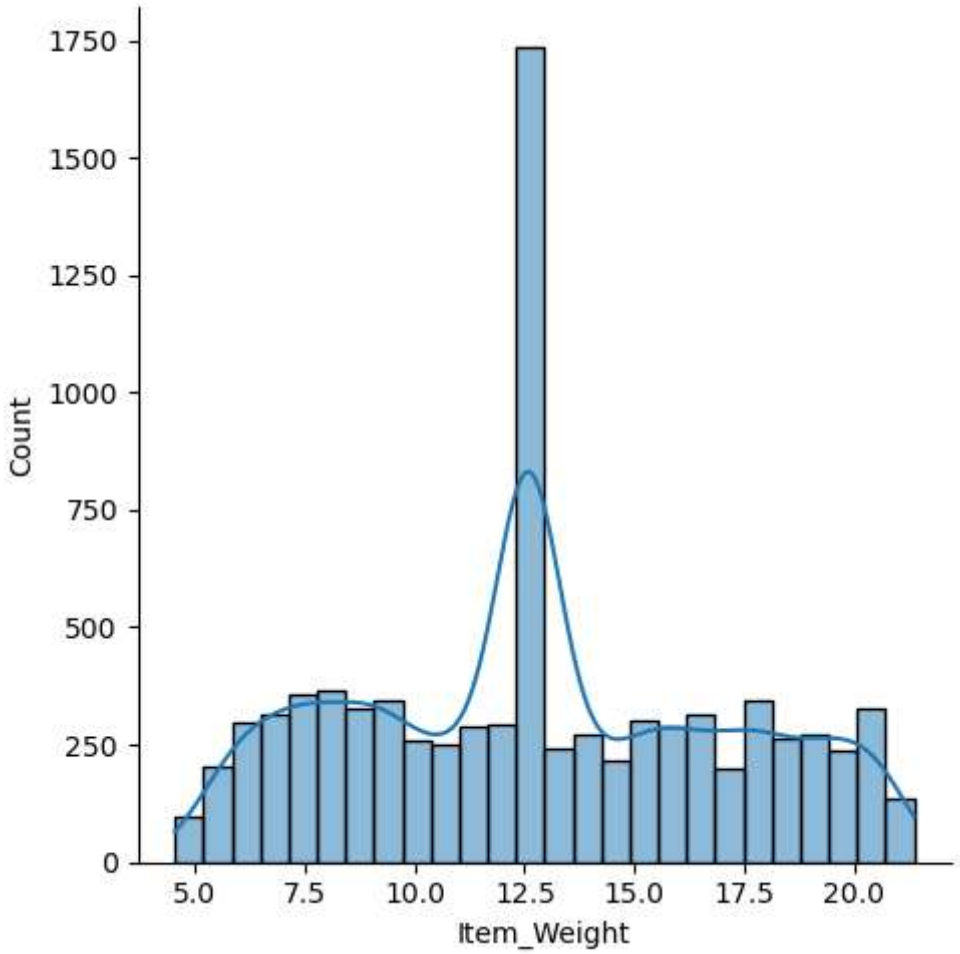
In [39]:
```
# Selecting only numerical columns
num_cols = train.select_dtypes(include=['float64', 'int64']).columns.tolist(

# Printing the numerical column names
print("Numerical features: ")
for col in num_cols:
    print(col)
```

```
Numerical features:
Item_Weight
Item_Visibility
Item_MRP
Outlet_Establishment_Year
Item_Outlet_Sales
```

1. Item Weight

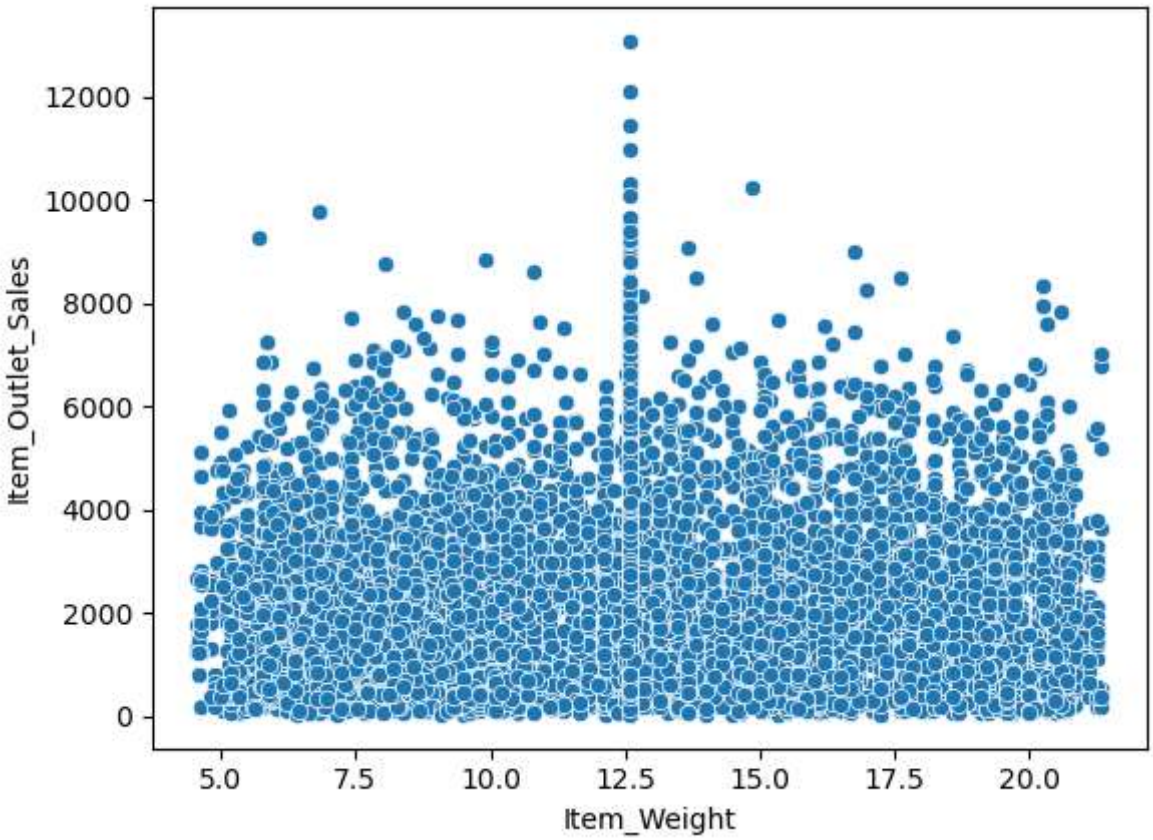
In [40]:
```
sns.displot(data=train, x=train.Item_Weight, kde=True)
```

