Data Analysis for BIGMART_SALES_DATASET. A BigMart Store want to determine the highest Item_Outlet_Sales with the Item_Type that are most sold and determine the Outlet_Location with highest distribution score then build a model using Linear Regression model to predict Item_Outlet_Sales.

DataSet Overview

BIGMART_SALES_DATASET.

All data come from one source which was csv file shared and contains details in columns as following.

* index: just index.

* Item_Identifier: Item Identifier Tag to locate Item.

* Item_Weight: The weight values of each items.

* Item_Fat_Content: The fat content information of each content.

* Item_Visibility: The visibility score of each item.

* Item_Type: Information about the Item Type.

* Item_MRP: Item market price informations.

* Outlet_Identifier: Outlet Identifier Tag for sorting items.

* Outlet_Establishment_Year: The year that the Outlet Establishment for item store.

* Outlet_Size: The size of the Outlet.

* Outlet_Location_Type: The location of each outlet store.

* Outlet_Type: The type of the outlet store.

* Item_Outlet_Sales: The outlet sales of each outlet store.

The dataset, named "BIGMART_SALES_DATASET," was collected from Kaggle and is available at the following URL: https://www.kaggle.com/datasets/brijbhushannanda1979/bigmart-sales-data. It consists of a CSV file containing information about various items sold in BigMart stores. The dataset includes details such as item weight, fat content, visibility, type, market price, outlet details, establishment year, size, location, type, and outlet sales.

Researchers and data enthusiasts can use this dataset for analyzing sales trends and building predictive models in the retail domain.

Table of Contents.

Tasks.

We will go through several tasks to archieve our goals and purposes:

Task 9: Checking for NAN/NULL values(missing values) for test dataset

Task 10: To get the details of train data colums nature information

Task 11: To get the details of test data colums nature information

Task 12: To get the full descriptive statistics chart table for train dataset with missing values

Task 13: To clean the data, i have to display the missing values colums for train dataset

Task 14: Descripyive analysis of 'Item_Weight', to get insights from the data, most important is to get the mean value of 'Item_Weight' so that we can use it to fill the missing value in 'Item_Weight' colum

Task 15: Computing mean value Item_Weight to the missing values in train and test datasets

Task 16: To check for null values in Item_Weight for train dataset

Task 17: To check for null values in Item_Weight for test dataset

Task 18: Descripyive analysis of 'Item_Weight', to get insights from the data.

Task 19: Descripyive analysis of 'Outlet_Size', to get insights from the data, most important is to get the mode value of 'Outlet_Size' so that we can use it to fill the missing value in 'Outlet_Size' colum for train data

Task 20: Descripyive analysis of 'Outlet_Size', to get insights from the data, most important is to get the mode value of 'Outlet_Size' so that we can use it to fill the missing value in 'Outlet_Size' colum for test data

Task 21: mode inputation on Outlet_Size colum to replace missing for train data

Task 22: mode inputation on Outlet_Size colum to replace missing for test data

Task 23: inserting the mode for train and test datasets

Task 24: checking for null value for train data

Task 25: checking for null value for test data

Task 26: Selecting features based on purpose or aim of the analysis, i have to drop Item_Identifier','Outlet_Identifier' colums because i don`t need them for my analysis

Task 27: Getting the details of 'Outlet_Establishment_Year' colum to obtain to 'Outlet_Establishment_Year'score

Task 28: Descriptive analysis of the train data

Task 29: Plotting the line graph distributions of the 'Item_Outlet_Sales' to obtain the visualization of the 'Item_Outlet_Sales' colum

Task 30: Graph of 'Outlet_Establishment_Year' distribution

Task 31: Plotting the scatter graph of 'Item_Outlet_Sales' and 'Outlet_Location_Type' to obtain the 'Outlet_Location_Type' that generated the highest sales

Task 32: Plotting the graph of 'Outlet_Type' to obtain the 'Outlet_Type' that generated highest sales or 'Outlet_Type' that has the major distribution score

Task 33: Plotting the graph of 'Outlet_Location_Type' colum

Task 34: Plotting the graph of 'Outlet_Size' to obtaing the 'Outlet_Size' that has the highest distribution

Task 35: Plotting the graph of 'Item_Fat_Content' to determine the 'Item_Fat_Content' that has the highest distributions

Task 36: Plotting 'Item_Visibility' graph

Task 37: Plotting the graph of Item_MRP(Item Market Rate Value)

Task 38: Plotting the graph of 'Item_Weight' to obtain the point where major distributions happenend in accordiance with the 'Item_Weight'.

Task 39: Generating a visualization of the number and frequency of categorical features

Task 40: Getting the correlation matrix of the train data

Task 41: Getting the numeric plot for every numeric distribution

Task 42: Checking for missing values

Task 43: Cleaning the train data performs data cleaning (drop duplicates & empty rows/cols)

Task 44: converts existing to more efficient dtypes, also called inside data_cleaning

Task 45: I am using Linear Regression Model, so i convert all the categorical values to numerical values

Task 46: I split my data in training and testing by using 20% for testing and 80% for training

Task 47: Standardizing the values of the train data so that i can use it to build my model

Task 48: Building my model by using Linear Regression mode

Task 49: To get the r2 score, mean absolute error score and mean square error score for the model report.

Task1.

Hypothesis Statement:

In the context of the BigMart Sales Dataset, we hypothesize that there exists a relationship between Item_Outlet_Sales and certain key factors within the dataset. Our primary focus is to determine the highest Item_Outlet_Sales, with a specific emphasis on identifying the Item_Type that experiences the highest sales volume. Additionally, we aim to investigate the Outlet_Location that contributes the highest distribution score.

We hypothesize that certain Item_Types play a crucial role in driving sales, and understanding their impact will enable BigMart to strategically manage inventory and marketing efforts. Furthermore, we anticipate that specific Outlet_Locations will exhibit a higher distribution score, providing insights into the most lucrative regions for the store.

To validate and quantify these hypotheses, we propose building a Linear Regression model. The model will be trained on relevant features to predict Item_Outlet_Sales, facilitating a quantitative analysis of the relationships identified in our initial hypotheses. By leveraging statistical and machine learning techniques, we aim to provide actionable insights that can inform strategic decision-making for optimizing sales and distribution strategies at BigMart.

Task 2.

Importing libraries.

```
[ ]  import pandas as pd
     import numpy as np
     %matplotlib inline
     import matplotlib.pyplot as plt
     import seaborn as sns
```

Task 3.

loading data for google colab.

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive

[ ] df_train= pd.read_csv(r'/content/drive/MyDrive/BIGMART_SALES_DATASET/Train.csv')
    df_test= pd.read_csv(r'/content/drive/MyDrive/BIGMART_SALES_DATASET/Test.csv')
```

Task 4.

Viewing the data, the head view of the 1st 5 rows of the train data.

```
[ ] df_train.head()
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Ou
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store	
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	

Task 5.

Viewing the data, the head view of the 1st 5 rows of the test data

```
[ ] df_test.head()
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type
0	FDW58	20.750	Low Fat	0.007565	Snack Foods	107.8622	OUT049	1999	Medium	Tier 1	Supermarket Type1
1	FDW14	8.300	reg	0.038428	Dairy	87.3198	OUT017	2007	NaN	Tier 2	Supermarket Type1
2	NCN55	14.600	Low Fat	0.099575	Others	241.7538	OUT010	1998	NaN	Tier 3	Grocery Store
3	FDQ58	7.315	Low Fat	0.015388	Snack Foods	155.0340	OUT017	2007	NaN	Tier 2	Supermarket Type1
4	FDY38	NaN	Regular	0.118599	Dairy	234.2300	OUT027	1985	Medium	Tier 3	Supermarket Type3

Task 6.

To view the shape of the train data.

