BIG MART SALES PREDICTION

* A report by Kaivan Gala

**INDEX**

|  |  |
| --- | --- |
| TABLE OF CONTENTS | PAGE NO’S |
| 1. Problem Statement | 3 |
| 1. EDA | 5 |
| 1. Data Preprocessing | 9 |
| 1. Model Selection | 13 |
| APPENDIX | 17 |
|  |  |

**PROBLEM STATEMENT**

The data scientists at BigMart have collected 2013 sales data for 1559 products across 10 stores in different cities. Also, certain attributes of each product and store have been defined. The aim is to build a predictive model and find out the sales of each product at a particular store.

Using this model, BigMart will try to understand the properties of products and stores which play a key role in increasing sales.

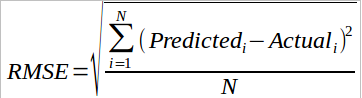
Data

We have train (8523) and test (5681) data set, train data set has both input and output variable(s). You need to predict the sales for test data set.

|  |  |
| --- | --- |
| **Variable** | **Description** |
| **Item\_Identifier** | Unique product ID |
| **Item\_Weight** | Weight of product |
| **Item\_Fat\_Content** | Whether the product is low fat or not |
| **Item\_Visibility** | The % of total display area of all products in a store allocated to the particular product |
| **Item\_Type** | The category to which the product belongs |
| **Item\_MRP** | Maximum Retail Price (list price) of the product |
| **Outlet\_Identifier** | Unique store ID |
| **Outlet\_Establishment\_Year** | The year in which store was established |
| **Outlet\_Size** | The size of the store in terms of ground area covered |
| **Outlet\_Location\_Type** | The type of city in which the store is located |
| **Outlet\_Type** | Whether the outlet is just a grocery store or some sort of supermarket |
| **Item\_Outlet\_Sales** | Sales of the product in the particular store. This is the outcome variable to be predicted. |

**Evaluation Metric:**

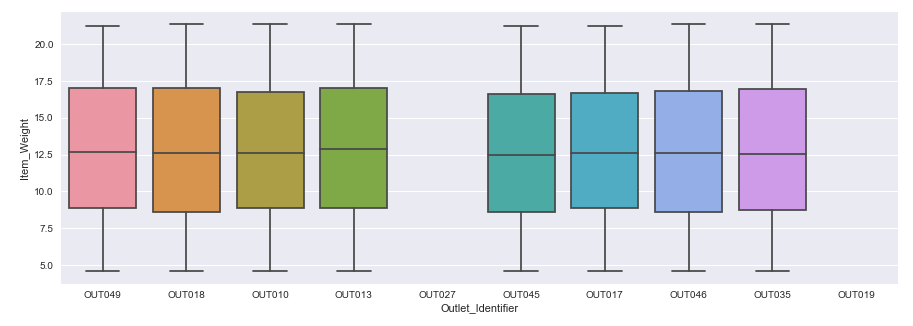
Your model performance will be evaluated based on your prediction of the sales for the test data (test.csv), which contains similar data-points as train except for the sales to be predicted



Where,  
N: total number of observations  
Predicted: the response entered by user  
Actual: actual values of sales

