PROBLEM STATEMENT:

The aim of this problem is to predict the price of an item on online market. Mercari is Japanese online company to sell the items.

It is a fusion of traditional machine learning and natural language problem.

CONTENTS OF REPORT:

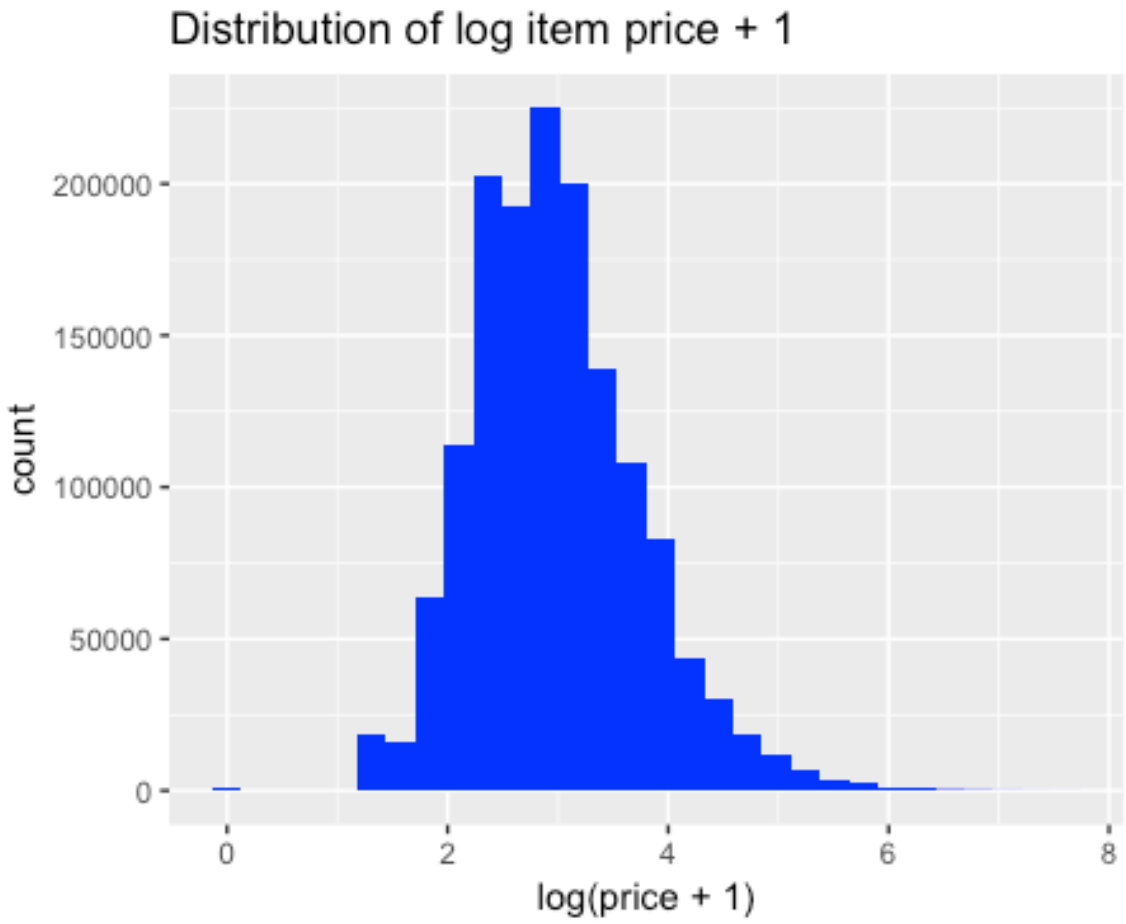
- Observations and analysis.
- Initial pre-processing and text processing.
- Training and modeling.
- predictions

Observations and analysis

The evaluation metrics is RMSLE, Which gets degrade if we underestimate the price .So as per the nature of RMSLE we take log of the price in our analysis..

