PROJECT IN R

As a part of this Project, install ‘swirl’ package. Then type, *library(swirl)* to initiate the package. Complete this interactive R tutorial. If you have followed this course thoroughly, this assignment should be an easy task for you!

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.

Please note that the data may have missing values as some stores might not report all the data due to technical glitches. Hence, it will be required to treat them accordingly.

Evaluation Metric:

Your model performance will be evaluated on the basis of 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. We at our end, have the actual sales for the test dataset, against which your predictions will be evaluated. We will use the Root Mean Square Error value to judge your response.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

Where,

N: total number of observations

Predicted: the response entered by user

Actual: actual values of sales

Exploratory Data Analysis in R

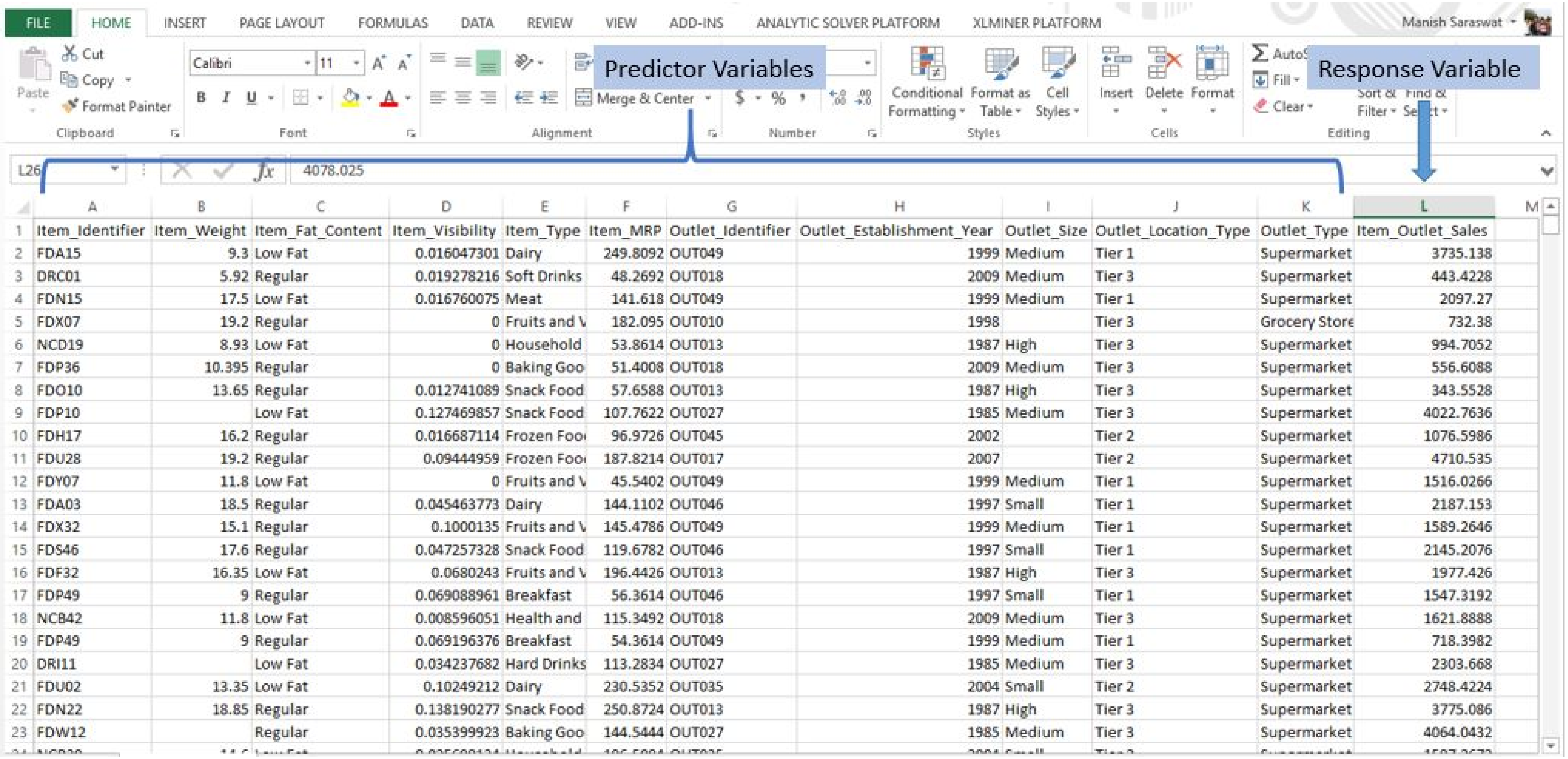
From this section onwards, we'll dive deep into various stages of predictive modeling. Hence, make sure you understand every aspect of this section. In case you find anything difficult to understand, ask me through mail.

Data Exploration is a crucial stage of predictive model. You can't build great and practical models unless you learn to explore the data from begin to end. This stage forms a concrete foundation for data manipulation (the very next stage). Let's understand it in R.

I've taken the data set from Big Mart Sales Prediction, find the data set attached with the mail. Before we start, you must get familiar with these terms:

Response Variable (a.k.a Dependent Variable): In a data set, the response variable (y) is one on which we make predictions. In this case, we'll predict 'Item_Outlet_Sales'. (Refer to image shown below)

Predictor Variable (a.k.a Independent Variable): In a data set, predictor variables (Xi) are those using which the prediction is made on response variable. (Image in next slide).



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_Outlet_Sales
FDA15	9.3	Low Fat	0.016047301	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket	3735.138
DRC01	5.92	Regular	0.019278216	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket	443.4228
FDN15	17.5	Low Fat	0.016760075	Meat	141.618	OUT049	1999	Medium	Tier 1	Supermarket	2097.27
FDX07	19.2	Regular	0	Fruits and V	182.095	OUT010	1998		Tier 3	Grocery Store	732.38
NCD19	8.93	Low Fat	0	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket	994.7052
FDP36	10.395	Regular	0	Baking Goo	51.4008	OUT018	2009	Medium	Tier 3	Supermarket	556.6088
FDO10	13.65	Regular	0.012741089	Snack Food	57.6588	OUT013	1987	High	Tier 3	Supermarket	343.5528
FDP10		Low Fat	0.127469857	Snack Food	107.7622	OUT027	1985	Medium	Tier 3	Supermarket	4022.7636
FDH17	16.2	Regular	0.016687114	Frozen Foo	96.9726	OUT045	2002		Tier 2	Supermarket	1076.5986
FDU28	19.2	Regular	0.09444959	Frozen Foo	187.8214	OUT017	2007		Tier 2	Supermarket	4710.535
FDY07	11.8	Low Fat	0	Fruits and V	45.5402	OUT049	1999	Medium	Tier 1	Supermarket	1516.0266
FDA03	18.5	Regular	0.045463773	Dairy	144.1102	OUT046	1997	Small	Tier 1	Supermarket	2187.153
FDX32	15.1	Regular	0.1000135	Fruits and V	145.4786	OUT049	1999	Medium	Tier 1	Supermarket	1589.2646
FDS46	17.6	Regular	0.047257328	Snack Food	119.6782	OUT046	1997	Small	Tier 1	Supermarket	2145.2076
FDF32	16.35	Low Fat	0.0680243	Fruits and V	196.4426	OUT013	1987	High	Tier 3	Supermarket	1977.426
FDP49	9	Regular	0.069088961	Breakfast	56.3614	OUT046	1997	Small	Tier 1	Supermarket	1547.3192
NCB42	11.8	Low Fat	0.008596051	Health and	115.3492	OUT018	2009	Medium	Tier 3	Supermarket	1621.8888
FDP49	9	Regular	0.069196376	Breakfast	54.3614	OUT049	1999	Medium	Tier 1	Supermarket	718.3982
DRI11		Low Fat	0.034237682	Hard Drinks	113.2834	OUT027	1985	Medium	Tier 3	Supermarket	2303.668
FDU02	13.35	Low Fat	0.10249212	Dairy	230.5352	OUT035	2004	Small	Tier 2	Supermarket	2748.4224
FDN22	18.85	Regular	0.138190277	Snack Food	250.8724	OUT013	1987	High	Tier 3	Supermarket	3775.086
FDW12		Regular	0.035399923	Baking Goo	144.5444	OUT027	1985	Medium	Tier 3	Supermarket	4064.0432