```
[ ] df_train.shape

    (8523, 12)
```

Train data contains 8523 files and 12 colums.

Task 7.

To view the shape of the test data.

```
[ ] df_test.shape

    (5681, 11)
```

Test data contains 5681 files and 11 colums.

Task 8.

Checking for NAN/NULL values(missing values) for train dataset.

```
[ ]  df_train.isnull().sum()

     Item_Identifier                 0
     Item_Weight                  1463
     Item_Fat_Content                0
     Item_Visibility                 0
     Item_Type                       0
     Item_MRP                        0
     Outlet_Identifier               0
     Outlet_Establishment_Year       0
     Outlet_Size                  2410
     Outlet_Location_Type            0
     Outlet_Type                     0
     Item_Outlet_Sales               0
     dtype: int64
```

Categorical Missing Values (Outlet_Size):

For the missing values in the Outlet_Size column, which is categorical, the mode method will be employed. The mode represents the most frequently occurring size among the outlets. Imputing missing values with the mode ensures that the imputed sizes align with the prevailing distribution, maintaining the integrity of the categorical feature.

Imputation Results:

Item_Weight: 1463 missing values imputed using the mean method.

Outlet_Size: 2410 missing values imputed using the mode method.

Rationale:

The mean and mode imputation methods were chosen due to their simplicity and effectiveness in handling missing values, especially in large datasets.

These methods provide a reasonable estimate of missing values without introducing significant bias.

Impact on Analysis:

Addressing missing values is pivotal for accurate analytics. The imputation ensures that the dataset remains comprehensive, allowing for robust statistical analysis, machine learning modeling, and overall reliable insights.

Conclusion:

By systematically handling missing values using appropriate imputation methods, the train dataset is now well-prepared for subsequent analysis. This meticulous approach enhances the dataset's quality, reinforcing the integrity of the findings and conclusions derived from the data.

This strategic handling of missing values aligns with best practices in data preprocessing, setting the foundation for a comprehensive and reliable data analytics process.

Task 9.

Checking for NAN/NULL values(missing values) for test dataset.

```
df_test.isnull().sum()
```

```
Item_Identifier                 0
Item_Weight                   976
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  1606
Outlet_Location_Type            0
Outlet_Type                     0
dtype: int64
```

There are 976 missing values in Item_Weight colume and 1606 missing values in Outlet_Size on test dataset respectively.

Task 10.

To get the details of train data colums nature information.

```
[ ]  df_train.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 8523 entries, 0 to 8522
     Data columns (total 12 columns):
      #   Column                     Non-Null Count  Dtype
     ---  ------                     --------------  -----
      0   Item_Identifier            8523 non-null   object
      1   Item_Weight                7060 non-null   float64
      2   Item_Fat_Content           8523 non-null   object
      3   Item_Visibility            8523 non-null   float64
      4   Item_Type                  8523 non-null   object
      5   Item_MRP                   8523 non-null   float64
      6   Outlet_Identifier          8523 non-null   object
      7   Outlet_Establishment_Year  8523 non-null   int64
      8   Outlet_Size                6113 non-null   object
      9   Outlet_Location_Type       8523 non-null   object
      10  Outlet_Type                8523 non-null   object
      11  Item_Outlet_Sales          8523 non-null   float64
     dtypes: float64(4), int64(1), object(7)
     memory usage: 799.2+ KB
```

There are 8523 entries, ranges from 0 to 8522 Data columns (total 12 columns) dtypes is float64(4), int64(1), object(7) in train dataset. I will convert the "Object" to "Categorical" for proper data cleaning order.

Task 11.

To get the details of test data colums nature information.

```
[ ]  df_test.info()

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

There are 5681 entries, ranges from 0 to 5680 Data columns (total 11 columns) dtypes is float64(3), int64(1), object(7) in test dataset. I will convert the "Object" to "Categorical" for proper data cleaning order.

Task 12.

To get the full descriptive statistics chart table for train dataset with missing values.

```
df_train.describe()
```

	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

We can see that max of Item_Visibility is 0.328391 and the min is 0.000000, total count is 7060.000000. Max of Item_Weight is 21.350000 and the min is 4.555000, total count is 8523.000000. Max of Item_MRP is 266.888400 and the min is 31.290000, total count is 8523.000000. Max of Outlet_Establishment_Year is 2009.000000 and the min is 1985.000000, total count is 8523.000000. Max of Item_Outlet_Sales is 13086.964800 and the min is 33.290000, total count is 8523.000000. And many more informations can be gotten from the chart.

Task 13.

To clean the data, i have to display the missing values colums for train dataset.

```
[ ]  df_train.isnull().sum()

     Item_Identifier                 0
     Item_Weight                  1463
     Item_Fat_Content                0
     Item_Visibility                 0
     Item_Type                       0
     Item_MRP                        0
     Outlet_Identifier               0
     Outlet_Establishment_Year       0
     Outlet_Size                  2410
     Outlet_Location_Type            0
     Outlet_Type                     0
     Item_Outlet_Sales               0
     dtype: int64
```

Item_Weight and Outlet_Size colums contains missing values.

Task 14.

Descripyive analysis of 'Item_Weight', to get insights from the data, most important is to get the mean value of 'Item_Weight' so that we can use it to fill the missing value in 'Item_Weight' colum.

```
[ ]  df_train['Item_Weight'].describe()

     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
```

Item_Weight is numerical column so i fill it with Mean Imputation. To remove null values in Item_Weight by computing the mean value of the column since Item_Weight is an numerical values.

Task 15.

Computing mean value Item_Weight to the missing values in train and test datasets.

```
[ ] df_train['Item_Weight'].fillna(df_train['Item_Weight'].mean(),inplace=True)
    df_test['Item_Weight'].fillna(df_test['Item_Weight'].mean(),inplace=True)
```

Task 16.

To check for null values in Item_Weight for train dataset.

```
[ ] df_train.isnull().sum()

    Item_Identifier                 0
    Item_Weight                     0
    Item_Fat_Content                0
    Item_Visibility                 0
    Item_Type                       0
    Item_MRP                        0
    Outlet_Identifier               0
    Outlet_Establishment_Year       0
    Outlet_Size                  2410
    Outlet_Location_Type            0
    Outlet_Type                     0
    Item_Outlet_Sales               0
    dtype: int64
```

we can observe that the missing values in 'Item_Weight' colum have been replaced with the mean value of 'Item_Weight'.

Task 17.

To check for null values in Item_Weight for test dataset.

```
df_test.isnull().sum()

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                  1606
Outlet_Location_Type            0
Outlet_Type                     0
dtype: int64
```

We can observe that all the missing values in Item_Weight colum in test dataset have been replaced with the mean of Item_Weight.

Task 18.

Descriptive analysis of 'Item_Weight', to get insights from the data.

```
[ ] df_train['Item_Weight'].describe()

count    8523.000000
mean       12.857645
std         4.226124
min         4.555000
25%         9.310000
50%        12.857645
75%        16.000000
max        21.350000
Name: Item_Weight, dtype: float64
```

We can see that the total count of Item_Weight has increase due to replacement of the missing values.

Outlet_Size colum contain missing values.

Task 19.

Descripyive analysis of 'Outlet_Size', to get insights from the data, most important is to get the mode value of 'Outlet_Size' so that we can use it to fill the missing value in 'Outlet_Size' colum.

```
[ ]  df_train['Outlet_Size'].value_counts()

     Medium    2793
     Small     2388
     High       932
     Name: Outlet_Size, dtype: int64
```

The most occuring number is 'Medium ', we will use it to fill the missing values of 'Outlet_Size' colum in train data.

Task 20.

