In [ ]:
```
# Load basic libraries

import seaborn as sns
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
```

In [ ]:
```
from pylab import rcParams

import warnings
warnings.filterwarnings("ignore")
```

FrameWork for EDA

Univariate Analysis- Single Variable Analysis

Numerical variables

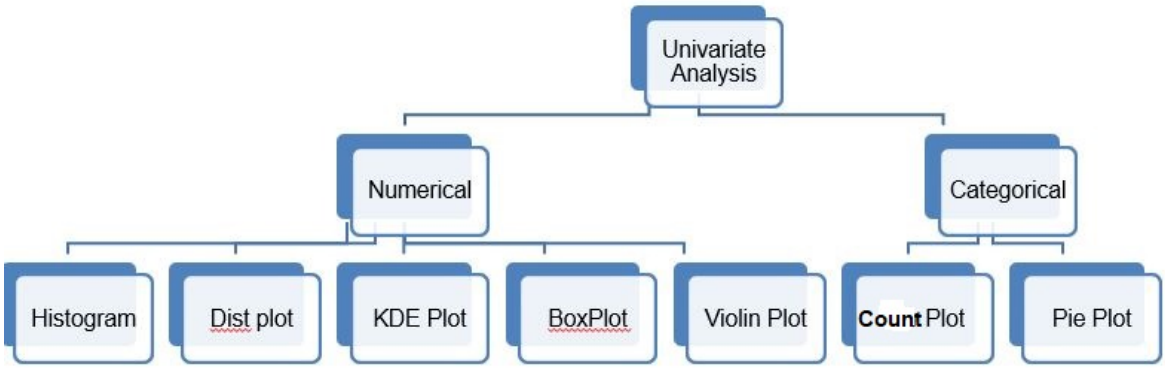
1. Distplots
2. Boxplots -outliers
3. Histograms
4. KDE Plots
5. Violin Plots

Categorical Variables

1. Countplot
2. Pie Plot

In [ ]:
```
from IPython.display import Image
Image(filename='Univariate.JPG',width=600,height=200)
```

Out[3]:



Bivariate Analysis

1. Bivariate can be done with one against another variable
2. Bivariate can be done with one variable with target variable.

2.1 Numerical vs Numerical - Scatter Plot, Pair plot

2.2 Category vs Numerical - Boxplot

2.3 Category vs Category - Crosstab (it is another form of frequency table)

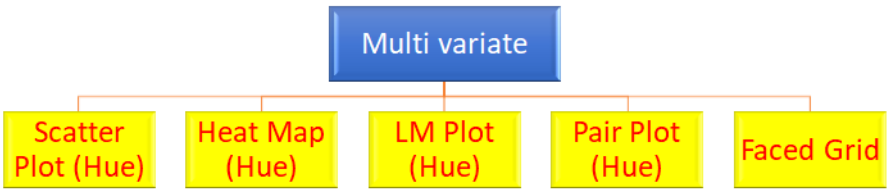
In [ ]:
```
from IPython.display import Image
Image(filename='Bivariate_1.png',width=300,height=300)
```

Out[4]:

Multivariate Analysis

In [ ]:
```
from IPython.display import Image
Image(filename='Multivariate.png',width=500,height=300)
```

Out[5]:



Missing Values

1. What are Missing values. How do you define missing values
2. How do you treat Missing values.

Outliers

1. What is outliers ?

2. How do you treat the missing values ?

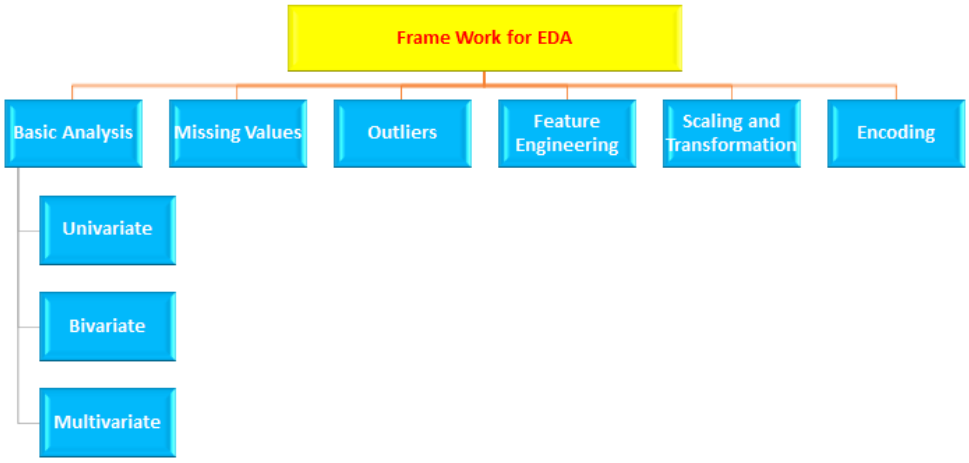
Feature Engineering

Scaling and Transformation

Encoding

In [ ]:
```
from IPython.display import Image
Image(filename='Frame Work-EDA.png',width=500,height=300)
```

Out[6]:



Lets visualize using the plots

In [ ]:
```
rcParams['figure.figsize'] = 7,6
```

Import the data file

In [ ]:
```
# df_sales = pd.read_csv("~/downloads/k_circle_sales.csv")
pd.read_csv("k_circle_sales.csv")
```

Out[8]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outle
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8	
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.3	
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6	
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.1	
4	NCD19	8.930	Low Fat	0.000000	Household	53.9	
...	...	...	...	...	...	...	
8518	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5	
8519	FDS36	8.380	Regular	0.046982	Baking Goods	108.2	
8520	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1	
8521	FDN46	7.210	Regular	0.145221	Snack Foods	103.1	
8522	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.5	

8523 rows × 12 columns

In [ ]:
```
df_sales = pd.read_csv("k_circle_sales.csv")
df_sales.head()
```

Out[9]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Id
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.3	
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6	
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.1	
4	NCD19	8.93	Low Fat	0.000000	Household	53.9	

check the number of rows and columns

In [ ]:
```
df_sales.shape
```

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

In [ ]:
```
df_sales.shape[0]
```

Out[11]:
```
8523
```

In [ ]:
```
print("No of rows",df_sales.shape[0])
print("No of columns", df_sales.shape[1])
```

```
No of rows 8523
No of columns 12
```

In [ ]:
```
df_sales.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       6473 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 [ ]:
```
df_sales.head(10)
```

Out[14]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Id
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8	
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.3	
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6	
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.1	
4	NCD19	8.930	Low Fat	0.000000	Household	53.9	
5	FDP36	10.395	Regular	0.000000	Baking Goods	51.4	
6	FDO10	13.650	Regular	0.012741	Snack Foods	57.7	
7	FDP10	NaN	Low Fat	0.127470	Snack Foods	107.8	
8	FDH17	16.200	Regular	0.016687	Frozen Foods	97.0	
9	FDU28	19.200	Regular	0.094450	Frozen Foods	187.8	

In [ ]:
```
df_sales.tail(10)
```

Out[15]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outle
8513	FDH31	12.000	Regular	0.020407	Meat	99.9	
8514	FDA01	15.000	Regular	0.054489	Canned	57.6	
8515	FDH24	20.700	Low Fat	0.021518	Baking Goods	157.5	
8516	NCJ19	18.600	Low Fat	0.118661	Others	58.8	
8517	FDF53	20.750	reg	0.083607	Frozen Foods	178.8	
8518	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5	
8519	FDS36	8.380	Regular	0.046982	Baking Goods	108.2	
8520	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1	
8521	FDN46	7.210	Regular	0.145221	Snack Foods	103.1	
8522	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.5	

Descriptive Statistics

In [ ]:
```
df_sales['Item_Weight'].mean()
```

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

In [ ]:
```
df_sales['Item_Weight'].median()
```

Out[17]:
```
12.6
```

In [ ]:
```
df_sales['Item_Weight'].mode()
```

Out[18]:
```
0    12.15
Name: Item_Weight, dtype: float64
```

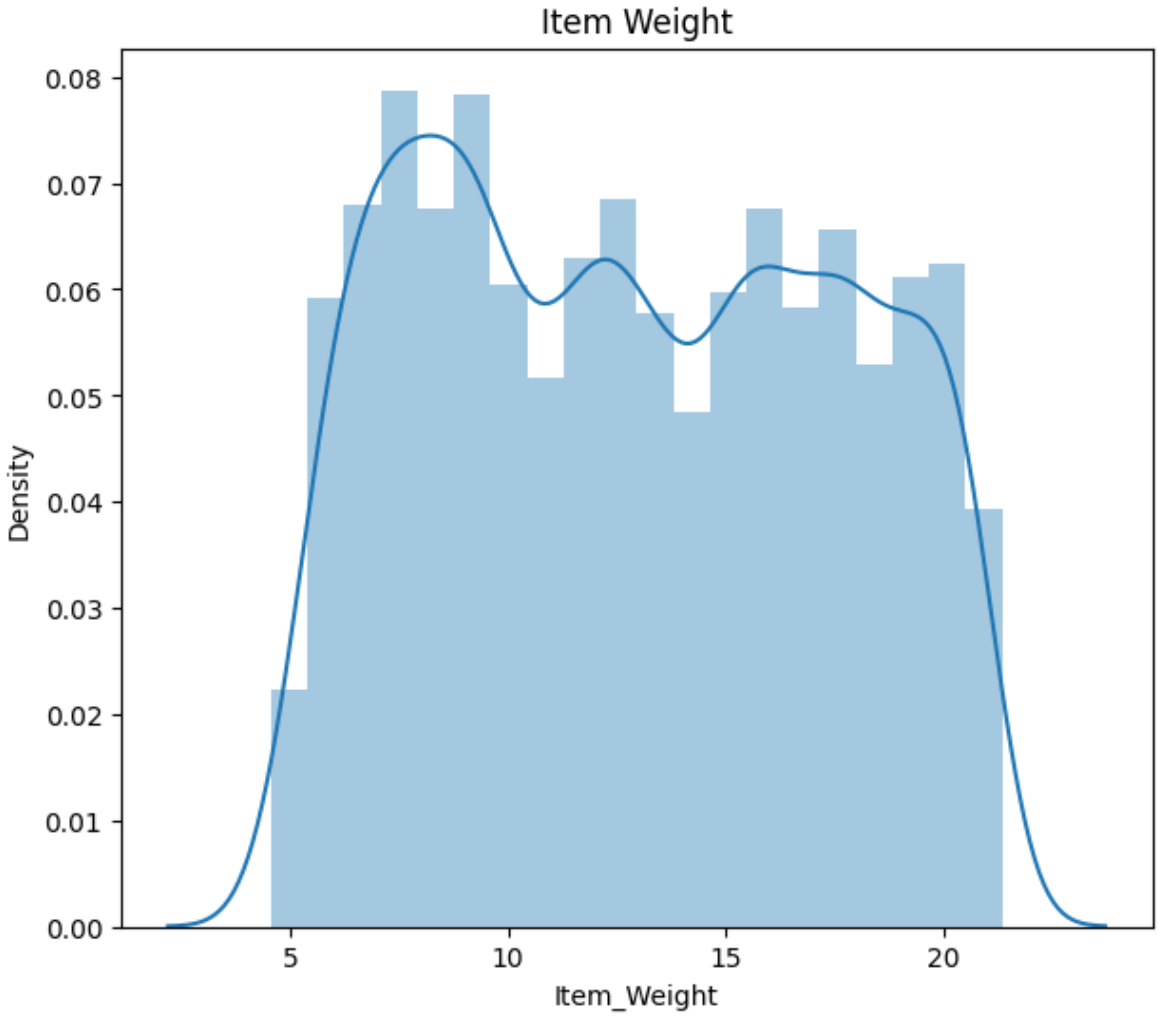
In [ ]:
```
df_sales['Item_Weight'].value_counts()
```

Out[19]:
```
12.150    86
17.600    82
13.650    77
11.800    76
15.100    68
          ..
7.275      2
7.685      1
9.420      1
6.520      1
5.400      1
Name: Item_Weight, Length: 415, dtype: int64
```

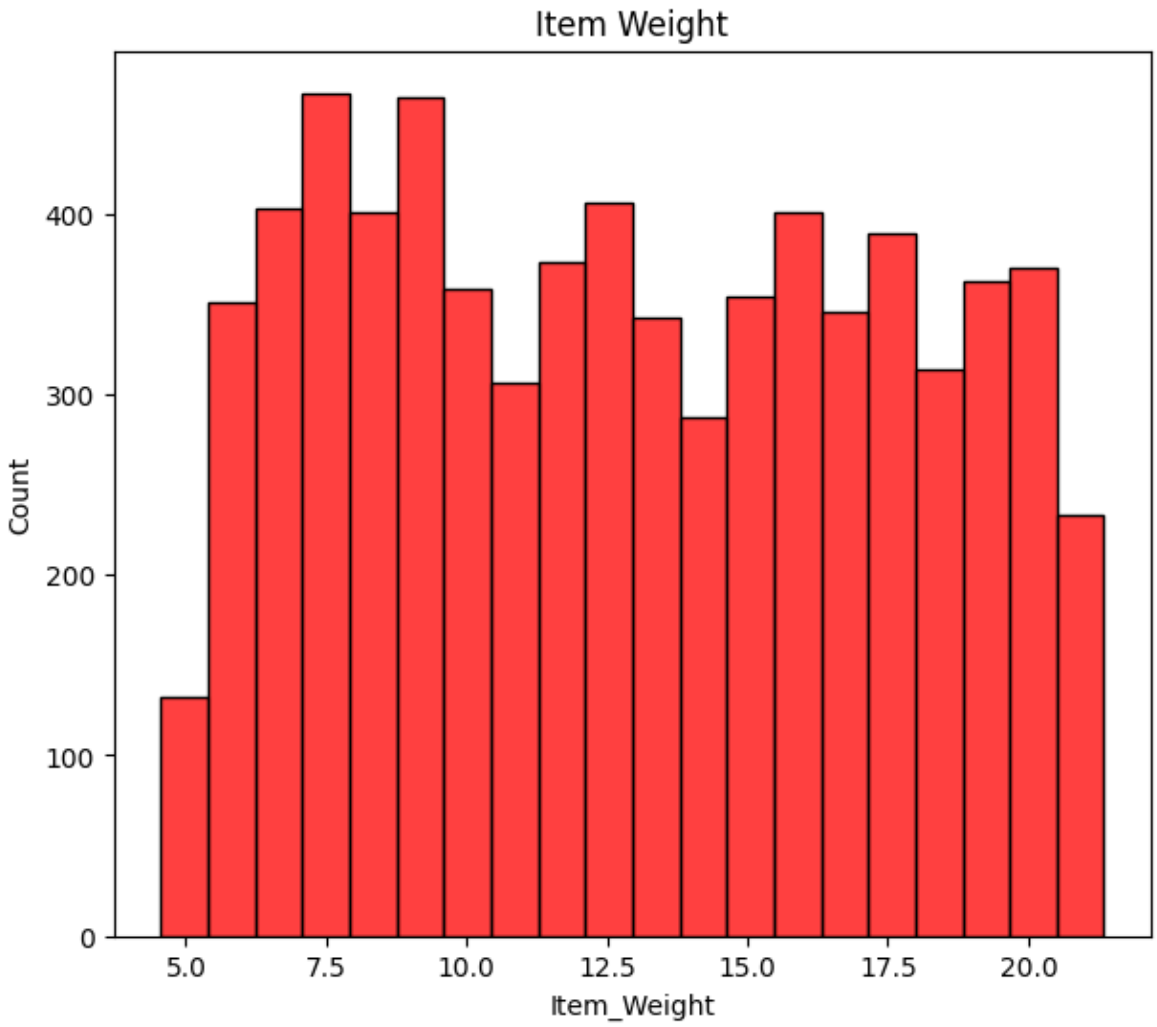
Univariate Analysis

From Seaborn Library

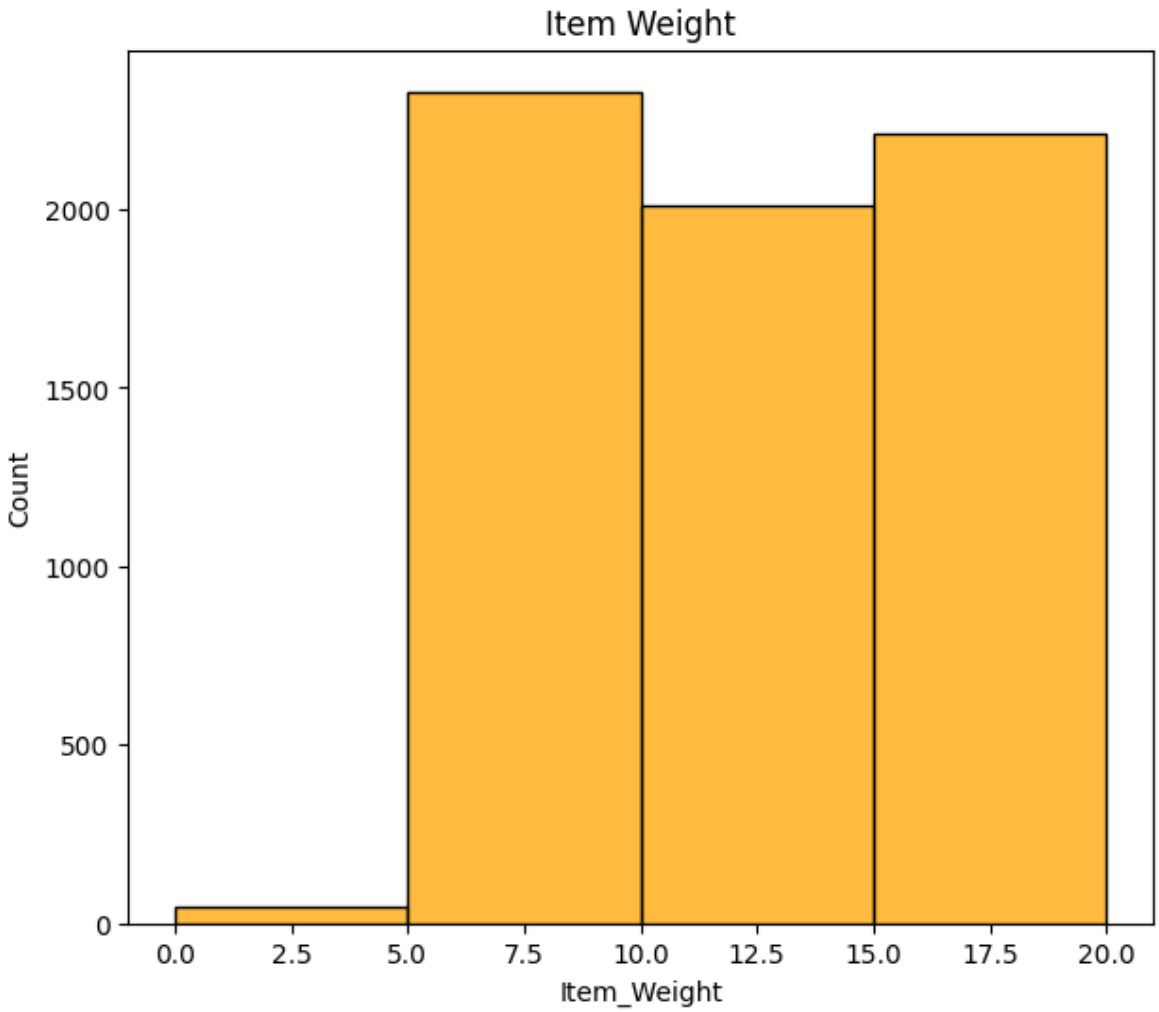
In [ ]:
```
sns.distplot(df_sales['Item_Weight'])
plt.title("Item Weight")
plt.show()
```



In [ ]:
```
sns.histplot(df_sales['Item_Weight'],color='red')
plt.title("Item Weight")
plt.show()
```

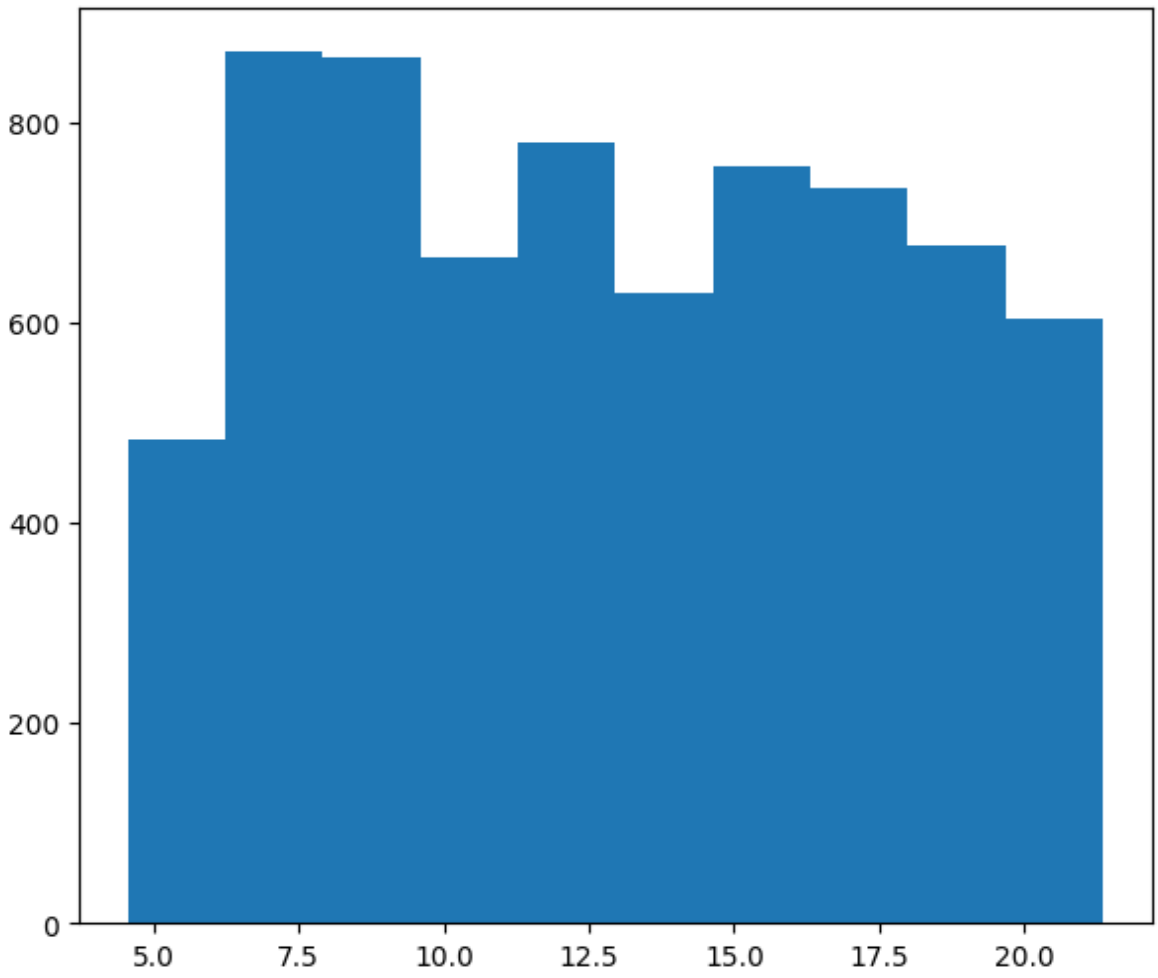


In [ ]:
```
sns.histplot(df_sales['Item_Weight'],bins=[0,5,10,15,20],color='orange')
plt.title("Item Weight")
plt.show()
```

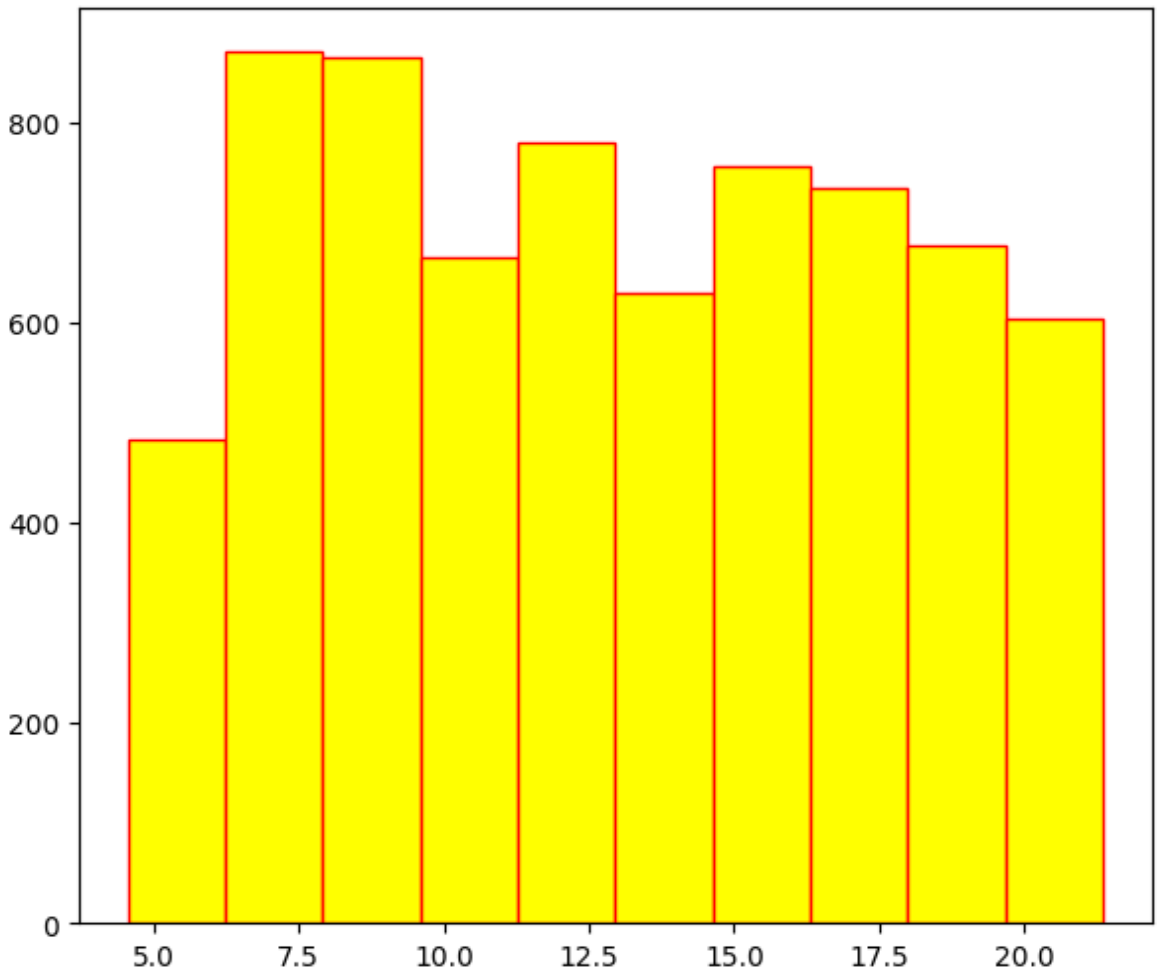


From Matplotlib

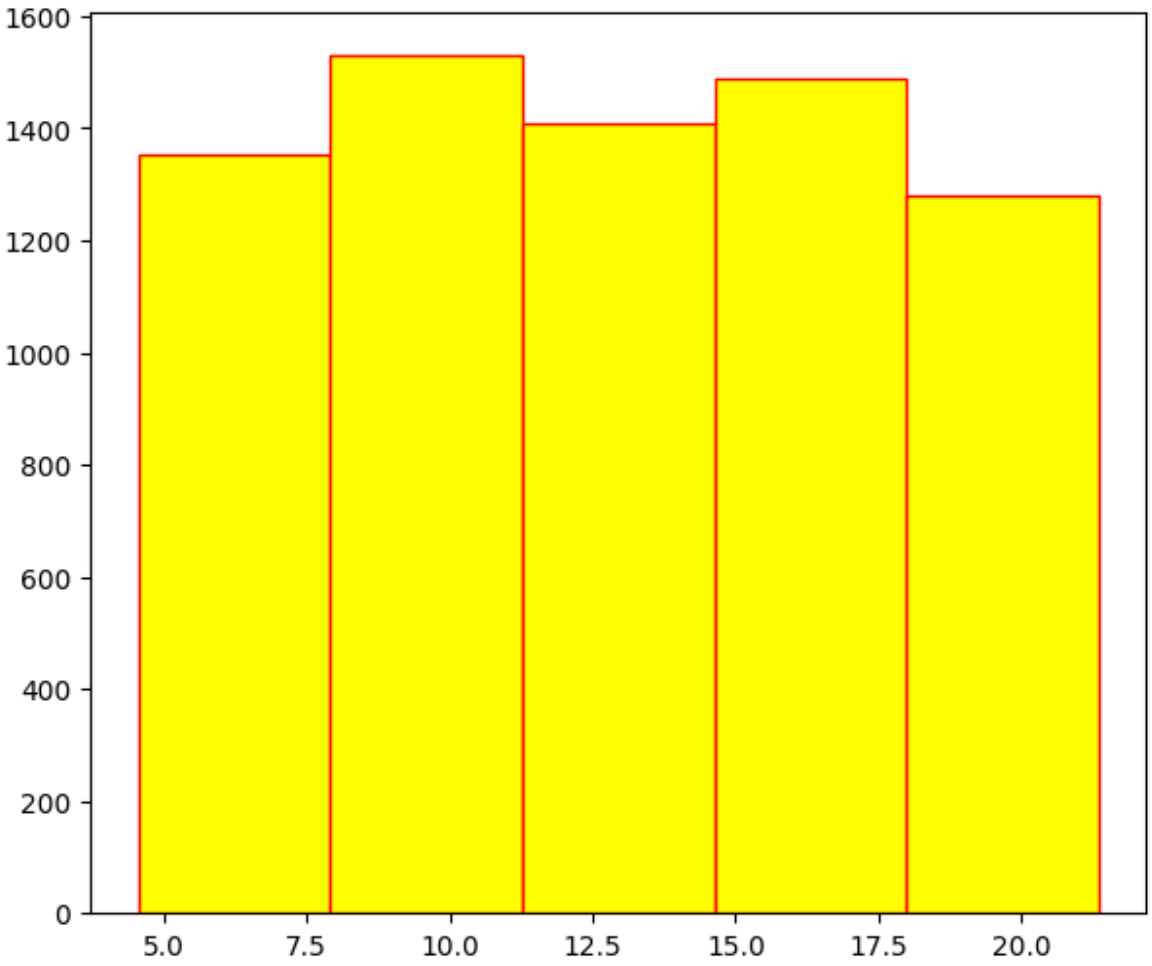
In [ ]:
```
plt.hist(x='Item_Weight', data=df_sales)
plt.show()
```



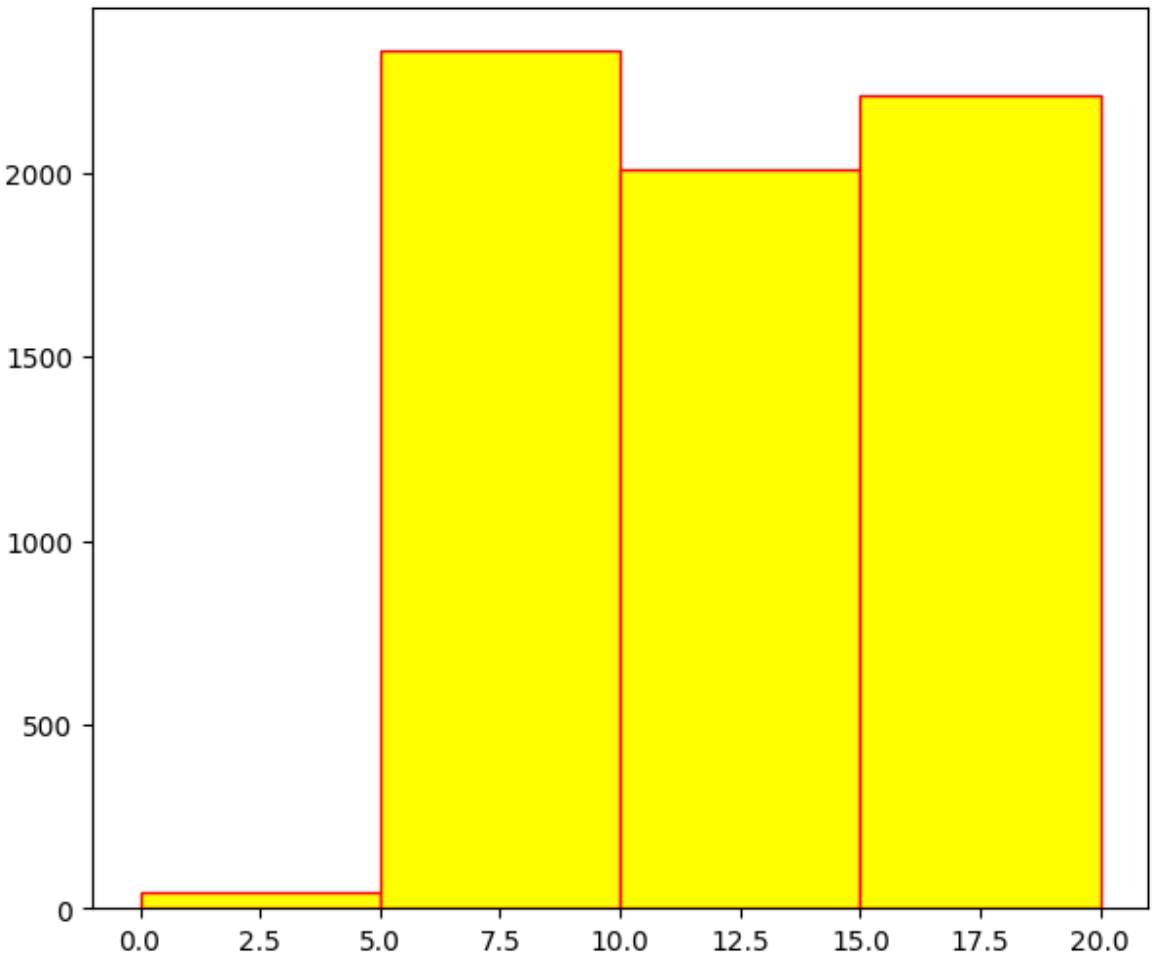
In [ ]:
```
plt.hist(x='Item_Weight', data=df_sales,color='yellow', edgecolor='red')
plt.show()
```



In [ ]:
```
plt.hist(x='Item_Weight', data=df_sales,bins=5, color='yellow', edgecolor='red
plt.show()
```



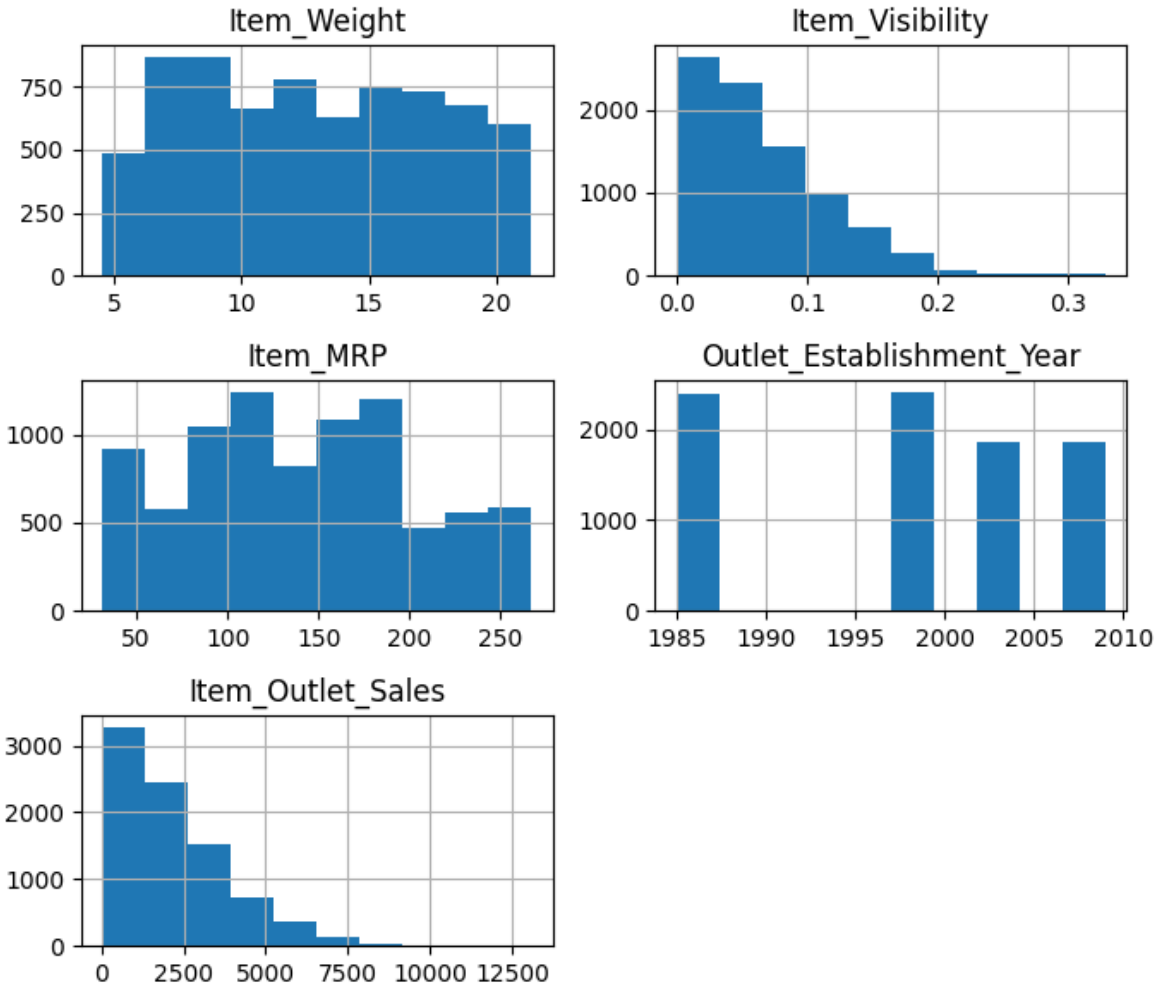
In [ ]:
```
plt.hist(x='Item_Weight', data=df_sales,bins=[0,5,10,15,20], color='yellow', e
plt.show()
```



Histogram on entire dataset

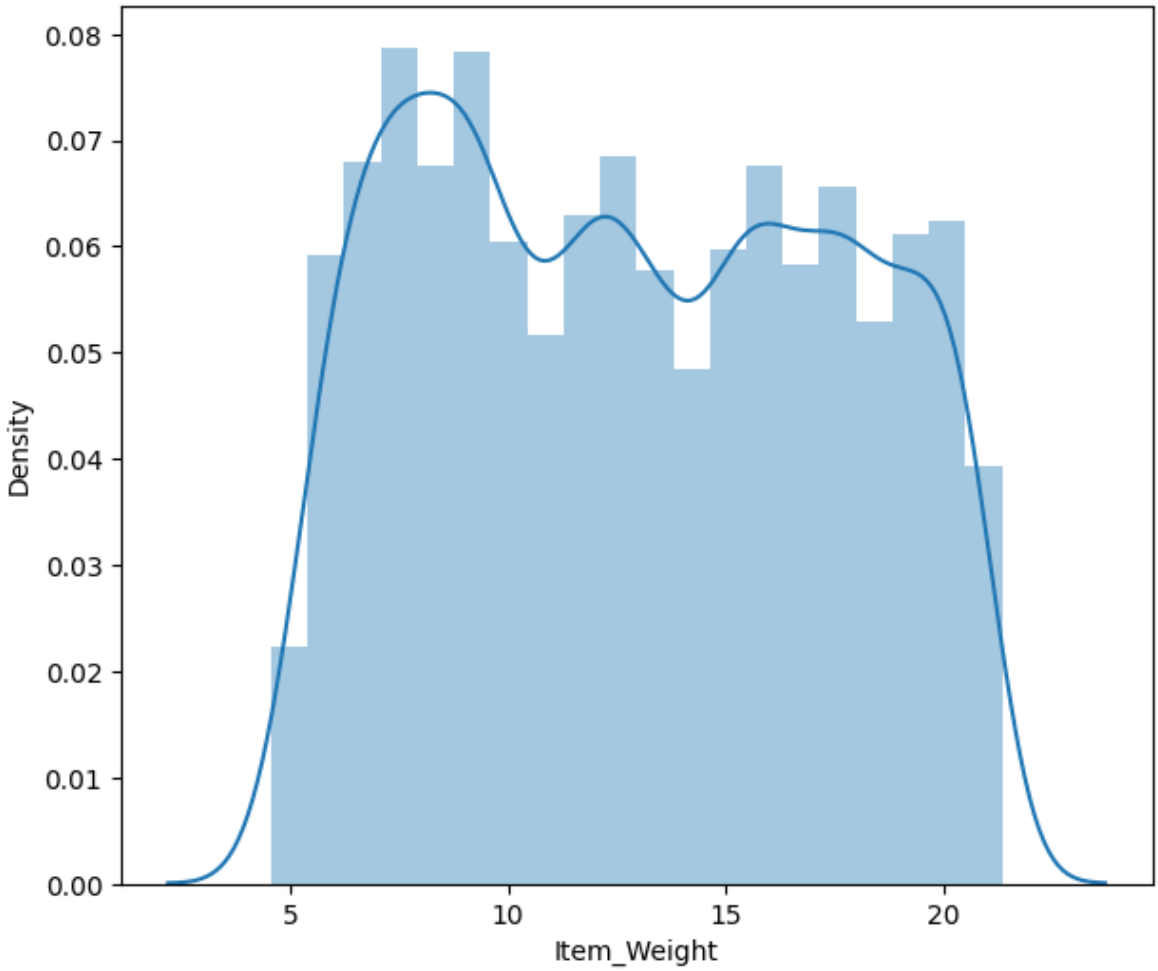
In [ ]:
```
plt.figure(figsize=(10,6))
df_sales.hist()

