Capstone Project - Mercari Price Predictor

Ram Subramaniam

February 22, 2018

Acknowledgement

Some of the information mentioned below are from Wikipedia so could have been from various sources.

Latent Dirichlet Allocation from the document written by Edwin Chen

Exploratory Data Analysis expanded on some of the item described by Troy Walters

I also would like to acknowledge the help from my mentor Dhiraj Khanna with some parts of the code where advanced topics are used which are not part of the course. This helped me in reading advanced topics. Thanks again to Dhiraj for his patience with me during the last 3 months and accomodating to some of my schedule change requests.

Code

The code embedded in the document is not executable. The source code can be found in the following GitHub link https://github.com/ramsubra1/Cap_Project (https://github.com/ramsubra1/Cap_Project)

Introduction

Mercari is a Japanese community powered shopping app. It gives an opportunity for people to sell their stuff - new or used. It is always challenging for anyone to set a fair price for the item they are trying to sell. This is contest currently active in Kaggle and it is for developers/enthusiasts to build a tool that will predict the price of an item(s) that people are trying to sell accurately.

Problem Statement

As people try to sell their good(s) in Mercari website, there needs to be a effective tool to predict the price based on characteristics of the item as entered by the seller. The characteristics entered by the seller are as follows

1. Short Item Description
2. Product Category
3. Detailed Description
4. Item Condition
5. Brand Name
6. Shipping (paid by seller or buyer)

Tragetted Users

The users who will use this will be sellers of products using Mercari app.

Data

As part of the Kaggle contest the data was provided to the contestants by Mercari. The data is located here

Datasets - Mercari Datasets (https://www.kaggle.com/c/mercari-price-suggestion-challenge/data)

The above link contains datasets for both Train and Test. The data structure of the train and test datasets are as below:

- test_id and train_id - the id of the listing
- name - the title of the listing.
- item_condition_id - the condition of the item provided by the seller
- category_name - category of the listing
- brand_name - brand name of the item being sold
- price - the price that the item was sold for. This is the target variable that you will predict. The unit is USD. This column doesn’t exist in test.tsv since that is what you will predict.
- shipping - 1 if shipping is paid for by the seller else 0 if by buyer
- item_description - description of the item

The Approach

The following steps highlight the strategy adopted for carrying out analysis

1. Exploratory Analysis
* Libraries Used
* Data Overview
* Target Variable Analysis (Price)
* Item Condition
* Shipping
* Item Categories
* Feature Interactions
* Item Descriptions
2. Data Wrangling and Cleaning
* Deal with missing values
* Dropping or combining columns
* Rearrange and transform dataset for cleaner analysis
* Removing Stop words from the descriptions
* Grouping items based on price
3. TOpic Modelling
* Topic Modelling Latent Dirichlet Allocation(LDA) model
4. Predictive Analysis
* Modeling the sample using XGBOOST and Root Mean Square Log Error
* Tuning the Model
* Testing out whether the outcome - predicting the price is accurate

1. Exploratory Analysis
In statistics, exploratory data analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modeling or hypothesis testing task.EDA is different from initial data

analysis (IDA),[1] which focuses more narrowly on checking assumptions required for model fitting and hypothesis testing, and handling missing values and making transformations of variables as needed. * Libraries Used Below are the list of libraries that were used as part of building this project

```
library(tidyverse)
library(data.table)
library(tidytext)
library(ggplot2)
library(scales)
library(gridExtra)
library(stringr)
library(tm)
library(text2vec)
library(quanteda)
library(tictoc)
library(xgboost)
library(caret) #for dummyVars
library(Metrics) #for performance evaluation
library(foreach)
library(doSNOW)
library(doParallel)
library(RecordLinkage)
library(wordcloud)
library(RColorBrewer)
```

```
* Data Overview
Reading the train dataset
```

```
df_train <- fread("train.tsv")
```

```
##
Read 13.5% of 1482535 rows
Read 27.7% of 1482535 rows
Read 37.1% of 1482535 rows
Read 47.9% of 1482535 rows
Read 66.8% of 1482535 rows
Read 82.3% of 1482535 rows
Read 83.6% of 1482535 rows
Read 1482535 rows and 8 (of 8) columns from 0.315 GB file in 00:00:10
```

```
Check the number of records in the file matches to the number of records read into R
```

```
 dim(df_train)
```

```
## [1] 1482535       8
```

```
Check the file size in MB matches to the file size in the system folder
```

```
  print(object.size(df_train), units = 'Mb')
```

```
## 417.3 Mb
```

```
Check the summary of the data file loaded into R
```

```
  summary(df_train)
```

```
##     train_id           name           item_condition_id category_name
##  Min.   :      0   Length:1482535     Min.   :1.000     Length:1482535
##  1st Qu.: 370634   Class :character   1st Qu.:1.000     Class :character
##  Median : 741267   Mode  :character   Median :2.000     Mode  :character
##  Mean   : 741267                      Mean   :1.907
##  3rd Qu.:1111901                      3rd Qu.:3.000
##  Max.   :1482534                      Max.   :5.000
##   brand_name            price             shipping      item_description
##  Length:1482535     Min.   :   0.00   Min.   :0.0000   Length:1482535
##  Class :character   1st Qu.:  10.00   1st Qu.:0.0000   Class :character
##  Mode  :character   Median :  17.00   Median :0.0000   Mode  :character
##                     Mean   :  26.74   Mean   :0.4473
##                     3rd Qu.:  29.00   3rd Qu.:1.0000
##                     Max.   :2009.00   Max.   :1.0000
```

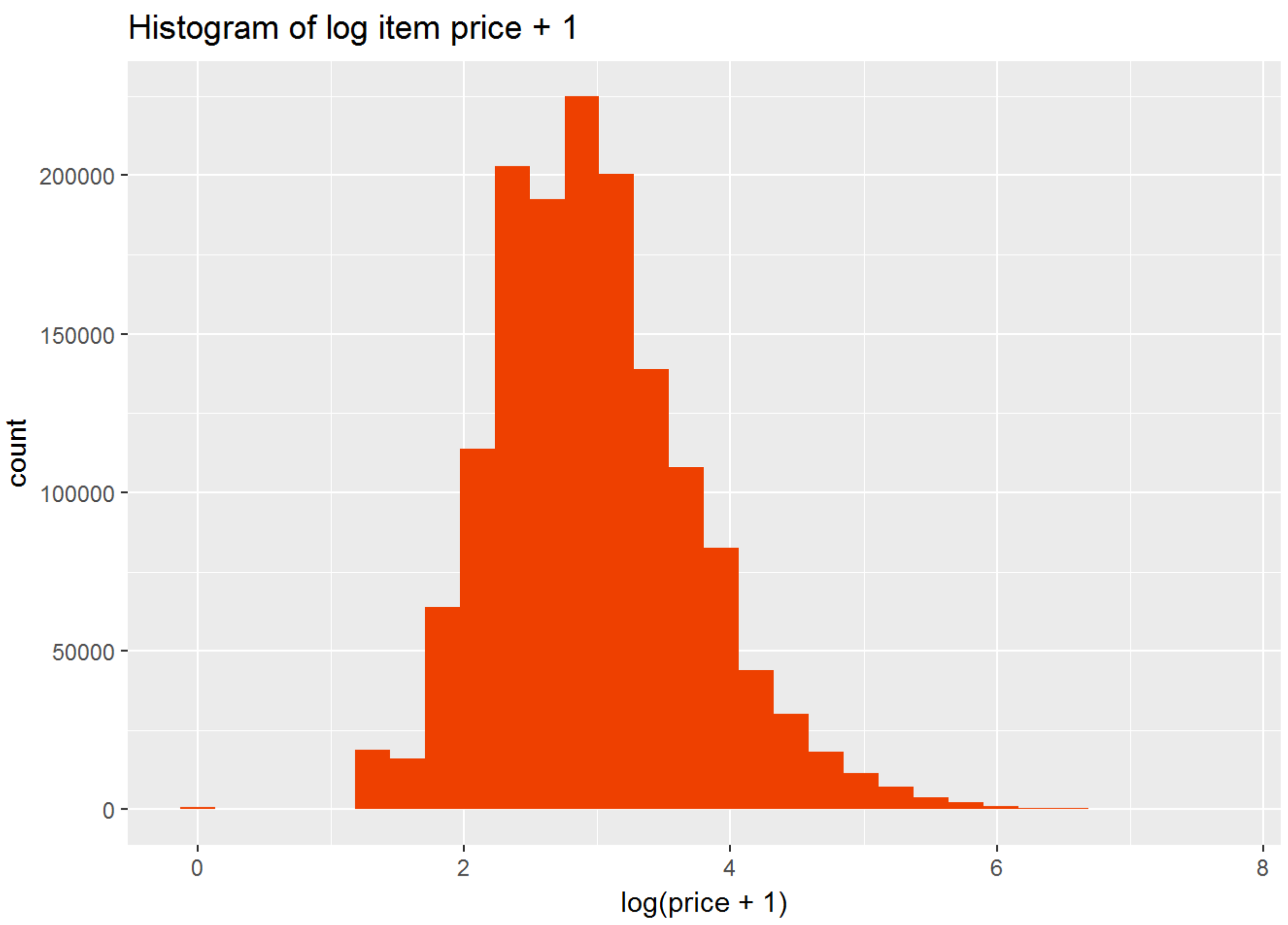
```
* Target Variable Analysis (Price)
```

```
range(df_train$price)
```

```
## [1]    0 2009
```

he item price ranges from 0 (I guess some items on Mercari are given away?) to $2009. Let’s look at the histogram of prices. Because price is likely skewed and because there are some 0s, we’ll plot the log of price + 1.

```
  ggplot(data = df_train, aes(x = log(price+1))) +
    geom_histogram(fill = 'orangered2') +
    labs(title = 'Histogram of log item price + 1')
```