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 particulat store. This is the outcome variable to be predicted.

Train Data: The predictive model is always built on train data set. An intuitive way to identify the train data is, that it always has the ‘response variable’ included.

Test Data: Once the model is built, it’s accuracy is ‘tested’ on test data. This data always contains less number of observations than train data set. Also, it does not include ‘response variable’.

Right now, you should download the data set. Take a good look at train and test data. Cross check the information shared above and then proceed.

Let’s now begin with **importing and exploring data**.

You can import the data directly using the import button or follow the steps below to do manually.

```
#working directory

path <- ".../Data/BigMartSales"

#set working directory

setwd(path)
```

As a beginner, I'll advise you to keep the train and test files in your working directly to avoid unnecessary directory troubles. Once the directory is set, we can easily import the .csv files using commands below.

```
#Load Datasets

train <- read.csv("Train_UWu5bXk.csv")

test <- read.csv("Test_u94Q5KV.csv")
```

In fact, even prior to loading data in R, it's a good practice to look at the data in Excel. This helps in strategizing the complete prediction modeling process. To check if the data set has been loaded successfully, look at R environment. The data can be seen there. Let's explore the data quickly.

```
#check dimesions ( number of row & columns) in data set

> dim(train)

 [1] 8523 12



> dim(test)

[1] 5681 11
```

We have 8523 rows and 12 columns in train data set and 5681 rows and 11 columns in data set. This makes sense. Test data should always have one column less . Let's get deeper in train data set now.

```
#check the variables and their types in train

> str(train)

'data.frame': 8523 obs. of 12 variables:
$ Item_Identifier : Factor w/ 1559 levels "DRA12","DRA24",..: 157 9 663 1122 1298 759
697 739 441 991 ...
$ Item_Weight : num 9.3 5.92 17.5 19.2 8.93 ...
$ Item_Fat_Content : Factor w/ 5 levels "LF","low fat",..: 3 5 3 5 3 5 5 3 5 5 ...
$ Item_Visibility : num 0.016 0.0193 0.0168 0 0 ...
$ Item_Type : Factor w/ 16 levels "Baking Goods",..: 5 15 11 7 10 1 14 14 6 6 ...
$ Item_MRP : num 249.8 48.3 141.6 182.1 53.9 ...
$ Outlet_Identifier : Factor w/ 10 levels "OUT010","OUT013",..: 10 4 10 1 2 4 2 6 8 3
...
$ Outlet_Establishment_Year: int 1999 2009 1999 1998 1987 2009 1987 1985 2002 2007 ...
$ Outlet_Size : Factor w/ 4 levels "","High","Medium",..: 3 3 3 1 2 3 2 3 1 1 ...
$ Outlet_Location_Type : Factor w/ 3 levels "Tier 1","Tier 2",..: 1 3 1 3 3 3 3 3 2 2
...
$ Outlet_Type : Factor w/ 4 levels "Grocery Store",..: 2 3 2 1 2 3 2 4 2 2 ...
$ Item_Outlet_Sales : num 3735 443 2097 732 995 ...
```

To begin with, I'll first check if this data has missing values. This can be done by using:

```
> table(is.na(train))

FALSE TRUE

100813 1463
```

In train data set, we have 1463 missing values. Let's check the variables in which these values are missing. It's important to find and locate these missing values. Many data scientists have repeatedly advised beginners to pay close attention to missing value in data exploration stages.

```
> colSums(is.na(train))

Item_Identifier Item_Weight
0                 1463
Item_Fat_Content Item_Visibility
0                 0
Item_Type         Item_MRP
0                 0
Outlet_Identifier Outlet_Establishment_Year
0                 0
Outlet_Size       Outlet_Location_Type
0                 0
Outlet_Type       Item_Outlet_Sales
0                 0
```

Hence, we see that column Item_Weight has 1463 missing values. Let's get more inferences from this data.

```
> summary(train)
```

Here are some quick inferences drawn from variables in train data set:

1. Item_Fat_Content has mismatched factor levels.
2. Minimum value of item_visibility is 0. Practically, this is not possible. If an item occupies shelf space in a grocery store, it ought to have some visibility. We’ll treat all 0’s as missing values.
3. Item_Weight has 1463 missing values (already explained above).
4. Outlet_Size has a unmatched factor levels.

These inference will help us in treating these variable more accurately.

Graphical Representation of Variables

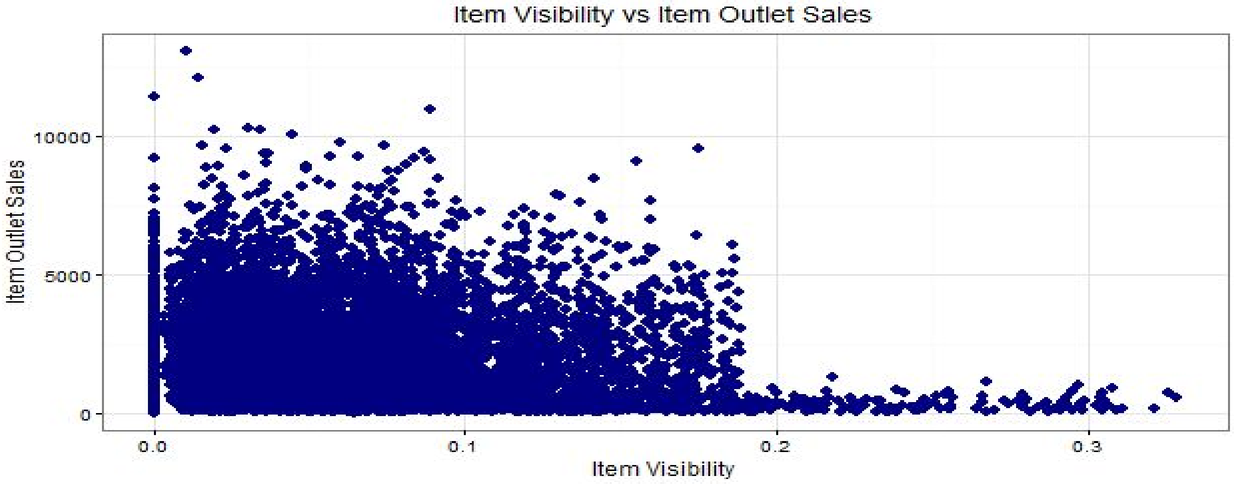
I'm sure you would understand these variables better when explained visually. Using graphs, we can analyze the data in 2 ways: Univariate Analysis and Bivariate Analysis.