Out[40]: <seaborn.axisgrid.FacetGrid at 0x28ec8889370>



In [41]:
```
sns.scatterplot(data=train, x='Item_Weight', y='Item_Outlet_Sales')
```

Out[41]: <AxesSubplot:xlabel='Item_Weight', ylabel='Item_Outlet_Sales'>



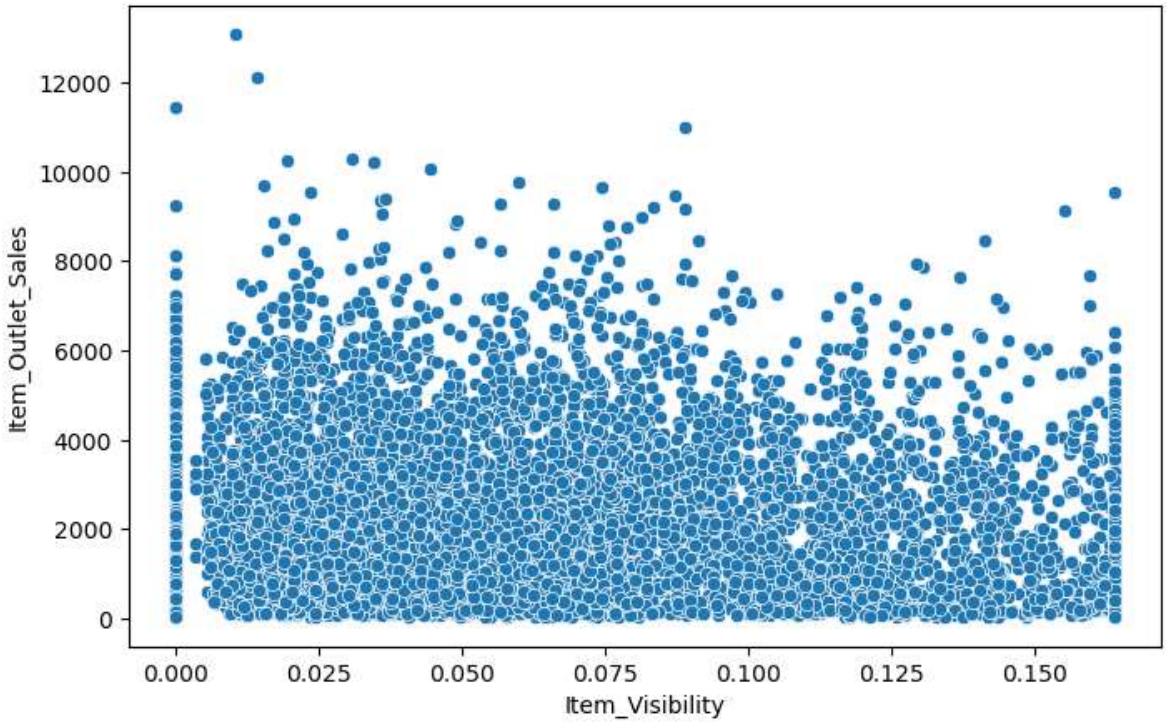
Observations:

We have more products of weight around 12.5, and sell is maximum for that weight

2. Item_Visibility

In [42]:
```
plt.figure(figsize=(8,5))
sns.scatterplot(data=train, x='Item_Visibility',y='Item_Outlet_Sales')
```

Out[42]: <AxesSubplot:xlabel='Item_Visibility', ylabel='Item_Outlet_Sales'>



In [43]:
```
train.Item_Visibility.min()
```

Out[43]: 0.0

Item_Visibility has a minimum value of zero. This makes no practical sense because when a product is being sold in a store, the visibility cannot be 0.

In [44]:
```
train['Item_Visibility'].mean()
```

Out[44]: 0.06435454226011966

In [45]:
```
train['Item_Visibility'].median()
```

Out[45]: 0.053930934

In [46]:
```
train['Item_Visibility'].mode()
```

Out[46]:
```
0    0.0
Name: Item_Visibility, dtype: float64
```

Lets consider it like missing information and impute it with median (as their are outlier) visibility of that product

In [47]:
```
train['Item_Visibility']=train['Item_Visibility'].replace(0, train['Item_Vis
test['Item_Visibility']=test['Item_Visibility'].replace(0, test['Item_Visibi
```

replace any occurrence of the value '0' in the 'Item_Visibility' column of a 'train' with the median value of the 'Item_Visibility' column.