So we load the test dataset and we have 1482535 observations with 8 variables.

Price variable is highly skewed in train dataset so we take the log of it and plot the histogram of price with and without log to check the distribution.



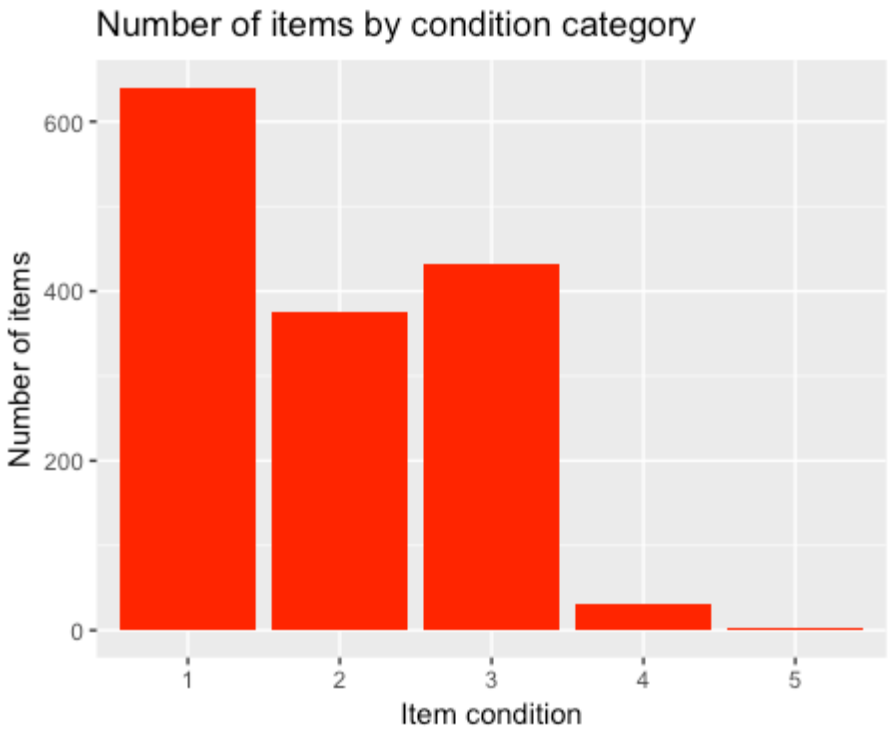
There is good normal distribution of log price.

While the price distribution without log is very much skewed.

So later in the project we might transform the value of price by applying the log function for pre-processing and feature engineering.

Lets us check some other variables and different scenarios.

Now lets check the number of items by condition category.



Item ranges from 1 to 5. Most number of items falls under condition 1 and least number of items falls under condition 4 and 5.