Univariate analysis is done with one variable. Bivariate analysis is done with two variables. Univariate analysis is a lot easy to do. Hence, I'll skip that part here. I'd recommend you to try it at your end. Let's now experiment doing bivariate analysis and carve out hidden insights.

For visualization, I'll use ggplot2 package.

These graphs would help us understand the distribution and frequency of variables in the data set.

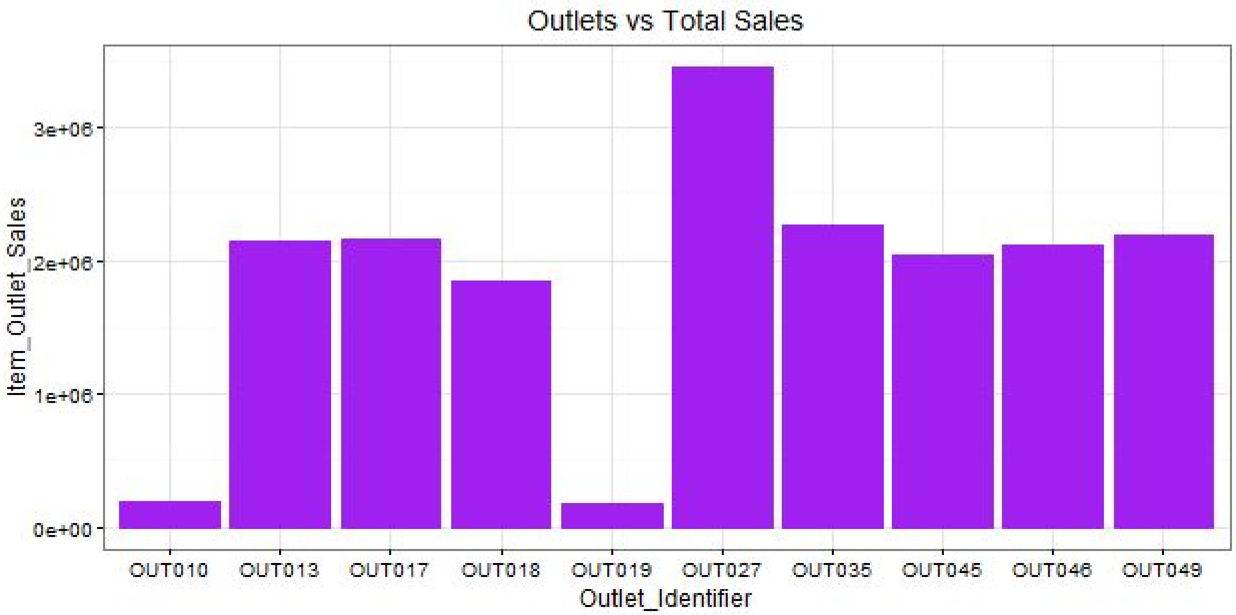
```
> ggplot(train, aes(x= Item_Visibility, y = Item_Outlet_Sales)) + geom_point(size = 2.5, color="navy") +
xlab("Item Visibility") + ylab("Item Outlet Sales") + ggtitle("Item Visibility vs Item Outlet Sales")
```



We can see that majority of sales has been obtained from products having visibility less than 0.2. This suggests that item_visibility < 2 must be an important factor in determining sales.

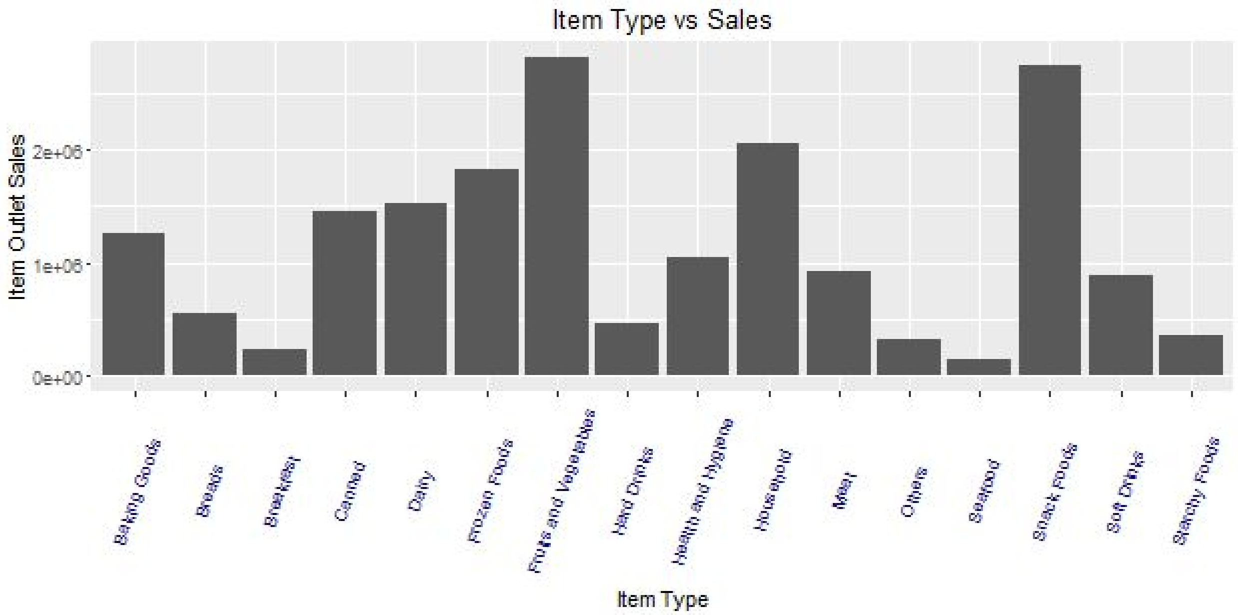
Let’s plot few more interesting graphs and explore such hidden stories.

```
> ggplot(train, aes(Outlet_Identifier, Item_Outlet_Sales)) + geom_bar(stat = "identity", color = "purple")
+theme(axis.text.x = element_text(angle = 70, vjust = 0.5, color = "black"))  + ggtitle("Outlets vs Total Sales")
+ theme_bw()
```



Here, we infer that OUT027 has contributed to majority of sales followed by OUT35. OUT10 and OUT19 have probably the least footfall, thereby contributing to the least outlet sales.