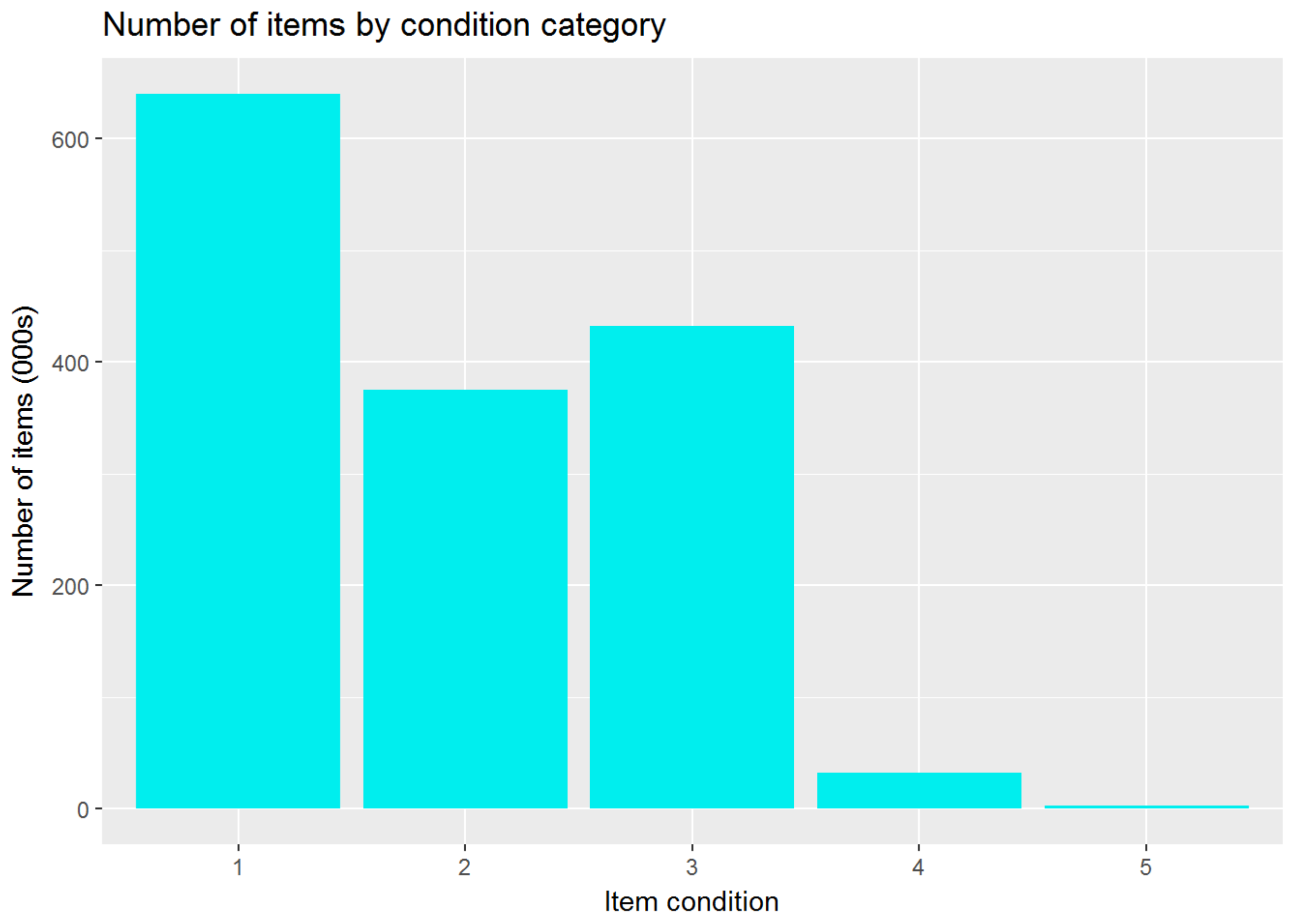
The (log price + 1) appears to be centered around 3 and has a longer right tail due to the 0 bound on the left.

- Item Condition Next let us look at Item Condition

```
table(df_train$item_condition_id)
```

```
##
##      1      2      3      4      5
## 640549 375479 432161  31962   2384
```

```
## Let us plot number of ITEM by CONDITION CATEGORY
df_train[, .N, by = item_condition_id] %>%
  ggplot(aes(x = as.factor(item_condition_id), y = N/1000)) +
  geom_bar(stat = 'identity', fill = 'cyan2') +
  labs(x = 'Item condition', y = 'Number of items (000s)', title = 'Number of items by condition
 category')
```

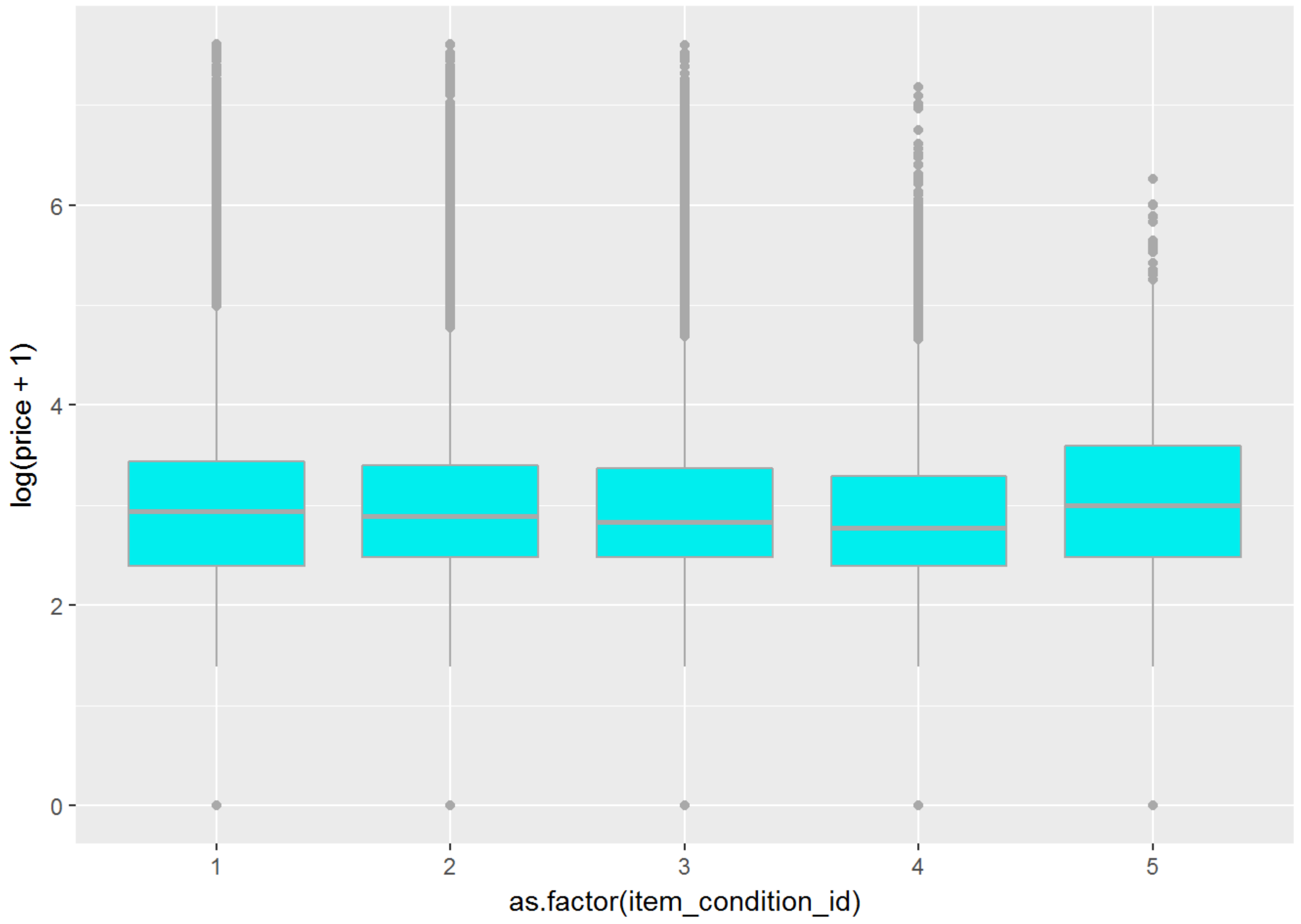


The item condition ranges from 1 to 5. There are more items of condition 1 than any other. Items of condition 4 and 5 are relatively rare. It's not clear from the data description what the ordinality of this variable is. My assumption is that since conditions 4 and 5 are so rare these are likely the better condition items. We can try and verify this. If a higher item condition is better, it should have a positive correlation with price. Let's see if that is the case.

```
table(df_train$item_condition_id)
```

```
##
##      1      2      3      4      5
## 640549 375479 432161  31962   2384
```

```
ggplot(data = df_train, aes(x = as.factor(item_condition_id), y = log(price + 1))) +   geom_boxp
lot(fill = 'cyan2', color = 'darkgrey')
```



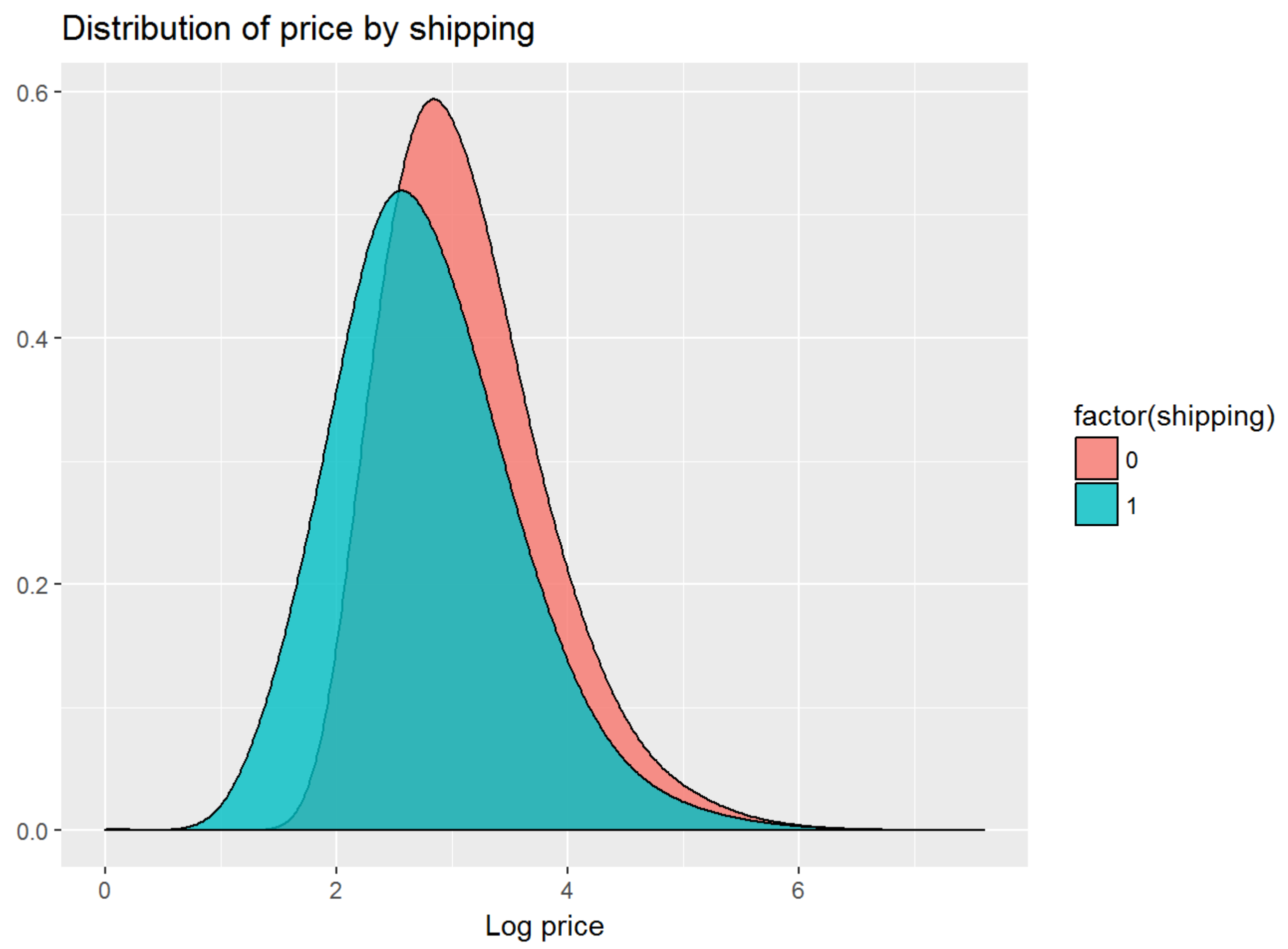
Looking at the average price by condition shows a relationship that is not quite as neat as expected. Condition 5 clearly has the highest price, however condition 1 has the next-highest price, followed by condition 2, then 3, then 4. Condition 5 is a bit of an anomaly in that it has the highest price. However, it also has the fewest number of items, so our point estimate has the most uncertainty.

