In [30]:
```
#!pip install ydata-profiling
```

In [31]:
```
import pandas as pd
```

In [32]:
```
df = pd.read_csv(r"C:\Users\HP\Downloads\9961_14084_bundle_archive\Train.csv"
```

In [33]:
```
df
```

Out[33]:

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

8523 rows × 12 columns

In [34]:
```
from ydata_profiling import ProfileReport
```

In [35]:
```
profile =ProfileReport(df,title="Pandas Profiling Report",explorative= True)
```

In [36]:
```
profile.to_notebook_iframe()
```

Summarize dataset: 100% 47/47 [00:03<00:00, 10.15it/s, Completed]

Generate report structure: 100% 1/1 [00:02<00:00, 2.67s/it]

Render HTML: 100% 1/1 [00:00<00:00, 1.34it/s]

ASCII

Value	Count	Frequency (%)
D	7254	17.0%
F	6449	15.1%
3	2285	5.4%
4	2280	5.4%
2	2279	5.3%
1	2261	5.3%
5	2249	5.3%
0	2141	5.0%
N	1920	4.5%
C	1903	4.5%
Other values (26)	11594	27.2%

Item_Weight

Real number (ℝ)

MISSING

Distinct	415
Distinct (%)	5.9%

In [37]:
```
# Example for numerical variables
```

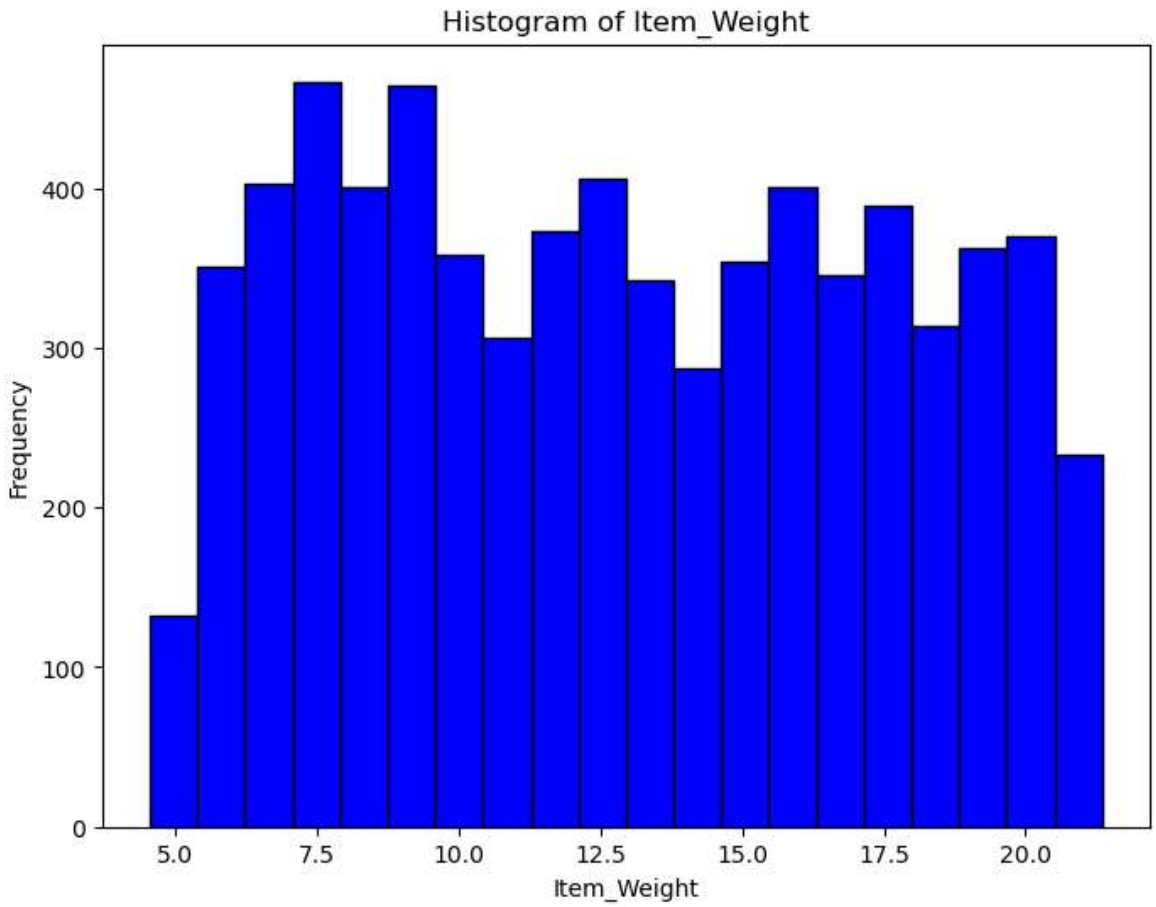
In [38]:
```
numerical_summary
```

Out[38]:

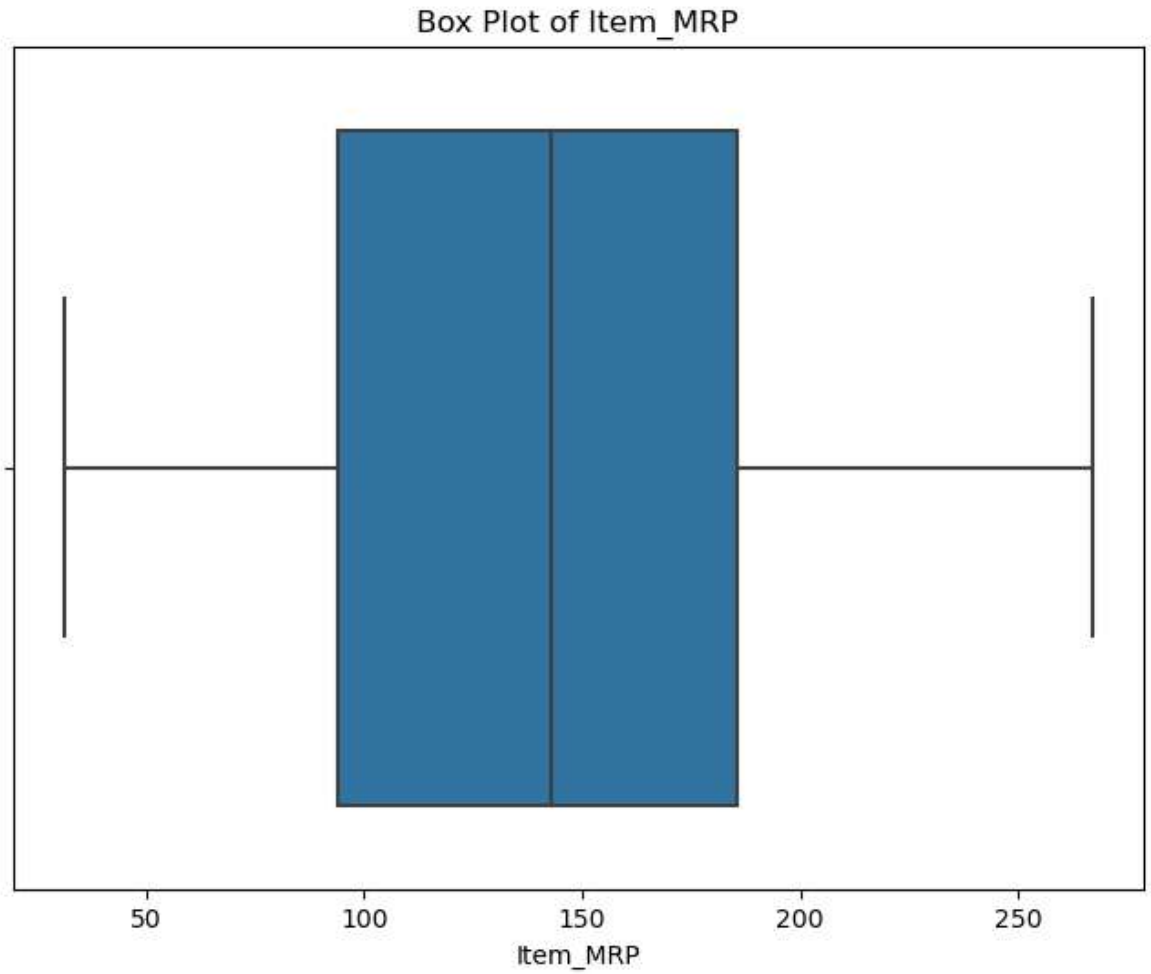
	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
count	7060.000000	8523.000000	8523.000000	8523.000000	8523.000000
mean	12.857645	0.066132	140.992782	1997.831867	2181.288914
std	4.643456	0.051598	62.275067	8.371760	1706.499616
min	4.555000	0.000000	31.290000	1985.000000	33.290000
25%	8.773750	0.026989	93.826500	1987.000000	834.247400
50%	12.600000	0.053931	143.012800	1999.000000	1794.331000
75%	16.850000	0.094585	185.643700	2004.000000	3101.296400
max	21.350000	0.328391	266.888400	2009.000000	13086.964800

In [39]:
```
%matplotlib inline
```

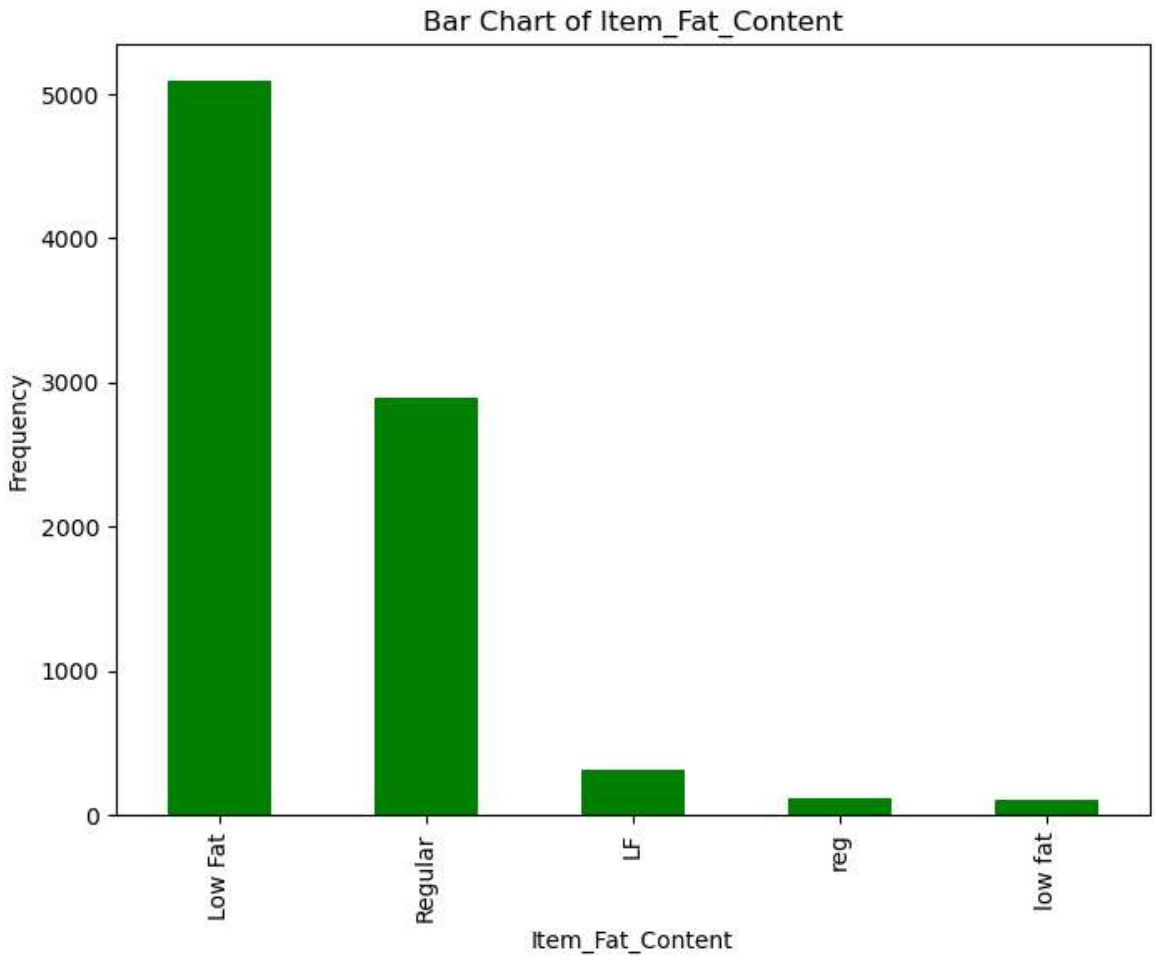
In [40]:
```
import matplotlib.pyplot as plt
import seaborn as sns
# Histogram for Item_Weight
plt.figure(figsize=(8, 6))
plt.hist(df['Item_Weight'].dropna(), bins=20, color='blue', edgecolor='black'
plt.title('Histogram of Item_Weight')
plt.xlabel('Item_Weight')
```



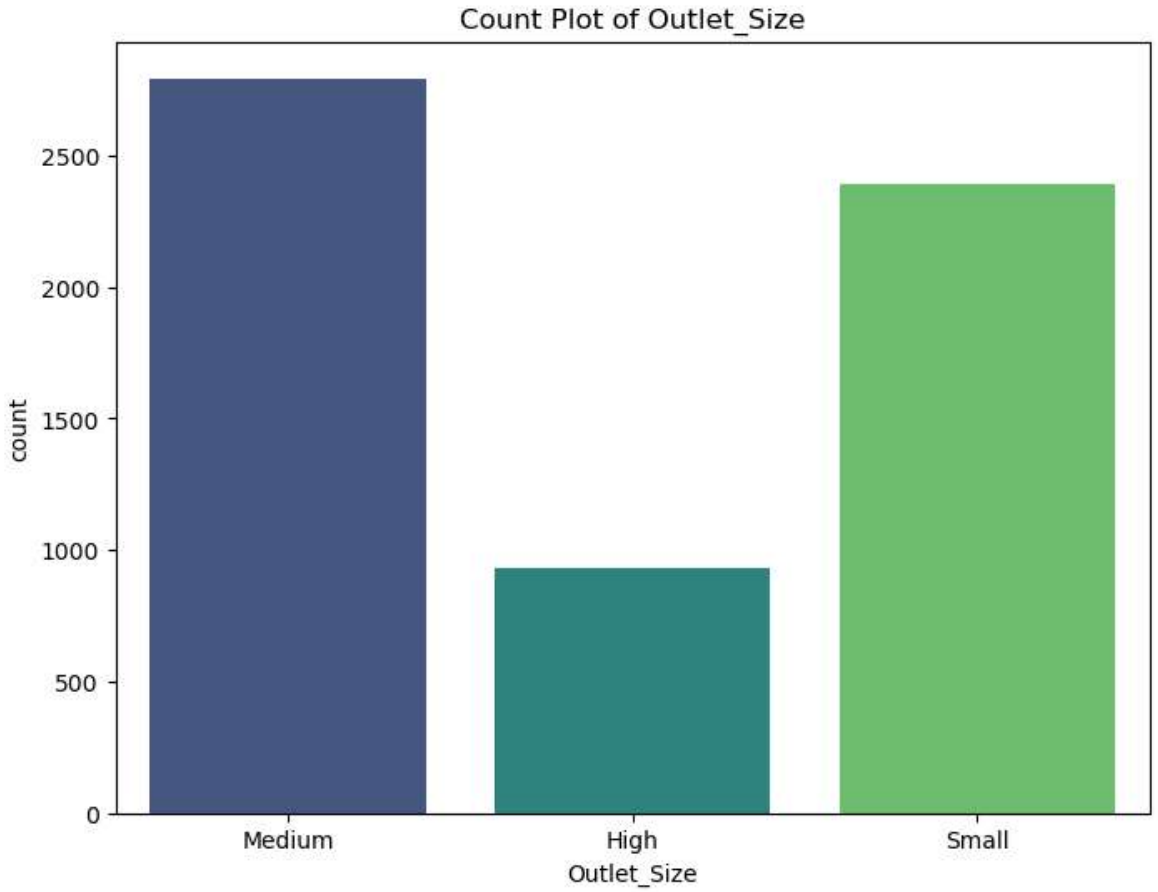
In [41]:
```
# Box plot for Item_MRP
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['Item_MRP'])
```