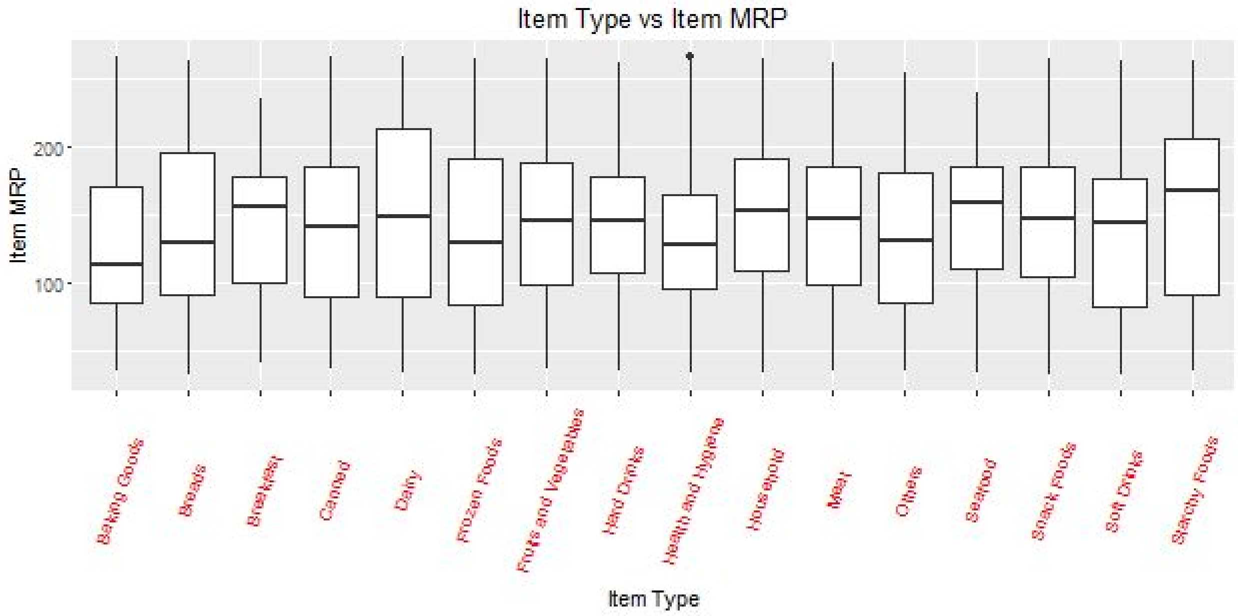
```
> ggplot(train, aes(Item_Type, Item_Outlet_Sales)) + geom_bar( stat = "identity") +theme(axis.text.x = element_text(angle = 70, vjust = 0.5, color = "navy")) + xlab("Item Type") + ylab("Item Outlet Sales")+ggtitle("Item Type vs Sales")
```



From this graph, we can infer that Fruits and Vegetables contribute to the highest amount of outlet sales followed by snack foods and household products. This information can also be represented using a box plot chart.

The benefit of using a box plot is, you get to see the outlier and mean deviation of corresponding levels of a variable (shown below).

```
> ggplot(train, aes(Item_Type, Item_MRP)) +geom_boxplot() +ggtitle("Box Plot") + theme(axis.text.x = element_text(angle = 70, vjust = 0.5, color = "red")) + xlab("Item Type") + ylab("Item MRP") + ggtitle("Item Type vs Item MRP")
```



The black point you see, is an outlier. The mid line you see in the box, is the mean value of each item type.

Now, we have an idea of the variables and their importance on response variable. Let's now move back to where we started. Missing values. Now we'll impute the missing values.

We saw variable Item_Weight has missing values. Item_Weight is an continuous variable. Hence, in this case we can impute missing values with mean / median of item_weight. These are the most commonly used methods of imputing missing value

Let's first combine the data sets. This will save our time as we don't need to write separate codes for train and test data sets. To combine the two data frames, we must make sure that they have equal columns, which is not the case.

```
> dim(train)

 [1] 8523 12

> dim(test)

[1] 5681 11
```

Test data set has one less column (response variable). Let's first add the column. We can give this column any value. An intuitive approach would be to extract the mean value of sales from train data set and use it as placeholder for test variable Item _Outlet_ Sales. Anyways, let's make it simple for now. I've taken a value 1. Now, we'll combine the data sets.

```
> test$Item_Outlet_Sales <-  1
```

```
> combi <- rbind(train, test)
```

Impute missing value by median. I'm using median because it is known to be highly robust to outliers. Moreover, for this problem, our evaluation metric is RMSE which is also highly affected by outliers. Hence, median is better in this case.

```
> combi$Item_Weight[is.na(combi$Item_Weight)] <- median(combi$Item_Weight, na.rm = TRUE)
```

```
> table(is.na(combi$Item_Weight))
```

```
FALSE
```

```
14204
```

Trouble with Continuous Variables & Categorical Variables

It's important to learn to deal with continuous and categorical variables separately in a data set. In other words, they need special attention. In this data set, we have only 3 continuous variables and rest are categorical in nature. If you are still confused, I'll suggest you to once again look at the data set using *str()* and proceed.

Let's take up *Item_Visibility*. In the graph above, we saw item visibility has zero value also, which is practically not feasible. Hence, we'll consider it as a missing value and once again make the imputation using median.

```
> combi$Item_Visibility <- ifelse(combi$Item_Visibility == 0,

                          median(combi$Item_Visibility), combi$Item_Visibility)
```

Let's proceed to categorical variables now. During exploration, we saw there are mis-matched levels in variables which needs to be corrected.

```
> levels(combi$Outlet_Size)[1] <- "Other"

> library(plyr)

> combi$Item_Fat_Content <- revalue(combi$Item_Fat_Content,

c("LF" = "Low Fat", "reg" = "Regular"))

> combi$Item_Fat_Content <- revalue(combi$Item_Fat_Content, c("low fat" = "Low Fat"))

> table(combi$Item_Fat_Content)

  Low Fat Regular

  9185    5019
```

Using the commands above, I've assigned the name 'Other' to unnamed level in *Outlet_Size* variable. Rest, I've simply renamed the various levels of Item_Fat_Content.

4. Data Manipulation in R

Let's call it as, the advanced level of data exploration. In this section we'll practically learn about feature engineering and other useful aspects.

Feature Engineering: This component separates an intelligent data scientist from a technically enabled data scientist. You might have access to large machines to run heavy computations and algorithms, but the power delivered by new features, just can't be matched. We create new variables to extract and provide as much 'new' information to the model, to help it make accurate predictions.

If you have been thinking all this time, great. But now is the time to think deeper. Look at the data set and ask yourself, what else (factor) could influence Item_Outlet_Sales ? Anyhow, the answer is below. But, I want you to try it out first, before scrolling down.