plt.tight_layout()
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```

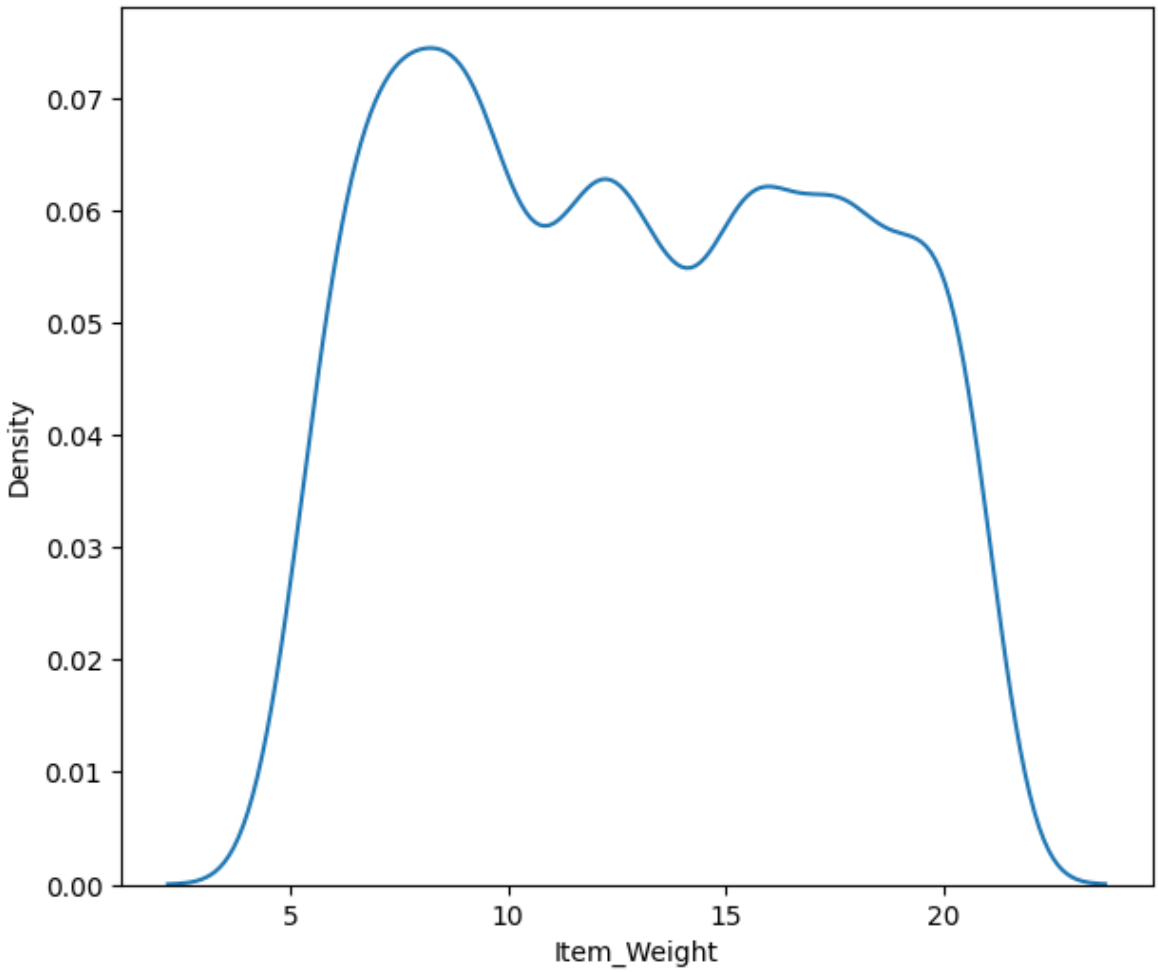


In [ ]:

In [ ]:
```
sns.distplot(df_sales.Item_Weight)
plt.show()
```

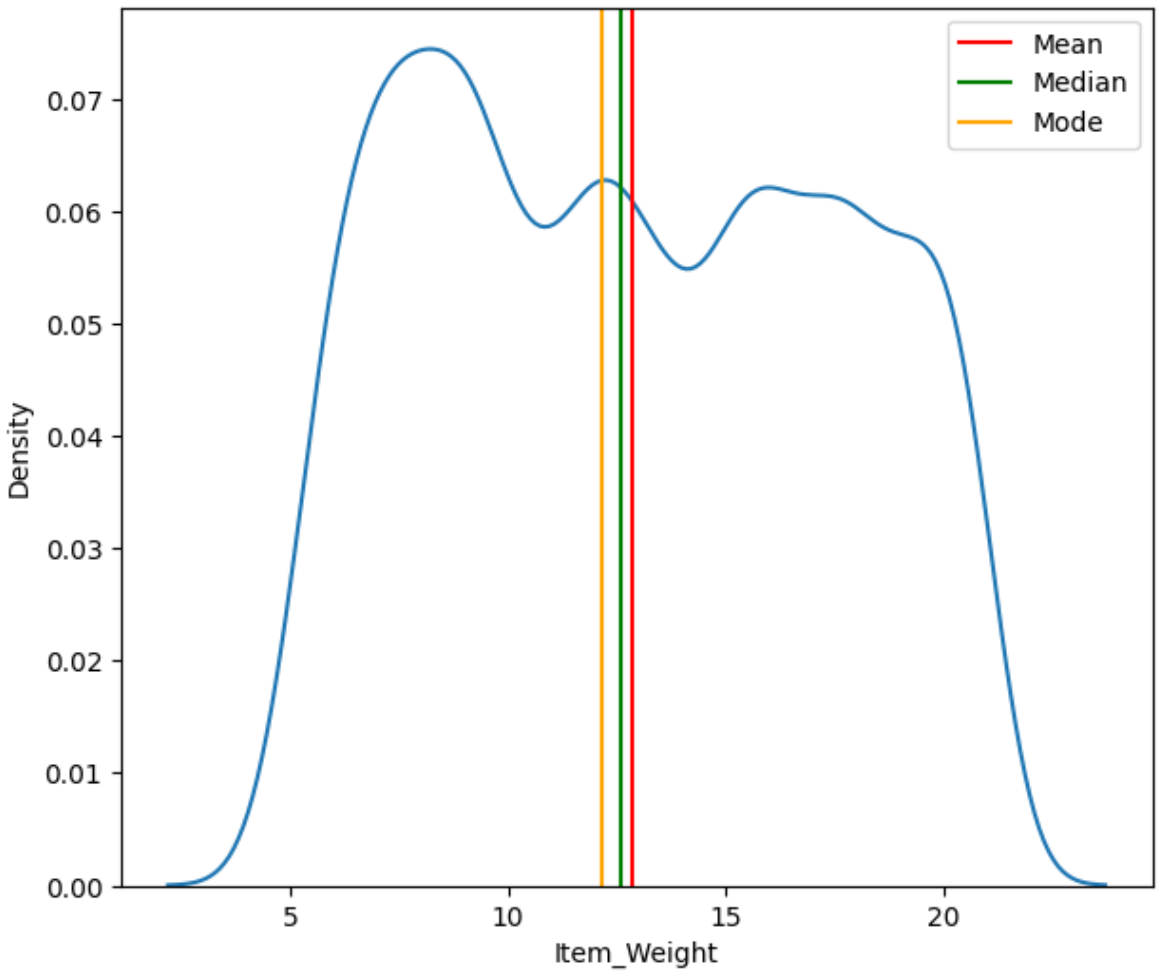


In [ ]:
```
# display the KDE plot
sns.kdeplot(df_sales['Item_Weight'])
plt.show()
```

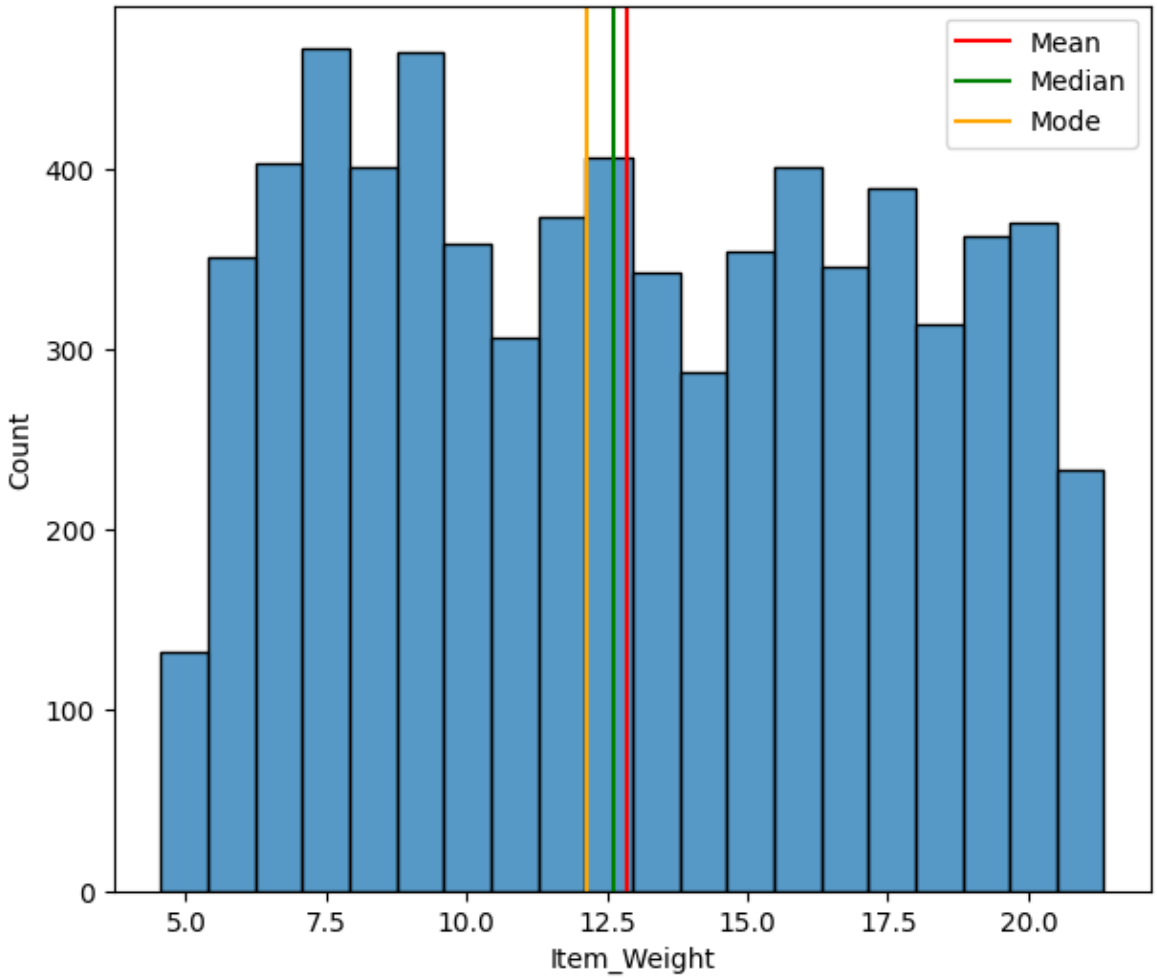


In [ ]:
```
me = df_sales['Item_Weight'].mean()
md = df_sales['Item_Weight'].median()
mo = df_sales['Item_Weight'].mode()[0]
```

In [ ]:
```
#diplay mean. median and mode
sns.kdeplot(df_sales['Item_Weight']);
plt.axvline(me,label='Mean',color='Red');
plt.axvline(md,label='Median',color='Green');
plt.axvline(mo,label='Mode',color='Orange');

plt.legend()
plt.show()
```



In [ ]:
```
#diplay mean. median and mode
sns.histplot(df_sales['Item_Weight']);
plt.axvline(me,label='Mean',color='Red');
plt.axvline(md,label='Median',color='Green');
plt.axvline(mo,label='Mode',color='Orange');

plt.legend()
plt.show()
```

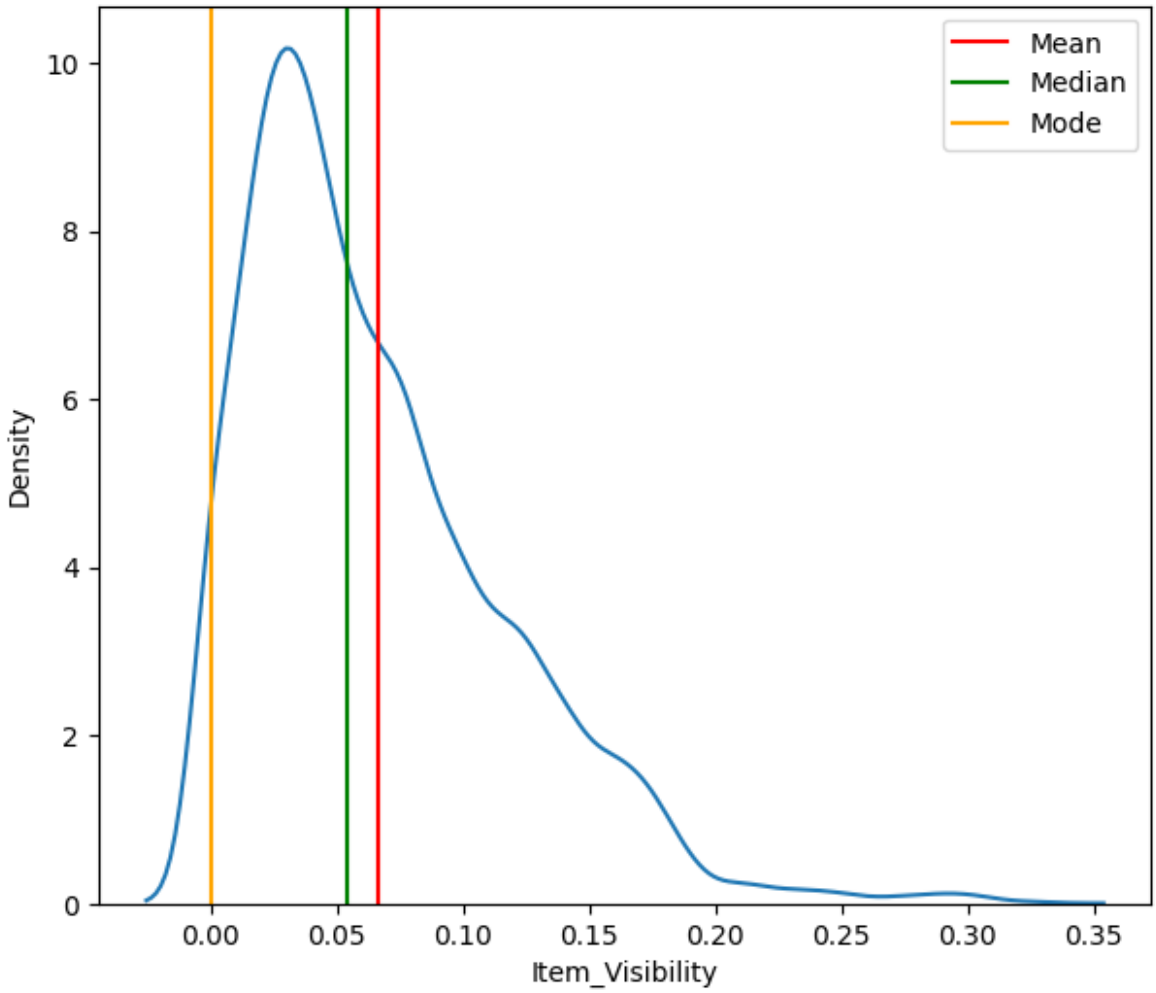


In [ ]:
```
me_vis = df_sales['Item_Visibility'].mean()
md_vis = df_sales['Item_Visibility'].median()
mo_vis = df_sales['Item_Visibility'].mode()[0]
```

In [ ]:
```
df_sales['Item_Visibility'].mode()
```

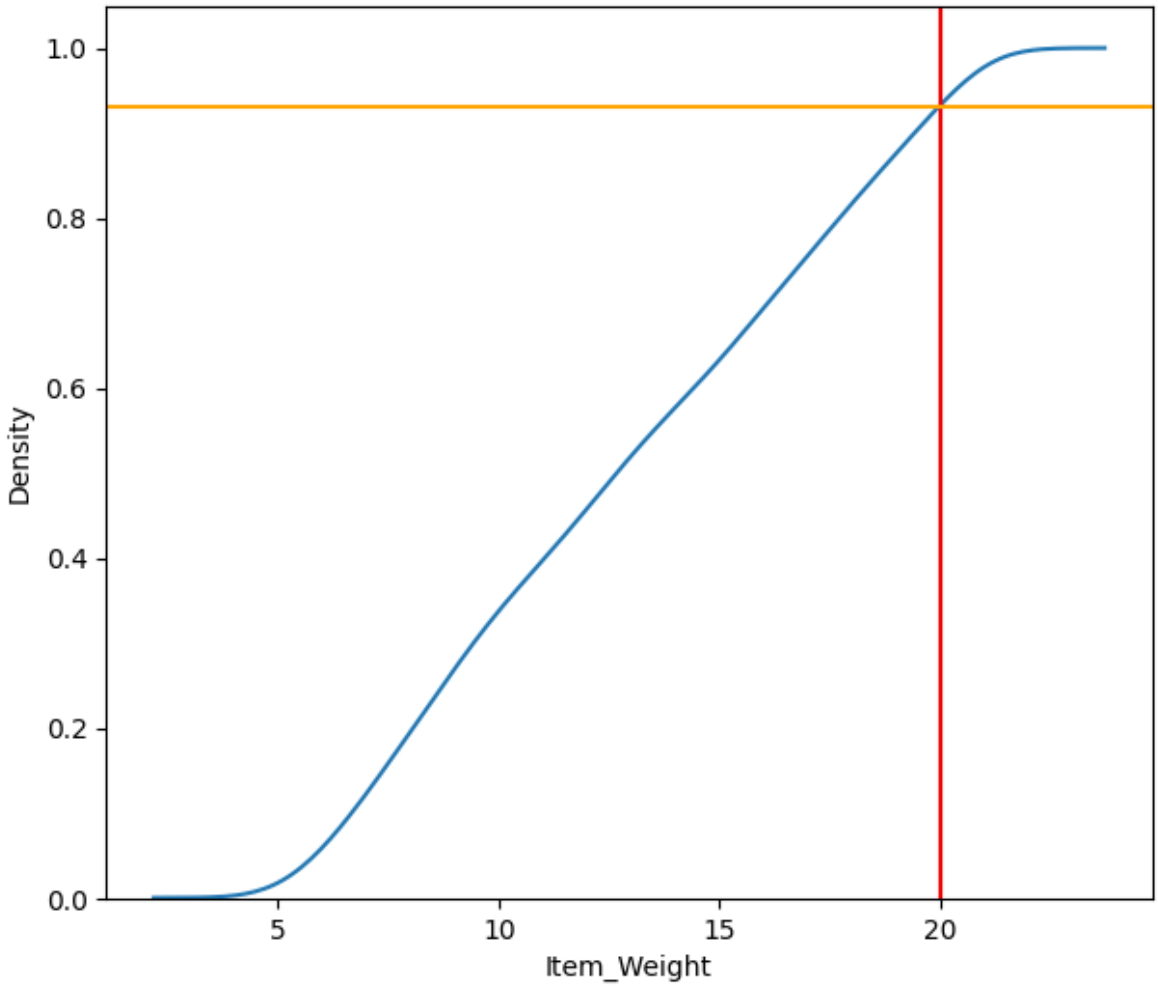
Out[34]:
```
0    0.0
Name: Item_Visibility, dtype: float64
```

In [ ]:
```
#diplay mean. median and mode
sns.kdeplot(df_sales['Item_Visibility']);
plt.axvline(me_vis,label='Mean',color='Red');
plt.axvline(md_vis,label='Median',color='Green');
plt.axvline(mo_vis,label='Mode',color='Orange');

plt.legend()
plt.show()
```



Analyzing cummalative KDE plots as part of Univariate Analysis

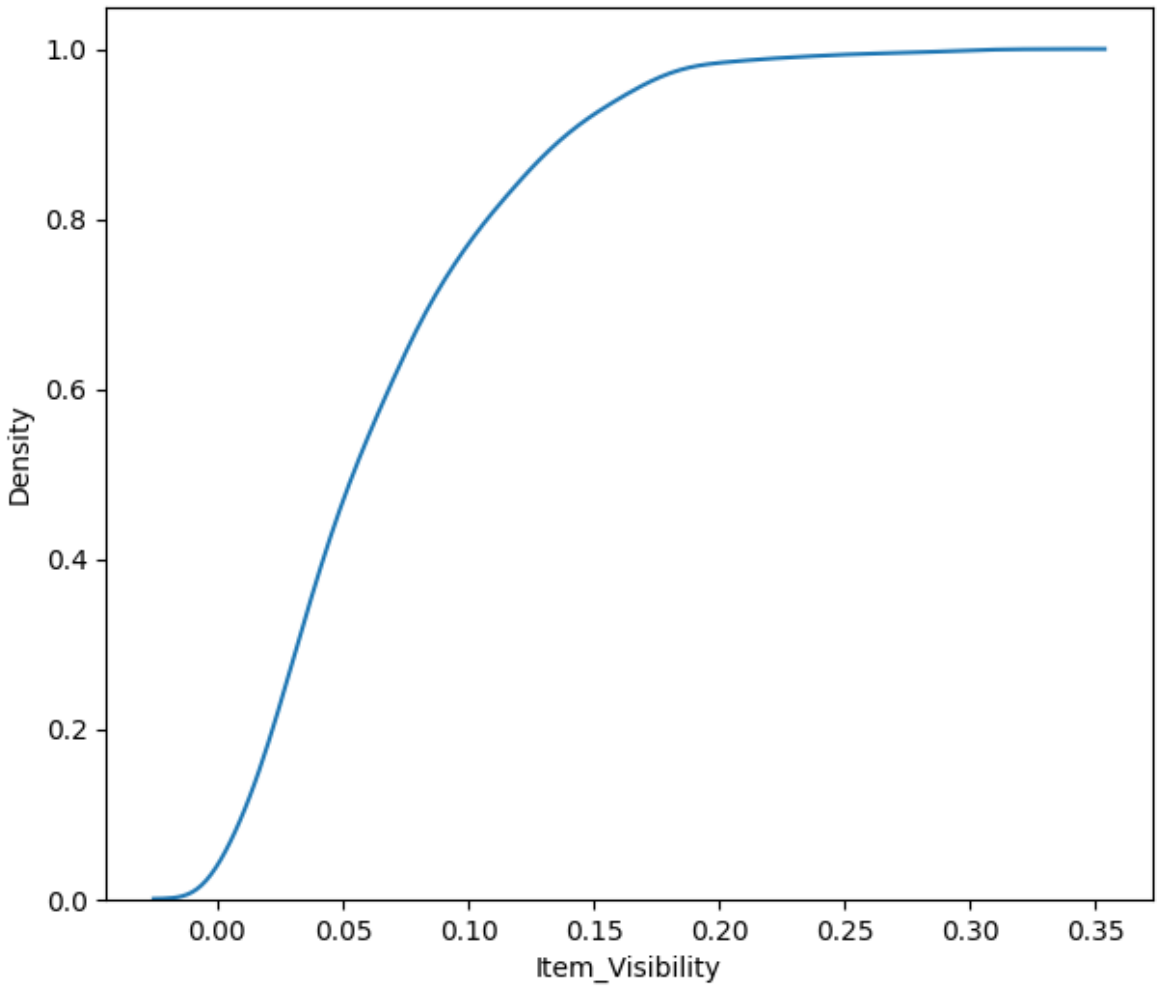
In [ ]:
```
# cummulative plot and its relevance
sns.kdeplot(x='Item_Weight',data=df_sales,cumulative=True) ;

plt.axvline(20,color='red')
plt.axhline(0.93,color='orange')

plt.show()
```



The cummulative plot is the density plot which gives the chances of that event happening.

The chances that Item weight is 20 units is 93%.

In [ ]:
```
# cummulative plot and its relevance
sns.kdeplot(x='Item_Visibility',data=df_sales,cumulative=True) ;
plt.show()
```



In [ ]:
```
```

Skewness and Kurtosis

In [ ]:
```
df_sales['Item_Weight'].skew()
```

Out[38]:
```
0.0824262091221237
```

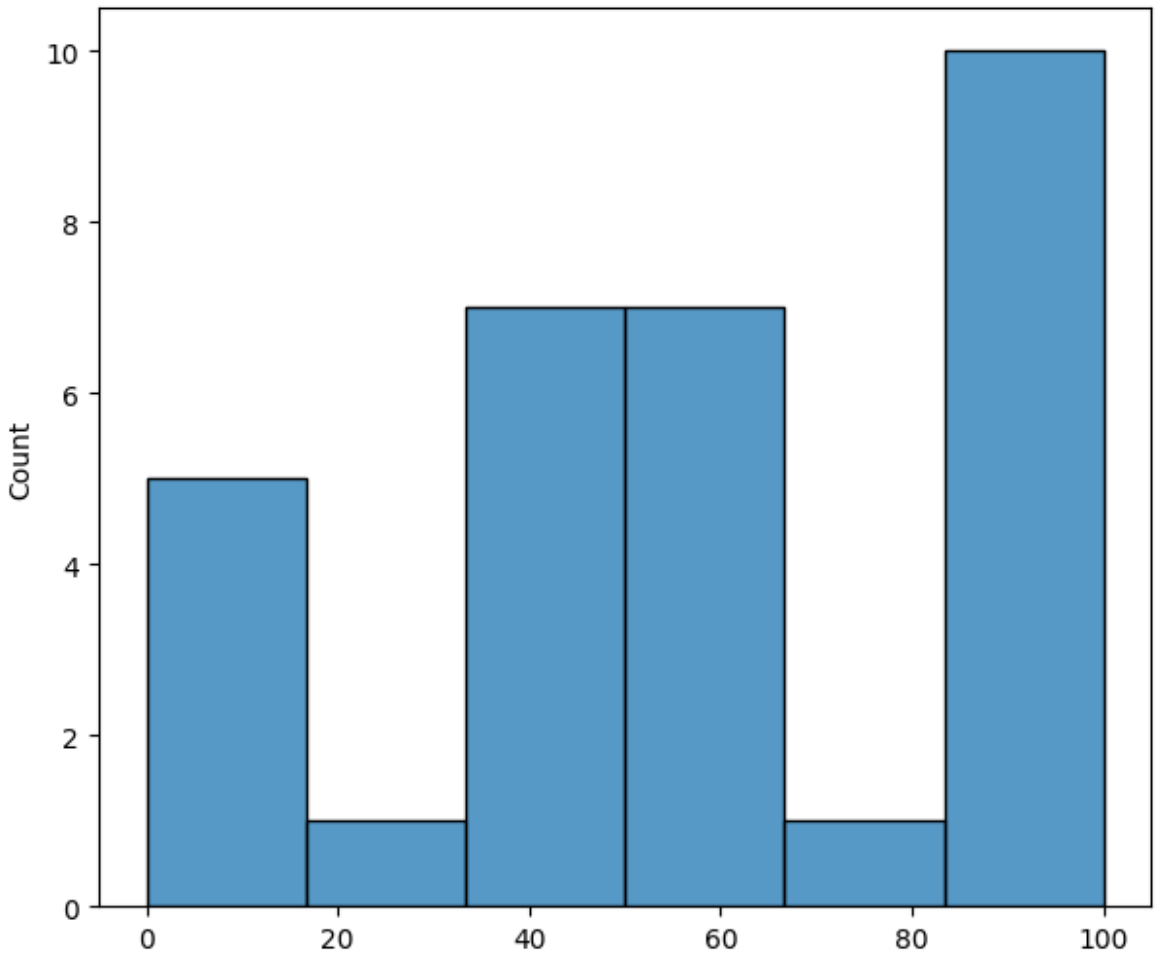
In [ ]:
```
df_sales['Item_Weight'].kurt()
```

Out[39]:
```
-1.2277664144376634
```

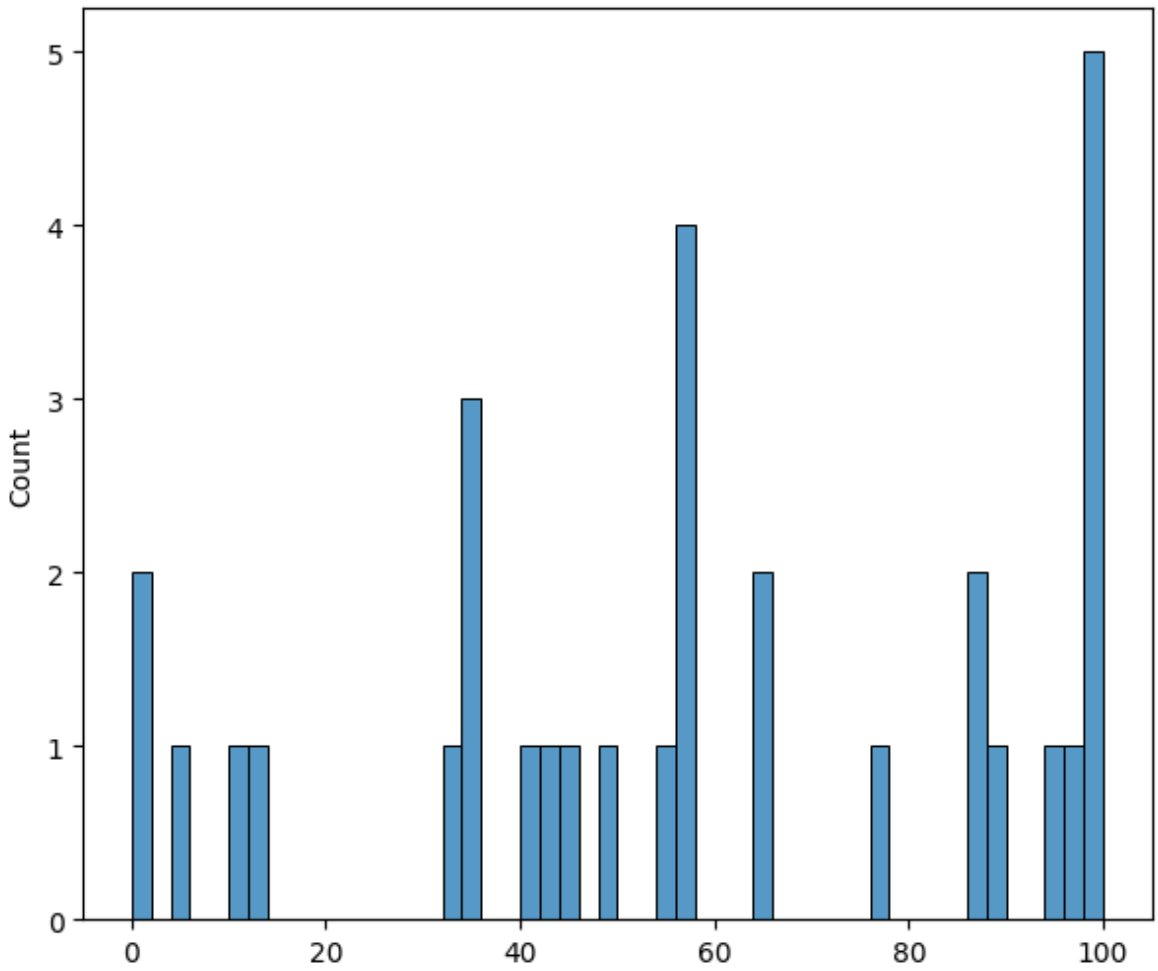
Both of them are positively skewed or towards the right.