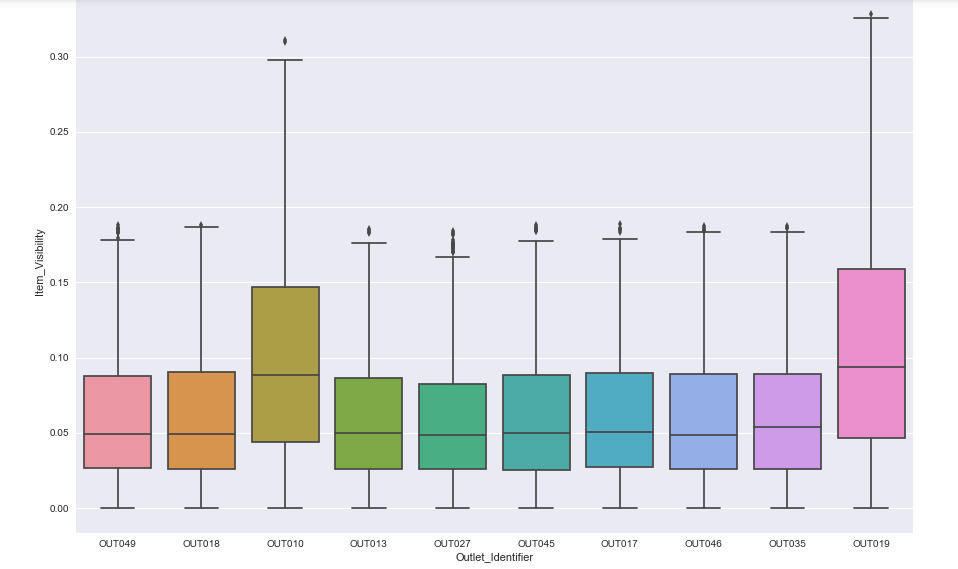
**EDA**

1. Item\_Weight vs Outlet\_Identifier EDA



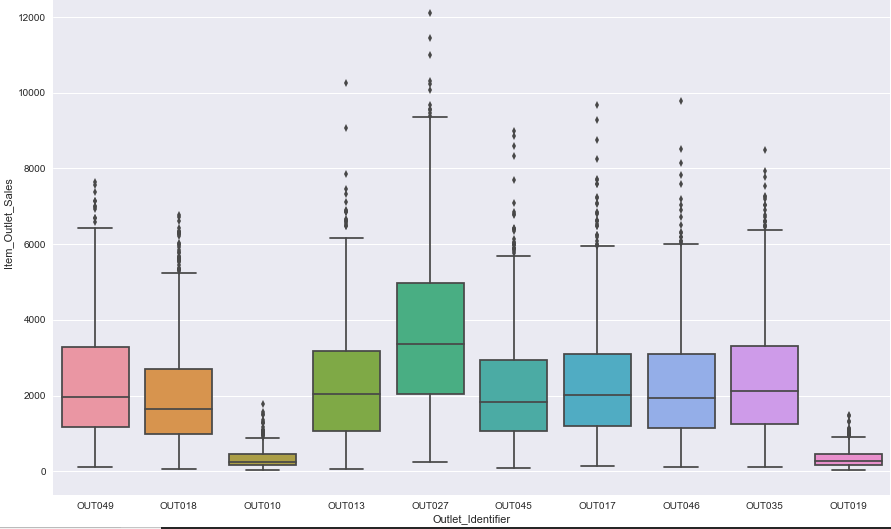
Looking at a boxplot of the weights grouped by the outlet identifier we notice that OUT019 and OUT027 have not reported any weight data

1. Item\_Visibilty vs Outlet\_Identifier Boxplot



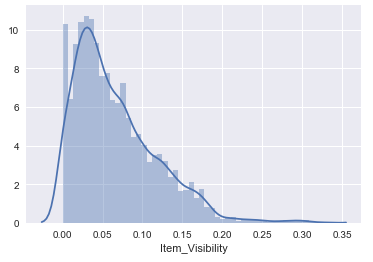
OUT010 and OUT019 have clearly more item visibility and which logically means that these 2 outlets would be smaller in size. Let’s try to prove this true in later stages.

