Importing Data

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
data_train = pd.read_csv("C:\Technocolab\9961_14084_bundle_archive\Train.csv")
data_train.head()
```

Out[3]:

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

In [4]:
```
data_test = pd.read_csv("C:\Technocolab\9961_14084_bundle_archive\Test.csv")
data_test.head()
```

Out[4]:

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

In [5]:
```
data_train.shape
```

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

In [6]:
```
data_test.shape
```

Out[6]:
```
(5681, 11)
```

In [7]:
```
data = data_train
data.head()
```

Out[7]:

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

In [ ]:

EDA

In [8]:
```
data.info()
```

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

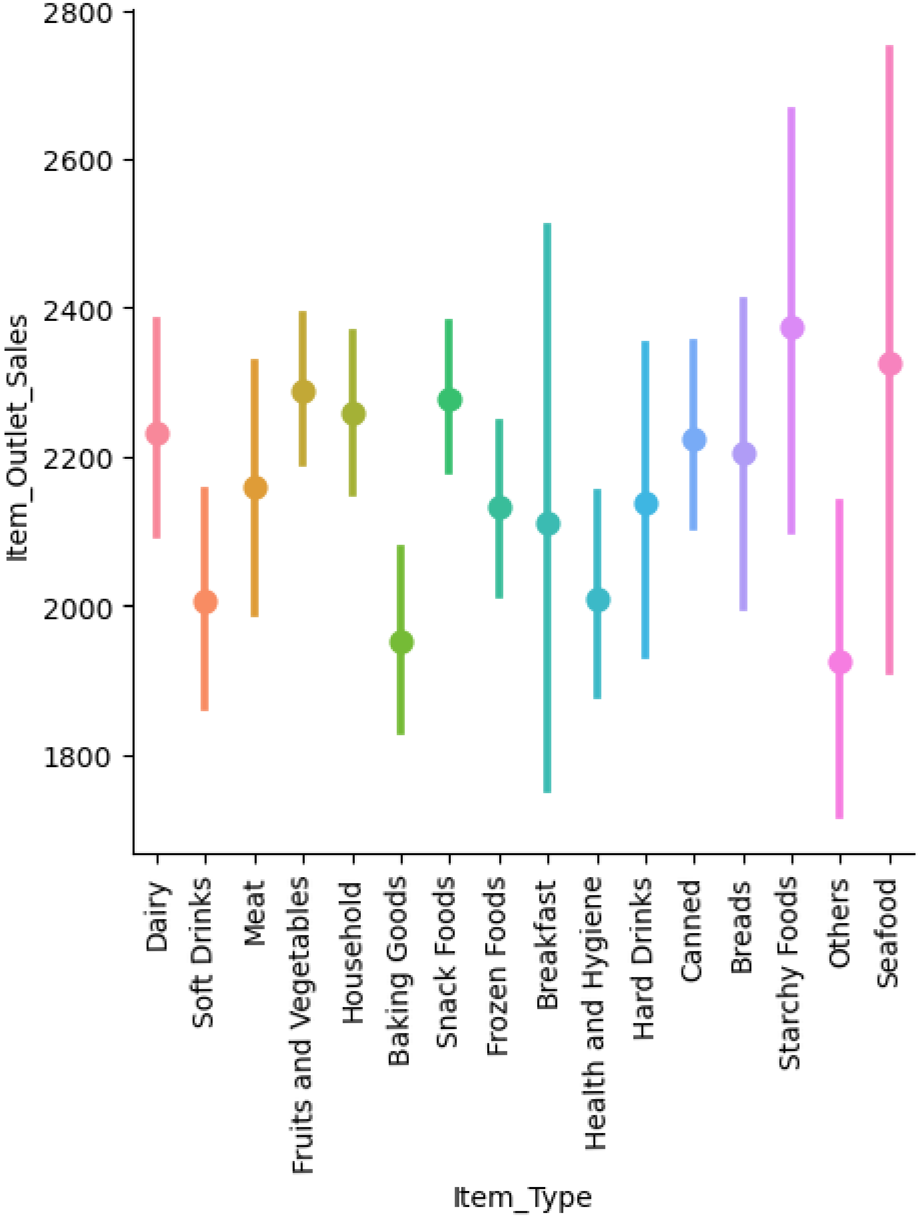
In [9]:
```
data.describe()
```

Out[9]:

	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

Data Visualization

In [10]:
```
sns.catplot(x = 'Item_Type', y = 'Item_Outlet_Sales', data = data, kind = 'point', hue = 'Item_Type')
plt.xticks(rotation = 90)
plt.show()
```


2800
2600
2400
2200
2000
1800
Item_Outlet_Sales
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