- Shipping The initial thought is that items where the shipping fee is paid by the seller will be higher-priced. However, there are a number of conflating factors. This may be true within specific product categories and item conditions, but not when comparing items on the aggregate. Let’s see.

```
table(df_train$shipping)
```

```
##
##      0      1
## 819435 663100
```

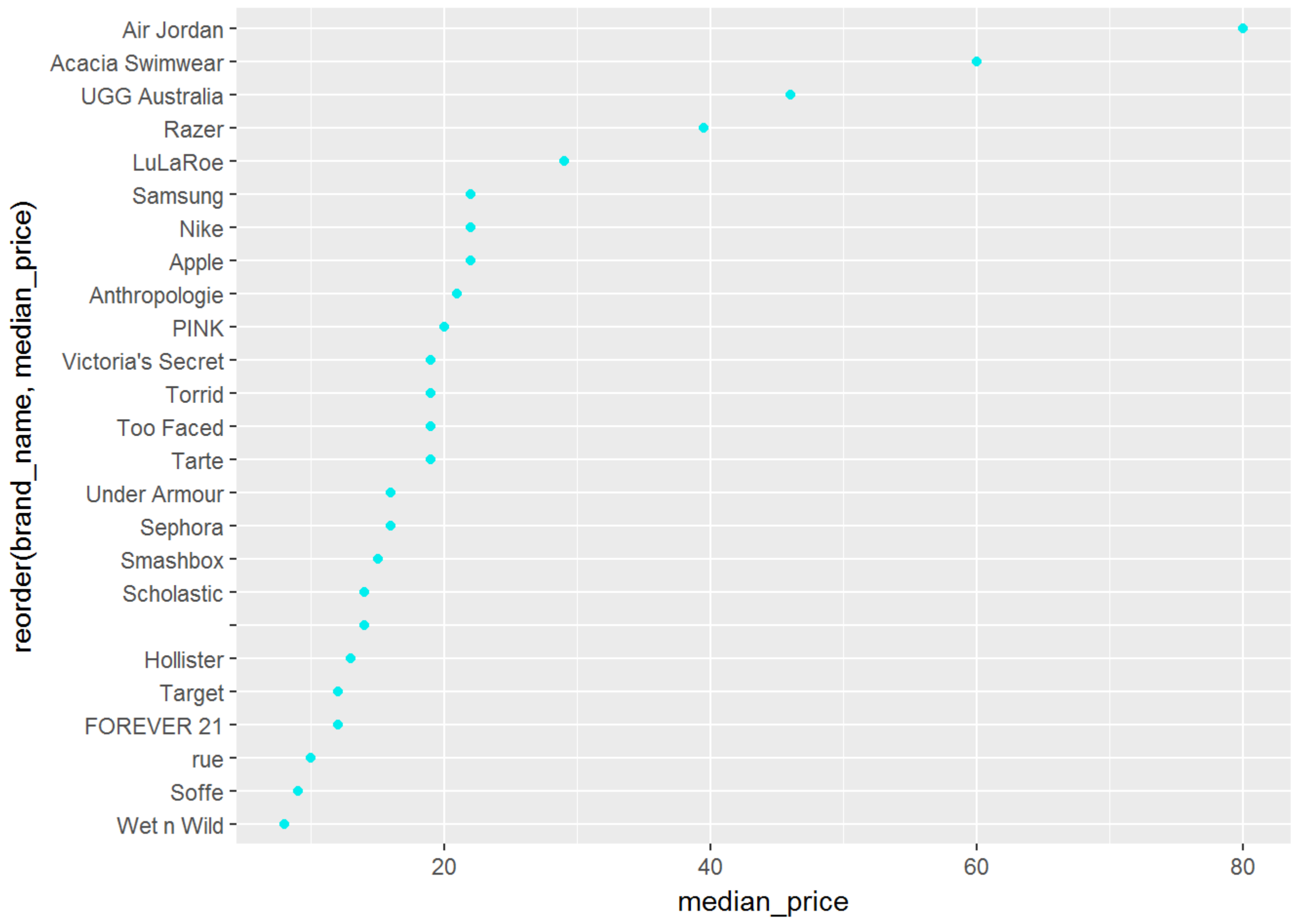
```
###Distribution of price by shipping where 1 is by seller and 0 is by purchaser
df_train %>%
  ggplot(aes(x = log(price+1), fill = factor(shipping))) +
  geom_density(adjust = 6, alpha = 0.8) +
  labs(x = 'Log price', y = '', title = 'Distribution of price by shipping')
```



Items where the shipping is paid by the seller have a lower average price.

- Brand

```
#plot to find the top 25 median price by brandname
df_train[, .(median_price = median(price)), by = brand_name] %>%
  head(25) %>%
  ggplot(aes(x = reorder(brand_name, median_price), y = median_price)) +
  geom_point(color = 'cyan2') + coord_flip()
```



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

- Item Categories Now let’s take a look at the categories. First, how many product categories are there?

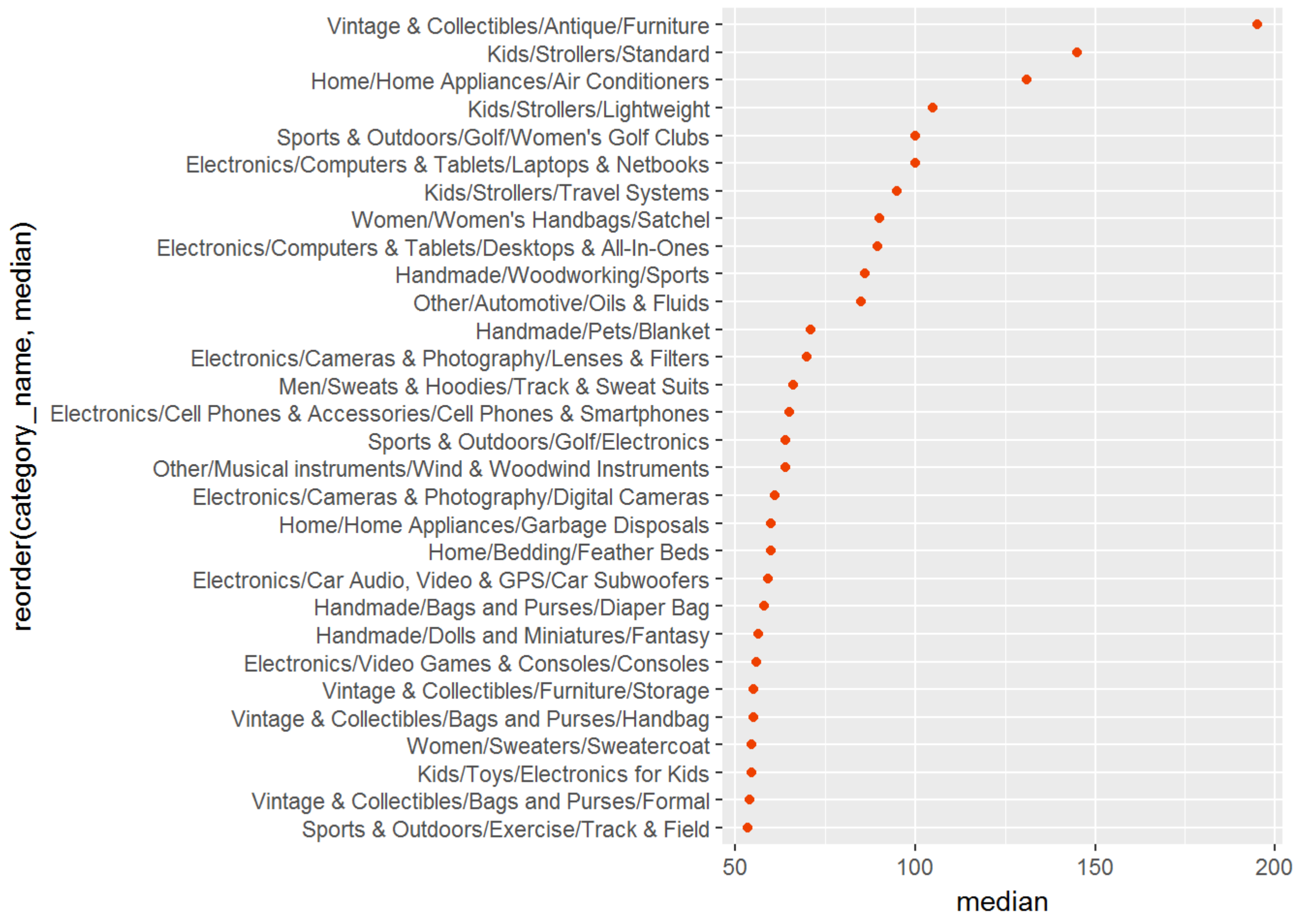
```
#find the number of unique categories
length(unique((df_train$category_name)))
```

```
## [1] 1288
```

```
sort(table(df_train$category_name), decreasing = TRUE)[1:10]
```

```
## 
##               Women/Athletic Apparel/Pants, Tights, Leggings 
##                                                        60177 
##                                Women/Tops & Blouses/T-Shirts 
##                                                        46380 
##                                           Beauty/Makeup/Face 
##                                                        34335 
##                                           Beauty/Makeup/Lips 
##                                                        29910 
##                         Electronics/Video Games & Consoles/Games 
##                                                        26557 
##                                           Beauty/Makeup/Eyes 
##                                                        25215 
## Electronics/Cell Phones & Accessories/Cases, Covers & Skins 
##                                                        24676 
##                                         Women/Underwear/Bras 
##                                                        21274 
##                                  Women/Tops & Blouses/Blouse 
##                                                        20284 
##                              Women/Tops & Blouses/Tank, Cami 
##                                                        20284
```

```
##Median price by item category
df_train[, .(median = median(price)), by = category_name][order(median, decreasing = TRUE)][1:30
] %>%
  ggplot(aes(x = reorder(category_name, median), y = median)) +
  geom_point(color = 'orangered2') +
  coord_flip()
```



Looking at the ten most popular categories shows that women's apparel is quite popular on Mercari. Of then top ten categories, 5 are women's apparel. Makeup is also a highly listed category as are electronics.

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

The category 'Vintage and Collectibles/Antique/Furniture' has the highest median sale price, followed by 'Kids/Strollers/Standard.

