BigMart Sales Prediction

In [95]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import os

from sklearn.impute import KNNImputer
from sklearn.metrics import accuracy_score, mean_squared_error
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler, LabelEncoder

#import xgboost as xg
from sklearn.svm import SVR
from sklearn.svm import LinearSVR
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import Lasso
from sklearn.ensemble import StackingRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split

import warnings
np.random.seed(0)
warnings.filterwarnings('ignore')
```

In [96]:
```
train = pd.read_csv('Train.csv')
test = pd.read_csv('Test.csv')
```

In [97]:
```
train.shape, test.shape
```

Out[97]:
```
((8523, 12), (5681, 11))
```

In [98]:
```
train.head()
```

Out[98]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Yea
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	199
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	200
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	199
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	199
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	198

In [99]:
```
test.head()
```

Out[99]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Yea
0	FDW58	20.750	Low Fat	0.007565	Snack Foods	107.8622	OUT049	199
1	FDW14	8.300	reg	0.038428	Dairy	87.3198	OUT017	200
2	NCN55	14.600	Low Fat	0.099575	Others	241.7538	OUT010	199

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Yea
3	FDQ58	7.315	Low Fat	0.015388	Snack Foods	155.0340	OUT017	200
4	FDY38	NaN	Regular	0.118599	Dairy	234.2300	OUT027	198

Data Cleaning and Pre-Processing of Train Dataset

In [100…
```
# Check the missing values if any
train.isnull().sum()
```

Out[100…
```
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
```

In [101…
```
# Summary Statistics
train.describe(include='all').T
```

Out[101…

	count	unique	top	freq	mean	std	min	25%	50%	75%	
Item_Identifier	8523	1559	FDW13	10	NaN	NaN	NaN	NaN	NaN	NaN	
Item_Weight	7060.0	NaN	NaN	NaN	12.857645	4.643456	4.555	8.77375	12.6	16.85	
Item_Fat_Content	8523	5	Low Fat	5089	NaN	NaN	NaN	NaN	NaN	NaN	
Item_Visibility	8523.0	NaN	NaN	NaN	0.066132	0.051598	0.0	0.026989	0.053931	0.094585	0
Item_Type	8523	16	Fruits and Vegetables	1232	NaN	NaN	NaN	NaN	NaN	NaN	
Item_MRP	8523.0	NaN	NaN	NaN	140.992782	62.275067	31.29	93.8265	143.0128	185.6437	2
Outlet_Identifier	8523	10	OUT027	935	NaN	NaN	NaN	NaN	NaN	NaN	
Outlet_Establishment_Year	8523.0	NaN	NaN	NaN	1997.831867	8.37176	1985.0	1987.0	1999.0	2004.0	
Outlet_Size	6113	3	Medium	2793	NaN	NaN	NaN	NaN	NaN	NaN	
Outlet_Location_Type	8523	3	Tier 3	3350	NaN	NaN	NaN	NaN	NaN	NaN	
Outlet_Type	8523	4	Supermarket Type1	5577	NaN	NaN	NaN	NaN	NaN	NaN	
Item_Outlet_Sales	8523.0	NaN	NaN	NaN	2181.288914	1706.499616	33.29	834.2474	1794.331	3101.2964	130

In [102…
```
# filling the missing values in the Item_Weight column using the Mean value
train['Item_Weight'].fillna(train['Item_Weight'].mean(), inplace = True)
```

In [103…
```
# We know that, the Outlet_size and Outlet_Type are related to each other. So we are filling the missing valu
# mode of the Outlet_size
mode_of_outlet_size = pd.DataFrame(train.pivot_table(values = 'Outlet_Size', columns = 'Outlet_Type', aggfunc
mode_of_outlet_size
```

Out[103…

Outlet_Type	Grocery Store	Supermarket Type1	Supermarket Type2	Supermarket Type3
Outlet_Size	Small	Small	Medium	Medium

In [104…
```
# filling the missing values of Outlet_Size with the mode values
train.loc[train['Outlet_Size'].isnull(), 'Outlet_Size'] = train.loc[train['Outlet_Size'].isnull(), 'Outlet_Ty
```

In [105…]
```
train.isnull().sum()
```

Out[105…]
```
Item_Identifier              0
Item_Weight                  0
Item_Fat_Content             0
Item_Visibility              0
Item_Type                    0
Item_MRP                     0
Outlet_Identifier            0
Outlet_Establishment_Year    0
Outlet_Size                  0
Outlet_Location_Type         0
Outlet_Type                  0
Item_Outlet_Sales            0
dtype: int64
```

In [106…]
```
# creating list of categorical columns for one hot encoding
categorical_columns = [col for col in train.columns if train.dtypes[col] == 'object']