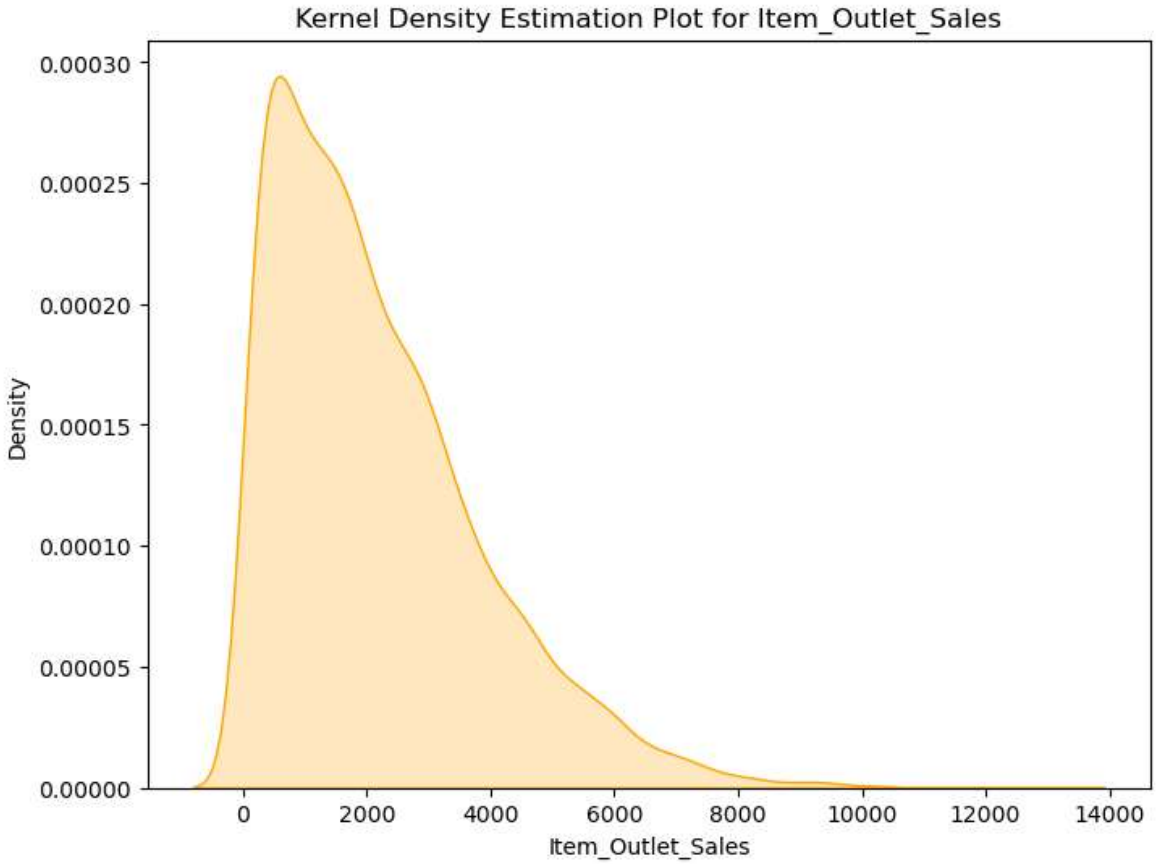
In [42]:
```
# Bar chart for Item_Fat_Content
plt.figure(figsize=(8, 6))
df['Item_Fat_Content'].value_counts().plot(kind='bar', color='green')
plt.title('Bar Chart of Item_Fat_Content')
plt.xlabel('Item_Fat_Content')
```



In [43]:
```
# Count plot for Outlet_Size
plt.figure(figsize=(8, 6))
sns.countplot(x='Outlet_Size', data=df, palette='viridis')
```



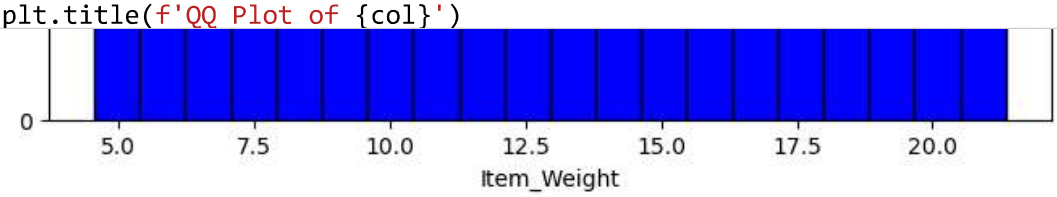
In [44]:
```
# Kernel Density Estimation (KDE) plot for Item_Outlet_Sales
plt.figure(figsize=(8, 6))
sns.kdeplot(df['Item_Outlet_Sales'], fill=True, color='orange')
plt.title('Kernel Density Estimation Plot for Item_Outlet_Sales')
plt.xlabel('Item_Outlet_Sales')
```



In [45]:
```
# Separate numerical and categorical variables
numerical_variable = df.select_dtypes(include=['float64', 'int64']).columns
```

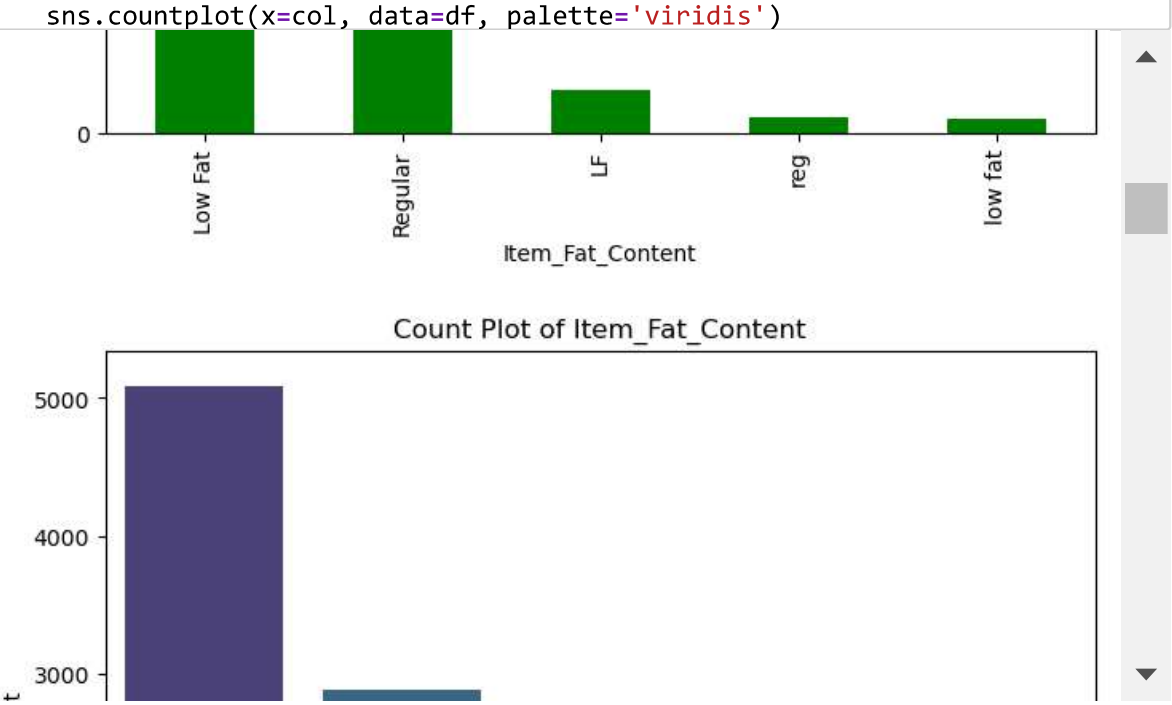
In [46]:

```
import statsmodels.api as sm
# Univariate analysis for Numerical Variables
for col in numerical_variable:
    plt.figure(figsize=(8, 6))

    # Histogram
    plt.hist(df[col].dropna(), bins=20, color='blue', edgecolor='black')
    plt.title(f'Histogram of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()

    # Box Plot
    plt.figure(figsize=(8, 6))
    sns.boxplot(x=df[col])
    plt.title(f'Box Plot of {col}')
    plt.show()

    # Kernel Density Estimation (KDE)
    plt.figure(figsize=(8, 6))
    sns.kdeplot(df[col], fill=True, color='orange')
    plt.title(f'Kernel Density Estimation Plot of {col}')
    plt.xlabel(col)
    plt.ylabel('Density')
    plt.show()

    # QQ Plot
    plt.figure(figsize=(8, 6))
    sm.qqplot(df[col], line='s')
    plt.title(f'QQ Plot of {col}')
```



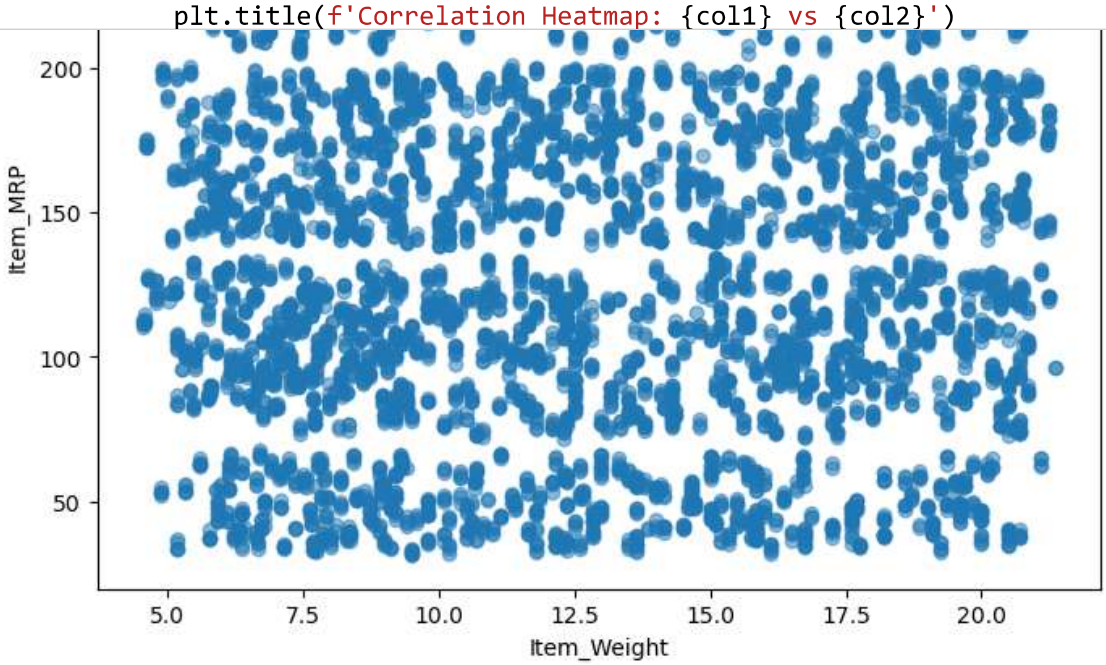
In [47]:

```
# Univariate analysis for Categorical Variables
for col in categorical_variable:
    plt.figure(figsize=(8, 6))

    # Bar Chart
    df[col].value_counts().plot(kind='bar', color='green')
    plt.title(f'Bar Chart of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()

    # Count Plot
    plt.figure(figsize=(8, 6))
    sns.countplot(x=col, data=df, palette='viridis')
```

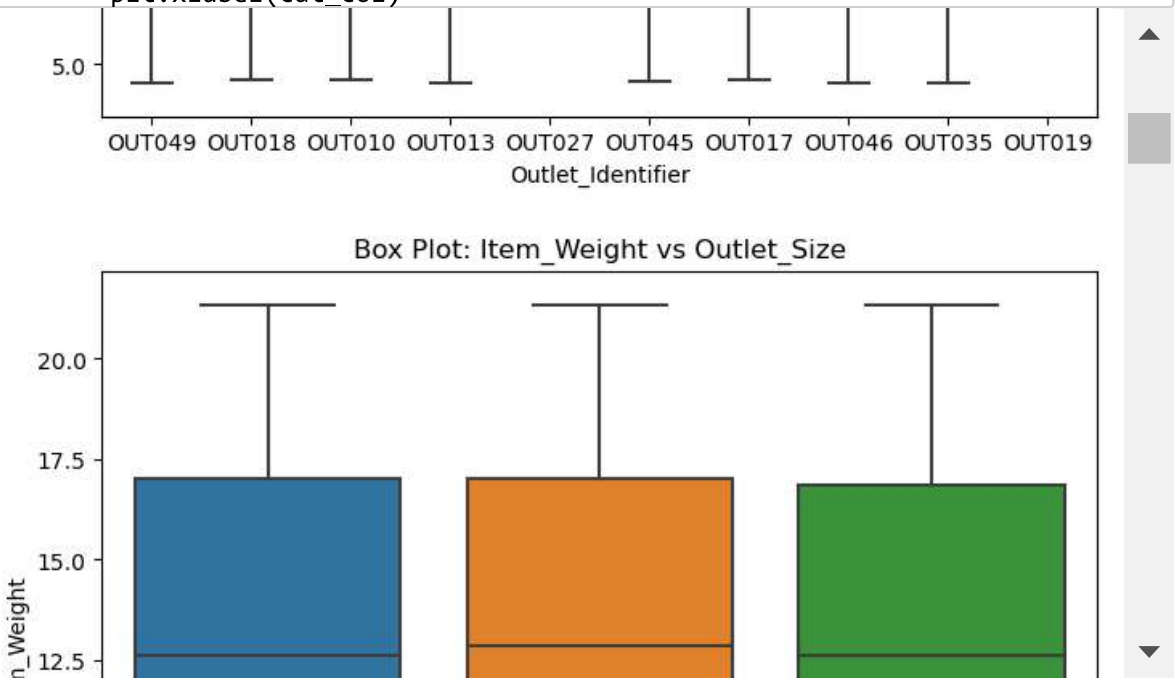


In [48]:

```
# Bivariate analysis for Numerical-Numerical variables
for col1 in numerical_variable:
    for col2 in numerical_variable:
        if col1 != col2:
            plt.figure(figsize=(8, 6))

            # Scatter plot
            plt.scatter(df[col1], df[col2], alpha=0.5)
            plt.title(f'Scatter Plot: {col1} vs {col2}')
            plt.xlabel(col1)
            plt.ylabel(col2)
            plt.show()

            # Pair plot for selected numerical variables
            sns.pairplot(df[[col1, col2]])
            plt.show()

            # Correlation heatmap
            correlation_matrix = df[[col1, col2]].corr()
            plt.figure(figsize=(8, 6))
            sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=
            plt.title(f'Correlation Heatmap: {col1} vs {col2}')
```