These product categories are quite specific. It would be interesting to aggregate these to broader categories and explore further. Since subcategories are divided by '/', we can easily obtain each individual item's first and second level categories using data.table's tstrsplit() function.

```
df_train[, c("level_1_cat", "level_2_cat") := tstrsplit(df_train$category_name, split = "/", kee
p = c(1,2))]

head(df_train[, c("level_1_cat", "level_2_cat")])
```

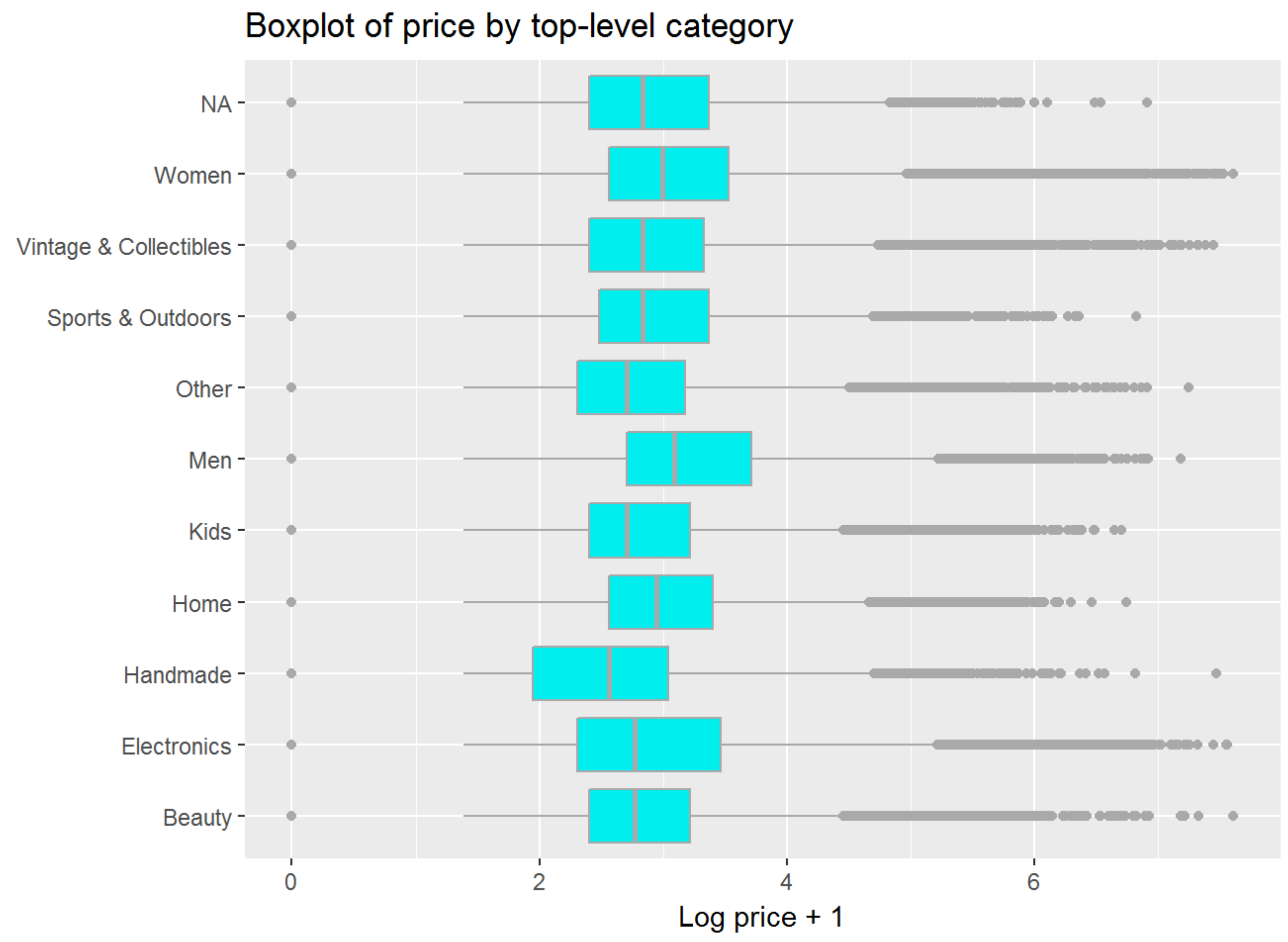
```
##    level_1_cat         level_2_cat
## 1:         Men                Tops
## 2: Electronics Computers & Tablets
## 3:       Women      Tops & Blouses
## 4:        Home         Home DÃ©cor
## 5:       Women             Jewelry
## 6:       Women               Other
```

How many top level categories are there?

```
table(df_train$level_1_cat)
```

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

```
df_train %>%
  ggplot(aes(x = level_1_cat, y = log(price+1))) +
  geom_boxplot(fill = 'cyan2', color = 'darkgrey') +
  coord_flip() +
  labs(x = '', y = 'Log price + 1', title = 'Boxplot of price by top-level category')
```



Interestingly, the Men’s category seems to have the highest median price. Let’s do the same for the second level categories.

```
df_train[, c("level_1_cat", "level_2_cat") := tstrsplit(df_train$category_name, split = "/", kee
p = c(1,2))]

head(df_train[, c("level_1_cat", "level_2_cat")])
```

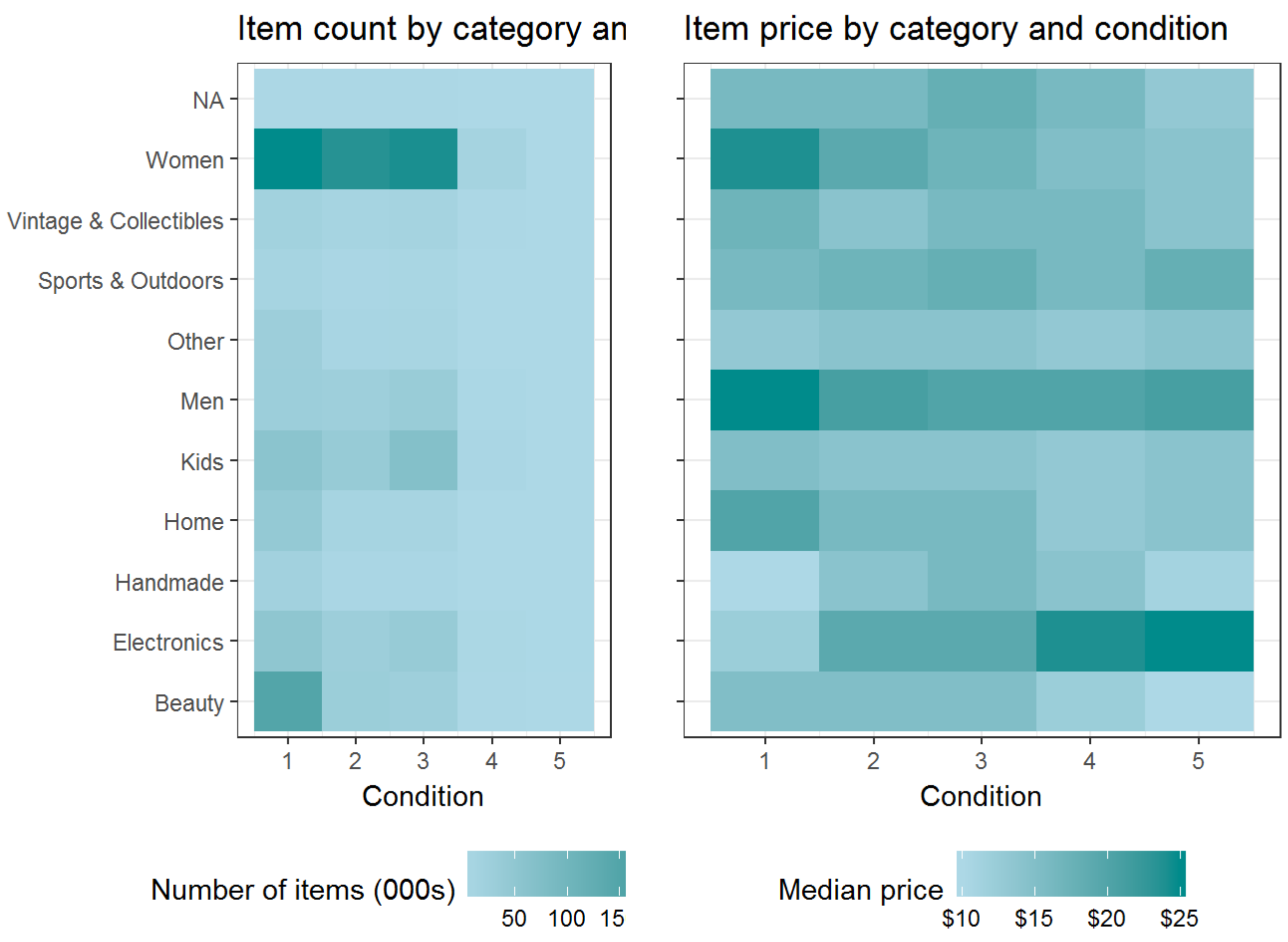
```
##    level_1_cat         level_2_cat
## 1:         Men                Tops
## 2: Electronics Computers & Tablets
## 3:       Women      Tops & Blouses
## 4:        Home         Home DÃ©cor
## 5:       Women             Jewelry
## 6:       Women               Other
```

How many top level categories are there?

```
##length(unique(df_train$level_2_cat))

#Create a box plot for level 2 categories
##df_train %>%
##  ggplot(aes(x = level_2_cat, y = log(price+1))) +
##  geom_boxplot(fill = 'cyan2', color = 'darkgrey') +
##  coord_flip() +
##  labs(x = '', y = 'Log price + 1', title = 'Boxplot of price by second-level category')
```

- Feature Interactions Now let us examine how item counts are distributed across top-level category and condition.

```
p1 <-
    df_train[, .N, by = c('level_1_cat', 'item_condition_id')] %>%
    ggplot(aes(x = item_condition_id, y = level_1_cat, fill = N/1000)) +
    geom_tile() +
    scale_fill_gradient(low = 'lightblue', high = 'cyan4') +
    labs(x = 'Condition', y = '', fill = 'Number of items (000s)', title = 'Item count by catego
ry and condition') +
    theme_bw() +
    theme(legend.position = 'bottom')

p2 <-
    df_train[, .(median_price = median(price)), by = c('level_1_cat', 'item_condition_id')] %>%
    ggplot(aes(x = item_condition_id, y = level_1_cat, fill = median_price)) +
    geom_tile() +
    scale_fill_gradient(low = 'lightblue', high = 'cyan4', labels = dollar) +
    labs(x = 'Condition', y = '', fill = 'Median price', title = 'Item price by category and con
dition') +
    theme_bw() +
    theme(legend.position = 'bottom', axis.text.y = element_blank())

grid.arrange(p1, p2, ncol = 2)
```



Women’s items of condition 1,2, and 3 are the most numerous. This is followed by Beauty products.

- Item Description At this point, we’ve already done a significant amount of data exploration and we haven’t even gotten to to real bulk of this problem, the description text. The description in unstructured data, so in order to explore it fully, we’ll need to do some text processing and normalization.

Is there a relationship between description length and price?

```
#Check if there is a relationship between the length of description and price
df_train[, desc_length := nchar(item_description)]

# set desc_length to NA where no description exists
df_train[item_description == 'No description yet', desc_length := NA]

cor(df_train$desc_length, df_train$price, use = 'complete.obs')
```

```
## [1] 0.04328234
```

There is no correlation between description length and price.

