Data Science Engineering Project

Northeastern University

Fall 2019

Supermarket Chain Sales Prediction

Introduction

Supermarkets have a lot of products in the store, some really popular and few not so much. If we could predict which products contribute the most in the sales, it would be beneficial for the owner to stock up those products in larger quantity and prevent shortage of the same. Likewise, to stock up the less popular products in small quantity to avoid wastage.

The dataset from Kaggle contains- 2013 deals information for 1559 items crosswise over 10 stores in various cities. Likewise, certain properties of every item and store have been characterized. With this information the corporation hopes to identify the products and stores which play a key role in their sales and use that information to take the correct measures to ensure success of their business.
My aim is to build an easily scalable model to provide detailed information and accurate predictions for sales volume for different type of products.

Dataset

This dataset is created to predict the sales of Supermarket chain. The data is of the year 2013 for 1559 products across 10 stores in different cities. We used this information to identify the products and stores which play a key role in their Sales and use this information to take correct measures for the success of their Business Steps like Data visualization, data quality, EDA, various machine learning algorithms and model's assumption checking have been implemented.

The target variable in the project is Item_Outlet_Sales

```
In [52]: train = pd.read_csv(f'{path}/train.csv')
         print(train.shape)
         train.head()
```

(8523, 12)

Out[52]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Locatio
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	

If we look at variable Item_Identifier, we can see different group of letters per each product such as ‘FD’ (Food), ‘DR’(Drinks) and ‘NC’ (Non-Consumable).

```
In [54]: #checking data types
         train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 12 columns):
Item_Identifier              8523 non-null object
Item_Weight                  7060 non-null float64
Item_Fat_Content             8523 non-null object
Item_Visibility              8523 non-null float64
Item_Type                    8523 non-null object
Item_MRP                     8523 non-null float64
Outlet_Identifier            8523 non-null object
Outlet_Establishment_Year    8523 non-null int64
Outlet_Size                  6113 non-null object
Outlet_Location_Type         8523 non-null object
Outlet_Type                  8523 non-null object
Item_Outlet_Sales            8523 non-null float64
dtypes: float64(4), int64(1), object(7)
memory usage: 799.1+ KB
```

Most of the items in the train dataset present 8523 non-null values. However, there are some cases such as Item_Weight and Outlet_Size which seem to present Null values. We always have to consider if this absence of values has a significant meaning. In this case it does not since all values should have weight higher than 0 and a store cannot exist with zero size.Moreover, from the 12 features, 5 are numeric and 7 categorical.

On the other hand, regarding Item_Visibility there are items with the value zero. This does not make sense, since this is indicating those items are not visible on the store.

In [55]:
```
#Description of statistic features (Sum, Average, Variance, minimum, 1st quartile, 2nd quartile, 3rd Quartile and Ma
train.describe()
```

Out[55]:

	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

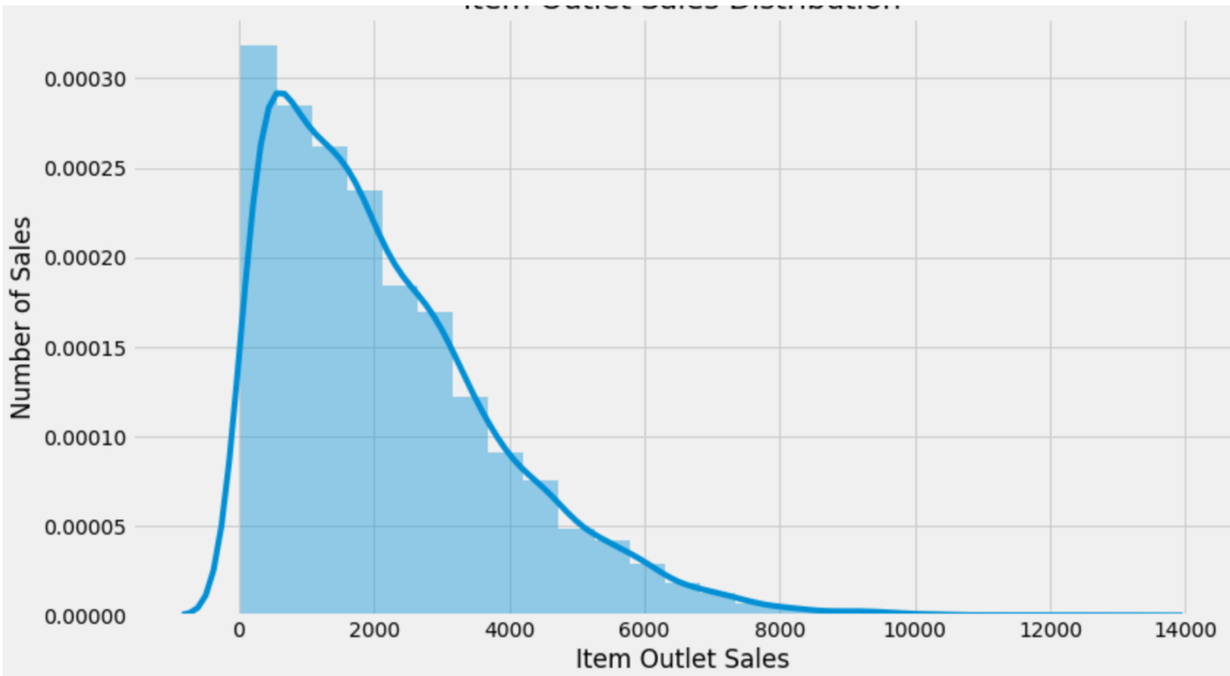
Methodology

To define the best regression model for predicting Item_Outlet_Sales the following steps were implemented

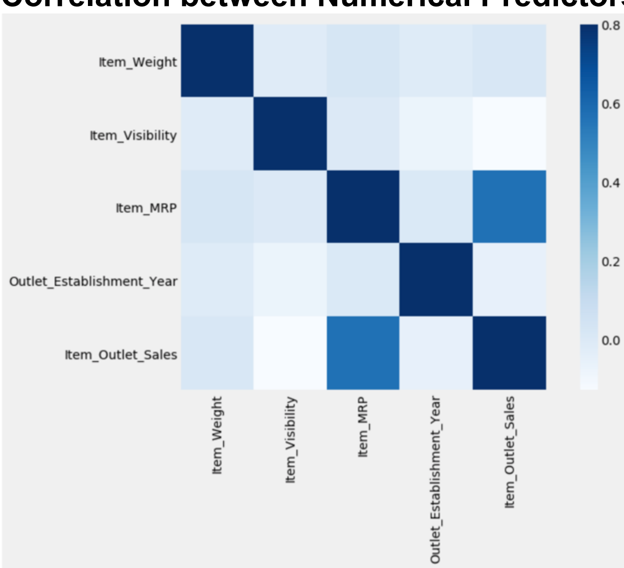
- Importing Packages
- Checking for duplicates
- Exploratory data analysis (EDA)
- Univariate Distribution
- Bivariate Distribution
- Data Pre-Processing
- Checking for missing values and data imputation
- Feature Engineering
- Feature Transformation
- Modeling

Univariate Analysis

To get an idea of the distribution of numerical variables, histograms are the best option. Therefore, generating histogram for **Item_Outlet_Sales**