Descripyive analysis of 'Outlet_Size', to get insights from the data, most important is to get the mode value of 'Outlet_Size' so that we can use it to fill the missing value in 'Outlet_Size' colum for test data.

```
[ ]  df_test['Outlet_Size'].value_counts()

     Medium    1862
     Small     1592
     High       621
     Name: Outlet_Size, dtype: int64
```

The most occuring number is 'Medium ', we will use it to fill the missing values of 'Outlet_Size' colum in test data.

Outlet_Size is catagorical column so we fill it with Mode Imputation.

Task 21.

mode inputation on Outlet_Size colum to replace missing for train data.

```
[ ] df_train['Outlet_Size'].mode()

    0    Medium
    Name: Outlet_Size, dtype: object
```

The mode is 'Medium'.

Task 22.

mode inputation on Outlet_Size colum to replace missing for test data.

```
[ ] df_test['Outlet_Size'].mode()

    0    Medium
    Name: Outlet_Size, dtype: object
```

The mode is 'Medium'.

Task 23.

Inserting the mode for train and test datasets.

```
[ ] df_train['Outlet_Size'].fillna(df_train['Outlet_Size'].mode()[0],inplace=True)
    df_test['Outlet_Size'].fillna(df_test['Outlet_Size'].mode()[0],inplace=True)
```

Task 24.

checking for null value for train data.

```
[ ]  df_train.isnull().sum()

     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
```

No null or missing value in train data.

Task 25.

checking for null value for test data.

```
[ ]  df_test.isnull().sum()

     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
```

No null or missing value. We can see that our data is clean from missing values. All the missing values have been filled up for both train and test dataset.

Task 26.

Selecting features based on purpose or aim of the analysis, i have to drop Item_Identifier','Outlet_Identifier' colums because i don`t need them for my analysis.

```
[ ] df_train.drop(['Item_Identifier','Outlet_Identifier'],axis=1,inplace=True)
    df_test.drop(['Item_Identifier','Outlet_Identifier'],axis=1,inplace=True)
```

```
[ ] df_train
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales
0	9.300	Low Fat	0.016047	Dairy	249.8092	1999	Medium	Tier 1	Supermarket Type1	3735.1380
1	5.920	Regular	0.019278	Soft Drinks	48.2692	2009	Medium	Tier 3	Supermarket Type2	443.4228
2	17.500	Low Fat	0.016760	Meat	141.6180	1999	Medium	Tier 1	Supermarket Type1	2097.2700
3	19.200	Regular	0.000000	Fruits and Vegetables	182.0950	1998	Medium	Tier 3	Grocery Store	732.3800
4	8.930	Low Fat	0.000000	Household	53.8614	1987	High	Tier 3	Supermarket Type1	994.7052
...	...	...	...	...	...	...	...	...	...	...
8518	6.865	Low Fat	0.056783	Snack Foods	214.5218	1987	High	Tier 3	Supermarket Type1	2778.3834
8519	8.380	Regular	0.046982	Baking Goods	108.1570	2002	Medium	Tier 2	Supermarket Type1	549.2850
8520	10.600	Low Fat	0.035186	Health and Hygiene	85.1224	2004	Small	Tier 2	Supermarket Type1	1193.1136
8521	7.210	Regular	0.145221	Snack Foods	103.1332	2009	Medium	Tier 3	Supermarket Type2	1845.5976
8522	14.800	Low Fat	0.044878	Soft Drinks	75.4670	1997	Small	Tier 1	Supermarket Type1	765.6700

8523 rows × 10 columns

Task 27.

Getting the details of 'Outlet_Establishment_Year' colum to obtain to 'Outlet_Establishment_Year'score.

```
[ ] df_train['Outlet_Establishment_Year'].value_counts()

    1985    1463
    1987     932
    1999     930
    1997     930
    2004     930
    2002     929
    2009     928
    2007     926
    1998     555
    Name: Outlet_Establishment_Year, dtype: int64
```

In our training dataset, the Outlet_Establishment_Year attribute indicates that the highest year of establishment for outlets is 1985 with 1463 outlets. This suggests that 1985 was the most common year for setting up outlets, providing insight into the distribution of establishment years in our data.

Task 28.

Descriptive analysis of the train data.

```
[ ] df_train.describe()
```

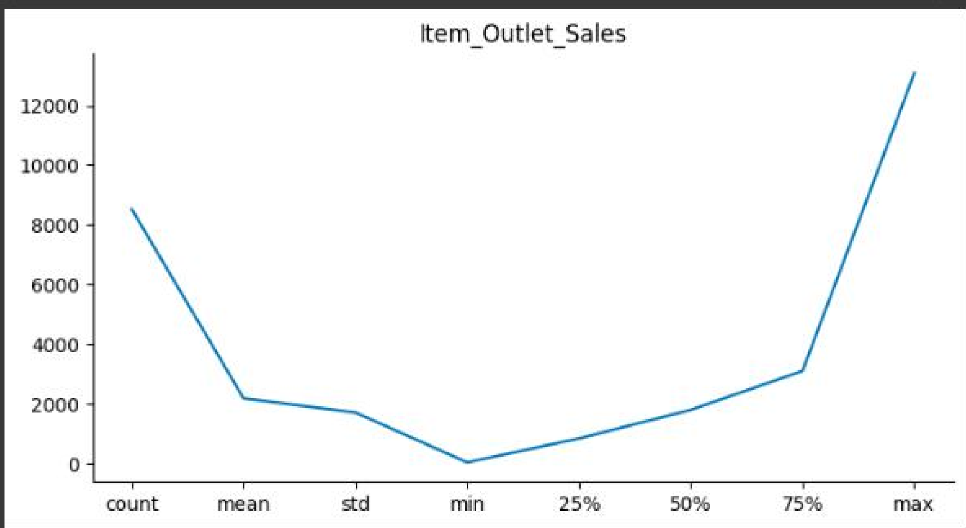
	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales
count	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000
mean	12.857646	1.369354	0.066132	7.226681	140.992767	1997.831867	1.170832	1.112871	1.201220	2181.288818
std	4.226124	0.644810	0.051598	4.209990	62.275066	8.371760	0.600327	0.812757	0.796459	1706.499634
min	4.555000	0.000000	0.000000	0.000000	31.290001	1985.000000	0.000000	0.000000	0.000000	33.290001
25%	9.310000	1.000000	0.026989	4.000000	93.826500	1987.000000	1.000000	0.000000	1.000000	834.247406
50%	12.857645	1.000000	0.053931	6.000000	143.012802	1999.000000	1.000000	1.000000	1.000000	1794.331055
75%	16.000000	2.000000	0.094585	10.000000	185.643700	2004.000000	2.000000	2.000000	1.000000	3101.296387
max	21.350000	4.000000	0.328391	15.000000	266.888397	2009.000000	2.000000	2.000000	3.000000	13086.964844

From the analysis view point, the Max of Item_Outlet_Sales is 13086.964844 while the Min of Item_Outlet_Sales is 33.290001, we can see the much difference between the maximum sales and the minimum sales, a wide difference when we consider the range.

Task 29.

Plotting the line graph distributions of the 'Item_Outlet_Sales' to obtain the visualization of the 'Item_Outlet_Sales' colum.