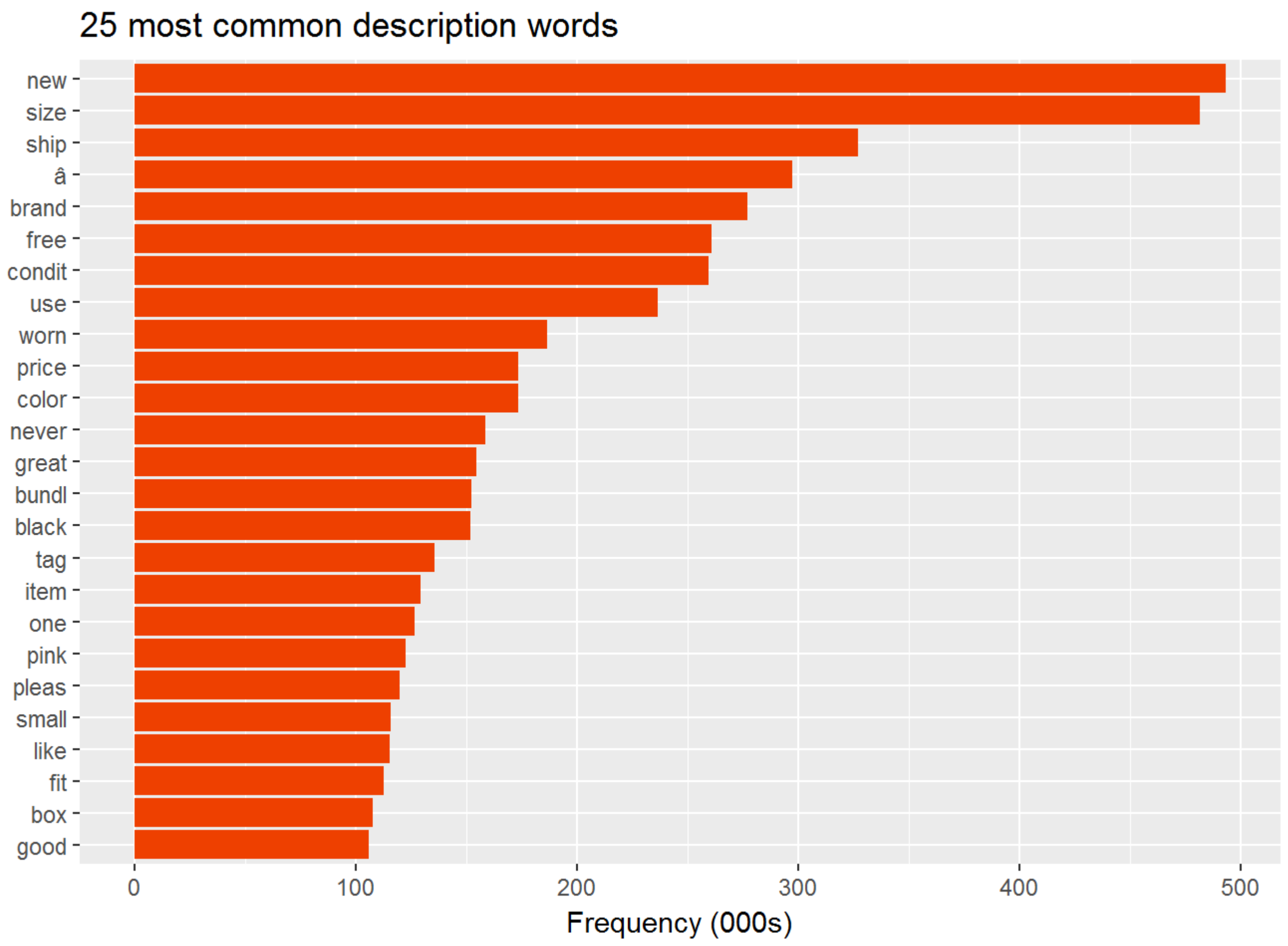
Now let’s begin the text analysis. We’ll use the quanteda package to do this. First, we will remove from the description column any occurrences of ‘No description yet’. Then we will convert the description column into a corpus object.

```
df_train[item_description == 'No description yet', item_description := NA]
# create the corpus object from the item_description column
dcorpus <- corpus(df_train$item_description)
# check first few lines of summary frame
summary(dcorpus)[1:5, ]
```

```
## Corpus consisting of 1482535 documents, showing 100 documents:
##
##   Text Types Tokens Sentences
##  text1     1      1         0
##  text2    32     39         3
##  text3    26     32         2
##  text4    34     41         8
##  text5     5      5         1
##
## Source:  C:/Users/rsubra02/Documents/Mercari/* on x86-64 by rsubra02
## Created: Thu Mar 29 13:41:53 2018
## Notes:
```

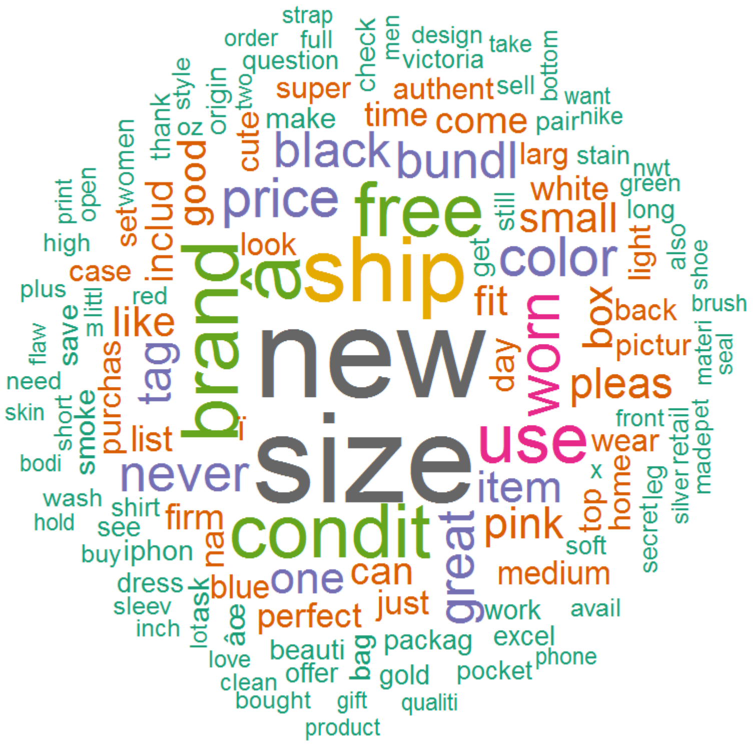
Now let's create a document-term matrix of the descriptions. To do this, we use the dfm() function and pass in our corpus object. We start with individual words, remove English stopwords and punctuation, and stem words.

```
dfm1 <- dfm(
    dcorpus,
    ngrams = 1,
    remove = c("rm", stopwords("english")),
    remove_punct = TRUE,
    remove_numbers = TRUE,
    stem = TRUE)

# get 25 most common words
tf <- topfeatures(dfm1, n = 25)

# convert to df and plot
data.frame(term = names(tf), freq = unname(tf)) %>%
    ggplot(aes(x = reorder(term, freq), y = freq/1000)) +
    geom_bar(stat = 'identity', fill = 'orangered2') +
    labs(x = '', y = 'Frequency (000s)', title = '25 most common description words') +
    coord_flip()
```



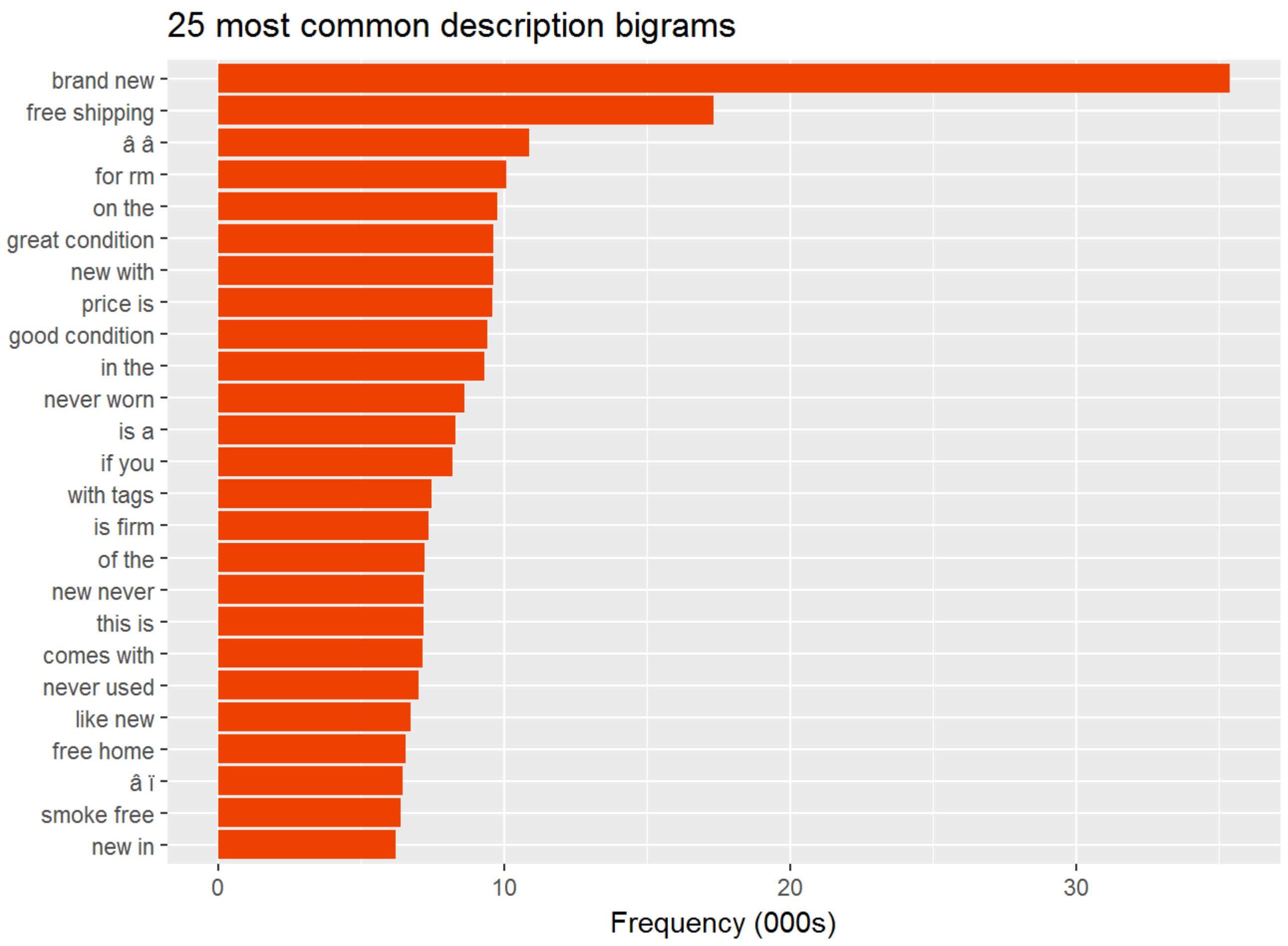
We can also use our dfm object to make a word cloud. Here I make a cloud with words that appear at least 30,000 times in the dataset.

```
set.seed(100)
textplot_wordcloud(dfm1, min.freq = 3e4, random.order = FALSE,
                   rot.per = .25,
                   colors = RColorBrewer::brewer.pal(8,"Dark2"))
```



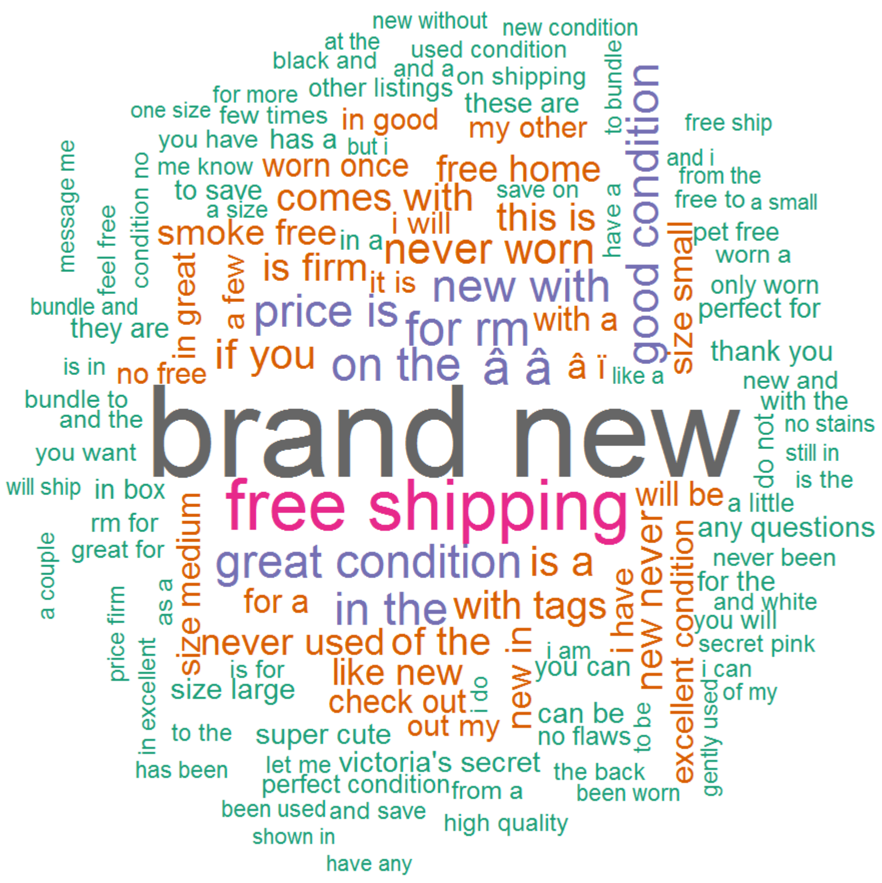
What if we wanted to look at ngrams? We can do so in a similar way, setting the ngrams argument in dfm() to 2. Because this will result in a very large document-term matrix and take a very long time, I use the sample_corpus() function to take a random sample of 15% of the documents in the corpus.

```
dfm2 <- dcorpus %>%
    corpus_sample(size = floor(ndoc(dcorpus) * 0.15)) %>%
    dfm(
        ngrams = 2,
        remove = c("rm", stopwords("english")),
        remove_punct = TRUE,
        remove_numbers = TRUE,
        concatenator = " "
    )
# get 25 most common bigrams
tf <- topfeatures(dfm2, n = 25)

# convert to df and plot
data.frame(term = names(tf), freq = unname(tf)) %>%
    ggplot(aes(x = reorder(term, freq), y = freq/1000)) +
    geom_bar(stat = 'identity', fill = 'orangered2') +
    labs(x = '', y = 'Frequency (000s)', title = '25 most common description bigrams') +
    coord_flip()
```