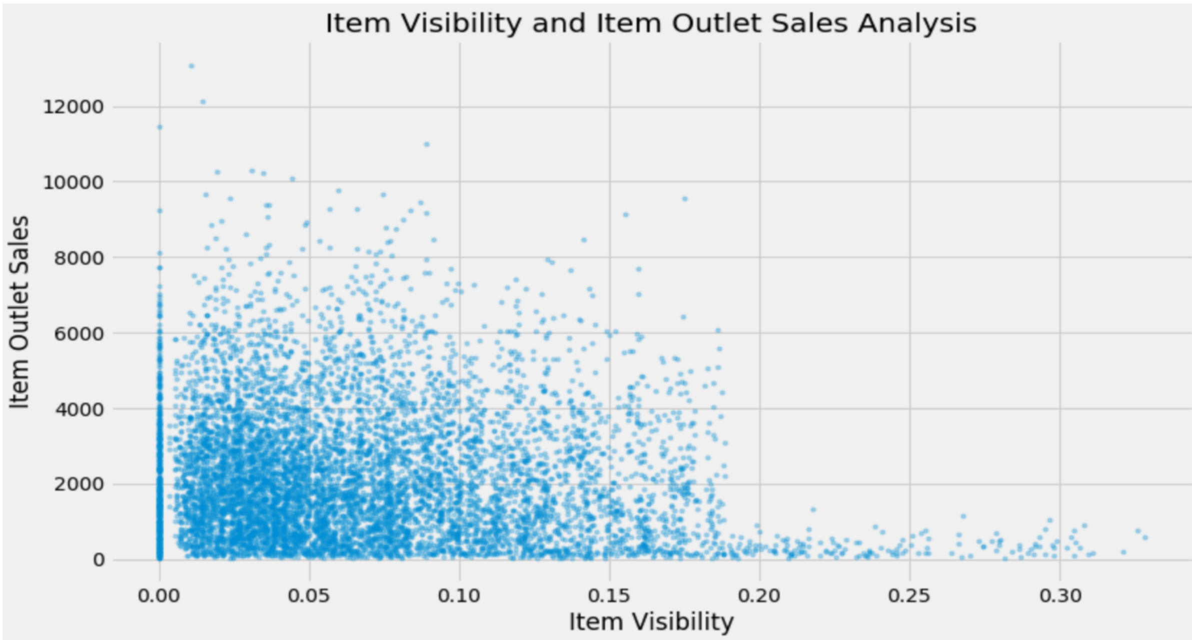


Correlation between Numerical Predictors and Item_Outlet_Sales

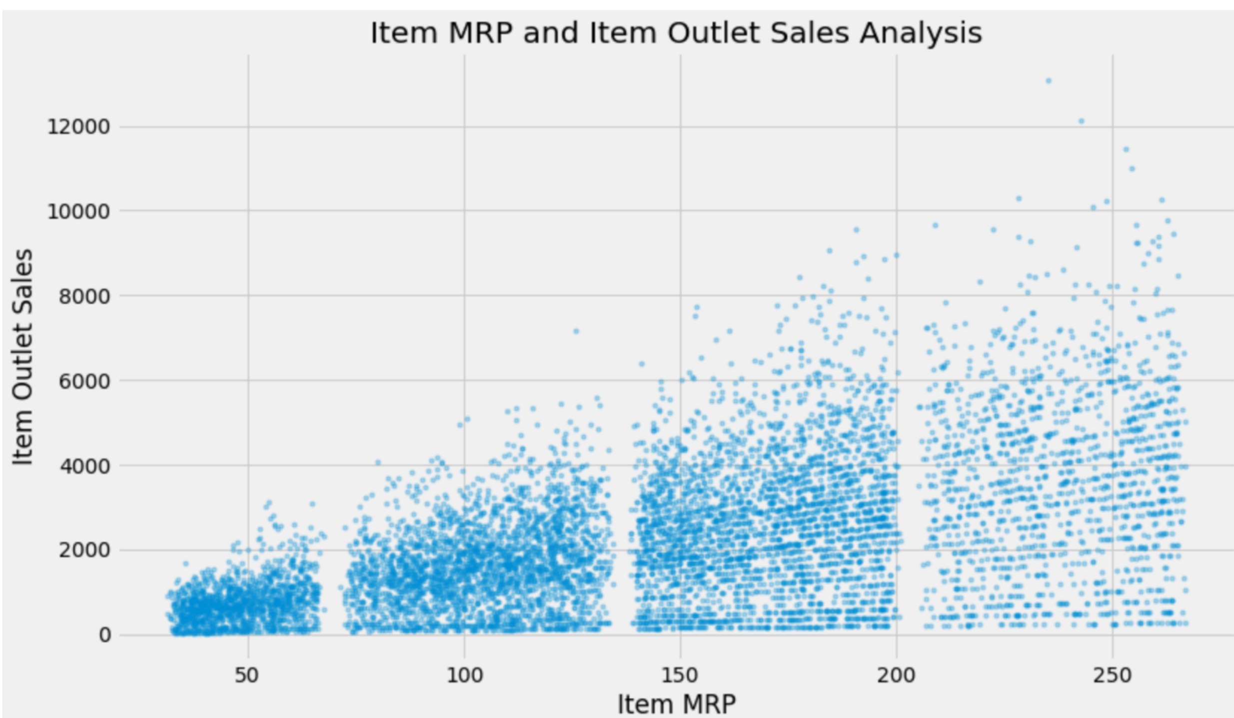


It is observed that the Item_Visibility is the feature with the lowest correlation with our target variable. Therefore, the less visible the product is in the store the higher the price will be. This feature has a negative correlation with all of the other features. The most positive correlation belongs to Item_MRP.