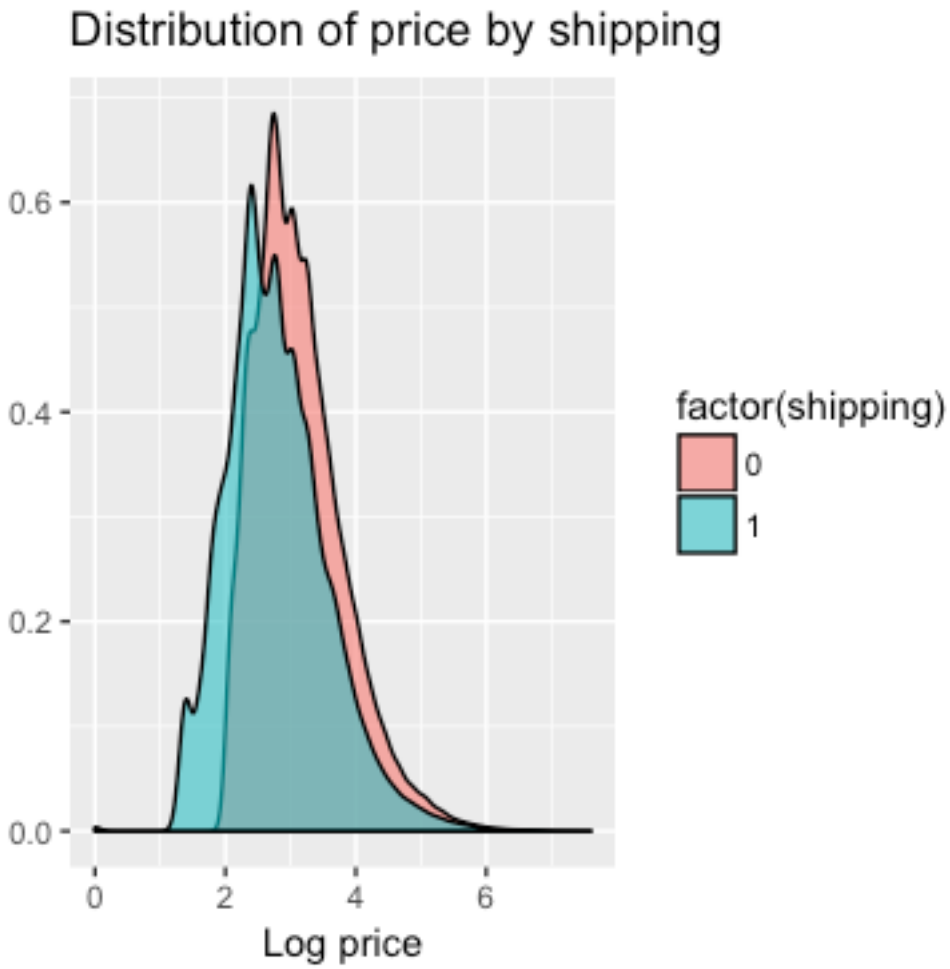
Shipping is the dummy variable having the value 0 and 1 , if shipping fee is paid by seller then “1” else “0”.

```
> table(train_set$shipping)

     0      1
819435 663100
>
```

We may assume that price of an item can be high if the shipping fee is paid by seller , but this can also be true for some items and not for all.

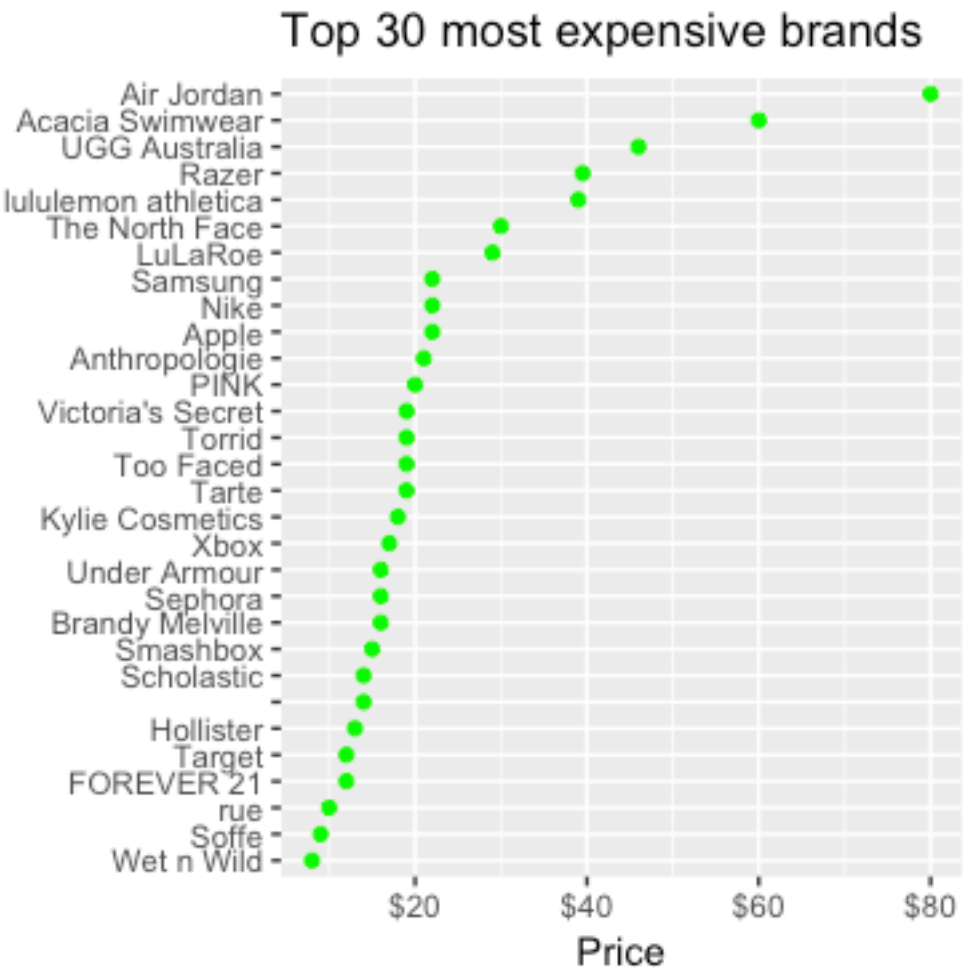
Lets verify this by checking the distribution of price around shipping by building a density graph.