1. **Count of Outlet Identifiers** – There are 10 unique outlets in this data. This variable will give us information on count of outlets in the data set. More the number of counts of an outlet, chances are more will be the sales contributed by it.

```
> library(dplyr)

> a <- combi%>%

            group_by(Outlet_Identifier)%>%

            tally()

 > head(a)

Source: local data frame [6 x 2] Outlet_Identifier n

(fctr)           (int)

1 OUT010         925

2 OUT013         1553

3 OUT017         1543

4 OUT018         1546

5 OUT019         880

6 OUT027         1559
```

```
> names(a)[2] <- "Outlet_Count"

> combi <- full_join(a, combi, by = "Outlet_Identifier")
```

As you can see, dplyr package makes data manipulation quite effortless. You no longer need to write long function. In the code above, I've simply stored the new data frame in a variable *a.* Later, the new column *Outlet_ Count* is added in our original 'combi' data set. To know more about dplyr, follow this tutorial.

2. **Count of Item Identifiers** – Similarly, we can compute count of item identifiers too. It's a good practice to fetch more information from unique ID variables using their count. This will help us to understand, which outlet has maximum frequency.

```
> b <- combi%>%

          group_by(Item_Identifier)%>%

          tally()

> names(b)[2] <- "Item_Count"

> head (b)
```

```
  Item_Identifier   Item_Count
  (fctr)            (int)
1 DRA12               9
2 DRA24              10
3 DRA59              10
4 DRB01               8
5 DRB13               9
6 DRB24               8
```

> combi <- merge(b, combi, by = “Item_Identifier”)

3. **Outlet Years** – This variable represent the information of existence of a particular outlet since year 2013. Why just 2013? You'll find the answer in problem statement. My hypothesis is, older the outlet, more footfall, large base of loyal customers and larger the outlet sales.

```
> c <- combi%>%
           select(Outlet_Establishment_Year)%>%
           mutate(Outlet_Year = 2013 - combi$Outlet_Establishment_Year)
 > head(c)
Outlet_Establishment_Year  Outlet_Year
1 1999                              14
2 2009                               4
3 1999                              14
4 1998                              15
5 1987                              26
6 2009                               4
> combi <- full_join(c, combi)
```

This suggests that outlets established in 1999 were 14 years old in 2013 and so on.

4. **Item Type New** – Now, pay attention to *Item_Identifiers.* We are about to discover a new trend. Look carefully, there is a pattern in the identifiers starting with "FD","DR","NC". Now, check the corresponding *Item_ Types* to these identifiers in the data set. You'll discover, items corresponding to "DR", are mostly eatables. Items corresponding to "FD", are drinks. And, item corresponding to "NC", are products which can't be consumed, let's call them non-consumable. Let's extract these variables into a new variable representing their counts.

Here I'll use *substr()*, *gsub()* function to extract and rename the variables respectively.

```
> q <- substr(combi$Item_Identifier,1,2)
> q <- gsub("FD","Food",q)
> q <- gsub("DR","Drinks",q)
> q <- gsub("NC","Non-Consumable",q)
> table(q)
   Drinks Food  Non-Consumable
   1317   10201 2686
```

Let's now add this information in our data set with a variable name 'Item_Type_New.

```
> combi$Item_Type_New <- q
```

I'll leave the rest of feature engineering intuition to you. You can think of more variables which could add more information to the model. But make sure, the variable aren't correlated. Since, they are emanating from a same set of variable, there is a high chance for them to be correlated. You can check the same in R using *cor()* function.

Label Encoding and One Hot Encoding

Just, one last aspect of feature engineering left. Label Encoding and One Hot Encoding.

Label Encoding, in simple words, is the practice of numerically encoding (replacing) different levels of a categorical variables. For example: In our data set, the variable *Item_ Fat_ Content* has 2 levels: Low Fat and Regular. So, we'll encode Low Fat as 0 and Regular as 1. This will help us convert a factor variable in numeric variable. This can be simply done using if else statement in R.

```
> combi$Item_Fat_Content <- ifelse(combi$Item_Fat_Content == "Regular",1,0)
```

One Hot Encoding, in simple words, is the splitting a categorical variable into its unique levels, and eventually removing the original variable from data set. Confused ? Here's an example: Let's take any categorical variable, say, *Outlet_ Location_ Type*. It has 3 levels. One hot encoding of this variable, will create 3 different variables consisting of 1s and 0s. 1s will represent the existence of variable and 0s will represent non-existence of variable. Let look at a sample on next slide

```
> sample <- select(combi, Outlet_Location_Type)
> demo_sample <- data.frame(model.matrix(~.-1,sample))
> head(demo_sample)
Outlet_Location_TypeTier.1 Outlet_Location_TypeTier.2 Outlet_Location_TypeTier.3
1             1                          0                          0
2             0                          0                          1
3             1                          0                          0
4             0                          0                          1
5             0                          0                          1
6             0                          0                          1
```

model.matrix creates a matrix of encoded variables. ~. -1 tells R, to encode all variables in the data frame, but suppress the intercept. So, what will happen if you don't write -1 ? model.matrix will skip the first level of the factor, thereby resulting in just 2 out of 3 factor levels (loss of information).

This was the demonstration of one hot encoding. Hope you have understood the concept now.

Let’s now apply this technique to all categorical variables in our data set (excluding ID variable).

```
>library(dummies)
```

```
>combi <- dummy.data.frame(combi, names = c('Outlet_Size','Outlet_Location_Type','Outlet_Type', 'Item_Type_New'),
sep='_')
```

With this, I have shared 2 different methods of performing one hot encoding in R. Let’s check if encoding has been done.

```
> str (combi)
```

```
$ Outlet_Size_Other : int 0 1 1 0 1 0 0 0 0 0 ...
$ Outlet_Size_High : int 0 0 0 1 0 0 0 0 0 0 ...
$ Outlet_Size_Medium : int 1 0 0 0 0 0 1 1 0 1 ...
$ Outlet_Size_Small : int 0 0 0 0 0 1 0 0 1 0 ...
$ Outlet_Location_Type_Tier 1 : int 1 0 0 0 0 0 0 0 1 0 ...
$ Outlet_Location_Type_Tier 2 : int 0 1 0 0 1 1 0 0 0 0 ...
$ Outlet_Location_Type_Tier 3 : int 0 0 1 1 0 0 1 1 0 1 ...
$ Outlet_Type_Grocery Store : int 0 0 1 0 0 0 0 0 0 0 ...
$ Outlet_Type_Supermarket Type1: int 1 1 0 1 1 1 0 0 1 0 ...
$ Outlet_Type_Supermarket Type2: int 0 0 0 0 0 0 0 1 0 0 ...
$ Outlet_Type_Supermarket Type3: int 0 0 0 0 0 0 1 0 0 1 ...
$ Item_Outlet_Sales : num 1 3829 284 2553 2553 ...
$ Year : num 14 11 15 26 6 9 28 4 16 28 ...
$ Item_Type_New_Drinks : int 1 1 1 1 1 1 1 1 1 1 ...
$ Item_Type_New_Food : int 0 0 0 0 0 0 0 0 0 0 ...
$ Item_Type_New_Non-Consumable : int 0 0 0 0 0 0 0 0 0 0 ...
```

As you can see, after one hot encoding, the original variables are removed automatically from the data set.