Item Visibility and Item Outlet Sales Analysis



Item MRP and Item Outlet Sales Analysis



Checking for missing values and data imputation

Looking for missing values

In [89]:
```
#Joining Train and Test Dataset
train['source']='train'
test['source']='test'

data = pd.concat([train,test], ignore_index = True)
data.to_csv(f'{path}/data.csv',index=False)
print(train.shape, test.shape, data.shape)
```

```
(8523, 13) (5681, 12) (14204, 13)
```

```
/Applications/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:5: FutureWarning: Sorting because non-con
catenation axis is not aligned. A future version
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

  """
```

In [91]:
```
#Taking the mean of the weights of all the products to fill in the missing values
def impute_weight(cols):
    Weight = cols[0]
    Identifier = cols[1]

    if pd.isnull(Weight):
        return item_avg_weight['Item_Weight'][item_avg_weight.index == Identifier]
    else:
        return Weight
```

From the output we can clearly see that originally there were 2439 missing values and after filling those rows there are 0 NaN values

In [92]:
```
print ('Orignal #missing: %d'%sum(data['Item_Weight'].isnull()))
data['Item_Weight'] = data[['Item_Weight','Item_Identifier']].apply(impute_weight,axis=1).astype(float)
print ('Final #missing: %d'%sum(data['Item_Weight'].isnull()))
```

```
Orignal #missing: 2439
Final #missing: 0
```

Feature Engineering

(1) Item_Visibility minimum value is 0

In [96]:
```
#As seen above the visibility of some items were 0 which is not possible because every product has some visibility a
#Therefore I consider these 0 values as missing values and imputed them by taking the mean
#Item_Visibility minimum value 0
#Getting all Item_Visibility mean values for respective Item_Identifier
visibility_item_avg = data.pivot_table(values='Item_Visibility',index='Item_Identifier')
```

In [97]:
```
def impute_visibility_mean(cols):
    visibility = cols[0]
    item = cols[1]
    if visibility == 0:
        return visibility_item_avg['Item_Visibility'][visibility_item_avg.index == item]
    else:
        return visibility

print ('Original #zeros: %d'%sum(data['Item_Visibility'] == 0))
data['Item_Visibility'] = data[['Item_Visibility','Item_Identifier']].apply(impute_visibility_mean,axis=1).astype(fl
print ('Final #zeros: %d'%sum(data['Item_Visibility'] == 0))
```

```
Original #zeros: 879
Final #zeros: 0
```

(2) Determine the years of operation of a store

In [112]:
```
#Determine the years of operation of a store
data['Outlet_Years'] = 2013 - data['Outlet_Establishment_Year']
data['Outlet_Years'].describe()
```

Out[112]:
```
count    14204.000000
mean        15.169319
std          8.371664
min          4.000000
25%          9.000000
50%         14.000000
75%         26.000000
max         28.000000
Name: Outlet_Years, dtype: float64
```

(3) Create a broad category of Item_Type

Creating a broad category of Type of Item

To improve our analysis I am going to combine these into broader categories, namely- Food,Non-Consumable,Drinks

In [113]:
```
#Getting the first two characters of ID:
data['Item_Type_Combined'] = data['Item_Identifier'].apply(lambda x: x[0:2])

#Renaming the characters to more intuitive categories:
data['Item_Type_Combined'] = data['Item_Type_Combined'].map({'FD':'Food',
                                                              'NC':'Non-Consumable',
                                                              'DR':'Drinks'})
data['Item_Type_Combined'].value_counts()
```

Out[113]:
```
Food              10201
Non-Consumable     2686
Drinks             1317
Name: Item_Type_Combined, dtype: int64
```

(4) Modify categories of Item_Fat_Content

Modifying categories of Item_Fat_Content

The same kind of categories are represented in different manners,hence I corrected the column names

In [114]:
```
#Changing categories of low fat:
print('Original Categories:')
print(data['Item_Fat_Content'].value_counts())

print('\nModified Categories:')
data['Item_Fat_Content'] = data['Item_Fat_Content'].replace({'LF':'Low Fat',
                                                             'reg':'Regular',
                                                             'low fat':'Low Fat'})

print(data['Item_Fat_Content'].value_counts())
```