1. Item\_Sales vs Outlet\_Identifier



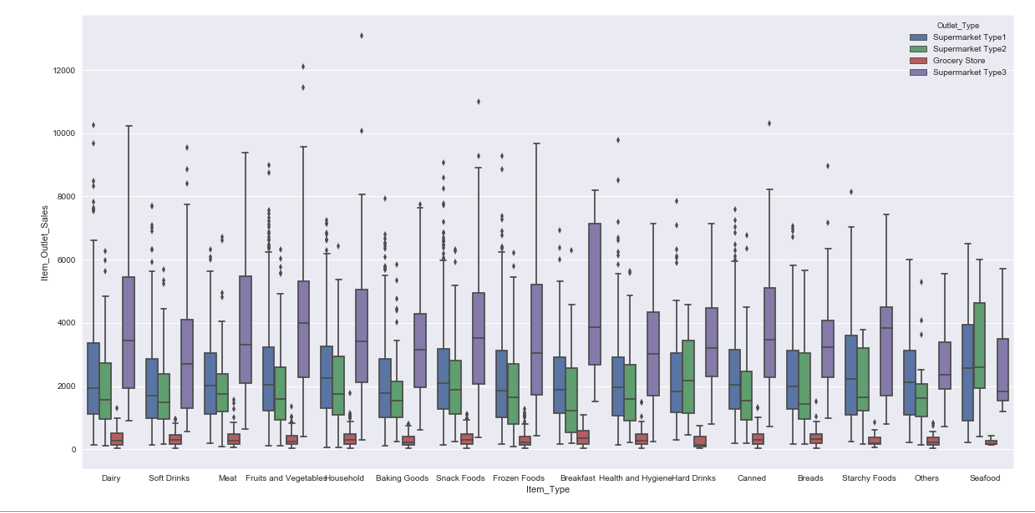
One way to prove that OUT010 and OUT019 are smaller stores is to look at their respective sales which is low when compared to sales of other Outlets.

1. Item\_Visibility Density Plot

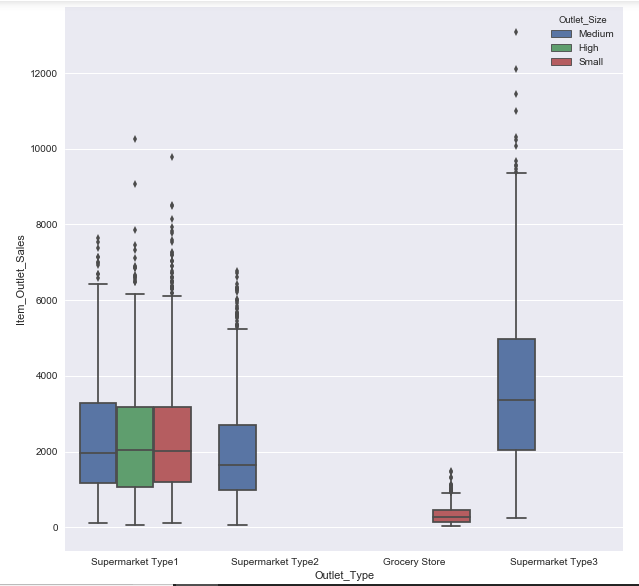


Clearly Item\_Visibility is right skewed, and we would have to apply a transformation to reduce its skewness.

1. Item\_Sales vs Item\_Type vs Outlet\_Type Grouped Boxplot



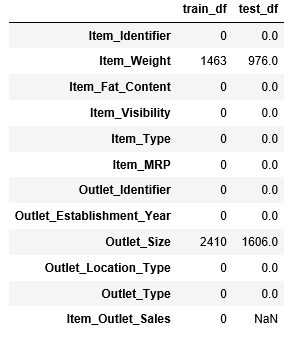
1. Sales vs Outlet Type



It can now be inferred from the plots that OUT010 and OUT019 are small stores. Let’s preprocess the data based on the intuition we have so far.

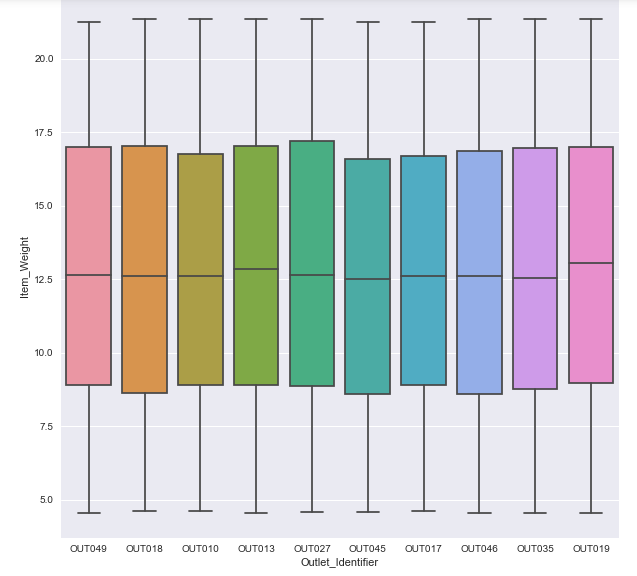
**DATA PREPROCESSING**

Let’s see which all variables have missing values.