```
[ ] from matplotlib import pyplot as plt
    _df_38['Item_Outlet_Sales'].plot(kind='line', figsize=(8, 4), title='Item_Outlet_Sales')
    plt.gca().spines[['top', 'right']].set_visible(False)
```

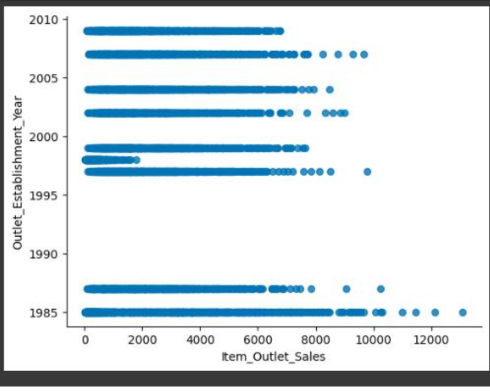


We can see the increase of the sales according to the 'Item_Outlet_Sales' line graph.

Task 30.

Graph of 'Outlet_Establishment_Year' distribution.

```
[ ] from matplotlib import pyplot as plt
    _df_9.plot(kind='scatter', x='Item_Outlet_Sales', y='Outlet_Establishment_Year', s=32, alpha=.8)
    plt.gca().spines[['top', 'right',]].set_visible(False)
```

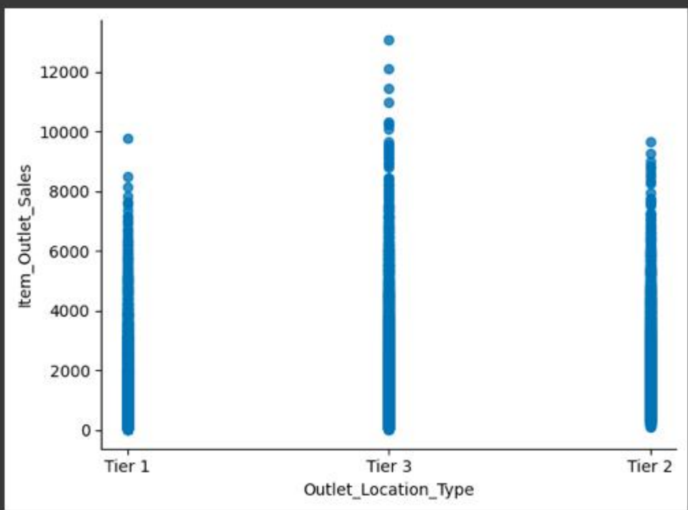


We can see that 1985 'Outlet_Establishment_Year' generated the highest distribution according to the 'Outlet_Establishment_Year' scatter distribution graph

Task 31.

Plotting the scatter graph of 'Item_Outlet_Sales' and 'Outlet_Location_Type' to obtain the 'Outlet_Location_Type' that generated the highest sales.

```
[ ] from matplotlib import pyplot as plt
    _df_9.plot(kind='scatter', x='Outlet_Location_Type', y='Item_Outlet_Sales', s=32, alpha=.8)
    plt.gca().spines[['top', 'right',]].set_visible(False)
```

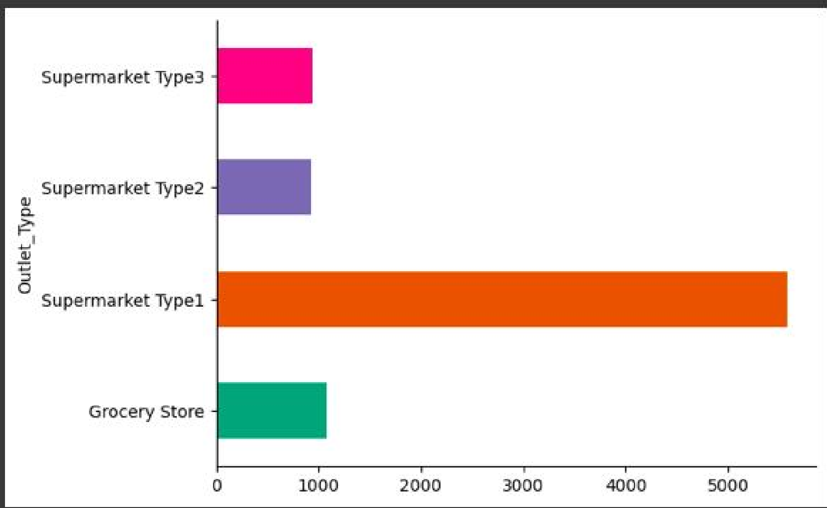


The scatter plot graph reveals distinct patterns in sales distribution across different Outlet_Location Tiers. Notably, Outlet_Location Tier3 stands out with the highest distribution, surpassing 12000 in sales. In contrast, Tier1 exhibits the least distribution, ranging from 8000 to 10000. This discrepancy highlights the significance of Tier3 locations in driving sales, potentially due to higher population density or increased customer demand. The data suggests that focusing on Outlet_Location Tier3 could be strategically advantageous for maximizing sales and catering to the majority of the distribution, reflecting the importance of geographical considerations in the retail landscape.

Task 32.

Plotting the graph of 'Outlet_Type' to obtain the 'Outlet_Type' that generated highest sales or 'Outlet_Type' that has the major distribution score.

```
[ ] from matplotlib import pyplot as plt
    import seaborn as sns
    _df_7.groupby('Outlet_Type').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
    plt.gca().spines[['top', 'right',]].set_visible(False)
```

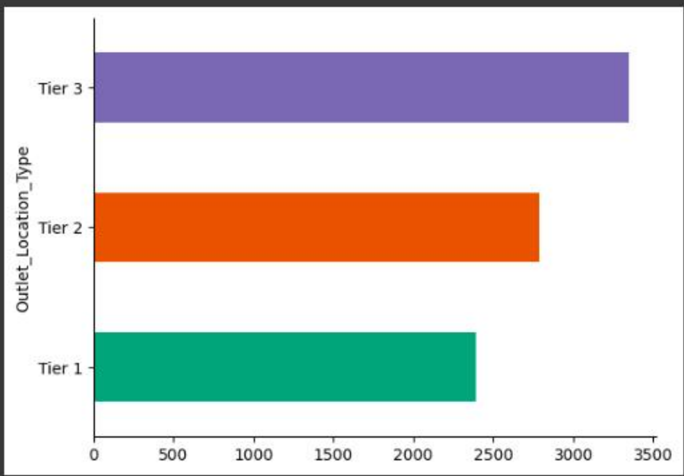


The sales distribution analysis reveals that Supermarket Type1 significantly outperforms other outlet types, reaching 5000 and above in sales. This stark difference in distribution is evident in the graph, where Supermarket Type1 dominates. The majority of sales are concentrated in this outlet type, indicating its popularity or strategic positioning. Understanding and leveraging the factors contributing to the success of Supermarket Type1 could provide valuable insights for optimizing sales strategies and potentially expanding the presence of similar outlet types. This underscores the importance of identifying and capitalizing on high-performing segments within the retail landscape for sustainable business growth.

Task 33.

Plotting the graph of 'Outlet_Location_Type' colum.

```
[ ] from matplotlib import pyplot as plt
    import seaborn as sns
    _df_6.groupby('Outlet_Location_Type').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
    plt.gca().spines[['top', 'right',]].set_visible(False)
```

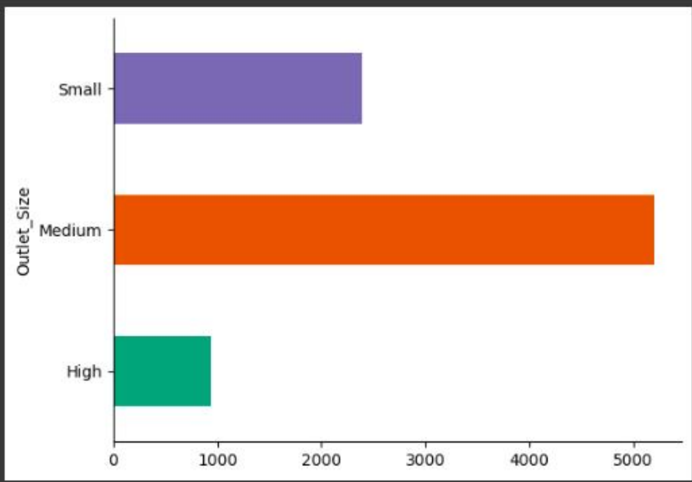


In the sales distribution analysis, Outlet_Location Tier3 emerges as the frontrunner, boasting 3000 and above in distribution, marking it as the highest-performing location. On the contrary, Tier1 lags behind with distributions of 2400 and less, ranking as the least among the outlet locations. This disparity in distribution underscores the significance of geographical placement and consumer demographics in shaping sales outcomes. Understanding the dynamics at play in Tier3, the top-performing location, could unveil valuable insights for optimizing strategies and potentially elevating the performance of Tier1 outlets. Identifying and addressing these location-specific nuances is crucial for strategic retail management and market penetration.

Task 34.

Plotting the graph of 'Outlet_Size' to obtaing the 'Outlet_Size' that has the highest distribution.