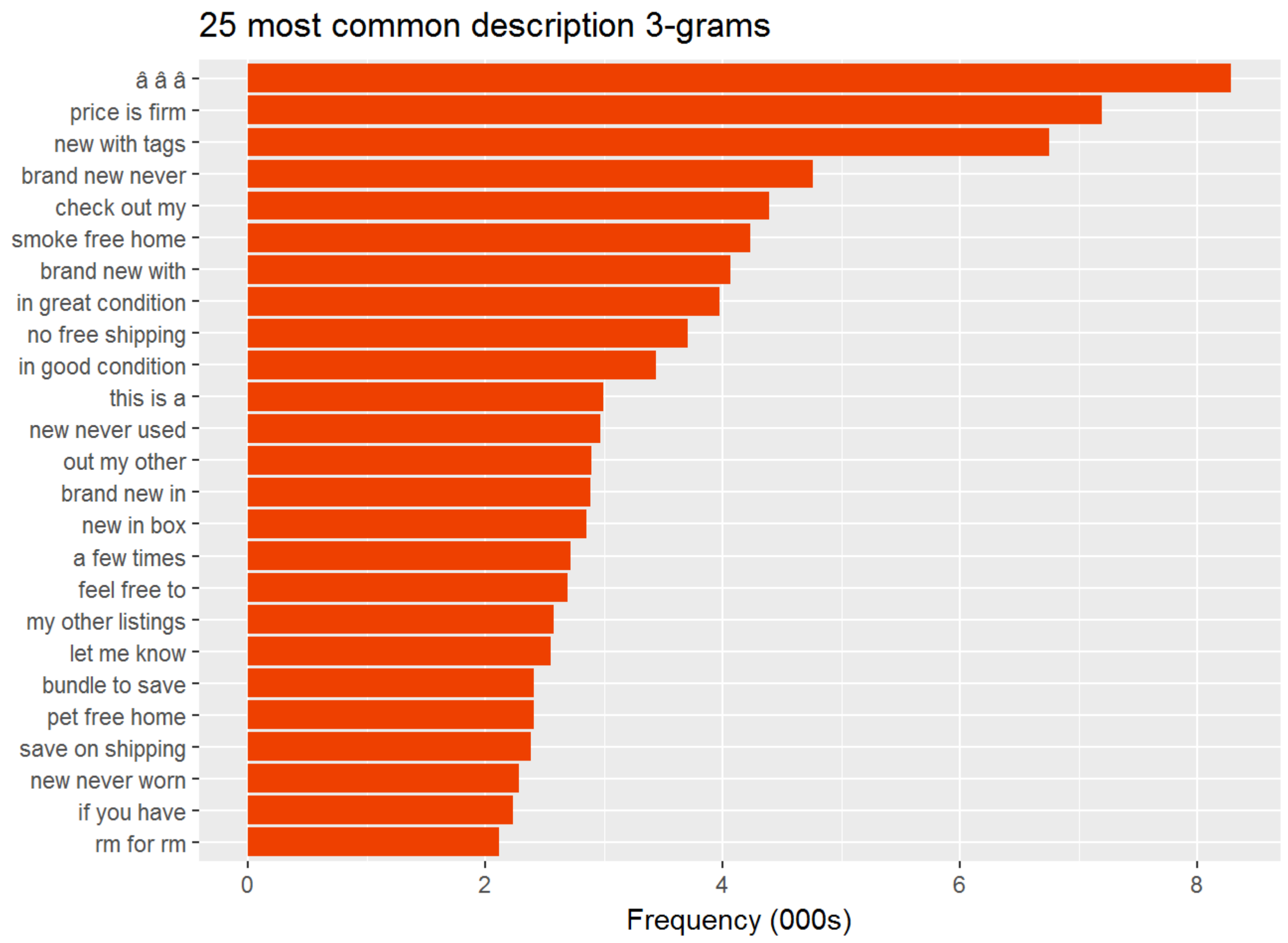
'Brand new' is the most commonly-occurring bigram follow by 'free shipping'. We can also make a word cloud of bigrams.

```
set.seed(100)
textplot_wordcloud(dfm2, min.freq = 2000, random.order = FALSE,
                   rot.per = .25,
                   colors = RColorBrewer::brewer.pal(8,"Dark2"))
```



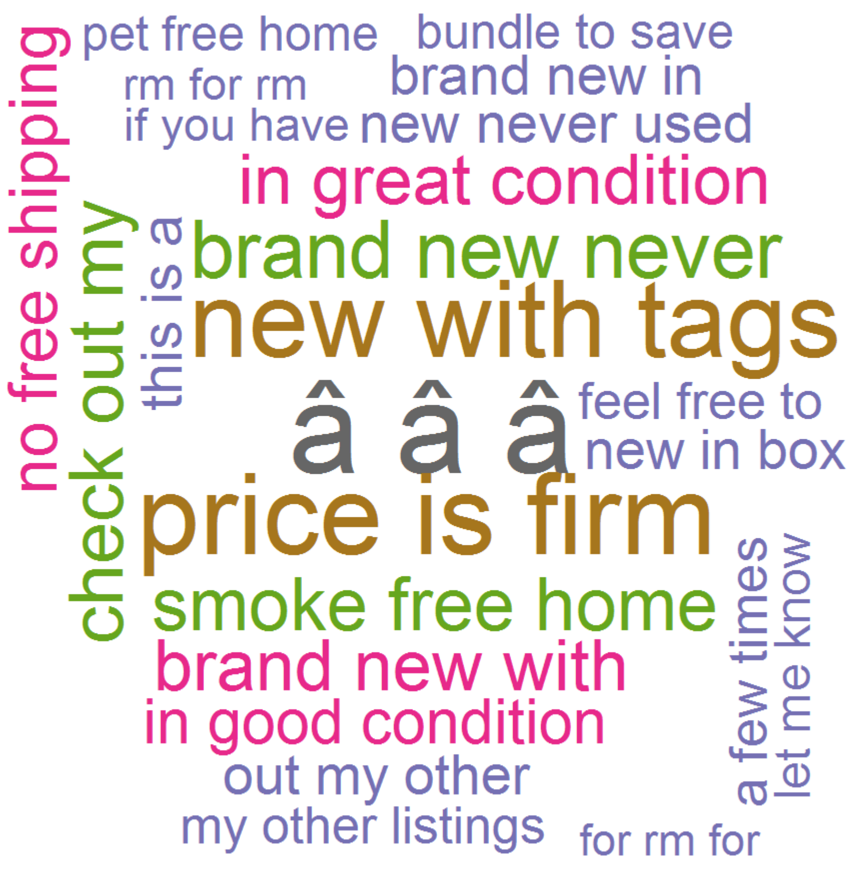
Let’s repeat this process one more time, this time with 3-grams.

```
dfm3 <- dcorpus %>%
    corpus_sample(size = floor(ndoc(dcorpus) * 0.15)) %>%
    dfm(
        ngrams = 3,
        remove = c("rm", stopwords("english")),
        remove_punct = TRUE,
        remove_numbers = TRUE,
        concatenator = " "
    )

# get 25 most common trigrams
tf <- topfeatures(dfm3, n = 25)

# convert to df and plot
data.frame(term = names(tf), freq = unname(tf)) %>%
    ggplot(aes(x = reorder(term, freq), y = freq/1000)) +
    geom_bar(stat = 'identity', fill = 'orangered2') +
    labs(x = '', y = 'Frequency (000s)', title = '25 most common description 3-grams') +
    coord_flip()
```



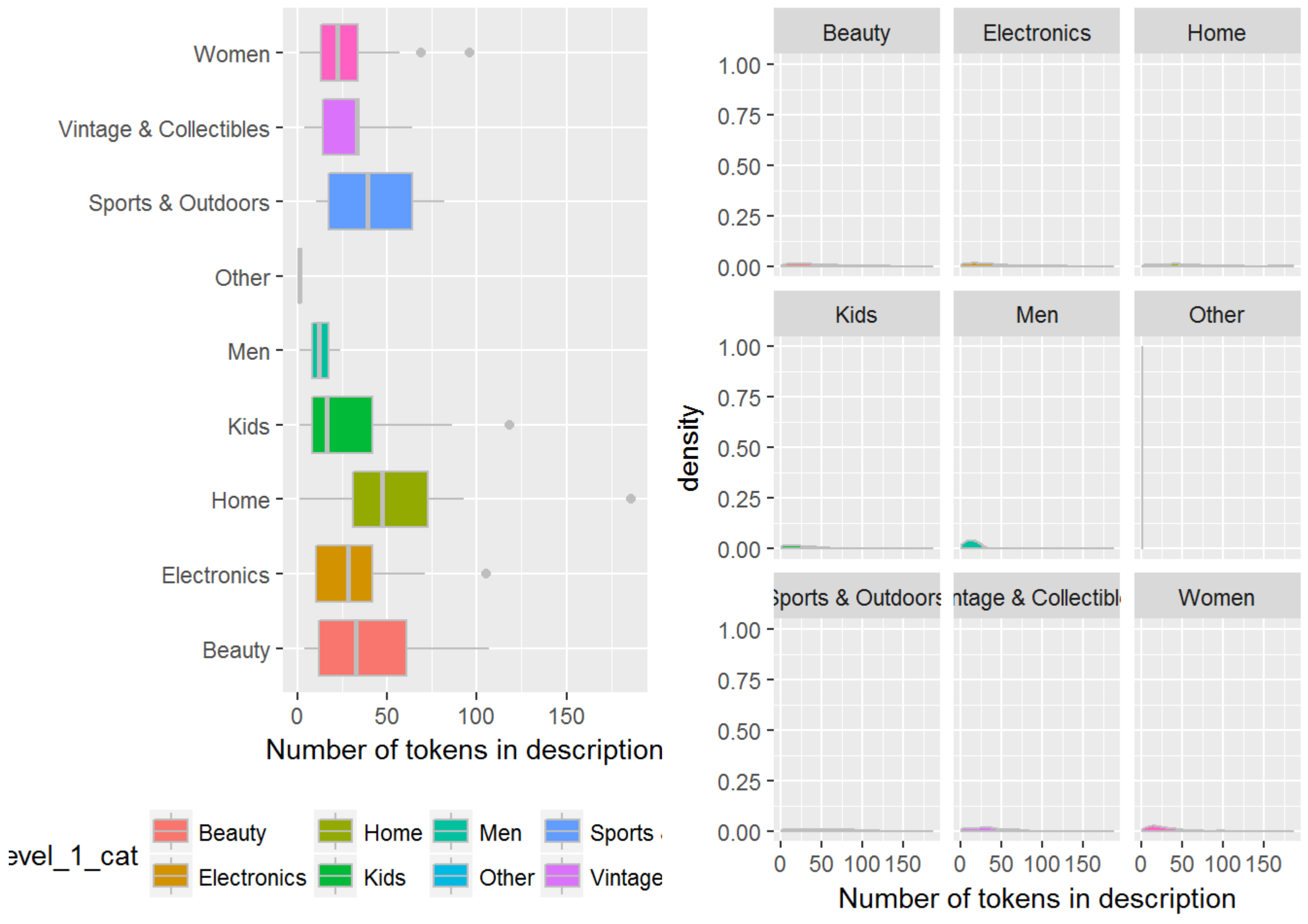
We see that 'price is firm' and 'new with tags' are the most common 3-grams by a large margin. 'Brand new never' (probably part of 'brand new never worn'), 'check out my', and 'smoke free home' are other popular trigrams.

```
set.seed(100)
textplot_wordcloud(dfm3, min.freq = 2000, random.order = FALSE,
                   rot.per = .25,
                   colors = RColorBrewer::brewer.pal(8,"Dark2"))
```