In [49]:
```
# Bivariate analysis for Numerical-Categorical variables
for num_col in numerical_variable:
    for cat_col in categorical_variable:
        plt.figure(figsize=(8, 6))

        # Box plot
        sns.boxplot(x=df[cat_col], y=df[num_col])
        plt.title(f'Box Plot: {num_col} vs {cat_col}')
        plt.xlabel(cat_col)
```



In [50]:
```
# Impute missing values
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_10868\2736611607.py:2: FutureWarnin
g: The default value of numeric_only in DataFrame.mean is deprecated. In a f
uture version, it will default to False. In addition, specifying 'numeric_on
ly=None' is deprecated. Select only valid columns or specify the value of nu
meric_only to silence this warning.
  df_imputed = df.fillna(df.mean())
```

In [51]:
```
# Assuming df_imputed is the DataFrame with imputed values
```

In [52]:
```
train
```

Out[52]:

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

8523 rows × 12 columns

In [55]:
```
test =pd.read_csv(r"C:\Users\HP\Downloads\9961_14084_bundle_archive\Test.csv"
```

In [56]:
```
test
```

Out[56]:

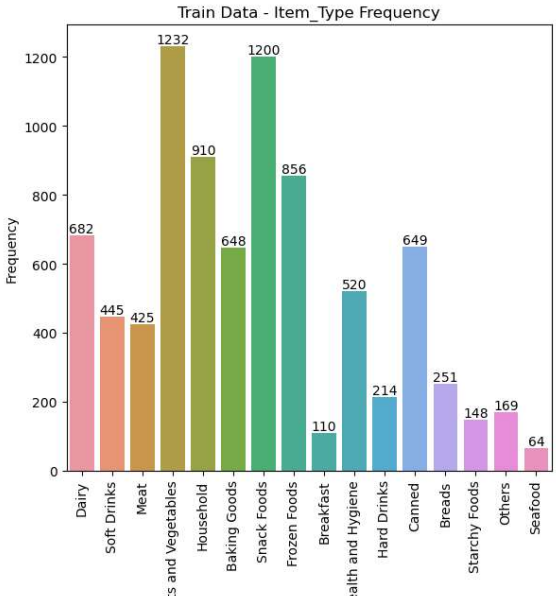
ı_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establis
20.750	Low Fat	0.007565	Snack Foods	107.8622	OUT049	
8.300	reg	0.038428	Dairy	87.3198	OUT017	
14.600	Low Fat	0.099575	Others	241.7538	OUT010	
7.315	Low Fat	0.015388	Snack Foods	155.0340	OUT017	
NaN	Regular	0.118599	Dairy	234.2300	OUT027	
...	...	...	...	...	...	
10.500	Regular	0.013496	Snack Foods	141.3154	OUT046	
7.600	Regular	0.142991	Starchy Foods	169.1448	OUT018	
10.000	Low Fat	0.073529	Health and Hygiene	118.7440	OUT045	
15.300	Regular	0.000000	Canned	214.6218	OUT017	
9.500	Regular	0.104720	Canned	79.7960	OUT045	

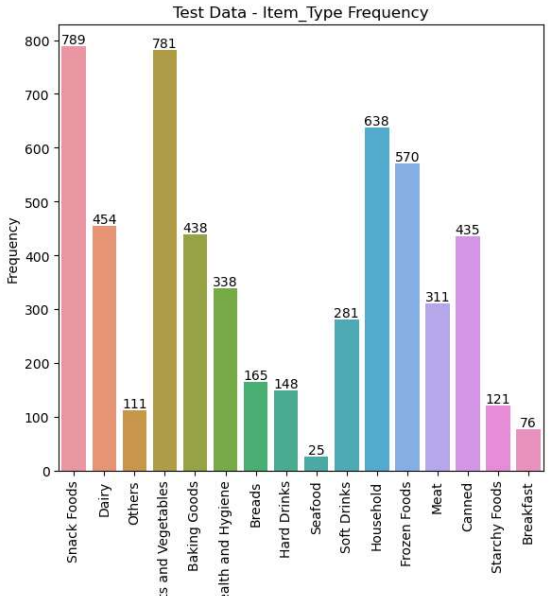
In [71]:
```
import os
import math
import optuna
import pickle
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from datetime import datetime
from pandas_profiling import ProfileReport
from sklearn.impute import KNNImputer
from sklearn.preprocessing import LabelEncoder, PolynomialFeatures, StandardS
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, cross_val_score, KFold
from sklearn.metrics import mean_absolute_error , mean_squared_error , r2_sco
from sklearn.model_selection import RandomizedSearchCV
```

```
All dependencies are imported.
```

In [72]:
```
fig, axes = plt.subplots(7, 2, figsize=(12, 48))
columns = ["Item_Type", "Item_Fat_Content", "Outlet_Size", "Outlet_Establishme
           "Outlet_Location_Type", "Outlet_Type"]
print("Visualizing Discrete Data Spread")
for i, label in enumerate(columns):
    row = i
    col1 = 0
    ax1 = sns.countplot(data=train, x=label, ax=axes[row, col1])
    ax1.set_xlabel(label)
    ax1.set_ylabel("Frequency")
    ax1.set_xticklabels(ax1.get_xticklabels(), rotation=90)
    ax1.set_title(f"Train Data - {label} Frequency")
    for container in ax1.containers:
        ax1.bar_label(container, label_type="edge")
    col2 = 1
    ax2 = sns.countplot(data=test, x=label, ax=axes[row, col2])
    ax2.set_xlabel(label)
    ax2.set_ylabel("Frequency")
    ax2.set_xticklabels(ax2.get_xticklabels(), rotation=90)
    ax2.set_title(f"Test Data - {label} Frequency")
    for container in ax2.containers:
        ax2.bar_label(container, label_type="edge")
```

Visualizing Discrete Data Spread





In [73]:
```
print("Filling Outlet Size null values with mode:")
train["Outlet_Size"].fillna(train["Outlet_Size"].mode()[0], inplace=True)
test["Outlet_Size"].fillna(test["Outlet_Size"].mode()[0], inplace=True)
print("Null Values:")
print("Train:")
display(train.isnull().sum())
print()
```

```
Filling Outlet Size null values with mode:
Null Values:
Train:

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
Item_Type_Code                     0
Is_Low_Fat                         0
Outlet_Age                         0
Item_Fat_Content_LabelEncoded      0
Item_Type_Breads                   0
Item_Type_Breakfast                0
Item_Type_Canned                   0
Item_Type_Dairy                    0
Item_Type_Frozen Foods             0
Item_Type_Fruits and Vegetables    0
Item_Type_Hard Drinks              0
Item_Type_Health and Hygiene       0
Item_Type_Household                0
Item_Type_Meat                     0
Item_Type_Others                   0
Item_Type_Seafood                  0
Item_Type_Snack Foods              0
Item_Type_Soft Drinks              0
Item_Type_Starchy Foods            0
Outlet_Size_Medium                 0
Outlet_Size_Small                  0
Outlet_Size_nan                    0
dtype: int64


Test:
```

```
Item_Identifier               0
Item_Weight                 976
Item_Fat_Content              0
Item_Visibility               0
Item_Type                     0
Item_MRP                      0
Outlet_Identifier             0
Outlet_Establishment_Year     0
Outlet_Size                   0
Outlet_Location_Type          0
Outlet_Type                   0
dtype: int64
```

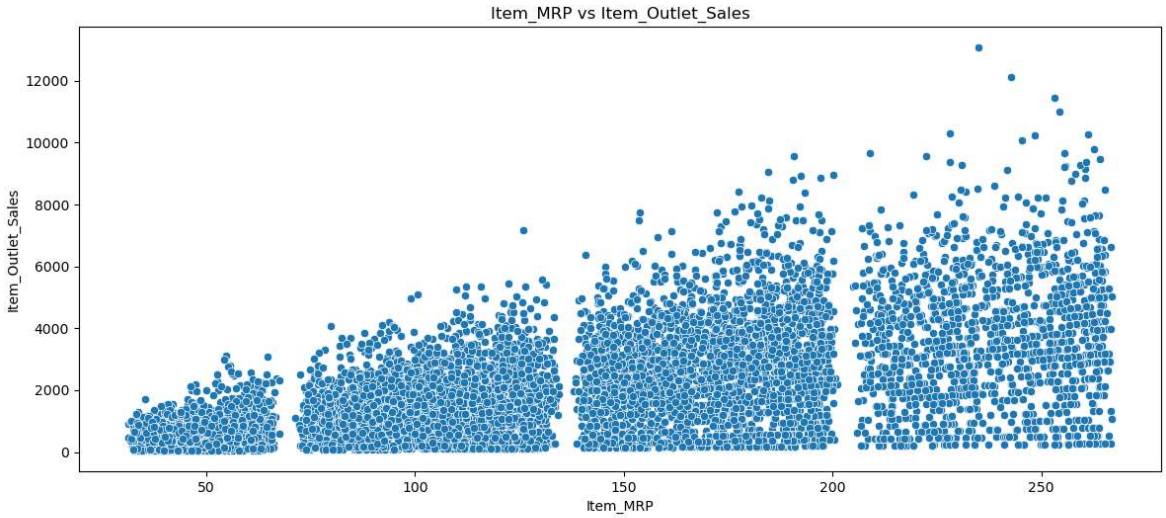
In [74]:
```
print("Filling Item Weight null values with mean:")
train["Item_Weight"].fillna(train["Item_Weight"].mean(),inplace=True)
test["Item_Weight"].fillna(train["Item_Weight"].mean(),inplace=True)
print("Null Values:")
print("Train:")
display(train.isnull().sum())
print()
```

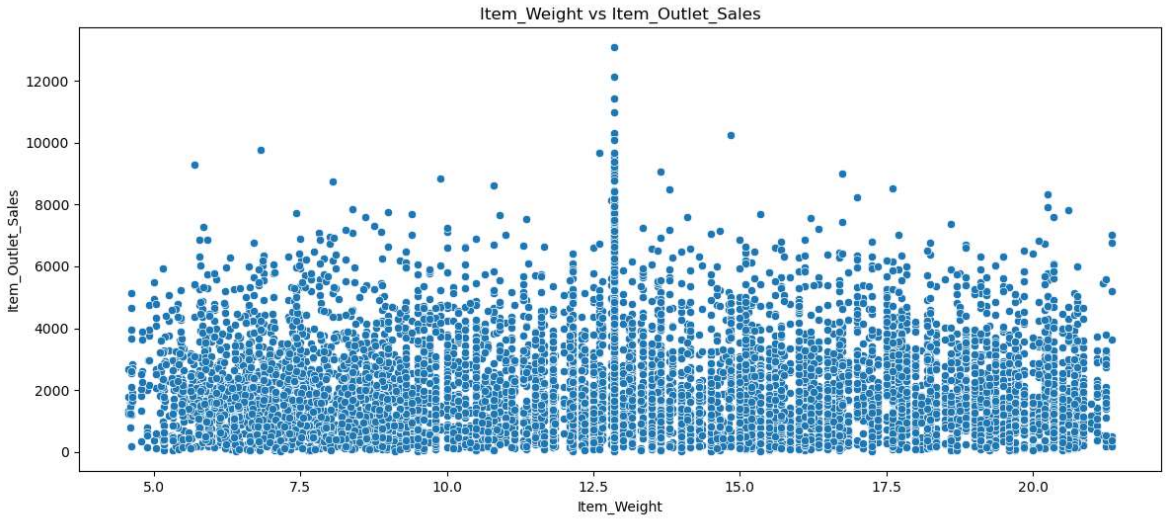
```
Filling Item Weight null values with mean:
Null Values:
Train:

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
Item_Type_Code                   0
Is_Low_Fat                       0
Outlet_Age                       0
Item_Fat_Content_LabelEncoded    0
Item_Type_Breads                 0
Item_Type_Breakfast              0
Item_Type_Canned                 0
Item_Type_Dairy                  0
Item_Type_Frozen Foods           0
Item_Type_Fruits and Vegetables  0
Item_Type_Hard Drinks            0
Item_Type_Health and Hygiene     0
Item_Type_Household              0
Item_Type_Meat                   0
Item_Type_Others                 0
Item_Type_Seafood                0
Item_Type_Snack Foods            0
Item_Type_Soft Drinks            0
Item_Type_Starchy Foods          0
Outlet_Size_Medium               0
Outlet_Size_Small                0
Outlet_Size_nan                  0
dtype: int64


Test:
```

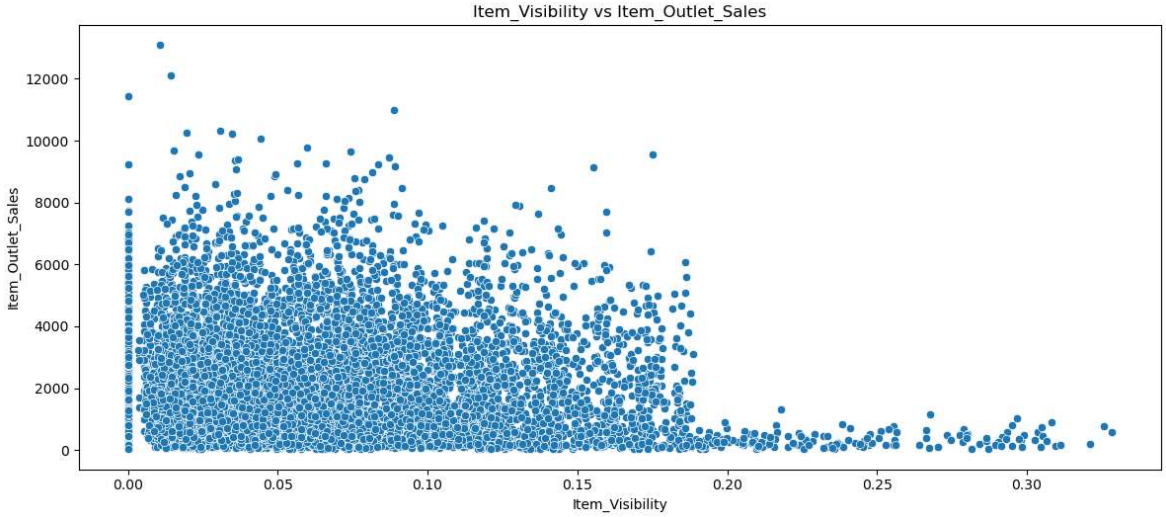
```
Item_Identifier              0
Item_Weight                  0
Item_Fat_Content             0
Item_Visibility              0
Item_Type                    0
Item_MRP                     0
Outlet_Identifier            0
Outlet_Establishment_Year    0
Outlet_Size                  0
Outlet_Location_Type         0
Outlet_Type                  0
dtype: int64
```

In [75]:
```
fig, axes = plt.subplots(3, 1, figsize=(12,16))
target="Item_Outlet_Sales"
columns = ["Item_MRP", "Item_Weight", "Item_Visibility"]
print(f"Visualizing Quantitative Data Spread W.R.T  {target}")
for i, label in enumerate(columns):
    ax = sns.scatterplot(data=train, x=label, y=target, ax=axes[i])
    ax.set_xlabel(label)
    ax.set_ylabel(f"{target}")
    ax.set_title(f"{label} vs {target}")
plt.subplots_adjust(hspace=0.5)
```

```
Visualizing Quantitative Data Spread W.R.T  Item_Outlet_Sales
```