# creating list of numerical columns to standardized data
numerical_columns = [col for col in train.columns if train.dtypes[col] != 'object']

print('Numerical Features are : ',numerical_columns)
print('Categorical Features are : ',categorical_columns)
```

```
Numerical Features are :  ['Item_Weight', 'Item_Visibility', 'Item_MRP', 'Outlet_Establishment_Year', 'Item_O
utlet_Sales']
Categorical Features are :  ['Item_Identifier', 'Item_Fat_Content', 'Item_Type', 'Outlet_Identifier', 'Outlet
_Size', 'Outlet_Location_Type', 'Outlet_Type']
```

In [107…]
```
unique_categories_count_list = [{col:len(train[col].unique())} for col in categorical_columns]
unique_categories_count_list
```

Out[107…]
```
[{'Item_Identifier': 1559},
 {'Item_Fat_Content': 5},
 {'Item_Type': 16},
 {'Outlet_Identifier': 10},
 {'Outlet_Size': 3},
 {'Outlet_Location_Type': 3},
 {'Outlet_Type': 4}]
```

In [108…]
```
unique_categories_list = [{col:train[col].unique()} for col in categorical_columns if col != 'Item_Identifier
unique_categories_list
```

Out[108…]
```
[{'Item_Fat_Content': array(['Low Fat', 'Regular', 'low fat', 'LF', 'reg'], dtype=object)},
 {'Item_Type': array(['Dairy', 'Soft Drinks', 'Meat', 'Fruits and Vegetables',
        'Household', 'Baking Goods', 'Snack Foods', 'Frozen Foods',
        'Breakfast', 'Health and Hygiene', 'Hard Drinks', 'Canned',
        'Breads', 'Starchy Foods', 'Others', 'Seafood'], dtype=object)},
 {'Outlet_Identifier': array(['OUT049', 'OUT018', 'OUT010', 'OUT013', 'OUT027', 'OUT045',
        'OUT017', 'OUT046', 'OUT035', 'OUT019'], dtype=object)},
 {'Outlet_Size': array(['Medium', 'Small', 'High'], dtype=object)},
 {'Outlet_Location_Type': array(['Tier 1', 'Tier 3', 'Tier 2'], dtype=object)},
 {'Outlet_Type': array(['Supermarket Type1', 'Supermarket Type2', 'Grocery Store',
        'Supermarket Type3'], dtype=object)}]
```

In [109…]
```
# As you can see the low fat and regular in item_fat_content is written differently so first we need to corre
train = train.replace({'Item_Fat_Content': r'^reg'}, {'Item_Fat_Content': 'Regular'}, regex=True)
train = train.replace({'Item_Fat_Content':[r'Low Fat',r'LF',r'low fat']}, {'Item_Fat_Content': 'Low Fat'}, re
```

In [110…]
```
train['Item_Fat_Content'].value_counts()
```

Out[110…]
```
Low Fat    5517
Regular    3006
Name: Item_Fat_Content, dtype: int64
```

Data Cleaning and Pre-processing of Test Dataset

In [111…]
```
# Check the missing values if any
test.isnull().sum()
```

Out[111…
```
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
```

In [112…
```
# filling the missing values in the Item_Weight column using the Mean value
test['Item_Weight'].fillna(test['Item_Weight'].mean(), inplace = True)
```

In [113…
```
# mode of the Outlet_size
mode_of_outlet_size1 = pd.DataFrame(test.pivot_table(values = 'Outlet_Size', columns = 'Outlet_Type', aggfunc
mode_of_outlet_size1
```

Out[113…

Outlet_Type	Grocery Store	Supermarket Type1	Supermarket Type2	Supermarket Type3
Outlet_Size	Small	Small	Medium	Medium

In [114…
```
# filling the missing values of Outlet_Size with the mode values
test.loc[test['Outlet_Size'].isnull(), 'Outlet_Size'] = test.loc[test['Outlet_Size'].isnull(), 'Outlet_Type']
```

In [115…
```
test.isnull().sum()
```

Out[115…
```
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 [116…
```
# creating list of categorical columns for one hot encoding
categorical_columns1 = [col for col in test.columns if test.dtypes[col] == 'object']

# creating list of numerical columns to standardized data
numerical_columns1 = [col for col in test.columns if test.dtypes[col] != 'object']

print('Numerical Features are : ',numerical_columns1)
print('Categorical Features are : ',categorical_columns1)
```

```
Numerical Features are :  ['Item_Weight', 'Item_Visibility', 'Item_MRP', 'Outlet_Establishment_Year']
Categorical Features are :  ['Item_Identifier', 'Item_Fat_Content', 'Item_Type', 'Outlet_Identifier', 'Outlet
_Size', 'Outlet_Location_Type', 'Outlet_Type']
```