In [12]:
```
sns.boxplot(x = 'Item_Fat_Content', y = 'Item_Outlet_Sales', data = data)
```

Out[12]:
```
<AxesSubplot:xlabel='Item_Fat_Content', ylabel='Item_Outlet_Sales'>
```

In [13]:
```
sns.barplot(x = 'Outlet_Location_Type', y = 'Item_Outlet_Sales', data = data, palette='Set1')
```

Out[13]:
```
<AxesSubplot:xlabel='Outlet_Location_Type', ylabel='Item_Outlet_Sales'>
```

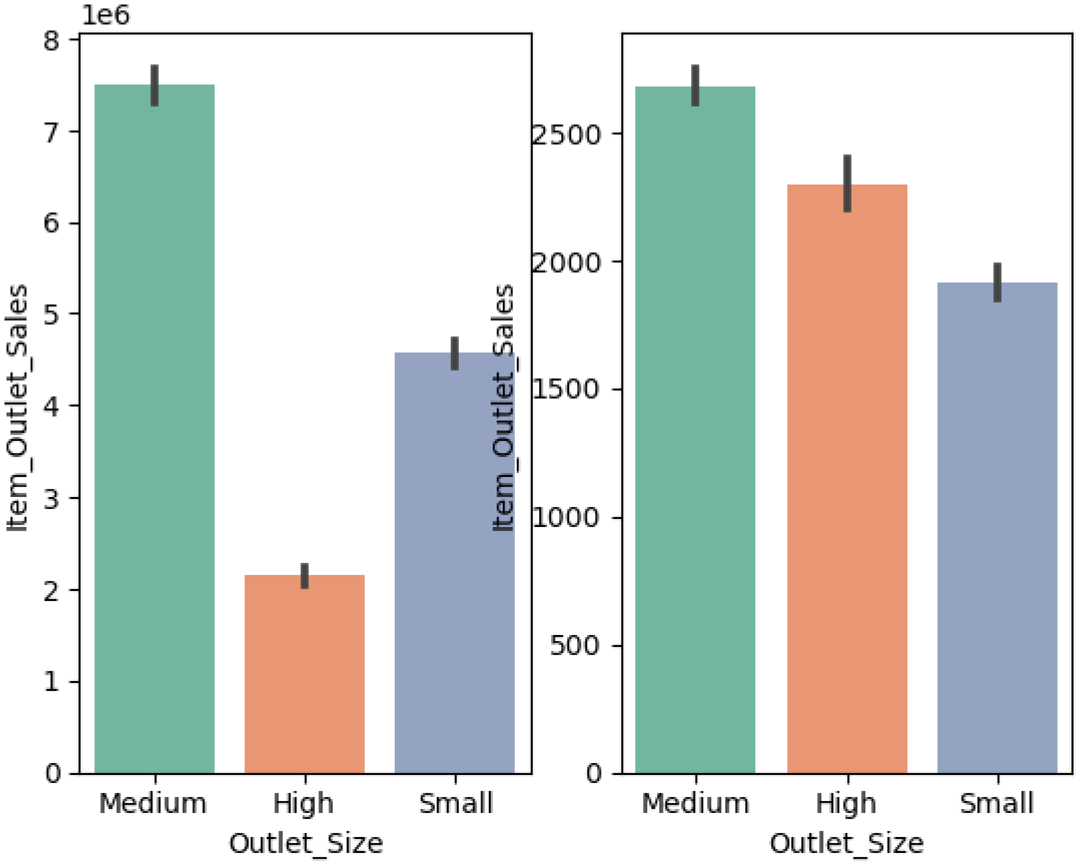
In [14]:
```
plt.subplot(2,1,1)
sns.barplot(x = 'Outlet_Establishment_Year', y = 'Item_Outlet_Sales', data = data, palette='Accent', estimator = sum)
plt.subplot(2,1,2)
sns.barplot(x = 'Outlet_Establishment_Year', y = 'Item_Outlet_Sales', data = data, palette='Accent')
```

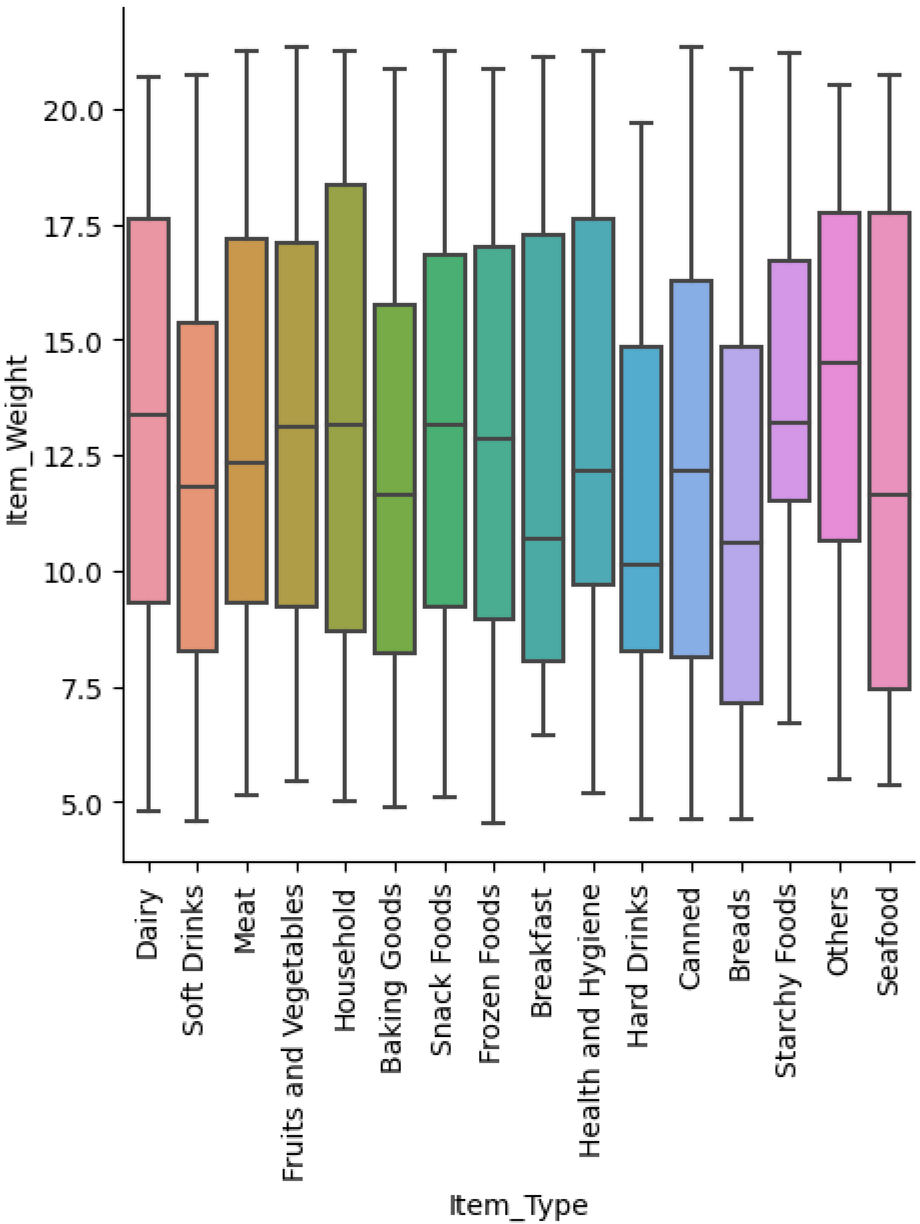
Out[14]:
```
<AxesSubplot:xlabel='Outlet_Establishment_Year', ylabel='Item_Outlet_Sales'>
```

In [15]:
```
plt.subplot(1,2,1)
sns.barplot(x = 'Outlet_Size', y = 'Item_Outlet_Sales', data = data, palette='Set2', estimator = sum)
plt.subplot(1,2,2)
sns.barplot(x = 'Outlet_Size', y = 'Item_Outlet_Sales', data = data, palette='Set2')
```

Out[15]:
```
<AxesSubplot:xlabel='Outlet_Size', ylabel='Item_Outlet_Sales'>
```

In [16]:
```