In [ ]:
```
#lets create a series of data and do some basic operation
prices= pd.Series([0,0,35,40,10,54,87,12,95,64,56,4,45,76,34,56,87,34,56,32,56
```

In [ ]:
```
sns.histplot(prices)
plt.show()
```



In [ ]:
```
sns.histplot(prices,binwidth=2)
plt.show()
```

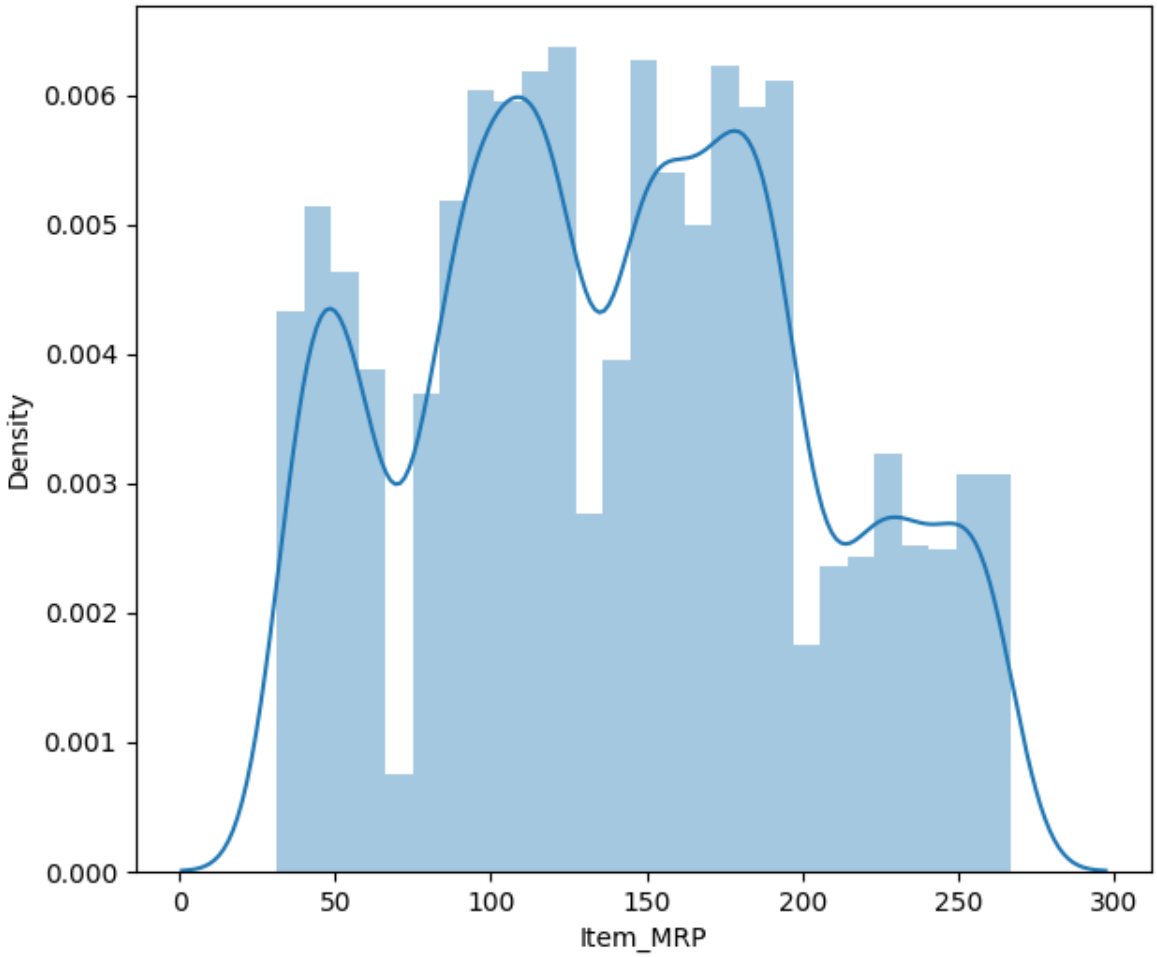


In [ ]:
```
df_sales.columns
```

Out[43]:
```
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')
```

Item MRP

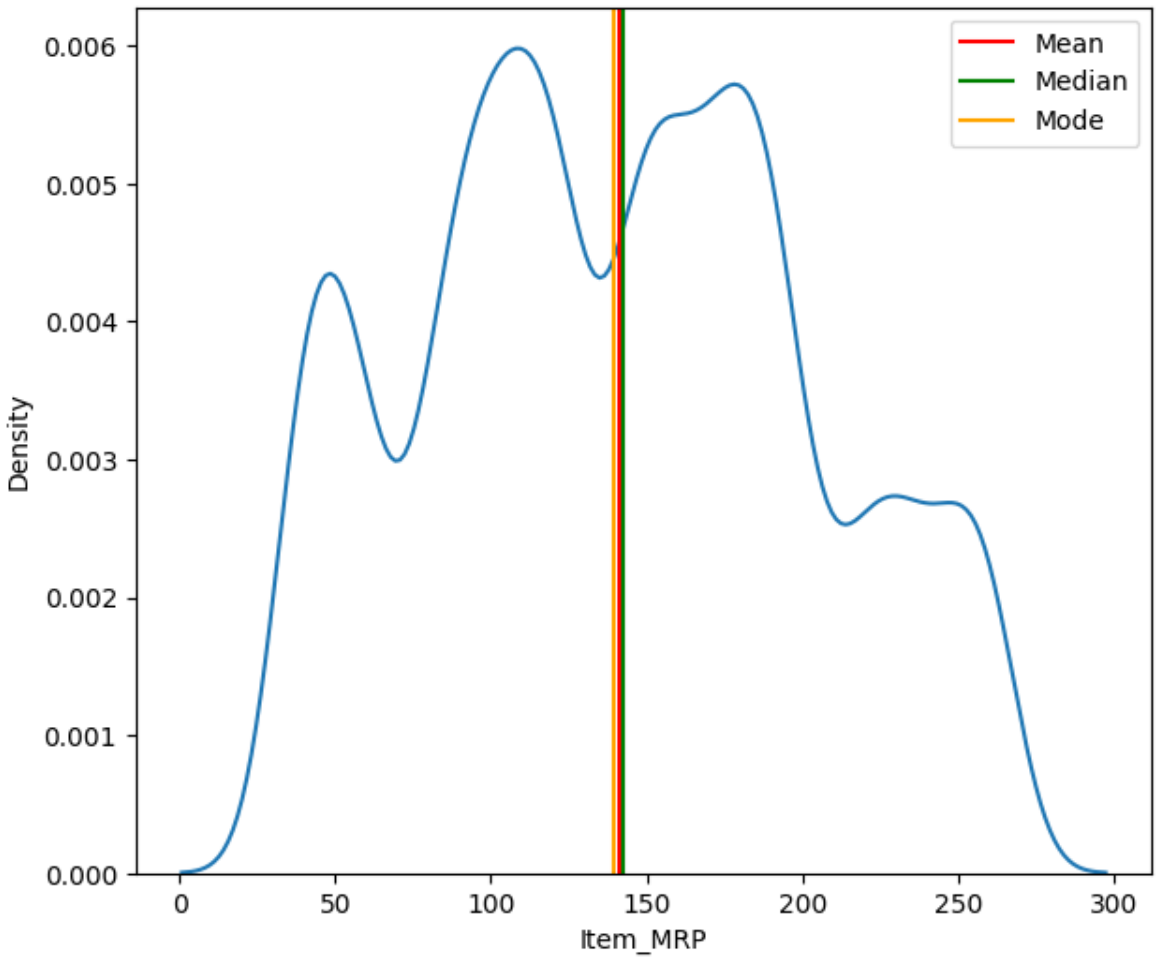
In [ ]:
```
sns.distplot(df_sales.Item_MRP)
plt.show()
```



1. It has multiple modes. Hence called as Mutimodal
2. There are different frquencies. It shows 4 different frequecy.
3. This is multimodel.

In [ ]:
```
me_mrp = df_sales['Item_MRP'].mean()
md_mrp = df_sales['Item_MRP'].median()
mo_mrp = df_sales['Item_MRP'].mode()[0]
```

In [ ]:
```
#diplay mean. median and mode
sns.kdeplot(df_sales['Item_MRP']);
plt.axvline(me_mrp,label='Mean',color='Red');
plt.axvline(md_mrp,label='Median',color='Green');
plt.axvline(mo_mrp,label='Mode',color='Orange');

plt.legend()
plt.show()
```



In [ ]:
```
df_sales['Item_MRP'].skew()
```

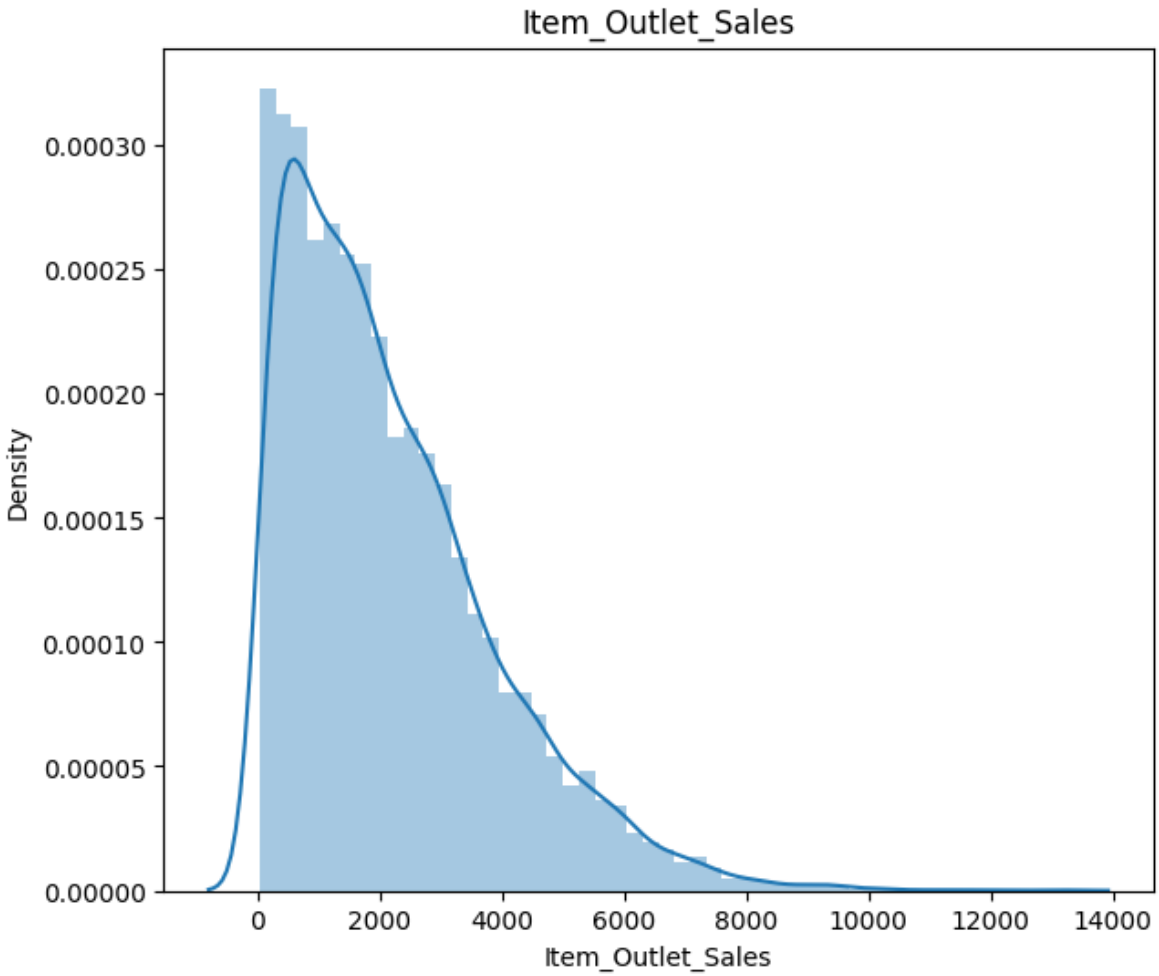
Out[47]:
```
0.12738985338612047
```

In [ ]:
```
df_sales['Item_MRP'].kurt()
```

Out[48]:
```
-0.8879008549272447
```

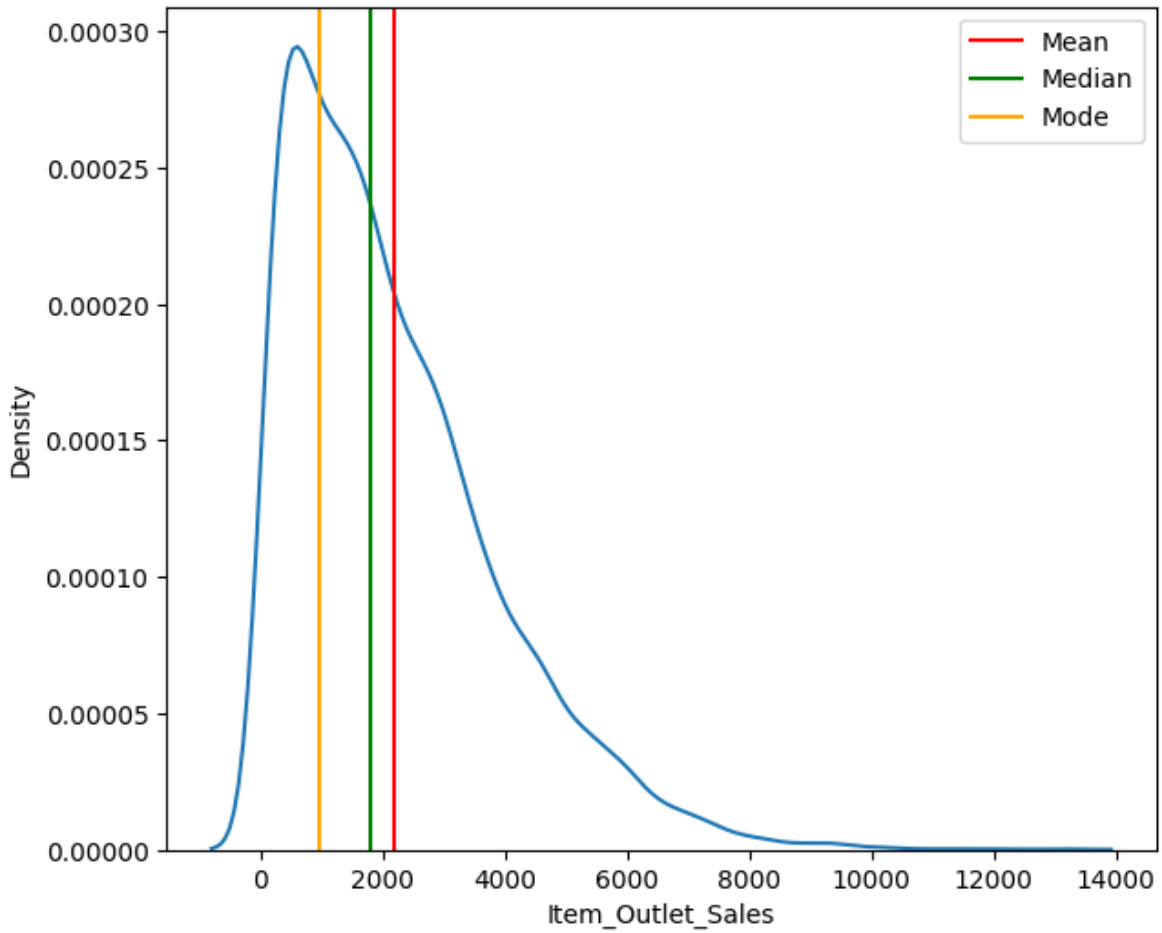
Item Outlet Sales

In [ ]:
```
sns.distplot(df_sales['Item_Outlet_Sales'])
plt.title("Item_Outlet_Sales")
plt.show()
```



In [ ]:
```
me_sales = df_sales['Item_Outlet_Sales'].mean()
md_sales = df_sales['Item_Outlet_Sales'].median()
mo_sales = df_sales['Item_Outlet_Sales'].mode()[0]
```

In [ ]:
```
#diplay mean. median and mode
sns.kdeplot(df_sales['Item_Outlet_Sales']);
plt.axvline(me_sales,label='Mean',color='Red');
plt.axvline(md_sales,label='Median',color='Green');
plt.axvline(mo_sales,label='Mode',color='Orange');

plt.legend()
plt.show()
```



Skewness and Kurtosis for all the data points

In [ ]:
```
# Finding the skewness of the sales
print("skewness for sales",df_sales['Item_Outlet_Sales'].skew())
```

```
skewness for sales 1.1775306028542796
```

In [ ]:
```
# Kurtosis of the sales
print("Kurtosis of sales", df_sales['Item_Outlet_Sales'].kurt())
```

```
Kurtosis of sales 1.6158766814287264
```

In [ ]:
```
df_sales.skew()
```

Out[54]:
```
Item_Weight                 0.082426
Item_Visibility             1.167091
Item_MRP                    0.127390
Outlet_Establishment_Year  -0.396641
Item_Outlet_Sales           1.177531
dtype: float64
```

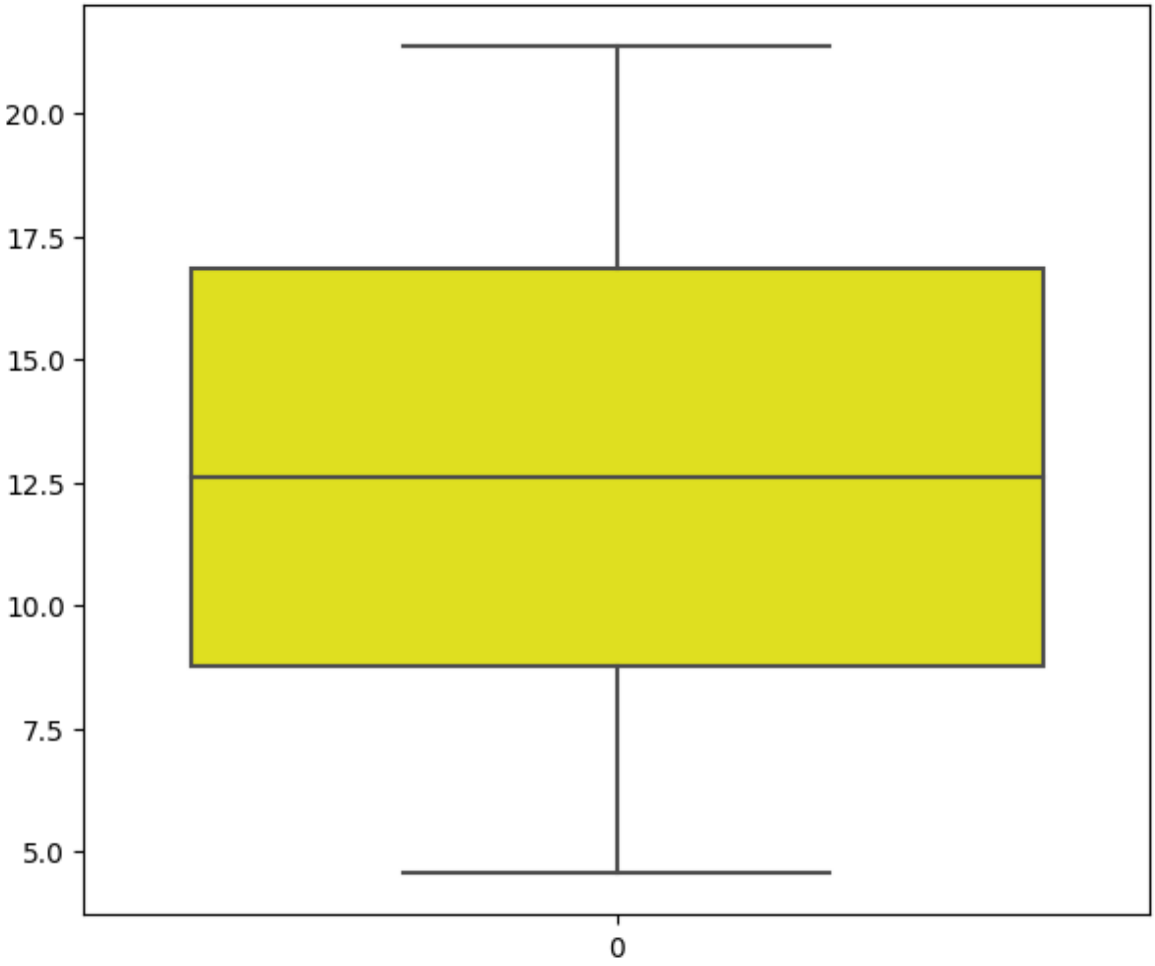
In [ ]:
```
df_sales.kurt()
```

Out[55]:
```
Item_Weight                -1.227766
Item_Visibility             1.679445
Item_MRP                   -0.887901
Outlet_Establishment_Year  -1.205694
Item_Outlet_Sales           1.615877
dtype: float64
```

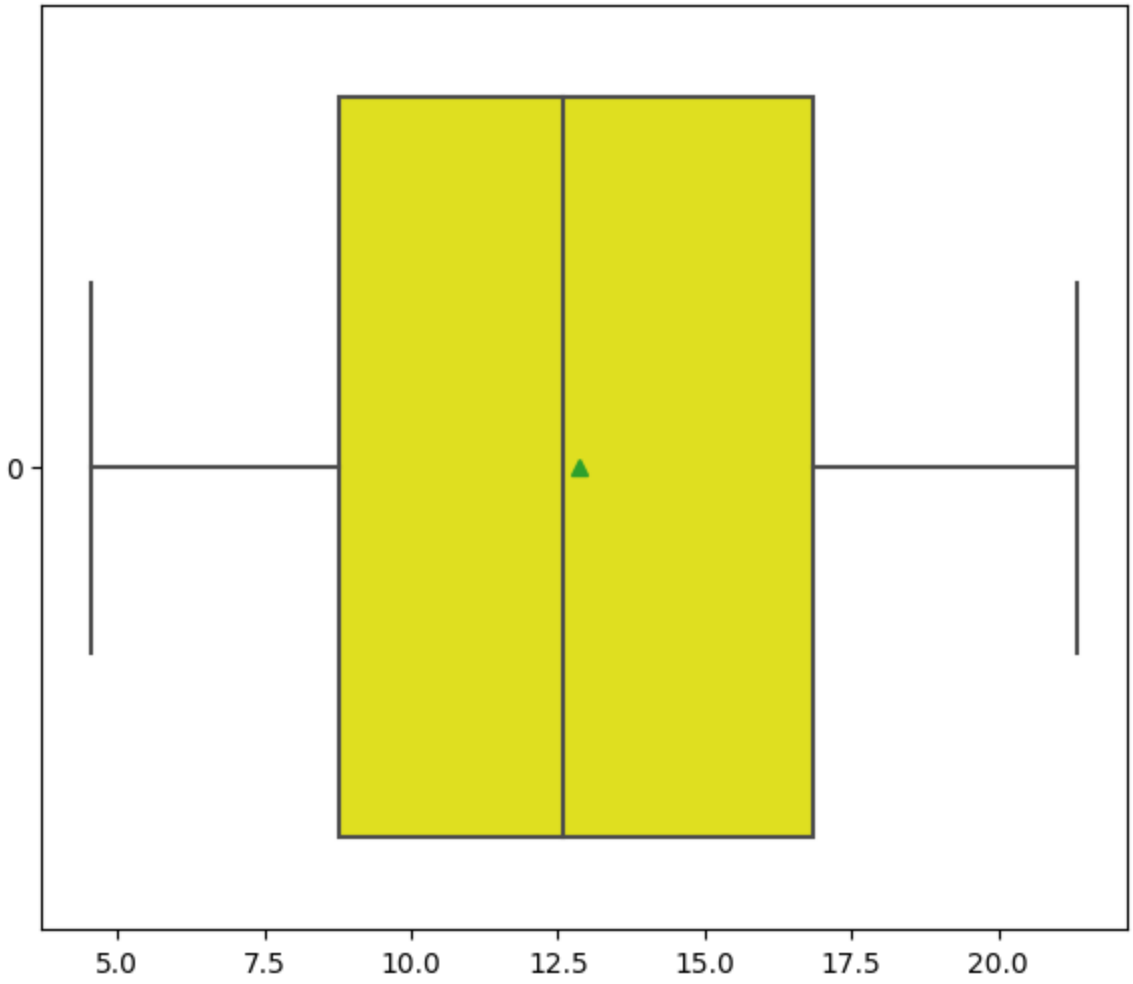
1. Skewness refers to the symmetricity of the data. If the data is not symmetric then we say there is presence of skewness.
2. If the skewness is towards the right then it is right skewed or positively skewed.
3. If the data is negatively skewed then we say it is left skewed or negatively skewed.
4. Guideline for normal distribution is that skewness is between -0.5 to +0.5
5. Skewness between 0.5 to 1 on either side is moderate skewed..
6. Skeweness greater than 1 on either side is highly skewed.
7. Categorical variables do not have skewness. Only numerical variables have skewness.

Univariate Analysis-Boxplot,Violin plots

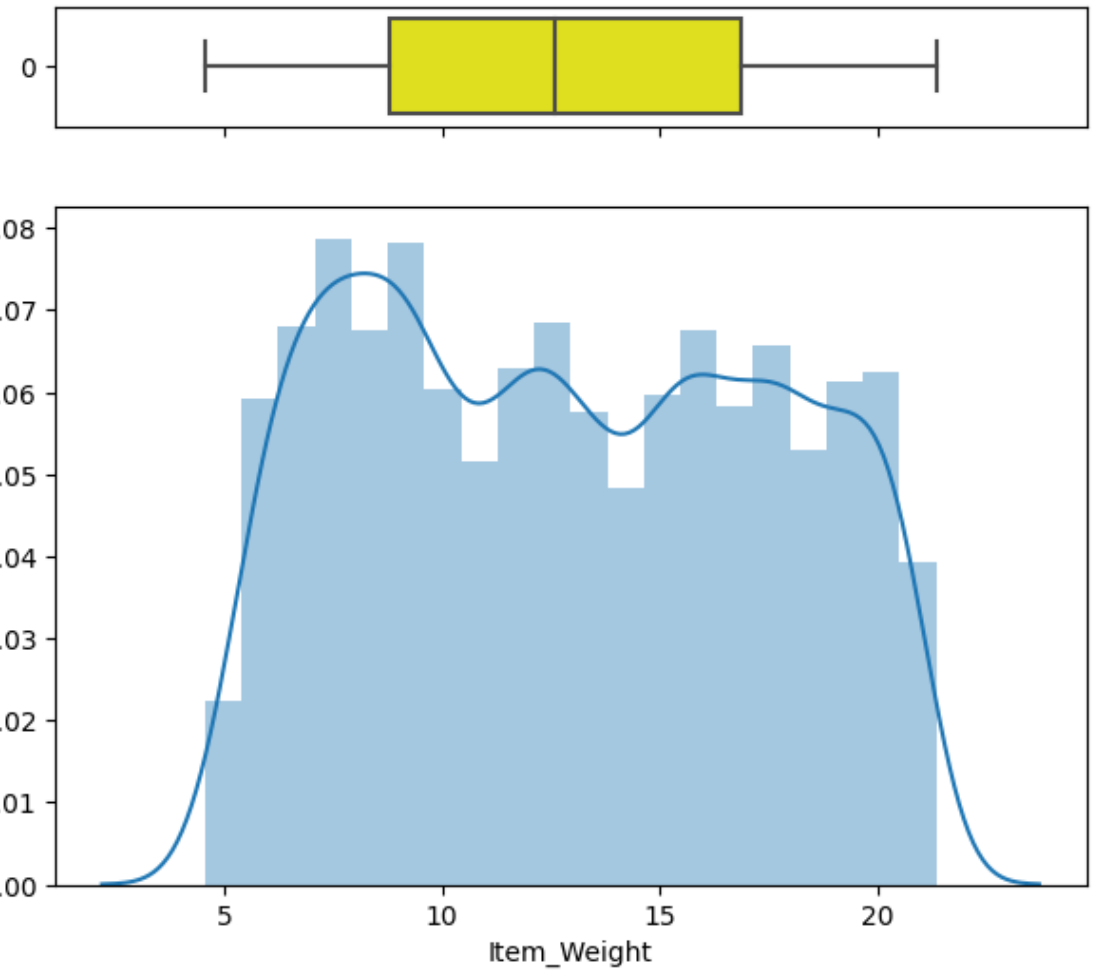
In [ ]:
```
sns.boxplot(df_sales['Item_Weight'],color='yellow');
```



In [ ]:
```
sns.boxplot(df_sales['Item_Weight'],color='yellow',showmeans=True,orient='h');
```

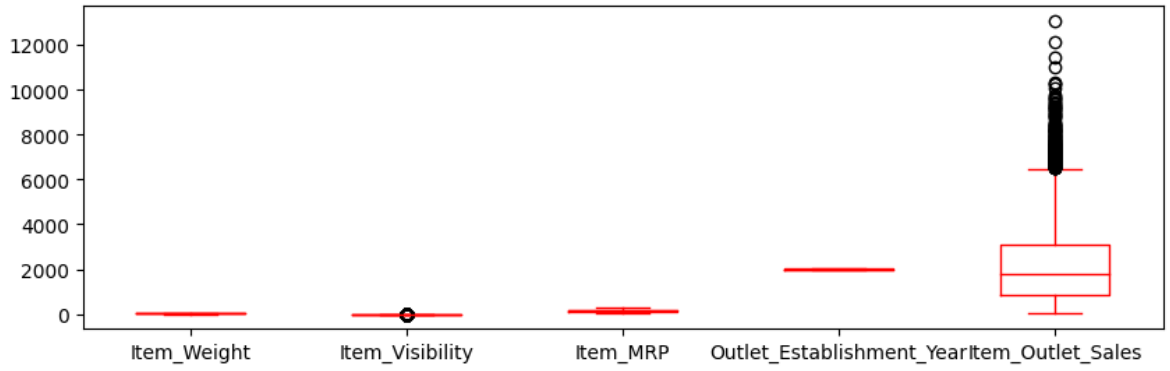


In [ ]:
```
# Lets look at the distribution of Item_Weight
area, (first_box, second_hist) = plt.subplots(2, sharex=True, gridspec_kw={"he
sns.boxplot(df_sales['Item_Weight'], ax=first_box, orient='h',color='yellow')
sns.distplot(df_sales['Item_Weight'], ax=second_hist)
plt.show()
```



In [ ]:

In [ ]:
```
# line, bar,hist,barh,box,kde
plt.rcParams['figure.figsize']=[10,3]
df_sales.plot(kind='box',color='red');
plt.show();
```

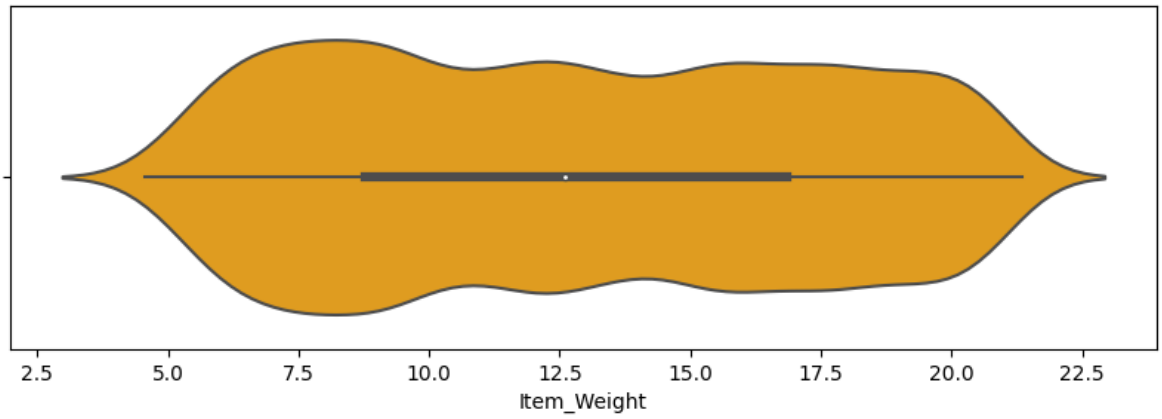


In [ ]:

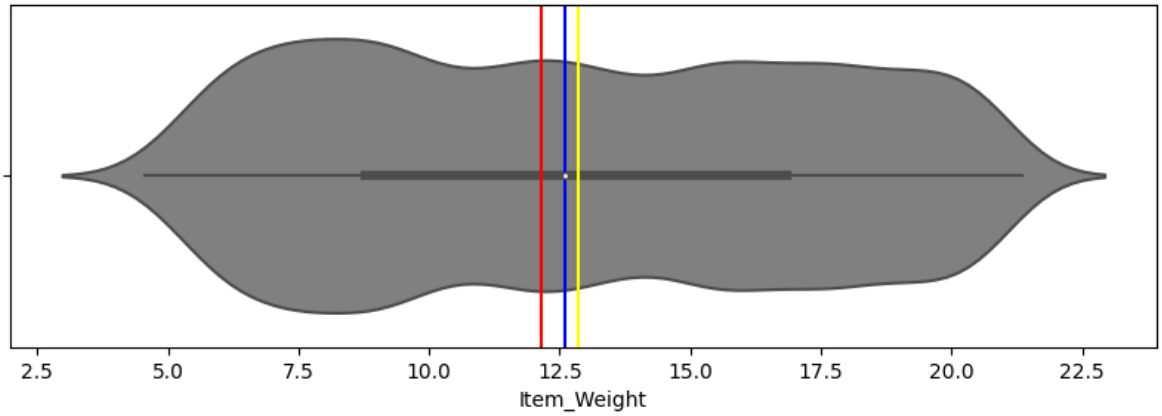
Violin Plots

In [ ]:
```
plt.rcParams['figure.figsize']=[10,3]
```

In [ ]:
```
sns.violinplot(x='Item_Weight',data=df_sales,color='orange');
plt.show()
```

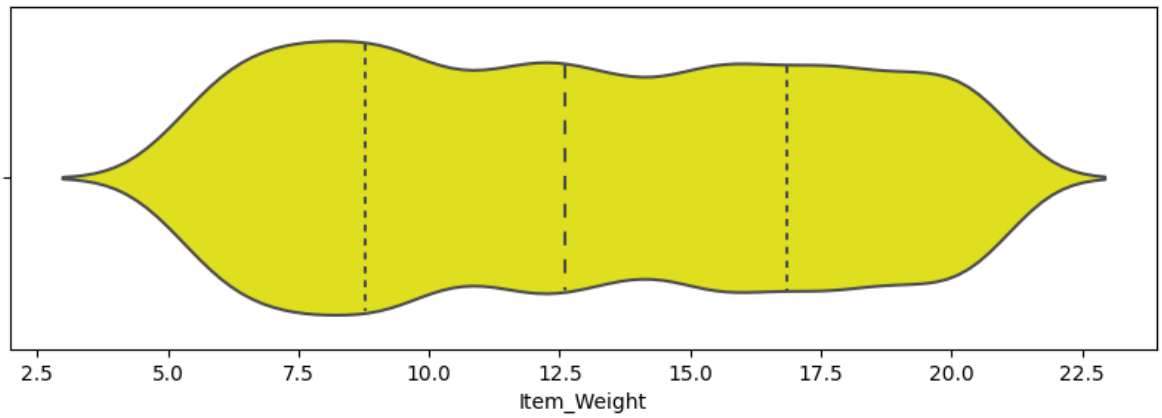


In [ ]:
```
#when we select inner = quartile the box inside the violinplot is replaced
sns.violinplot(x='Item_Weight',data=df_sales,color='grey')
plt.axvline(df_sales['Item_Weight'].mean(),color='yellow')
plt.axvline(df_sales['Item_Weight'].median(),color='blue')
plt.axvline(df_sales['Item_Weight'].mode()[0],color='red')



plt.show()
```



In [ ]:
```
sns.violinplot(x='Item_Weight',data=df_sales,color='yellow', inner='quartiles'
```

Out[63]:
```
<Axes: xlabel='Item_Weight'>
```



In [ ]:
```
```

Range of the data

In [ ]:
```
max(df_sales['Item_Outlet_Sales'])-min(df_sales['Item_Outlet_Sales'])
```

Out[64]:
```
13053.674799999999
```

In [ ]:
```
df_sales['Item_Outlet_Sales'].std()
```

Out[65]:
```
1706.499615733833
```

Range is impacted by the outliers

Lets find the variance

In [ ]:
```
df_sales['Item_Outlet_Sales'].var()
```

Out[66]:
```
2912140.93849972
```

In [ ]:
```
df_sales.var()
```

Out[67]:
```
Item_Weight                  2.156169e+01
Item_Visibility              2.662335e-03
Item_MRP                     3.876071e+03
Outlet_Establishment_Year    7.008637e+01
Item_Outlet_Sales            2.912141e+06
dtype: float64
```

Where is the highest and lowest variance in dataset

In [ ]:
```
min(df_sales.var())
```

Out[68]:
```
0.0026623352682834294
```

In [ ]:
```
max(df_sales.var())
```

Out[69]:
```
2912140.93849972
```

1. Variance measures the spread of the data.
2. Variance is how far is the data spread from the mean. Higher the variance, higher the variation from the mean.
3. Variance is impacted by the mean

Standard deviation

Square root of variance is Standard Deviation

In [ ]:
```
df_sales.std()
```

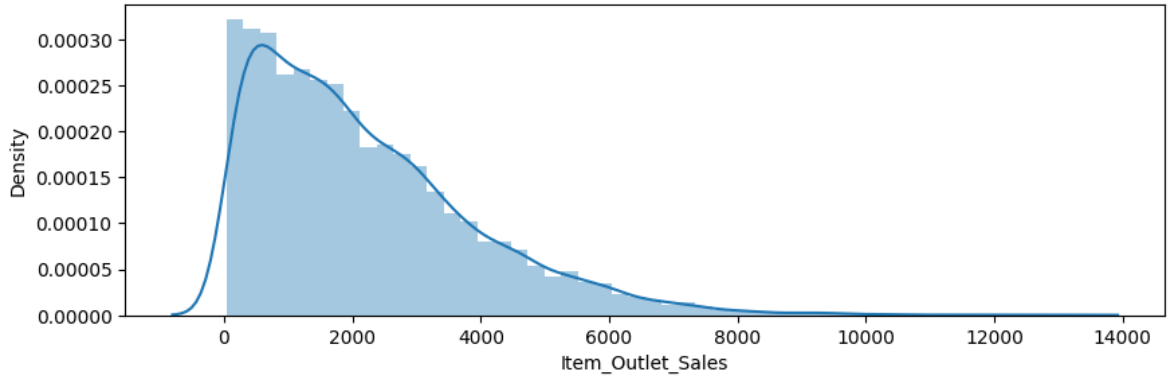
Out[70]:
```
Item_Weight                  4.643456
Item_Visibility              0.051598
Item_MRP                    62.258099
Outlet_Establishment_Year    8.371760
Item_Outlet_Sales         1706.499616
dtype: float64
```

1. Standard deviation measures how spread out the observations are from the mean.
2. Higher the standard deviation, higher is the data spread out from its mean.