Let’s add some of the other training features to our corpus so that we can do further analysis. We can do this using the docvars() function from the quanteda package.

```
##Add other features as docvars
docvars(dcorpus, "price") <- df_train$price
docvars(dcorpus, "brand_name") <- df_train$brand_name
docvars(dcorpus, "item_condition_id") <- df_train$item_condition_id
docvars(dcorpus, "level_1_cat") <- df_train$level_1_cat
docvars(dcorpus, "level_2_cat") <- df_train$level_2_cat

p1 <- summary(dcorpus) %>%
    ggplot(aes(x = level_1_cat, y = Tokens)) +
    geom_boxplot(aes(fill = level_1_cat), color = 'grey') +
    coord_flip() +
    theme(legend.position = 'bottom') +
    labs(x = '', y = 'Number of tokens in description')

p2 <- summary(dcorpus) %>%
    ggplot(aes(x = Tokens)) +
    geom_density(aes(fill = level_1_cat), color = 'grey') +
    facet_wrap(~level_1_cat) +
    theme(legend.position = "none") +
    labs(x = 'Number of tokens in description')

grid.arrange(p1, p2, ncol = 2)
```



The most interesting thing that we see here is that the Men's category has the lowest average number of tokens in the description. Not only that it has the tightest distribution around the mean. Due to the lack of information this may mean that accurately predicting prices in this category will be most difficult.

2. Data Wrangling and Cleaning

Item description column is the one column where the most attempt was made towards cleaning it for further analysis. It was felt that this was the column that will be most helpful to do Topic Modelling towards predicting the price of an item being sold. The following was done for cleaning the data * Remove special characters * convert all text to lower case for easy comparison * remove multiple spaces between words * As there were enough data to work with and the percentage of records with missing data was negligible, those rows were ignored * Stop words were removed

```
#Cleanup Function
prep_fun = function(x) {
  x %>%
    # make text lower case
    str_to_lower %>%
    # remove non-alphanumeric symbols
    str_replace_all("[^[:alpha:]]", " ") %>%
    # collapse multiple spaces
    str_replace_all("\\s+", " ")
}
```

- Adding a column(s) for better analysis
  - Column for the length of item description

    - Column for number of words in item description
    - Column for number of Categories in the item category column
- To have a more tightly packed item price the log of price was taken rather than the actual price provided in source data.

```
#Get top 3 categories from the category_name column
comb[, c("cat_1","cat_2") := tstrsplit(comb$category_name, split = "/", keep = c(1,2))]

#features from the item description

#feature 1: number of words in item_description
comb$num_words <- sapply(comb$item_description, function(x) length(unlist(strsplit(as.character
(x), "\\W+"))))

#feature 2: length of words in description
comb$len_words <- str_length(comb$item_description)

#feature 3: average length of words in description
comb$avg_len_words <- comb$len_words / comb$num_words

#Convert price to log(price)
train$price <- log(train$price + 1)
```

3. Exploratory Analysis

- Grouping of data based on price into 5 different categories
- Low
- Mid
- High
- Higher
- Highest

```
comb$Class <- ifelse(comb$price<=2.398,"Low",
                       ifelse(comb$price<=2.89,"Mid",
                              ifelse(comb$price<=3.401,"High",
                                     ifelse(comb$price<=5,"Higher","Highest"))))
```

A new column is created in the data frame named Class is is assigned the value Low, Mid, High, Higher or Highest based on the log of price.

- Topic Modelling using Latent Dirichlet Allocation(LDA) model

1. What is Topic Modelling? Topic Modelling provides an algorithmic solution to managing, organizing and annotating large texts. The annotations aid in tasks of information retrieval, classification and exploration. topic models are generative models which provide a probabilistic framework for the term frequency occurrences in documents.

LDA represents documents as mixtures of topics that spit out words with certain probabilities. It assumes that documents are produced in the following fashion: when writing each document.

Decide on the number of words N the document will have (say, according to a Poisson distribution). Choose a topic mixture for the document (according to a Dirichlet distribution over a fixed set of K topics). For example, assuming that we have the two food and cute animal topics above, you might choose the document to consist of

1/3 food and 2/3 cute animals.

Generate each word w_i in the document by: First picking a topic (according to the multinomial distribution that you sampled above; for example, you might pick the food topic with 1/3 probability and the cute animals topic with 2/3 probability). Using the topic to generate the word itself (according to the topic's multinomial distribution). For example, if we selected the food topic, we might generate the word "broccoli" with 30% probability, "bananas" with 15% probability, and so on.

Assuming this generative model for a collection of documents, LDA then tries to backtrack from the documents to find a set of topics that are likely to have generated the collection.

Here the item description data field is the one on which topic modelling will be performed as the thought here is this field hold the information which will help in creating the model to predict the price of an item the seller is trying to sell in the Mercari site. For this project the topic modelling technique used is Latent Dirichlet Allocation(LDA) model

2. What is Latent Dirichlet Allocation(LDA) model This is the simplest form of topic modelling. In LDA, each document may be viewed as a mixture of various topics where each document is considered to have a set of topics that are assigned to it via LDA. This is identical to probabilistic latent semantic analysis (pLSA), except that in LDA the topic distribution is assumed to have a sparse Dirichlet prior. The sparse Dirichlet priors encode the intuition that documents cover only a small set of topics and that topics use only a small set of words frequently.

In this exercise LDA is used to identify the most frequently used words in each classification(Low, Mid, High, Higher, Highest). The number of occurrences of words in each item description for every record in the data set for every classification is counted. The maximum number words based on classification is used to set the price classification as Low, Mid, High, Higher or Highest.

```
LDAFunc <- function(x){
  #x$item_description <- prep_fun(x$item_description)
  tokens = x$item_description %>%
    tolower %>%
    word_tokenizer
  it = itoken(tokens, progressbar = FALSE)
  v = create_vocabulary(it,stopwords = stop_words$word) %>%
    prune_vocabulary(term_count_min = 100, doc_proportion_max = 0.3)
  vectorizer = vocab_vectorizer(v)
  dtm = create_dtm(it, vectorizer, type = "dgTMatrix")

  lda_model = LDA$new(n_topics = 5, doc_topic_prior = 0.1, topic_word_prior = 0.01)
  doc_topic_distr =
    lda_model$fit_transform(x = dtm, n_iter = 1000,
                            convergence_tol = 0.001, n_check_convergence = 25,
                            progressbar = FALSE)
  #lda_model$get_top_words(n = 10, lambda = 1)
  unique(as.vector(lda_model$get_top_words(n = 10, lambda = 1)))

}

Low <- filter(comb, Class=='Low')
LowWords <- LDAFunc(Low)
Mid <- filter(comb, Class=='Mid')
MidWords <- LDAFunc(Mid)
High <- filter(comb, Class=='High')
HighWords <- LDAFunc(High)
Higher <- filter(comb, Class=='Higher')
HigherWords <- LDAFunc(Higher)
Highest <- filter(comb, Class=='Highest')
HighestWords <- LDAFunc(Highest)
rm(Low,Mid,High,Higher,Highest)

  train <- comb[1:num_rows_train,]
  test <- comb[(num_rows_train+1) : nrow(comb),]
  rm(comb)

  # test$item_description <- prep_fun(test$item_description)
    x <- list(LowWords,MidWords,HighWords,HigherWords,HighestWords)
    classes <- c("Low","Mid","High","Higher","Highest")
    # test$Class <- vector("character",nrow(test))
    scores <- vector("numeric",5)
    cores=detectCores()
    cl <- makeCluster(cores,outfile="ramout.txt")
    registerDoSNOW(cl)
    tic()
    foreach(i = 1:nrow(test)) %dopar% {
      for(j in 1:5){
        scores[j] <- sum(sapply(x[[j]],grepl,test$item_description[i]))
      }
      test$Class[i] <- classes[which.max(scores)]
      print(i)
    }
    stopCluster(cl)
    toc()
```

The above being a time consuming process parallel processing techniques was also used based on the number of cores available in the machine the process is running using detectcores() and makeCluster functions.

4. Modelling for Predictive Pricing

Even thought the TRAIN and TEST data sets are available since the dataset is part of a contest the final price on the TEST dataset is not known. So what is done here is to use the TRAIN dataset to train the model as well as test the model for accuracy. Some percentage of the TRAIN dataset is used as both TRAIN and TEST dataset. where the TRAIN dataset is used to train the model and the TEST dataset part from the TRAIN dataset is used.

```
#Select the variables
myTrain <- train %>% select(-id,-name,-category_name,-brand_name,-item_description,
                            -Class)
##  myTrain <- train %>% select(-id,-name,-category_name,-brand_name,-item_description,
##                              -level4cat,-level5cat,-Class,-LogPrice)
#Binarize all factors
tic()
dmy <- dummyVars("~.",data=myTrain)
myTrainDmy <- data.frame(predict(dmy, newdata = myTrain))
toc()
```

To train the dataset the XGBOOST method in the XGBOOST package was used based on the predictor for this exercise - item price. Then the Root Mean Square Error function is used on the TEST dataset to predict the error variation of the model.

```
#Outcome name
outcomeName <- c('price')
#List of predictors
predictors <- names(myTrainDmy)[!names(myTrainDmy)%in%outcomeName]

#Train on 10% of the data
trainPortion <- floor(nrow(myTrainDmy)*0.1)
trainSet <- myTrainDmy[1:floor(trainPortion/2),]
testSet <- myTrainDmy[floor(trainPortion/2)+1:trainPortion,]

#Model Tuning
smallestError <- 100
for (depth in seq(1,20,1))  {
  for (rounds in seq(1,20,1)) {

    # train
    bst <- xgboost(data = as.matrix(trainSet[,predictors]),
                   label = trainSet[,outcomeName],
                   max.depth=depth, nround=rounds,
                   objective = "reg:linear", verbose=0,nthread=8)
    gc()

    # predict
    predictions <- abs(predict(bst, as.matrix(testSet[,predictors]), outputmargin=TRUE))
    err <- Metrics::rmse((as.numeric(testSet[,outcomeName])+1), (as.numeric(predictions)+1))

    if (err < smallestError) {
      smallestError = err
      print(paste(depth,rounds,err))
    }
  }
}
```