Item\_Weight and Outlet\_Size have missing values in almost equal proportion in train and test data set. Let’s impute them.

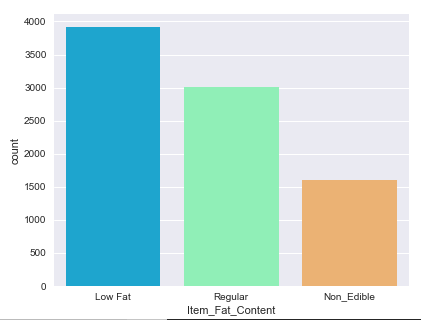
Item weights are missing in OU010 and OUT019 and it seems like these outlets must have forgotten to report their item weights. One thing that is strange is the distribution of item weight is similar across all the other outlets. Let’s assume the distribution of item weight wouldn’t have changed much in OUT010 and OUT019 as well. We will make a similar distribution for those 2 outlets. This can be done by imputing item weight by pointing its corresponding item identifier. So, if an Item ABC weighs 1 pound in a bigger outlet, it would same in the smaller outlet. Now let’s plot the distribution of item weight vs Outlet Identifier



The original data contain five distinct levels for the fat content: LF, low fat, Low Fat, reg, and Regular. Clearly, LF, low fat, and Low Fat are the same, as are reg and Regular. Hence, we replace LF and low fat by Low Fat and reg by Regular.

Further, certain types of non-consumables, i.e. those in the categories Health and Hygiene, Household and Others are either Low Fat or Regular according to the data. Clearly, this makes no sense. Hence, we introduce a new fat level None for non-consumables.

Next, I added new column value Non-Edible for Health and hygiene, household & others



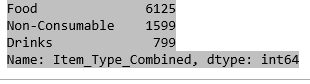
A problem is that plenty of visibilities in the data are 0. Clearly, this is non-sensical. If an item is not physically on display in a store it cannot be sold there. The simplest approach would be to replace those zeroes by the median visibilities. Also, I removed skewness by applying log transformation

Next, let’s create a broad category for Item\_Types

‘FD':'Food',

'NC':'Non-Consumable',

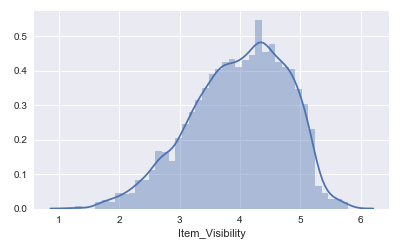
'DR':'Drinks'



Next, let’s replace the Outlet Size by mode of Outlet Size which are grouped by Outlet Type

In the dataset the following were considered as categorical variables, 'Item\_Fat\_Content', 'Item\_Type', 'Outlet\_Location\_Type' ,'Outlet\_Type', 'Outlet\_Size', 'Outlet\_Identifier' and 'Item\_Type\_Combined'

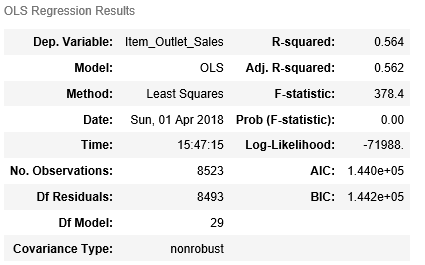
Applying log transformation to item\_visibility gives us the following density plot