sns.catplot(x = 'Item_Type', y = 'Item_Weight', data = data, kind = 'box')
plt.xticks(rotation = 90)
plt.show()
```




20.0
17.5
15.0
12.5
10.0
7.5
5.0
Item_Weight
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

Correlation

In [17]:
```
sns.heatmap(data.corr(), annot = True)
```

Out[17]:
```
<AxesSubplot:>
```

Feature Engineering

Encoding

In [18]:
```
from sklearn.preprocessing import OrdinalEncoder

ord_enc = OrdinalEncoder()
data["Outlet_Type"] = ord_enc.fit_transform(data[["Outlet_Type"]])
data['Outlet_Location_Type'] =ord_enc.fit_transform(data[["Outlet_Location_Type"]])
data['Outlet_Size'] =ord_enc.fit_transform(data[["Outlet_Size"]])
data['Item_Fat_Content'] =ord_enc.fit_transform(data[["Item_Fat_Content"]])
data['Item_Type'] =ord_enc.fit_transform(data[["Item_Type"]])
```

In [19]:
```
data
```

Out[19]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size
0	FDA15	9.300	1.0	0.016047	4.0	249.8092	OUT049	1999	1.0
1	DRC01	5.920	2.0	0.019278	14.0	48.2692	OUT018	2009	1.0
2	FDN15	17.500	1.0	0.016760	10.0	141.6180	OUT049	1999	1.0
3	FDX07	19.200	2.0	0.000000	6.0	182.0950	OUT010	1998	NaN
4	NCD19	8.930	1.0	0.000000	9.0	53.8614	OUT013	1987	0.0
...	...	...	...	...	...	...	...	...	...