5. Predictive Modeling using Machine Learning

Finally, we'll drop the columns which have either been converted using other variables or are identifier variables. This can be accomplished using *select* from dplyr package.

```
> combi <- select(combi, -c(Item_Identifier, Outlet_Identifier, Item_Fat_Content,
Outlet_Establishment_Year,Item_Type))
```

```
> str(combi)
```

In this section, I'll cover Regression, Decision Trees and Random Forest. A detailed explanation of these algorithms is provided in links attached. These algorithms have been satisfactorily explained in the links for useful resources.

As you can see, we have encoded all our categorical variables. Now, this data set is good to take forward to modeling. Since, we started from Train and Test, let's now divide the data sets.

```
> new_train <- combi[1:nrow(train),]
```

```
> new_test <- combi[-(1:nrow(train)),]
```

Linear (Multiple) Regression

Multiple Regression is used when response variable is continuous in nature and predictors are many. Had it been categorical, we would have used Logistic Regression. Before you proceed, sharpen your basics of Regression here.

Linear Regression takes following assumptions:

1. There exists a linear relationship between response and predictor variables
2. The predictor (independent) variables are not correlated with each other. Presence of collinearity leads to a phenomenon known as multicollinearity.
3. The error terms are uncorrelated. Otherwise, it will lead to autocorrelation.
4. Error terms must have constant variance. Non-constant variance leads to heteroskedasticity.

Let's now build out first regression model on this data set. R uses *lm()* function for regression.

```
> linear_model <- lm(Item_Outlet_Sales ~ ., data = new_train)
```

```
> summary(linear_model)
```

Adjusted R^2 measures the goodness of fit of a regression model. Higher the R^2, better is the model. Our $\mathbf{R^2 = 0.2085}$. It means we really did something drastically wrong. Let's figure it out.

In our case, I could find our new variables aren't helping much i.e. Item count, Outlet Count and Item_Type_New. Neither of these variables are significant. Significant variables are denoted by '*' sign.

As we know, correlated predictor variables brings down the model accuracy. Let's find out the amount of correlation present in our predictor variables. This can be simply calculated using:

```
> cor(new_train)
```

Alternatively, you can also use corrplot package for some fancy correlation plots. Scrolling through the long list of correlation coefficients, I could find a deadly correlation coefficient:

```
cor(new_train$Outlet_Count, new_train$`Outlet_Type_Grocery Store`)
```

```
[1] -0.9991203
```

Outlet_Count is highly correlated (negatively) with Outlet Type Grocery Store. Here are some problems I could find in this model:

1. We have correlated predictor variables.
2. We did one hot encoding and label encoding. That's not necessary since linear regression handle categorical variables by creating dummy variables intrinsically.

3. The new variables (item count, outlet count, item type new) created in feature engineering are not significant.

Let's try to create a more robust regression model. This time, I'll be using a building a simple model without encoding and new features. Below is the entire code:

```
#load directory
> path <- "C:/Users/manish/desktop/Data/February 2016"
> setwd(path)
#load data
> train <- read.csv("train_Big.csv")
> test <- read.csv("test_Big.csv")
#create a new variable in test file
> test$Item_Outlet_Sales <- 1
#combine train and test data
> combi <- rbind(train, test)
impute missing value in Item_Weight
> combi$Item_Weight[is.na(combi$Item_Weight)] <- median(combi$Item_Weight, na.rm = TRUE)
#impute 0 in item_visibility
> combi$Item_Visibility <- ifelse(combi$Item_Visibility == 0, median(combi$Item_Visibility),
combi$Item_Visibility)
```

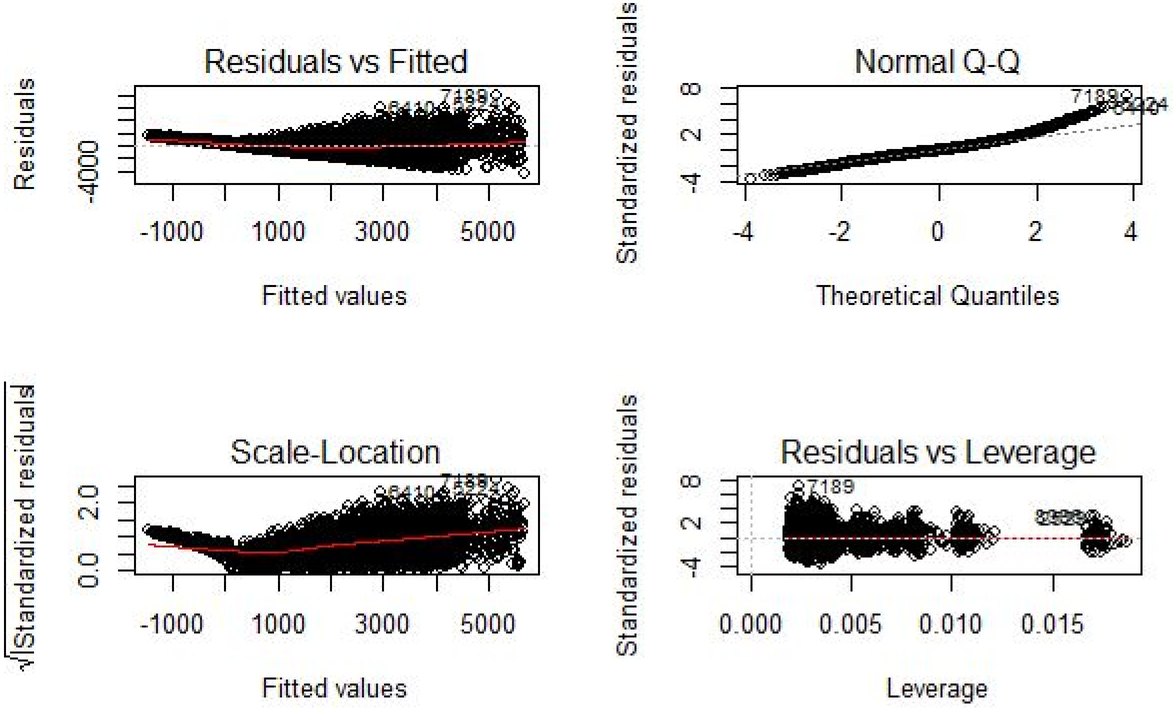
```
##rename level in Outlet_Size
> levels(combi$Outlet_Size)[1] <- "Other"
#rename levels of Item_Fat_Content
> library(plyr)
> combi$Item_Fat_Content <- revalue(combi$Item_Fat_Content,c("LF" = "Low Fat", "reg" =
"Regular"))
> combi$Item_Fat_Content <- revalue(combi$Item_Fat_Content, c("low fat" = "Low Fat"))
#create a new column 2013 - Year
> combi$Year <- 2013 - combi$Outlet_Establishment_Year
#drop variables not required in modeling
> library(dplyr)
> combi <- select(combi, -c(Item_Identifier, Outlet_Identifier, Outlet_Establishment_Year))
#divide data set
> new_train <- combi[1:nrow(train),]
> new_test <- combi[-(1:nrow(train)),]
#linear regression
> linear_model <- lm(Item_Outlet_Sales ~ ., data = new_train)
> summary(linear_model)
```

Now we have got **R^2 = 0.5623**. This teaches us that, sometimes all you need is simple thought process to get high accuracy. Quite a good improvement from previous model. Next, time when you work on any model, always remember to start with a simple model.