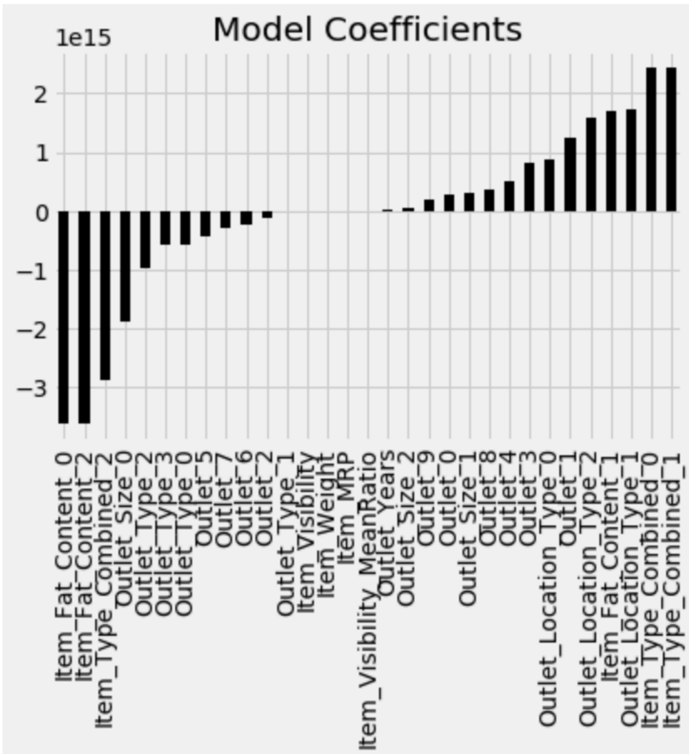
```
Original Categories:
Low Fat    8485
Regular    4824
LF          522
reg         195
low fat     178
Name: Item_Fat_Content, dtype: int64

Modified Categories:
Low Fat    9185
Regular    5019
Name: Item_Fat_Content, dtype: int64
```

Data Model

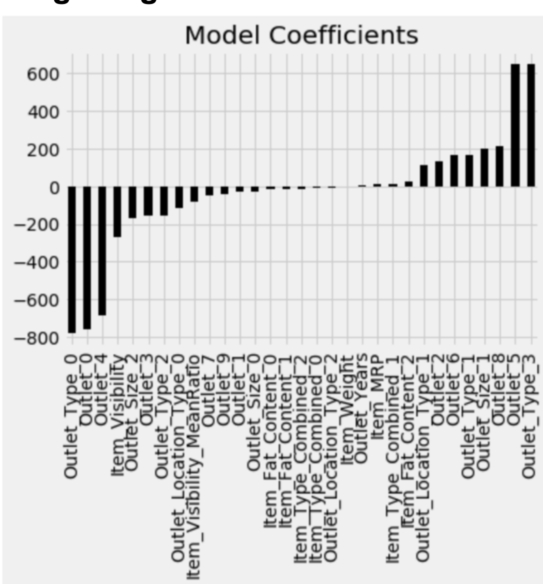
Defining a generic function which takes the algorithm and data as input and makes the model, performs cross-validation and generates submission.