But here we can see that items where shipping fee is paid by the seller has the lower average price.

Now let us check price distribution around brand variable.

Here we will extract top 30 most expensive brands.



Air Jordan and Acacia Swimwear brands are by far the most expensive brands, with a median price of $80 and $60.

Now let us analyze the item category variable. Lets see the item count under top 10 category.

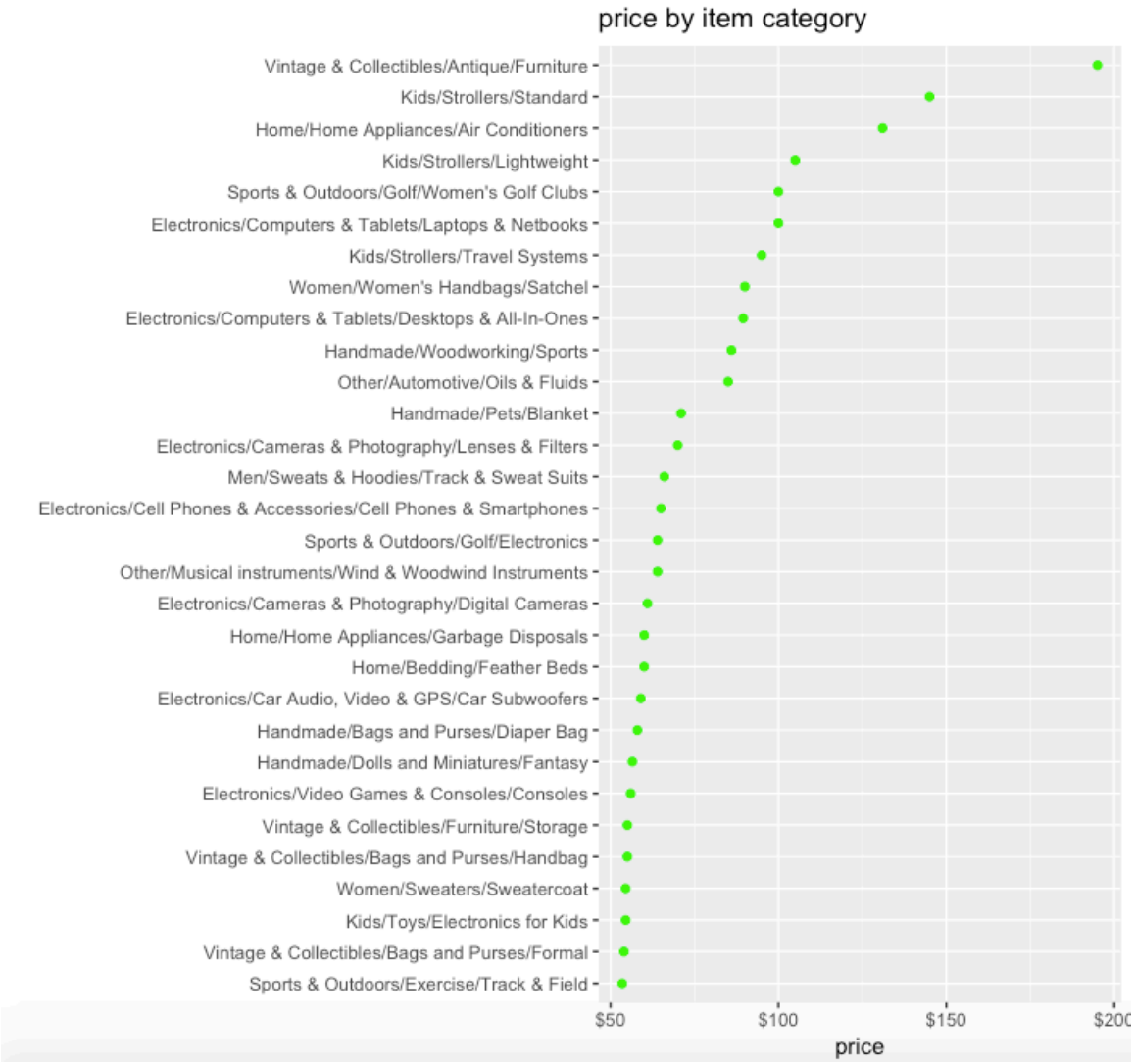
Products under women and beauty category has the highest number of count can be verified by below image.