8518	FDF22	6.865	1.0	0.056783	13.0	214.5218	OUT013	1987	0.0
8519	FDS36	8.380	2.0	0.046982	0.0	108.1570	OUT045	2002	NaN
8520	NCJ29	10.600	1.0	0.035186	8.0	85.1224	OUT035	2004	2.0
8521	FDN46	7.210	2.0	0.145221	13.0	103.1332	OUT018	2009	1.0
8522	DRG01	14.800	1.0	0.044878	14.0	75.4670	OUT046	1997	2.0

8523 rows × 12 columns

In [20]:
```
data.drop(['Item_Identifier', 'Outlet_Identifier'], inplace=True, axis = 1)
```

In [21]:
```
data.head()
```

Out[21]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type
0	9.30	1.0	0.016047	4.0	249.8092	1999	1.0	0.0	1.0
1	5.92	2.0	0.019278	14.0	48.2692	2009	1.0	2.0	2.0
2	17.50	1.0	0.016760	10.0	141.6180	1999	1.0	0.0	1.0
3	19.20	2.0	0.000000	6.0	182.0950	1998	NaN	2.0	0.0
4	8.93	1.0	0.000000	9.0	53.8614	1987	0.0	2.0	1.0

Handling Outliers

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

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

In [23]:
```
q1 = data.Item_Visibility.quantile(0.25)
q3 = data.Item_Visibility.quantile(0.75)
ll = q1 - 1.5*(q3 - q1)
ul = q3 + 1.5*(q3 - q1)
```

In [24]:
```
def capping(x):
    if x < ll:
        x = ll
        return x
    elif x > ul:
        x = ul
        return x
    else:
        return x
```

In [25]:
```
data.Item_Visibility = data.Item_Visibility.apply(capping)
```

In [26]:
```
sns.boxplot(x = 'Item_Visibility', data = data)
```

Out[26]:
```
<AxesSubplot:xlabel='Item_Visibility'>
```

Handling Missing Values

In [27]:
```
data.isna().mean()*100
```

Out[27]:
```
Item_Weight                  17.165317
Item_Fat_Content              0.000000
Item_Visibility               0.000000
Item_Type                     0.000000
Item_MRP                      0.000000
Outlet_Establishment_Year     0.000000
Outlet_Size                  28.276428
Outlet_Location_Type          0.000000
Outlet_Type                   0.000000
Item_Outlet_Sales             0.000000
dtype: float64
```

In [28]:
```
from sklearn.impute import KNNImputer
imputer = KNNImputer(n_neighbors=3)
```

In [29]:
```
imputed = imputer.fit_transform(data)
```

In [30]:
```
data1 = pd.DataFrame(imputed, columns = data.columns)
data1
```

Out[30]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Ty
0	9.300	1.0	0.016047	4.0	249.8092	1999.0	1.0	0.0	
1	5.920	2.0	0.019278	14.0	48.2692	2009.0	1.0	2.0	
2	17.500	1.0	0.016760	10.0	141.6180	1999.0	1.0	0.0	
3	19.200	2.0	0.000000	6.0	182.0950	1998.0	2.0	2.0	
4	8.930	1.0	0.000000	9.0	53.8614	1987.0	0.0	2.0	
...	...	...	...	...	...	...	...	...	