In [117…
```
unique_categories_count_list1 = [{col:len(test[col].unique())} for col in categorical_columns1]
unique_categories_count_list1
```

Out[117…
```
[{'Item_Identifier': 1543},
 {'Item_Fat_Content': 5},
 {'Item_Type': 16},
 {'Outlet_Identifier': 10},
 {'Outlet_Size': 3},
 {'Outlet_Location_Type': 3},
 {'Outlet_Type': 4}]
```

In [118…
```
unique_categories_list1 = [{col:test[col].unique()} for col in categorical_columns1 if col != 'Item_Identifie
unique_categories_list1
```

Out[118…
```
[{'Item_Fat_Content': array(['Low Fat', 'reg', 'Regular', 'LF', 'low fat'], dtype=object)},
 {'Item_Type': array(['Snack Foods', 'Dairy', 'Others', 'Fruits and Vegetables',
        'Baking Goods', 'Health and Hygiene', 'Breads', 'Hard Drinks',
        'Seafood', 'Soft Drinks', 'Household', 'Frozen Foods', 'Meat',
```

```
        'Canned', 'Starchy Foods', 'Breakfast'], dtype=object)},
 {'Outlet_Identifier': array(['OUT049', 'OUT017', 'OUT010', 'OUT027', 'OUT046', 'OUT018',
        'OUT045', 'OUT019', 'OUT013', 'OUT035'], dtype=object)},
 {'Outlet_Size': array(['Medium', 'Small', 'High'], dtype=object)},
 {'Outlet_Location_Type': array(['Tier 1', 'Tier 2', 'Tier 3'], dtype=object)},
 {'Outlet_Type': array(['Supermarket Type1', 'Grocery Store', 'Supermarket Type3',
        'Supermarket Type2'], dtype=object)}]
```

In [119…

```
# As you can see the low fat and regular in item_fat_content is written differently so first we need to corre
test = test.replace({'Item_Fat_Content': r'^reg'}, {'Item_Fat_Content': 'Regular'}, regex=True)
test = test.replace({'Item_Fat_Content':[r'Low Fat',r'LF',r'low fat']}, {'Item_Fat_Content': 'Low Fat'}, rege
```

In [120…

```
test['Item_Fat_Content'].value_counts()
```

Out[120…

```
Low Fat    3668
Regular    2013
Name: Item_Fat_Content, dtype: int64
```

Ecoding, Splitting and Scalling Data

In [121…

```
for i in train.columns:
    if train[i].dtype=='object':
            label_encoder=LabelEncoder()
            train[i]=label_encoder.fit_transform(train[i])
```