Statistical Summary of the data

In [ ]:
```
sns.distplot(df_sales['Item_Outlet_Sales'])
```

Out[71]:
```
<Axes: xlabel='Item_Outlet_Sales', ylabel='Density'>
```



In [ ]:
```
df_sales['Item_Outlet_Sales'].describe()
```

Out[72]:
```
count     8523.000000
mean      2181.288914
std       1706.499616
min         33.290000
25%        834.247400
50%       1794.331000
75%       3101.296400
max      13086.964800
Name: Item_Outlet_Sales, dtype: float64
```

1. This is also called 5 point summary
2. Here the mean is greater than median and hence positively skewed or right skewed.

Coefficient of Variation- CV

1. C.V. measures the dispersion of data around the mean.
2. C.V. is expressed in terms of percent and is unit free.
3. C.V. is used to measure between 2 or more groups
4. C.V. is percent variation in mean where as SD is total variation in mean.

In [ ]:
```
df_sales['Item_Outlet_Sales'].std() / df_sales['Item_Outlet_Sales'].mean()
```

Out[73]:
```
0.7823354371415929
```

In [ ]:
```
cv_sales =df_sales['Item_Outlet_Sales'].std() / df_sales['Item_Outlet_Sales'].
cv_sales
```

Out[74]:
```
0.7823354371415929
```

In [ ]:
```
cv_weight =df_sales['Item_Weight'].std() / df_sales['Item_Weight'].mean()
cv_weight
```

Out[75]:
```
0.36114361787768157
```

In [ ]:
```
cv_mrp =df_sales['Item_MRP'].std() / df_sales['Item_MRP'].mean()
cv_mrp
```

Out[76]:
```
0.4415504385557983
```

In [ ]:
```
df_sales.std() / df_sales.mean()
```

Out[77]:
```
Item_Weight                 0.361144
Item_Visibility             0.780224
Item_MRP                    0.441550
Outlet_Establishment_Year   0.004190
Item_Outlet_Sales           0.782335
dtype: float64
```

1. CV of Item_weight cannot be compared with CV of Item_MRP or anyother variable.

Relationship between the variables

Covariance

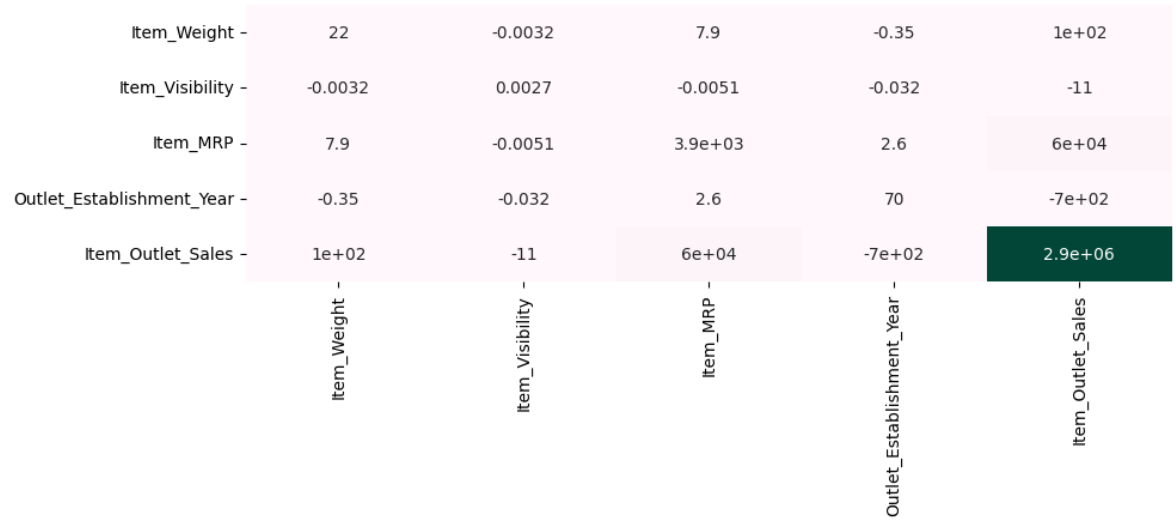
1. Variance explains how the variable varies from its mean.
2. Covariance explains how the two variables variables together vary from its mean.
3. Covariance gives a number but does not tell whether is strong or weak covariance.
4. Covariance tells about the direction of the movement of the variable - X and Y.

In [ ]:
```
df_sales.cov()
```

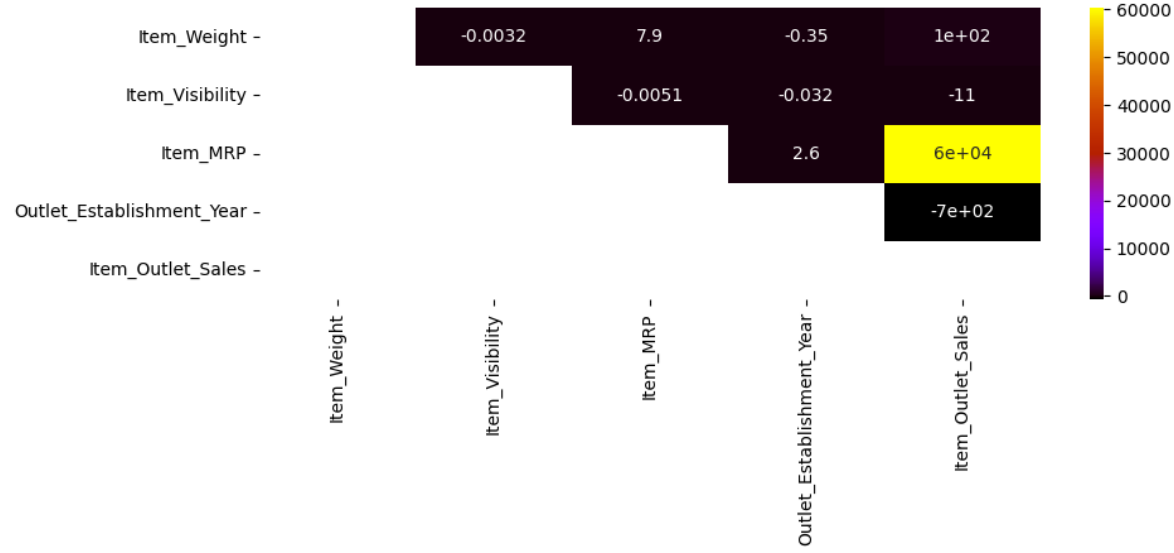
Out[78]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year
Item_Weight	21.561688	-0.003172	7.907894	-0.354551
Item_Visibility	-0.003172	0.002662	-0.005131	-0.032325
Item_MRP	7.907894	-0.005131	3876.070885	2.629821
Outlet_Establishment_Year	-0.354551	-0.032325	2.629821	70.086372
Item_Outlet_Sales	100.560811	-11.325611	60299.006078	-701.962133

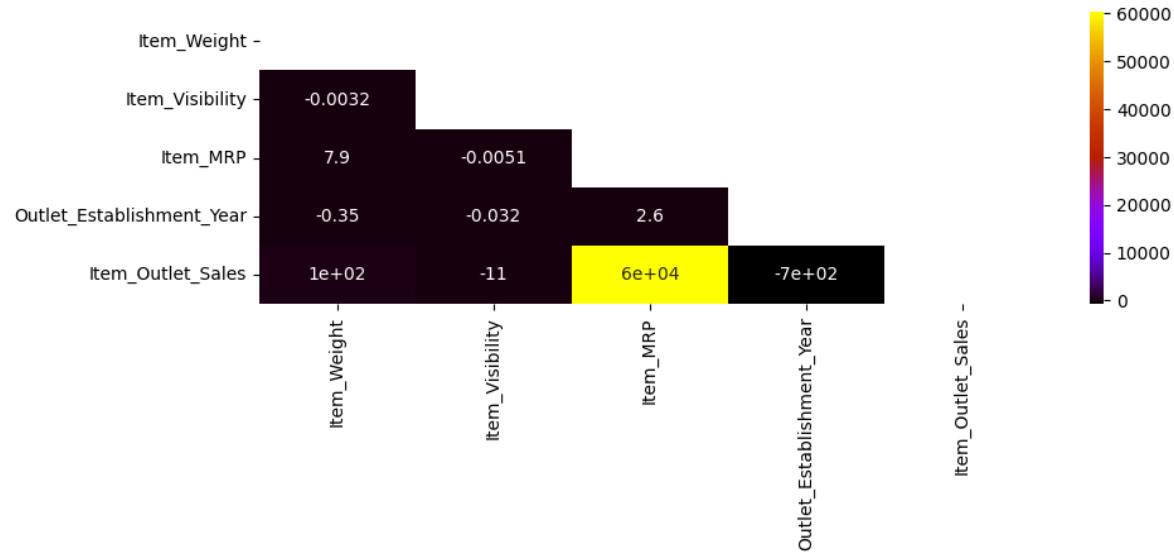
In [ ]:
```
sns.heatmap(df_sales.cov(),annot=True,cmap='PuBuGn', cbar=False);
```



In [ ]:
```
sns.heatmap(df_sales.cov(),mask=np.tril(df_sales.cov()),annot=True,cmap='gnupl
plt.show()
```



In [ ]:
```
sns.heatmap(df_sales.cov(),mask=np.triu(df_sales.cov()),annot=True,cmap='gnupl
plt.show()
```



In [ ]:

Correlation

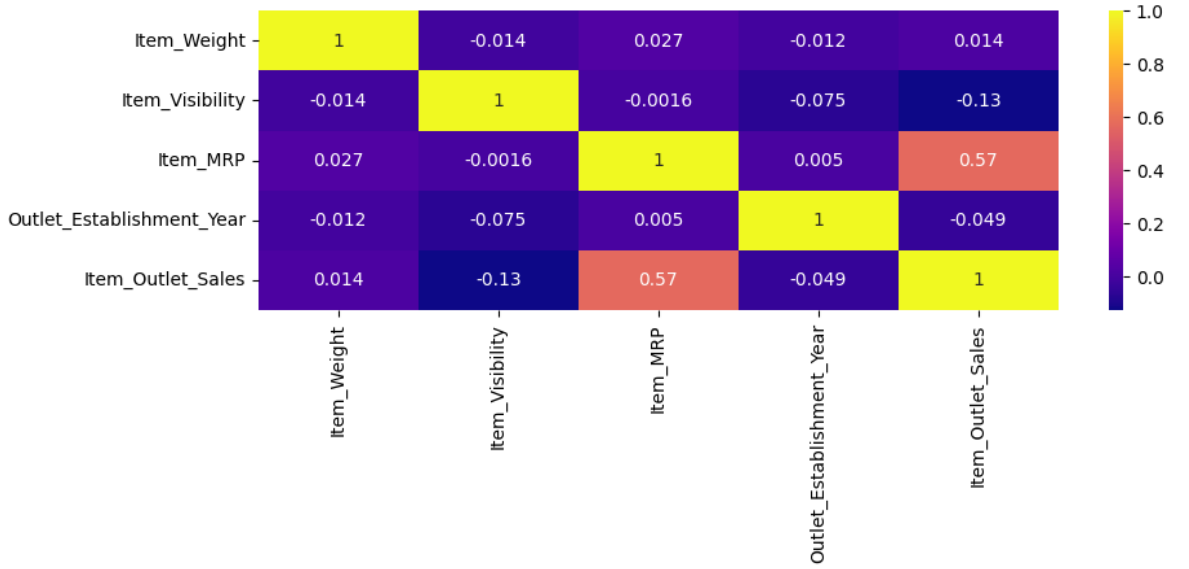
1. It measures the linear relationship between 2 variables.
2. Relationship can be increasing or decreasing.
3. Range of correlation is between -1 to +1.
4. Positive correlation exist if both variable increase/decrease simultaneously.
5. Negative correlation exist if one variable increase and other variable decrease simultaneously.
6. If the covariance is negative then correlation is also negative.
7. Correlation tells the strength of relationship between X and Y.

In [ ]:
```
df_sales.corr()
```

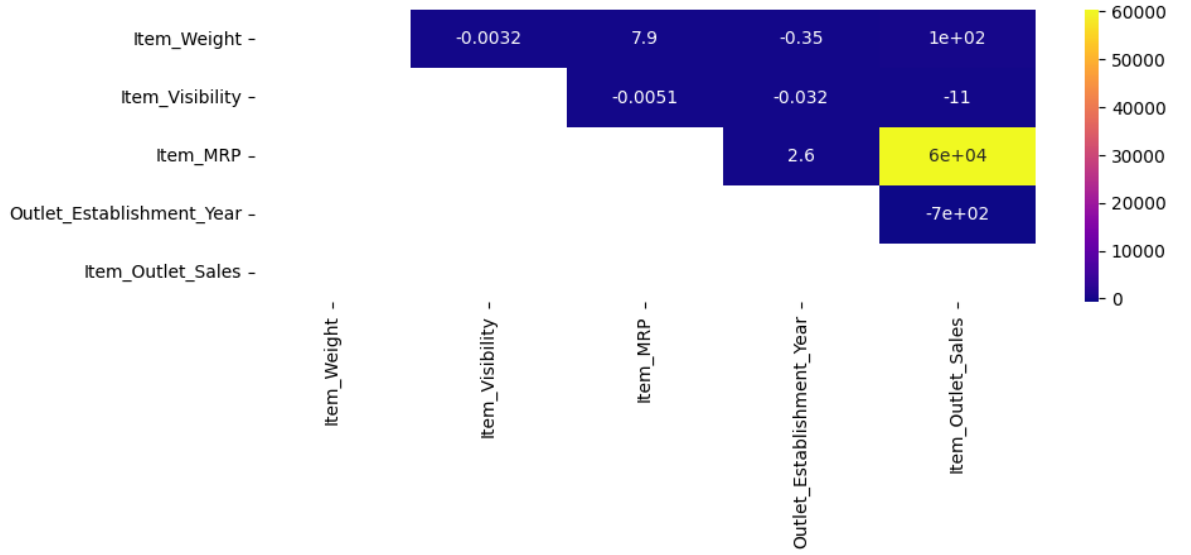
Out[82]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Ite
Item_Weight	1.000000	-0.014048	0.027295	-0.011588	
Item_Visibility	-0.014048	1.000000	-0.001597	-0.074834	
Item_MRP	0.027295	-0.001597	1.000000	0.005046	
Outlet_Establishment_Year	-0.011588	-0.074834	0.005046	1.000000	
Item_Outlet_Sales	0.014123	-0.128625	0.567555	-0.049135	

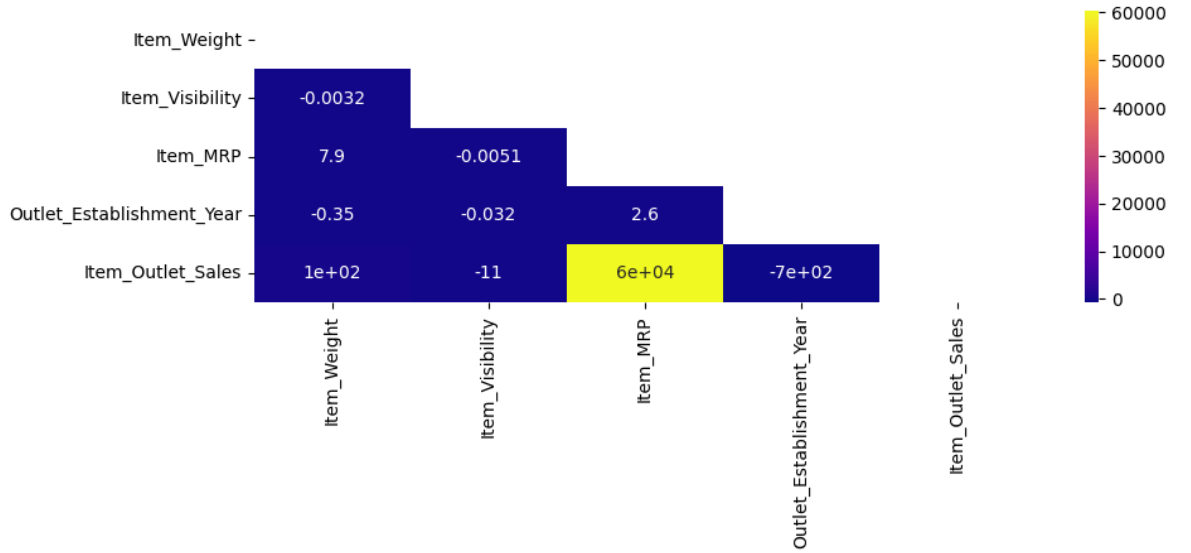
In [ ]:
```
sns.heatmap(df_sales.corr(),annot=True,cmap='plasma', cbar=True);
```



In [ ]:
```
sns.heatmap(df_sales.cov(),mask=np.tril(df_sales.cov()),annot=True,cmap='plasm
plt.show()
```



In [ ]:
```
sns.heatmap(df_sales.cov(),mask=np.triu(df_sales.cov()),annot=True,cmap='plasm
plt.show()
```



In [ ]:

Univariate Analysis- Categorical Variables

In [ ]:
```
df_sales.select_dtypes(include="object")
```

Out[86]:

	Item_Identifier	Item_Fat_Content	Item_Type	Outlet_Identifier	Outlet_Size	Outlet_Location_
0	FDA15	Low Fat	Dairy	OUT049	Medium	
1	DRC01	Regular	Soft Drinks	OUT018	Medium	
2	FDN15	Low Fat	Meat	OUT049	Medium	
3	FDX07	Regular	Fruits and Vegetables	OUT010	NaN	
4	NCD19	Low Fat	Household	OUT013	High	
...	...	...	...	...	...	
8518	FDF22	Low Fat	Snack Foods	OUT013	High	
8519	FDS36	Regular	Baking Goods	OUT045	NaN	
8520	NCJ29	Low Fat	Health and Hygiene	OUT035	Small	
8521	FDN46	Regular	Snack Foods	OUT018	Medium	
8522	DRG01	Low Fat	Soft Drinks	OUT046	Small	

8523 rows × 7 columns

In [ ]:
```
df_sales.select_dtypes(include=np.number)
```

Out[87]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
0	9.300	0.016047	249.8	1999	3735.1380
1	5.920	0.019278	48.3	2009	443.4228
2	17.500	0.016760	141.6	1999	2097.2700
3	19.200	0.000000	182.1	1998	732.3800
4	8.930	0.000000	53.9	1987	994.7052
...	...	...	...	...	...
8518	6.865	0.056783	214.5	1987	2778.3834
8519	8.380	0.046982	108.2	2002	549.2850
8520	10.600	0.035186	85.1	2004	1193.1136
8521	7.210	0.145221	103.1	2009	1845.5976
8522	14.800	0.044878	75.5	1997	765.6700

8523 rows × 5 columns

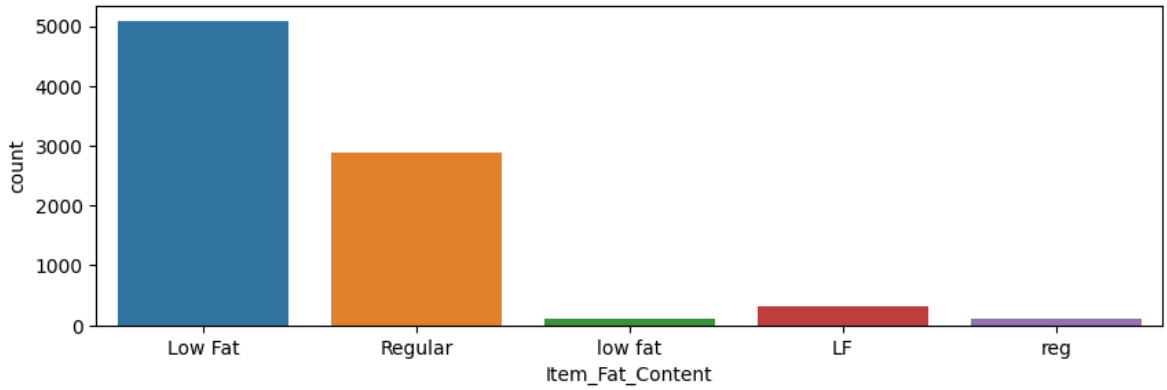
In [ ]:
```
df_sales.select_dtypes('object').columns
```

Out[88]:
```
Index(['Item_Identifier', 'Item_Fat_Content', 'Item_Type', 'Outlet_Identifie
r',
       'Outlet_Size', 'Outlet_Location_Type', 'Outlet_Type'],
      dtype='object')
```

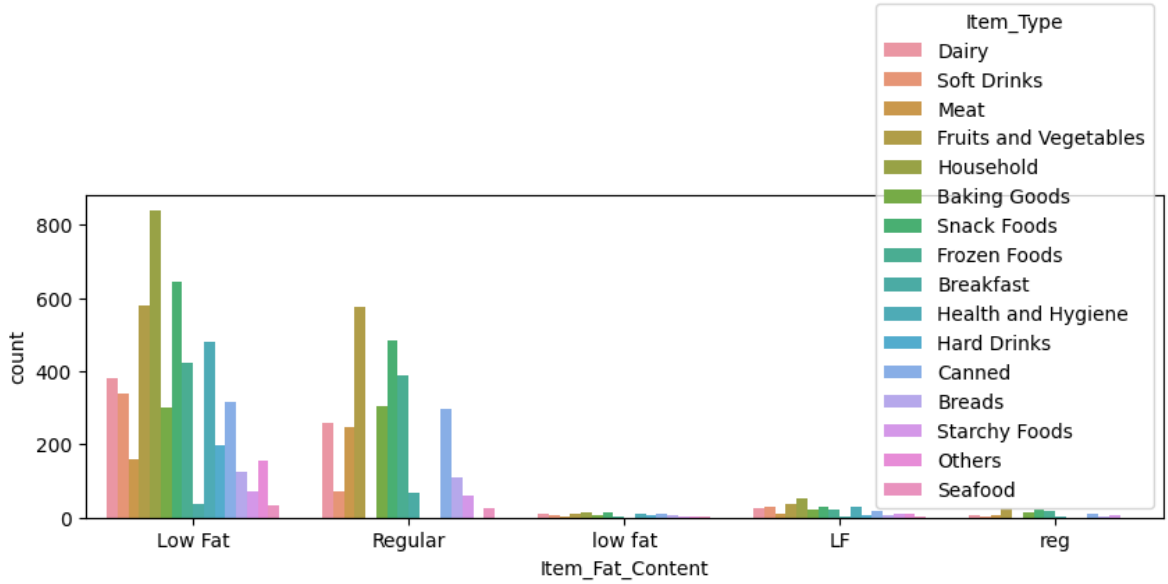
Use of For Loops

In [ ]:
```
cols=['Item_Fat_Content', 'Item_Type', 'Outlet_Identifier',
       'Outlet_Size', 'Outlet_Location_Type', 'Outlet_Type']
```

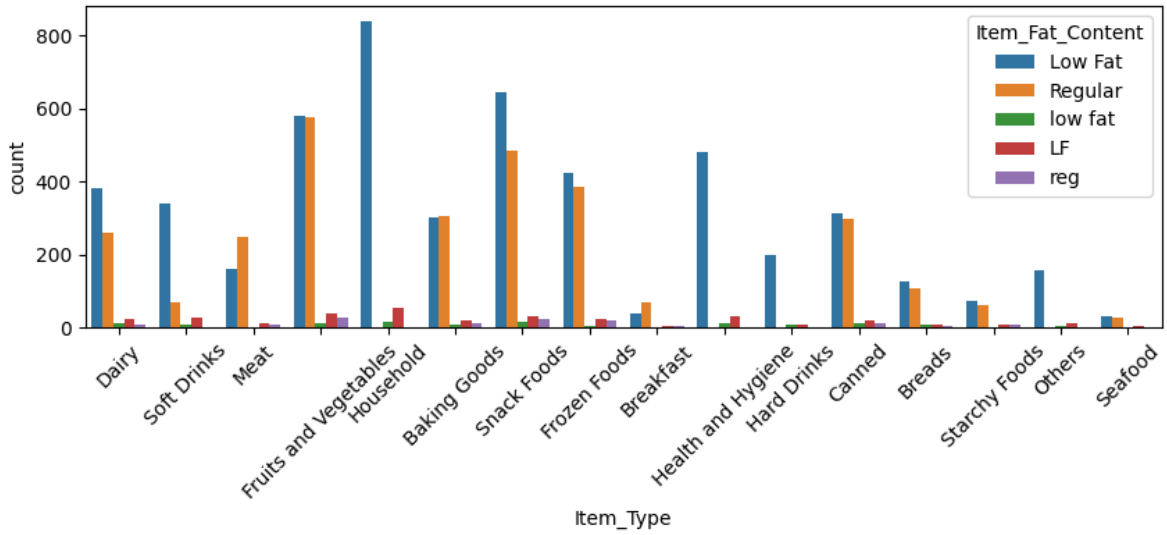
In [ ]:
```
sns.countplot(x=df_sales['Item_Fat_Content'],data=df_sales);
```



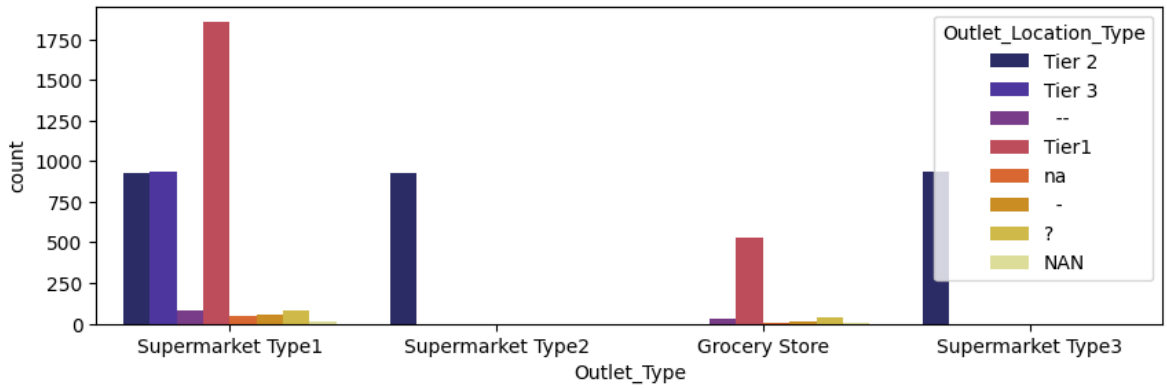
In [ ]:
```
sns.countplot(x=df_sales['Item_Fat_Content'],data=df_sales, hue='Item_Type');
```



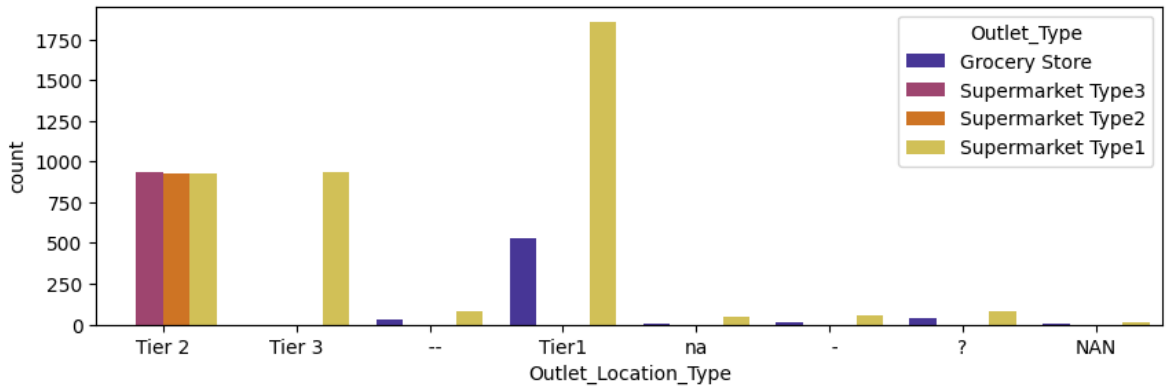
In [ ]:
```
sns.countplot(x=df_sales['Item_Type'],data=df_sales, hue='Item_Fat_Content');
plt.xticks(rotation=45)
plt.show()
```



In [ ]:
```
sns.countplot(x='Outlet_Type',hue='Outlet_Location_Type',data=df_sales,palette
plt.show()
```

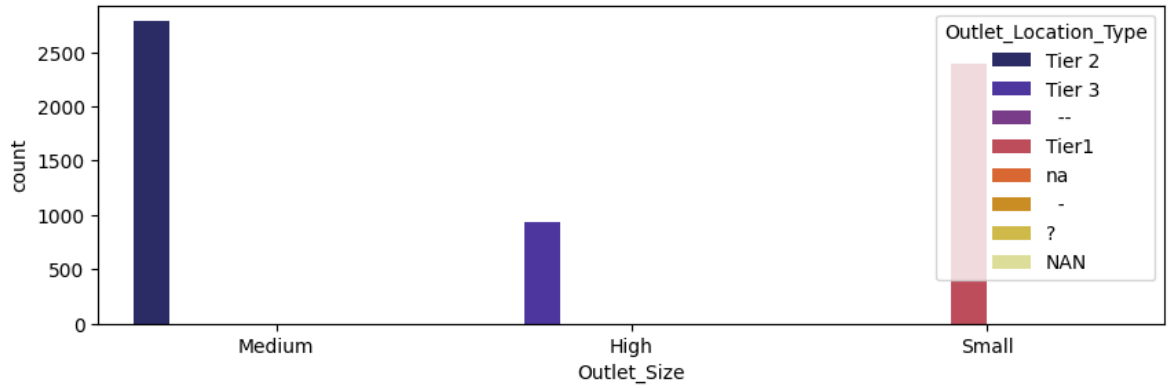


In [ ]:
```
hue_orders=['Grocery Store','Supermarket Type3','Supermarket Type2','Supermark
sns.countplot(x='Outlet_Location_Type',hue='Outlet_Type',data=df_sales,palette
plt.show()
```

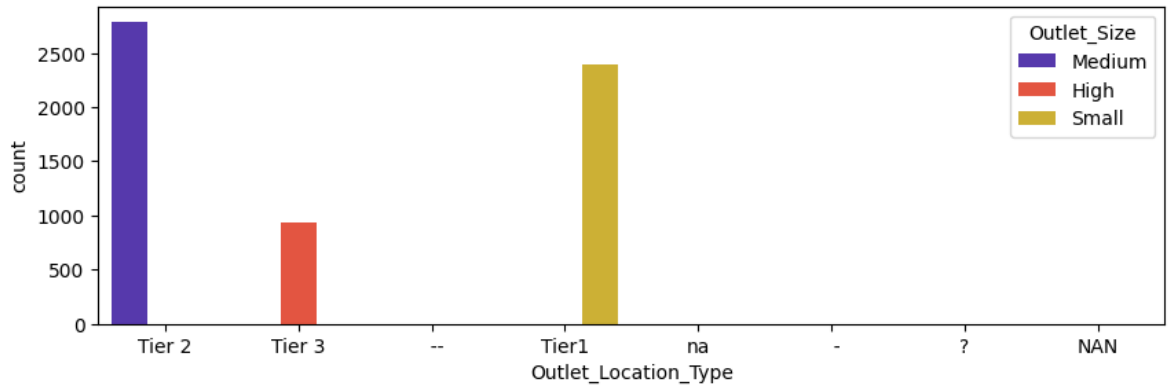


In [ ]:

In [ ]:
```
sns.countplot(x='Outlet_Size',hue='Outlet_Location_Type',data=df_sales,palette
plt.show()
```

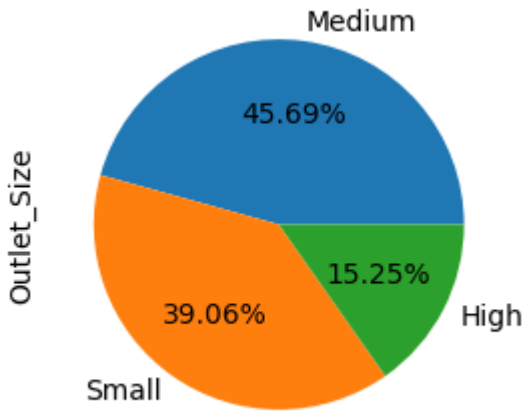


In [ ]:
```
sns.countplot(x='Outlet_Location_Type',hue='Outlet_Size',data=df_sales,palette
plt.show()
```

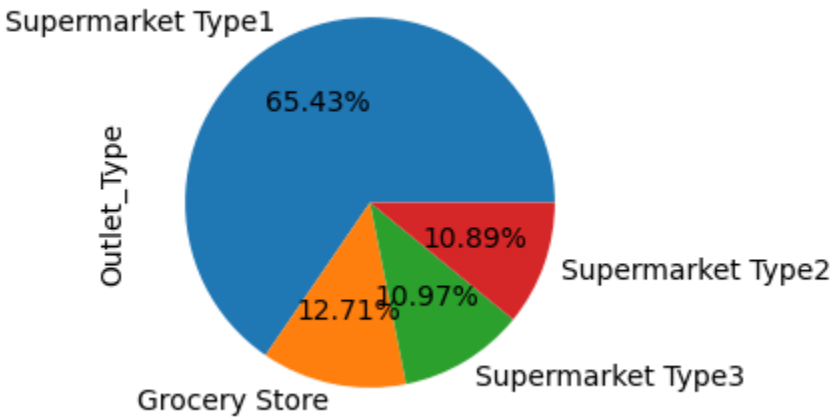


In [ ]:
```

```

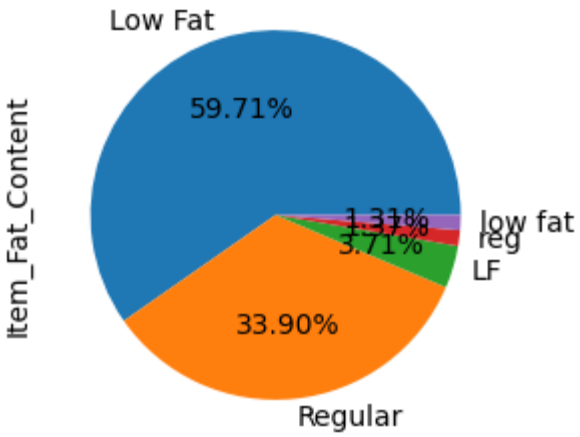
In [ ]:
```
df_sales['Outlet_Size'].value_counts().plot(kind='pie',autopct='%.2f%%');
```