```
[ ]  from matplotlib import pyplot as plt
     import seaborn as sns
     _df_5.groupby('Outlet_Size').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
     plt.gca().spines[['top', 'right',]].set_visible(False)
```

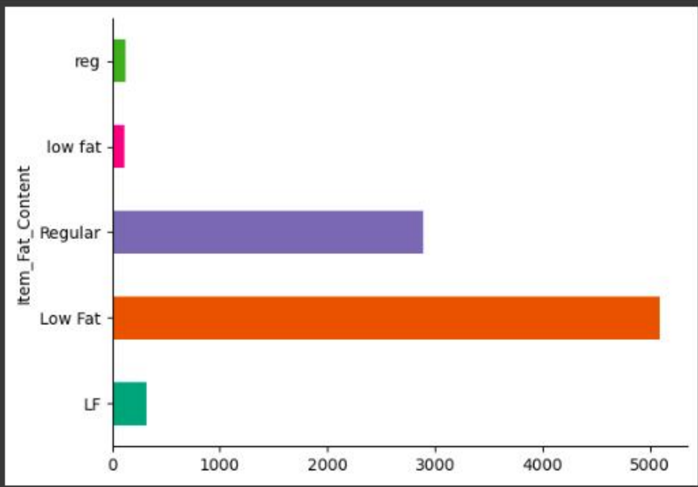


In the realm of outlet size distribution, Medium outlets shine as the top contenders, boasting a robust 5000 and above in distribution. This category outperforms others significantly, showcasing its prominence in the market. On the flip side, High outlets lag behind with meager distributions of 900 and less, marking them as the least distributed among outlet sizes. This disparity underscores the critical role of outlet size in influencing consumer engagement and sales. Recognizing the impact of size-based distinctions is pivotal for retail strategies, aiding in the development of targeted approaches to cater to diverse consumer preferences and optimize overall performance.

Task 35.

Plotting the graph of 'Item_Fat_Content' to determine the 'Item_Fat_Content' that has the highest distributions.

```
[ ] from matplotlib import pyplot as plt
    import seaborn as sns
    _df_4.groupby('Item_Fat_Content').size().plot(kind='barh', color=sns.palettes.mpl_palette('Dark2'))
    plt.gca().spines[['top', 'right',]].set_visible(False)
```

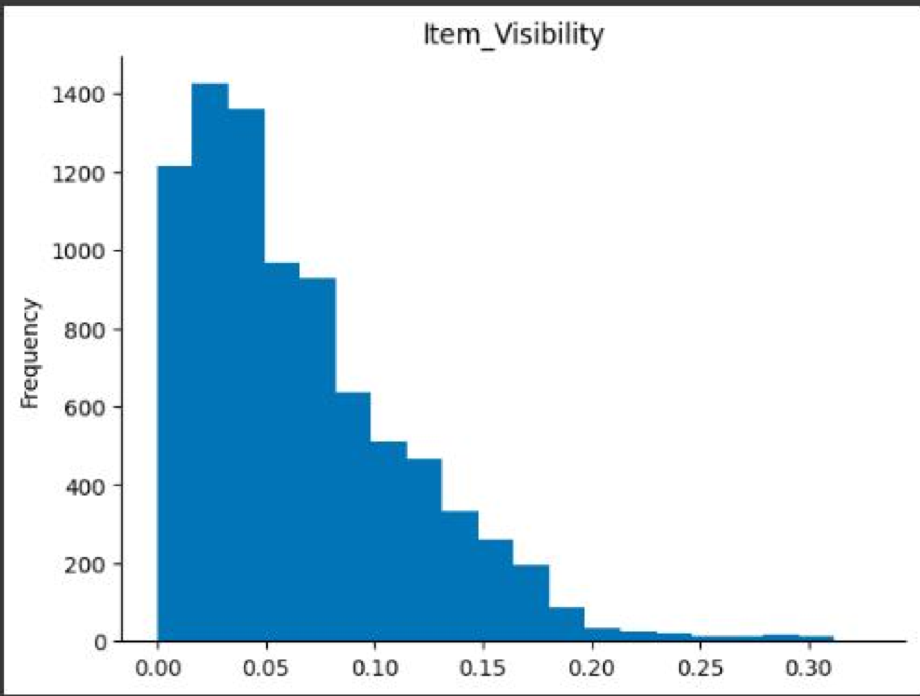


Low Fat items dominate the product landscape, emerging as the undisputed leaders with a distribution of 5000 and above, making them the most sought-after in the market. The consumer preference for Low Fat items is evident from this robust distribution, indicating a prevailing health-conscious trend among buyers. The emphasis on healthier choices aligns with the evolving lifestyles and wellness priorities of the population. Understanding and capitalizing on this preference for Low Fat items can empower retailers to tailor their product offerings, ensuring alignment with consumer demands and optimizing sales in this health-conscious era.

Task 36.

Plotting 'Item_Visibility' graph.

```
[ ] from matplotlib import pyplot as plt
    _df_2['Item_Visibility'].plot(kind='hist', bins=20, title='Item_Visibility')
    plt.gca().spines[['top', 'right',]].set_visible(False)
```

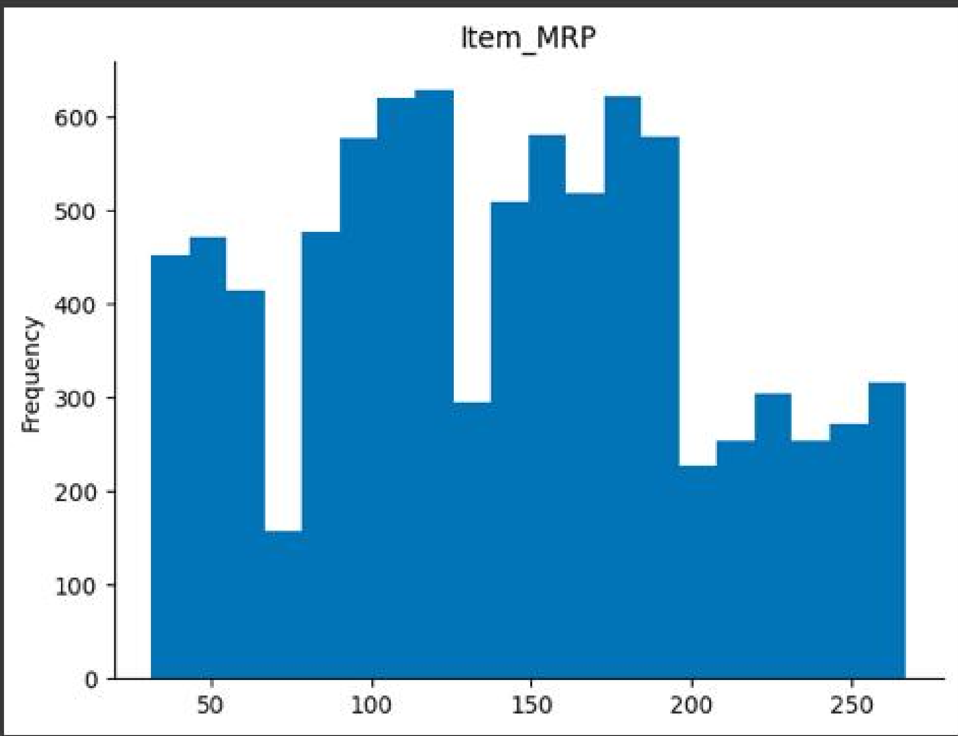


Analyzing the item_visibility graph reveals a fascinating trend where visibility rates ranging from 0.02 to 0.04 experience the highest distribution scores. This insight indicates that items falling within this visibility range are more prominently featured and, consequently, witness increased consumer engagement. However, as item_visibility rises beyond this range, the frequency of distribution shows a noticeable decline. This inverse relationship suggests that excessively visible items may not necessarily translate to higher distribution scores. Retailers can leverage this understanding to strategically manage product placement, ensuring optimal visibility within the sweet spot for increased sales and customer attention.

Task 37.

Plotting the graph of Item_MRP(Item Market Rate Value).