In [122…

```
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   int32
 1   Item_Weight                8523 non-null   float64
 2   Item_Fat_Content           8523 non-null   int32
 3   Item_Visibility            8523 non-null   float64
 4   Item_Type                  8523 non-null   int32
 5   Item_MRP                   8523 non-null   float64
 6   Outlet_Identifier          8523 non-null   int32
 7   Outlet_Establishment_Year  8523 non-null   int64
 8   Outlet_Size                8523 non-null   int32
 9   Outlet_Location_Type       8523 non-null   int32
 10  Outlet_Type                8523 non-null   int32
 11  Item_Outlet_Sales          8523 non-null   float64
dtypes: float64(4), int32(7), int64(1)
memory usage: 566.1 KB
```

In [123…

```
X = train.drop(columns='Item_Outlet_Sales', axis=1)
Y = train['Item_Outlet_Sales']
```

In [124…

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

In [125…

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=3)
```

In [126…

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(8523, 11) (5966, 11) (2557, 11)
```

Model Building

Linear Regression

In [127…

```
lin = LinearRegression(normalize=True,fit_intercept= True)
lin.fit(X_train,Y_train)
predlin=lin.predict(X_test)
linmse = mean_squared_error(Y_test, predlin,squared=False)
linmse
```

Out[127…

```
1189.2798483760962
```

SVM

In [128…

```
svm = LinearSVR()
svm.fit(X_train,Y_train)
predsvm = svm.predict(X_test)
svmse = mean_squared_error(Y_test, predsvm,squared=False)
svmse
```

Out[128…

```
1419.0174795756775
```

XGBoost

In [129…

```
xg = XGBRegressor()
xg.fit(X_train, Y_train)
predxg = xg.predict(X_test)
xgmse = mean_squared_error(Y_test, predxg,squared=False)
xgmse
```

Out[129…

```
1174.9307585672818
```

Laaso

In [130…

```
ls = Lasso(alpha = 0.01)
ls.fit(X_train,Y_train)
prels = ls.predict(X_test)
lsmse = mean_squared_error(Y_test, prels,squared=False)
lsmse
```

Out[130…

```
1189.2788335378048
```

K Nearest Neighbors

In [131…

```
knn = KNeighborsRegressor()
knn.fit(X_train,Y_train)
predknn = knn.predict(X_test)
knnmse = mean_squared_error(Y_test,predknn,squared=False)
knnmse
```

Out[131…

```
1157.314645126738
```

Random Forest

In [132…

```
rfr = RandomForestRegressor(n_estimators=100, random_state=0)
rfr.fit(X_train, Y_train)
predrf = rfr.predict(X_test)
rfrmse = mean_squared_error(Y_test, predrf,squared=False)
rfrmse
```

Out[132…

```
1126.9690143073085
```

Decision Tree

In [133…

```
dt = DecisionTreeRegressor()
dt.fit(X_train, Y_train)
preddt = dt.predict(X_test)
dtmse = mean_squared_error(Y_test,preddt,squared=False)
dtmse
```

Out[133…

```
1548.7886168214825
```

Stacking Regressor

In [134…

```
estimators = [('lr', RidgeCV()),('svr', LinearSVR(random_state=42))]
stack_reg = StackingRegressor(estimators=estimators,final_estimator=RandomForestRegressor(n_estimators=10,ran
stack_reg.fit(X_train,Y_train)
predstack = stack_reg.predict(X_test)
stackmse = mean_squared_error(Y_test,predstack,squared=False)
stackmse
```

Out[134…

```
1327.3929825470716
```

Final Results

In [135…

```
res = pd.DataFrame({'Model' : ['LINEAR REGRESSION','SVM','XG BOOST','LAASO','K NEAREST NEIGHBOR','RANDOM FORE
                    'Score' : [linmse,svmse,xgmse,lsmse,knnmse,rfrmse,dtmse,stackmse]})
res_df = res.sort_values(by='Score',ascending=True)
res_df = res_df.set_index('Score')
res_df
```

Out[135…

Score	Model
1126.969014	RANDOM FOREST
1157.314645	K NEAREST NEIGHBOR
1174.930759	XG BOOST
1189.278834	LAASO
1189.279848	LINEAR REGRESSION
1327.392983	STACKING REGRESSO
1419.017480	SVM
1548.788617	DECISION TREE

As we can see that Random Forest has the least mean squared error, it is the best model for BigMart Sales prediction.