In [ ]:
```
df_sales['Outlet_Type'].value_counts().plot(kind='pie',autopct='%.2f%%');
```



In [ ]:
```
df_sales['Item_Fat_Content'].value_counts().plot(kind='pie',autopct='%.2f%%');
```



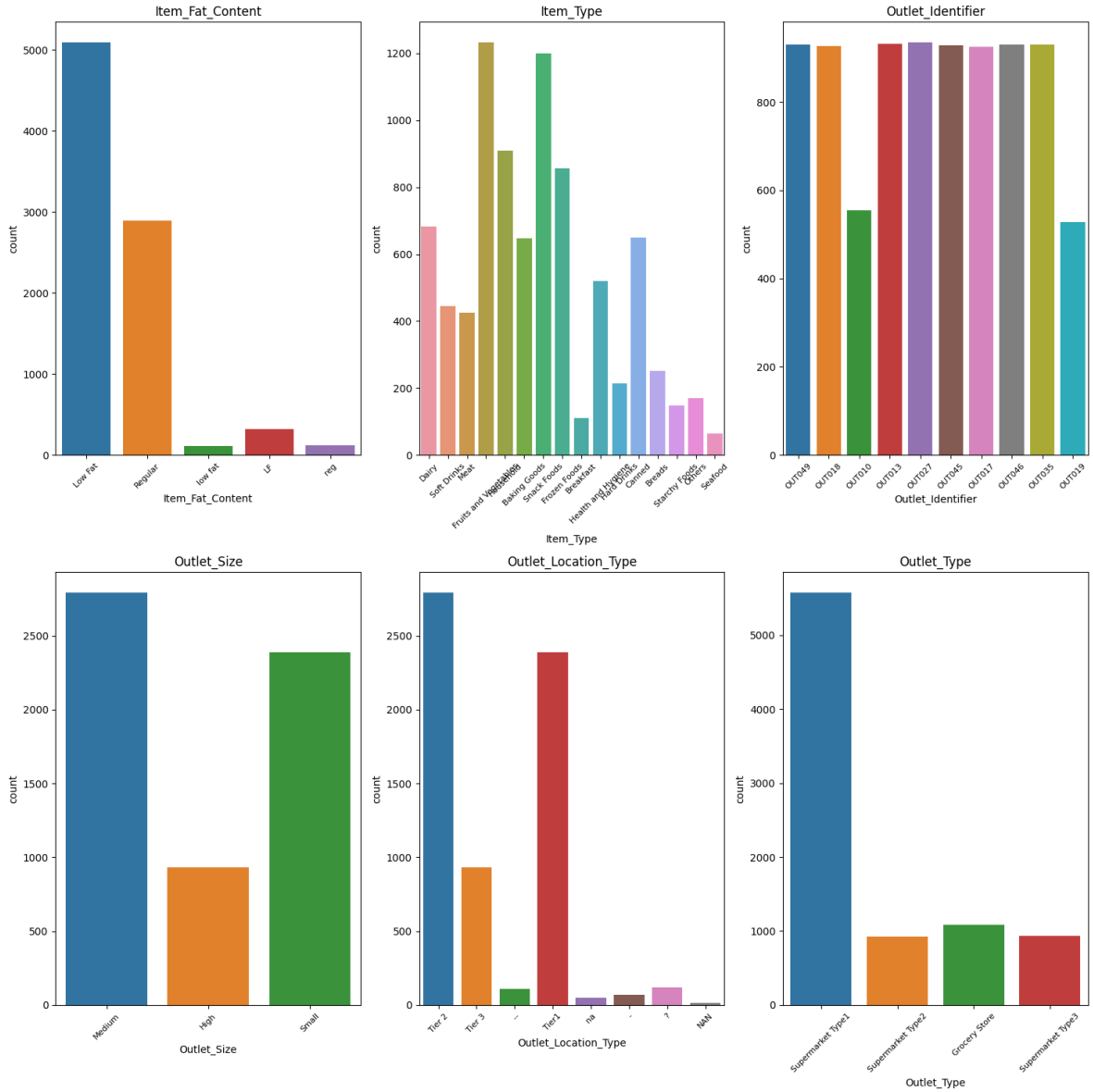
In [ ]:
```
```

In [ ]:
```
df_sales.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       6473 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 [ ]:
```
rcParams['figure.figsize'] = 15,15
```

In [ ]:
```
j=1
for i in cols:
  plt.subplot(2,3,j)
  sns.countplot(x=df_sales.loc[:,i])
  plt.title(i)
  j = j+1

  plt.xticks(rotation=45, fontsize=8)

plt.tight_layout()
plt.show()
```



1. The Item type like snack food, vegetables, fruits are top selling items.
2. Frequency of outlets is highest in tier 3 cities.
3. Low Fat products are sold the most.
4. Outlets of Medium size are more than the large and small sized outlet.

Inferenence in terms of Business -Observed patterns

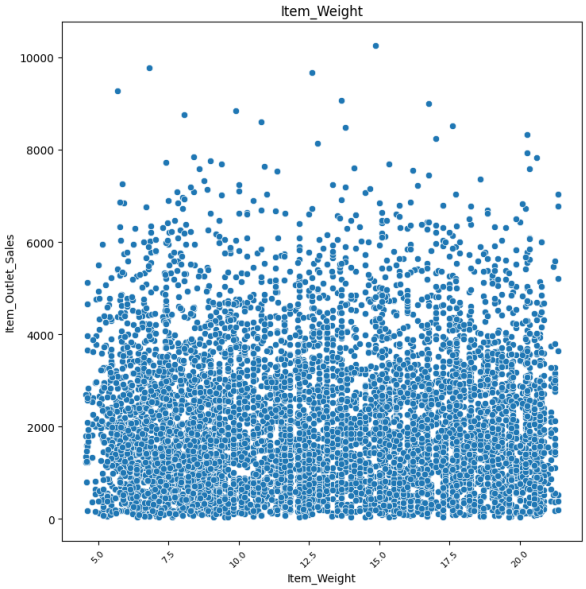
Bivariate Analysis - Numerical to Numerical

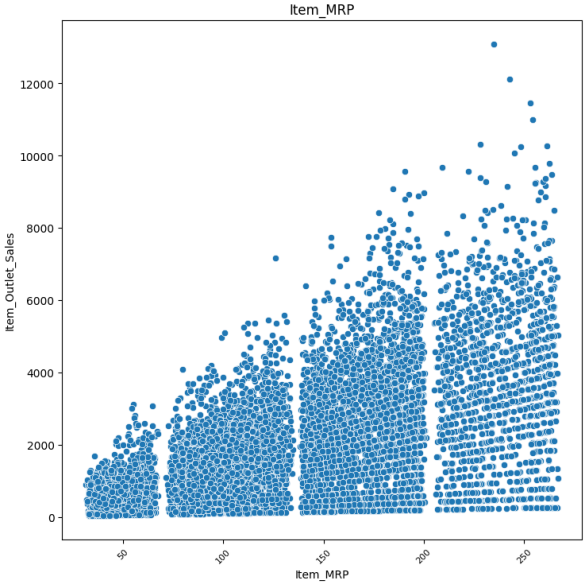
In [ ]:
```
df_sales.select_dtypes(include=np.number).columns
```

Out[103]:
```
Index(['Item_Weight', 'Item_Visibility', 'Item_MRP',
       'Outlet_Establishment_Year', 'Item_Outlet_Sales'],
      dtype='object')
```

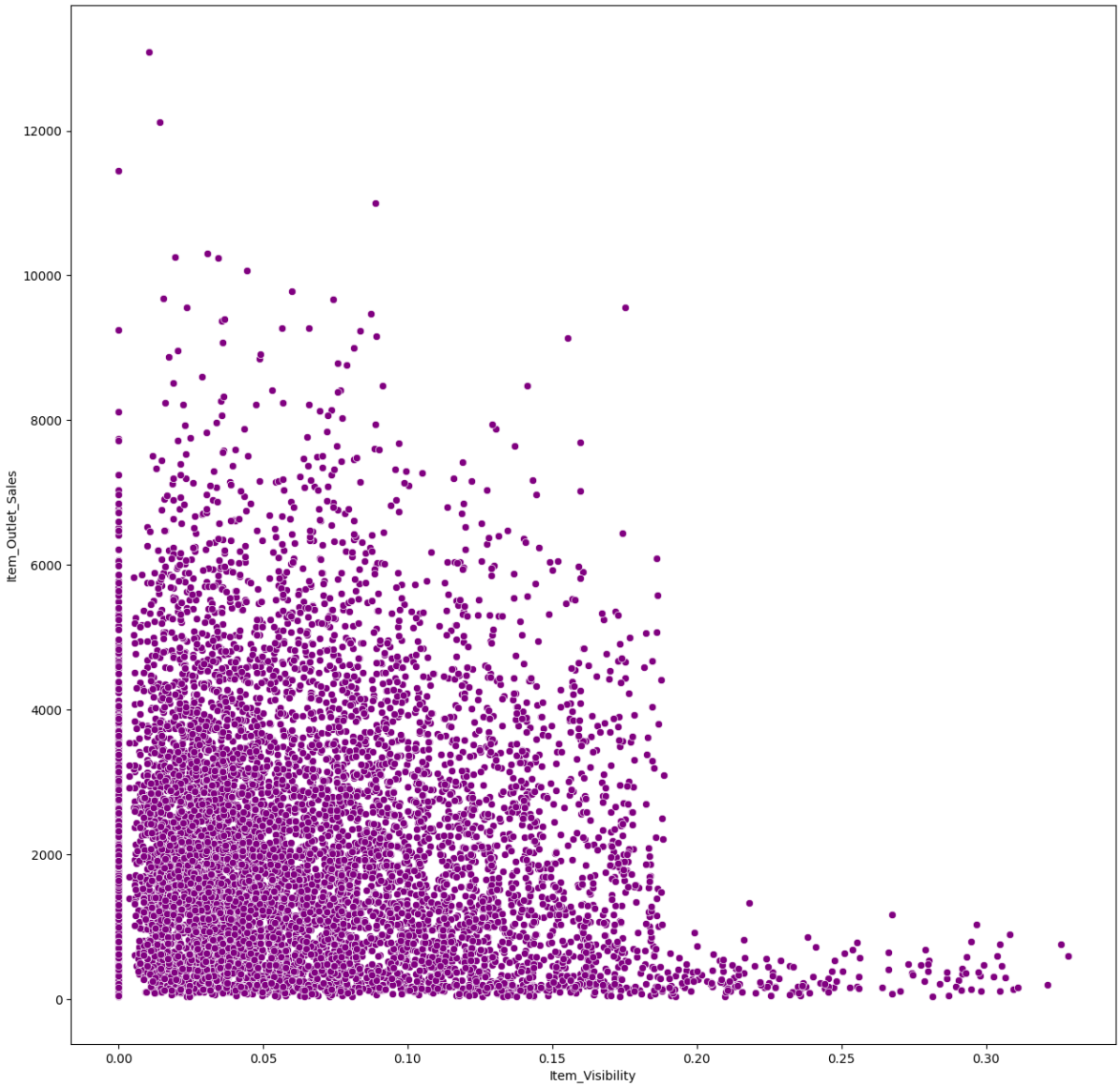
In [ ]:
```
cols=['Item_Weight', 'Item_Visibility', 'Item_MRP']
```

In [ ]:
```
j=1
for i in cols:
  plt.subplot(2,2,j)
  sns.scatterplot(x=df_sales.loc[:,i], y = df_sales['Item_Outlet_Sales'])
  plt.title(i)
  j = j+1

  plt.xticks(rotation=45, fontsize=8)

plt.tight_layout()
plt.show()
```





In [ ]:
```
sns.scatterplot(x=df_sales['Item_Visibility'], y=df_sales['Item_Outlet_Sales'])
plt.show()
```

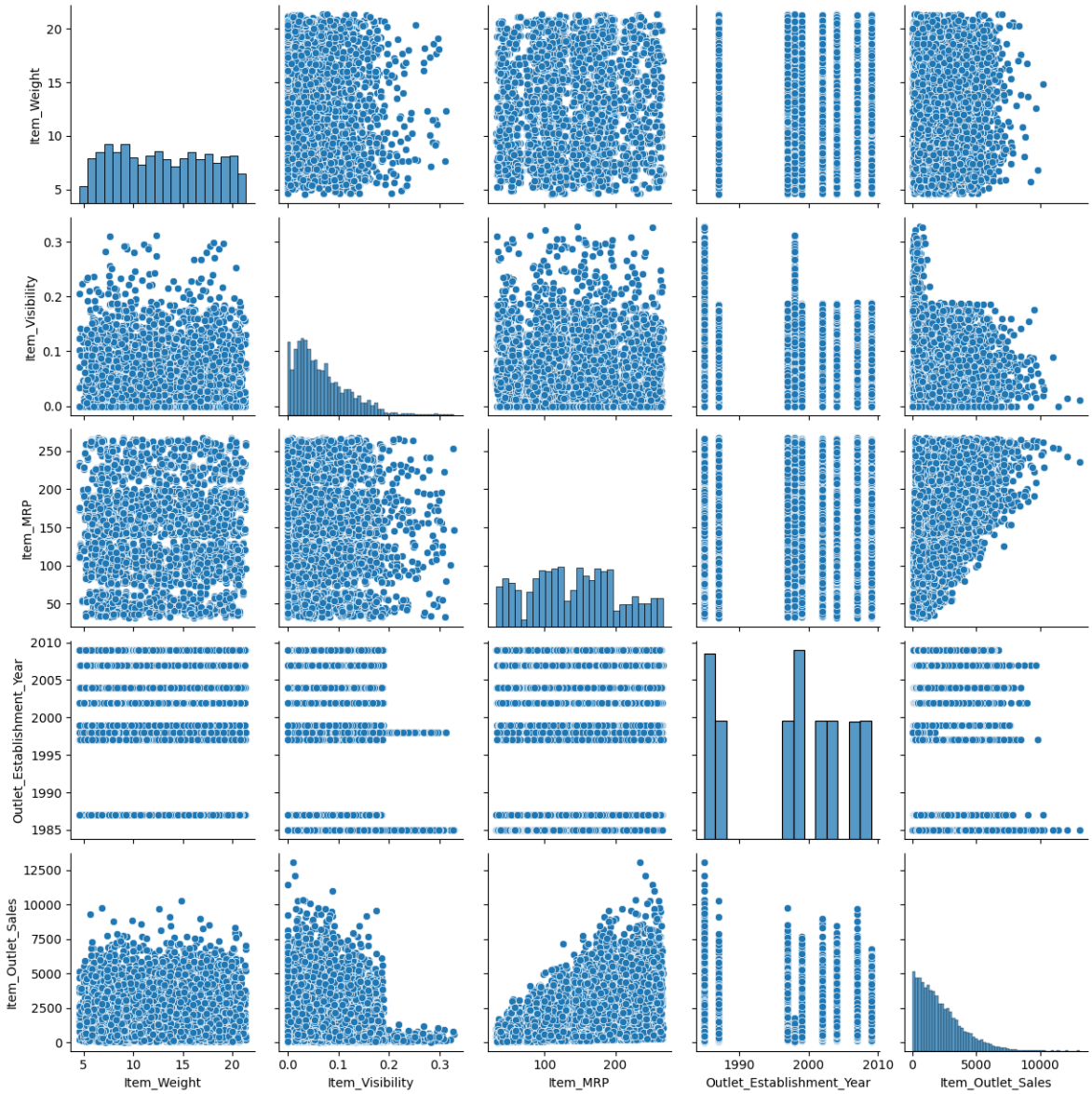


Inference

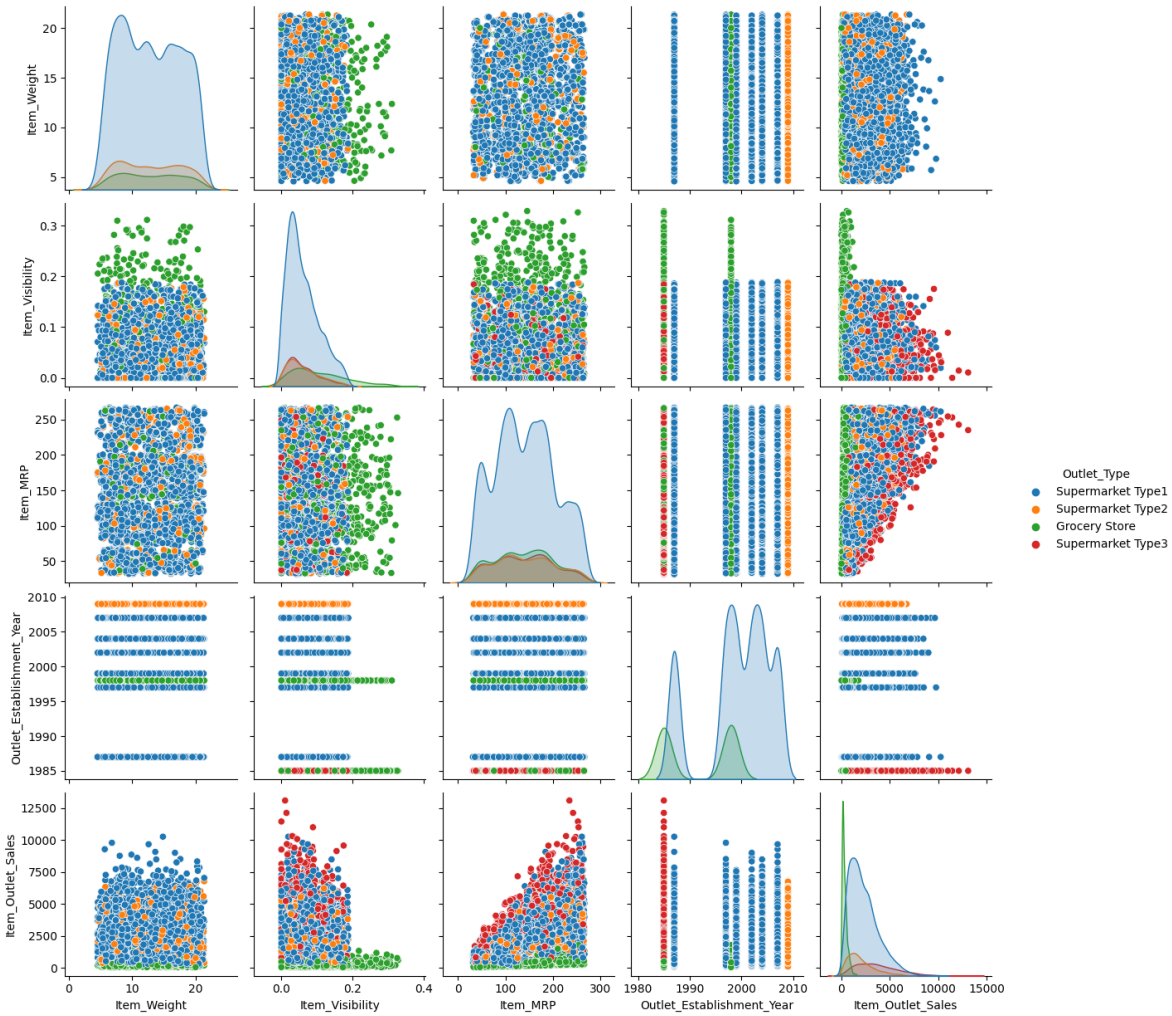
1. There is no specific pattern observed between Item weight and outlet sales.
2. There is very weak correlation between Item weight and outlet sales.
3. As the MRP increases, the outlet sales increases. This is a observed pattern.
4. The bigger products have higher visibility. High visibility have lower sales.
5. Smaller products have low visibility but has higher sales.
6. Item visibility with 0 has high sales-anamoly.

Pair Plots

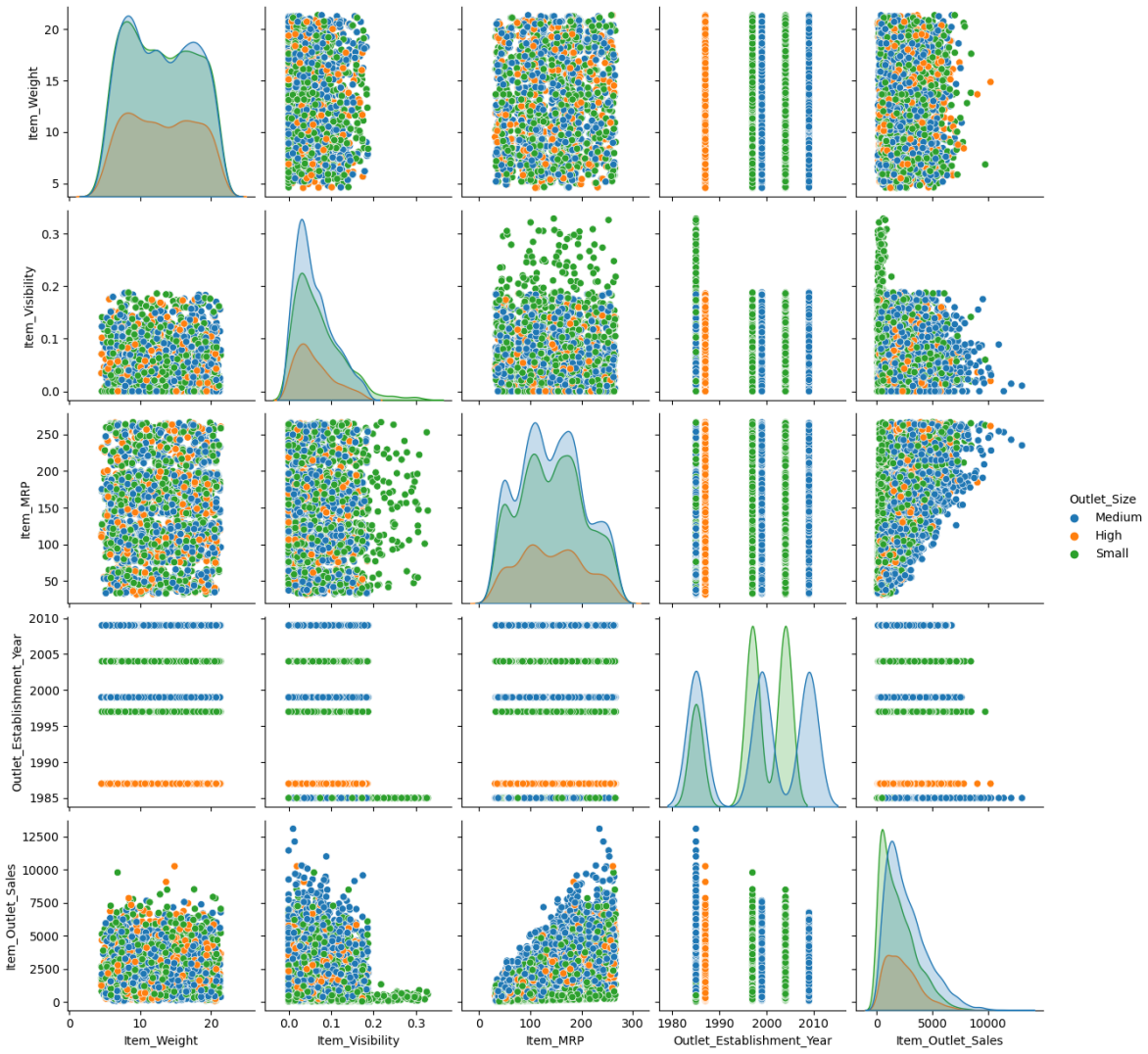
In [ ]:
```
sns.pairplot(df_sales);
```



In [ ]:
```
sns.pairplot(df_sales,hue='Outlet_Type');
```

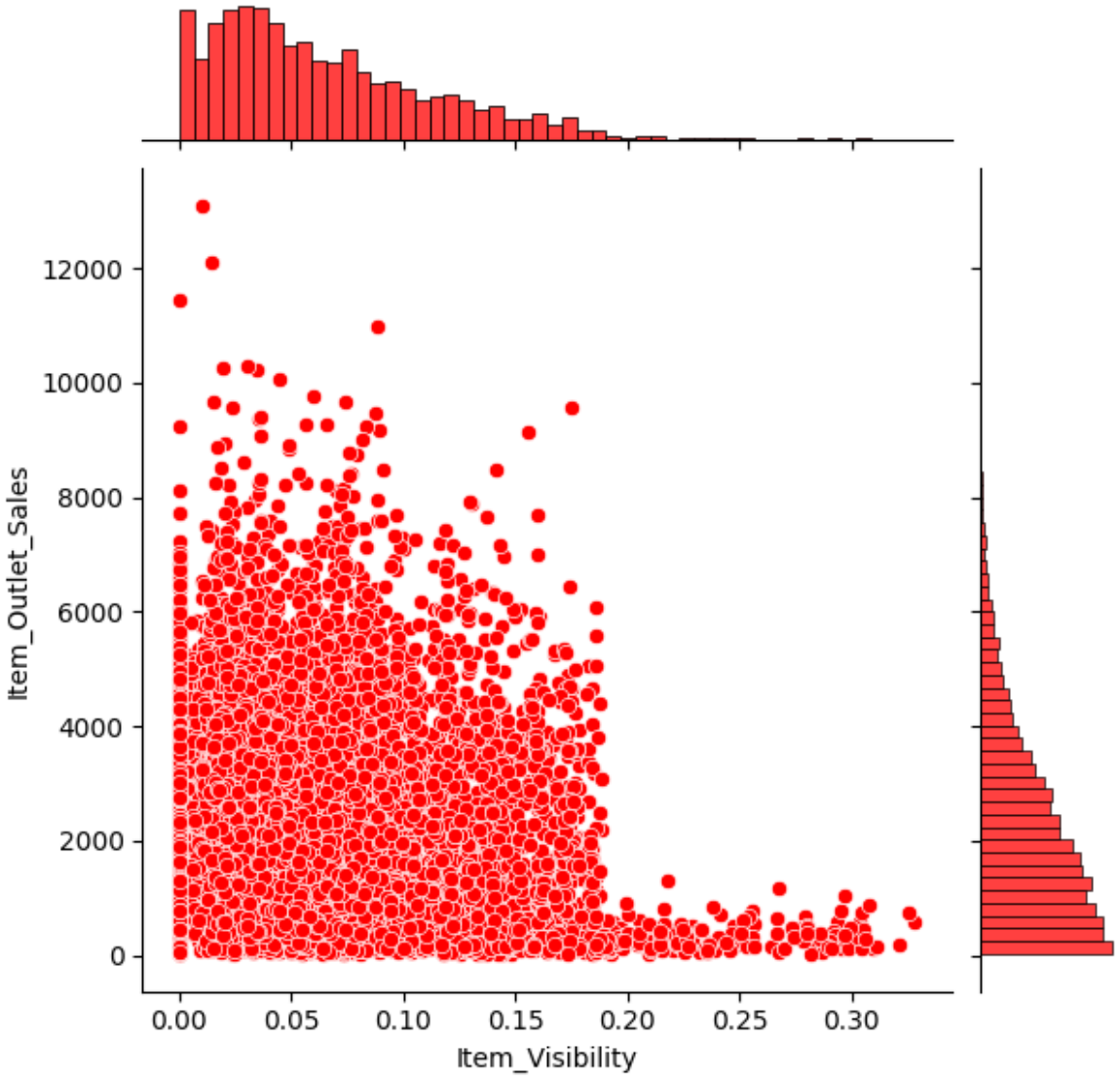


In [ ]:
```
sns.pairplot(df_sales,hue='Outlet_Size');
```

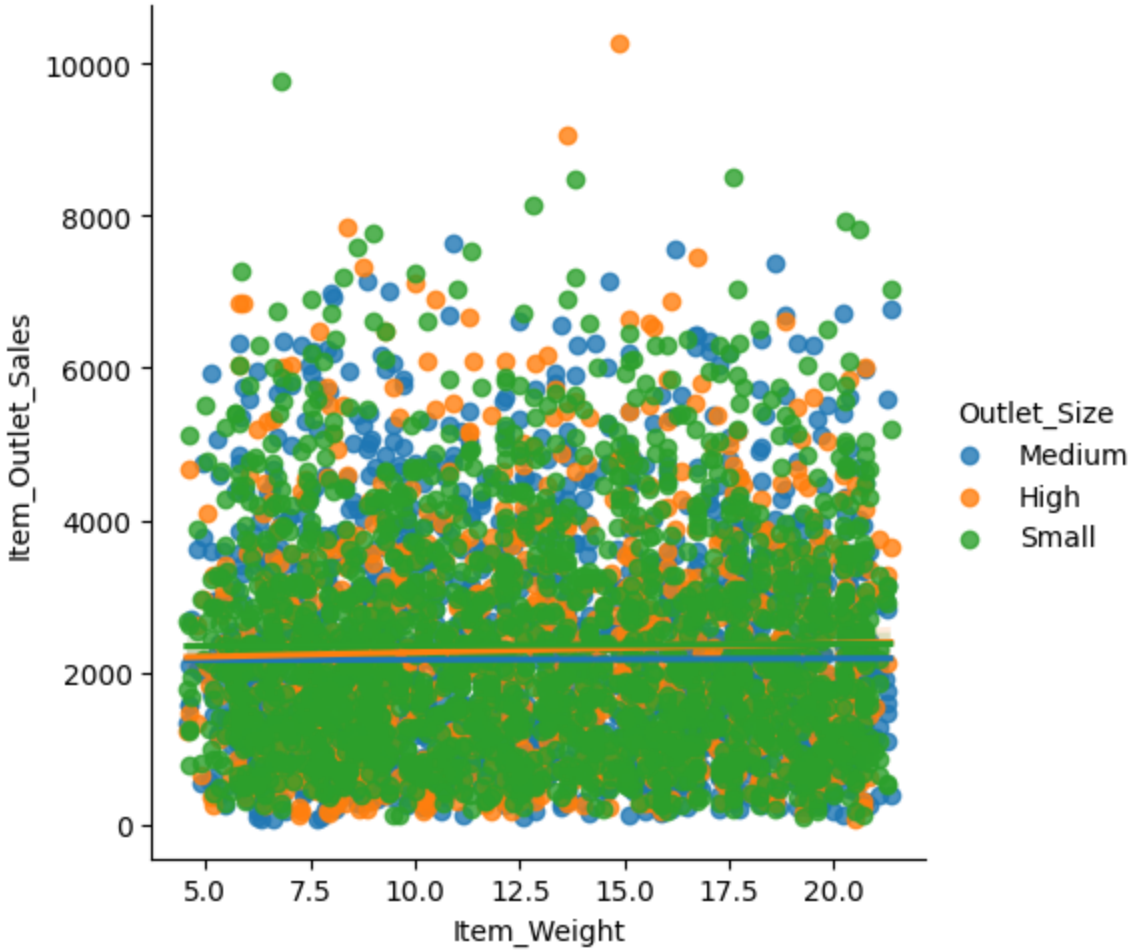


Joint Plot

In [ ]:
```
# kind : { "scatter" | "kde" | "hist" | "hex" | "reg" | "resid" }
sns.jointplot(x='Item_Visibility',y='Item_Outlet_Sales',data=df_sales,color='r
plt.show()
```



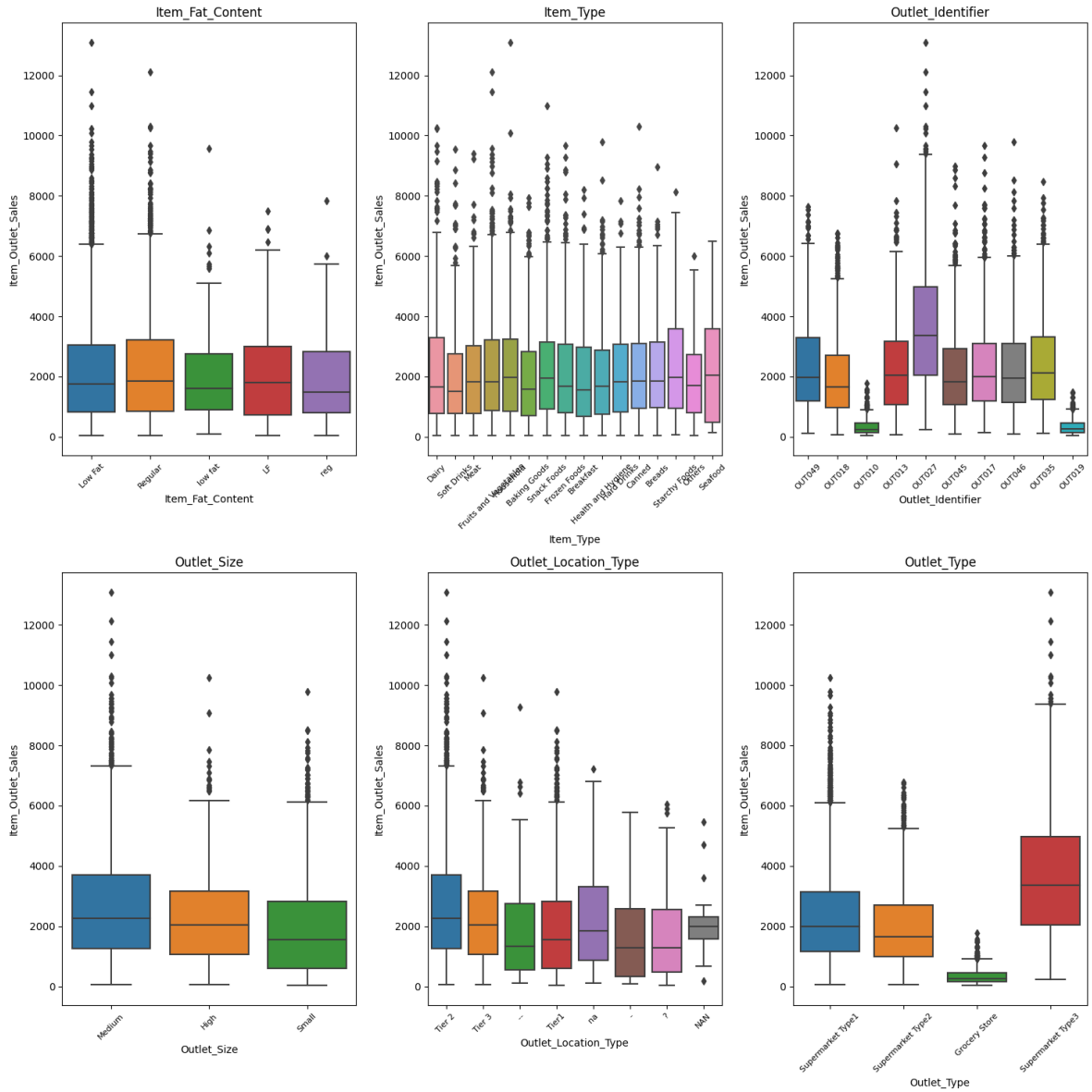
In [ ]:
```
sns.lmplot(x='Item_Weight',y='Item_Outlet_Sales',data=df_sales,hue='Outlet_Siz
plt.show()
```



Bivariate Ananlysis - Categorical to Numerical

In [ ]:
```
cols=['Item_Fat_Content', 'Item_Type', 'Outlet_Identifier',
       'Outlet_Size', 'Outlet_Location_Type', 'Outlet_Type']
```

In [ ]:
```
j=1
for i in cols:
  plt.subplot(2,3,j)
  sns.boxplot(x=df_sales.loc[:,i], y = df_sales['Item_Outlet_Sales'])
  plt.title(i)
  j = j+1

  plt.xticks(rotation=45, fontsize=8)