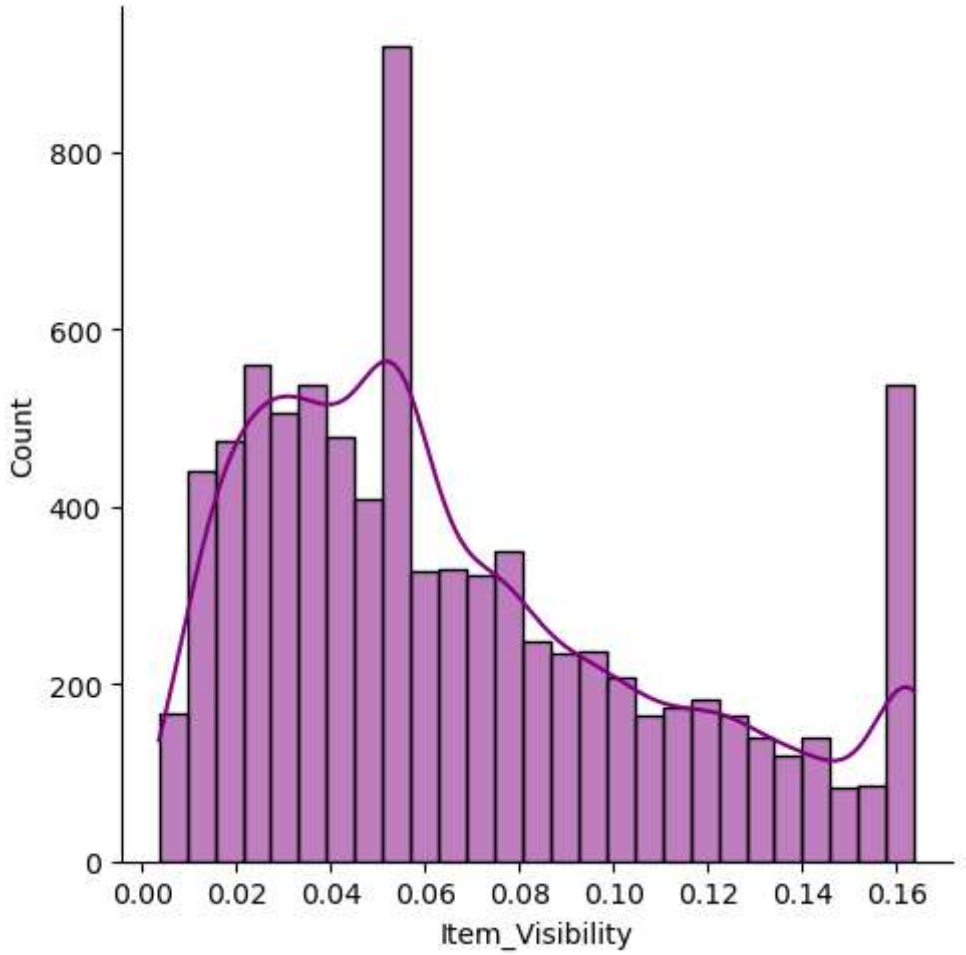
In [48]:
```
train.Item_Visibility.min()
```

Out[48]:
```
0.003574698
```

We can see that now visibility is not exactly zero and it has some value indicating that Item is rarely purchased by the customers

In [49]:
```
sns.displot(x=train.Item_Visibility, color='purple', kde=True)
```

Out[49]:
```
<seaborn.axisgrid.FacetGrid at 0x28ec6865610>
```

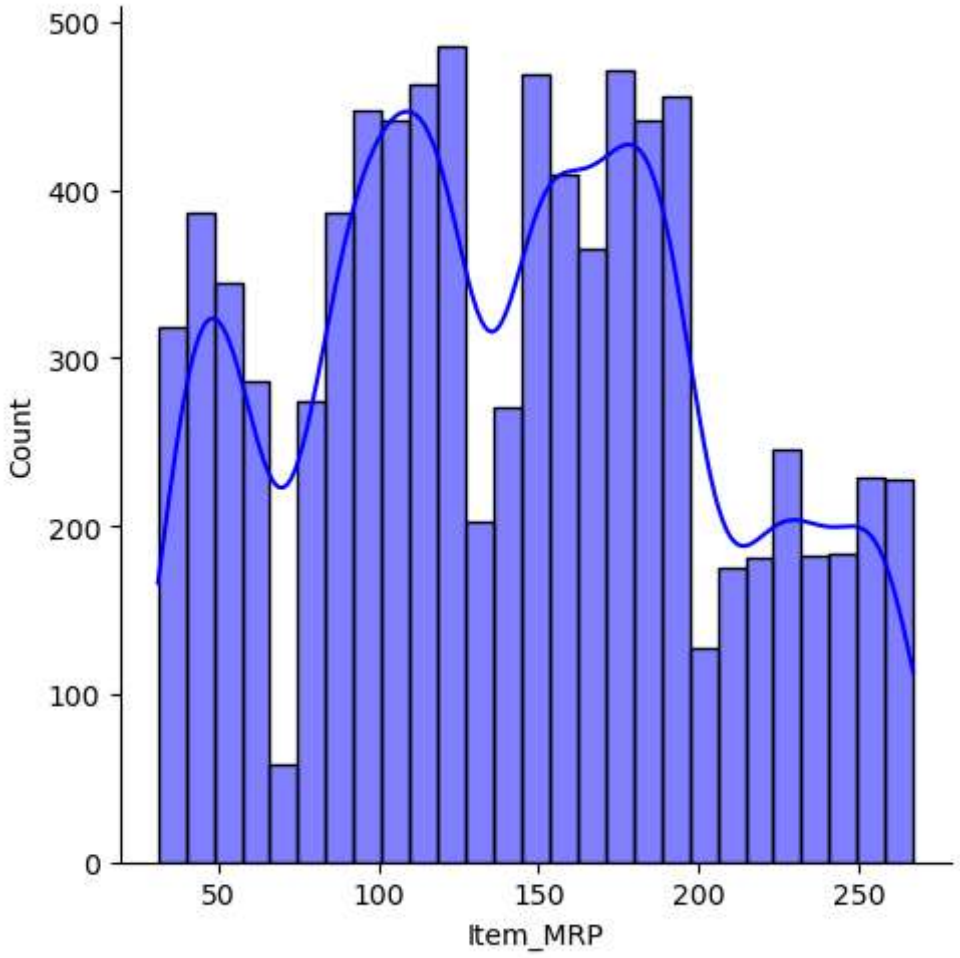


Observation:

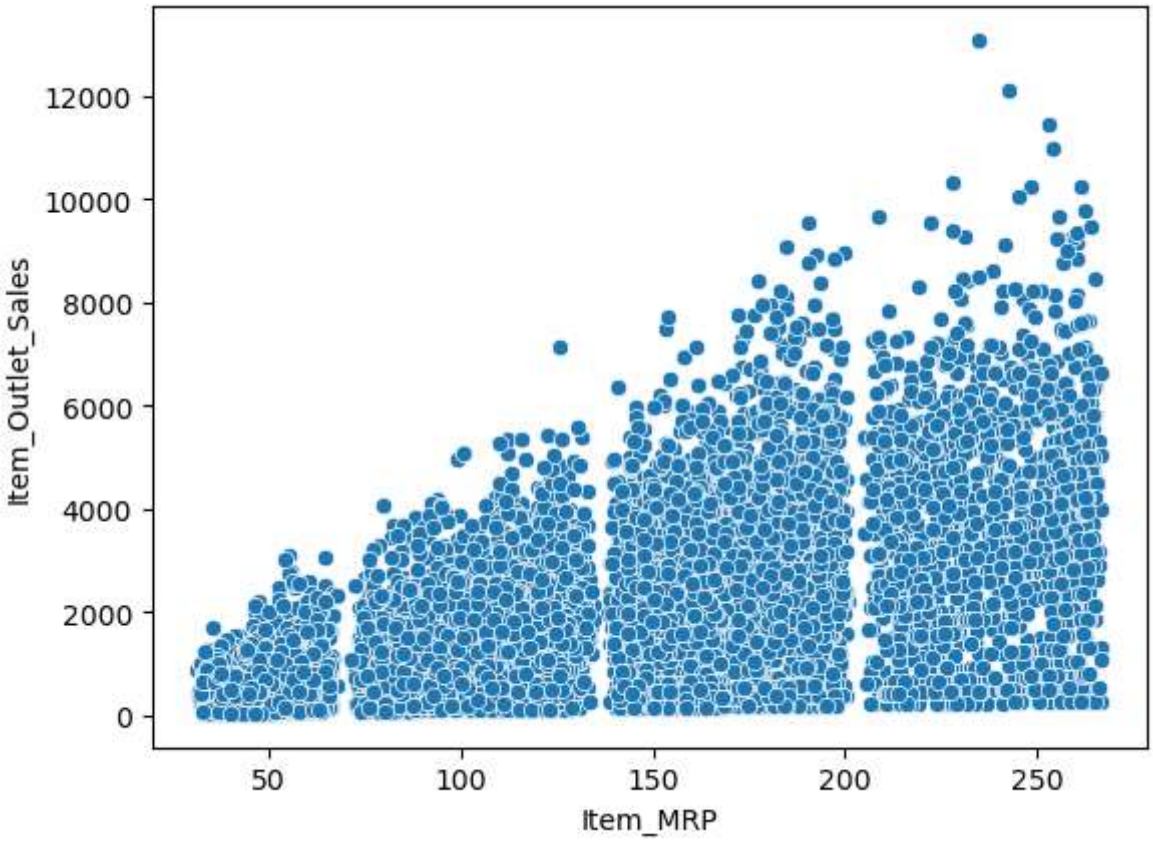
1. We have Items having Visibility 0 to 0.2 is more.
2. And Items having Visibility around 0.05 is maximum.
3. Sales is more for Items having Visibility 0 to 0.2
4. Positive skewness

3. Item MRP

In [50]:
```
sns.displot(x=train.Item_MRP, color='blue', kde=True)
plt.show()
```



In [51]:
```
sns.scatterplot(x='Item_MRP',y='Item_Outlet_Sales',data=train)
plt.show()
```



Observations:

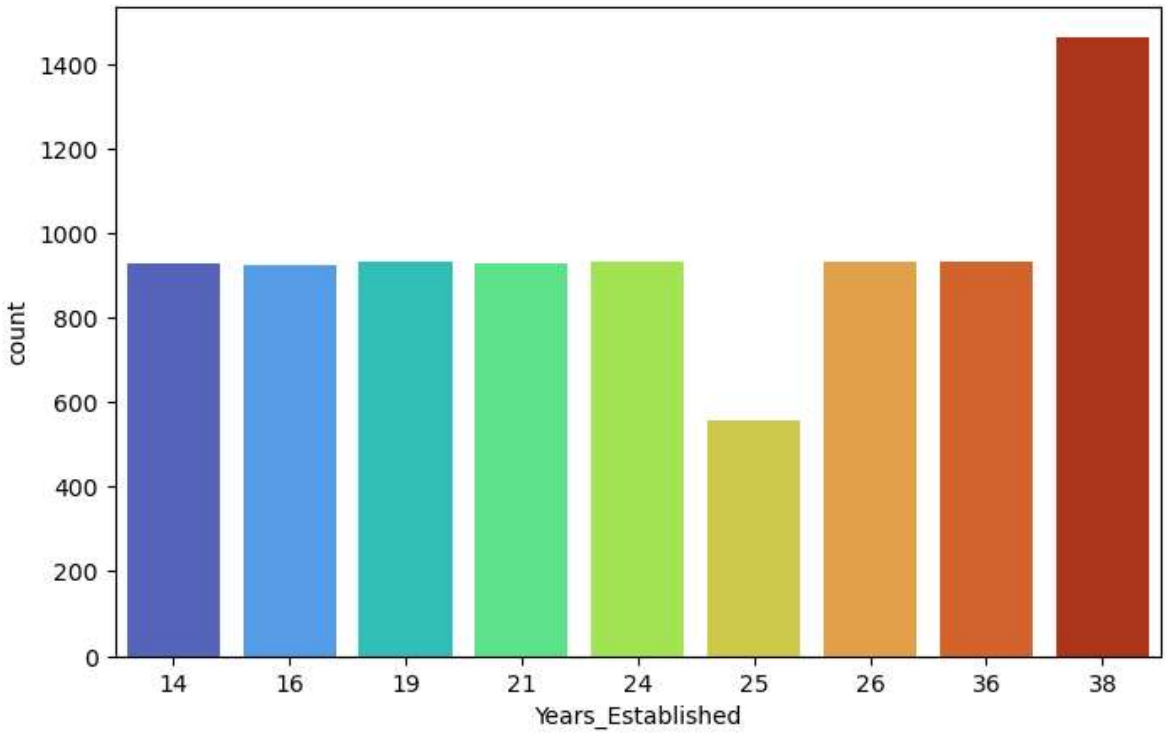
1. We have good amount of products for 50 MRP, 100 MRP ,180 MRP
2. But MRP ranging from 200-250 dollars is having high Sales.