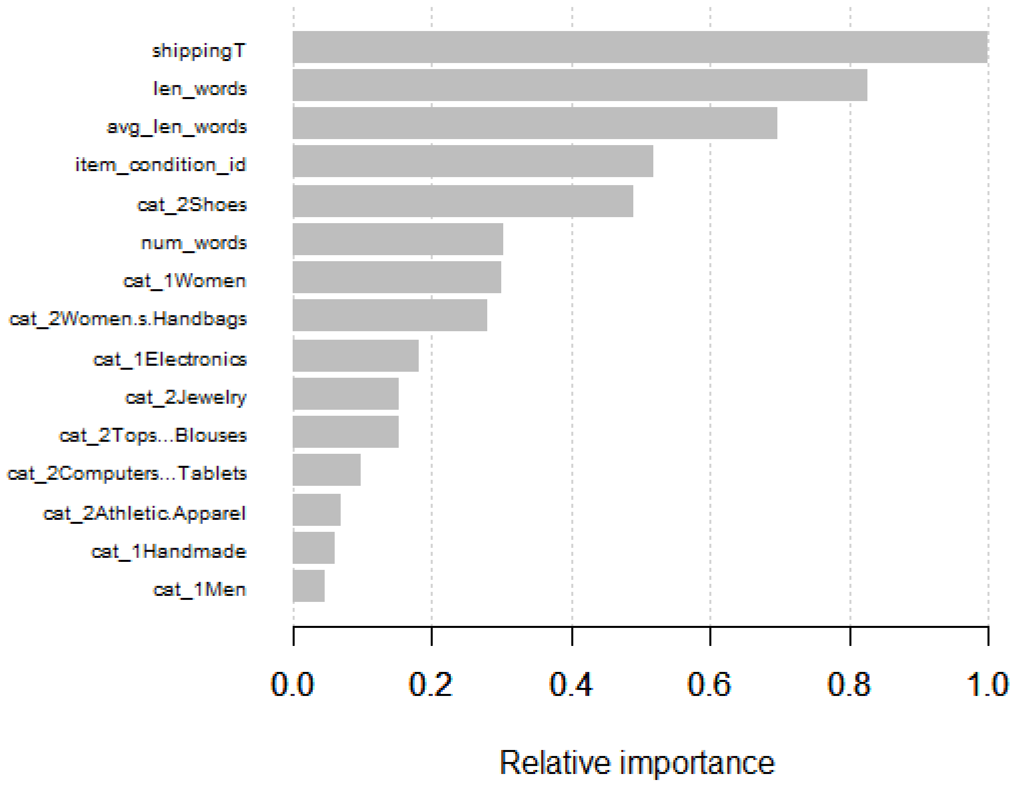
- Cross Validation of the Model

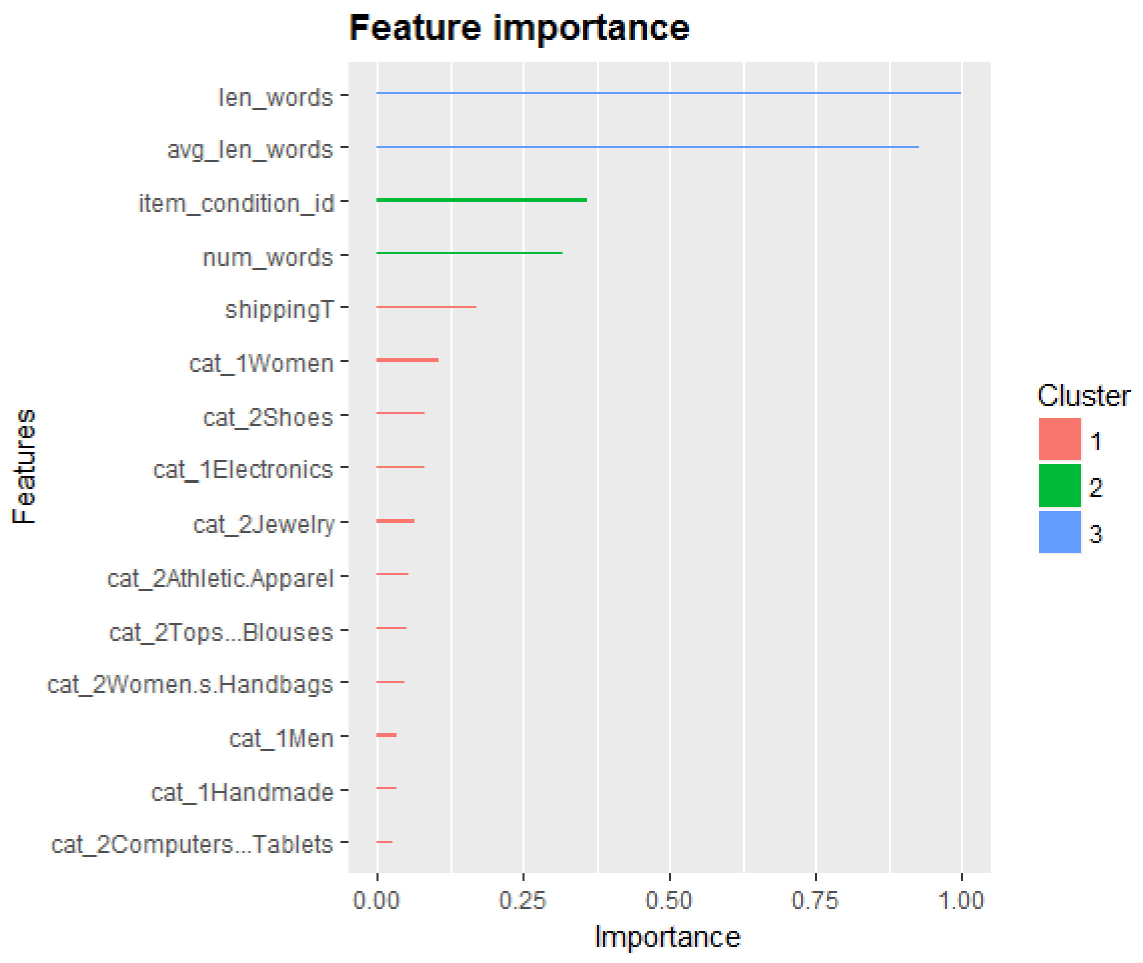
The next step is to cross validate the model

```
cv <- 30
trainSet <- myTrainDmy[1:trainPortion,]
cvDivider <- floor(nrow(trainSet) / (cv+1))

smallestError <- 100
for (depth in seq(1,10,1)) {
  for (rounds in seq(1,20,1)) {
    totalError <- c()
    indexCount <- 1
    for (cv in seq(1:cv)) {
      # assign chunk to data test
      dataTestIndex <- c((cv * cvDivider):(cv * cvDivider + cvDivider))
      dataTest <- trainSet[dataTestIndex,]
      # everything else to train
      dataTrain <- trainSet[-dataTestIndex,]

      bst <- xgboost(data = as.matrix(dataTrain[,predictors]),
                     label = dataTrain[,outcomeName],
                     max.depth=depth, nround=rounds,
                     objective = "reg:linear", verbose=0,
                     nthread=8)
      gc()
      predictions <- predict(bst, as.matrix(dataTest[,predictors]), outputmargin=TRUE)

      err <- rmse(as.numeric(dataTest[,outcomeName]), as.numeric(predictions))
      totalError <- c(totalError, err)
    }
    if (mean(totalError) < smallestError) {
      smallestError = mean(totalError)
      print(paste(depth,rounds,smallestError))
    }
  }
}
```

- Feature Importance Instead of using all the data points from the dataset it is much more ideal to use only the important features or columns from the dataset to predict in this case item price

```
#Testing the models
trainSet <- myTrainDmy[ 1:trainPortion,]

# assign everything else to test
testSet <- myTrainDmy[(trainPortion+1):nrow(myTrainDmy)/2,]

bst_final <- xgboost(data = as.matrix(trainSet[,predictors]),
                     label = trainSet[,outcomeName],
                     max.depth=7, nround=20, objective = "reg:linear", verbose=0)

importance_matrix <- xgb.importance(predictors, model = bst_final)

xgb.plot.importance(importance_matrix[1:15], rel_to_first = TRUE, xlab = "Relative importance")


(gg <- xgb.ggplot.importance(importance_matrix[1:15], measure = "Frequency", rel_to_first = TRUE
))
gg + ggplot2::ylab("Frequency")
```




Feature importance
len_words
avg_len_words
item_condition_id
num_words
shippingT
cat_1Women
cat_2Shoes
cat_1Electronics
cat_2Jewelry
cat_2Athletic.Apparel
cat_2Tops...Blouses
cat_2Women.s.Handbags
cat_1Men
cat_1Handmade
cat_2Computers...Tablets
Features
0.00
0.25
0.50
0.75
1.00
Importance
Cluster
1
2
3

```
topPredictors <- cat(toString(importance_matrix[1:15,1]))

predictors <- c("shippingT", "item_condition_id", "len_words", "cat_2Shoes", "cat_2Women.s.Handb
ags", "avg_len_words", "num_words", "cat_1Women", "cat_2Tops...Blouses", "cat_1Electronics", "ca
t_1Men", "cat_2Jewelry", "cat_2Athletic.Apparel", "cat_1Handmade", "cat_2Computers...Tablets")
trainPortion <- floor(nrow(myTrainDmy)*0.1)
trainSet <- myTrainDmy[1:floor(trainPortion/2),]
testSet <- myTrainDmy[floor(trainPortion/2)+1:trainPortion,]

#Model Tuning
smallestError <- 100
for (depth in seq(1,20,1))  {
  for (rounds in seq(1,20,1)) {

    # train
    bst <- xgboost(data = as.matrix(trainSet[,predictors]),
                   label = trainSet[,outcomeName],
                   max.depth=depth, nround=rounds,
                   objective = "reg:linear", verbose=0,nthread=8)
    gc()

    # predict
    predictions <- abs(predict(bst, as.matrix(testSet[,predictors]), outputmargin=TRUE))
    err <- Metrics::rmse((as.numeric(testSet[,outcomeName])+1), (as.numeric(predictions)+1))

    if (err < smallestError) {
      smallestError = err
      print(paste(depth,rounds,err))
    }
  }
}
```

Head of Submission will give the sample outcome or predicted item price

- Testing the models Use the remaining part of the TRAIN dataset as the new TEST dataset and then use that to predict the prices of items. This was useful since the TRAIN dataset which had the price was used and once we had the output values that could be compared with the actual price with the variance of the error that was calculated using the RMSE function.

```
trainSet <- myTrainDmy[ 1:trainPortion,]

# assign everything else to test
testSet <- myTrainDmy[(trainPortion+1):nrow(myTrainDmy),]

bst_adhoc <- xgboost(data = as.matrix(trainSet[,predictors]),
               label = trainSet[,outcomeName],
               max.depth=8, nround=9, objective = "reg:linear", verbose=0)
pred <- predict(bst_adhoc, as.matrix(testSet[,predictors]), outputmargin=TRUE)
rmse(as.numeric(testSet[,outcomeName]), as.numeric(pred))
[1] 0.6874568
submission <- as.data.frame(cbind(train$id[(trainPortion+1):nrow(myTrainDmy)],exp(pred)))
colnames(submission) <- c("test_id","price")
head(summary,10)
 test_id    price
1    14825 19.79368
2    14826 11.85752
3    14827 14.60265
4    14828 10.18738
5    14829 12.92065
6    14830 13.55039
7    14831 11.75021
8    14832 15.36275
9    14833 19.63104
10   14834 18.62362
```