```
[ ] from matplotlib import pyplot as plt
    _df_3['Item_MRP'].plot(kind='hist', bins=20, title='Item_MRP')
    plt.gca().spines[['top', 'right',]].set_visible(False)
```

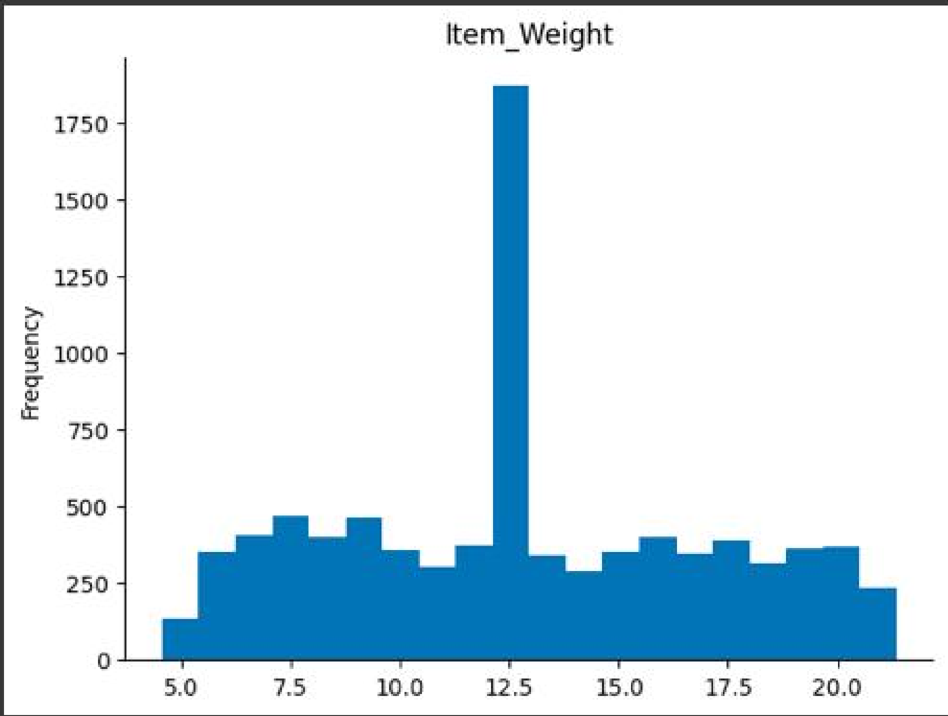


Examining the distribution graph of item_MRP (market rate price) reveals intriguing patterns. The highest MRP falls within the range of 100 to 199, suggesting a concentration of items in this price bracket. However, the distribution is not uniform or consistent; it fluctuates based on frequency. The price rates exhibit variability, with noticeable shifts in both upward and downward directions, as vividly illustrated in the graph. This inconsistency implies that consumers encounter diverse pricing across products, possibly influenced by factors such as brand positioning or promotional strategies. Retailers can strategically navigate this variance to optimize pricing strategies and enhance overall sales performance.

Task 38.

Plotting the graph of 'Item_Weight' to obtain the point where major distributions happenend in accordance with the 'Item_Weight'.

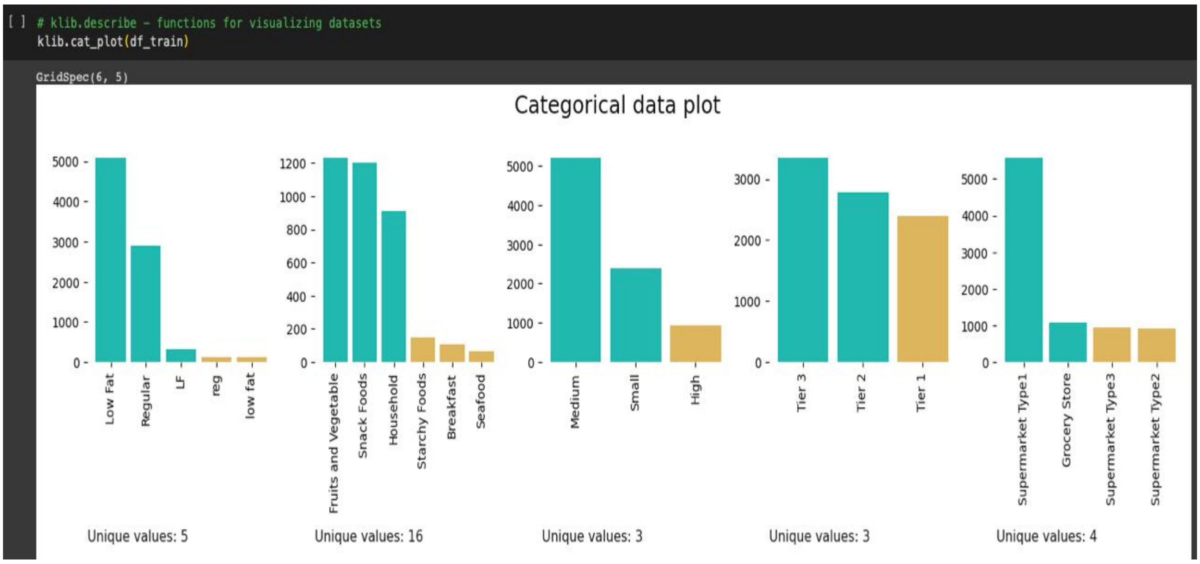
```
[ ] from matplotlib import pyplot as plt
    _df_1['Item_Weight'].plot(kind='hist', bins=20, title='Item_Weight')
    plt.gca().spines[['top', 'right',]].set_visible(False)
```



Examining the item_weight distribution graph highlights a distinct pattern, with the highest concentration observed at point 12.5. This particular weight value stands out prominently, showcasing a substantial departure from the distribution of other item weights. The graph clearly illustrates that the majority of items cluster around the point 12.5 region, suggesting a prevalence of items with this specific weight. Understanding this weight-related distribution can inform inventory management strategies, helping retailers prioritize and optimize the stocking of items at or around the pivotal weight value of 12.5 to meet consumer demand effectively.

Task 39.

Generating a visualization of the number and frequency of categorical features.



Analyzing the sales distribution graph reveals several key insights into product performance and consumer preferences. 'Low Fat' items dominate the market, with a significant sales volume of 5000, surpassing other categories. Within the 'Item_Type' category, 'Fruits and Vegetables' emerge as the top-selling items, recording a substantial value of 1200, while 'Seafood' represents the least popular category. Medium-sized outlets experience higher sales compared to other outlet sizes, while 'High' outlets trail with the least sales. In terms of location, Outlet_Location Tier3 commands the highest distribution, contrasting with Tier1, which exhibits the lowest distribution. Additionally, Supermarket Type1 leads in distribution, outperforming Supermarket Type2 with the least sales.

Task 40.

Getting the correlation matrix of the train data.