8518	6.865	1.0	0.056783	13.0	214.5218	1987.0	0.0	2.0	
8519	8.380	2.0	0.046982	0.0	108.1570	2002.0	1.0	1.0	
8520	10.600	1.0	0.035186	8.0	85.1224	2004.0	2.0	1.0	
8521	7.210	2.0	0.145221	13.0	103.1332	2009.0	1.0	2.0	
8522	14.800	1.0	0.044878	14.0	75.4670	1997.0	2.0	0.0	

8523 rows × 10 columns

In [31]:
```
data1.isna().mean()*100
```

Out[31]:
```
Item_Weight                  0.0
Item_Fat_Content             0.0
Item_Visibility              0.0
Item_Type                    0.0
Item_MRP                     0.0
Outlet_Establishment_Year    0.0
Outlet_Size                  0.0
Outlet_Location_Type         0.0
Outlet_Type                  0.0
Item_Outlet_Sales            0.0
dtype: float64
```

In [32]:
```
data1.Outlet_Size = round(data1.Outlet_Size,)
```

ML Model Building

Data Splitting

In [33]:
```
data1.head()
```

Out[33]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type
0	9.30	1.0	0.016047	4.0	249.8092	1999.0	1.0	0.0	1.0
1	5.92	2.0	0.019278	14.0	48.2692	2009.0	1.0	2.0	2.0
2	17.50	1.0	0.016760	10.0	141.6180	1999.0	1.0	0.0	1.0
3	19.20	2.0	0.000000	6.0	182.0950	1998.0	2.0	2.0	0.0
4	8.93	1.0	0.000000	9.0	53.8614	1987.0	0.0	2.0	1.0

In [34]:
```
X = data1.iloc[:,:-1]
X
```

Out[34]:

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Ty
0	9.300	1.0	0.016047	4.0	249.8092	1999.0	1.0	0.0	
1	5.920	2.0	0.019278	14.0	48.2692	2009.0	1.0	2.0	
2	17.500	1.0	0.016760	10.0	141.6180	1999.0	1.0	0.0	
3	19.200	2.0	0.000000	6.0	182.0950	1998.0	2.0	2.0	
4	8.930	1.0	0.000000	9.0	53.8614	1987.0	0.0	2.0	
...	...	...	...	...	...	...	...	...	
8518	6.865	1.0	0.056783	13.0	214.5218	1987.0	0.0	2.0	
8519	8.380	2.0	0.046982	0.0	108.1570	2002.0	1.0	1.0	
8520	10.600	1.0	0.035186	8.0	85.1224	2004.0	2.0	1.0	
8521	7.210	2.0	0.145221	13.0	103.1332	2009.0	1.0	2.0	
8522	14.800	1.0	0.044878	14.0	75.4670	1997.0	2.0	0.0	

8523 rows × 9 columns

In [35]:
```
y = data1.Item_Outlet_Sales
y
```

Out[35]:
```
0       3735.1380
1        443.4228
2       2097.2700
3        732.3800
4        994.7052
          ...
8518    2778.3834
8519     549.2850
8520    1193.1136
8521    1845.5976
8522     765.6700
Name: Item_Outlet_Sales, Length: 8523, dtype: float64
```

In [36]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

In [38]:
```
print(f'Samples of trained data = {X_train.shape[0]} and Samples of testing data = {X_test.shape[0]}')
```

```
Samples of trained data = 6818 and Samples of testing data = 1705
```

In [39]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

In [ ]:
```

```

Linear Regression

In [40]:
```
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg_model = reg.fit(X_train, y_train)
lr_pred = reg.predict(X_test)
```

In [41]:
```
MSE=mean_squared_error(y_test,lr_pred)
MAE=mean_absolute_error(y_test,lr_pred)
r2=r2_score(y_test,lr_pred)
RMSE = np.sqrt(MSE)
print("R squared value: ", r2)
print("Root Mean Squared Error : ", RMSE)
print("Mean Absolute Error : ", MAE)
```

```
R squared value:  0.5230197859838783
Root Mean Squared Error :  1138.603506082893
Mean Absolute Error :  857.8120480090059
```

In [ ]:

Ridge Regression

In [42]:
```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1.0)
ridge_model = ridge.fit(X_train, y_train)
ridge_pred = ridge.predict(X_test)
```

In [43]:
```
MSE=mean_squared_error(y_test,ridge_pred)
MAE=mean_absolute_error(y_test,ridge_pred)
r2=r2_score(y_test,ridge_pred)
RMSE = np.sqrt(MSE)
print("R squared value: ", r2)
print("Root Mean Squared Error : ", RMSE)
print("Mean Absolute Error : ", MAE)
```

```
R squared value:  0.5231530154893569