4. Outlet_Establishment_Year

In [52]:
```
train['Years_Established'] = train['Outlet_Establishment_Year'].apply(lambda
test['Years_Established'] = test['Outlet_Establishment_Year'].apply(lambda x
```

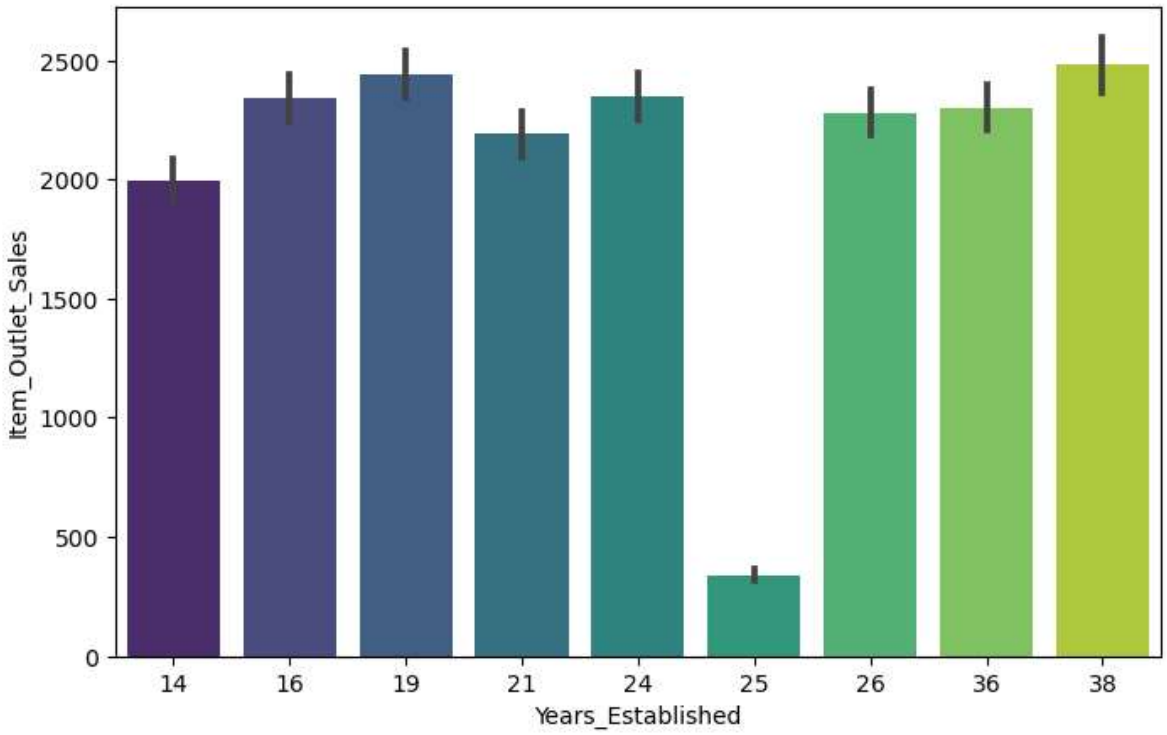
This code creates two new columns in the named train and test called 'Years_Established'. The values in the new columns are calculated by subtracting the 'Outlet_Establishment_Year' column from the year 2023, which gives the number of years that each outlet has been in operation.

In [53]:
```
plt.figure(figsize=(8,5))
sns.countplot(x='Years_Established',data=train,palette='turbo')
plt.show()
```



In [54]:
```
plt.figure(figsize=(8,5))
sns.barplot(x='Years_Established',y='Item_Outlet_Sales',data=train,palette='
```

Out[54]: <AxesSubplot:xlabel='Years_Established', ylabel='Item_Outlet_Sales'>



Observations:

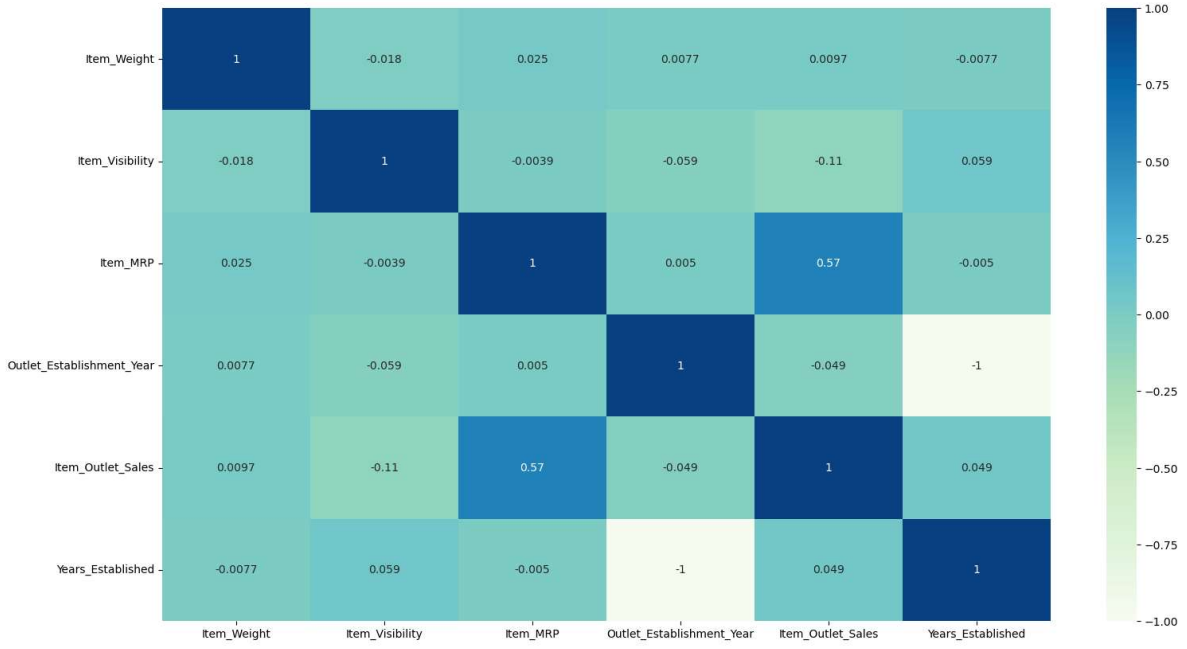
1. It is quiet evident that Outlets established 36 years before is having good Sales margin.
2. We also have a outlet which was established before 23 years has the lowest sales margin, so established years wouldn't improve the Sales unless the products are sold according to customer's interest.

In [55]:
```
train.head()
```

Out[55]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Iden
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	0.053931	Fruits and Vegetables	182.0950	OU
4	NCD19	8.93	Low Fat	0.053931	Household	53.8614	OU

In [56]:
```
plt.figure(figsize=(18,10))
sns.heatmap(train.corr() ,cmap='GnBu', annot=True)
plt.show()
```



Pre-Processing the Dataset

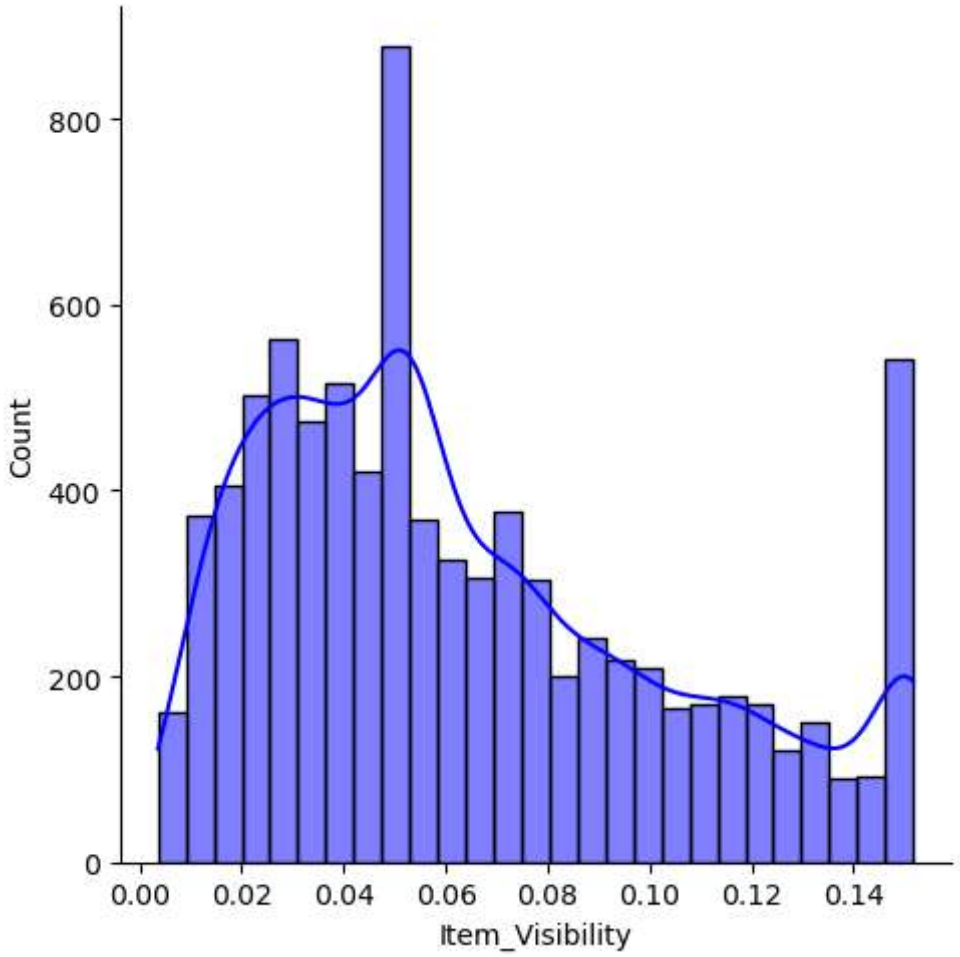
Remove the skweness

In [57]:
```
train.Item_Visibility.describe()
```

Out[57]:
```
count    8523.000000
mean        0.067683
std         0.043731
min         0.003575
25%         0.033085
50%         0.053931
75%         0.094585
max         0.163806
Name: Item_Visibility, dtype: float64
```

In [58]:
```
train.Item_Visibility = train.Item_Visibility.apply(np.log1p)
```

In [59]:
```
sns.displot(x=train.Item_Visibility, color='blue', kde=True)
plt.show()
```