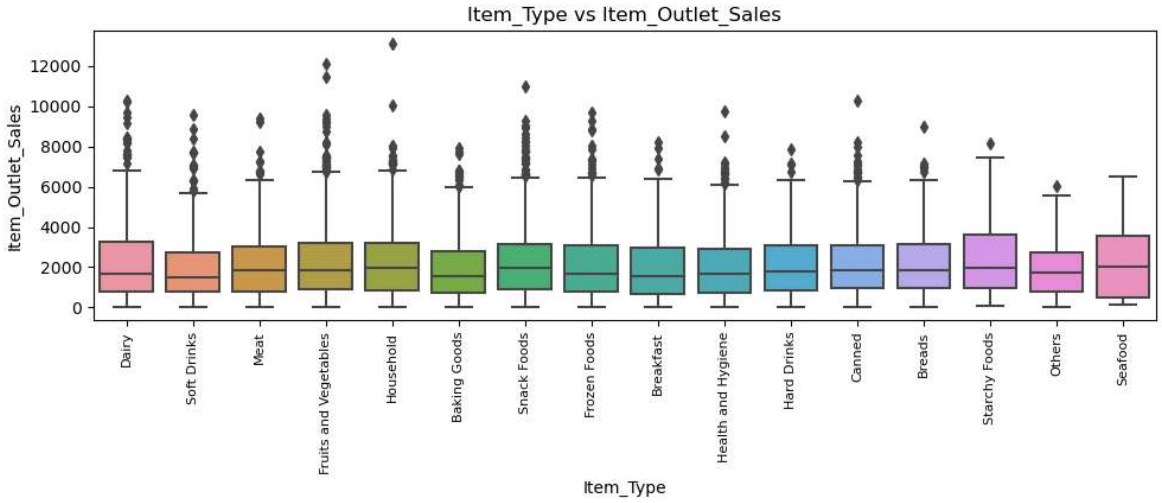

Item_MRP vs Item_Outlet_Sales
Item_Outlet_Sales
Item_MRP
0
2000
4000
6000
8000
10000
12000
50
100
150
200
250

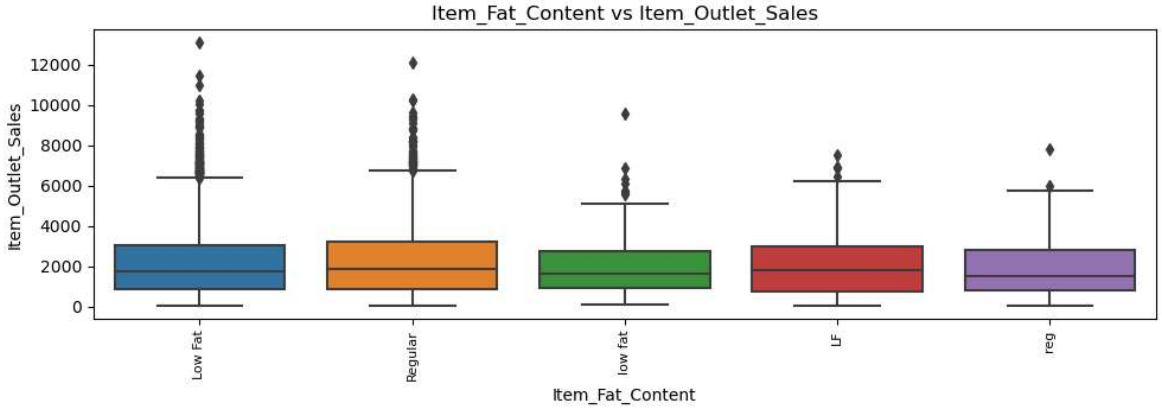

Item_Weight vs Item_Outlet_Sales
Item_Outlet_Sales
Item_Weight
0
2000
4000
6000
8000
10000
12000
5.0
7.5
10.0
12.5
15.0
17.5
20.0

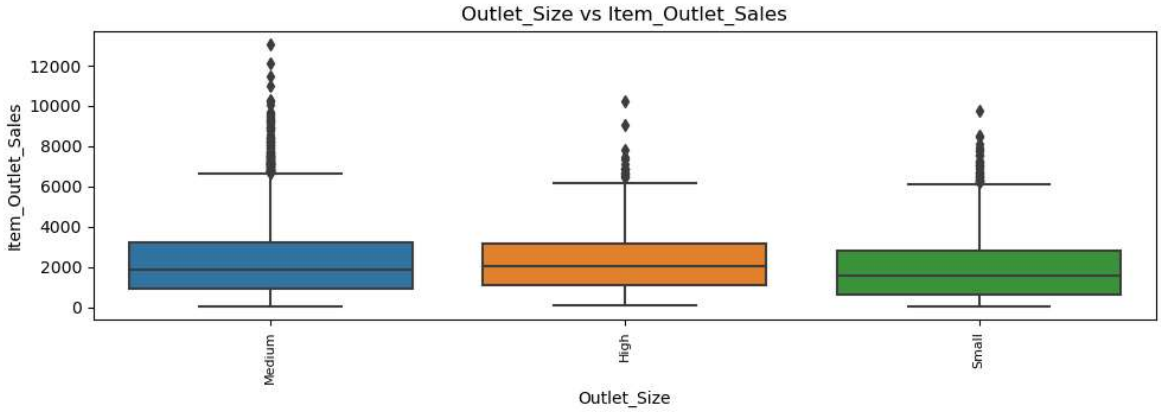

Item_Visibility vs Item_Outlet_Sales
Item_Outlet_Sales
Item_Visibility
0
2000
4000
6000
8000
10000
12000
0.00
0.05
0.10
0.15
0.20
0.25
0.30

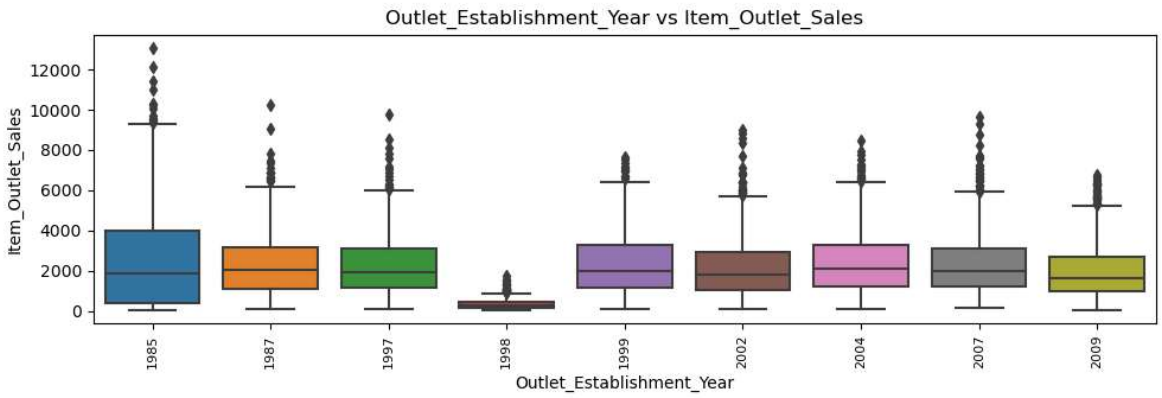
In [76]:
```
fig, axes = plt.subplots(7, 1, figsize=(10, 30))
target="Item_Outlet_Sales"
columns = ["Item_Type", "Item_Fat_Content", "Outlet_Size", "Outlet_Establishm
           "Outlet_Location_Type", "Outlet_Type"]
print(f"Visualizing Quantitative Data Spread W.R.T  {target}")
for i, label in enumerate(columns):
    ax = sns.boxplot(data=train, x=label, y=target, ax=axes[i])
    ax.set_xlabel(label)
    ax.set_ylabel(f"{target}")
    ax.set_title(f"{label} vs {target}")
    ax.tick_params(axis='x',rotation=90, labelsize=8)
plt.subplots_adjust(hspace=0.5)
```

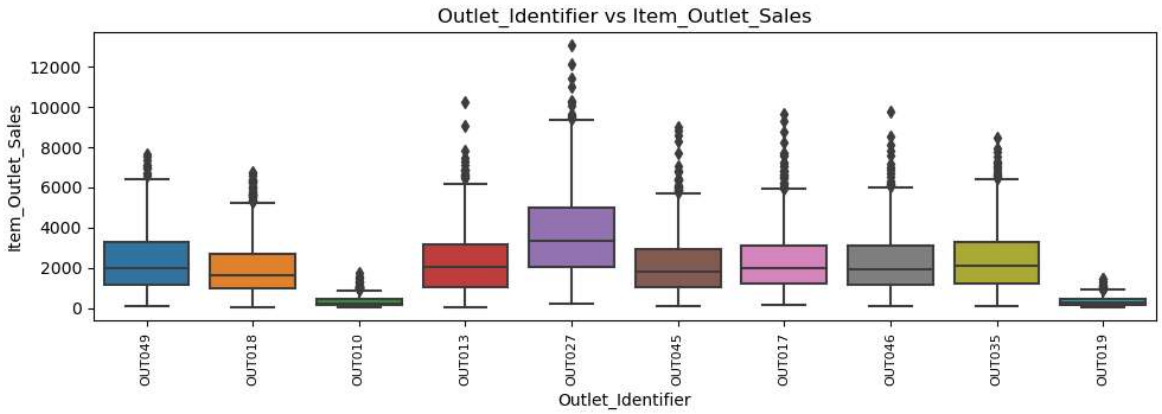
```
Visualizing Quantitative Data Spread W.R.T  Item_Outlet_Sales
```

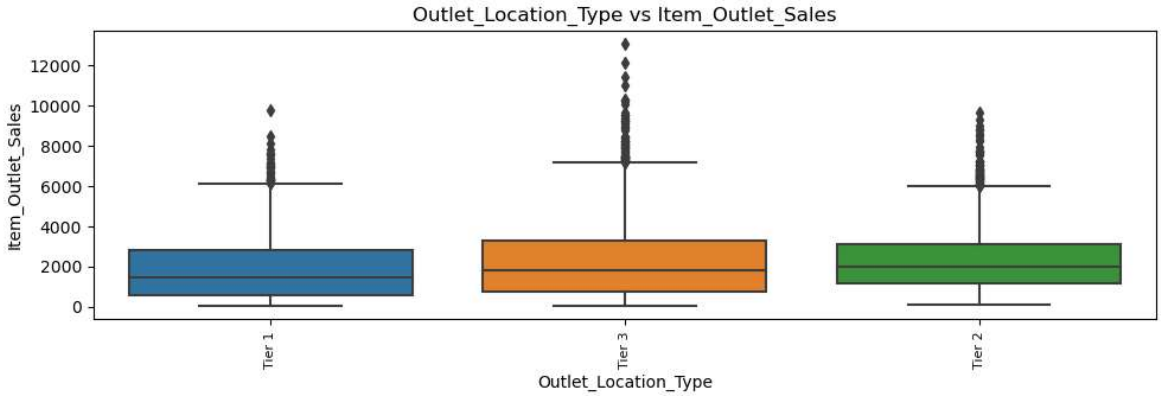

Item_Type vs Item_Outlet_Sales
Item_Outlet_Sales
0
2000
4000
6000
8000
10000
12000
Dairy
Soft Drinks
Meat
Fruits and Vegetables
Household
Baking Goods
Snack Foods
Frozen Foods
Breakfast
Health and Hygiene
Hard Drinks
Canned
Breads
Starchy Foods
Others
Seafood
Item_Type

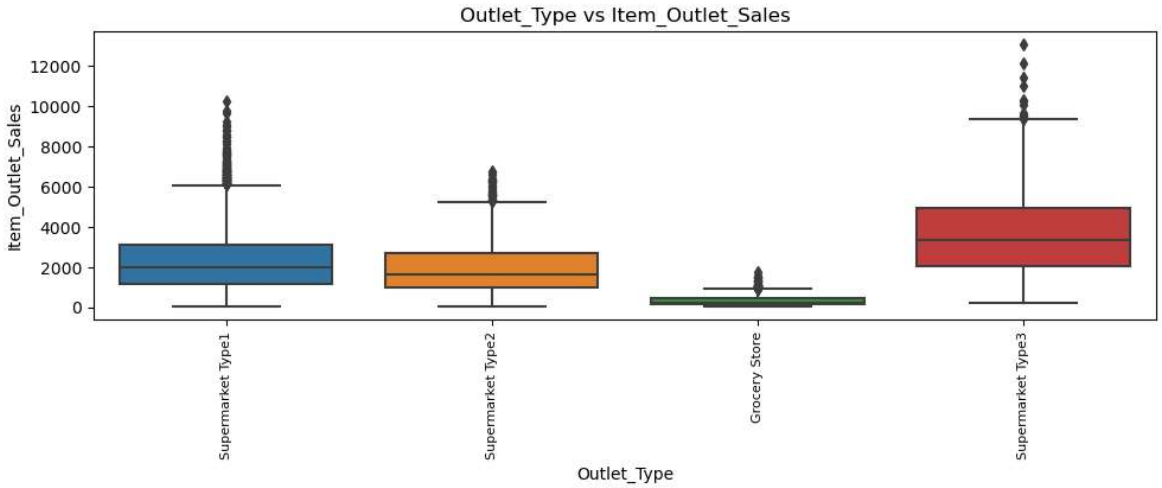

Item_Fat_Content vs Item_Outlet_Sales
Item_Outlet_Sales
0
2000
4000
6000
8000
10000
12000
Low Fat
Regular
low fat
LF
reg
Item_Fat_Content


Outlet_Size vs Item_Outlet_Sales
Item_Outlet_Sales
0
2000
4000
6000
8000
10000
12000
Medium
High
Small
Outlet_Size


Outlet_Establishment_Year vs Item_Outlet_Sales
Item_Outlet_Sales
0
2000
4000
6000
8000
10000
12000
1985
1987
1997
1998
1999
2002
2004
2007
2009
Outlet_Establishment_Year


Outlet_Identifier vs Item_Outlet_Sales
Item_Outlet_Sales
0
2000
4000
6000
8000
10000
12000
OUT049
OUT018
OUT010
OUT013
OUT027
OUT045
OUT017
OUT046
OUT035
OUT019
Outlet_Identifier


Outlet_Location_Type vs Item_Outlet_Sales
Item_Outlet_Sales
0
2000
4000
6000
8000
10000
12000
Tier 1
Tier 3
Tier 2
Outlet_Location_Type


Outlet_Type vs Item_Outlet_Sales
Item_Outlet_Sales
0
2000
4000
6000
8000
10000
12000
Supermarket Type1
Supermarket Type2
Grocery Store
Supermarket Type3
Outlet_Type

In [77]:
```
print("Function to detect outliers:")
def detect_outliers(df, feature):
    q1=df[feature].quantile(0.25)
    q3=df[feature].quantile(0.75)
    iqr=q3-q1
    upper=q3+1.5*iqr
    lower=q1-1.5*iqr
    return upper, lower
outlier_columns=["Item_Visibility","Item_Outlet_Sales"]
for column in outlier_columns:
    upper,lower=detect_outliers(train,column)
    print(f"Upper and Lower limit of {column} are: {upper} & {lower}.")
```

```
Function to detect outliers:
Upper and Lower limit of Item_Visibility are: 0.195979015 & -0.0744042450000
0001.
Removing the outliers in Item_Visibility.
Upper and Lower limit of Item_Outlet_Sales are: 6499.2067 & -2499.7460999999
994.
Removing the outliers in Item_Outlet_Sales.
```