Now let's analyze prices by category. What are the product categories with the highest selling price.

We have extracted the top 30 categories with highest prices which can be seen in below image.

Vintage items has the highest median price of around $200.

While 2nd and 3rd best categories are kids and home appliances.



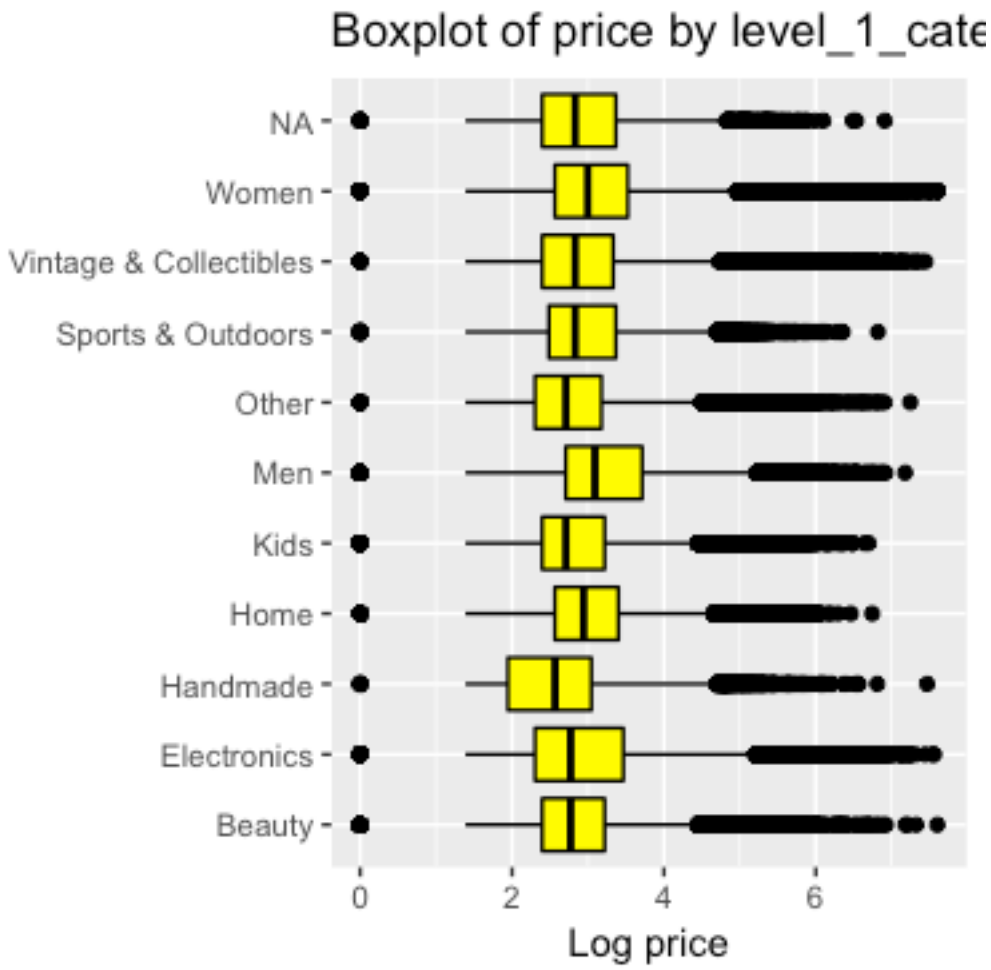
Here the subcategories are divided by " / ". we can split the categories and analyze it further.

```
> head(train_set[, c("level_1_category", "level_2_category")])
   level_1_category    level_2_category
1:              Men                Tops
2:      Electronics Computers & Tablets