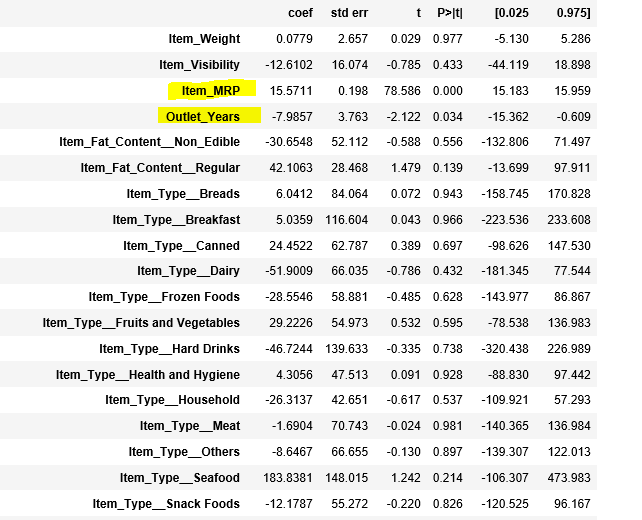
**MODEL BUILDING & MODEL SLECTION**

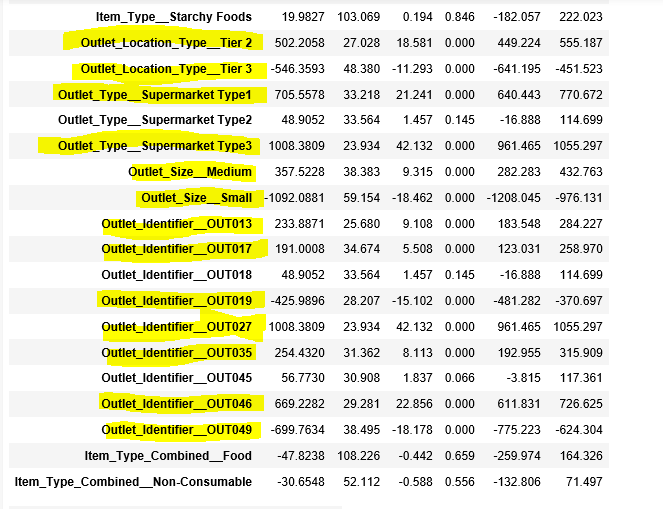
This problem can be solved by using regression techniques. We will you 3 regression techniques and choose the one with least RMSE.

1. Linear Regression
2. Random Forest Regression
3. Extreme Gradient Boosting Regression (XGBoost)
4. Linear Regression:
5. An OLS regression was applied on the dataset. Below is the snapshot of the model summary.



The dependent variable is Item\_outlet\_Sales. OLS was applied on 29 independent variables. And below is statistical result of each independent variable. The R-Squared value is 56% which is a good score. It means, 56% of variance in Outlet Sales is explained by all the independent variables.

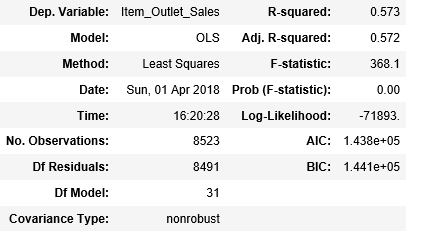




Out of all the independent variables Item\_MRP, Outlet\_years and most of the outlet related attributes are significant and can explain the variation in Outlet sales.

1. Let’s add 2 interaction variables by considering the joint effects of Item\_MRP and Outlet\_Size

Now let’s run an OLS and summarize the results.





The R-squared value increased by 1% and both the interacting variables were significant which means Outlet size and MRP have a collaborative effect in prediction of Outlet Sales.

So, I predicted the Outlet sales by building 3 models. I trained the model on the training dataset and tested it on the test data set which was provided. Below are the results after submitting the test file on to the website.

|  |  |  |
| --- | --- | --- |
| Model Name | Model RMSE | MODEL MEAN CROSS VALIDATION SCORE |
| Linear Regression | 1202.4 | 0.558 |
| Random Forest Regression | 1205.2 | 0.556 |
| XGBoost Regression | 1152.2 | 0.59 |

\*Linear regression was without interaction effects

The model’s mean CV (cross validation) score was calculated based on R square with 10 cross folds. XGBoost clearly had a lower RMSE and a higher CV score.