In [78]:
```
print("Making corrections in the label names of Item Fat Content:")
train['Item_Fat_Content'] = train['Item_Fat_Content'].map({'Low Fat' :'Low Fa
                                                           'low fat' :"Low Fa
                                                           'LF'      :"Low Fa
                                                           'Regular' :'Regula
                                                           'reg'     :"Regula
                                                          })

test['Item_Fat_Content'] = test['Item_Fat_Content'].map({'Low Fat' :'Low Fat'
                                                         'low fat' :"Low Fa
                                                         'LF'      :"Low Fa
                                                         'Regular' :'Regula
```

```
Making corrections in the label names of Item Fat Content:
```

In [79]:
```
current_year=datetime.now().year
print("Replacing the Outlet Establishment Year with Outlet Age.")
train['Outlet_Age'] = current_year - train['Outlet_Establishment_Year']
test['Outlet_Age'] = current_year - test['Outlet_Establishment_Year']
del train['Outlet_Establishment_Year']
del test['Outlet_Establishment_Year']
print("Train Data:")
display(train)
print()
```

```
Replacing the Outlet Establishment Year with Outlet Age.
Train Data:
```

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

8193 rows × 33 columns

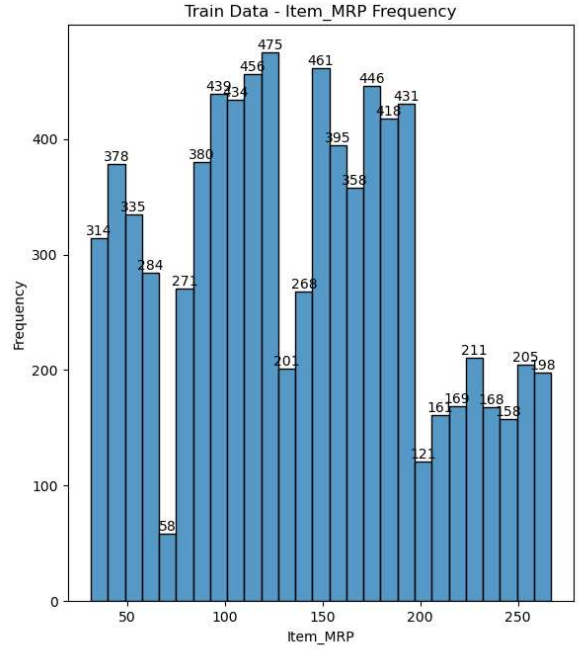
```
Test Data
```

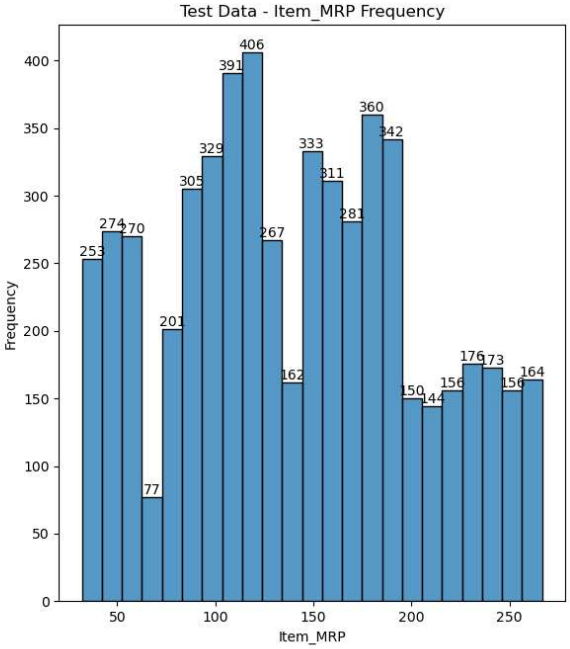
	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outle
0	FDW58	20.750000	Low Fat	0.007565	Snack Foods	107.8622	
1	FDW14	8.300000	Regular	0.038428	Dairy	87.3198	
2	NCN55	14.600000	Low Fat	0.099575	Others	241.7538	
3	FDQ58	7.315000	Low Fat	0.015388	Snack Foods	155.0340	
4	FDY38	12.857645	Regular	0.118599	Dairy	234.2300	
...	...	...	...	...	...	...	
5676	FDB58	10.500000	Regular	0.013496	Snack Foods	141.3154	
5677	FDD47	7.600000	Regular	0.142991	Starchy Foods	169.1448	
5678	NCO17	10.000000	Low Fat	0.073529	Health and Hygiene	118.7440	
5679	FDJ26	15.300000	Regular	0.000000	Canned	214.6218	
5680	FDU37	9.500000	Regular	0.104720	Canned	79.7960	

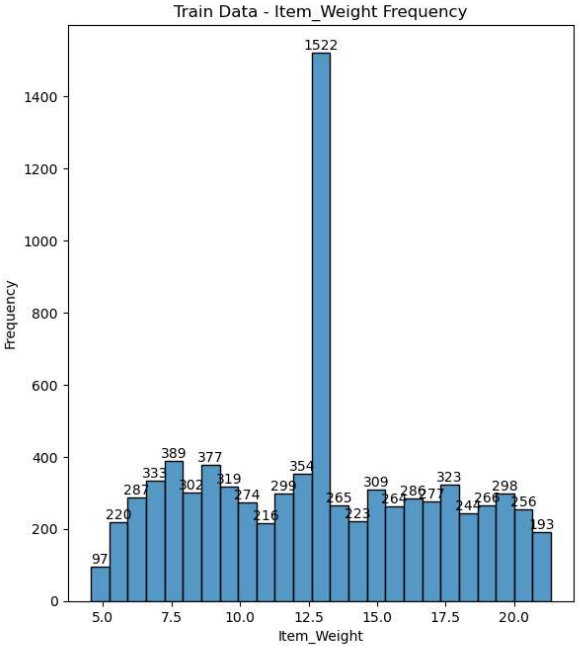
5681 rows × 11 columns

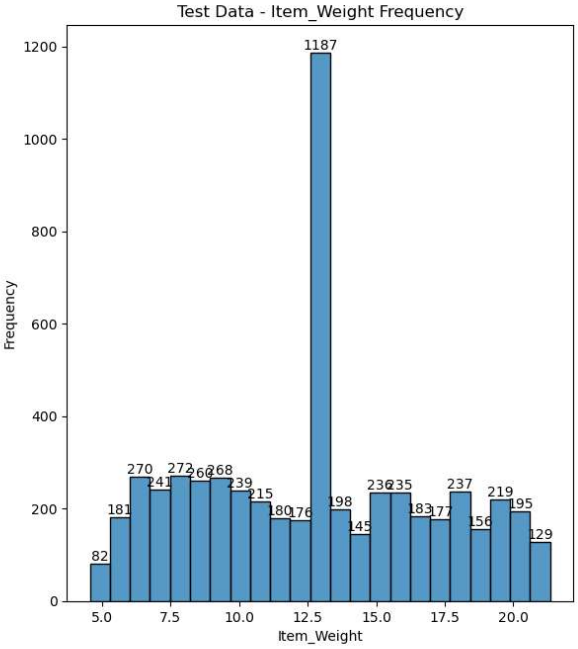
In [80]:
```
fig, axes = plt.subplots(3, 2, figsize=(12, 20))
columns = ["Item_MRP", "Item_Weight", "Item_Visibility"]
print("Visualizing Quantitative Data Spread")
for i, label in enumerate(columns):
    row = i
    ax1 = sns.histplot(train[label], ax=axes[row, 0])
    ax1.set_xlabel(label)
    ax1.set_ylabel("Frequency")
    ax1.set_title(f"Train Data - {label} Frequency")
    for container in ax1.containers:
        ax1.bar_label(container, label_type="edge")
    ax2 = sns.histplot(test[label], ax=axes[row, 1])
    ax2.set_xlabel(label)
    ax2.set_ylabel("Frequency")
    ax2.set_title(f"Test Data - {label} Frequency")
    for container in ax2.containers:
        ax2.bar_label(container, label_type="edge")
```

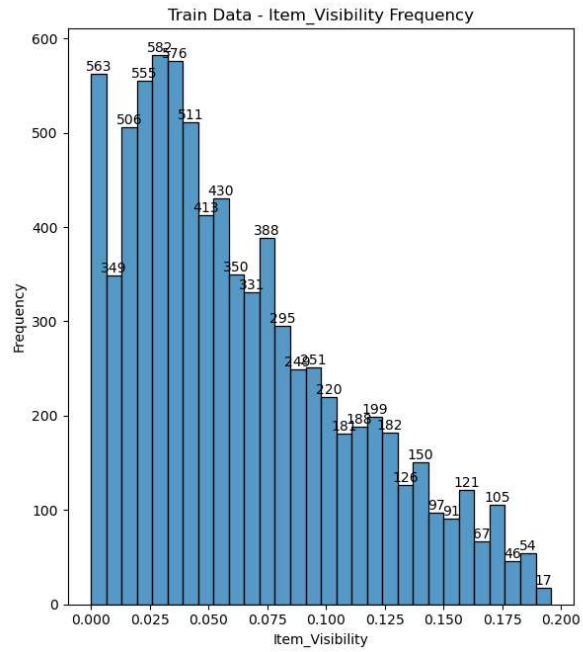
```
Visualizing Quantitative Data Spread
```

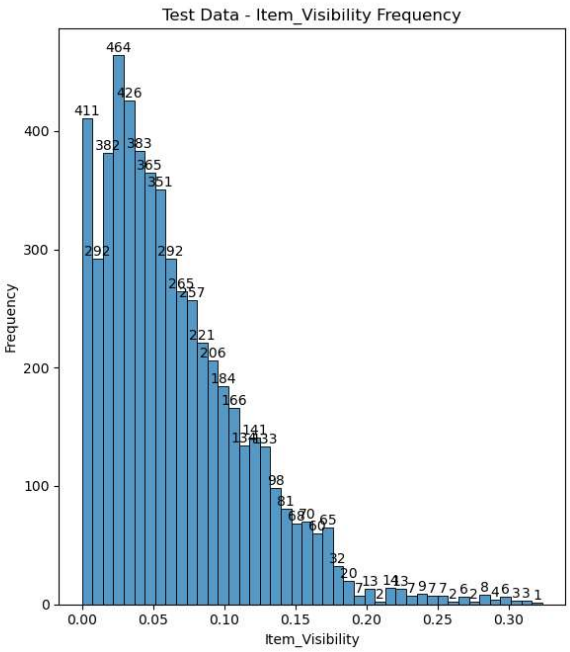

Train Data - Item_MRP Frequency
Frequency
Item_MRP


Test Data - Item_MRP Frequency
Frequency
Item_MRP

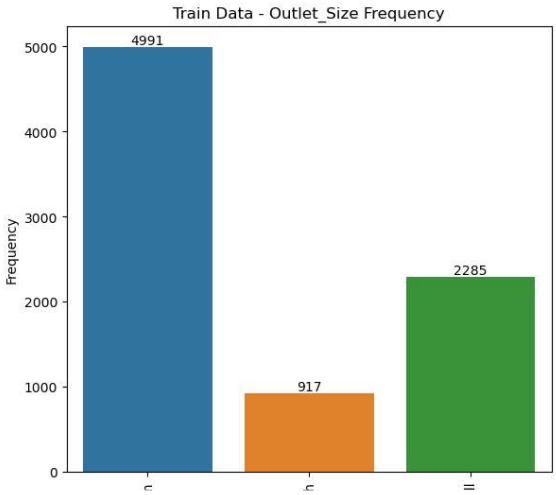

Train Data - Item_Weight Frequency
Frequency
Item_Weight

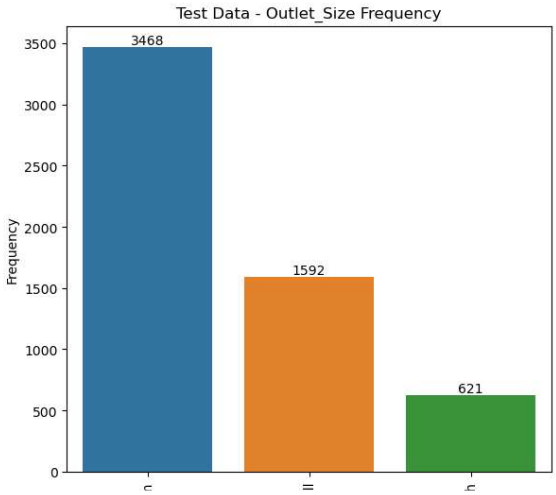

Test Data - Item_Weight Frequency
Frequency
Item_Weight


Train Data - Item_Visibility Frequency
Frequency
Item_Visibility


Test Data - Item_Visibility Frequency
Frequency
Item_Visibility

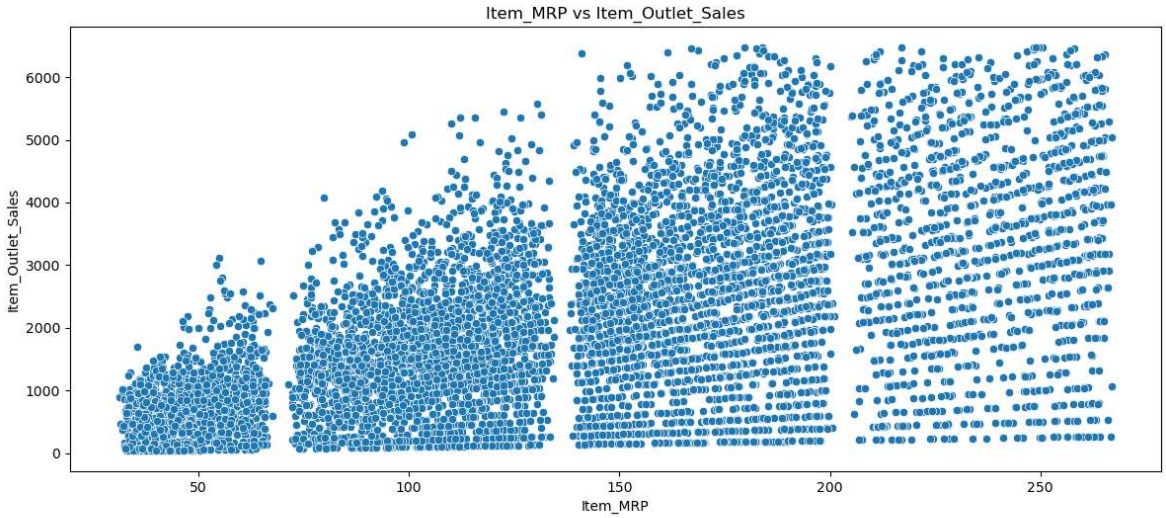
In [81]:
```
fig, axes = plt.subplots(7, 2, figsize=(12, 48))
columns = ["Item_Type", "Item_Fat_Content", "Outlet_Size", "Outlet_Age", "Out
           "Outlet_Location_Type", "Outlet_Type"]
print("Visualizing Discrete Data Spread")
for i, label in enumerate(columns):
    row = i
    col1 = 0
    ax1 = sns.countplot(data=train, x=label, ax=axes[row, col1])
    ax1.set_xlabel(label)
    ax1.set_ylabel("Frequency")
    ax1.set_xticklabels(ax1.get_xticklabels(), rotation=90)
    ax1.set_title(f"Train Data - {label} Frequency")
    for container in ax1.containers:
        ax1.bar_label(container, label_type="edge")
    col2 = 1
    ax2 = sns.countplot(data=test, x=label, ax=axes[row, col2])
    ax2.set_xlabel(label)
    ax2.set_ylabel("Frequency")
    ax2.set_xticklabels(ax2.get_xticklabels(), rotation=90)
    ax2.set_title(f"Test Data - {label} Frequency")
    for container in ax2.containers:
        ax2.bar_label(container, label_type="edge")
```