3:            Women      Tops & Blouses
4:             Home   Home D\303\251cor
5:            Women             Jewelry
6:            Women               Other
```

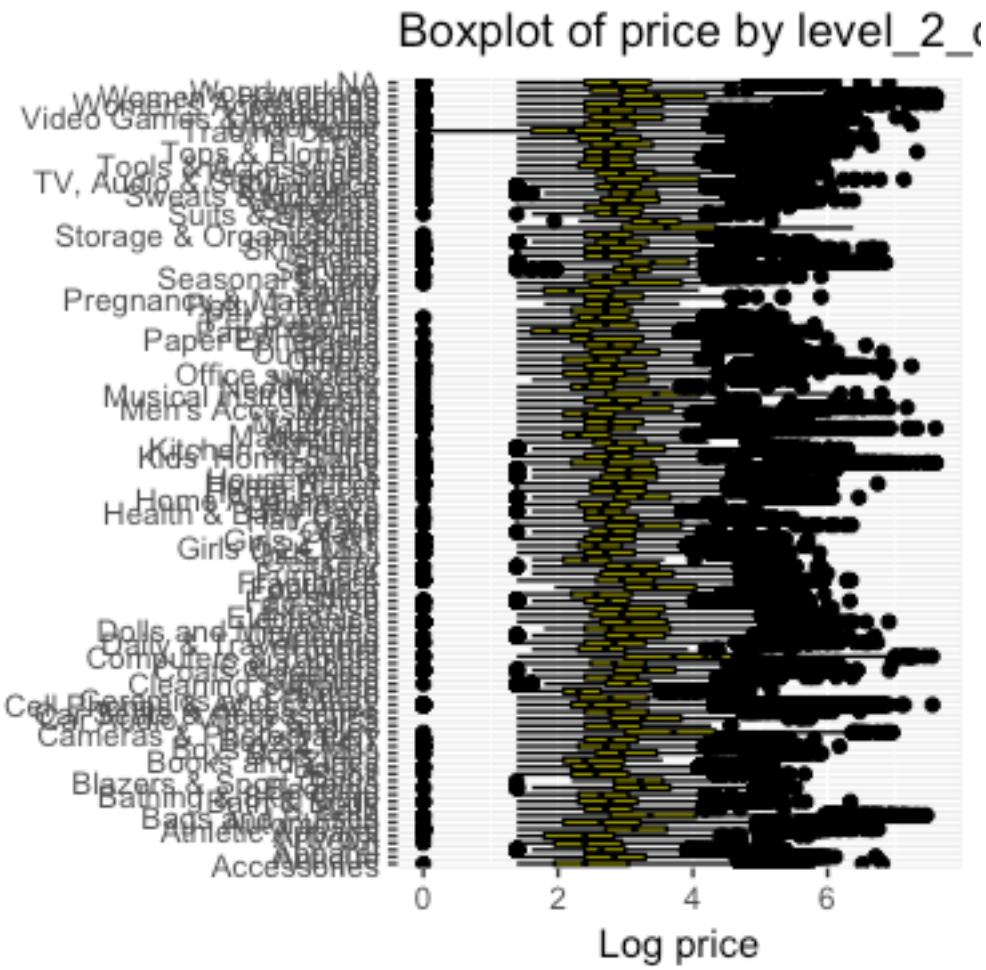
```
> table(train_set$level_1_category)

              Beauty          Electronics            Handmade                Home
              207828               122690                30842               67871
                Kids                 Men                 Other    Sports & Outdoors
              171689               93680                45351               25342
Vintage & Collectibles               Women
               46530              664385
>
```