In the appendix section I have explained more about XGBoost and Random Forest.

**APPENDIX**

What exactly is an XGBoost model all about?

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

The XGBoost library implements the gradient boosting decision tree algorithm. This algorithm goes by lots of different names such as gradient boosting, multiple additive regression trees, stochastic gradient boosting or gradient boosting machines.

Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. A popular example is the Adaptive Boosting algorithm that weights data points that are hard to predict.

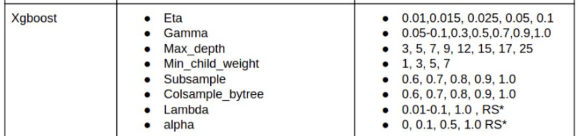
Gradient boosting is an approach where new models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

This approach supports both regression and classification predictive modeling problems.

Advantages of using XGBoost:

1. **Regularization: It is an important concept in machine learning. Regularization helps us prevent over fitting of the data by penalizing model. There are 3 types of regularization:**
2. **L1 Regularization also called as Lasso**
3. L2 Regularization also called as Ridge
4. L1/L2 Regularization also called as Elastic Net
5. Parallel processing: XGBoost implements parallel processing and is **blazingly faster** as compared gradient boosting.
6. **Built-in Cross-Validation:** XGBoost allows user to run a **cross-validation at each iteration** of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.

Some important parameters in XGBoost



The second column shows the names of the parameters and third column shows all the possible value it accepts for an optimum result. This table is very handy while performing parameter tuning.

**Random Forest for Regression**

**Random Forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because it’s simplicity and the fact that it can be used for both classification and regression tasks.**Random forest builds multiple decision trees and merges them to get a more accurate and a stable prediction

**Important Features of Random Forest**

1. Important Feature Selection:

Another excellent quality of the random forest algorithm is that it is very easy to measure the relative importance of each feature on the prediction. Sklearn (Python’s machine learning library) provides a great tool for this, that measures a features importance by looking at how much the tree nodes, which use that feature, reduce impurity across all trees in the forest. It computes this score automatically for each feature after training and scales the results, so that the sum of all importance is equal to 1.

Through looking at the feature importance, you can decide which features you may want to drop, because they don’t contribute enough or nothing to the prediction process. This is important, because a general rule in machine learning is that the more features you have, the more likely your model will suffer from overfitting and vice versa.

### **Important Hyperparameters:**

The parameters in random forest are either used to increase the predictive power of the model or to make the model faster. I will here talk about the hyperparameters of sklearns built-in random forest function.

**1. Increasing the Predictive Power**

Firstly, there is the **“n\_estimators” hyperparameter**, which is just the number of trees the algorithm builds before taking the maximum voting or taking averages of predictions. In general, a higher number of trees increases the performance and makes the predictions more stable, but it also slows down the computation.

Another important hyperparameter is **“max\_features “**, which is the maximum number of features Random Forest can try in an individual tree.

The last important hyper-parameter we will talk about in terms of speed, is **“min\_sample\_leaf “**. This determines, like its name already says, the number of leaves.

**2. Increasing the Models Speed**

The **“n\_jobs“** hyperparameter tells the engine how many processors it can use. If it has a value of 1, it can only use one processor. A value of “-1” means that there is no limit.

**“random\_state**” makes the model’s output replicable. The model will always produce the same results when it has a definite value of random\_state and if it has been given the same parameters and the same training data.

Lastly, there is the **“oob\_score”** (also called oob sampling), which is a random forest cross validation method. In this sampling, about one-third of the data is not used to train the model and can be used to evaluate its performance. These samples are called the out of bag samples. It is very similar to the leave-one-out cross-validation method, but almost no additional computational burden goes along with it.

Finally, I have applied Grid Search as well to find the ideal value for hyper parameters for XGBoost model.