Let's check out regression plot to find out more ways to improve this model.

```
> par(mfrow=c(2,2))
```

```
> plot(linear_model)
```



You can zoom these graphs in R Studio at your end. All these plots have a different story to tell. But the most important story is being portrayed by Residuals vs Fitted graph.

Residual values are the difference between actual and predicted outcome values. Fitted values are the predicted values. If you see carefully, you'll discover it as a funnel shape graph (from right to left). The shape of this graph suggests that our model is suffering from heteroskedasticity (unequal variance in error terms). Had there been constant variance, there would be no pattern visible in this graph.

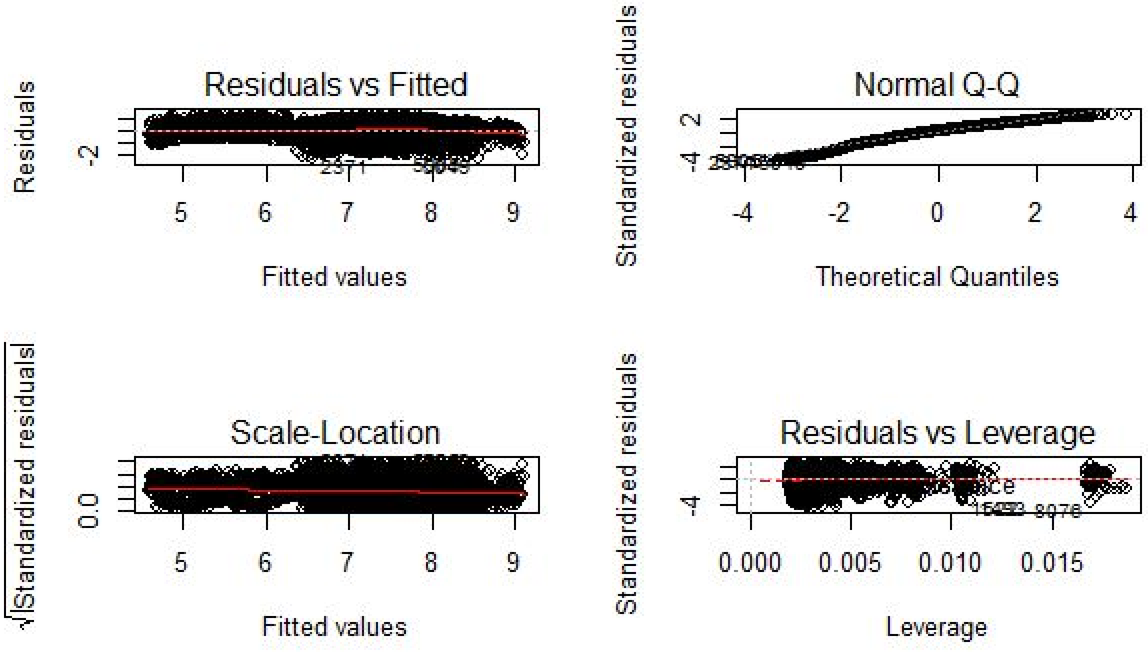
A common practice to tackle heteroskedasticity is by taking the log of response variable. Let's do it and check if we can get further improvement.

```
> linear_model <- lm(log(Item_Outlet_Sales) ~ ., data = new_train)
> summary(linear_model)
```

And, here's a snapshot of my model output. Congrats! We have got an improved model with **R^2 = 0.72**. Now, we are on the right path.

```
Item_TypeSnack Foods
Item_TypeSoft Drinks
Item_TypeStarchy Foods
Item_MRP                        ***
Outlet_SizeHigh                 **
Outlet_SizeMedium               *
Outlet_SizeSmall                **
Outlet_Location_TypeTier 2
Outlet_Location_TypeTier 3      *
Outlet_TypeSupermarket Type1    ***
Outlet_TypeSupermarket Type2    ***
Outlet_TypeSupermarket Type3    ***
Year                            **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5378 on 8494 degrees of freedom
Multiple R-squared:  0.7214,    Adjusted R-squared:  0.7205
F-statistic: 785.4 on 28 and 8494 DF,  p-value: < 2.2e-16
```

Once again you can check the residual plots (you might zoom it). You'll find there is no longer a trend in residual vs fitted value plot.



This model can be further improved by detecting outliers and high leverage points. For now, I leave that part to you! I shall write a separate post on mysteries of regression soon. For now, let’s check our RMSE so that we can compare it with other algorithms demonstrated below.

To calculate RMSE, we can load a package named *Metrics.*

```
> install.packages("Metrics")

> library(Metrics)

> rmse(new_train$Item_Outlet_Sales, exp(linear_model$fitted.values))

[1] 1140.004
```

Let’s proceed to decision tree algorithm and try to improve our RMSE score.

Decision Trees

Before you start, I'd recommend you to glance through the basics of decision tree algorithms. To understand what makes it superior than linear regression, check this tutorial Part 1 and Part 2.

In R, decision tree algorithm can be implemented using *rpart package.* In addition, we'll use *caret package* for doing cross validation. Cross validation is a technique to build robust models which are not prone to overfitting. Read more about Cross Validation.