plt.tight_layout()
plt.show()
```

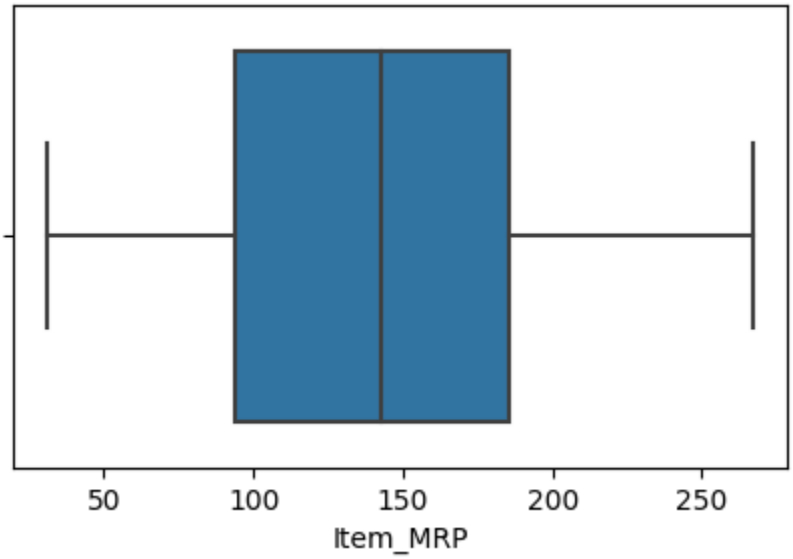


1. Outlet 27 are the ones which are selling the highest.
2. Low fat items are selling more.
3. Household are selling the most.
4. All the plots shows outliers.
5. Outlet size of medium are greater in number.
6. Tier 3 cities have the highest sales.
7. Super market type 3 has the highest sales.

In [ ]:
```
rcParams['figure.figsize'] = 5,3
```

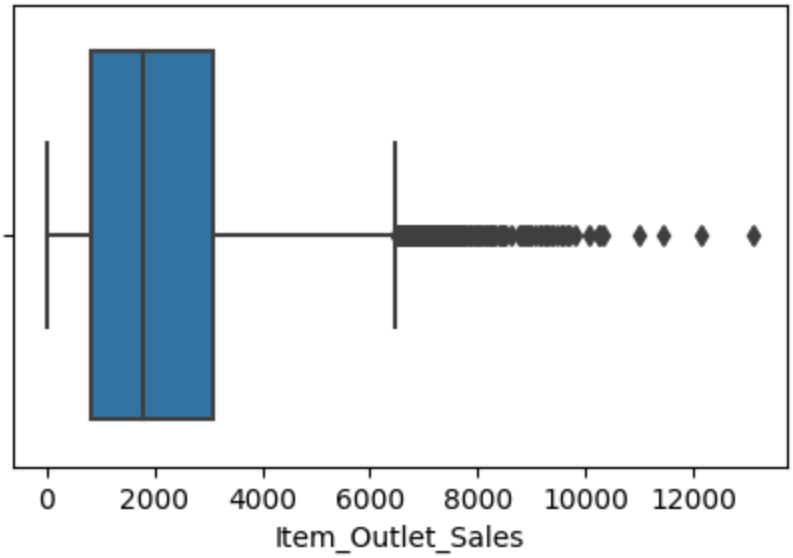
In [ ]:
```
sns.boxplot(x=df_sales['Item_MRP'])
```

Out[1041]:
```
<Axes: xlabel='Item_MRP'>
```



In [ ]:
```
sns.boxplot(x=df_sales['Item_Outlet_Sales'])
```

Out[1042]:
```
<Axes: xlabel='Item_Outlet_Sales'>
```

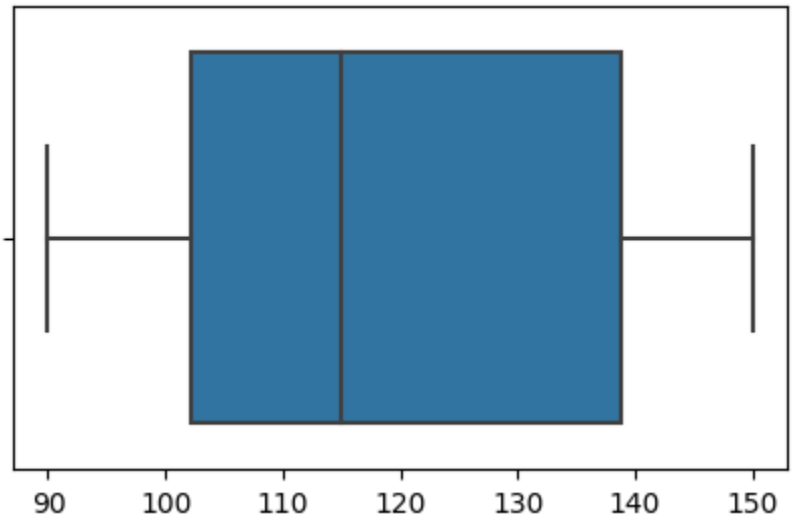


In [ ]:
```
Grade_1 = pd.Series([150,145,143,140,135,130,120,110,105,103,102,100,95,90])
Grade_2 = pd.Series([250,145,143,140,135,130,120,110,105,103,102,101,100,95,12
Grade_3 = pd.Series([125,123,122,120,120,119,119,110,105,103,102,101,100,95,90
Grade_4 = pd.Series([130,129,129,128,127,126,120,110,105,103,102,101,100,80,80
```

CASE 1 : Upper Limit and Lower Limit will stretch till the Max and Min values

In [ ]:
```
sns.boxplot(x=Grade_1)
```

Out[1044]:
```
<Axes: >
```



In [ ]:
```
Grade_1.describe()
```

Out[1045]:
```
count     14.000000
mean     119.142857
std       20.813668
min       90.000000
25%      102.250000
50%      115.000000
75%      138.750000
max      150.000000
dtype: float64
```

In [ ]:
```
# Upper Value and Lower Value
q1 =102.25
q3 = 138.75
```

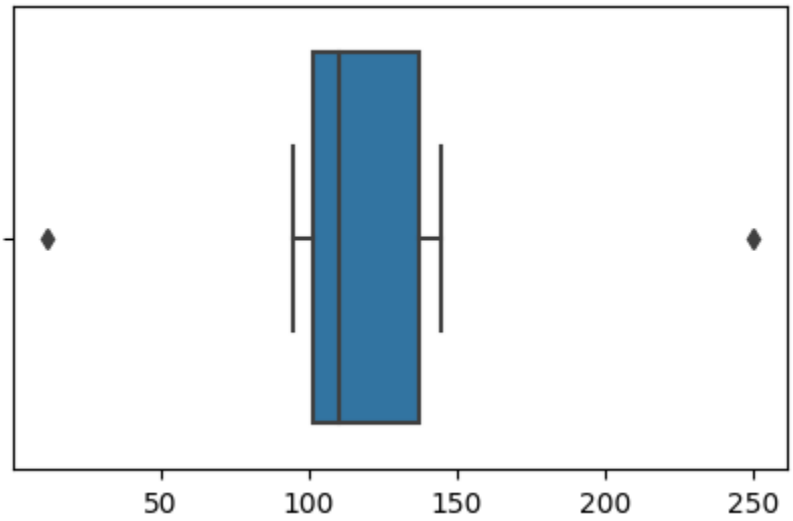
In [ ]:
```

iqr = q3-q1
UL =q3 + 1.5 * iqr
LL = q1 - 1.5 * iqr

print('Upper Limit is', UL)
print('Lower Limit is',LL)
```

```
Upper Limit is 193.5
Lower Limit is 47.5
```

CASE 2 : Max values are above Upper Limit and Min value is lower than Lower

In [ ]:
```
sns.boxplot(x=Grade_2)
```

Out[1048]: <Axes: >



In [ ]:
```
Grade_2.describe()
```

Out[1049]:
```
count     15.000000
mean     119.400000
std       48.374728
min       12.000000
25%      101.500000
50%      110.000000
75%      137.500000
max      250.000000
dtype: float64
```

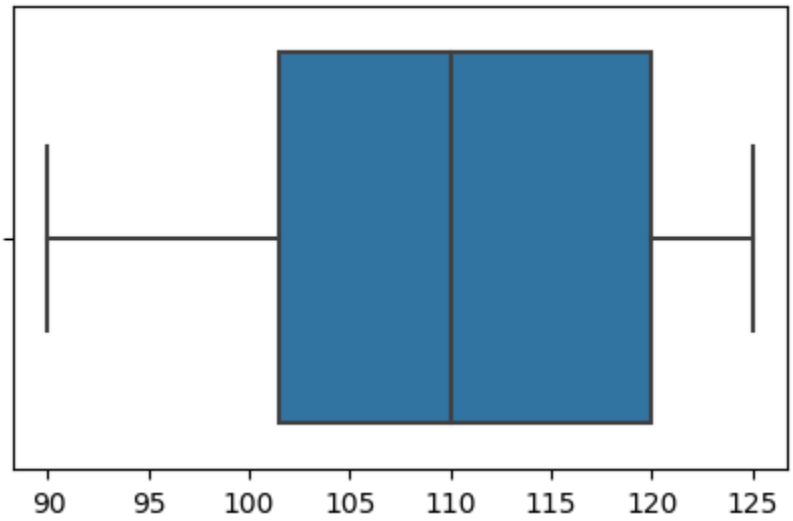
In [ ]:
```
# Upper Value and Lower Value
q1 =101.5
q3 = 137.5


iqr = q3-q1
UL =q3 + 1.5 * iqr
LL = q1 - 1.5 * iqr

print('Upper Limit is', UL)
print('Lower Limit is',LL)
```

```
Upper Limit is 191.5
Lower Limit is 47.5
```

CASE 3 : Max values are below Upper Limit and Min value is above than Lower

In [ ]:
```
sns.boxplot(x=Grade_3)
```

Out[1051]:
```
<Axes: >
```



In [ ]:
```
Grade_3.describe()
```

Out[1052]:
```
count     15.000000
mean     110.266667
std       11.473365
min       90.000000
25%      101.500000
50%      110.000000
75%      120.000000
max      125.000000
dtype: float64
```

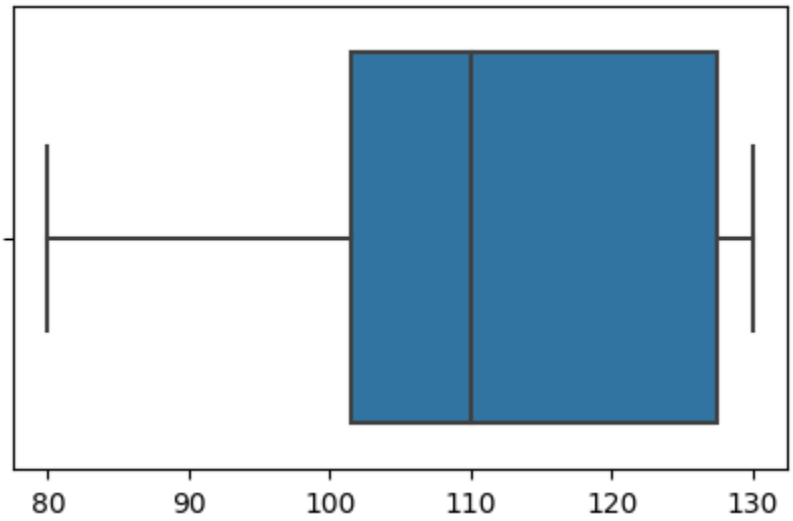
In [ ]:
```
# Upper Value and Lower Value
q1 = 90
q3 = 120


iqr = q3-q1
UL =q3 + 1.5 * iqr
LL = q1 - 1.5 * iqr

print('Upper Limit is', UL)
print('Lower Limit is',LL)
```

```
Upper Limit is 165.0
Lower Limit is 45.0
```

CASE 4 : Max values are below Upper Limit and Min value is above than Lower

In [ ]:
```
sns.boxplot(x=Grade_4)
```

Out[1054]: <Axes: >



In [ ]:
```
Grade_4.describe()
```

Out[1055]:
```
count     15.000000
mean     111.333333
std       17.286108
min       80.000000
25%      101.500000
50%      110.000000
75%      127.500000
max      130.000000
dtype: float64
```

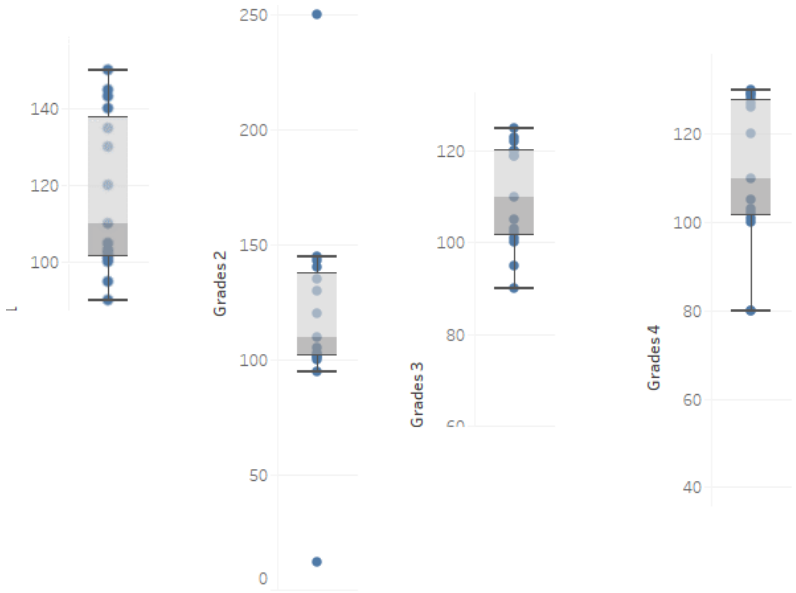
In [ ]:
```
# Upper Value and Lower Value
q1 = 101.50
q3 = 127.50


iqr = q3-q1
UL =q3 + 1.5 * iqr
LL = q1 - 1.5 * iqr

print('Upper Limit is', UL)
print('Lower Limit is',LL)
```

```
Upper Limit is 166.5
Lower Limit is 62.5
```

Snapshots of All the above cases

In [ ]:
```
from IPython.display import Image
Image(filename='Boxplots.png',width=500,height=300)
```

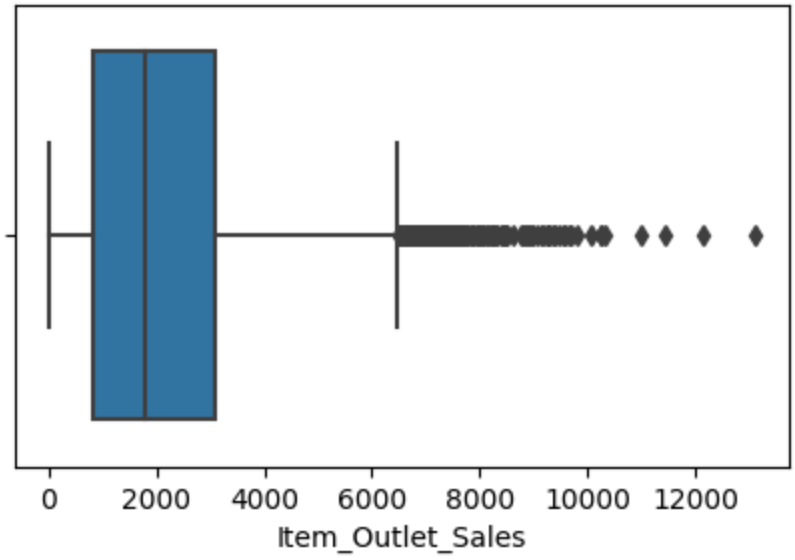
Out[1057]:



Lets calculate the quartiles for Outlet sales

In [ ]:
```
sns.boxplot(x=df_sales['Item_Outlet_Sales'])
```

Out[1058]:
```
<Axes: xlabel='Item_Outlet_Sales'>
```



In [ ]:
```
df_sales['Item_Outlet_Sales'].describe()
```

Out[1059]:
```
count     8523.000000
mean      2181.288914
std       1706.499616
min         33.290000
25%        834.247400
50%       1794.331000
75%       3101.296400
max      13086.964800
Name: Item_Outlet_Sales, dtype: float64
```

In [ ]:
```
print("1st quantile -q1",df_sales['Item_Outlet_Sales'].quantile(0.25))
print("2nd quantile -q2",df_sales['Item_Outlet_Sales'].quantile(0.50))
print("3rd quantile -q3",df_sales['Item_Outlet_Sales'].quantile(0.75))
```

```
1st quantile -q1 834.2474
2nd quantile -q2 1794.331
3rd quantile -q3 3101.2964
```

In [ ]:
```
q1 = df_sales['Item_Outlet_Sales'].quantile(0.25)
q3 = df_sales['Item_Outlet_Sales'].quantile(0.75)

print("Inter Quantile Range- iqr", q3-q1)
```

```
Inter Quantile Range- iqr 2267.049
```

In [ ]:
```
df_sales.quantile(0.25)
```

Out[1062]:
```
Item_Weight                   8.773750
Item_Visibility               0.026989
Item_MRP                     93.800000
Outlet_Establishment_Year  1987.000000
Item_Outlet_Sales           834.247400
Name: 0.25, dtype: float64
```

1. Outliers are defined as per the standards. Data points above q3 +1.5* iqr and Data points below q1-1.5* iqr are considered as outliers.
2. Quantile function works for numerical variables only.

In [ ]:
```
df_sales['Item_Fat_Content'].value_counts()
```

Out[1063]:
```
Low Fat    5089
Regular    2889
LF          316
reg         117
low fat     112
Name: Item_Fat_Content, dtype: int64
```

In [ ]:
```
df_sales['Item_Fat_Content'] = df_sales['Item_Fat_Content'].replace('LF','Low
```

In [ ]:
```
df_sales['Item_Fat_Content'].value_counts()
```

Out[1065]:
```
Low Fat    5517
Regular    3006
Name: Item_Fat_Content, dtype: int64
```

Missing Values - How to Treat and Deal

In [ ]:
```
df_sales.isnull().sum()
```

Out[1066]:
```
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       2050
Outlet_Type                   0
Item_Outlet_Sales             0
dtype: int64
```

In [ ]:
```
(df_sales.isnull().sum() / df_sales.shape[0]  * 100 )
```

Out[1067]:
```
Item_Identifier            0.000000
Item_Weight               17.165317
Item_Fat_Content           0.000000
Item_Visibility            0.000000
Item_Type                  0.000000
Item_MRP                   0.000000
Outlet_Identifier          0.000000
Outlet_Establishment_Year  0.000000
Outlet_Size               28.276428
Outlet_Location_Type      24.052564
Outlet_Type                0.000000
Item_Outlet_Sales          0.000000
dtype: float64
```

In [ ]:
```
df_sales.isnull().sum().sort_values(ascending=False)
```

Out[1068]:
```
Outlet_Size                2410
Outlet_Location_Type       2050
Item_Weight                1463
Item_Identifier               0
Item_Fat_Content              0
Item_Visibility               0
Item_Type                     0
Item_MRP                      0
Outlet_Identifier             0
Outlet_Establishment_Year     0
Outlet_Type                   0
Item_Outlet_Sales             0
dtype: int64
```

Variables that have missing values are of 2 Types - Numerical and Categorical

In [ ]:
```
(df_sales.isnull().sum() / df_sales.shape[0]  * 100 ).sort_values(ascending=Fa
```

Out[1069]:
```
Outlet_Size                 28.276428
Outlet_Location_Type        24.052564
Item_Weight                 17.165317
Item_Identifier              0.000000
Item_Fat_Content             0.000000
Item_Visibility              0.000000
Item_Type                    0.000000
Item_MRP                     0.000000
Outlet_Identifier            0.000000
Outlet_Establishment_Year    0.000000
Outlet_Type                  0.000000
Item_Outlet_Sales            0.000000
dtype: float64
```

Filter for values where it is not Zero

In [ ]:
```
df_sales.isnull().sum()[df_sales.isnull().sum()>0]
```

Out[1070]:
```
Item_Weight             1463
Outlet_Size             2410
Outlet_Location_Type    2050
dtype: int64
```

In [ ]:
```
df_sales.head()
```

Out[1071]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Ic
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.3	
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6	
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.1	
4	NCD19	8.93	Low Fat	0.000000	Household	53.9	

Missing Value Treatment for Numerical Variable

1. Numerical - Item Weight
2. Categorical - Outlet Size, Outlet_Location_Size

Lets say we want to impute the missing value for Item Weight

Case 1 :

In [ ]:
```
df_sales.groupby("Item_Fat_Content")['Item_Weight'].describe()
```

Out[1072]:

Item_Fat_Content	count	mean	std	min	25%	50%	75%	max
Low Fat	4566.0	12.937387	4.653787	4.590	8.775	12.65	17.1	21.35
Regular	2494.0	12.711654	4.621851	4.555	8.770	12.50	16.7	21.20

Case 2:

In [ ]:
```
df_sales.groupby("Item_Type")['Item_Weight'].describe()
```

Out[1073]:

Item_Type	count	mean	std	min	25%	50%	75%	max
Baking Goods	536.0	12.277108	4.773622	4.880	8.23500	11.650	15.75	20.85
Breads	204.0	11.346936	4.440540	4.635	7.12875	10.600	14.85	20.85
Breakfast	89.0	12.768202	5.038131	6.425	8.06000	10.695	17.25	21.10
Canned	539.0	12.305705	4.586564	4.615	8.11500	12.150	16.25	21.35
Dairy	566.0	13.426069	4.686532	4.805	9.27750	13.350	17.60	20.70
Frozen Foods	718.0	12.867061	4.507383	4.555	8.93500	12.850	17.00	20.85
Fruits and Vegetables	1019.0	13.224769	4.575275	5.460	9.19500	13.100	17.10	21.35
Hard Drinks	183.0	11.400328	4.239144	4.610	8.26000	10.100	14.85	19.70
Health and Hygiene	430.0	13.142314	4.512313	5.175	9.69500	12.150	17.60	21.25
Household	759.0	13.384736	4.998845	5.030	8.69500	13.150	18.35	21.25
Meat	337.0	12.817344	4.670812	5.150	9.30000	12.350	17.20	21.25
Others	137.0	13.853285	4.225534	5.500	10.65000	14.500	17.75	20.50
Seafood	51.0	12.552843	5.473830	5.365	7.42000	11.650	17.75	20.75
Snack Foods	988.0	12.987880	4.531256	5.095	9.19500	13.150	16.85	21.25
Soft Drinks	374.0	11.847460	4.403699	4.590	8.26000	11.800	15.35	20.75
Starchy Foods	130.0	13.690731	4.010061	6.695	11.50000	13.175	16.70	21.20

Case 3:

In [ ]:
```
df_sales.groupby("Item_Identifier")['Item_Weight'].describe()
```

Out[1074]:

Item_Identifier	count	mean	std	min	25%	50%	75%	max
DRA12	6.0	11.600	0.000000e+00	11.600	11.600	11.600	11.600	11.600
DRA24	5.0	19.350	0.000000e+00	19.350	19.350	19.350	19.350	19.350
DRA59	6.0	8.270	0.000000e+00	8.270	8.270	8.270	8.270	8.270
DRB01	2.0	7.390	0.000000e+00	7.390	7.390	7.390	7.390	7.390
DRB13	5.0	6.115	0.000000e+00	6.115	6.115	6.115	6.115	6.115
...	...	...	...	...	...	...	...	...
NCZ30	6.0	6.590	9.729507e-16	6.590	6.590	6.590	6.590	6.590
NCZ41	5.0	19.850	0.000000e+00	19.850	19.850	19.850	19.850	19.850
NCZ42	5.0	10.500	0.000000e+00	10.500	10.500	10.500	10.500	10.500
NCZ53	4.0	9.600	0.000000e+00	9.600	9.600	9.600	9.600	9.600
NCZ54	5.0	14.650	0.000000e+00	14.650	14.650	14.650	14.650	14.650

1559 rows × 8 columns

In [ ]:
```
df_sales[['Item_Identifier','Item_Weight','Item_Type']][df_sales.Item_Identifi
```

Out[1075]:

	Item_Identifier	Item_Weight	Item_Type
0	FDA15	9.3	Dairy
831	FDA15	9.3	Dairy
2599	FDA15	9.3	Dairy
2643	FDA15	9.3	Dairy
4874	FDA15	9.3	Dairy
5413	FDA15	9.3	Dairy
6696	FDA15	NaN	Dairy
7543	FDA15	9.3	Dairy

In [ ]:
```
df_sales[['Item_Identifier','Item_Weight','Item_Type']][df_sales.Item_Identifi
```

Out[1076]:

	Item_Identifier	Item_Weight	Item_Type
1208	NCZ05	8.485	Health and Hygiene
3244	NCZ05	8.485	Health and Hygiene
7873	NCZ05	NaN	Health and Hygiene

In [ ]:
```
df_sales[['Item_Identifier','Item_Weight','Item_Type']][df_sales.Item_Identifi
```

Out[1077]:

	Item_Identifier	Item_Weight	Item_Type
3	FDX07	19.2	Fruits and Vegetables
1491	FDX07	19.2	Fruits and Vegetables
2459	FDX07	19.2	Fruits and Vegetables
3089	FDX07	19.2	Fruits and Vegetables
5906	FDX07	19.2	Fruits and Vegetables
7735	FDX07	NaN	Fruits and Vegetables

In [ ]:
```
df_sales[['Item_Identifier','Item_Weight','Item_Type']][df_sales.Item_Identifi
```

Out[1078]:

	Item_Identifier	Item_Weight	Item_Type
1148	DRA24	19.35	Soft Drinks
2879	DRA24	NaN	Soft Drinks
4130	DRA24	19.35	Soft Drinks
4416	DRA24	NaN	Soft Drinks
4900	DRA24	19.35	Soft Drinks
6863	DRA24	19.35	Soft Drinks
8195	DRA24	19.35	Soft Drinks

In [ ]:
```
df_sales.groupby('Item_Identifier')['Item_Weight'].median()
```

Out[1079]:
```
Item_Identifier
DRA12    11.600
DRA24    19.350
DRA59     8.270
DRB01     7.390
DRB13     6.115
          ...
NCZ30     6.590
NCZ41    19.850
NCZ42    10.500
NCZ53     9.600
NCZ54    14.650
Name: Item_Weight, Length: 1559, dtype: float64
```

Case 4:

In [ ]:
```
df_sales['Item_Weight'].describe()
```

Out[1080]:
```
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
```

1. The weights have to been the same in each of the FDA15 PACKAGE. Hence, the closest possible value will be the value for each Item_Identifier.

Lets now impute Missing value for Item Weight

In [ ]:
```
weight = df_sales.groupby('Item_Identifier')['Item_Weight'].median().to_dict()
weight
```

Out[1081]:
```
{'DRA12': 11.6,
 'DRA24': 19.35,
 'DRA59': 8.27,
 'DRB01': 7.39,
 'DRB13': 6.115,
 'DRB24': 8.785,
 'DRB25': 12.3,
 'DRB48': 16.75,
 'DRC01': 5.92,
 'DRC12': 17.85,
 'DRC13': 8.26,
 'DRC24': 17.85,
 'DRC25': 5.73,
 'DRC27': 13.8,
 'DRC36': 13.0,
 'DRC49': 8.67,
 'DRD01': 12.1,
 'DRD12': 6.96,
 'DRD13': 15.0,
```

In [ ]:
```
df_sales['Item_Weight'] = df_sales.groupby('Item_Identifier')['Item_Weight'].f
```

In [ ]:
```
df_sales.isnull().sum().sort_values(ascending=False)
```

Out[1083]:
```
Outlet_Size                 2410
Outlet_Location_Type        2050
Item_Weight                 1463
Item_Identifier                0
Item_Fat_Content               0
Item_Visibility                0
Item_Type                      0
Item_MRP                       0
Outlet_Identifier              0
Outlet_Establishment_Year      0
Outlet_Type                    0
Item_Outlet_Sales              0
dtype: int64
```

In [ ]:
```
df_sales[df_sales['Item_Weight'].isnull()]
```

Out[1084]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outle
7	FDP10	NaN	Low Fat	0.127470	Snack Foods	107.8	
18	DRI11	NaN	Low Fat	0.034238	Hard Drinks	113.3	
21	FDW12	NaN	Regular	0.035400	Baking Goods	144.5	
23	FDC37	NaN	Low Fat	0.057557	Baking Goods	107.7	
29	FDC14	NaN	Regular	0.072222	Canned	43.6	
...	...	...	...	...	...	...	
8485	DRK37	NaN	Low Fat	0.043792	Soft Drinks	189.1	
8487	DRG13	NaN	Low Fat	0.037006	Soft Drinks	164.8	
8488	NCN14	NaN	Low Fat	0.091473	Others	184.7	
8490	FDU44	NaN	Regular	0.102296	Fruits and Vegetables	162.4	
8504	NCN18	NaN	Low Fat	0.124111	Household	111.8	

1463 rows × 12 columns

In [ ]:
```
# df_sales['Item_Weight']= df_sales.groupby('Item_Identifier')['Item_Weight'].
```

In [ ]:
```
df_sales['Item_Weight']= df_sales.groupby('Item_Identifier')['Item_Weight'].tr
```

In [ ]:
```
df_sales.isnull().sum().sort_values(ascending=False)
```

Out[1087]:
```
Outlet_Size                2410
Outlet_Location_Type       2050
Item_Weight                   4
Item_Identifier               0
Item_Fat_Content              0
Item_Visibility               0
Item_Type                     0
Item_MRP                      0
Outlet_Identifier             0
Outlet_Establishment_Year     0
Outlet_Type                   0
Item_Outlet_Sales             0
dtype: int64
```

In [ ]:
```
df_sales.loc[df_sales['Item_Weight'].isnull()]
```

Out[1088]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outle
927	FDN52	NaN	Regular	0.130933	Frozen Foods	86.9	
1922	FDK57	NaN	Low Fat	0.079904	Snack Foods	120.0	
4187	FDE52	NaN	Regular	0.029742	Dairy	89.0	
5022	FDQ60	NaN	Regular	0.191501	Baking Goods	121.2	

1. The values are null as these are the unique Item_Identifier where we do not have entries. For every Item_Identifier, there were multiple entries. However, Item_Identifier has only one entry with null value for Item_Weight.
2. We need to check Item_Identifier with Item_Fat_Content and look for pattern.

In [ ]:
```
# condition Item_Type=Frozen Food. Item_weight.median()
# we will apply filter on the data

df_sales.loc[df_sales['Item_Type']=="Frozen Foods",]
```

Out[1089]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outle
8	FDH17	16.200	Regular	0.016687	Frozen Foods	97.0	
9	FDU28	19.200	Regular	0.094450	Frozen Foods	187.8	
24	FDR28	13.850	Regular	0.025896	Frozen Foods	165.0	
51	FDM40	10.195	Low Fat	0.159804	Frozen Foods	141.5	
63	FDY40	15.500	Regular	0.150286	Frozen Foods	51.1	
...	...	...	...	...	...	...	
8477	FDC05	13.100	Regular	0.099343	Frozen Foods	198.2	
8482	FDZ28	20.000	Regular	0.051702	Frozen Foods	125.9	
8507	FDN28	5.880	Regular	0.030242	Frozen Foods	101.8	
8511	FDF05	17.500	Low Fat	0.026980	Frozen Foods	262.6	
8517	FDF53	20.750	Regular	0.083607	Frozen Foods	178.8	

856 rows × 12 columns

In [ ]:
```
#It will return the list of all iterm weight
df_sales.loc[df_sales['Item_Type']=="Frozen Foods","Item_Weight"]
```

Out[1090]:
```
8       16.200
9       19.200
24      13.850
51      10.195
63      15.500
         ...
8477    13.100
8482    20.000
8507     5.880
8511    17.500
8517    20.750
Name: Item_Weight, Length: 856, dtype: float64
```

In [ ]:
```
# .loc[ condition , Name of column]. apply the function
df_sales.loc[df_sales['Item_Type']=="Frozen Foods","Item_Weight"].median()
```

Out[1091]:
```
12.85
```

In [ ]:
```
# Take the Index number where it is missing and replace that column with media
df_sales.loc[927,'Item_Weight']=12.85
```

Lets impute other missing values

In [ ]:
```

df_sales.loc[df_sales['Item_Type']=="Snack Foods","Item_Weight"].median()
```

Out[1093]:
```
13.15
```

In [ ]:
```
df_sales.loc[1922,'Item_Weight']=13.15
```

In [ ]:
```
df_sales.loc[df_sales['Item_Type']=="Dairy","Item_Weight"].median()
```

Out[1095]:
```
13.35
```

In [ ]:
```
df_sales.loc[4187,'Item_Weight']=13.35
```

In [ ]:
```
df_sales.loc[df_sales['Item_Type']=="Baking Goods","Item_Weight"].median()
```

Out[1097]:
```
11.65
```

In [ ]:
```
df_sales.loc[5022,'Item_Weight']=11.65
```

In [ ]:
```
df_sales.isnull().sum()
```

Out[1099]:
```
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         2050
Outlet_Type                     0
Item_Outlet_Sales               0
dtype: int64
```

Imputing Missing Values for Categorical Variable

Outlet Size

In [ ]:
```
df_sales['Outlet_Size'].isnull().sum()
```

Out[1100]:
```
2410
```

Case 1

In [ ]:
```
df_sales.groupby("Item_Fat_Content")['Outlet_Size'].describe()
```

Out[1101]:

Item_Fat_Content	count	unique	top	freq
Low Fat	3955	3	Medium	1799
Regular	2158	3	Medium	994

Case 2

In [ ]:
```
df_sales.groupby("Outlet_Type")['Outlet_Size'].describe()
```

Out[1102]:

Outlet_Type	count	unique	top	freq
Grocery Store	528	1	Small	528
Supermarket Type1	3722	3	Small	1860
Supermarket Type2	928	1	Medium	928
Supermarket Type3	935	1	Medium	935

Case 3

In [ ]:
```
df_sales.groupby("Outlet_Location_Type")['Outlet_Size'].describe()
```

Out[1103]:

Outlet_Location_Type	count	unique	top	freq
-	0	0	NaN	NaN
--	0	0	NaN	NaN
?	0	0	NaN	NaN
NAN	0	0	NaN	NaN
Tier 2	2793	1	Medium	2793
Tier 3	932	1	High	932
Tier1	2388	1	Small	2388
na	0	0	NaN	NaN

Case 4

In [ ]:
```
df_sales.groupby(["Outlet_Type","Outlet_Location_Type"])['Outlet_Size'].descri
```

Out[1104]:

Outlet_Type	Outlet_Location_Type	count	unique	top	freq
Grocery Store	-	0	0	NaN	NaN
	--	0	0	NaN	NaN
	?	0	0	NaN	NaN
	NAN	0	0	NaN	NaN
	Tier1	528	1	Small	528
	na	0	0	NaN	NaN
Supermarket Type1	-	0	0	NaN	NaN
	--	0	0	NaN	NaN
	?	0	0	NaN	NaN
	NAN	0	0	NaN	NaN
	Tier 2	930	1	Medium	930
	Tier 3	932	1	High	932
	Tier1	1860	1	Small	1860
	na	0	0	NaN	NaN
Supermarket Type2	Tier 2	928	1	Medium	928
Supermarket Type3	Tier 2	935	1	Medium	935

OR

In [ ]:
```
df_sales.groupby(["Outlet_Size","Outlet_Location_Type"])['Outlet_Type'].descri
```

Out[1105]:

		count	unique	top	freq
Outlet_Size	**Outlet_Location_Type**				
High	**Tier 3**	932	1	Supermarket Type1	932
Medium	**Tier 2**	2793	3	Supermarket Type3	935
Small	**Tier1**	2388	2	Supermarket Type1	1860

Lets take the Case 2 and replace the value with the mode for each category

In [ ]:
```
df_sales['Outlet_Size']= df_sales.groupby('Outlet_Type')['Outlet_Size'].apply(
    lambda x : x.fillna(x.mode()[0]))
```

In [ ]:
```
df_sales.isnull().sum()
```

Out[1107]:
```
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       2050
Outlet_Type                   0
Item_Outlet_Sales             0
dtype: int64
```

Imputation Of Missing Values Categorical Variable- Outlet_Location_Type

Outlet_Location_Type

1. There are some data points that are not missing values but are incorrectly written ex: Nan, ? , -, etc. This also needs to be treated.
2. Lets make it null value and then treat them the way we treat null values.

In [ ]:
```
df_sales.groupby('Outlet_Location_Type').count()
```

Out[1108]:

Outlet_Location_Type	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type
-	67	67	67	67	67
--	109	109	109	109	109
?	120	120	120	120	120
NAN	16	16	16	16	16
Tier 2	2793	2793	2793	2793	2793
Tier 3	932	932	932	932	932
Tier1	2388	2388	2388	2388	2388
na	48	48	48	48	48

In [ ]:
```
df_sales['Outlet_Location_Type'].isnull().sum()
```

Out[1109]:
```
2050
```

In [ ]:
```
df_sales['Outlet_Location_Type'].value_counts()
```

Out[1110]:
```
Tier 2    2793
Tier1     2388
Tier 3     932
?          120
  --       109
  -         67
na          48
NAN         16
Name: Outlet_Location_Type, dtype: int64
```

In [ ]:
```
# Total garbase characters
print('Total Garbage' ,120+109+67+48+16)
print('Missing values',2050)
print('Garbage + null', 2050+360)
```

```
Total Garbage 360
Missing values 2050
Garbage + null 2410
```

The characters that need rectification are

1. -
2. --
3. ?
4. NAN
5. na

Lets replace all of these values with null or blank values

In [ ]:
```
df_sales['Outlet_Location_Type'] = df_sales['Outlet_Location_Type'].replace('
```

In [ ]:
```
df_sales['Outlet_Location_Type'].isnull().sum()
```

Out[1113]:
```
2410
```

In [ ]:

In [ ]:

Case 1

In [ ]:
```
df_sales.groupby("Outlet_Size")['Outlet_Location_Type'].describe()
```

Out[1114]:

Outlet_Size	count	unique	top	freq
High	932	1	Tier 3	932
Medium	2793	1	Tier 2	2793
Small	2388	1	Tier1	2388

Case 2

In [ ]:
```
df_sales.groupby("Outlet_Type")['Outlet_Location_Type'].describe()
```

Out[1115]:

Outlet_Type	count	unique	top	freq
Grocery Store	528	1	Tier1	528
Supermarket Type1	3722	3	Tier1	1860
Supermarket Type2	928	1	Tier 2	928
Supermarket Type3	935	1	Tier 2	935

Case 3

In [ ]:
```
df_sales.groupby(["Outlet_Size","Outlet_Type"])['Outlet_Location_Type'].descri
```

Out[1116]:

Outlet_Size	Outlet_Type	count	unique	top	freq
High	**Supermarket Type1**	932	1	Tier 3	932
	Supermarket Type1	930	1	Tier 2	930
Medium	**Supermarket Type2**	928	1	Tier 2	928
	Supermarket Type3	935	1	Tier 2	935
Small	**Grocery Store**	528	1	Tier1	528
	Supermarket Type1	1860	1	Tier1	1860

We find better pattern identification and more granular with case 3. Lets use mode to replace the missing values.

In [ ]:
```
df_sales['Outlet_Location_Type']= df_sales.groupby(["Outlet_Size","Outlet_Type
    lambda x : x.fillna(x.mode()[0]))
```

In [ ]:
```
df_sales.isnull().sum()
```

Out[1118]:
```
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 [ ]:
```

```

In [ ]:
```

```

Outlier Treatment

###Ourlier Treatment

In [ ]:
```
df_sales.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                8523 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                8523 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 [ ]:
```
df_sales.drop('Outlet_Establishment_Year',axis = 1, inplace=True)
```

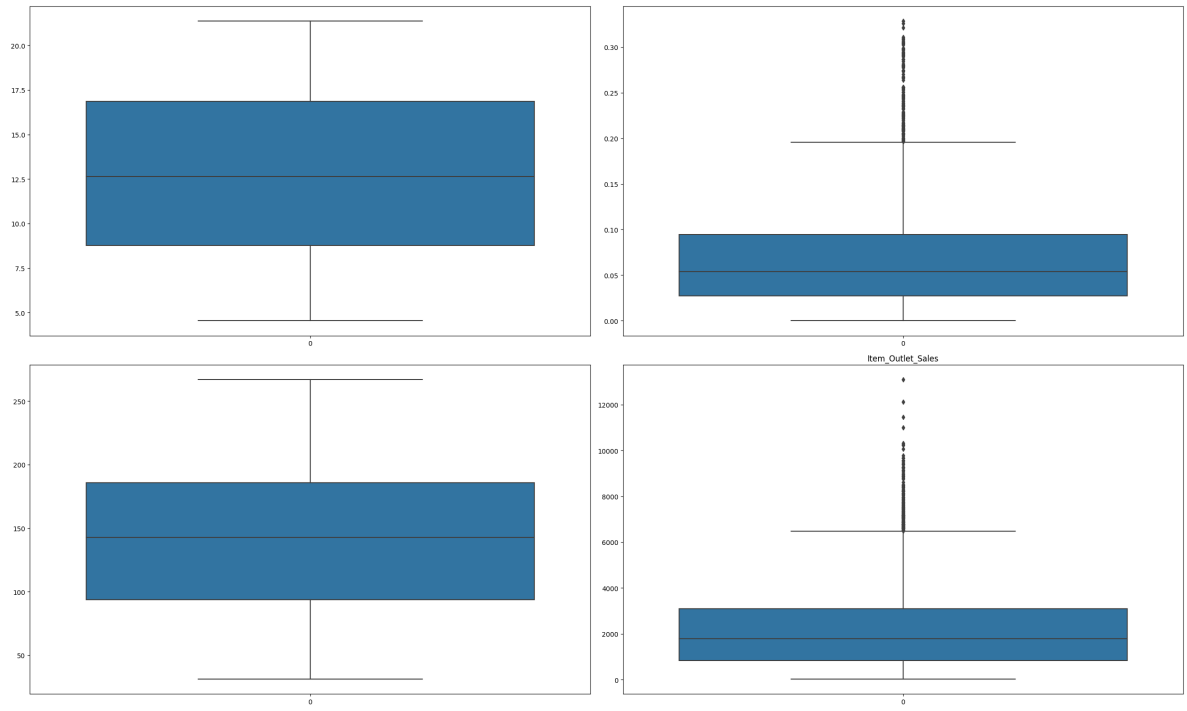
In [ ]:
```
num_cols = df_sales.select_dtypes(include=np.number)
num_cols.columns
```

Out[1121]:
```
Index(['Item_Weight', 'Item_Visibility', 'Item_MRP', 'Item_Outlet_Sales'], d
type='object')
```

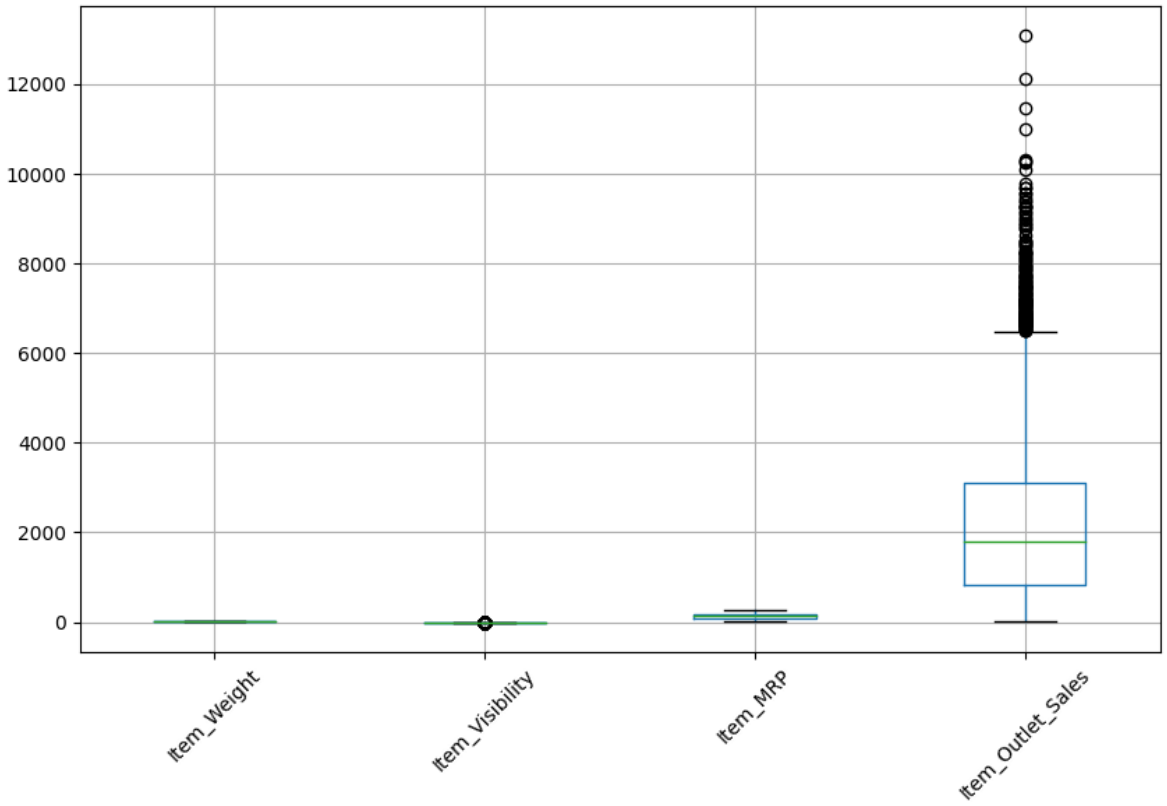
In [ ]:
```
# cols_outlier = ['Item_Weight', 'Item_Visibility', 'Item_MRP', 'Item_Outlet_S
```

In [ ]:
```
# df_sales_new['Item_Type_PCNT'] = df_sales_new['Item_Type_PCNT'].astype('obje
# df_sales_new['Outlet_Id_Trg']  = df_sales_new['Outlet_Id_Trg'].astype('objec
# df_sales_new['Item_Type_PCNT'] = df_sales_new['Item_Type_PCNT'].astype('obje
# df_sales_new['Outlet_Size']    = df_sales_new['Outlet_Size'].astype('object'
```

In [ ]:
```
fig, ax= plt.subplots(2,2, figsize=(25,15))

for i, subplot in zip(num_cols, ax.flatten()):
    sns.boxplot(df_sales[i], ax=subplot)
    plt.title(i)

plt.tight_layout()
plt.show()
```



In [ ]:
```
plt.rcParams['figure.figsize']=[10,6]
df_sales.boxplot()
plt.xticks(rotation=45)
plt.show()
```



You can remove Outliers by using 2 techniques

1. Box plot - IQR Technique. value > q3 + 1.5 *iqr or value < q1 -1.5 * iqr
2. Z Score technique. Z score > 3 or Z Score < -3

In [ ]:
```
# Calculate the quantiles and Interquantile range

q1 = df_sales.quantile(0.25)
q3 = df_sales.quantile(0.75)
iqr = q3-q1

upper_limit = q3 + 1.5 * iqr
lower_limit = q1 - 1.5 * iqr
```

In [ ]:
```
df_sales > upper_limit
```

Out[1127]:

	Item_Fat_Content	Item_Identifier	Item_MRP	Item_Outlet_Sales	Item_Type	Item_Visibility
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
8518	False	False	False	False	False	False
8519	False	False	False	False	False	False
8520	False	False	False	False	False	False
8521	False	False	False	False	False	False
8522	False	False	False	False	False	False

8523 rows × 11 columns

In [ ]:
```
df_sales< lower_limit
```

Out[1128]:

	Item_Fat_Content	Item_Identifier	Item_MRP	Item_Outlet_Sales	Item_Type	Item_Visibility
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
8518	False	False	False	False	False	False
8519	False	False	False	False	False	False
8520	False	False	False	False	False	False
8521	False	False	False	False	False	False
8522	False	False	False	False	False	False

8523 rows × 11 columns

In [ ]:
```
(df_sales< lower_limit ) | (df_sales > upper_limit )
```

Out[1129]:

	Item_Fat_Content	Item_Identifier	Item_MRP	Item_Outlet_Sales	Item_Type	Item_Visibility
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	False	False	False	False	False
...	...	...	...	...	...	...
8518	False	False	False	False	False	False
8519	False	False	False	False	False	False
8520	False	False	False	False	False	False
8521	False	False	False	False	False	False
8522	False	False	False	False	False	False

8523 rows × 11 columns

In [ ]:
```
df_sales[~(df_sales< lower_limit ) | (df_sales> upper_limit )]
```

Out[1130]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outle
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8	
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.3	
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6	
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.1	
4	NCD19	8.930	Low Fat	0.000000	Household	53.9	
...	...	...	...	...	...	...	
8518	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5	
8519	FDS36	8.380	Regular	0.046982	Baking Goods	108.2	
8520	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1	
8521	FDN46	7.210	Regular	0.145221	Snack Foods	103.1	
8522	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.5	

8523 rows × 11 columns

1. Return the output row by row. Instead of Column by column.
2. Check rowwise, if the condition is met and return that data. In the second row, where the condition is met, retun the data. If the condition is not met then do not return the data.

3. any means- if the condition is met anywhere, return this data.
4. ~: Not represents returning all the values which are false.
5. ex: not(false) = True. , Not(True) = False.
6. .any(axis=1) : represents checking of the condition row wise. And whenever this condition is met, it returns the data.
7. Return the data which is the layer.
8. if any() is not used then it will check one column with entire dataset. We need to avoid that.

In [ ]:
```
df_sales[~((df_sales< (lower_limit)) | (df_sales> (upper_limit))).any(axis=1)]
```

Out[1131]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outle
0	FDA15	9.300	Low Fat	0.016047	Dairy	249.8	
1	DRC01	5.920	Regular	0.019278	Soft Drinks	48.3	
2	FDN15	17.500	Low Fat	0.016760	Meat	141.6	
3	FDX07	19.200	Regular	0.000000	Fruits and Vegetables	182.1	
4	NCD19	8.930	Low Fat	0.000000	Household	53.9	
...	...	...	...	...	...	...	
8518	FDF22	6.865	Low Fat	0.056783	Snack Foods	214.5	
8519	FDS36	8.380	Regular	0.046982	Baking Goods	108.2	
8520	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1	
8521	FDN46	7.210	Regular	0.145221	Snack Foods	103.1	
8522	DRG01	14.800	Low Fat	0.044878	Soft Drinks	75.5	

8193 rows × 11 columns

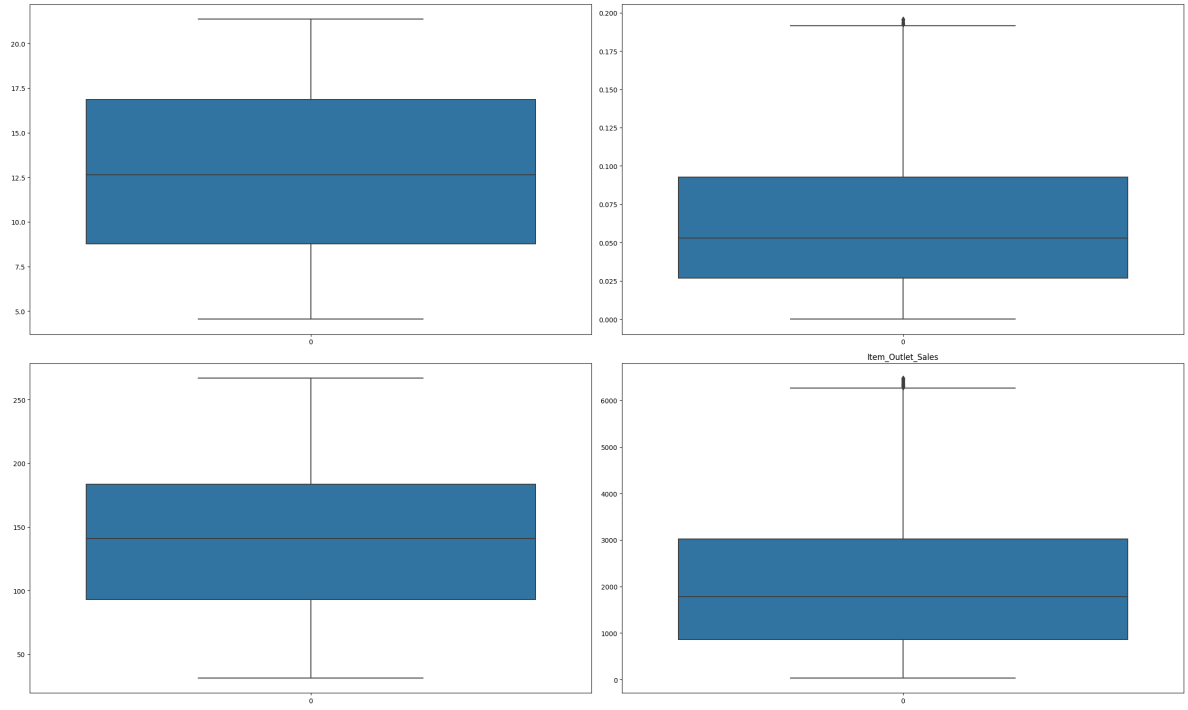
In [ ]:
```
# df_sales = df_sales[~((df_sales< (lower_limit)) | (df_sales> (upper_limit)))]
df_rem_out = df_sales[~((df_sales< (lower_limit)) | (df_sales> (upper_limit)))]
```

In [ ]:
```
print("Shape before removing outlier", df_sales.shape)
print("Shape after removing outlier", df_rem_out.shape)
```

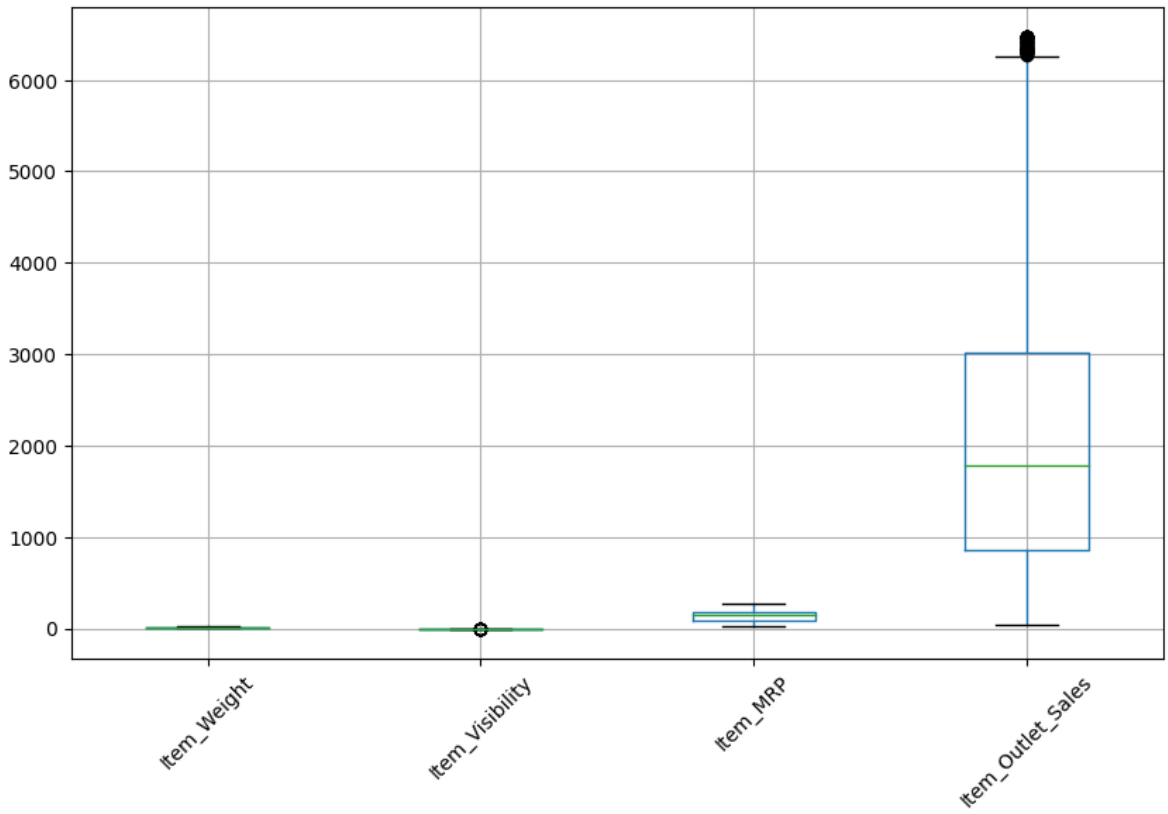
```
Shape before removing outlier (8523, 11)
Shape after removing outlier (8193, 11)
```

In [ ]:

In [ ]:
```
fig, ax= plt.subplots(2,2, figsize=(25,15))

for i, subplot in zip(num_cols, ax.flatten()):
    sns.boxplot(df_rem_out[i], ax=subplot)
    plt.title(i)

plt.tight_layout()
plt.show()
```



In [ ]:
```
plt.rcParams['figure.figsize']=[10,6]
df_rem_out.boxplot()
plt.xticks(rotation=45)
plt.show()
```



Lets Treat the outliers and prepare the final Dataset without Outliers

In [ ]:
```
# we will make a copy of data with outliers as this will be required for showc
df_sales_copy = df_sales.copy()
```

In [ ]:
```
df_sales_copy.shape
```

Out[1137]: (8523, 11)

In [ ]:
```
df_sales = df_sales[~((df_sales< (lower_limit)) | (df_sales> (upper_limit))).a
```

In [ ]:
```
df_sales.shape
```

Out[1139]: (8193, 11)

In [ ]:

SCALING

1. Standard Scaler
2. Min-Max Scaler

In [ ]:
```
df_sales.head()
```

Out[1140]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Ic
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.3	
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6	
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.1	
4	NCD19	8.93	Low Fat	0.000000	Household	53.9	

In [ ]:
```
df_num = df_sales.select_dtypes(include= np.number)

df_num.head()
```

Out[1141]:

	Item_Weight	Item_Visibility	Item_MRP	Item_Outlet_Sales
0	9.30	0.016047	249.8	3735.1380
1	5.92	0.019278	48.3	443.4228
2	17.50	0.016760	141.6	2097.2700
3	19.20	0.000000	182.1	732.3800
4	8.93	0.000000	53.9	994.7052

In [ ]:
```
df_sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8193 entries, 0 to 8522
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Item_Identifier       8193 non-null   object
 1   Item_Weight           8193 non-null   float64
 2   Item_Fat_Content      8193 non-null   object
 3   Item_Visibility       8193 non-null   float64
 4   Item_Type             8193 non-null   object
 5   Item_MRP              8193 non-null   float64
 6   Outlet_Identifier     8193 non-null   object
 7   Outlet_Size           8193 non-null   object
 8   Outlet_Location_Type  8193 non-null   object
 9   Outlet_Type           8193 non-null   object
 10  Item_Outlet_Sales     8193 non-null   float64
dtypes: float64(4), object(7)
memory usage: 768.1+ KB
```

Normal Distribution

1. The normal distributions are brought on a common scale of Mean =0 and SD=1 which is called as Standard Normal Distribution.
2. It is probability distribution which has its total area = 1.
3. Also called as Bell Curve.
4. 68% of data lies within 1 SD. 95% of data lies within 2 SD. 99.7% of data lies within 3 SD.

In [ ]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

In [ ]:
```
sc=StandardScaler()
mm = MinMaxScaler()
```

In [ ]:
```
sc.fit_transform(df_num)
```

Out[1145]:
```
array([[-0.77063068, -1.01927283,  1.79939636,  1.09603737],
       [-1.49810135, -0.94946348, -1.47623319, -1.09638334],
       [ 0.994239  , -1.00387215,  0.04047271,  0.00514842],
       ...,
       [-0.49083427, -0.60574318, -0.87800407, -0.59705775],
       [-1.22045722,  1.77173463, -0.585392  , -0.16247598],
       [ 0.41312337, -0.39633102, -1.03406384, -0.88175316]])
```

Standard Scaler returns the value in array. This array is converted into DataFrame

In [ ]:
```
df_scaled_sc = pd.DataFrame(sc.fit_transform(df_num), columns = df_num.columns
df_scaled_sc.head()
```

Out[1146]:

	Item_Weight	Item_Visibility	Item_MRP	Item_Outlet_Sales
0	-0.770631	-1.019273	1.799396	1.096037
1	-1.498101	-0.949463	-1.476233	-1.096383
2	0.994239	-1.003872	0.040473	0.005148
3	1.360127	-1.366002	0.698850	-0.903926
4	-0.850265	-1.366002	-1.385198	-0.729206

In [ ]:
```
df_scaled_sc.describe()
```

Out[1147]:

	Item_Weight	Item_Visibility	Item_MRP	Item_Outlet_Sales
count	8.193000e+03	8.193000e+03	8.193000e+03	8.193000e+03
mean	4.002657e-16	-2.638761e-16	3.414549e-16	-2.059733e-17
std	1.000061e+00	1.000061e+00	1.000061e+00	1.000061e+00
min	-1.791888e+00	-1.366002e+00	-1.752589e+00	-1.369549e+00
25%	-8.814731e-01	-7.865495e-01	-7.495799e-01	-8.205570e-01
50%	-4.961685e-02	-2.267000e-01	2.746772e-02	-2.037169e-01
75%	8.543408e-01	6.387288e-01	7.248598e-01	6.197712e-01
max	1.822867e+00	2.862882e+00	2.077378e+00	2.923055e+00

Min Max Scaler - Normalization

In [ ]:
```
# Min Max Scaler
mm.fit_transform(df_num)
```

Out[1148]:
```
array([[0.28252456, 0.08199064, 0.92741935, 0.57438017],
       [0.08127419, 0.09849839, 0.0721562 , 0.06363636],
       [0.77076511, 0.08563243, 0.46816638, 0.32024793],
       ...,
       [0.35992855, 0.17977758, 0.22835314, 0.17995868],
       [0.15808276, 0.74197737, 0.30475382, 0.28119835],
       [0.61000298, 0.22929707, 0.18760611, 0.11363636]])
```

In [ ]:
```
df_scaled_mm = pd.DataFrame(mm.fit_transform(df_num),columns=df_num.columns)
df_scaled_mm.head()
```

Out[1149]:

	Item_Weight	Item_Visibility	Item_MRP	Item_Outlet_Sales
0	0.282525	0.081991	0.927419	0.574380
1	0.081274	0.098498	0.072156	0.063636
2	0.770765	0.085632	0.468166	0.320248
3	0.871986	0.000000	0.640068	0.108471
4	0.260494	0.000000	0.095925	0.149174

In [ ]:
```
df_scaled_mm.describe()
```

Out[1150]:

	Item_Weight	Item_Visibility	Item_MRP	Item_Outlet_Sales
count	8193.000000	8193.000000	8193.000000	8193.000000
mean	0.495715	0.323017	0.457599	0.319049
std	0.276661	0.236483	0.261115	0.232973
min	0.000000	0.000000	0.000000	0.000000
25%	0.251861	0.137022	0.261885	0.127893
50%	0.481989	0.269410	0.464771	0.271591
75%	0.732063	0.474057	0.646859	0.463430
max	1.000000	1.000000	1.000000	1.000000

Type *Markdown* and LaTeX: α^2

Type *Markdown* and LaTeX: α^2

Second Way Of Treating The Outliers

Using Z Score Technique

1. Convert the data into Standard Scale
2. Find the outliers using Z score values > 3 and < -3

We will work with copy data of df_sales as this is data with outliers The second method works on concept of ZScore to remove outliers

In [ ]:
```
pd.get_dummies(df_sales_copy).dtypes
```

Out[1151]:
```
Item_Weight                      float64
Item_Visibility                  float64
Item_MRP                         float64
Item_Outlet_Sales                float64
Item_Identifier_DRA12              uint8
                                   ...
Outlet_Location_Type_Tier1         uint8
Outlet_Type_Grocery Store          uint8
Outlet_Type_Supermarket Type1      uint8
Outlet_Type_Supermarket Type2      uint8
Outlet_Type_Supermarket Type3      uint8
Length: 1601, dtype: object
```

Do not convert into number before scaling

In [ ]:
```
df_sales_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Item_Identifier       8523 non-null   object 
 1   Item_Weight           8523 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_Size           8523 non-null   object 
 8   Outlet_Location_Type  8523 non-null   object 
 9   Outlet_Type           8523 non-null   object 
 10  Item_Outlet_Sales     8523 non-null   float64
dtypes: float64(4), object(7)
memory usage: 732.6+ KB
```

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

In [ ]:
```
num_copy = df_sales_copy.select_dtypes(np.number)
```

Bring the data on Zscale before removing the outliers

In [ ]:
```
df_scaled_copy = pd.DataFrame(sc.fit_transform(num_copy), columns = num_copy.c
df_scaled_copy.head()
```

Out[1155]:

	Item_Weight	Item_Visibility	Item_MRP	Item_Outlet_Sales
0	-0.769763	-0.970732	1.747685	0.910601
1	-1.497465	-0.908111	-1.489032	-1.018440
2	0.995668	-0.956917	0.009657	-0.049238
3	1.361672	-1.281758	0.660212	-0.849103
4	-0.849422	-1.281758	-1.399078	-0.695373

In [ ]:
```
# df_sales[~((df_sales< (lower_limit)) | (df_sales> (upper_limit))).any(axis=1
```

1. In the above code, we removed the outliers using lower limit and upper limit
2. Instead of Upper limit and Lower limit, lets replace with standard deviation of > 3 and < -3 to detect the outliers.

In [ ]:
```
df_rem_zscore = df_scaled_copy[~((df_scaled_copy< (-3)) | (df_scaled_copy> (3)
df_rem_zscore.head()
```

Out[1157]:

	Item_Weight	Item_Visibility	Item_MRP	Item_Outlet_Sales
0	-0.769763	-0.970732	1.747685	0.910601
1	-1.497465	-0.908111	-1.489032	-1.018440
2	0.995668	-0.956917	0.009657	-0.049238
3	1.361672	-1.281758	0.660212	-0.849103
4	-0.849422	-1.281758	-1.399078	-0.695373

In [ ]:
```
print("Shape before removing outlier", df_sales_copy.shape)
print("Shape after removing outlier", df_rem_zscore.shape)
```

```
Shape before removing outlier (8523, 11)
Shape after removing outlier (8338, 4)
```

In [ ]:

In [ ]:

How to Deal with Categorical Variable

1. One Hot Encoding
2. Label Encoding
3. Frequency Encoding
4. Target Encoding.

In [ ]:
```
df_sales.select_dtypes(include='object').columns
```

Out[1159]:
```
Index(['Item_Identifier', 'Item_Fat_Content', 'Item_Type', 'Outlet_Identifie
r',
       'Outlet_Size', 'Outlet_Location_Type', 'Outlet_Type'],
      dtype='object')
```

In [ ]:
```
df_sales['Item_Type'].value_counts()
```

Out[1160]:
```
Fruits and Vegetables    1175
Snack Foods              1154
Household                 877
Frozen Foods              830
Dairy                     649
Canned                    626
Baking Goods              622
Health and Hygiene        507
Soft Drinks               428
Meat                      409
Breads                    242
Hard Drinks               207
Others                    167
Starchy Foods             139
Breakfast                 101
Seafood                    60
Name: Item_Type, dtype: int64
```

One Hot Encoding

In [ ]:
```
df_cat = df_sales.select_dtypes(include='object')
df_cat.head()
```

Out[1161]:

	Item_Identifier	Item_Fat_Content	Item_Type	Outlet_Identifier	Outlet_Size	Outlet_Location_Ty
0	FDA15	Low Fat	Dairy	OUT049	Medium	Tier
1	DRC01	Regular	Soft Drinks	OUT018	Medium	Tier
2	FDN15	Low Fat	Meat	OUT049	Medium	Tier
3	FDX07	Regular	Fruits and Vegetables	OUT010	Small	Tie
4	NCD19	Low Fat	Household	OUT013	High	Tier

In [ ]:
```
pd.get_dummies(df_cat)
```

Out[1162]:

	Item_Identifier_DRA12	Item_Identifier_DRA24	Item_Identifier_DRA59	Item_Identifier_DRB01
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
...	...	...	...	...
8518	0	0	0	0
8519	0	0	0	0
8520	0	0	0	0
8521	0	0	0	0
8522	0	0	0	0

8193 rows × 1597 columns

In [ ]:
```
df_cat.shape
```

Out[1163]:
```
(8193, 7)
```

In [ ]:
```
pd.get_dummies(df_cat).shape
```

Out[1164]:
```
(8193, 1597)
```

The number of columns will increase from 7 to 1600.

Label Encoding

In [ ]:
```
df_sales.columns
```

Out[1165]:
```
Index(['Item_Identifier', 'Item_Weight', 'Item_Fat_Content', 'Item_Visibilit
y',
       'Item_Type', 'Item_MRP', 'Outlet_Identifier', 'Outlet_Size',
       'Outlet_Location_Type', 'Outlet_Type', 'Item_Outlet_Sales'],
      dtype='object')
```

In [ ]:
```
df_cat.columns
```

Out[1166]:
```
Index(['Item_Identifier', 'Item_Fat_Content', 'Item_Type', 'Outlet_Identifie
r',
       'Outlet_Size', 'Outlet_Location_Type', 'Outlet_Type'],
      dtype='object')
```

In [ ]:
```
df_sales['Outlet_Size'].value_counts()
```

Out[1167]:
```
Small     4600
Medium    2676
High       917
Name: Outlet_Size, dtype: int64
```

using replace function

In [ ]:
```
df_sales.replace({'Small':1,'Medium':2,"High":3},inplace=True)
```

In [ ]:
```
df_sales['Outlet_Size'].value_counts()
```

Out[1169]:
```
1    4600
2    2676
3     917
Name: Outlet_Size, dtype: int64
```

In [ ]:
```
df_sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8193 entries, 0 to 8522
Data columns (total 11 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Item_Identifier      8193 non-null   object
 1   Item_Weight          8193 non-null   float64
 2   Item_Fat_Content     8193 non-null   object
 3   Item_Visibility      8193 non-null   float64
 4   Item_Type            8193 non-null   object
 5   Item_MRP             8193 non-null   float64
 6   Outlet_Identifier    8193 non-null   object
 7   Outlet_Size          8193 non-null   int64
 8   Outlet_Location_Type 8193 non-null   object
 9   Outlet_Type          8193 non-null   object
 10  Item_Outlet_Sales    8193 non-null   float64
dtypes: float64(4), int64(1), object(6)
memory usage: 768.1+ KB
```

Frequency Encoding

Item Type

In [ ]:
```
df_sales['Item_Type'].value_counts(normalize=True)
```

Out[1171]:
```
Fruits and Vegetables    0.143415
Snack Foods              0.140852
Household                0.107043
Frozen Foods             0.101306
Dairy                    0.079214
Canned                   0.076407
Baking Goods             0.075918
Health and Hygiene       0.061882
Soft Drinks              0.052240
Meat                     0.049921
Breads                   0.029537
Hard Drinks              0.025265
Others                   0.020383
Starchy Foods            0.016966
Breakfast                0.012328
Seafood                  0.007323
Name: Item_Type, dtype: float64
```

1. For each variable level, we get the frequency of occurance. We can replace this values for the specific category of that variable.
2. Frequency encoding will not work if category have same frequency as the machine will not understand.
3. So if you have various unique categories then it works better.

In [ ]:
```
df_sales['Item_Type'].value_counts(normalize=True).to_dict()
```

Out[1172]:
```
{'Fruits and Vegetables': 0.1434151104601489,
 'Snack Foods': 0.14085194678383986,
 'Household': 0.107042597339192,
 'Frozen Foods': 0.10130599292078604,
 'Dairy': 0.07921396313926522,
 'Canned': 0.07640668863664103,
 'Baking Goods': 0.07591846698401074,
 'Health and Hygiene': 0.061882094470889784,
 'Soft Drinks': 0.05223971683144148,
 'Meat': 0.04992066398144758,
 'Breads': 0.029537409984132797,
 'Hard Drinks': 0.025265470523617724,
 'Others': 0.02038325399731478,
 'Starchy Foods': 0.01696570242890272,
 'Breakfast': 0.012327596728914928,
 'Seafood': 0.007323324789454412}
```

In [ ]:
```
norm_items = df_sales['Item_Type'].value_counts(normalize=True).to_dict()
```

In [ ]:
```
df_sales['Item_Type_PCNT']= df_sales['Item_Type'].map(norm_items)
```

In [ ]:
```
df_sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8193 entries, 0 to 8522
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Item_Identifier       8193 non-null   object 
 1   Item_Weight           8193 non-null   float64
 2   Item_Fat_Content      8193 non-null   object 
 3   Item_Visibility       8193 non-null   float64
 4   Item_Type             8193 non-null   object 
 5   Item_MRP              8193 non-null   float64
 6   Outlet_Identifier     8193 non-null   object 
 7   Outlet_Size           8193 non-null   int64  
 8   Outlet_Location_Type  8193 non-null   object 
 9   Outlet_Type           8193 non-null   object 
 10  Item_Outlet_Sales     8193 non-null   float64
 11  Item_Type_PCNT        8193 non-null   float64
dtypes: float64(5), int64(1), object(6)
memory usage: 832.1+ KB
```

In [ ]:
```
df_sales.head()
```

Out[1176]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Ic
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.3	
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6	
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.1	
4	NCD19	8.93	Low Fat	0.000000	Household	53.9	

Outlet Indentifier

In [ ]:
```
df_sales['Outlet_Identifier'].value_counts()
```

Out[1177]:
```
OUT018    925
OUT045    920
OUT046    920
OUT049    919
OUT013    917
OUT035    914
OUT017    907
OUT027    832
OUT010    488
OUT019    451
Name: Outlet_Identifier, dtype: int64
```

1. **If the frequency of occurance is the same then frequency encoding can not be applied. During Model building, machine are unable to recognize the patterns with same frequency**

2. **If the frequency match, we can try out different encoding technique - Target Encoding**

Target Encoding

Outlet Identifier

Check how good the correlation is between Outlet Identifier and Sales

ANNOVA

In [ ]:
```
df_sales.groupby('Outlet_Identifier')['Item_Outlet_Sales'].median()
```

Out[1178]:
```
Outlet_Identifier
OUT010     249.6750
OUT013    2014.7108
OUT017    1970.7680
OUT018    1651.1840
OUT019     256.9988
OUT027    3098.6332
OUT035    2090.9449
OUT045    1818.9656
OUT046    1924.8278
OUT049    1957.4520
Name: Item_Outlet_Sales, dtype: float64
```

Outlet 10 and 19 are not performing compared to other outlets. Outlet 27 is the best performing outlet

In [ ]:
```
df_sales.groupby('Outlet_Identifier')['Item_Outlet_Sales'].median().to_dict()
```

Out[1179]:
```
{'OUT010': 249.675,
 'OUT013': 2014.7108,
 'OUT017': 1970.768,
 'OUT018': 1651.184,
 'OUT019': 256.9988,
 'OUT027': 3098.6332,
 'OUT035': 2090.9449,
 'OUT045': 1818.9656,
 'OUT046': 1924.8278,
 'OUT049': 1957.452}
```

In [ ]:
```
outlet_id = df_sales.groupby('Outlet_Identifier')['Item_Outlet_Sales'].median(
```

In [ ]:
```
df_sales['Outlet_Id_Trg'] = df_sales['Outlet_Identifier'].map(outlet_id)
```

In [ ]:
```
df_sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8193 entries, 0 to 8522
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Item_Identifier       8193 non-null   object 
 1   Item_Weight           8193 non-null   float64
 2   Item_Fat_Content      8193 non-null   object 
 3   Item_Visibility       8193 non-null   float64
 4   Item_Type             8193 non-null   object 
 5   Item_MRP              8193 non-null   float64
 6   Outlet_Identifier     8193 non-null   object 
 7   Outlet_Size           8193 non-null   int64  
 8   Outlet_Location_Type  8193 non-null   object 
 9   Outlet_Type           8193 non-null   object 
 10  Item_Outlet_Sales     8193 non-null   float64
 11  Item_Type_PCNT        8193 non-null   float64
 12  Outlet_Id_Trg         8193 non-null   float64
dtypes: float64(6), int64(1), object(6)
memory usage: 896.1+ KB
```

In [ ]:
```
df_sales.head(2)
```

Out[1183]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Id
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.3	

Check how good the correlation is between Outlet Id Trg and Sales

In [ ]:
```
df_sales[['Outlet_Id_Trg','Item_Outlet_Sales']].corr()
```

Out[1184]:

	Outlet_Id_Trg	Item_Outlet_Sales
Outlet_Id_Trg	1.000000	0.462543
Item_Outlet_Sales	0.462543	1.000000

There is high correlation between Outlet Id and Sales. Which is good case for Target Encoding

Item Type

In [ ]:
```
df_sales['Item_Type'].nunique()
```

Out[1185]: 16

1. We cannot apply One Hot encoding here as the number of columns will increase. Leading to curse of dimensionality
2. Lets Try Frequency Encoding works but may not be good decision to apply Target encoding.
3. There can be a better way to group some of the similar appearing items into one group.

Grouping

In [ ]:
```
df_sales['Item_Type'].unique()
```

Out[1186]:
```
array(['Dairy', 'Soft Drinks', 'Meat', 'Fruits and Vegetables',
       'Household', 'Baking Goods', 'Snack Foods', 'Frozen Foods',
       'Breakfast', 'Health and Hygiene', 'Hard Drinks', 'Canned',
       'Breads', 'Starchy Foods', 'Others', 'Seafood'], dtype=object)
```

In [ ]:
```
# You can create 4 groups of Veg, Nov Veg, drinks, others  / Perishable and No
```

In [ ]:
```
Veg=['Dairy','Fruits and Vegetables','Baking Goods','Snack Foods','Frozen Food
     'Canned','Breads','Starchy Foods']
Non_Veg = ['Meat','Seafood']
Drinks =['Soft Drinks','Hard Drinks']
others =['Health and Hygiene','Others']
```

In [ ]:
```
def food_type(x):
  if x in Veg:
    return('Veg')
  elif x in Non_Veg:
    return("Non_Veg")
  elif x in Drinks:
    return("Drinks")
  else:
    return("Others")
```

In [ ]:
```
df_sales['Item_Type'].apply(food_type)
```

Out[1190]:
```
0           Veg
1        Drinks
2       Non_Veg
3           Veg
4        Others
         ...
8518        Veg
8519        Veg
8520     Others
8521        Veg
8522     Drinks
Name: Item_Type, Length: 8193, dtype: object
```

In [ ]:
```
df_sales['Item_Type_GRP'] =df_sales['Item_Type'].apply(food_type)
```

In [ ]:
```
df_sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8193 entries, 0 to 8522
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Item_Identifier       8193 non-null   object 
 1   Item_Weight           8193 non-null   float64
 2   Item_Fat_Content      8193 non-null   object 
 3   Item_Visibility       8193 non-null   float64
 4   Item_Type             8193 non-null   object 
 5   Item_MRP              8193 non-null   float64
 6   Outlet_Identifier     8193 non-null   object 
 7   Outlet_Size           8193 non-null   int64  
 8   Outlet_Location_Type  8193 non-null   object 
 9   Outlet_Type           8193 non-null   object 
 10  Item_Outlet_Sales     8193 non-null   float64
 11  Item_Type_PCNT        8193 non-null   float64
 12  Outlet_Id_Trg         8193 non-null   float64
 13  Item_Type_GRP         8193 non-null   object 
dtypes: float64(6), int64(1), object(7)
memory usage: 960.1+ KB
```

In [ ]:
```
pd.get_dummies(df_sales['Item_Type_GRP'])
```

Out[1193]:

	Drinks	Non_Veg	Others	Veg
0	0	0	0	1
1	1	0	0	0
2	0	1	0	0
3	0	0	0	1
4	0	0	1	0
...	...	...	...	...
8518	0	0	0	1
8519	0	0	0	1
8520	0	0	1	0
8521	0	0	0	1
8522	1	0	0	0

8193 rows × 4 columns

In [ ]:
```
pd.get_dummies(df_sales['Item_Type_GRP'], drop_first=True)
```

Out[1194]:

	Non_Veg	Others	Veg
0	0	0	1
1	0	0	0
2	1	0	0
3	0	0	1
4	0	1	0
...	...	...	...
8518	0	0	1
8519	0	0	1
8520	0	1	0
8521	0	0	1
8522	0	0	0

8193 rows × 3 columns

In [ ]:
```
df_sales.head(2)
```

Out[1195]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Id
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8	(
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.3	(

Drop the variable which are Insignificant

In [ ]:
```
df_sales= df_sales.drop(['Outlet_Identifier'],axis=1)
```

In [ ]:
```
df_sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8193 entries, 0 to 8522
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Item_Identifier       8193 non-null   object
 1   Item_Weight           8193 non-null   float64
 2   Item_Fat_Content      8193 non-null   object
 3   Item_Visibility       8193 non-null   float64
 4   Item_Type             8193 non-null   object
 5   Item_MRP              8193 non-null   float64
 6   Outlet_Size           8193 non-null   int64
 7   Outlet_Location_Type  8193 non-null   object
 8   Outlet_Type           8193 non-null   object
 9   Item_Outlet_Sales     8193 non-null   float64
 10  Item_Type_PCNT        8193 non-null   float64
 11  Outlet_Id_Trg         8193 non-null   float64
 12  Item_Type_GRP         8193 non-null   object
dtypes: float64(6), int64(1), object(6)
memory usage: 896.1+ KB
```

In [ ]:
```
df_sales.head(2)
```

Out[1198]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_S
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.3	

In [ ]:
```
df_sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8193 entries, 0 to 8522
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Item_Identifier       8193 non-null   object
 1   Item_Weight           8193 non-null   float64
 2   Item_Fat_Content      8193 non-null   object
 3   Item_Visibility       8193 non-null   float64
 4   Item_Type             8193 non-null   object
 5   Item_MRP              8193 non-null   float64
 6   Outlet_Size           8193 non-null   int64
 7   Outlet_Location_Type  8193 non-null   object
 8   Outlet_Type           8193 non-null   object
 9   Item_Outlet_Sales     8193 non-null   float64
 10  Item_Type_PCNT        8193 non-null   float64
 11  Outlet_Id_Trg         8193 non-null   float64
 12  Item_Type_GRP         8193 non-null   object
dtypes: float64(6), int64(1), object(6)
memory usage: 896.1+ KB
```

Feature Engineering

Frequency Encoding

In [ ]:
```
df_sales['Item_Identifier'].nunique()
```

Out[1200]:
```
1559
```

In [ ]:
```
df_sales['Item_Identifier'][0][:2]
```

Out[1201]:
```
'FD'
```

In [ ]:
```
Food_ID = []