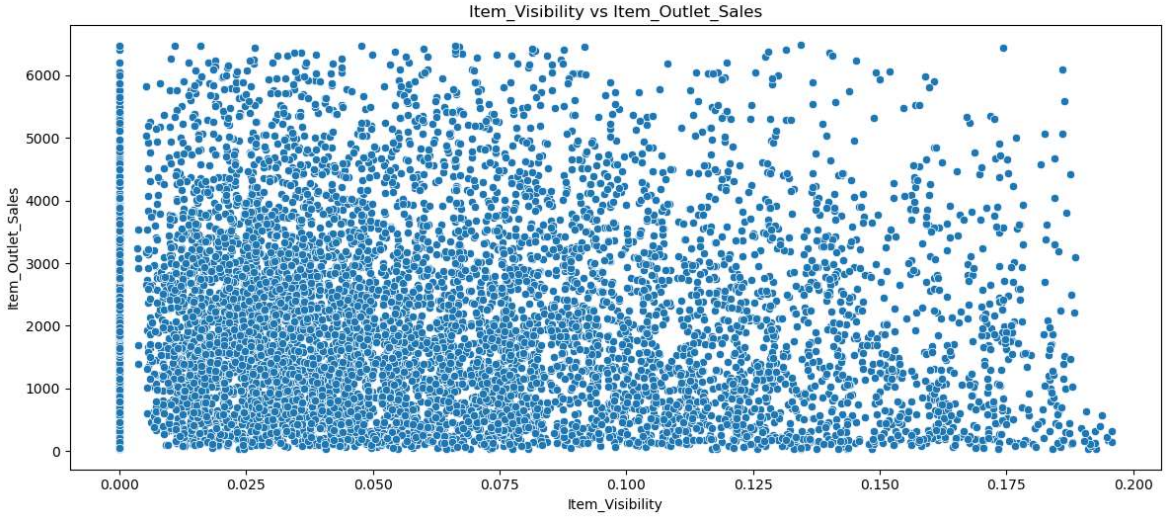


In [82]:
```
fig, axes = plt.subplots(3, 1, figsize=(12,16))
target="Item_Outlet_Sales"
columns = ["Item_MRP", "Item_Weight", "Item_Visibility"]
print(f"Visualizing Quantitative Data Spread W.R.T  {target}")
for i, label in enumerate(columns):
    ax = sns.scatterplot(data=train, x=label, y=target, ax=axes[i])
    ax.set_xlabel(label)
    ax.set_ylabel(f"{target}")
    ax.set_title(f"{label} vs {target}")
plt.subplots_adjust(hspace=0.5)
```

```
Visualizing Quantitative Data Spread W.R.T  Item_Outlet_Sales
```

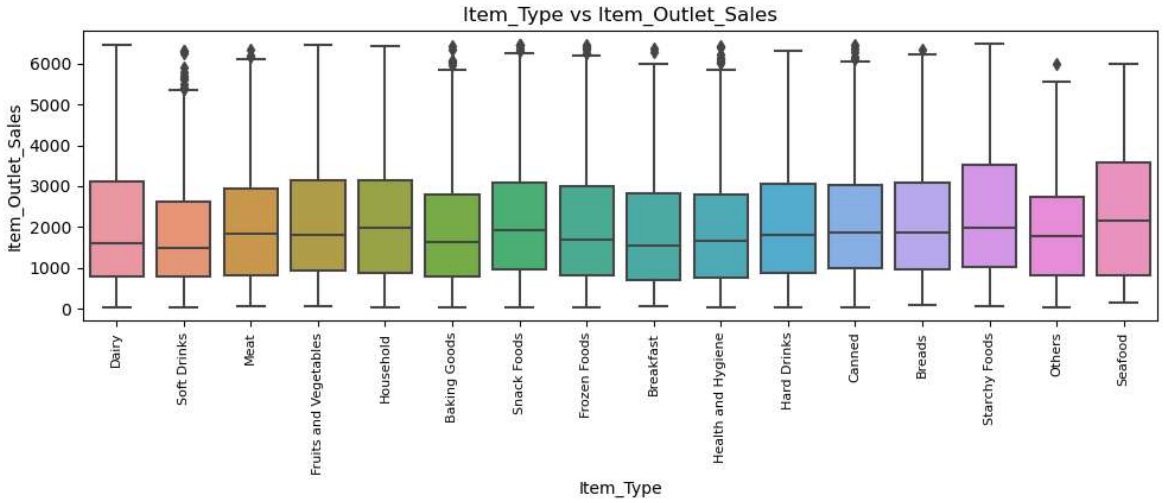

Item_MRP vs Item_Outlet_Sales
Item_Outlet_Sales
Item_MRP
0
1000
2000
3000
4000
5000
6000
50
100
150
200
250

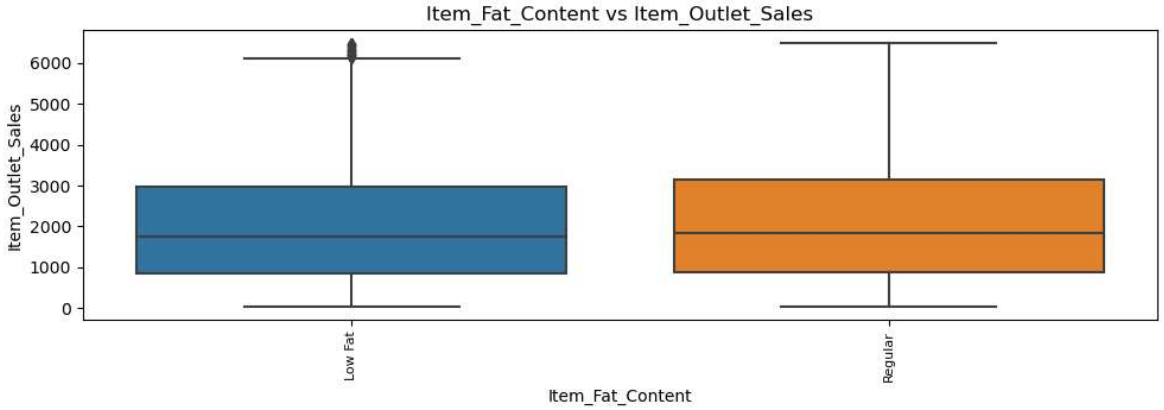

Item_Weight vs Item_Outlet_Sales
Item_Outlet_Sales
Item_Weight
0
1000
2000
3000
4000
5000
6000
5.0
7.5
10.0
12.5
15.0
17.5
20.0

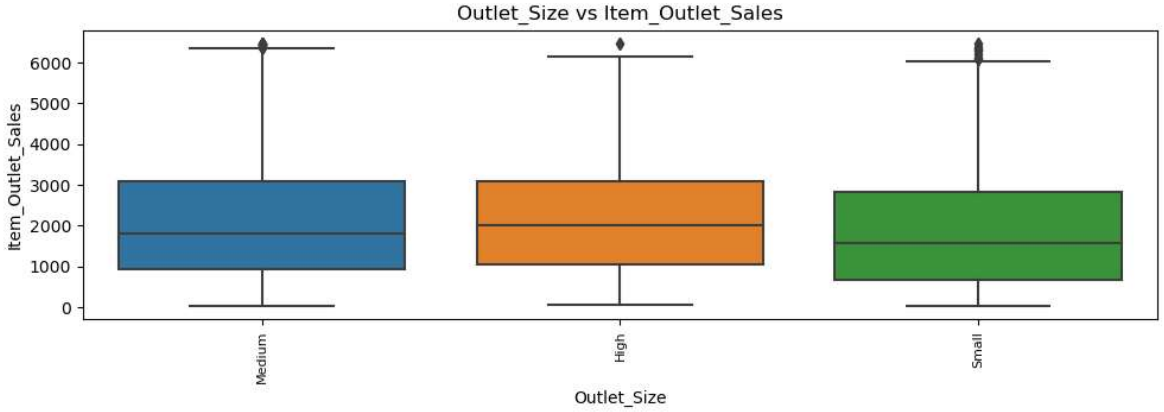

Item_Visibility vs Item_Outlet_Sales
Item_Outlet_Sales
Item_Visibility
0
1000
2000
3000
4000
5000
6000
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
0.200

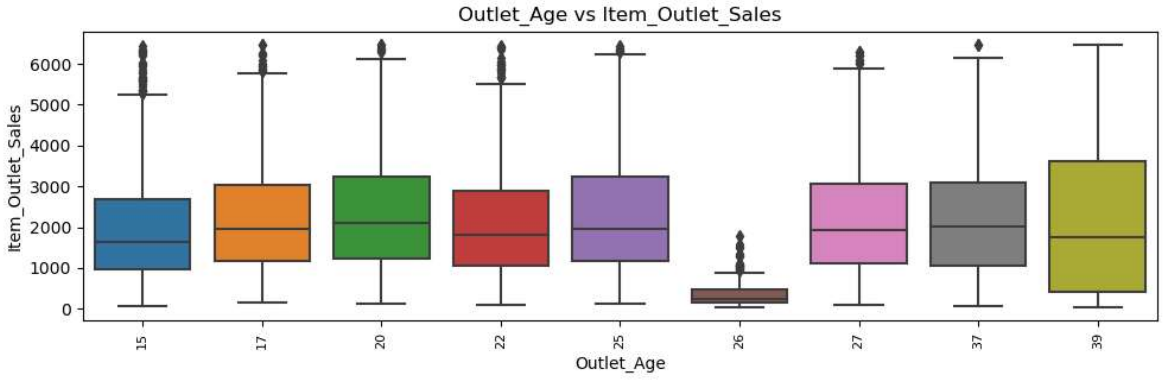
In [83]:
```
fig, axes = plt.subplots(7, 1, figsize=(10, 30))
target="Item_Outlet_Sales"
columns = ["Item_Type", "Item_Fat_Content", "Outlet_Size", "Outlet_Age", "Out
           "Outlet_Location_Type", "Outlet_Type"]
print(f"Visualizing Quantitative Data Spread W.R.T  {target}")
for i, label in enumerate(columns):
    ax = sns.boxplot(data=train, x=label, y=target, ax=axes[i])
    ax.set_xlabel(label)
    ax.set_ylabel(f"{target}")
    ax.set_title(f"{label} vs {target}")
    ax.tick_params(axis='x',rotation=90, labelsize=8)
plt.subplots_adjust(hspace=0.5)
```

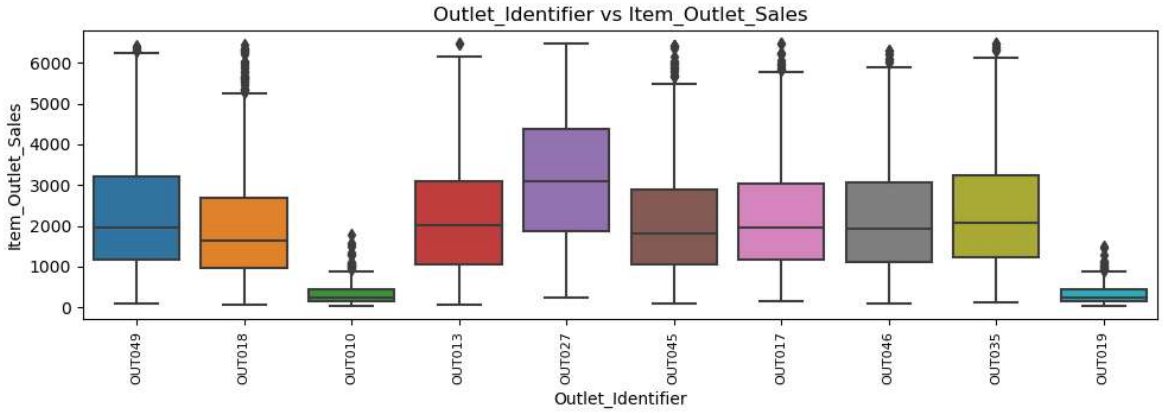
```
Visualizing Quantitative Data Spread W.R.T  Item_Outlet_Sales
```

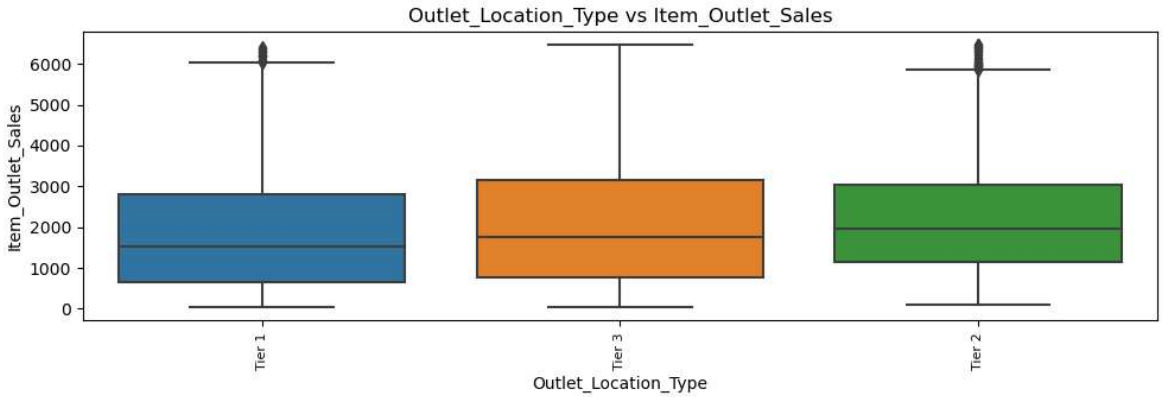

Item_Type vs Item_Outlet_Sales
Item_Outlet_Sales
0
1000
2000
3000
4000
5000
6000
Dairy
Soft Drinks
Meat
Fruits and Vegetables
Household
Baking Goods
Snack Foods
Frozen Foods
Breakfast
Health and Hygiene
Hard Drinks
Canned
Breads
Starchy Foods
Others
Seafood
Item_Type

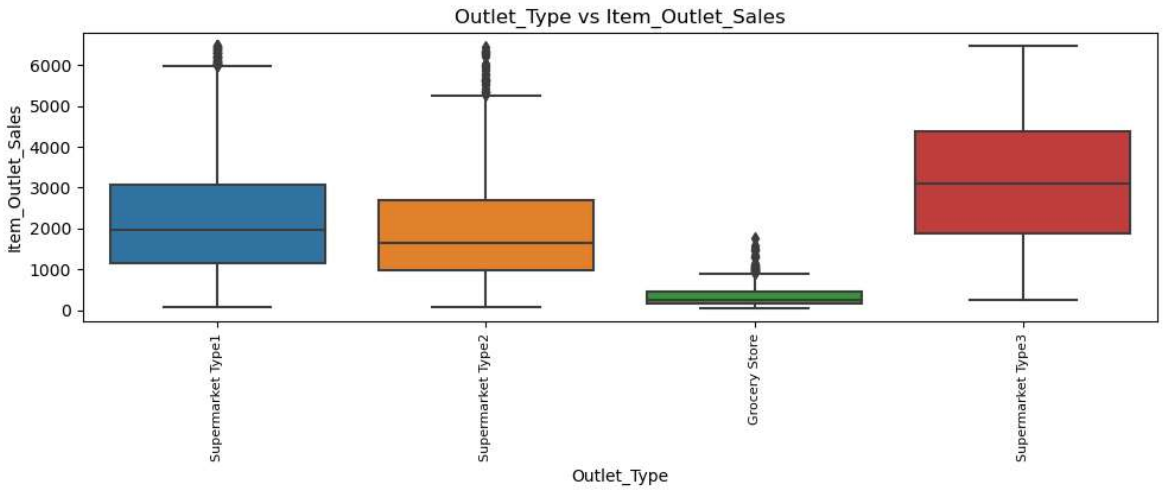

Item_Fat_Content vs Item_Outlet_Sales
Item_Outlet_Sales
0
1000
2000
3000
4000
5000
6000
Low Fat
Regular
Item_Fat_Content