Linear Regression Model



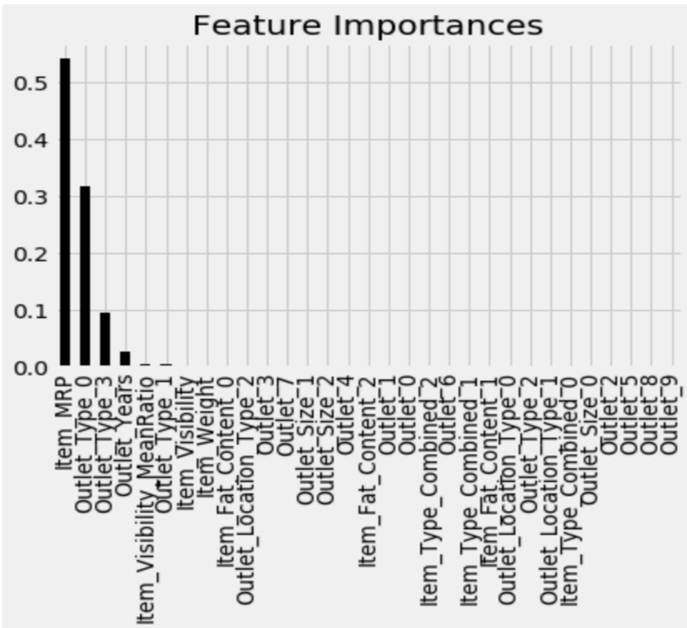
RMSE:1127

Ridge Regression Model



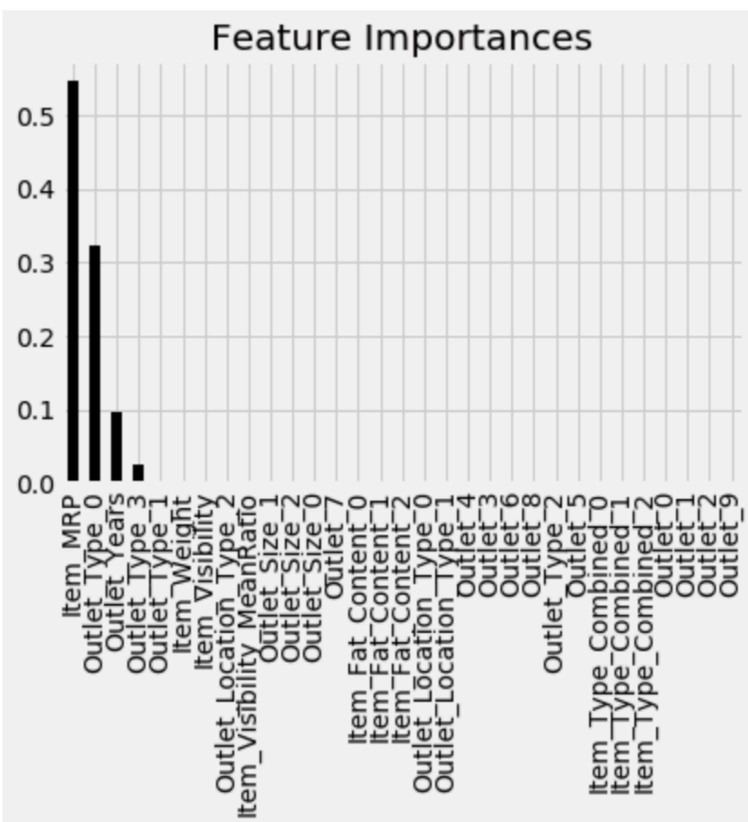
RMSE:1129

Decision Tree Model



RMSE:1058

Random Forest Model



RMSE:1069

XGBoost

RMSE=1052

XGBoost

In [141]:
```
from xgboost import XGBRegressor

my_model = XGBRegressor(n_estimators=1000, learning_rate=0.05)
my_model.fit(train_df[predictors], train_df[target], early_stopping_rounds=5,
             eval_set=[(test_df[predictors], test_df[target])], verbose=False)
```

```
/Applications/anaconda3/lib/python3.7/site-packages/xgboost/core.py:587: FutureWarning: Series.base is deprecated a
nd will be removed in a future version
  if getattr(data, 'base', None) is not None and \
/Applications/anaconda3/lib/python3.7/site-packages/xgboost/core.py:588: FutureWarning: Series.base is deprecated a
nd will be removed in a future version
  data.base is not None and isinstance(data, np.ndarray) \
```

```
[02:41:33] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated in favor of reg:squarederror.
```

Out[141]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.05, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=1000,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

In [156]:
```
#Predicting training set:
train_df_predictions = my_model.predict(train_df[predictors])

#Making predictions
predictions = my_model.predict(test_df[predictors])
```

In [158]:
```
from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error : " + str(mean_absolute_error(predictions, test_df[target])))
print("RMSE : %.4g" % np.sqrt(metrics.mean_squared_error((train_df[target]).values, train_df_predictions)))
IDcol.append(target)
submission = pd.DataFrame({ x: test_df[x] for x in IDcol})
submission.to_csv("XGboost.csv", index=False)
```

```
Mean Absolute Error : 129.9078038223221
RMSE : 1052
```

Results

From the above snippets of various supervised algorithm, we can conclude that XGBoosting regressor performed the best with RMSE=1052. Below results show the results on target variable.

```
In [150]: Output.head(20)
```

Out[150]:

	Item_Identifier	Outlet_Identifier	Item_Outlet_Sales
0	FDW58	OUT049	1509.432313
1	FDW14	OUT017	1364.049154
2	NCN55	OUT010	542.975540
3	FDQ58	OUT017	2384.015126
4	FDY38	OUT027	5669.163351
5	FDH56	OUT046	1874.430960
6	FDL48	OUT018	754.596096
7	FDC48	OUT027	2513.079855
8	FDN33	OUT045	1554.007076
9	FDA36	OUT017	3087.507511
10	FDT44	OUT017	1874.430960
11	FDQ56	OUT045	1364.049154
12	NCC54	OUT019	542.975540
13	FDU11	OUT049	2054.215707
14	DRL59	OUT013	754.596096
15	FDM24	OUT049	2384.015126
16	FDI57	OUT045	2880.969561
17	DRC12	OUT018	2880.969561
18	NCM42	OUT027	3175.335988
19	FDA46	OUT010	542.975540