```
[ ] klib.corr_mat(df_train) # returns a color-encoded correlation matrix
```

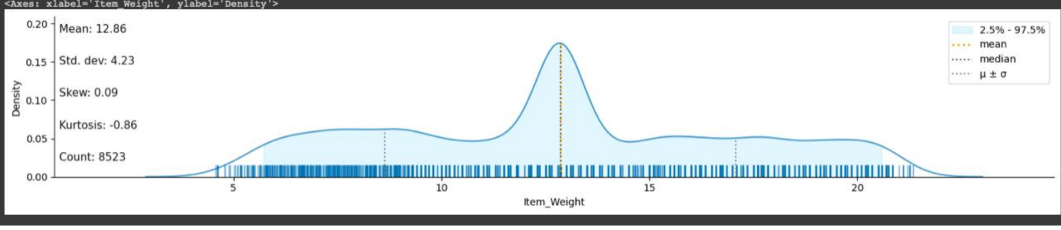
	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
Item_Weight	1.00	-0.01	0.02	-0.01	0.01
Item_Visibility	-0.01	1.00	-0.00	-0.07	-0.13
Item_MRP	0.02	-0.00	1.00	0.01	0.57
Outlet_Establishment_Year	-0.01	-0.07	0.01	1.00	-0.05
Item_Outlet_Sales	0.01	-0.13	0.57	-0.05	1.00

Task 41.

Getting the numeric plot for every numeric distribution.

```
[ ] klib.dist_plot(df_train) # returns a distribution plot for every numeric feature
```

<Axes: xlabel='Item_Weight', ylabel='Density'>



Task 42.

Checking for missing values.

```
[ ] klib.missingval_plot(df_train) # returns a figure containing information about missing values

    No missing values found in the dataset.
```

No missing value in the train data

No missing value in the train data.

Data Cleaning using Klib Library.

Task 43.

Cleaning the train data performs data cleaning (drop duplicates & empty rows/cols).

```
[ ] # klib.clean - functions for cleaning datasets
    klib.data_cleaning(df_train)
```

```
Shape of cleaned data: (8523, 10) - Remaining NAs: 0


Dropped rows: 0
     of which 0 duplicates. (Rows (first 150 shown): [])

Dropped columns: 0
     of which 0 single valued.     Columns: []
Dropped missing values: 0
Reduced memory by at least: 0.46 MB (-70.77%)
```

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_establishment_year	outlet_size	outlet_location_type	outlet_type	item_outlet_sales
0	9.300000	Low Fat	0.016047	Dairy	249.809204	1999	Medium	Tier 1	Supermarket Type1	3735.137939
1	5.920000	Regular	0.019278	Soft Drinks	48.269199	2009	Medium	Tier 3	Supermarket Type2	443.422791
2	17.500000	Low Fat	0.016760	Meat	141.617996	1999	Medium	Tier 1	Supermarket Type1	2097.270020
3	19.200001	Regular	0.000000	Fruits and Vegetables	182.095001	1998	Medium	Tier 3	Grocery Store	732.380005
4	8.930000	Low Fat	0.000000	Household	53.861401	1987	High	Tier 3	Supermarket Type1	994.705200
...	...	...	...	...	...	...	...	...	...	...
8518	6.865000	Low Fat	0.056783	Snack Foods	214.521805	1987	High	Tier 3	Supermarket Type1	2778.383301
8519	8.380000	Regular	0.046982	Baking Goods	108.156998	2002	Medium	Tier 2	Supermarket Type1	549.284973
8520	10.600000	Low Fat	0.035186	Health and Hygiene	85.122398	2004	Small	Tier 2	Supermarket Type1	1193.113647
8521	7.210000	Regular	0.145221	Snack Foods	103.133202	2009	Medium	Tier 3	Supermarket Type2	1845.597656
8522	14.800000	Low Fat	0.044878	Soft Drinks	75.467003	1997	Small	Tier 1	Supermarket Type1	765.669983

8523 rows × 10 columns

Cleaning the test data performs data cleaning (drop duplicates & empty rows/cols).

```
[ ] klib.data_cleaning(df_test)
```

```
Shape of cleaned data: (5681, 9) - Remaining NAs: 0


Dropped rows: 0
     of which 0 duplicates. (Rows (first 150 shown): [])

Dropped columns: 0
     of which 0 single valued.     Columns: []
Dropped missing values: 0
Reduced memory by at least: 0.29 MB (-74.36%)
```

	item_weight	item_fat_content	item_visibility	item_type	item_mrp	outlet_establishment_year	outlet_size	outlet_location_type	outlet_type
0	20.750000	Low Fat	0.007565	Snack Foods	107.862198	1999	Medium	Tier 1	Supermarket Type1
1	8.300000	reg	0.038428	Dairy	87.319801	2007	Medium	Tier 2	Supermarket Type1
2	14.600000	Low Fat	0.099575	Others	241.753799	1998	Medium	Tier 3	Grocery Store
3	7.315000	Low Fat	0.015388	Snack Foods	155.033997	2007	Medium	Tier 2	Supermarket Type1
4	12.695633	Regular	0.118599	Dairy	234.229996	1985	Medium	Tier 3	Supermarket Type3
...	...	...	...	...	...	...	...	...	...
5676	10.500000	Regular	0.013496	Snack Foods	141.315399	1997	Small	Tier 1	Supermarket Type1
5677	7.600000	Regular	0.142991	Starchy Foods	169.144806	2009	Medium	Tier 3	Supermarket Type2
5678	10.000000	Low Fat	0.073529	Health and Hygiene	118.744003	2002	Medium	Tier 2	Supermarket Type1
5679	15.300000	Regular	0.000000	Canned	214.621796	2007	Medium	Tier 2	Supermarket Type1
5680	9.500000	Regular	0.104720	Canned	79.795998	2002	Medium	Tier 2	Supermarket Type1

5681 rows × 9 columns

Our data is totally clean

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Weight                8523 non-null   float64
 1   Item_Fat_Content           8523 non-null   object
 2   Item_Visibility            8523 non-null   float64
 3   Item_Type                  8523 non-null   object
 4   Item_MRP                   8523 non-null   float64
 5   Outlet_Establishment_Year  8523 non-null   int64
 6   Outlet_Size                8523 non-null   object
 7   Outlet_Location_Type       8523 non-null   object
 8   Outlet_Type                8523 non-null   object
 9   Item_Outlet_Sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(5)
memory usage: 666.0+ KB
```

I converted datatype from object to category.

Task 44.

converts existing to more efficient dtypes, also called inside data_cleaning.

```
[ ] df_train=klib.convert_datatypes(df_train)
    df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Weight                8523 non-null   float32
 1   Item_Fat_Content           8523 non-null   category
 2   Item_Visibility            8523 non-null   float32
 3   Item_Type                  8523 non-null   category
 4   Item_MRP                   8523 non-null   float32
 5   Outlet_Establishment_Year  8523 non-null   int16
 6   Outlet_Size                8523 non-null   category
 7   Outlet_Location_Type       8523 non-null   category
 8   Outlet_Type                8523 non-null   category
 9   Item_Outlet_Sales          8523 non-null   float32
dtypes: category(5), float32(4), int16(1)
memory usage: 192.9 KB
```

Preprocessing Task before Model Building.

1) Label Encoding

Task 45.