Here there are 10 level 1 categories. Let's analyze the price around level 1 category by building boxplot.



Here we can see an interesting thing that men's category has the highest price. Lets do the same for 2nd level category. Here there are 114 level 2 category.



the boxplot looks quite messy here.

Initial Pre-Processing and Text processing.

- Here we load the Train and Test data by ignoring missing values.
- Now we filter the records which has price = 0.
- Adding new variable price_log by applying log function to the price variable. And other variables like name_length , description_length , name_words , description_words.
- Later we split the category names and handling the missing values.
- All the above steps were done on train dataset.

- Same steps needs to be done for test data as we need same kind of processing for both datasets.
- After pre-processing both train and test datasets we combine them into a new dataset by column bind.
- Now we will start processing the text.
- We perform tokenization on “item_description” variable by removing numbers , punctuations , hyphens , symbols. Then we convert these tokens into lower case.
- Now we remove stopwords and perform stemming.
- Document frequency matrix is build from this tokens.
- Considering only to tokens which appeared minimum of 300 time we build TF-IDF matrix.
- Now same steps are performed on “Name” variable but in names we take the minimum count of 100.
- Now we will create the Sparse matrix with the variables we select except name and description variable because we have processed it seperately, and we will make the datatype of sparse matrix and TF-IDF matrix same for description and name variable.
- After this we will combine the sparse matrix and TF-IDFs into a new variable.

Training and modeling.

- Splitting the data into train and test.
- Setting label as price because it is our target variable which we need to predict.
- Construct the dense matrix(which has non zero values) from our train and test data.
- Setting up the XG Boost parameters required to pass , regression technique used is linear.
- Now pass the train data set and let the model learn.
- After that make prediction using separate variable by passing the test dataset into the trained model.
- Convert the price variable back to exp1.
- Finally store the predicted values in a separate csv file.
- Note that RMSE value will be displayed in every 50 rounds and it will go on till 500 rounds,
- The final RMSE value is showed in the below image.

```
+                    early_stopping_rounds = 100)
[1]     train-rmse:2.470026
Will train until train_rmse hasn't improved in 100 rounds.

[51]    train-rmse:0.657193
[101]   train-rmse:0.605327
[151]   train-rmse:0.590233
[201]   train-rmse:0.579392
[251]   train-rmse:0.571046
[301]   train-rmse:0.564254
[351]   train-rmse:0.558267
[401]   train-rmse:0.553077
[451]   train-rmse:0.548563
[500]   train-rmse:0.544730
>
```