Root Mean Squared Error :  1138.4444783371632
Mean Absolute Error :  857.6418174505641
```

In [ ]:

Lasso Regression

In [44]:
```
from sklearn.linear_model import Lasso
Lasso = Lasso(alpha=1)
Lasso_model = Lasso.fit(X_train, y_train)
Lasso_pred = Lasso.predict(X_test)
```

In [45]:
```
MSE=mean_squared_error(y_test,Lasso_pred)
MAE=mean_absolute_error(y_test,Lasso_pred)
r2=r2_score(y_test,Lasso_pred)
RMSE = np.sqrt(MSE)
print("R squared value: ", r2)
print("Root Mean Squared Error : ", RMSE)
print("Mean Absolute Error : ", MAE)
```

```
R squared value:  0.5235058338097819
Root Mean Squared Error :  1138.0232337797443
Mean Absolute Error :  857.0861568464775
```

In [ ]:

ElasticNet Regression

In [46]:
```
from sklearn.linear_model import ElasticNet
ElasticNet = ElasticNet(l1_ratio = 1)
ElasticNet_model = ElasticNet.fit(X_train, y_train)
ElasticNet_pred = ElasticNet.predict(X_test)
```

In [47]:
```
MSE=mean_squared_error(y_test,ElasticNet_pred)
MAE=mean_absolute_error(y_test,ElasticNet_pred)
r2=r2_score(y_test,ElasticNet_pred)
RMSE = np.sqrt(MSE)
print("R squared value: ", r2)
print("Root Mean Squared Error : ", RMSE)
print("Mean Absolute Error : ", MAE)
```

```
R squared value:  0.5235058338097819
Root Mean Squared Error :  1138.0232337797443
Mean Absolute Error :  857.0861568464775
```

In [ ]:

Random Forest Regressor

In [48]:
```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor(max_depth=6, random_state=0)
rfr_model = rfr.fit(X_train, y_train)
rfr_model_pred = rfr_model.predict(X_test)
```

In [49]:
```
MSE=mean_squared_error(y_test,rfr_model_pred)
MAE=mean_absolute_error(y_test,rfr_model_pred)
r2=r2_score(y_test,rfr_model_pred)
RMSE = np.sqrt(MSE)
print("R squared value: ", r2)
print("Root Mean Squared Error : ", RMSE)
print("Mean Absolute Error : ", MAE)
```

```
R squared value:  0.6170336731528743
Root Mean Squared Error :  1020.2406642026398
Mean Absolute Error :  711.5250204428983
```

In [ ]:

XGBoost Regressor

In [50]:
```
from sklearn.ensemble import GradientBoostingRegressor
gbr = GradientBoostingRegressor(random_state=0)
gbr_model = gbr.fit(X_train, y_train)
gbr_model_pred = gbr_model.predict(X_test)
```

In [51]:
```
MSE=mean_squared_error(y_test,gbr_model_pred)
MAE=mean_absolute_error(y_test,gbr_model_pred)
r2=r2_score(y_test,gbr_model_pred)
RMSE = np.sqrt(MSE)
print("R squared value: ", r2)
print("Root Mean Squared Error : ", RMSE)
print("Mean Absolute Error : ", MAE)
```

```
R squared value:  0.6089663354221819
Root Mean Squared Error :  1030.9305482727502
Mean Absolute Error :  721.1821851870911
```

Summary:

The linear regression, ridge regression, lasso regression, and elasticnet regression models show similar performance with R-squared values around 0.52. They have relatively high RMSE and MAE values, indicating some degree of prediction error.

The random forest regressor and XGBoost regressor outperform the linear-based models, with higher R-squared values (0.6170 and 0.6090, respectively) and lower RMSE and MAE values. This suggests that these ensemble models provide better predictive accuracy on the given dataset.

The random forest regressor appears to perform slightly better than the XGBoost regressor in terms of RMSE and MAE.

In conclusion, based on the provided metrics, the ensemble models (Random Forest and XGBoost) seem to be more effective in capturing the underlying patterns in the data and making more accurate predictions compared to the traditional linear regression and its regularized variants.

In [ ]:

```
```