for i in df_sales.Item_Identifier:
  Food_ID.append(i[:2])
```

In [ ]:
```
pd.Series(Food_ID).sort_values(ascending=False).isnull().sum()
```

Out[1203]:
```
0
```

In [ ]:
```
pd.Series(Food_ID).sort_values(ascending=False).count()
```

Out[1204]:
```
8193
```

In [ ]:
```
pd.DataFrame(pd.Series(Food_ID))
```

Out[1205]:

	0
0	FD
1	DR
2	FD
3	FD
4	NC
...	...
8188	FD
8189	FD
8190	NC
8191	FD
8192	DR

8193 rows × 1 columns

In [ ]:
```
df_sales['Item_Identifier'][100]
```

Out[1206]:
```
'FDT28'
```

In [ ]:
```
df_sales['Item_Identifier'].tail(316)
```

Out[1207]:
```
8192    FDR22
8193    FDZ10
8195    DRA24
8196    FDV37
8197    FDO10
        ...
8518    FDF22
8519    FDS36
8520    NCJ29
8521    FDN46
8522    DRG01
Name: Item_Identifier, Length: 316, dtype: object
```

In [ ]:
```
df_sales['Item_Identifier'].shape
```

Out[1208]:
```
(8193,)
```

In [ ]:
```
pd.DataFrame(pd.Series(Food_ID))
```

Out[1209]:

	0
0	FD
1	DR
2	FD
3	FD
4	NC
...	...
8188	FD
8189	FD
8190	NC
8191	FD
8192	DR

8193 rows × 1 columns

In [ ]:
```
def food_id(x):

  return(x[:2])
```

In [ ]:
```
df_sales['Item_ID_Code'] =df_sales['Item_Identifier'].apply(food_id)
df_sales.head()
```

Out[1211]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_S
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.3	
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6	
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.1	
4	NCD19	8.93	Low Fat	0.000000	Household	53.9	

In [ ]:
```
df_sales['Item_Identifier'].value_counts()
```

Out[1212]:
```
FDW13    10
DRN47     9
NCJ30     9
FDX31     9
FDW49     9
         ..
FDC23     1
DRG25     1
FDP15     1
DRF48     1
FDK57     1
Name: Item_Identifier, Length: 1559, dtype: int64
```

In [ ]:
```
df_sales['Item_ID_Code'].value_counts()
```

Out[1213]:
```
FD    5870
NC    1551
DR     772
Name: Item_ID_Code, dtype: int64
```

In [ ]:
```
df_sales['Item_Identifier'].isnull().sum()
```

Out[1214]:
```
0
```

In [ ]:
```
df_sales['Item_ID_Code'].isnull().sum()
```

Out[1215]:
```
0
```

In [ ]:
```
df_sales.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8193 entries, 0 to 8522
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Item_Identifier       8193 non-null   object
 1   Item_Weight           8193 non-null   float64
 2   Item_Fat_Content      8193 non-null   object
 3   Item_Visibility       8193 non-null   float64
 4   Item_Type             8193 non-null   object
 5   Item_MRP              8193 non-null   float64
 6   Outlet_Size           8193 non-null   int64
 7   Outlet_Location_Type  8193 non-null   object
 8   Outlet_Type           8193 non-null   object
 9   Item_Outlet_Sales     8193 non-null   float64
 10  Item_Type_PCNT        8193 non-null   float64
 11  Outlet_Id_Trg         8193 non-null   float64
 12  Item_Type_GRP         8193 non-null   object
 13  Item_ID_Code          8193 non-null   object
dtypes: float64(6), int64(1), object(7)
memory usage: 1.2+ MB
```

There are 3 features which can be used for one hot encoding

In [ ]:
```
pd.get_dummies(df_sales['Item_ID_Code'])
```

Out[1217]:

	DR	FD	NC
0	0	1	0
1	1	0	0
2	0	1	0
3	0	1	0
4	0	0	1
...	...	...	...
8518	0	1	0
8519	0	1	0
8520	0	0	1
8521	0	1	0
8522	1	0	0

8193 rows × 3 columns

In [ ]:
```
pd.get_dummies(df_sales['Item_ID_Code'],drop_first=True)
```

Out[1218]:

	FD	NC
0	1	0
1	0	0
2	1	0
3	1	0
4	0	1
...	...	...
8518	1	0
8519	1	0
8520	0	1
8521	1	0
8522	0	0

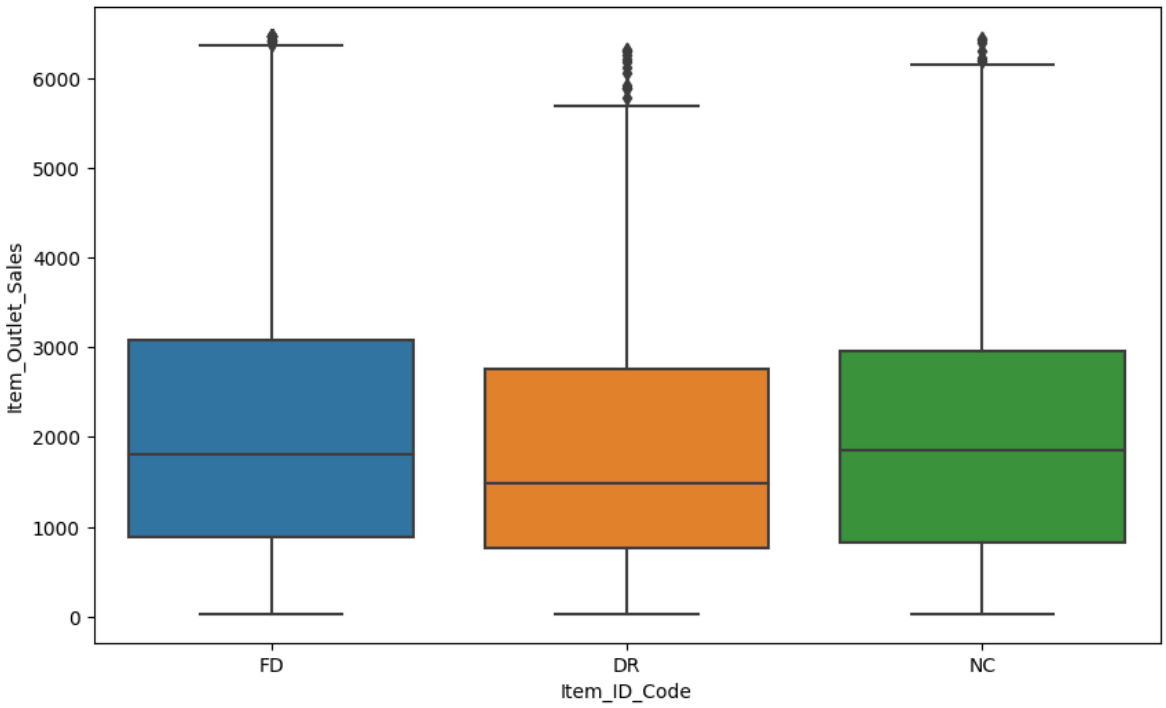
8193 rows × 2 columns

In [ ]:

In [ ]:

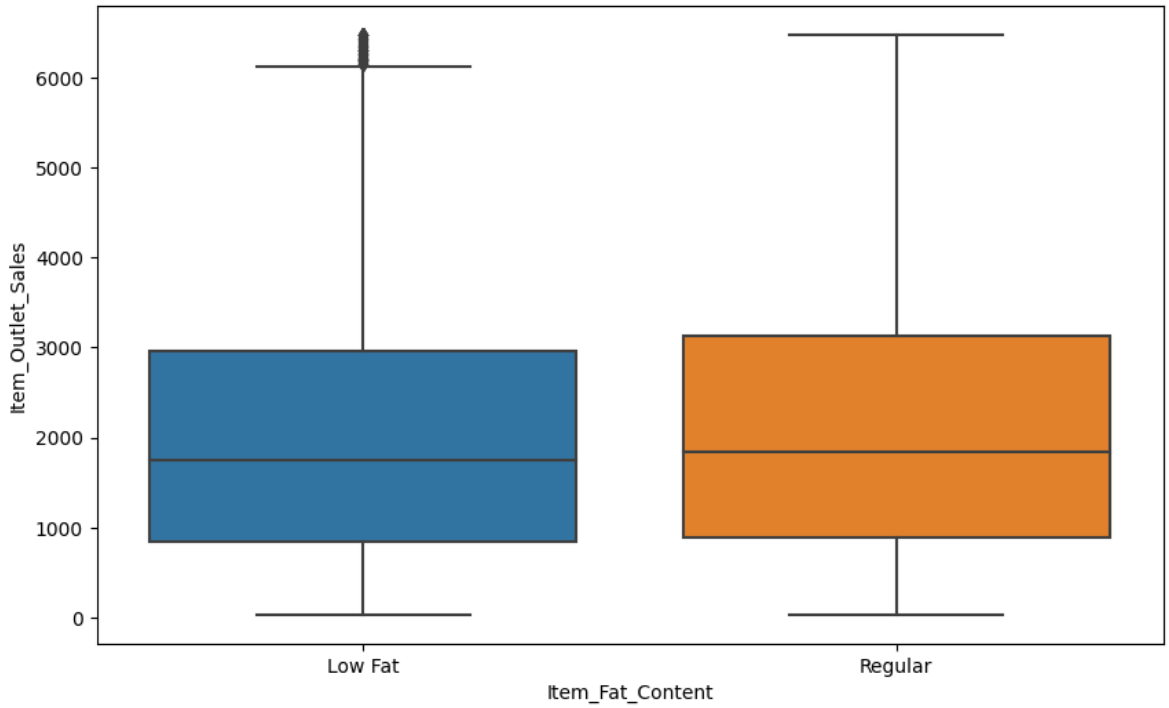
Box Plot Between Item Identifier and Sales

In [ ]:
```
sns.boxplot(x="Item_ID_Code", y="Item_Outlet_Sales", data=df_sales)
plt.show()
```



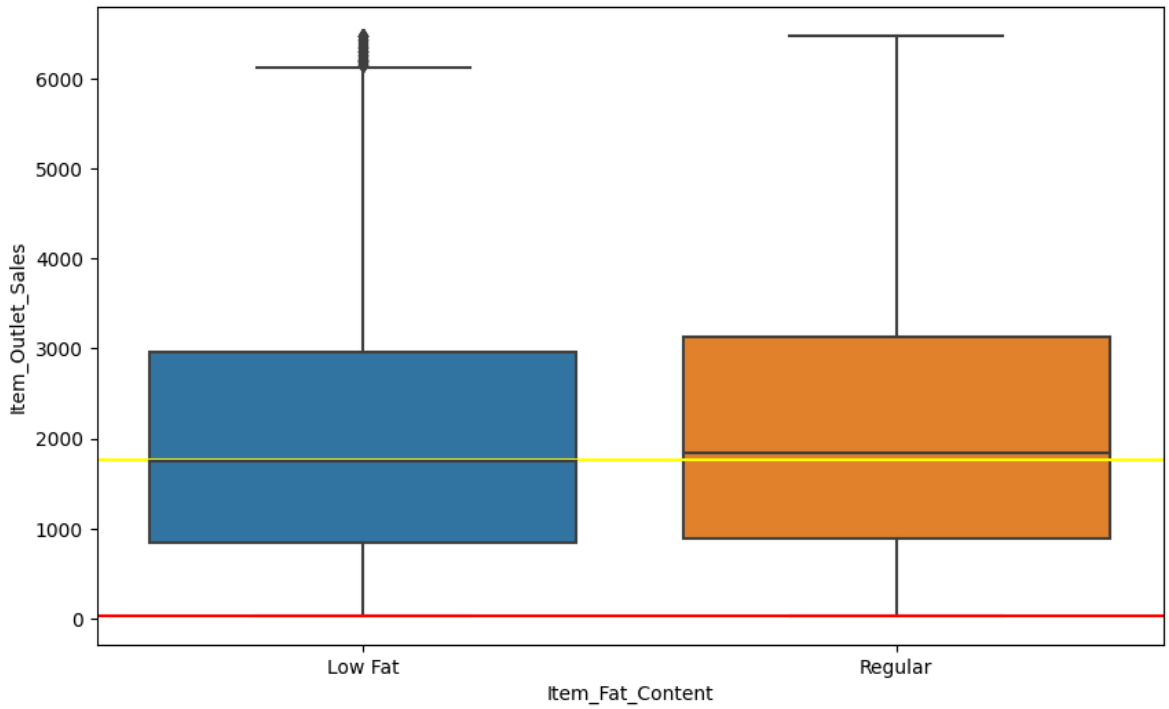
Food is more in demand compared with Non consumable and the drink

In [ ]:
```
sns.boxplot(x="Item_Fat_Content", y="Item_Outlet_Sales", data=df_sales)
plt.show()
```



Low Fat content are selling more than Regular fat content

In [ ]:
```
sns.boxplot(x="Item_Fat_Content", y="Item_Outlet_Sales", data=df_sales)
plt.axhline(1765.0358, color='yellow')
plt.axhline(33.2900, color='red')
plt.show()
```



In [ ]:
```
df_sales.groupby("Item_Fat_Content")['Item_Outlet_Sales'].describe()
```

Out[1222]:

Item_Fat_Content	count	mean	std	min	25%	50%	75%	
Low Fat	5309.0	2059.730242	1487.677235	33.2900	844.2344	1749.7224	2960.1468	64
Regular	2884.0	2144.415520	1525.350740	33.9558	885.5140	1843.6002	3131.9232	64

In [ ]:
```
df_sales.loc[df_sales['Item_ID_Code']=='NC']
```

Out[1223]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outle
4	NCD19	8.930	Low Fat	0.000000	Household	53.9	
16	NCB42	11.800	Low Fat	0.008596	Health and Hygiene	115.3	
22	NCB30	14.600	Low Fat	0.025698	Household	196.5	
25	NCD06	13.000	Low Fat	0.099887	Household	45.9	
31	NCS17	18.600	Low Fat	0.080829	Health and Hygiene	96.4	
...	...	...	...	...	...	...	...
8500	NCQ42	20.350	Low Fat	0.000000	Household	125.2	
8502	NCH43	8.420	Low Fat	0.070712	Household	216.4	
8504	NCN18	8.895	Low Fat	0.124111	Household	111.8	
8516	NCJ19	18.600	Low Fat	0.118661	Others	58.8	
8520	NCJ29	10.600	Low Fat	0.035186	Health and Hygiene	85.1	

1551 rows × 14 columns

In [ ]:
```
df_sales.loc[df_sales['Item_ID_Code']=='NC','Item_Fat_Content']
```

Out[1224]:
```
4       Low Fat
16      Low Fat
22      Low Fat
25      Low Fat
31      Low Fat
         ...
8500    Low Fat
8502    Low Fat
8504    Low Fat
8516    Low Fat
8520    Low Fat
Name: Item_Fat_Content, Length: 1551, dtype: object
```

Item Visibility -Replacing the 0's

Though the missing values are none, however there seems to be anamoly in the dataset.
Ex: There are 0's which does not seem to logical for visibility

In [ ]:
```
df_sales['Item_Visibility'].value_counts()
```

Out[1225]:
```
0.000000    512
0.076975      3
0.162462      2
0.156802      2
0.026950      2
           ...
0.128806      1
0.112162      1
0.083359      1
0.090473      1
0.044878      1
Name: Item_Visibility, Length: 7571, dtype: int64
```

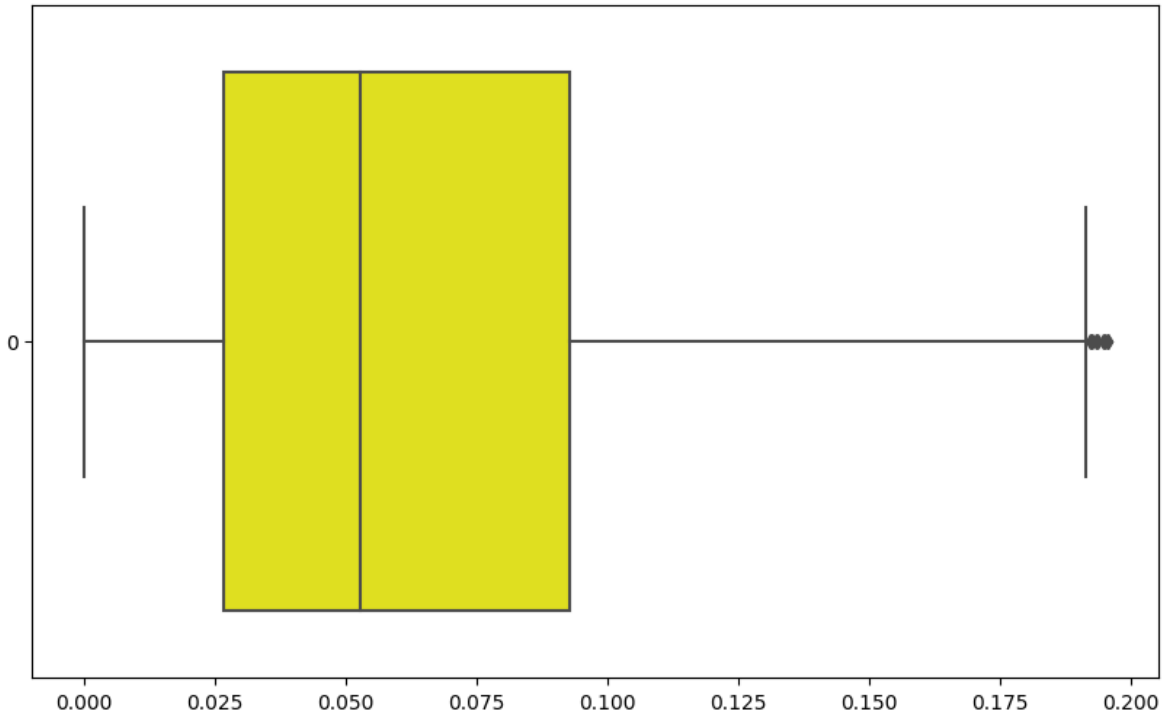
There are several zero's. It does not mean that there is no item placed.

In [ ]:
```
df_sales.groupby("Item_ID_Code")['Item_Visibility'].median()
```

Out[1226]:
```
Item_ID_Code
DR    0.048051
FD    0.055141
NC    0.044608
Name: Item_Visibility, dtype: float64
```

In [ ]:
```
sns.boxplot(df_sales['Item_Visibility'], orient='h',color='yellow')
```

Out[1227]:
```
<Axes: >
```



In [ ]:
```
df_sales.groupby("Item_ID_Code")['Item_Visibility'].apply(
    lambda x: x.replace(0,x.median()))
```

Out[1228]:
```
0       0.016047
1       0.019278
2       0.016760
3       0.055141
4       0.044608
          ...
8518    0.056783
8519    0.046982
8520    0.035186
8521    0.145221
8522    0.044878
Name: Item_Visibility, Length: 8193, dtype: float64
```

In [ ]:
```
df_sales['Item_Visibility'] = df_sales.groupby("Item_ID_Code")['Item_Visibilit
    lambda x: x.replace(0,x.median()))
```

In [ ]:
```
df_sales['Item_Visibility'].value_counts()
```

Out[1230]:
```
0.055141    355
0.044608     93
0.048051     65
0.076975      3
0.093914      2
           ...
0.128806      1
0.112162      1
0.083359      1
0.090473      1
0.044878      1
Name: Item_Visibility, Length: 7572, dtype: int64
```

In [ ]:
```
df_sales['Item_Visibility'].describe()
```

Out[1231]:
```
count    8193.000000
mean        0.066493
std         0.043478
min         0.003575
25%         0.032653
50%         0.055141
75%         0.092783
max         0.195721
Name: Item_Visibility, dtype: float64
```

In [ ]:
```
df_sales.shape
```

Out[1232]:
```
(8193, 14)
```

In [ ]:
```
df_sales.isnull().sum()
```

Out[1233]:
```
Item_Identifier         0
Item_Weight             0
Item_Fat_Content        0
Item_Visibility         0
Item_Type               0
Item_MRP                0
Outlet_Size             0
Outlet_Location_Type    0
Outlet_Type             0
Item_Outlet_Sales       0
Item_Type_PCNT          0
Outlet_Id_Trg           0
Item_Type_GRP           0
Item_ID_Code            0
dtype: int64
```

In [ ]:
```
df_sales.head(2)
```

Out[1234]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_S
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.3	

Type *Markdown* and LaTeX: α^2

So before we finalize the dataset. Lets remove the columns that are duplicated or not required.

In [ ]:
```
df_sales.shape
```

Out[1235]:
```
(8193, 14)
```

In [ ]:
```
df_sales['Outlet_Location_Type'].value_counts()
```

Out[1236]:
```
Tier1     4600
Tier 2    2676
Tier 3     917
Name: Outlet_Location_Type, dtype: int64
```

In [ ]:
```
df_sales.groupby('Outlet_Location_Type').count()
```

Out[1237]:

Outlet_Location_Type	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type
Tier 2	2676	2676	2676	2676	2676
Tier 3	917	917	917	917	917
Tier1	4600	4600	4600	4600	4600

In [ ]:
```
df_sales.isnull().sum()
```

Out[1238]:
```
Item_Identifier         0
Item_Weight             0
Item_Fat_Content        0
Item_Visibility         0
Item_Type               0
Item_MRP                0
Outlet_Size             0
Outlet_Location_Type    0
Outlet_Type             0
Item_Outlet_Sales       0
Item_Type_PCNT          0
Outlet_Id_Trg           0
Item_Type_GRP           0
Item_ID_Code            0
dtype: int64
```

Columns not required / Duplicated

1. Item_Identifier - we have replaced with Item_ID_Code
2. Item_Type - we have replaced with Item_Type_GRP

In [ ]:
```
df_sales_new = df_sales.drop(['Item_Identifier','Item_Type'],axis=1)
df_sales_new.shape
```

Out[1239]:
```
(8193, 12)
```

In [ ]:
```
df_sales_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8193 entries, 0 to 8522
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Item_Weight           8193 non-null   float64
 1   Item_Fat_Content      8193 non-null   object
 2   Item_Visibility       8193 non-null   float64
 3   Item_MRP              8193 non-null   float64
 4   Outlet_Size           8193 non-null   int64
 5   Outlet_Location_Type  8193 non-null   object
 6   Outlet_Type           8193 non-null   object
 7   Item_Outlet_Sales     8193 non-null   float64
 8   Item_Type_PCNT        8193 non-null   float64
 9   Outlet_Id_Trg         8193 non-null   float64
 10  Item_Type_GRP         8193 non-null   object
 11  Item_ID_Code          8193 non-null   object
dtypes: float64(6), int64(1), object(5)
memory usage: 1.1+ MB
```

Dummy Encoding all of them

In [ ]:
```
pd.get_dummies(df_sales_new) [:2]
```

Out[1241]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Size	Item_Outlet_Sales	Item_Type_PCNT	Outl
0	9.30	0.016047	249.8	2	3735.1380	0.079214	
1	5.92	0.019278	48.3	2	443.4228	0.052240	

2 rows × 23 columns

In [ ]:
```
pd.get_dummies(df_sales_new,drop_first=True)[:2]
```

Out[1242]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Size	Item_Outlet_Sales	Item_Type_PCNT	Outl
0	9.30	0.016047	249.8	2	3735.1380	0.079214	
1	5.92	0.019278	48.3	2	443.4228	0.052240	

1. **The columns -Item_Type_PCNT, Outlet_Id_Trg, Outlet_Size are actually categorical. Since they are encoded, it is displayed as int64**
2. **The numerical columns are -Item_Weight, Item_Visibility,Item_MRP and Item_Outlet_Sales**

In [ ]:
```
df_cat = df_sales_new.drop(['Item_Weight','Item_Visibility','Item_MRP'],axis=1
df_cat.columns
```

Out[1243]:
```
Index(['Item_Fat_Content', 'Outlet_Size', 'Outlet_Location_Type',
       'Outlet_Type', 'Item_Outlet_Sales', 'Item_Type_PCNT', 'Outlet_Id_Tr
g',
       'Item_Type_GRP', 'Item_ID_Code'],
      dtype='object')
```

1. We will dummy encode the categorical variable.
2. Scale the numerical variables.
3. Concantenate the scaled numerical and dummy encoded variables.

In [ ]:
```
pd.get_dummies(df_cat)[:2]
```

Out[1244]:

	Outlet_Size	Item_Outlet_Sales	Item_Type_PCNT	Outlet_Id_Trg	Item_Fat_Content_Low Fat	Item_F
0	2	3735.1380	0.079214	1957.452	1	
1	2	443.4228	0.052240	1651.184	0	

In [ ]:
```
pd.get_dummies(df_cat,drop_first=True)[:2]
```

Out[1245]:

	Outlet_Size	Item_Outlet_Sales	Item_Type_PCNT	Outlet_Id_Trg	Item_Fat_Content_Regular	Ou
0	2	3735.1380	0.079214	1957.452	0	
1	2	443.4228	0.052240	1651.184	1	

In [ ]:
```
df_dummy = pd.get_dummies(df_cat,drop_first=True)
df_dummy.head(2)
```

Out[1246]:

	Outlet_Size	Item_Outlet_Sales	Item_Type_PCNT	Outlet_Id_Trg	Item_Fat_Content_Regular	Ou
0	2	3735.1380	0.079214	1957.452	0	
1	2	443.4228	0.052240	1651.184	1	

In [ ]:
```
df_dummy.shape
```

Out[1247]:
```
(8193, 15)
```

In [ ]:
```
df_scaled_sc.head(2)
```

Out[1248]:

	Item_Weight	Item_Visibility	Item_MRP	Item_Outlet_Sales
0	-0.770631	-1.019273	1.799396	1.096037
1	-1.498101	-0.949463	-1.476233	-1.096383

In [ ]:
```
df_scaled_sc.shape
```

Out[1249]:
```
(8193, 4)
```

In [ ]:
```
df_dummy.reset_index(inplace=True,drop=True)
df_scaled_sc.reset_index(inplace=True,drop=True)
```

In [ ]:
```
df_sales_final = pd.concat([df_scaled_sc,df_dummy],axis=1)
```

In [ ]:
```
df_sales_final.reset_index(inplace=True,drop=True)
```

In [ ]:
```
df_sales_final.head()
```

Out[1253]:

	Item_Weight	Item_Visibility	Item_MRP	Item_Outlet_Sales	Outlet_Size	Item_Outlet_Sales	Iter
0	-0.770631	-1.019273	1.799396	1.096037	2	3735.1380	
1	-1.498101	-0.949463	-1.476233	-1.096383	2	443.4228	
2	0.994239	-1.003872	0.040473	0.005148	2	2097.2700	
3	1.360127	-1.366002	0.698850	-0.903926	1	732.3800	
4	-0.850265	-1.366002	-1.385198	-0.729206	3	994.7052	

In [ ]:
```
df_sales_final.shape
```

Out[1254]:
```
(8193, 19)
```

In [ ]:
```
df_sales_final.isnull().sum()
```

Out[1255]:
```
Item_Weight                      0
Item_Visibility                  0
Item_MRP                         0
Item_Outlet_Sales                0
Outlet_Size                      0
Item_Outlet_Sales                0
Item_Type_PCNT                   0
Outlet_Id_Trg                    0
Item_Fat_Content_Regular         0
Outlet_Location_Type_Tier 3      0
Outlet_Location_Type_Tier1       0
Outlet_Type_Supermarket Type1    0
Outlet_Type_Supermarket Type2    0
Outlet_Type_Supermarket Type3    0
Item_Type_GRP_Non_Veg            0
Item_Type_GRP_Others             0
Item_Type_GRP_Veg                0
Item_ID_Code_FD                  0
Item_ID_Code_NC                  0
dtype: int64
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```
# num_cols = df_sales_new.select_dtypes(include=np.number)
# num_cols.head(2)
```

In [ ]:
```
# num_cols.columns
```

In [ ]:
```
# cat_cols = df_sales_new.select_dtypes(exclude=np.number)
# cat_cols.head(2)
```

In [ ]:
```
# cat_cols.columns
```

In [ ]:

Modify the list of categorical columns

In [ ]:
```
# # cat_cols =['Outlet_Size','Item_Type_PCNT', 'Outlet_Id_Trg',
#             'Item_Fat_Content', 'Outlet_Location_Type', 'Outlet_Type',
#             'Item_Type_GRP', 'Item_ID_Code',]
```

In [ ]:
```
# num_cols =['Item_Weight', 'Item_Visibility', 'Item_MRP', 'Item_Outlet_Sales'
```

Take the numerical columns that are scaled

In [ ]:
```
# df_scaled_sc.columns
```

In [ ]:
```

```

Dummy encode the categorical variables

In [ ]:
```
# pd.get_dummies(cat_cols)
```

In [ ]:
```
# df_cat = pd.DataFrame(pd.get_dummies(cat_cols,drop_first=True),columns=cat_c
# df_cat.head(2)
```

In [ ]:
```

```