Outlet_Size vs Item_Outlet_Sales
Item_Outlet_Sales
0
1000
2000
3000
4000
5000
6000
Medium
High
Small
Outlet_Size


Outlet_Age vs Item_Outlet_Sales
Item_Outlet_Sales
0
1000
2000
3000
4000
5000
6000
15
17
20
22
25
26
27
37
39
Outlet_Age


Outlet_Identifier vs Item_Outlet_Sales
Item_Outlet_Sales
0
1000
2000
3000
4000
5000
6000
OUT049
OUT018
OUT010
OUT013
OUT027
OUT045
OUT017
OUT046
OUT035
OUT019
Outlet_Identifier


Outlet_Location_Type vs Item_Outlet_Sales
Item_Outlet_Sales
0
1000
2000
3000
4000
5000
6000
Tier 1
Tier 3
Tier 2
Outlet_Location_Type


Outlet_Type vs Item_Outlet_Sales
Item_Outlet_Sales
0
1000
2000
3000
4000
5000
6000
Supermarket Type1
Supermarket Type2
Grocery Store
Supermarket Type3
Outlet_Type

In [84]:
```
train['Outlet_Size'] = train['Outlet_Size'].map({'Small'  : 1,
                                                 'Medium' : 2,
                                                 'High'   : 3
                                                }).astype(int)

test['Outlet_Size'] = test['Outlet_Size'].map({'Small'  : 1,
                                               'Medium' : 2,
                                               'High'   : 3
                                              }).astype(int)
print("Encoding Outlet Size for:")
print("Train Data:")
display(train)
print()
```

```
Encoding Outlet Size for:
Train Data:
```

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

8193 rows × 33 columns

```
Test Data:
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outle
0	FDW58	20.750000	Low Fat	0.007565	Snack Foods	107.8622	
1	FDW14	8.300000	Regular	0.038428	Dairy	87.3198	
2	NCN55	14.600000	Low Fat	0.099575	Others	241.7538	
3	FDQ58	7.315000	Low Fat	0.015388	Snack Foods	155.0340	
4	FDY38	12.857645	Regular	0.118599	Dairy	234.2300	
...	...	...	...	...	...	...	
5676	FDB58	10.500000	Regular	0.013496	Snack Foods	141.3154	
5677	FDD47	7.600000	Regular	0.142991	Starchy Foods	169.1448	
5678	NCO17	10.000000	Low Fat	0.073529	Health and Hygiene	118.7440	
5679	FDJ26	15.300000	Regular	0.000000	Canned	214.6218	
5680	FDU37	9.500000	Regular	0.104720	Canned	79.7960	

5681 rows × 11 columns

In [85]:
```
train['Outlet_Location_Type'] = train['Outlet_Location_Type'].str[-1:].astype
test['Outlet_Location_Type']  = test['Outlet_Location_Type'].str[-1:].astype(
print("Removing the word Tier and converting the digits to int for:")
print("Train data:")
display(train)
print()
```

```
Removing the word Tier and converting the digits to int for:
Train data:
```

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

8193 rows × 33 columns

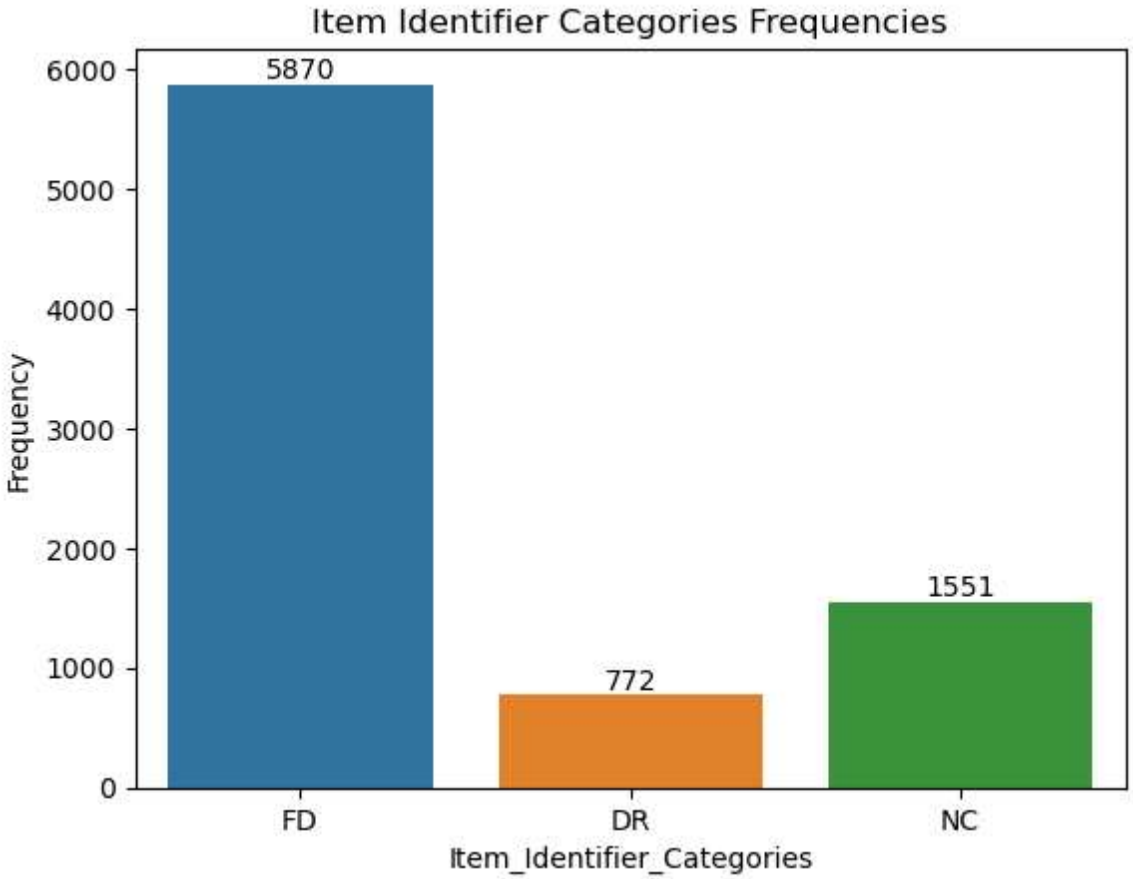
```
Test Data:
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outle
0	FDW58	20.750000	Low Fat	0.007565	Snack Foods	107.8622	
1	FDW14	8.300000	Regular	0.038428	Dairy	87.3198	
2	NCN55	14.600000	Low Fat	0.099575	Others	241.7538	
3	FDQ58	7.315000	Low Fat	0.015388	Snack Foods	155.0340	
4	FDY38	12.857645	Regular	0.118599	Dairy	234.2300	
...	...	...	...	...	...	...	
5676	FDB58	10.500000	Regular	0.013496	Snack Foods	141.3154	
5677	FDD47	7.600000	Regular	0.142991	Starchy Foods	169.1448	
5678	NCO17	10.000000	Low Fat	0.073529	Health and Hygiene	118.7440	
5679	FDJ26	15.300000	Regular	0.000000	Canned	214.6218	
5680	FDU37	9.500000	Regular	0.104720	Canned	79.7960	

5681 rows × 11 columns

In [86]:
```
print("Categorizing Item Identifier using the rule: FD-Food, NC-Non-Consumable
train['Item_Identifier_Categories'] = train['Item_Identifier'].str[0:2]
test['Item_Identifier_Categories']  = test['Item_Identifier'].str[0:2]
ax=sns.countplot(x="Item_Identifier_Categories", data=train)
for container in ax.containers:
    ax.bar_label(container, label_type="edge")
plt.xlabel("Item_Identifier_Categories")
plt.ylabel("Frequency")
```

```
Categorizing Item Identifier using the rule: FD-Food, NC-Non-Consumables, &
DR-Drinks.
```



In [87]:
```
print("Dropping the column Item Identifier:")
train.drop(columns="Item_Identifier",axis=1, inplace=True)
test.drop(columns="Item_Identifier",axis=1, inplace=True)
print("Train Data:")
display(train)
```

```
Dropping the column Item Identifier:
Train Data:
```

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

8193 rows × 33 columns

```
Test Data:
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Ou
0	20.750000	Low Fat	0.007565	Snack Foods	107.8622	OUT049	
1	8.300000	Regular	0.038428	Dairy	87.3198	OUT017	
2	14.600000	Low Fat	0.099575	Others	241.7538	OUT010	
3	7.315000	Low Fat	0.015388	Snack Foods	155.0340	OUT017	
4	12.857645	Regular	0.118599	Dairy	234.2300	OUT027	
...	...	...	...	...	...	...	
5676	10.500000	Regular	0.013496	Snack Foods	141.3154	OUT046	
5677	7.600000	Regular	0.142991	Starchy Foods	169.1448	OUT018	
5678	10.000000	Low Fat	0.073529	Health and Hygiene	118.7440	OUT045	
5679	15.300000	Regular	0.000000	Canned	214.6218	OUT017	
5680	9.500000	Regular	0.104720	Canned	79.7960	OUT045	

5681 rows × 11 columns

In [88]:
```
le=LabelEncoder()
ordinal_columns=["Item_Fat_Content", "Outlet_Type","Outlet_Location_Type"]
print("Label encoding the ordinal columns like:")
k=1
for column in ordinal_columns:
    print(f"{k}. {column}.")
    train[column]=le.fit_transform(train[column])
    test[column]=le.fit_transform(test[column])
    k+=1
print("After label Encoding:")
print("Train data:")
display(train)
print()
```

```
Label encoding the ordinal columns like:
1. Item_Fat_Content.
2. Outlet_Type.
3. Outlet_Location_Type.
After label Encoding:
Train data:
```

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

8193 rows × 33 columns

```
Test data:
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Ou
0	20.750000	0	0.007565	Snack Foods	107.8622	OUT049	
1	8.300000	1	0.038428	Dairy	87.3198	OUT017	
2	14.600000	0	0.099575	Others	241.7538	OUT010	
3	7.315000	0	0.015388	Snack Foods	155.0340	OUT017	
4	12.857645	1	0.118599	Dairy	234.2300	OUT027	
...	...	...	...	...	...	...	
5676	10.500000	1	0.013496	Snack Foods	141.3154	OUT046	
5677	7.600000	1	0.142991	Starchy Foods	169.1448	OUT018	
5678	10.000000	0	0.073529	Health and Hygiene	118.7440	OUT045	
5679	15.300000	1	0.000000	Canned	214.6218	OUT017	
5680	9.500000	1	0.104720	Canned	79.7960	OUT045	

5681 rows × 11 columns

In [89]:
```
non_ordinal_columns=["Item_Type","Item_Identifier_Categories","Outlet_Identif
train=pd.get_dummies(train, columns=non_ordinal_columns, drop_first=True, dty
test=pd.get_dummies(test, columns=non_ordinal_columns, drop_first=True, dtype

print(f"One hot encoding the non-ordinal columns: {non_ordinal_columns}.")
print("For training data:")
display(train)
print()
```

```
One hot encoding the non-ordinal columns: ['Item_Type', 'Item_Identifier_Cat
egories', 'Outlet_Identifier'].
For training data:
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	Outlet_Location_Ty
0	9.300	0	0.016047	249.8092	2	
1	5.920	1	0.019278	48.2692	2	
2	17.500	0	0.016760	141.6180	2	
3	19.200	1	0.000000	182.0950	2	
4	8.930	0	0.000000	53.8614	3	
...	...	...	...	...	...	
8518	6.865	0	0.056783	214.5218	3	
8519	8.380	1	0.046982	108.1570	2	
8520	10.600	0	0.035186	85.1224	1	
8521	7.210	1	0.145221	103.1332	2	
8522	14.800	0	0.044878	75.4670	1	

8193 rows × 56 columns

```
For testing data:
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	Outlet_Location_Ty
0	20.750000	0	0.007565	107.8622	2	
1	8.300000	1	0.038428	87.3198	2	
2	14.600000	0	0.099575	241.7538	2	
3	7.315000	0	0.015388	155.0340	2	
4	12.857645	1	0.118599	234.2300	2	
...	...	...	...	...	...	
5676	10.500000	1	0.013496	141.3154	1	
5677	7.600000	1	0.142991	169.1448	2	
5678	10.000000	0	0.073529	118.7440	2	
5679	15.300000	1	0.000000	214.6218	2	
5680	9.500000	1	0.104720	79.7960	2	

5681 rows × 34 columns

In [90]:
```
train.to_csv("train_processed.csv")
```

Saving the train and test DF in the form of CSV file.

In [91]:
```
import os

# Define the paths for train and test data
train_paths = ["C:/Users/HP/Downloads/train_processed.csv", "C:/Users/HP/Down
test_paths = ["C:/Users/HP/Downloads/test_possessed.csv", "C:/Users/HP/Downlo

# Find the existing path for train data
train_path = next((path for path in train_paths if os.path.exists(path)), Non

# Print the result
if train_path:
    print(f"Train data extracted from {train_path}.")
else:
    print("Train data path not found.")

# Find the existing path for test data
test_path = next((path for path in test_paths if os.path.exists(path)), None)

# Print the result
if test_path:
    print(f"Test data extracted from {test_path}.")
else:
```

```
Train data extracted from C:/Users/HP/Downloads/train_processed.csv.
Test data extracted from C:/Users/HP/Downloads/test_possessed.csv.
```

In [92]:
```
train=pd.read_csv(train_path)
test=pd.read_csv(test_path)
print("Extracted datas:")
print("Train Data:")
display(train)
print()
```

```
Extracted datas:
Train Data:
```

	Unnamed: 0	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	Outlet_
0	0	9.300	0	0.016047	249.8092	2	
1	1	5.920	1	0.019278	48.2692	2	
2	2	17.500	0	0.016760	141.6180	2	
3	3	19.200	1	0.000000	182.0950	2	
4	4	8.930	0	0.000000	53.8614	3	
...	...	...	...	...	...	...	
8188	8518	6.865	0	0.056783	214.5218	3	
8189	8519	8.380	1	0.046982	108.1570	2	
8190	8520	10.600	0	0.035186	85.1224	1	
8191	8521	7.210	1	0.145221	103.1332	2	
8192	8522	14.800	0	0.044878	75.4670	1	

8193 rows × 41 columns

```
Test Data:
```

	Unnamed: 0	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	Outlet_
0	0	20.750000	0	0.007565	107.8622	2	
1	1	8.300000	1	0.038428	87.3198	2	
2	2	14.600000	0	0.099575	241.7538	2	
3	3	7.315000	0	0.015388	155.0340	2	
4	4	12.857645	1	0.118599	234.2300	2	
...	...	...	...	...	...	...	
5676	5676	10.500000	1	0.013496	141.3154	1	
5677	5677	7.600000	1	0.142991	169.1448	2	
5678	5678	10.000000	0	0.073529	118.7440	2	
5679	5679	15.300000	1	0.000000	214.6218	2	
5680	5680	9.500000	1	0.104720	79.7960	2	