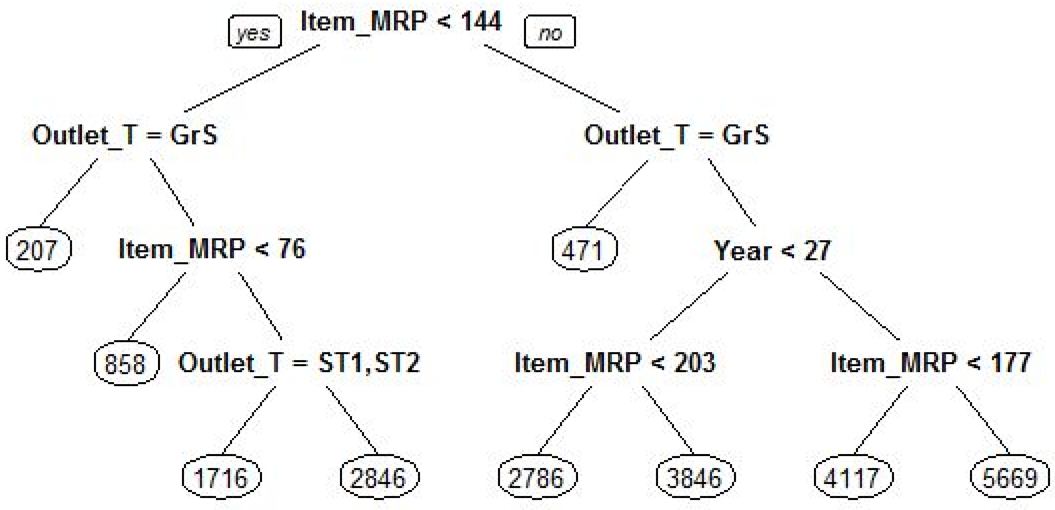
In R, decision tree uses a complexity parameter (*cp*). It measures the tradeoff between model complexity and accuracy on training set. A smaller cp will lead to a bigger tree, which might overfit the model. Conversely, a large cp value might underfit the model. Underfitting occurs when the model does not capture underlying trends properly. Let's find out the optimum cp value for our model with 5 fold cross validation.

```
#loading required libraries

> library(rpart)

> library(e1071)

> library(rpart.plot)

> library(caret)

#setting the tree control parameters

> fitControl <- trainControl(method = "cv", number = 5)

> cartGrid <- expand.grid(.cp=(1:50)*0.01)

#decision tree

> tree_model <- train(Item_Outlet_Sales ~ ., data = new_train, method = "rpart", trControl = fitControl, tuneGrid
= cartGrid)

> print(tree_model)
```

The final value for cp = 0.01. You can also check the table populated in console for more information. The model with cp = 0.01 has the least RMSE. Let’s now build a decision tree with 0.01 as complexity parameter.

```
> main_tree <- rpart(Item_Outlet_Sales ~ ., data = new_train, control = rpart.control(cp=0.01))
```

```
> prp(main_tree)
```



Here is the tree structure of our model. If you have gone through the basics, you would now understand that this algorithm has marked Item_MRP as the most important variable (being the root node). Let's check the RMSE of this model and see if this is any better than regression.

```
> pre_score <- predict(main_tree, type = "vector")

> rmse(new_train$Item_Outlet_Sales, pre_score)

[1] 1102.774
```

As you can see, our RMSE has further improved from 1140 to 1102.77 with decision tree. To improve this score further, you can further tune the parameters for greater accuracy.

Random Forest

Random Forest is a powerful algorithm which holistically takes care of missing values, outliers and other non-linearities in the data set. It's simply a collection of classification trees, hence the name 'forest'. I'd suggest you to quickly refresh your basics of random forest with this tutorial.

In R, random forest algorithm can be implement using *randomForest* package. Again, we'll use train package for cross validation and finding optimum value of model parameters.

For this problem, I'll focus on two parameters of random forest. *mtry* and *ntree*. ntree is the number of trees to be grown in the forest. *mtry* is the number of variables taken at each node to build a tree. And, we'll do a 5 fold cross validation.

```
#load randomForest library

> library(randomForest)

#set tuning parameters

> control <- trainControl(method = "cv", number = 5)

#random forest model

> rf_model <- train(Item_Outlet_Sales ~ ., data = new_train, method = "parRF", trControl =
control, prox = TRUE, allowParallel = TRUE)

#check optimal parameters

> print(rf_model)
```

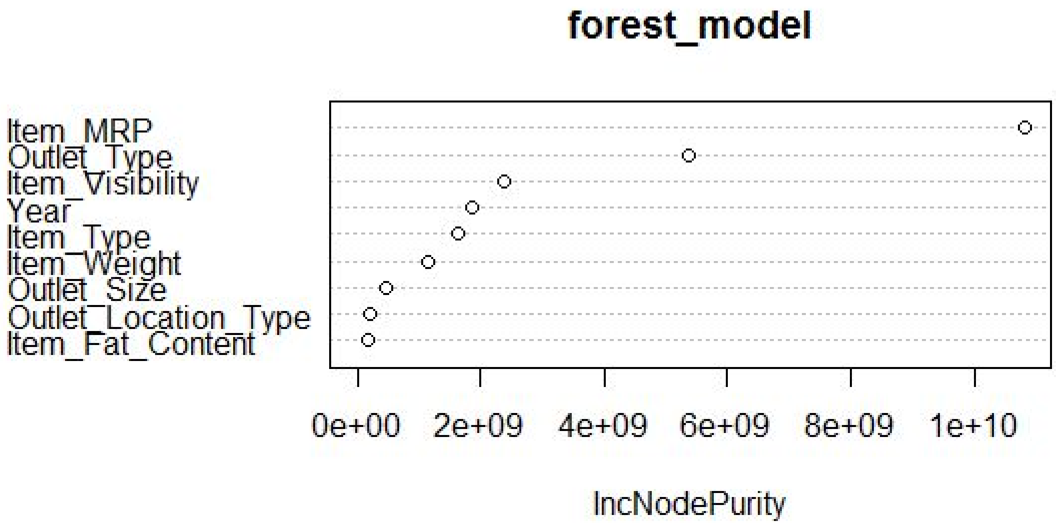
If you notice, you'll see I've used method = "parRF". This is parallel random forest. This is parallel implementation of random forest. This package causes your local machine to take less time in random forest computation. Alternatively, you can also use method = "rf" as a standard random forest function.

```
Parallel Random Forest

8523 samples
   9 predictor

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 6819, 6818, 6818, 6819, 6818
Resampling results across tuning parameters:

  mtry  RMSE      Rsquared   RMSE SD    Rsquared SD
   2    1293.032  0.5157854   7.845358  0.01852388
  15    1122.530  0.5697400  16.109357  0.01307624
  28    1135.780  0.5613481  16.825174  0.01294525

RMSE was used to select the optimal model using  the smallest value.
The final value used for the model was mtry = 15.
```

Now we've got the optimal value of mtry = 15. Let's use 1000 trees for computation.

```
#random forest model

> forest_model <- randomForest(Item_Outlet_Sales ~ ., data = new_train, mtry = 15, ntree = 1000)

> print(forest_model)

> varImpPlot(forest_model)
```

This model throws RMSE = 1132.04 which is not an improvement over decision tree model. Random forest has a feature of presenting the important variables. We see that the most important variable is Item_MRP (also shown by decision tree algorithm).



This model can be further improved by tuning parameters. But this can also be your output for submission if you want.

```
> main_predict <- predict(main_tree, newdata = new_test, type = "vector")

> sub_file <- data.frame(Item_Identifier = test$Item_Identifier, Outlet_Identifier = test$Outlet_Identifier,
Item_Outlet_Sales = main_predict)

> write.csv(sub_file, 'Decision_tree_sales.csv')
```

When predicted on out of sample data, our RMSE has come out to be 1174.33. Here are some things you can do to improve this model further:

1. Since we did not use encoding, I encourage you to use one hot encoding and label encoding for random forest model.
2. Parameters Tuning will help.
3. Use Gradient Boosting.
4. Build an ensemble of these models. Read more about Ensemble Modeling.

Thank You