I am using Linear Regression Model, so i convert all the categorical values to numerical values.

```
[ ] from sklearn.preprocessing import LabelEncoder
    le=LabelEncoder()
```

```
[ ] df_train['Item_Fat_Content']= le.fit_transform(df_train['Item_Fat_Content'])
    df_train['Item_Type']= le.fit_transform(df_train['Item_Type'])
    df_train['Outlet_Size']= le.fit_transform(df_train['Outlet_Size'])
    df_train['Outlet_Location_Type']= le.fit_transform(df_train['Outlet_Location_Type'])
    df_train['Outlet_Type']= le.fit_transform(df_train['Outlet_Type'])
```

```
[ ] df_train
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales
0	9.300000	1	0.016047	4	249.809204	1999	1	0	1	3735.137939
1	5.920000	2	0.019278	14	48.269199	2009	1	2	2	443.422791
2	17.500000	1	0.016760	10	141.617996	1999	1	0	1	2097.270020
3	19.200001	2	0.000000	6	182.095001	1998	1	2	0	732.380005
4	8.930000	1	0.000000	9	53.861401	1987	0	2	1	994.705200
...	...	...	...	...	...	...	...	...	...	...
8518	6.865000	1	0.056783	13	214.521805	1987	0	2	1	2778.383301
8519	8.380000	2	0.046982	0	108.156998	2002	1	1	1	549.284973
8520	10.600000	1	0.035186	8	85.122398	2004	2	1	1	1193.113647
8521	7.210000	2	0.145221	13	103.133202	2009	1	2	2	1845.597656
8522	14.800000	1	0.044878	14	75.467003	1997	2	0	1	765.669983

8523 rows × 10 columns

We can see that all the datas have been converted to numeric values so that i can be able to use it to build my model.

2) Splitting our data into train and test

Task 46.

I split my data in training and testing by using 20% for testing and 80% for training.

```
[ ] X=df_train.drop('Item_Outlet_Sales',axis=1)
```

```
[ ] Y=df_train['Item_Outlet_Sales']
```

```
[ ] from sklearn.model_selection import train_test_split

    X_train, X_test, Y_train, Y_test = train_test_split(X,Y, random_state=101, test_size=0.2)
```

3) Standarization

Task 47.

Standardizing the values of the train data so that i can use it to build my model.

```
[ ] X.describe()
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type
count	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000	8523.000000
mean	12.857646	1.369354	0.066132	7.226681	140.992767	1997.831867	1.170832	1.112871	1.201220
std	4.226124	0.644810	0.051598	4.209990	62.275066	8.371760	0.600327	0.812757	0.796459
min	4.555000	0.000000	0.000000	0.000000	31.290001	1985.000000	0.000000	0.000000	0.000000
25%	9.310000	1.000000	0.026989	4.000000	93.826500	1987.000000	1.000000	0.000000	1.000000
50%	12.857645	1.000000	0.053931	6.000000	143.012802	1999.000000	1.000000	1.000000	1.000000
75%	16.000000	2.000000	0.094585	10.000000	185.643700	2004.000000	2.000000	2.000000	1.000000
max	21.350000	4.000000	0.328391	15.000000	266.888397	2009.000000	2.000000	2.000000	3.000000

```
[ ] # standadizing the values
    from sklearn.preprocessing import StandardScaler
    sc= StandardScaler()
```

```
[ ] X_train_std= sc.fit_transform(X_train)
```

```
[ ] X_test_std= sc.transform(X_test)
```

```
[ ] X_train_std

    array([[ 1.52290023, -0.57382672,  0.68469731, ..., -1.95699503,
             1.08786619, -0.25964107],
           [-1.239856  , -0.57382672, -0.09514746, ..., -0.28872895,
            -0.13870429, -0.25964107],
           [ 1.54667619,  0.97378032, -0.0083859 , ..., -0.28872895,
            -0.13870429, -0.25964107],
           ...,
           [-0.08197109, -0.57382672, -0.91916229, ...,  1.37953713,
            -1.36527477, -0.25964107],
           [-0.74888436,  0.97378032,  1.21363045, ..., -0.28872895,
            -0.13870429, -0.25964107],
           [ 0.67885675, -0.57382672,  1.83915361, ..., -0.28872895,
             1.08786619,  0.98524841]])
```

X_train standardized values.

```
[ ] X_test_std

    array([[-0.43860916, -0.57382672, -0.21609253, ..., -0.28872895,
             1.08786619,  0.98524841],
           [ 1.22570184, -0.57382672, -0.52943464, ..., -1.95699503,
             1.08786619, -0.25964107],
           [-1.2184578 ,  0.97378032,  0.16277341, ...,  1.37953713,
            -1.36527477, -0.25964107],
           ...,
           [ 0.65508101, -0.57382672,  0.8782423 , ..., -0.28872895,
             1.08786619, -1.50453056],
           [ 1.01171909, -0.57382672, -1.28409256, ..., -0.28872895,
             1.08786619,  0.98524841],
           [-1.56558541,  0.97378032, -1.09265374, ..., -0.28872895,
            -0.13870429, -0.25964107]])
```

X_test standardized values.

```
[ ] Y_train

    3684     163.786804
    1935    1607.241211
    5142    1510.034424
    4978    1784.343994
    2299    3558.035156
                ...
    599     5502.836914
    5695    1436.796387
    8006    2167.844727
    1361    2700.484863
    1547     829.586792
    Name: Item_Outlet_Sales, Length: 6818, dtype: float32
```

Y_train values.

```
[ ] Y_test

    8179     904.822205
    8355    2795.694092
    3411    1947.464966
    7089     872.863770
    6954    2450.144043
                ...
    1317    1721.093018
    4996     914.809204
    531      370.184814
    3891    1358.232056
    6629    2418.185547
    Name: Item_Outlet_Sales, Length: 1705, dtype: float32
```

Y_test values.

Model Building.

Task 48.

Building my model by using Linear Regression mode.

```
[ ] from sklearn.linear_model import LinearRegression
    lr= LinearRegression()
```

```
[ ] lr.fit(X_train_std,Y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
[ ] X_test.head()
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type
8179	11.000000	1	0.055163	8	100.335800	2009	1	2	2
8355	18.000000	1	0.038979	13	148.641800	1987	0	2	1
3411	7.720000	2	0.074731	1	77.598602	1997	2	0	1
7089	20.700001	1	0.049035	6	39.950600	2007	1	1	1
6954	7.550000	1	0.027225	3	152.934006	2002	1	1	1

```
[ ] Y_pred_lr=lr.predict(X_test_std)
```

```
[ ] from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

Task 49.

To get the r2 score, mean absolute error score and mean square error score for the model report.

```
[ ] print(r2_score(Y_test,Y_pred_lr))
    print(mean_absolute_error(Y_test,Y_pred_lr))
    print(np.sqrt(mean_squared_error(Y_test,Y_pred_lr)))

    0.5041875773270634
    880.9999044084501
    1162.4412631603452
```

The linear regression model applied to predict 'Item_Outlet_Sales' yielded an R2 score of 50%, indicating a moderate level of explained variance in the target variable. This metric suggests that approximately half of the variability in the sales can be attributed to the features used in the model.

In terms of absolute performance metrics, the mean absolute error (MAE) for the model is 880.9999. This implies that, on average, the predicted sales values deviate by approximately 881 units from the actual values. The MAE provides a straightforward measure of the model's accuracy, with lower values indicating better performance.

Additionally, the mean squared error (MSE) for the model is 1162.4413. The MSE penalizes larger errors more significantly than the MAE and, in this case, reflects the average squared difference between predicted and actual sales. A lower MSE indicates better accuracy, emphasizing the importance of minimizing both small and large errors in predictions.