In [60]:
```
train.Item_Visibility.describe()
```

Out[60]:
```
count    8523.000000
mean        0.064668
std         0.040395
min         0.003568
25%         0.032550
50%         0.052527
75%         0.090376
max         0.151695
Name: Item_Visibility, dtype: float64
```

Feature Engineering

In [61]:
```
from sklearn.preprocessing import LabelEncoder
```

In [62]:
```
from sklearn.preprocessing import LabelEncoder

# Create a LabelEncoder object
le = LabelEncoder()

# Encode categorical variables in the training dataset
for i in categorical.columns:
    train[i] = le.fit_transform(train[i])

# Encode categorical variables in the testing dataset
for i in categorical.columns:
    test[i] = le.fit_transform(test[i])
```

In [63]:
```
train.head()
```

Out[63]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Iden
0	156	9.30	0	0.015920	4	249.8092	
1	8	5.92	1	0.019095	14	48.2692	
2	662	17.50	0	0.016621	10	141.6180	
3	1121	19.20	1	0.052527	6	182.0950	
4	1297	8.93	0	0.052527	9	53.8614	

We can see Item_Outlet_Sales is highly correlated with Item_MRP, i.e. if Item_MRP increases, Item_Outlet_Sales increases.

Train Test Split

In [64]:
```
train.head(1)
```

Out[64]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Iden
0	156	9.3	0	0.01592	4	249.8092	

In [65]:
```
X = train.drop('Item_Outlet_Sales', axis=1)
y = train['Item_Outlet_Sales']

X.head(2)
```

Out[65]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Iden
0	156	9.30	0	0.015920	4	249.8092	
1	8	5.92	1	0.019095	14	48.2692	

In [66]:
```
y
```

Out[66]:
```
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 [67]:
```
from sklearn.model_selection import train_test_split

# split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, rand
```

In [68]:
```
X.shape, y.shape
```

Out[68]:
```
((8523, 12), (8523,))
```

In [69]:
```
X_train.shape, X_test.shape, y_test.shape, y_train.shape
```

Out[69]:
```
((6818, 12), (1705, 12), (1705,), (6818,))
```

In [70]:
```
import statsmodels.api as sm
```

In [71]:

```
import statsmodels.api as sm

# Create a linear regression model and fit it to the data
Lr_model = sm.OLS(y_train, X_train)
results = Lr_model.fit()

# Print the regression coefficients and other statistics
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:      Item_Outlet_Sales   R-squared:                       0.514
Model:                            OLS   Adj. R-squared:                  0.514
Method:                 Least Squares   F-statistic:                     655.7
Date:                Sun, 07 May 2023   Prob (F-statistic):               0.00
Time:                        09:20:34   Log-Likelihood:                -57940.
No. Observations:                6818   AIC:                         1.159e+05
Df Residuals:                    6806   BIC:                         1.160e+05
Df Model:                          11
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
Item_Identifier    0.0149      0.032      0.461      0.645      -0.048       0.078
Item_Weight       -0.9144      3.417     -0.268      0.789      -7.613       5.784
Item_Fat_Content  67.1235     30.606      2.193      0.028       7.126     127.121
Item_Visibility -1320.0088    367.137     -3.595      0.000   -2039.712    -600.306
Item_Type         -0.4048      3.465     -0.117      0.907      -7.197       6.388
Item_MRP          15.6413      0.231     67.691      0.000      15.188      16.094
Outlet_Identifier 61.1318     10.286      5.943      0.000      40.968      81.296
Outlet_Establishment_Year    -0.3006      0.061     -4.966      0.000      -0.419      -0.182
Outlet_Size     -361.9633     31.360    -11.542      0.000    -423.439    -300.488
Outlet_Location_Type -230.1280     42.660     -5.394      0.000    -313.755    -146.501
Outlet_Type      883.4404     29.193     30.263      0.000     826.214     940.667
Years_Established   -1.1775      1.763     -0.668      0.504      -4.634       2.279
==============================================================================
Omnibus:                      734.184   Durbin-Watson:                   2.016
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1620.511
Skew:                           0.663   Prob(JB):                         0.00
```

```
Kurtosis:                       4.987   Cond. No.                         5.50e+0
4
==============================================================================
=

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correc
tly specified.
[2] The condition number is large, 5.5e+04. This might indicate that there ar
e
strong multicollinearity or other numerical problems.
```

In [72]:

```python
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, r2_score

# create a Lasso model and fit it to the training data
lasso_model = Lasso(alpha=0.1) # set the regularization strength
lasso_model.fit(X_train, y_train)

# make predictions on the test data
y_pred = lasso_model.predict(X_test)

# calculate the mean squared error and R-squared for the test data
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# print the model coefficients and other statistics
print("Lasso Regression Report\n")
print("Model coefficients:")
print(lasso_model.coef_)
print("\nModel intercept:")
print(lasso_model.intercept_)
print("\nModel performance:")
print("Mean squared error: {:.2f}".format(mse))
print("R-squared: {:.2f}".format(r2))
```

```
Lasso Regression Report

Model coefficients:
[ 1.49664235e-02 -9.01999165e-01  6.64275607e+01 -1.25591343e+03
 -3.87219371e-01  1.56416531e+01  6.14312488e+01 -1.28819707e+01
 -3.61765612e+02 -2.28935095e+02  8.83123794e+02 -1.37735253e+01]

Model intercept:
25445.629508801667

Model performance:
Mean squared error: 1500873.65
R-squared: 0.49
```

In [73]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# create a Random Forest model and fit it to the training data
rf_model = RandomForestRegressor(n_estimators=100, random_state=100)
rf_model.fit(X_train, y_train)

# make predictions on the test data
y_pred = rf_model.predict(X_test)

# calculate the mean squared error and R-squared for the test data
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# print the model feature importances and other statistics
print("Random Forest Regression Report\n")
print("Feature importances:")
print(rf_model.feature_importances_)
print("\nModel performance:")
print("Mean squared error: {:.2f}".format(mse))
print("R-squared: {:.2f}".format(r2))
```

```
Random Forest Regression Report

Feature importances:
[0.06754306 0.04321793 0.0076875  0.08968094 0.03213827 0.43609104
 0.01591721 0.02272032 0.00776986 0.00580111 0.22871422 0.04271854]

Model performance:
Mean squared error: 1324909.41
R-squared: 0.55
```

In [74]:
```
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Create and fit the XGBoost model
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=100)
xgb_model.fit(X_train, y_train)

# Predict on the test set and calculate metrics
y_pred = xgb_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the report
print("XGBoost Regression Report\n")
print(f"Mean squared error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
```

```
XGBoost Regression Report

Mean squared error: 1402978.58
R-squared: 0.52
```

In [75]:
```
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score

#Create and fit the XGBoost model with hyperparameters
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=100, lea
xgb_model.fit(X_train, y_train)

# Predict on the test set and calculate metrics
y_pred = xgb_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the report
print("XGBoost Regression Report\n")
print(f"Mean squared error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
```

```
XGBoost Regression Report

Mean squared error: 1214832.47
R-squared: 0.59
```

Add some hyperparameters to increase the model performance

In [76]:
```
from xgboost import XGBRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV

# Create the XGBoost model
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=100)

# Define the hyperparameters to search
params = {'n_estimators': [50, 100, 200],
          'learning_rate': [0.01, 0.05, 0.1],
          'max_depth': [3, 5, 7],
          'min_child_weight': [1, 3, 5]}

# Create the grid search object
grid_search = GridSearchCV(xgb_model, params, scoring='neg_mean_squared_erro

# Fit the grid search to the training data
grid_search.fit(X_train, y_train)

# Get the best model from the grid search
best_model = grid_search.best_estimator_

# Predict on the test set and calculate metrics
y_pred = best_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print the report
print("XGBoost Regression Report\n")
print(f"Mean squared error: {mse:.2f}")
print(f"R-squared: {r2:.2f}")
print(f"Best parameters: {grid_search.best_params_}")
```

```
XGBoost Regression Report

Mean squared error: 1209476.77
R-squared: 0.59
Best parameters: {'learning_rate': 0.1, 'max_depth': 3, 'min_child_weight':
5, 'n_estimators': 50}
```

In [77]:
```
import pickle

# save the trained model in a file
with open('xgb_model.pkl', 'wb') as file:
    pickle.dump(xgb_model, file)
```

In [79]:
```
train.head(2)
```

Out[79]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Iden
0	156	9.30	0	0.015920	4	249.8092	
1	8	5.92	1	0.019095	14	48.2692	

In [81]:
```
# Define the preprocess_input function to preprocess the input data
def preprocess_input(input_data):
    # Your preprocessing code goes here
    preprocessed_data = input_data # Replace this with your actual preproces
    return preprocessed_data
```

In [82]:
```
# Define the preprocess_input function to preprocess the input data
def preprocess_input(input_data):
    # Your preprocessing code goes here
    preprocessed_data = input_data # Replace this with your actual preproces
    return preprocessed_data

# Collect user input
input_features = []
input_features.append(float(input("Enter value for Item_Identifier: ")))
input_features.append(float(input("Enter value for Item_Weight: ")))
input_features.append(float(input("Enter value for Item_Fat_Content: ")))
input_features.append(float(input("Enter value for Item_Visibility: ")))
input_features.append(float(input("Enter value for Outlet_Identifier: ")))
input_features.append(float(input("Enter value for Item_Type: ")))
input_features.append(float(input("Enter value for Item_MRP: ")))
input_features.append(float(input("Enter value for Outlet_Establishment_Year
input_features.append(float(input("Enter value for Outlet_Size: ")))
input_features.append(float(input("Enter value for Outlet_Location_Type: "))
input_features.append(float(input("Enter value for Outlet_Type: ")))
input_features.append(float(input("Enter value for Years_Established: ")))

# Preprocess the input
input_data = [input_features]
preprocessed_input = preprocess_input(input_data)

# Use the trained model to make predictions
predicted_value = xgb_model.predict(preprocessed_input)

# Print the predicted value
print(f"The predicted value is: {predicted_value[0]:.2f}")
```

```
Enter value for Item_Identifier: 2
Enter value for Item_Weight: 2
Enter value for Item_Fat_Content: 35
Enter value for Item_Visibility: 2
Enter value for Outlet_Identifier: 35
Enter value for Item_Type: 35
Enter value for Item_MRP: 5
Enter value for Outlet_Establishment_Year: 55
Enter value for Outlet_Size: 54
Enter value for Outlet_Location_Type: 5
Enter value for Outlet_Type: 6
Enter value for Years_Established: 9
```

```
---------------------------------------------------------------------------
NotFittedError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_15412\2849440533.py in <module>
     25
     26 # Use the trained model to make predictions
---> 27 predicted_value = xgb_model.predict(preprocessed_input)
     28
     29 # Print the predicted value

~\AppData\Roaming\Python\Python39\site-packages\xgboost\sklearn.py in predict
(self, X, output_margin, ntree_limit, validate_features, base_margin, iterati
on_range)
   1107         with config_context(verbosity=self.verbosity):
   1108             iteration_range = _convert_ntree_limit(
-> 1109                 self.get_booster(), ntree_limit, iteration_range
   1110             )
   1111             iteration_range = self._get_iteration_range(iteration_ran
ge)

~\AppData\Roaming\Python\Python39\site-packages\xgboost\sklearn.py in get_boo
ster(self)
    647             from sklearn.exceptions import NotFittedError
    648
--> 649             raise NotFittedError("need to call fit or load_model befo
rehand")
    650         return self._Booster
    651

NotFittedError: need to call fit or load_model beforehand
```

In [ ]:
```
X.shape, X.columns
```

In [ ]:
```

```