5681 rows × 35 columns

In [93]:
```
train.drop(columns="Unnamed: 0", axis=1,inplace=True)
test.drop(columns="Unnamed: 0", axis=1, inplace=True)
print("Train Data:")
display(train)
print()
```

```
Train Data:
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	Outlet_Location_Ty
0	9.300	0	0.016047	249.8092	2	
1	5.920	1	0.019278	48.2692	2	
2	17.500	0	0.016760	141.6180	2	
3	19.200	1	0.000000	182.0950	2	
4	8.930	0	0.000000	53.8614	3	
...	...	...	...	...	...	
8188	6.865	0	0.056783	214.5218	3	
8189	8.380	1	0.046982	108.1570	2	
8190	10.600	0	0.035186	85.1224	1	
8191	7.210	1	0.145221	103.1332	2	
8192	14.800	0	0.044878	75.4670	1	

8193 rows × 40 columns

```
Test Data:
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	Outlet_Location_Ty
0	20.750000	0	0.007565	107.8622	2	
1	8.300000	1	0.038428	87.3198	2	
2	14.600000	0	0.099575	241.7538	2	
3	7.315000	0	0.015388	155.0340	2	
4	12.857645	1	0.118599	234.2300	2	
...	...	...	...	...	...	
5676	10.500000	1	0.013496	141.3154	1	
5677	7.600000	1	0.142991	169.1448	2	
5678	10.000000	0	0.073529	118.7440	2	
5679	15.300000	1	0.000000	214.6218	2	
5680	9.500000	1	0.104720	79.7960	2	

5681 rows × 34 columns

In [94]:
```
X=train.drop("Item_Outlet_Sales", axis=1)
y=train["Item_Outlet_Sales"]
print("Inputs:")
display(X)
print()
```

```
Inputs:
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_MRP	Outlet_Size	Outlet_Location
0	9.300	0	0.016047	249.8092	2	
1	5.920	1	0.019278	48.2692	2	
2	17.500	0	0.016760	141.6180	2	
3	19.200	1	0.000000	182.0950	2	
4	8.930	0	0.000000	53.8614	3	
...	...	...	...	...	...	
8188	6.865	0	0.056783	214.5218	3	
8189	8.380	1	0.046982	108.1570	2	
8190	10.600	0	0.035186	85.1224	1	

In [95]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.2)
```

```
Shape of the X_train, X_test, y_train, & y_test:
(6554, 39) (1639, 39) (6554,) (1639,)
```

In [96]:
```
lr=LinearRegression().fit(X_train, y_train)
lr_pred = lr.predict(X_test)
print(f"Applying the model:")
display(lr)
```

```
Applying the model:
```

LinearRegression

LinearRegression()

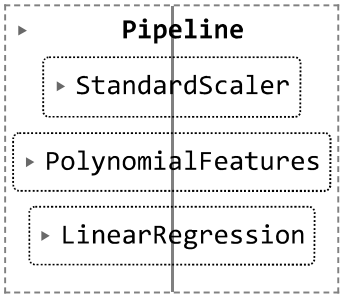
```
Training score  : 0.9237570503340224
Test score      : 0.9234554256677513
```

In [97]:
```
lr_mse  = mean_squared_error(y_test , lr_pred)
lr_rmse = math.sqrt(lr_mse)
lr_r2   = r2_score(y_test, lr_pred)
print(f"Evaluation of {lr} model.")
```

```
Evaluation of LinearRegression() model.
RMSE: 419.989534608304
R2 Score: 0.9234554256677513
```

In [98]:
```
print("Applying Standard Scaler, Polynomial Features, and  ")
steps = [
    ('scaler', StandardScaler()),
    ('poly',   PolynomialFeatures(degree=2)),
    ('model',  LinearRegression())
       ]

lr_pipeline = Pipeline(steps)
print("Model Used:")
display(lr_pipeline)
lr_pipeline.fit(X_train, y_train)
```

```
Applying Standard Scaler, Polynomial Features, and
Model Used:
```



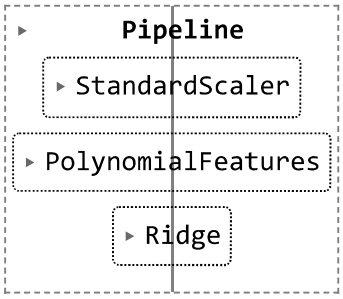

```
Training score  : 0.9903309435953069
Test score      : 0.9896440662197595
```

In [99]:
```
lr_pipeline_pred = lr_pipeline.predict(X_test)
lr_pipeline_mse  = mean_squared_error(y_test , lr_pipeline_pred)
lr_pipeline_rmse = math.sqrt(lr_pipeline_mse)
lr_pipeline_r2   = r2_score(y_test, lr_pipeline_pred)
print(f"Evaluation of {lr_pipeline} model.")
```

```
Evaluation of Pipeline(steps=[('scaler', StandardScaler()), ('poly', Polynom
ialFeatures()),
                ('model', LinearRegression())]) model.
RMSE: 154.48129268406376
R2 Score: 0.9896440662197595
```

In [100]:
```
print("Applying ridge regression:")
steps=[('scaler', StandardScaler()),('poly', PolynomialFeatures(degree=2)),
       ('model', Ridge(alpha=7, fit_intercept=True))]
ridge_pipeline=Pipeline(steps)
ridge_pipeline.fit(X_train, y_train)
print("Model used:")
display(ridge_pipeline)
train_score=ridge_pipeline.score(X_train, y_train)
test_score=ridge_pipeline.score(X_test, y_test)
```

```
Applying ridge regression:
Model used:
```



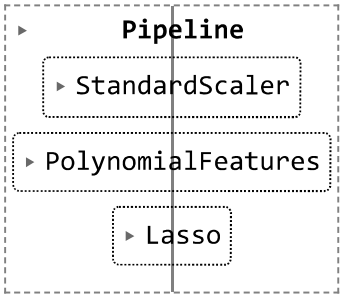

```
Training Score: 0.9902712912646704.
Testing Score: 0.9895915741737631.
```

In [101]:
```
ridge_predictions = ridge_pipeline.predict(X_test)
ridge_mse  = mean_squared_error(y_test , ridge_predictions)
ridge_rmse = math.sqrt(ridge_mse)
ridge_r2   = r2_score(y_test, ridge_predictions)
print(f"Evaluation of {ridge_pipeline} model.")
```

```
Evaluation of Pipeline(steps=[('scaler', StandardScaler()), ('poly', Polynom
ialFeatures()),
                ('model', Ridge(alpha=7))]) model.
RMSE: 154.87231436791294
R2 Score: 0.9895915741737631
```

In [102]:
```
print("Apply Lasso regression:")
steps = [
            ('scaler', StandardScaler()),
            ('poly', PolynomialFeatures(degree=2)),
            ('model', Lasso(alpha=0.2, fit_intercept=True))
        ]

lasso_pipeline = Pipeline(steps)
print("Model used:")
display(lasso_pipeline)
lasso_pipeline.fit(X_train, y_train)
```

```
Apply Lasso regression:
Model used:
```



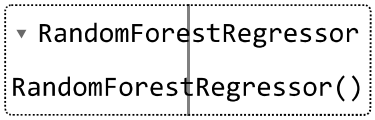

```
Training score  : 0.9903036448601646
Test score      : 0.9897481182850665
```

In [103]:
```
lasso_predictions = lasso_pipeline.predict(X_test)
lasso_mse  = mean_squared_error(y_test , lasso_predictions)
lasso_rmse = math.sqrt(lasso_mse)
lasso_r2   = r2_score(y_test, lasso_predictions)
print(f"Evaluation of {lasso_pipeline} model.")
```

```
Evaluation of Pipeline(steps=[('scaler', StandardScaler()), ('poly', Polynom
ialFeatures()),
                ('model', Lasso(alpha=0.2))]) model.
RMSE: 153.70325187843284
R2 Score: 0.9897481182850665
```

In [104]:
```
print("Applying Random Forest Regressor:")
rand_forest_model = RandomForestRegressor()
rand_forest_model.fit(X_train, y_train)
print("Model used:")
display(rand_forest_model)
```

```
Applying Random Forest Regressor:
Model used:
```

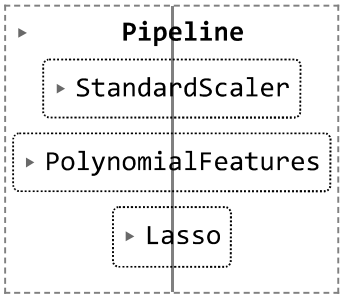



```
Training score  : 0.9999028537116261
Test score      : 0.9994340677187704
```

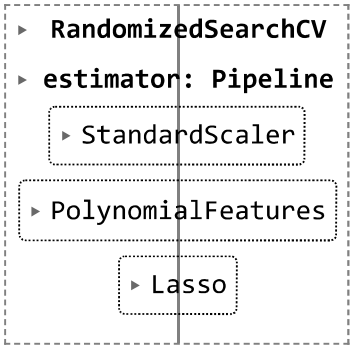
In [105]:

```
param_dist = {
    'model__alpha': [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
}

steps = [
    ('scaler', StandardScaler()),
    ('poly', PolynomialFeatures(degree=2)),
    ('model', Lasso())
]
print("Steps Involved:")
display(steps)

lasso_pipeline = Pipeline(steps)
print("Base model used:")
display(lasso_pipeline)

random_search = RandomizedSearchCV(
    lasso_pipeline, param_distributions=param_dist, n_iter=10, cv=5, n_jobs=-1

print("Model used:")
display(random_search)

random_search.fit(X_train, y_train)

print("Best hyperparameters found:")
print(random_search.best_params_)

print("Best score found: {:.4f}".format(random_search.best_score_))

best_lasso_model = random_search.best_estimator_
print("Best model:")
display(best_lasso_model)

test_score = best_lasso_model.score(X_test, y_test)
```

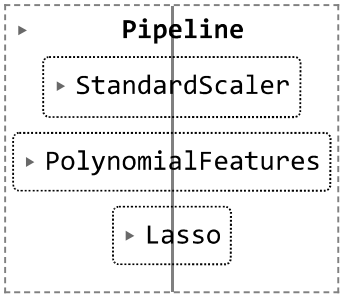
```
Steps Involved:

[('scaler', StandardScaler()),
 ('poly', PolynomialFeatures()),
 ('model', Lasso())]

Base model used:
```




```
Model used:
```




```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best hyperparameters found:
{'model__alpha': 1.0}
Best score found: 0.9894
Best model:
```

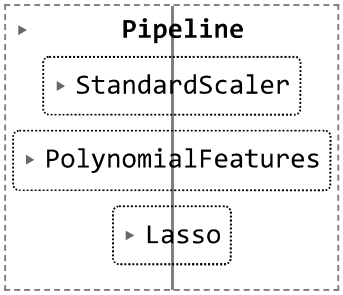



```
Test score for the best model: 0.9899
```

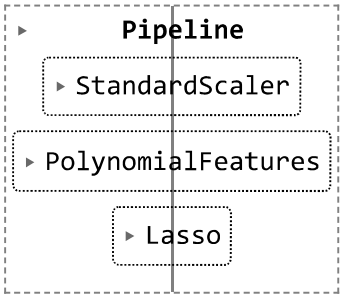
In [106]:

```
param_dist = {
    'model__alpha': [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
}

# Define the pipeline steps
steps = [
    ('scaler', StandardScaler()),
    ('poly', PolynomialFeatures(degree=2)),
    ('model', Lasso(alpha=0.2, fit_intercept=True))
]

print("Steps Involved:")
display(steps)


lasso_pipeline = Pipeline(steps)

print("Model used:")
display(lasso_pipeline)


lasso_pipeline.fit(X_train, y_train)
print('Training score: {:.4f}'.format(lasso_pipeline.score(X_train, y_train)))
print('Test score: {:.4f}'.format(lasso_pipeline.score(X_test, y_test)))


lasso_pipeline = Pipeline(steps)

print("Base model used:")
display(lasso_pipeline)


random_search = RandomizedSearchCV(
    lasso_pipeline, param_distributions=param_dist, n_iter=10, cv=5, n_jobs=-

print("Model used:")
display(random_search)

random_search.fit(X_train, y_train)

print("Best hyperparameters found:")
print(random_search.best_params_)

print("Best score found: {:.4f}".format(random_search.best_score_))

best_lasso_model = random_search.best_estimator_
print("Best model:")
display(best_lasso_model)
```

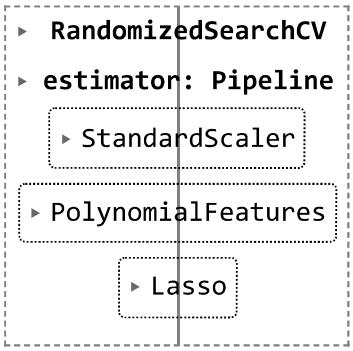
```
Steps Involved:

[('scaler', StandardScaler()),
 ('poly', PolynomialFeatures()),
 ('model', Lasso(alpha=0.2))]

Model used:
```



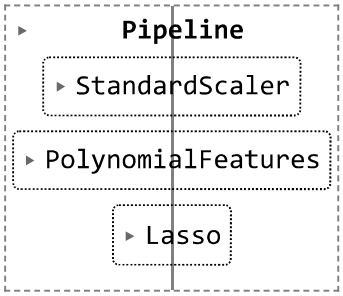

```
Training score: 0.9903
Test score: 0.9897
Base model used:
```




```
Model used:
```




```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
Best hyperparameters found:
{'model__alpha': 1.0}
Best score found: 0.9894
Best model:
```

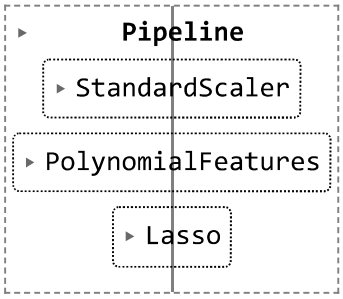



```
Test score for the best model: 0.9899
```

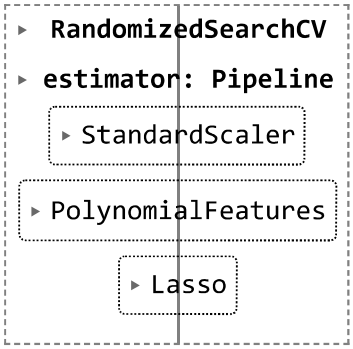
In [*]:

```
param_dist = {
    'model__alpha': [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
}

steps = [
    ('scaler', StandardScaler()),
    ('poly', PolynomialFeatures(degree=2)),
    ('model', Lasso())
]
print("Steps Involved:")
display(steps)

lasso_pipeline = Pipeline(steps)
print("Base model used:")
display(lasso_pipeline)

random_search = RandomizedSearchCV(
    lasso_pipeline, param_distributions=param_dist, n_iter=10, cv=5, n_jobs=-1

print("Model used:")
display(random_search)

random_search.fit(X_train, y_train)

print("Best hyperparameters found:")
print(random_search.best_params_)

print("Best score found: {:.4f}".format(random_search.best_score_))

best_lasso_model = random_search.best_estimator_
print("Best model:")
display(best_lasso_model)

test_score = best_lasso_model.score(X_test, y_test)
```

```
Steps Involved:

[('scaler', StandardScaler()),
 ('poly', PolynomialFeatures()),
 ('model', Lasso())]

Base model used:
```




```
Model used:
```




```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

In [ ]: