Sales of summer clothes e-commerce Wish

Wahidullah Hessarey

05 11 2020

Contents

1) Introduction & purpose

1.1 Purpose and model evaluation

The purpose of this paper is to train machine learning (ML) algorithms in order to predict units sold of already existing products for existing merchants on the E-commerce platform called **Wish**. The target of the ML prediction model based on real data set of the month of August, 2020 is to predict **units sold** minimizing the Root Mean Square Error (RMSE) of model output $\hat{Y}$ vs. true units sold Y .
Since the data set consists of one single month with no other time related information, time series analysis is not subject to the ML.

$$RMSE = \sqrt{\frac{1}{N} * \sum_{n=1}^{N} (Y_n - \hat{Y}_n)^2},$$

with n = number of observations or rows.

1.2 Data set description

This data set contains product ratings and sales performance which comes from the platform 'Wish.com'. Basically, the products listed in the data set are those that would appear if you type "summer" in the search field of the platform. Following columns are found in the data set and described (see https:///www.kaggle.com/jmmvutu/summer-products-and-sales-in-ecommerce-wish):

- title: Title for localized for European countries. May be the same as title_orig if the seller did not offer a translation.
- title_orig: Original English title of the product.
- price: price you would pay to get the product.
- retail_price: reference price for similar articles on the market, or in other stores. Used by the seller to indicate a regular value or the price before discount.
- currency_buyer: currency of the prices.
- units_sold: Number of units sold.
- uses_ad_boosts: Whether the seller paid to boost his product within the platform (highlighting, better placement).
- rating: Mean product rating.
- rating_count: Total number of ratings of the product.
- rating_five_count, rating_four_count, rating_three_count, rating_two_count, rating_one_count: Number of 5, 4, 3, 2, and 1-star ratings.
- badges_count: Number of badges the product or the seller have.
- badge_local_product: A badge that denotes the product is a local product. Conditions may vary (being produced locally). Some people may prefer buying local products. 1 means product has the badge.
- badge_product_quality: Badge awarded when many buyers consistently gave good evaluations. 1 means product has the badge.
- badge_fast_shipping: Badge awarded when this product's order is consistently shipped rapidly.
- tags: tags set by the seller.
- product_color: Product's main color.
- product_variation_size_id: One of the available size variation for this product.
- product_variation_inventory: Inventory the seller has. Max allowed quantity is 50.
- shipping_option_name :Name of shipping option.
- shipping_option_price: shipping price.
- shipping_is_express: whether the shipping is express or not. 1 for True.
- countries_shipped_to: Number of countries this product is shipped to. Sellers may choose to limit where they ship a product to.
- inventory_total: Total inventory for all the product's variations (size/color variations for instance).
- has_urgency_banner: whether there was an urgency banner with an urgency.

- urgency_text: A text banner that appear over some products in the search results.
- origin_country: Country in which of merchant lives.
- merchant_title: Merchant's displayed name (show in the UI as the seller's shop name).
- merchant_name: Merchant's canonical name. A name not shown publicly. Used by the website under the hood as a canonical name.
- merchant_info_subtitle: The subtitle text as shown on a seller's info section to the user. (raw, not preprocessed). The website shows this to the user to give an overview of the seller's stats to the user. Mostly consists of % positive_feedbacks (rating_count reviews) written in French.
- merchant_rating_count: Number of ratings of this seller.
- merchant_rating: merchant's rating.
- merchant_id: merchant unique id.
- merchant_has_profile_picture: Convenience Boolean that says whether there is a `merchant_profile_picture` URL link.
- merchant_profile_picture: Custom profile picture of the seller (if the seller has one). Empty otherwise.
- product_URL: URL to the product page.
- product_picture: Picture of the product.
- product_id: product identifier.
- theme: the search term used in the search bar of the website to get these search results.
- crawl_month: (meta: for info only).

1.3 Executive summary

The data set consists of dependent feature *units_sold* and predictors as products rating, merchant rating and many others. Data exploratory analysis shows that the relationship between the predictors and *units_sold* is not marked by strong correlations and many of the features could be removed.
Regarding modeling, 3 general methods are considered:

- Generalized Linear Model (logistic regression model) or *glm*
- k-Nearest Neighbors *knn*
- Random Forests *rf*

Additionally, 7 further models arbitrarily were utilized by using an ensemble model. At the end the ensemble model consisting of 10 base models performs best and outperforms the baseline model as well as the individual methods *rf*, *knn*, and *glm*.

2) Data exploratory analysis

2.1 Data wrangling

```r
class(raw_data)
```

```
## [1] "spec_tbl_df" "tbl_df"      "tbl"         "data.frame"
```

```r
dim(raw_data)
```

```
## [1] 1573   43
```

```r
glimpse(raw_data)
```

```
## Rows: 1,573
## Columns: 43
## $ title                        <chr> "2020 Summer Vintage Flamingo Print  Pajamas Set...
## $ title_orig                   <chr> "2020 Summer Vintage Flamingo Print  Pajamas Set...
## $ price                        <dbl> 16.00, 8.00, 8.00, 8.00, 2.72, 3.92, 7.00, 12.00...
## $ retail_price                 <dbl> 14, 22, 43, 8, 3, 9, 6, 11, 84, 22, 5, 8, 6, 42,...
## $ currency_buyer               <chr> "EUR", "EUR", "EUR", "EUR", "EUR", "EUR", "EUR",...
## $ units_sold                   <dbl> 1e+02, 2e+04, 1e+02, 5e+03, 1e+02, 1e+01, 5e+04,...
## $ uses_ad_boosts               <dbl> 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, ...
## $ rating                       <dbl> 3.76, 3.45, 3.57, 4.03, 3.10, 5.00, 3.84, 3.76, ...
## $ rating_count                 <dbl> 54, 6135, 14, 579, 20, 1, 6742, 286, 15, 687, 61...
## $ rating_five_count            <dbl> 26, 2269, 5, 295, 6, 1, 3172, 120, 6, 287, 245, ...
## $ rating_four_count            <dbl> 8, 1027, 4, 119, 4, 0, 1352, 56, 2, 128, 101, 4,...
## $ rating_three_count           <dbl> 10, 1118, 2, 87, 2, 0, 971, 61, 3, 92, 81, 3, 24...
## $ rating_two_count             <dbl> 1, 644, 0, 42, 2, 0, 490, 18, 1, 68, 61, 0, 14, ...
## $ rating_one_count             <dbl> 9, 1077, 3, 36, 6, 0, 757, 31, 3, 112, 125, 3, 2...
## $ badges_count                 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ badge_local_product          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ badge_product_quality        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ badge_fast_shipping          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ tags                         <chr> "Summer,Fashion,womenunderwearsuit,printedpajama...
## $ product_color                <chr> "white", "green", "leopardprint", "black", "yell...
## $ product_variation_size_id    <chr> "M", "XS", "XS", "M", "S", "Size-XS", "XS", "M."...
## $ product_variation_inventory  <dbl> 50, 50, 1, 50, 1, 1, 50, 50, 50, 50, 2, 2, 1, 50...
## $ shipping_option_name         <chr> "Livraison standard", "Livraison standard", "Liv...
## $ shipping_option_price        <dbl> 4, 2, 3, 2, 1, 1, 2, 3, 2, 2, 2, 2, 1, 2, 1, 3, ...
## $ shipping_is_express          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ countries_shipped_to         <dbl> 34, 41, 36, 41, 35, 40, 31, 139, 36, 33, 25, 40,...
## $ inventory_total              <dbl> 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, ...
## $ has_urgency_banner           <dbl> 1, 1, 1, NA, 1, NA, NA, NA, 1, NA, 1, 1, NA, NA,...
## $ urgency_text                 <chr> "Quantité limitée !", "Quantité limitée !", "Qua...
## $ origin_country               <chr> "CN", "CN", "CN", "CN", "CN", "CN", "CN", "CN", ...
## $ merchant_title               <chr> "zgrdejia", "SaraHouse", "hxt520", "allenfan", "...
## $ merchant_name                <chr> "zgrdejia", "sarahouse", "hxt520", "allenfan", "...
## $ merchant_info_subtitle       <chr> "(568 notes)", "83 % avis positifs (17,752 notes...
## $ merchant_rating_count        <dbl> 568, 17752, 295, 23832, 14482, 65, 10194, 342, 3...
## $ merchant_rating              <dbl> 4.128521, 3.899673, 3.989831, 4.020435, 4.001588...
## $ merchant_id                  <chr> "595097d6a26f6e070cb878d1", "56458aa03a698c35c90...
## $ merchant_has_profile_picture <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ merchant_profile_picture     <chr> NA, NA, NA, NA, NA, NA, "https://s3-us-west-1.am...
## $ product_url                  <chr> "https://www.wish.com/c/5e9ae51d43d6a96e303acdb0...
```

```
## $ product_picture              <chr> "https://contestimg.wish.com/api/webimage/5e9ae5...
## $ product_id                   <chr> "5e9ae51d43d6a96e303acdb0", "58940d436a0d3d5da4e...
## $ theme                        <chr> "summer", "summer", "summer", "summer", "summer"...
## $ crawl_month                  <chr> "2020-08", "2020-08", "2020-08", "2020-08", "202...
```

One can observe that the raw data is mainly in tidy format but contains some NA values (not all chunks are in the report due to readability):

```
##  merchant_profile_picture        has_urgency_banner              urgency_text
##                      1347                      1100                      1100
##         rating_five_count         rating_four_count        rating_three_count
##                        45                        45                        45
##          rating_two_count          rating_one_count             product_color
##                        45                        45                        41
##            origin_country product_variation_size_id             merchant_name
##                        17                        14                         4
##    merchant_info_subtitle
##                         1
```

Features as *merchant_profile_picture* and *has_urgency_banner* have the most NA values. *Merchant_profile_picture* corresponds to *merchant_has_profile_picture* and thus, the whole column can be removed. *Urgency_text* and *urgency_banner* correspond to each other, thus the column *urgency_text* can be removed.
The features *rating_count_five* until *rating_count_one* contain NA values. They are replaced with '0', assuming these merchants have no ratings in the corresponding categories yet.

New features:
The feature *merchant_pos_feedback_rate* is derived from *merchant_info_subtitle* and calculated. New features as *Tags_number*, *shipping_price_percentage*, *discount_percent*, and *merchant_has_feedback_rate* (binary) are calculated.

Shipping_price_percentage ($\frac{price_{shippingoption}}{price_{product}}$) and *discount_percent* ($\frac{price_{retail}-price_{product}}{price_{retail}}$) could be more informative than their absolute values.
Features as *merchant_info_subtitle* and *tags* are removed afterwards. *Product_variation_size_id* is unified and simplified.

Further data preprocessing is conducted as follows:
Features with absolutely unique values as *crawl_month*, *currency_buyer*, and *theme* do not impact *units_sold* and are consequently removed. *Product URL* is not a valuable feature for a model since it does not impact *units_sold* and is consequently removed.
The specific case of whether a merchant has *title* in a foreign language besides product's *orig_title* is not being analyzed assuming that it would not have much impact on *units_sold*.
Since all products have *product_picture* and there is no other detailed information about the pictures, this feature will be removed, too.

```r
dim(processed_data)
```

```
## [1] 1573   37
```

```r
summary(processed_data)
```

```
##      price        retail_price      units_sold     uses_ad_boosts       rating
##  Min.   : 1.000   Min.   :  1.00   Min.   :     1   Min.   :0.0000   Min.   :0.000
##  1st Qu.: 5.810   1st Qu.:  7.00   1st Qu.:   100   1st Qu.:0.0000   1st Qu.:3.500
##  Median : 8.000   Median : 10.00   Median :  1000   Median :0.0000   Median :3.800
```

```
##  Mean   : 8.325   Mean   : 23.29   Mean   :  4339   Mean   :0.4329   Mean   :3.679
##  3rd Qu.:11.000   3rd Qu.: 26.00   3rd Qu.:  5000   3rd Qu.:1.0000   3rd Qu.:4.100
##  Max.   :49.000   Max.   :252.00   Max.   :100000   Max.   :1.0000   Max.   :5.000
##
##   rating_count     rating_five_count rating_four_count rating_three_count
##  Min.   :    0.0   Min.   :    0.0   Min.   :   0.0    Min.   :   0.0
##  1st Qu.:   24.0   1st Qu.:   10.0   1st Qu.:   4.0    1st Qu.:   3.0
##  Median :  150.0   Median :   72.0   Median :  29.0    Median :  22.0
##  Mean   :  889.7   Mean   :  429.6   Mean   : 174.5    Mean   : 130.7
##  3rd Qu.:  855.0   3rd Qu.:  394.0   3rd Qu.: 163.0    3rd Qu.: 121.0
##  Max.   :20744.0   Max.   :11548.0   Max.   :4152.0    Max.   :3658.0
##
##  rating_two_count  rating_one_count  badges_count    badge_local_product
##  Min.   :   0.00   Min.   :   0     Min.   :0.0000   Min.   :0.00000
##  1st Qu.:   1.00   1st Qu.:   3     1st Qu.:0.0000   1st Qu.:0.00000
##  Median :  10.00   Median :  18     Median :0.0000   Median :0.00000
##  Mean   :  61.89   Mean   :  93     Mean   :0.1055   Mean   :0.01844
##  3rd Qu.:  59.00   3rd Qu.:  90     3rd Qu.:0.0000   3rd Qu.:0.00000
##  Max.   :2003.00   Max.   :2789     Max.   :3.0000   Max.   :1.00000
##
##  badge_product_quality badge_fast_shipping product_color      product_variation_size_id
##  Min.   :0.00000       Min.   :0.00000     Length:1573        Length:1573
##  1st Qu.:0.00000       1st Qu.:0.00000     Class :character   Class :character
##  Median :0.00000       Median :0.00000     Mode  :character   Mode  :character
##  Mean   :0.07438       Mean   :0.01271
##  3rd Qu.:0.00000       3rd Qu.:0.00000
##  Max.   :1.00000       Max.   :1.00000
##
##  product_variation_inventory shipping_option_name shipping_option_price
##  Min.   : 1.00               Length:1573          Min.   : 1.000
##  1st Qu.: 6.00               Class :character     1st Qu.: 2.000
##  Median :50.00               Mode  :character     Median : 2.000
##  Mean   :33.08                                    Mean   : 2.345
##  3rd Qu.:50.00                                    3rd Qu.: 3.000
##  Max.   :50.00                                    Max.   :12.000
##
##  shipping_is_express countries_shipped_to inventory_total has_urgency_banner
##  Min.   :0.000000    Min.   :  6.00       Min.   : 1.00   Min.   :0.0000
##  1st Qu.:0.000000    1st Qu.: 31.00       1st Qu.:50.00   1st Qu.:0.0000
##  Median :0.000000    Median : 40.00       Median :50.00   Median :0.0000
##  Mean   :0.002543    Mean   : 40.46       Mean   :49.82   Mean   :0.3007
##  3rd Qu.:0.000000    3rd Qu.: 43.00       3rd Qu.:50.00   3rd Qu.:1.0000
##  Max.   :1.000000    Max.   :140.00       Max.   :50.00   Max.   :1.0000
##
##  origin_country     merchant_title     merchant_name      merchant_rating_count
##  Length:1573        Length:1573        Length:1573        Min.   :      0
##  Class :character   Class :character   Class :character   1st Qu.:   1987
##  Mode  :character   Mode  :character   Mode  :character   Median :   7936
##                                                           Mean   :  26496
##                                                           3rd Qu.:  24564
##                                                           Max.   :2174765
##
##  merchant_rating merchant_id        merchant_has_profile_picture  product_id
##  Min.   :2.300   Length:1573        Min.   :0.0000               Length:1573
```

```
##  1st Qu.:3.900   Class :character   1st Qu.:0.0000               Class :character
##  Median :4.000   Mode  :character   Median :0.0000               Mode  :character
##  Mean   :4.033                      Mean   :0.1437
##  3rd Qu.:4.200                      3rd Qu.:0.0000
##  Max.   :5.000                      Max.   :1.0000
##
##  merchant_pos_feedback_rate merchant_has_feedback_rate  tags_number    discount_percent
##  Min.   : 33.00             Min.   :0.0000             Min.   : 7.00   Min.   :0.0000
##  1st Qu.: 83.00             1st Qu.:1.0000             1st Qu.:13.00   1st Qu.:0.0000
##  Median : 86.00             Median :1.0000             Median :16.00   Median :0.0600
##  Mean   : 85.42             Mean   :0.8118             Mean   :16.39   Mean   :0.3089
##  3rd Qu.: 88.00             3rd Qu.:1.0000             3rd Qu.:19.00   3rd Qu.:0.7200
##  Max.   :100.00             Max.   :1.0000             Max.   :40.00   Max.   :0.9700
##  NA's   :296
##  shipping_price_percentage
##  Min.   :0.1700
##  1st Qu.:0.2500
##  Median :0.2900
##  Mean   :0.3008
##  3rd Qu.:0.3300
##  Max.   :1.1700
##
```

Still four features have NA values with *product_color* leading. These NAs are not removed immediately before analyzing the impact of the corresponding features or columns on *units_sold*.
Data contains 1341 unique products and 958 unique merchants.

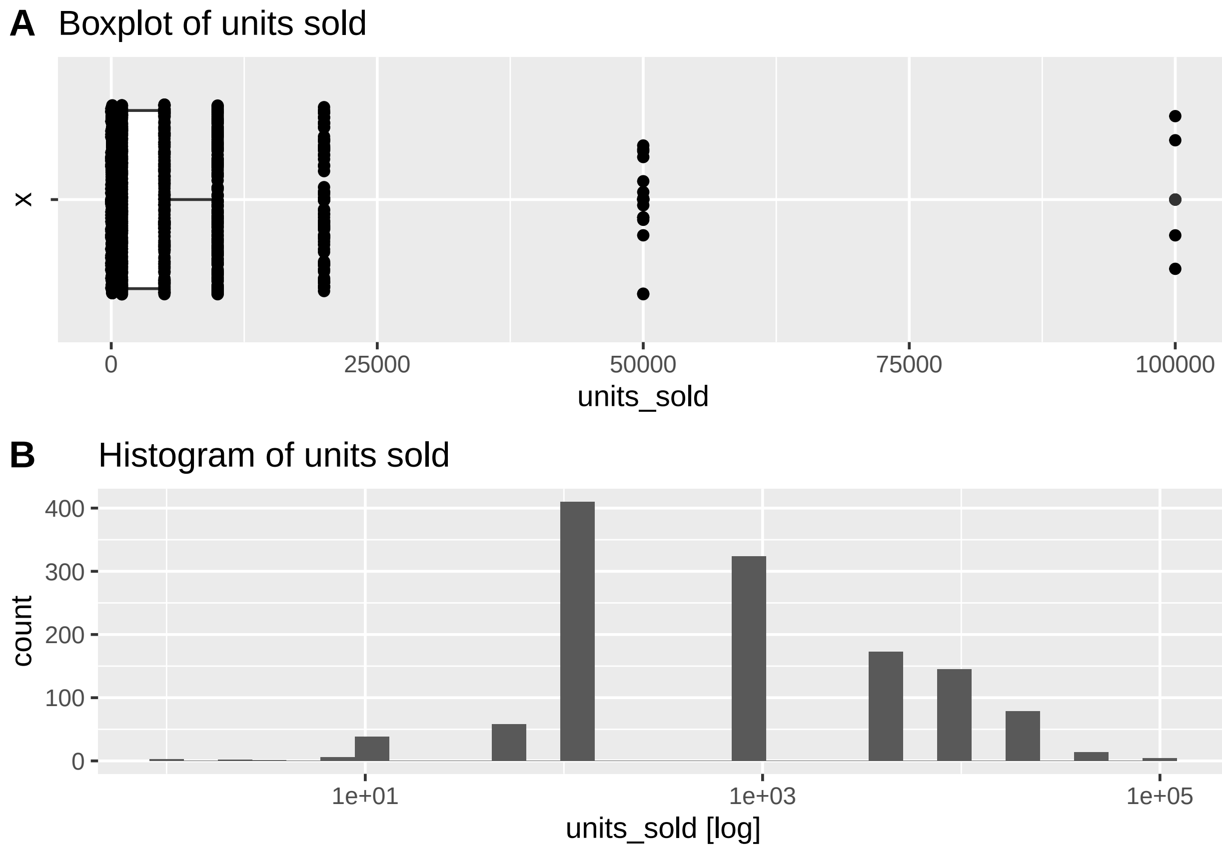
Data partition:
Train set for model training purposes and validation set for calculating final RMSE value are created. Whole dat set has only 1573 observations, thus validation set should consist of 20% of data to have a valid data amount to evaluate model performance.

```
set.seed(1, sample.kind = "Rounding")
test_index<- createDataPartition(processed_data$units_sold, times = 1, p=.2, list = F)
train_set<-processed_data[-test_index,]
validation<-processed_data[test_index,]
rm(test_index)
```

2.2 Data exploration

2.2.1 Distribution of units_sold

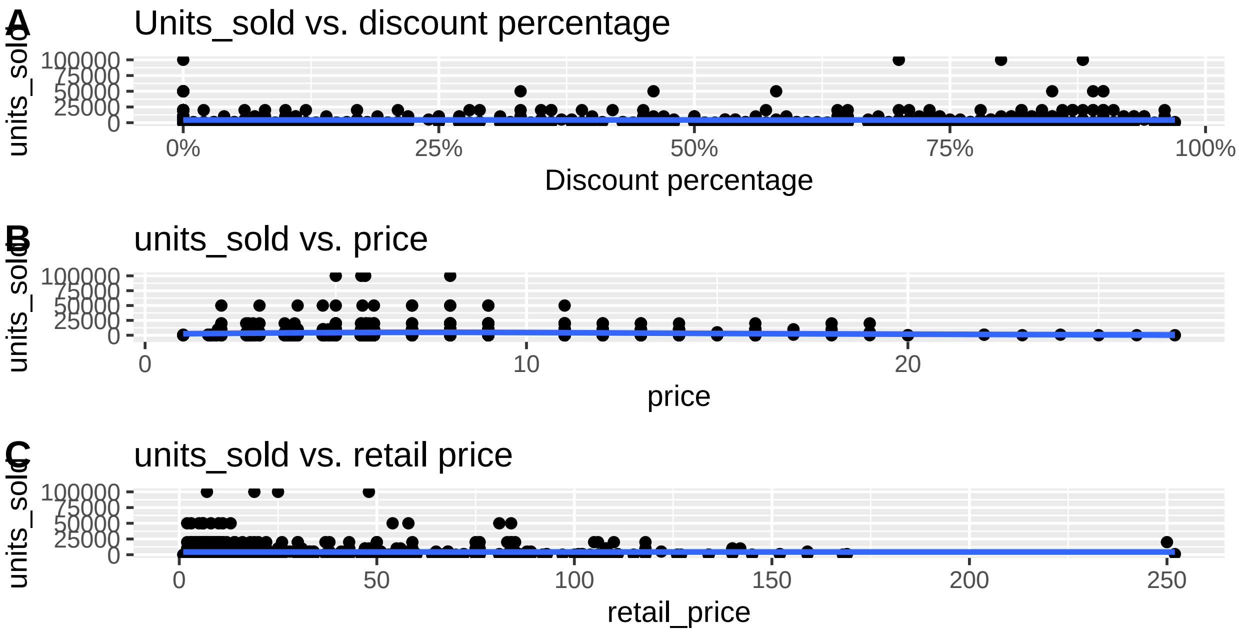
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       1     100    1000    4267    5000  100000
```



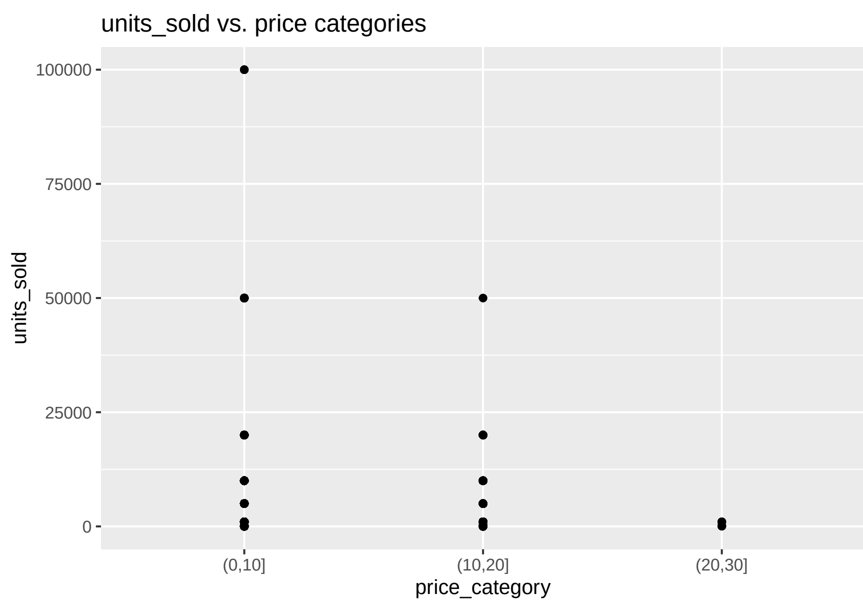
A and B: Regarding the distribution of the feature 'units_sold' one observes that the its median is 1000 units with a mean of 4267 units (showing slight skewness of data to the right). Most common values are 100 units and 1,000 units. There are very less merchants having *units_sold* consisting of 100,000 units and no merchant with zero units.
units_sold over 12,000 units could be outliers but without any further information (which is not existent in the data) they should not be removed. It is notable that there are "white spaces", i.e. no values for example between 100 and 1,000.

2.2.2 Price sensitivity analysis

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   5.830   8.000   8.298  11.000  27.000
```

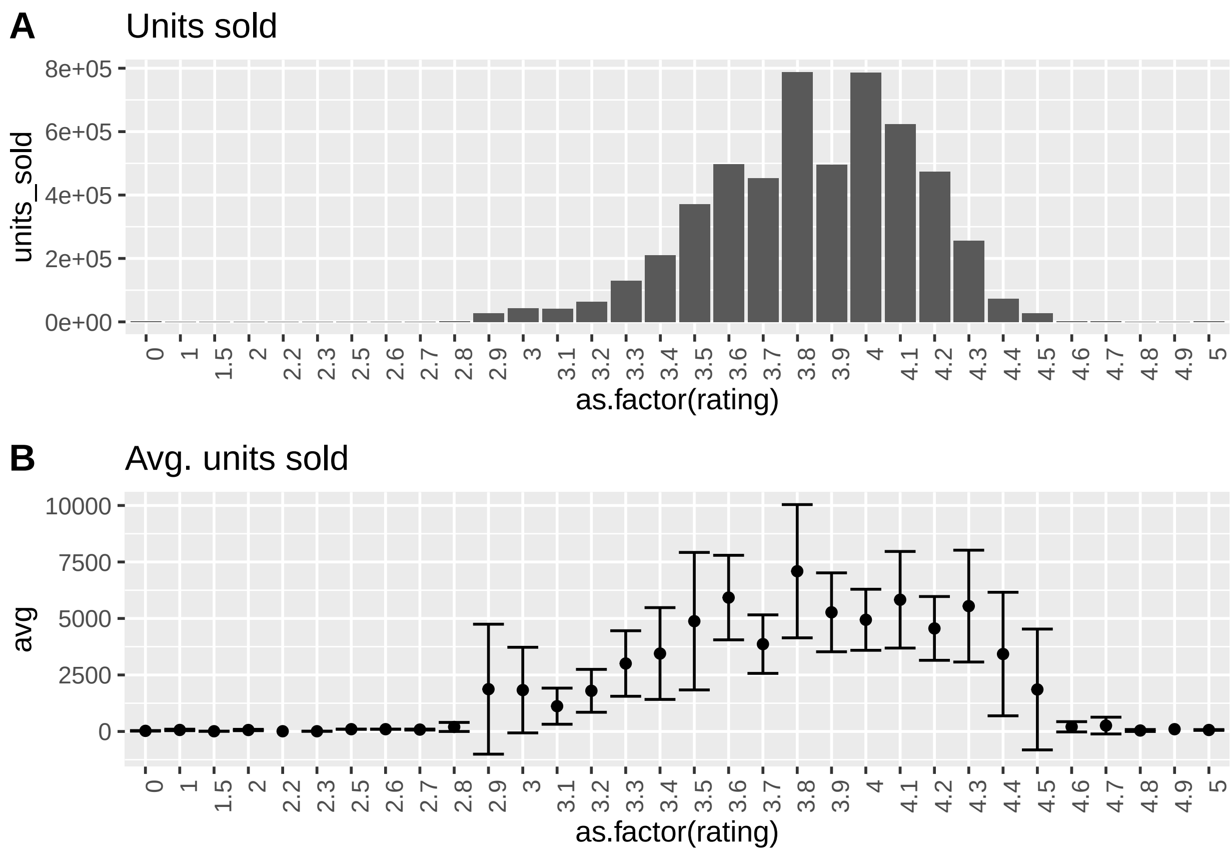


The median of price is 8.00 EUR and its mean is 8.3 EUR, thus 50% of products having prices below the mean.
A until C: According to the plots *price* and *retail_price* seem to have low negative impact on *units_sold*. The quotient of these two features *discount_percent* very slightly impacts *units_sold*. Thus, customers are mainly price insensitive.

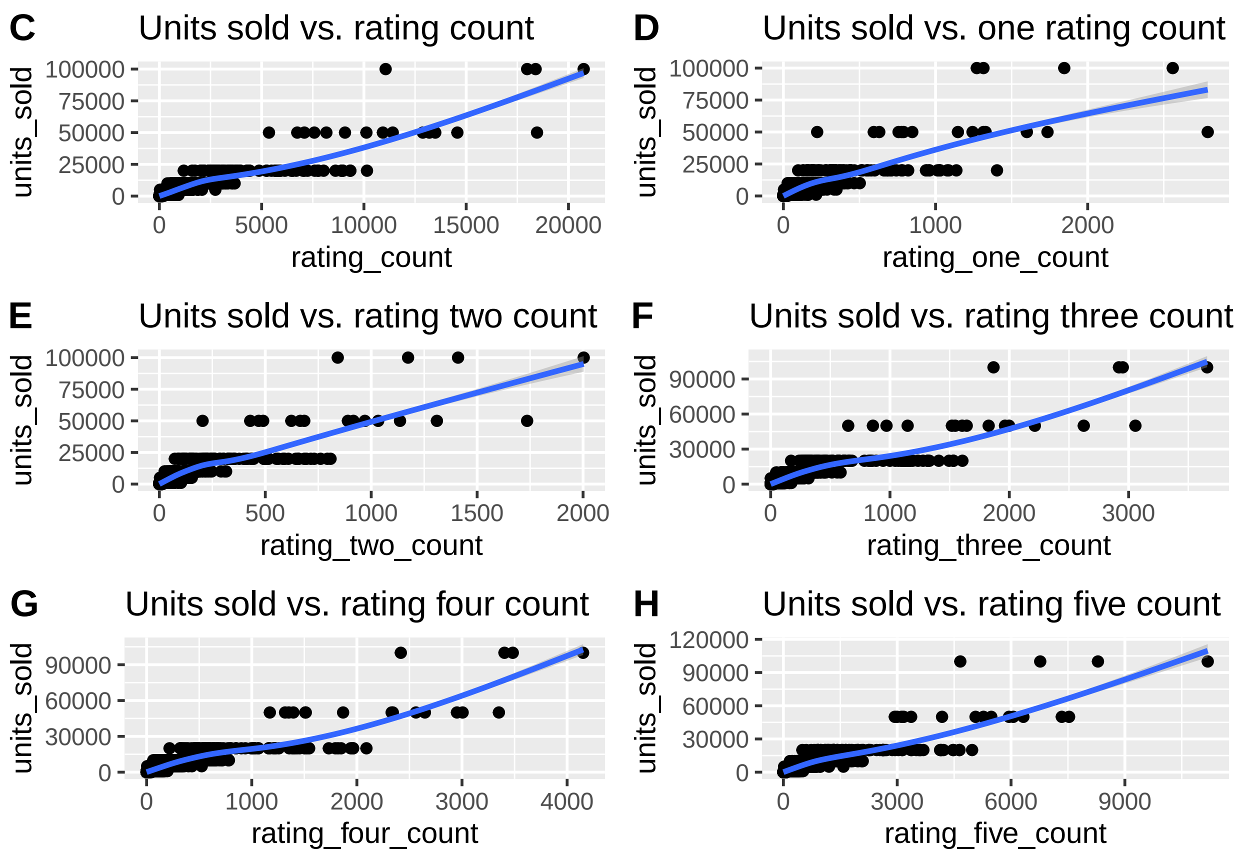


The last plot clearly shows the negative impact of the feature *price* or price categories on the outcome *units_sold*.

2.2.3 Product attribute analysis

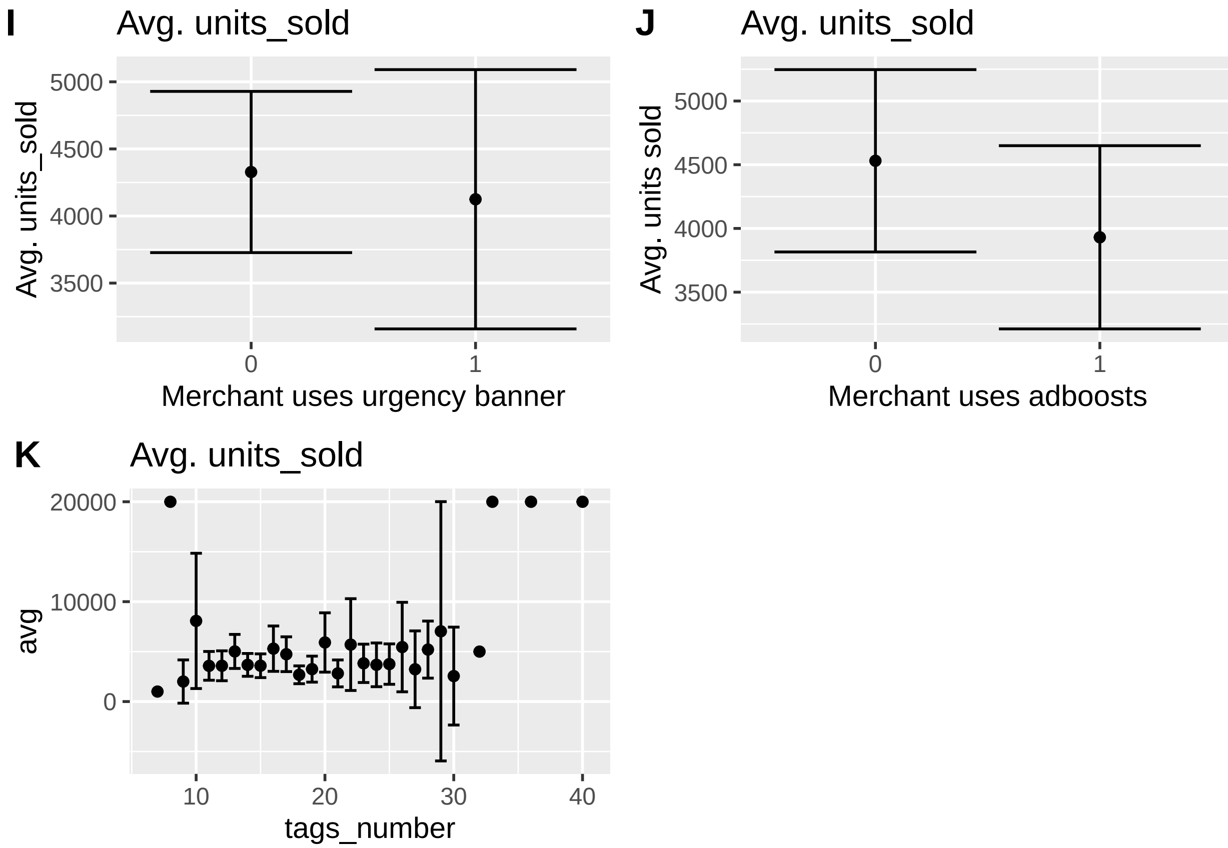


A & B: Plots indicate that product *rating* has a strong impact on *units_sold*.

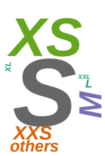


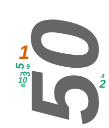
C: *Rating_count* has an impact on *units_sold*, too. This makes sense because higher *units_sold* high probably leads to more and more product rating counts. The direction of action is strictly speaking one-way: higher *units_sold* means higher rating counts, thus **rating count can not** be considered in the model. Furthermore, *rating_count* is a mixed combination of positive and negative (1 and 2 star) ratings making it not suitable for modeling.

D until H: All 1, 2, 3, 4, and 5 star ratings are positively correlated with *units_sold*. However, the slope of the smooth function for 1 and 2 star ratings gets smaller the higher the rating counts.

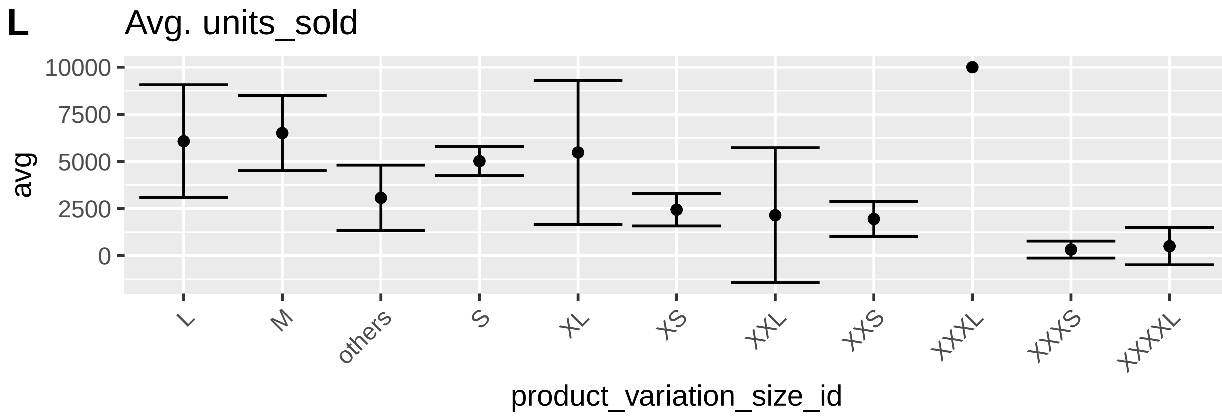


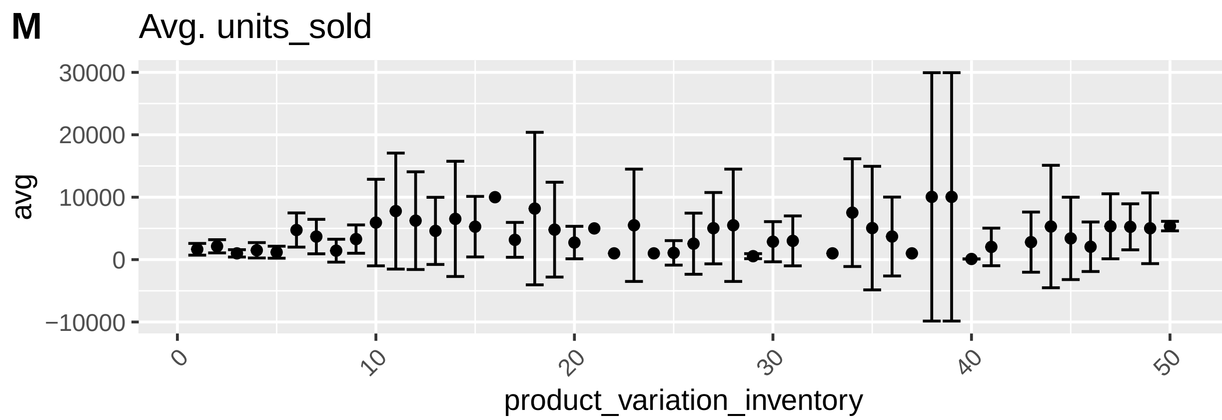
I until K: Features as *merchant_use_adboosts*, *tags_number* or *urgency_banner* do not impact *units_sold.* Hypothetically, merchants having low level of *units_sold* use adboosts expecting higher volumes of revenue.





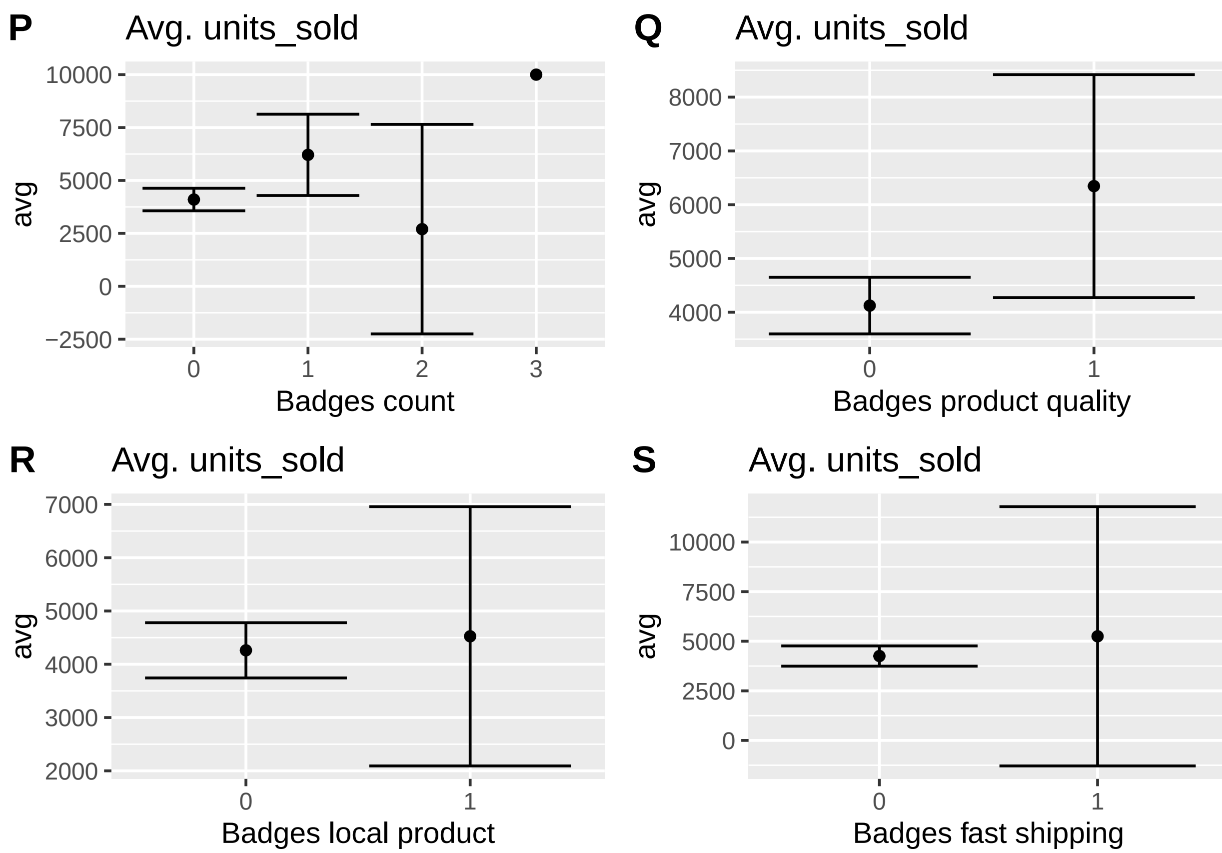
‘S’, ‘XS’ , and ‘M’ are the most prevalent sizes. The most prevalent product variation inventory is by far ‘50’ units.





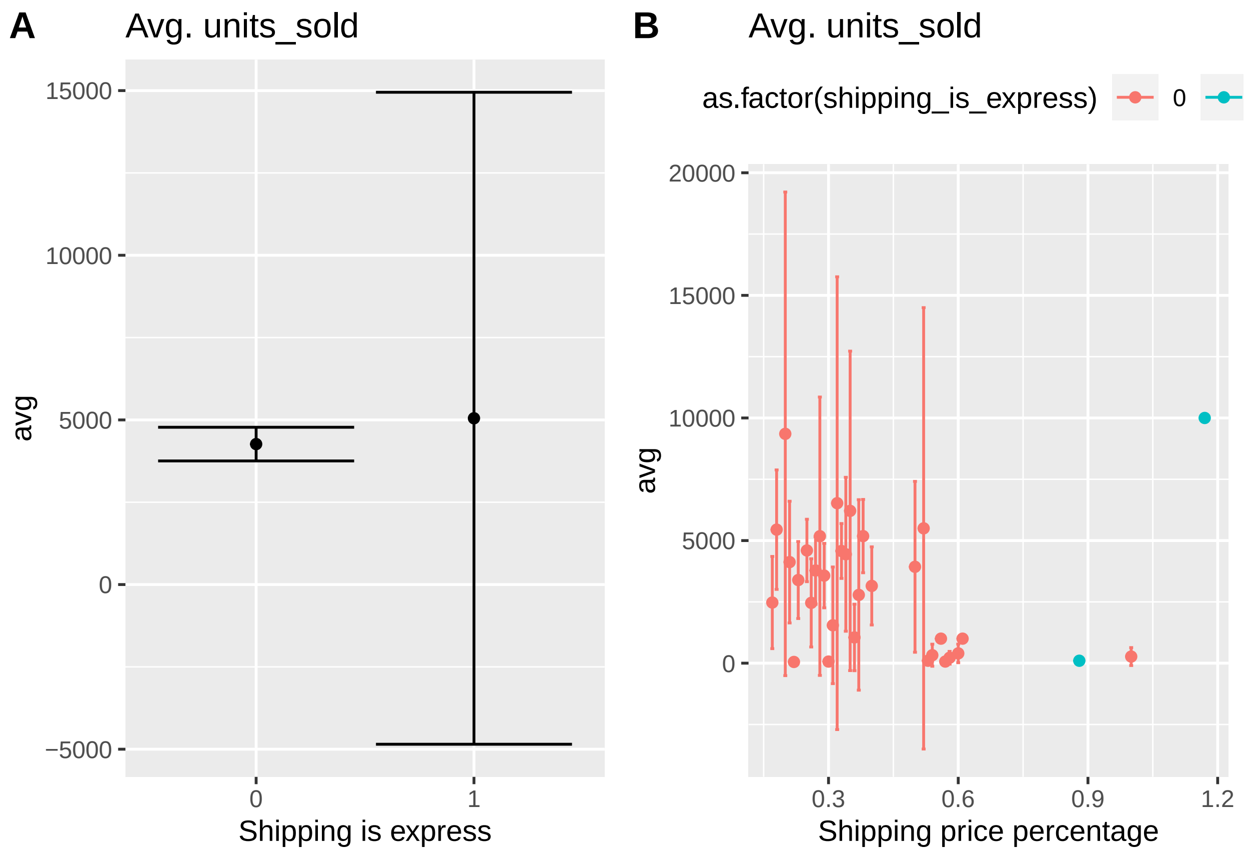
L: *Product_variation_size* impacts *units_sold* as the mean of the different variations differ.
M: However, *product_variation_inventory* very poorly affects *units_sold*.

N: *Product_colors* as black, white, grey, blue , or green are very popular and demanded.
O: Plot shows that the feature *product_color* do not necessarily impact *units_sold*.

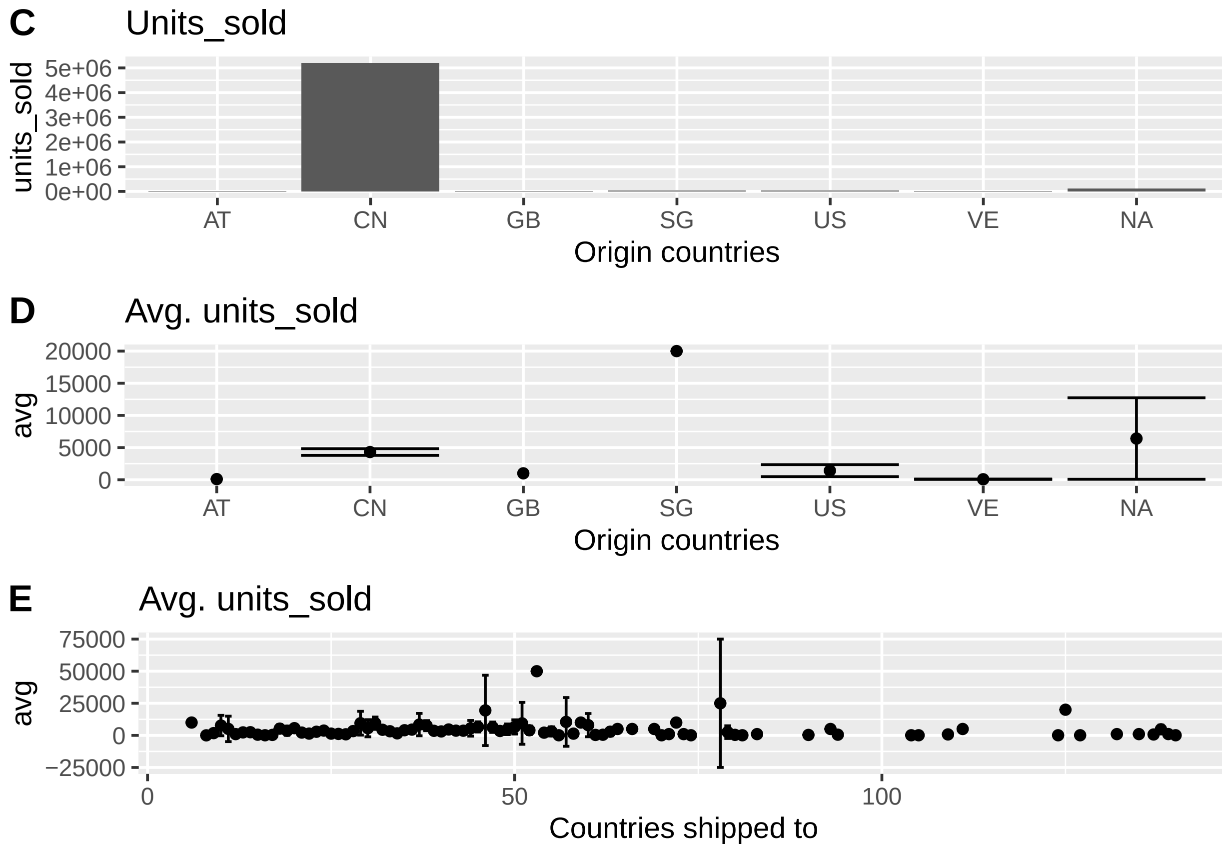


P: A higher number of *badges* does not necessarily lead to higher *units_sold*.
Q: The badge *product_quality* seems to impact *units_sold*.
R & S: Badges as *local_product* or *fast_shipping* do not necessarily lead to higher *units_sold*.

2.2.4 Product logistics

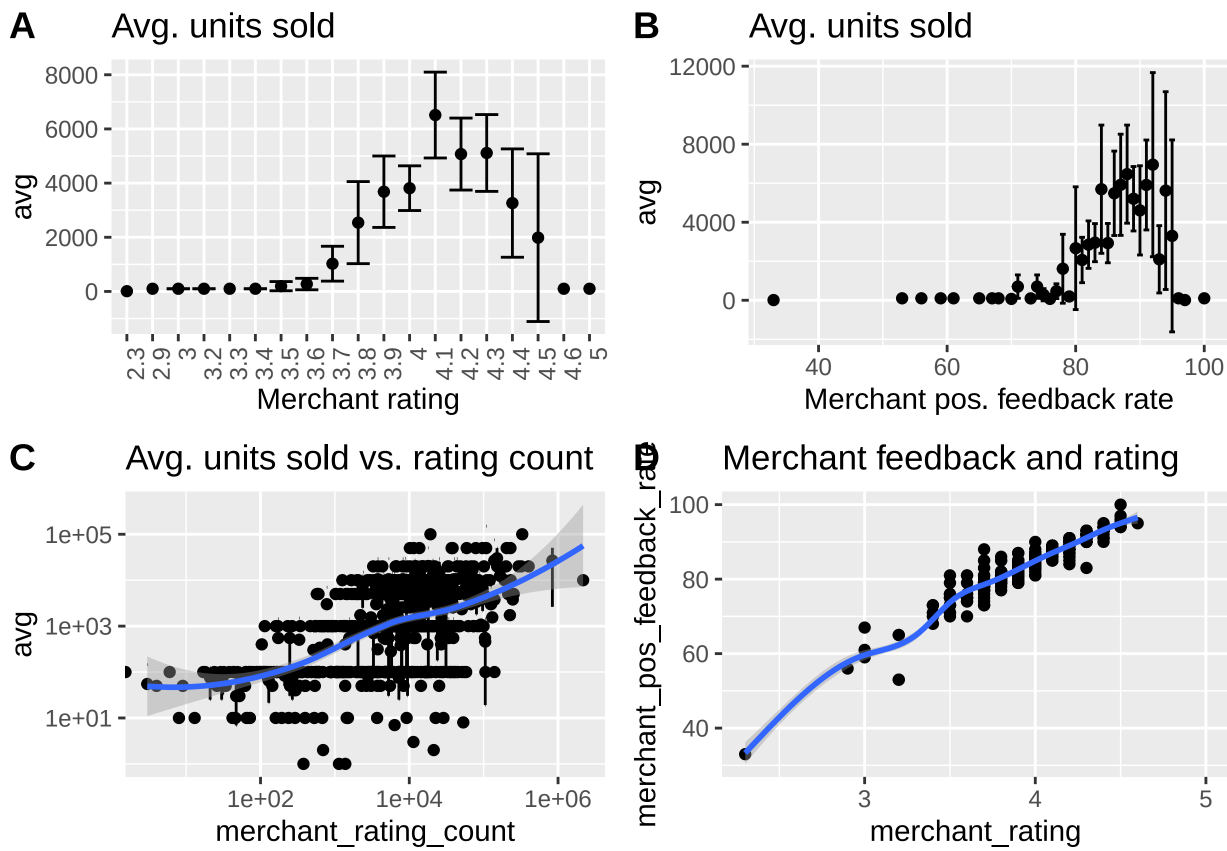


A: Products marked as '*shipping_is_express*' do not lead to higher *units_sold*.
B: *Shipping_price_percentage* has a slight negative impact on *units_sold*.

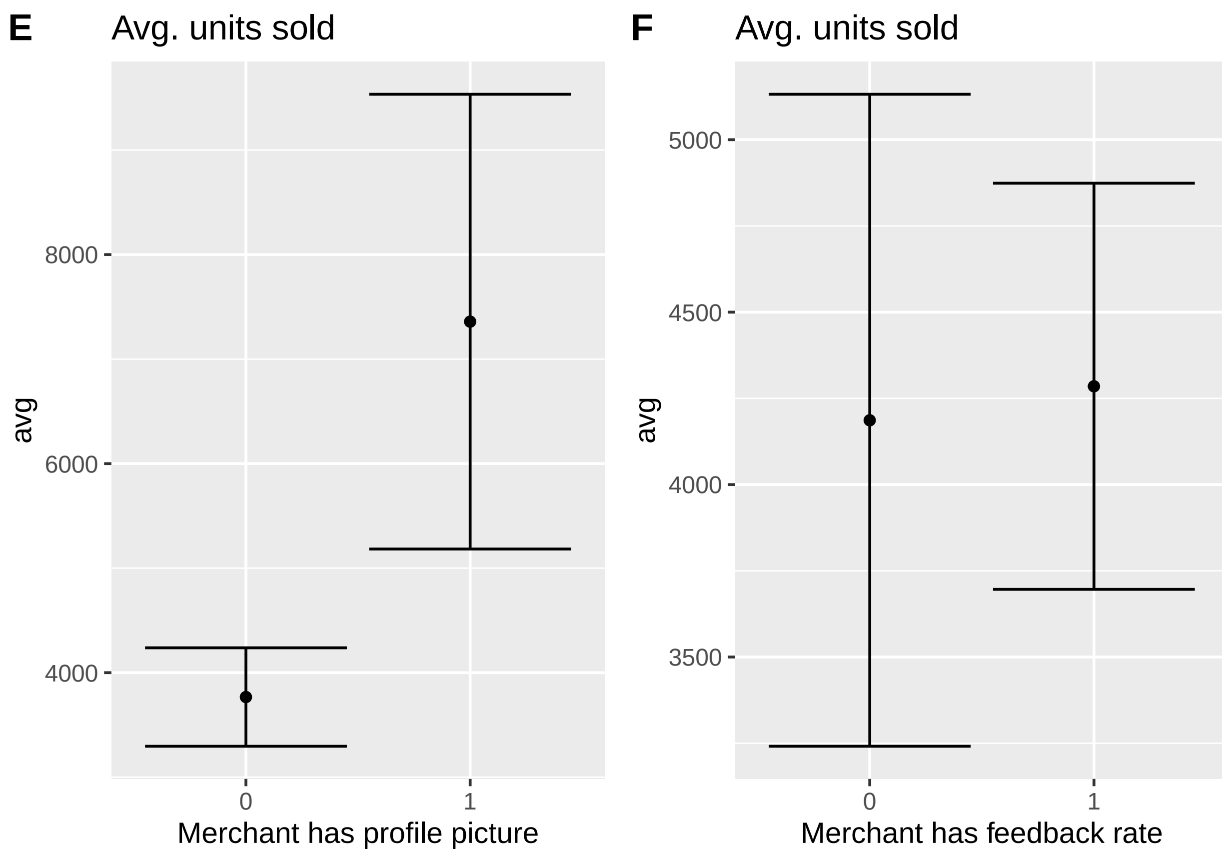


C until E: *Units_sold* do not differ regarding geographical features as *origin_country* or *countries_shipped_to.*

2.2.5 Merchant attribute analysis



A until C: The diagrams above indicate positive correlations between average *units_sold* and the features *merchant_rating*, *merchant_rating_count*, and *merchant_pos_feedback_rate.*
Merchant_rating_count is not considered in the model for the same reasons as product *rating_count.*
D: The features *merchant_rating* and *merchant_pos_feedback_rate* are highly correlated, thus these two features should be summarized as one feature for the sake of dimension reduction. The new created feature *merchant_pos_feedback_rate* will be removed.



E: According to the error bar diagram *merchant_has_profile_picture* has a positive impact on average *units_sold*.
F: Whether a merchant has a feedback rate in the merchant info subtitle or not, it does not impact *units_sold*.

Finally, train set and validation set are adjusted according to the insights from data exploratory analysis. For example, *Product_variation_size_id* is 'one-hot-encoded'.

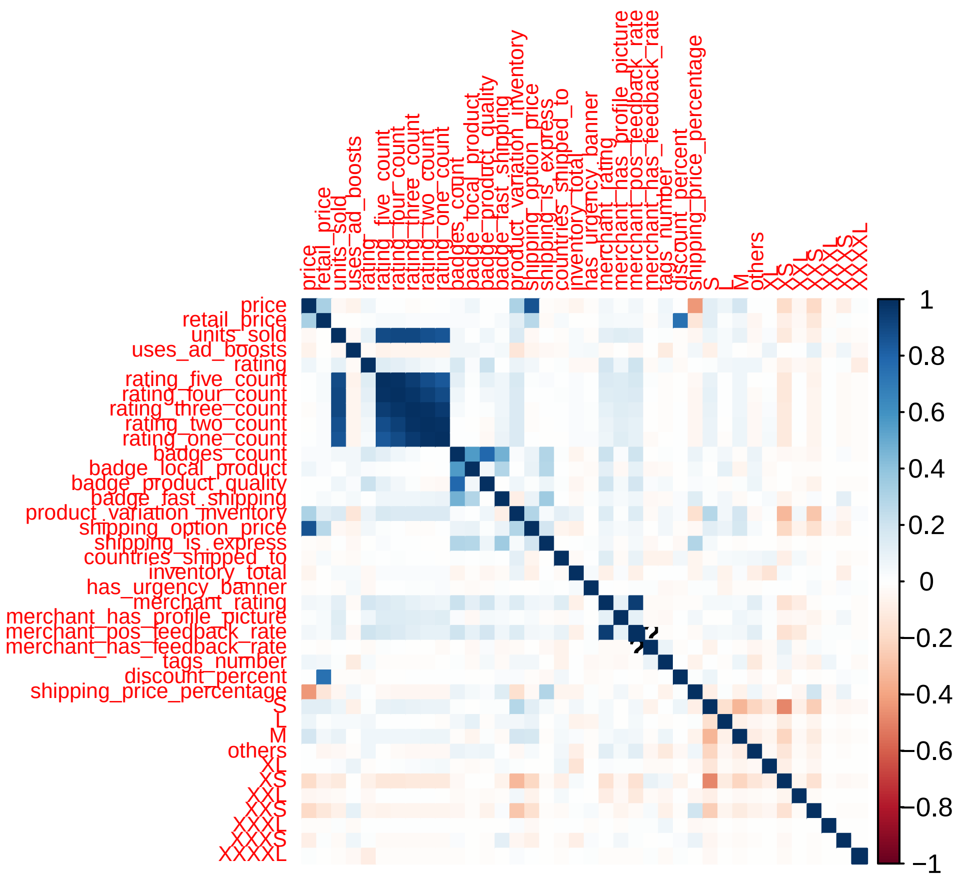
2.3 Data preprocessing

2.3.1 Correlation heat map

A correlation heat map is an adequate method for summarizing correlations between all numerical variables. This analysis is used to identify irrelevant features as well as highly correlated features.

Train set dimension:

```
## [1] 1257   38
```



Following statements can be derived from the plot above:

- Features positively correlated to *units_sold*:
  - rating
  - rating_five_count, rating_four_count, rating_three_count, rating_two_count (decreasing slope beginning with approximately 1500 ratings!), rating_one_count (decreasing slope beginning with approximately 2000 ratings!)
  - badges_count, badge_product_quality
  - product_variation_inventory
  - merchant_rating
  - merchant_has_profile_picture
  - merchant_positive_feedback

- Features negatively correlated to *units_sold*:
  - shipping_price_percentage
  - shipping_option_price

    - merchant_use_adboosts
    - price

- Features having very low or no correlation with *units_sold*:
    - retail_price
    - badge_local_product, badge_fast_shipping
    - shipping_is_express
    - countries_shipped_to
    - inventory_total
    - urgency_banner
    - tags_number
    - discount_percent
    - merchant_with_feedback vs. no feedback
    - all product variation sizes

- Correlated features are:
    - rating_count, rating_five_count until rating_one_count
    - merchant_rating_count and merchant_rating with rating_count, rating_five_count until rating_one_count
    - merchant_rating and product_rating
    - merchant_rating and merchant_positive_feedback

Consequently, features or columns having no correlation with *units_sold* are removed from the data set.

```
train_set <- train_set %>% select(-c(retail_price,
    badge_local_product, badge_fast_shipping, shipping_is_express,
    countries_shipped_to, inventory_total, has_urgency_banner,
    tags_number, discount_percent, merchant_has_feedback_rate,
    merchant_pos_feedback_rate, product_variation_size_id,
    S, L, M, others, XL, XS, XXL, XXS, XXXL, XXXS,
    XXXXL))

validation <- validation %>% select(-c(retail_price,
    badge_local_product, badge_fast_shipping, shipping_is_express,
    countries_shipped_to, inventory_total, has_urgency_banner,
    tags_number, discount_percent, merchant_has_feedback_rate,
    merchant_pos_feedback_rate, product_variation_size_id,
    S, L, M, others, XL, XS, XXL, XXS, XXXL, XXXS,
    XXXXL))
```

```
top_6_merchant<-train_set %>% filter(units_sold>=100000)
top_6_merchant
```

```
## # A tibble: 4 x 23
##   price units_sold uses_ad_boosts rating rating_five_cou~ rating_four_cou~
##   <dbl>      <dbl>          <dbl>  <dbl>            <dbl>            <dbl>
## 1  5        100000              1    3.8             8290             3483
## 2  5.77     100000              0    4.1            11184             4152
## 3  8        100000              1    3.8             4663             2418
## 4  5.67     100000              0    3.5             6769             3404
## # ... with 17 more variables: rating_three_count <dbl>, rating_two_count <dbl>,
## #   rating_one_count <dbl>, badges_count <dbl>, badge_product_quality <dbl>,
## #   product_color <chr>, product_variation_inventory <dbl>, shipping_option_name <chr>,
## #   shipping_option_price <dbl>, origin_country <chr>, merchant_title <chr>,
```

```
## #   merchant_name <chr>, merchant_rating <dbl>, merchant_id <chr>,
## #   merchant_has_profile_picture <dbl>, product_id <chr>, shipping_price_percentage <dbl>
```

Illustratively, the table above shows the top six merchants. It could be observed that these merchants have high product and merchant ratings (>3.5), only three of them use ad boosts without any badges, they have proportionally high numbers of 5-star ratings, all operate from Canada, and offering product sizes S and M.

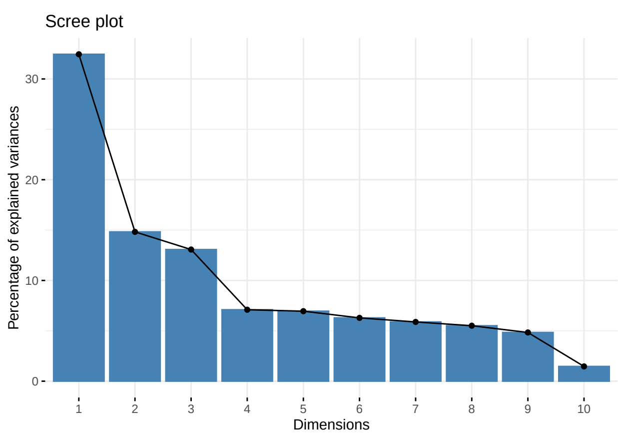
2.3.2 Principle component analysis

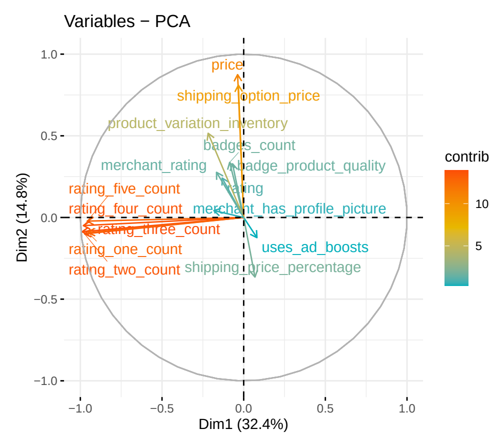
Principle component analysis (PCA) is a common method for dimension reduction regarding data sets with high number of features. The train data set has still 23 features while it consists of 1257 observations (rows). Thus, a dimension reduction would be reasonable.

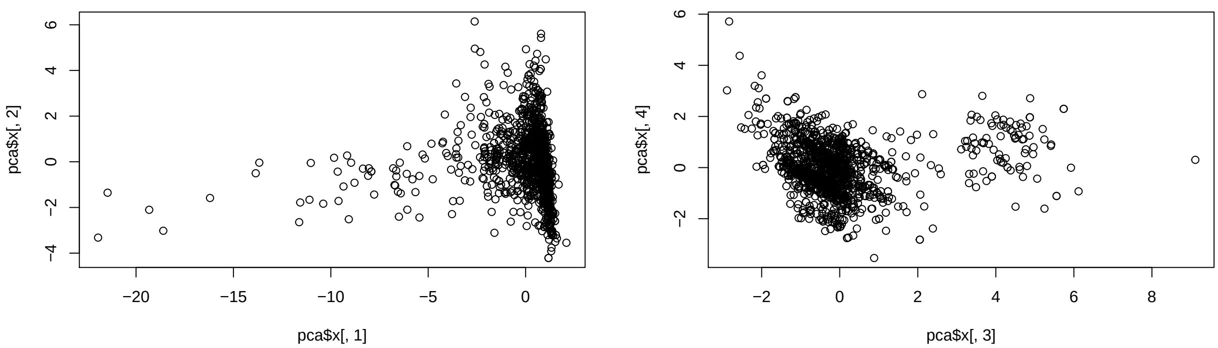
```
train_x<-select_if(train_set, is.numeric) %>% select(-units_sold) %>% as.matrix()
train_y<-train_set$units_sold

pca<-prcomp(train_x, center = T, scale. = T)
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6     PC7     PC8
## Standard deviation     2.2062 1.4911 1.4000 1.03144 1.02098 0.97086 0.93927 0.90879
## Proportion of Variance 0.3245 0.1482 0.1307 0.07092 0.06949 0.06284 0.05882 0.05506
## Cumulative Proportion  0.3245 0.4727 0.6034 0.67431 0.74381 0.80664 0.86546 0.92052
##                            PC9    PC10    PC11    PC12    PC13    PC14    PC15
## Standard deviation     0.85172 0.46867 0.41842 0.21312 0.13854 0.07517 0.04242
## Proportion of Variance 0.04836 0.01464 0.01167 0.00303 0.00128 0.00038 0.00012
## Cumulative Proportion  0.96888 0.98352 0.99520 0.99822 0.99950 0.99988 1.00000
```

Only 8 PCs account for over 90% of the data set variation with PC1 accounting for the most variance. Thus, the following analysis is conducted on these PCs.







The plots of the x-values of the PC's one until four do not show any obvious patterns.
As the PCA loadings show the features *rating_star_counts* have the most contribution to PC1 and the feature *price* has the most contribution to PC2 followed by *shipping_option_price.*

According to results of PCA train set and validation set are transformed.

```
imp<-8
train_x_imp<-pca$x[,1:imp]
validation_x<-select_if(validation, is.numeric)  %>% select(-units_sold)%>% as.matrix()
validation_y<-validation$units_sold
validation_x_mean_0 <- sweep(validation_x, 2, colMeans(validation_x))
validation_x_standardized <- sweep(validation_x_mean_0, 2, colSds(validation_x), FUN = "/")
validation_x_pca<-validation_x_standardized %*% pca$rotation
validation_x_imp <- validation_x_pca[,1:imp]
rm(validation_x_mean_0, validation_x_standardized, validation_x_pca)
```

Now data has reasonable features and dimension complexity ready for modeling.

3) Modeling and results

3.1 Prediction models

Before models are trained and finally evaluated on validation set, the train set is divided in train_sub (90%) and test set in order to test (10%) and compare the different prediction algorithms and choose the final model.

```
set.seed(1, sample.kind = "Rounding")
test_index<- createDataPartition(train_y, times = 1, p=.1, list = F)
train_x_imp_sub<-train_x_imp[-test_index,]
test_x_imp<-train_x_imp[test_index,]
train_y_sub<-train_y[-test_index]
test_y<-train_y[test_index]
rm(test_index)
```

If one is asked to give a 'guess' about the model output *units_sold*, a meaningful 'guess' would be the average of it which basically minimizes RMSE. Thus, a baseline model is built which is the average of Y (train set) for comparison purposes regarding RMSE values of different ML algorithms.
Furthermore, following common models from caret package are used in order to build first prediction models:

- Generalized Linear Model (logistic regression model) or *glm*
- k-Nearest Neighbors *knn*
- Random Forests *rf*

knn and rf-models are tuned in order to find the best model parameter as number of neighbors (k=1, 2, 3, …, 20) or number of variables randomly sampled as candidates at each split (mtry= 1, 2, 3, …, 8; due to 8 important PCs). Since there are only few features highly correlating with *units_sold* as product rating and rating star counts, and thus predictive, *rf* should be a good method for prediction.

```
y_hat_baseline<-mean(train_y_sub)
RMSE_baseline<-RMSE(test_y, y_hat_baseline)
RMSE_baseline
```

```
## [1] 6648.378
```

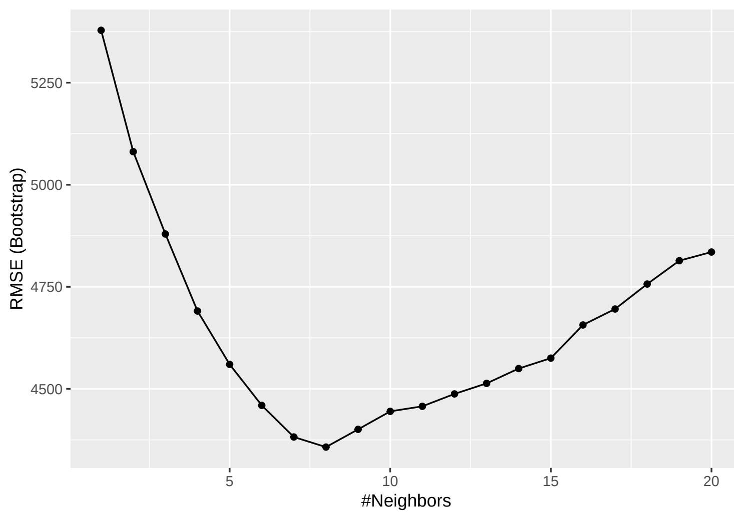
```
set.seed(100, sample.kind = "Rounding")
fit_glm<-caret::train(train_x_imp_sub, train_y_sub, method = "glm")
fit_glm$finalModel
```

```
##
## Call:  NULL
##
## Coefficients:
## (Intercept)          PC1          PC2          PC3          PC4          PC5
##    4282.187    -3829.290     -493.473       12.829      322.906       -9.473
##         PC6          PC7          PC8
##      57.484      109.979     -123.163
##
## Degrees of Freedom: 1128 Total (i.e. Null);  1120 Residual
## Null Deviance:       9.721e+10
## Residual Deviance: 1.609e+10     AIC: 21820
```

```
y_hat_glm <- predict(fit_glm, test_x_imp)
RMSE_glm<-RMSE(test_y, y_hat_glm)
RMSE_glm
```

```
## [1] 4040.449
```

```r
set.seed(100, sample.kind = "Rounding")
k<-seq(1,20,1)
fit_knn<-caret::train(train_x_imp_sub, train_y_sub, method = "knn", tuneGrid = data.frame(k=k))
ggplot(fit_knn)
```



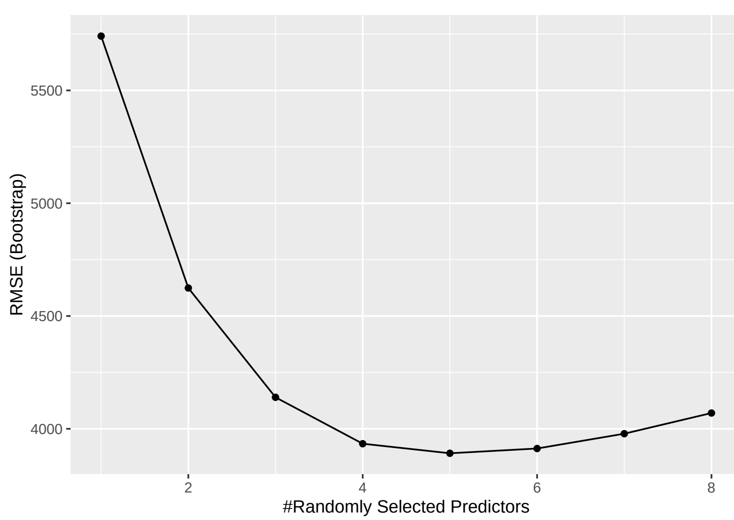

```r
fit_knn$finalModel
```

```
## 8-nearest neighbor regression model
```

```r
y_hat_knn <- predict(fit_knn, test_x_imp)
RMSE_knn<-RMSE(test_y, y_hat_knn)
RMSE_knn
```

```
## [1] 3237.567
```

```r
set.seed(100, sample.kind = "Rounding")
mtry<-seq(1, 8, 1)
fit_rf<-caret::train(train_x_imp_sub, train_y_sub, method = "rf", tuneGrid=data.frame(mtry=mtry))
ggplot(fit_rf)
```



```r
fit_rf$finalMode
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 15026913
##                     % Var explained: 82.55
```

```r
varImp(fit_rf)
```

```
## rf variable importance
##
##      Overall
## PC1 100.0000
## PC2   4.8456
## PC5   2.1116
## PC4   1.8243
## PC6   1.3907
## PC8   0.8670
## PC3   0.7346
## PC7   0.0000
```

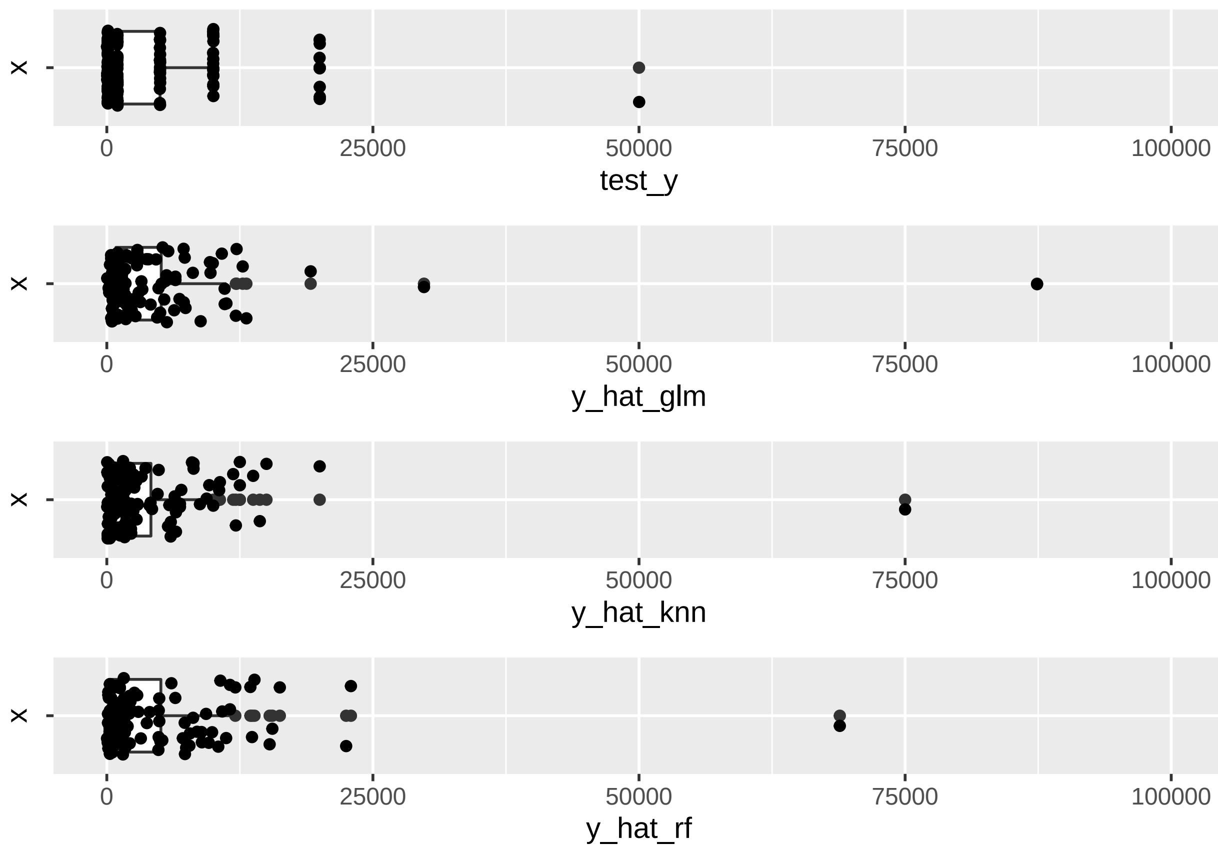
```r
y_hat_rf <- predict(fit_rf, test_x_imp)
RMSE_rf<-RMSE(test_y, y_hat_rf)
RMSE_rf
```

```
## [1] 2542.107
```

The baseline model has a RMSE of 6648. Best performing algorithm is *Random Forest* which has a RMSE of 2542 with the best tune parameter 5 and PC1 as expected as the most important feature:

$$\frac{RMSE(rf)}{RMSE(baseline)} = 0.38$$

However, *rf* requires much more computing capacity and time than all other ML algorithms.
After a first attempt one could say *rf* seems to be a reasonable model predicting future *units_sold*. Before choosing a final model it would be interesting to analyze the distributions of all the three ML algorithms:



Plot reveal that the two highest units of 50,000 units is best predicted by *rf*. Combining these ML algorithms could probably lead to better predictions through ensembling.

3.2 Ensemble model

In order to build an ensemble model the *caretEnsemble* package for making ensembles of caret models is being utilized.
caretEnsemble has three primary functions: *caretList*, *caretEnsemble* and *caretStack.*
caretList is used to build lists of caret models on the same training data, with the same re-sampling parameters. *caretEnsemble* and *caretStack* are used to create ensemble models from such lists of caret models. *caretEnsemble* uses a *glm* algorithm to create a simple linear blend of models and caretStack uses a caret model to combine the outputs from several component caret models. In this paper the *caretEnsemble* is being used.

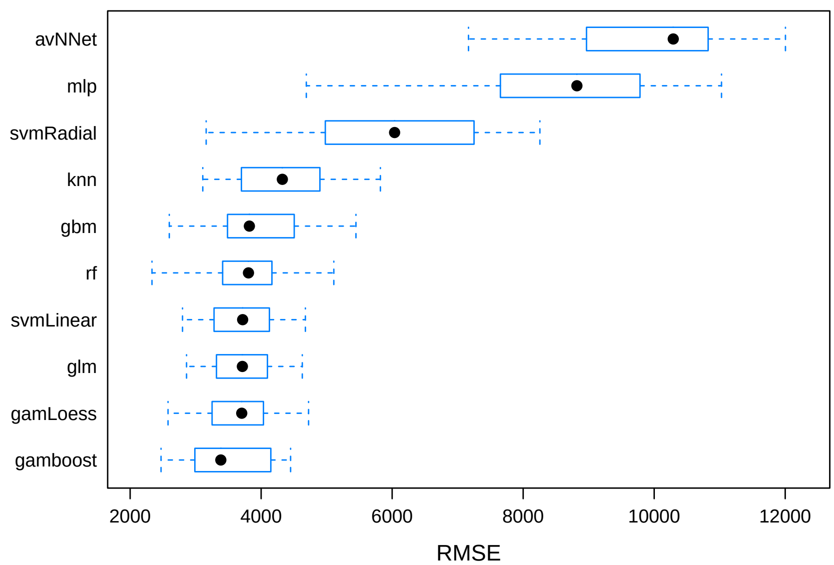
Following 10 base models from caret package are used in order to build an ensemble model:

- Generalized Linear Model (logistic regression model) or *glm*
- Support Vector Machines as *svmLinear*, and *svmRadial*
- Generalized Additive Models as *gamboost*, and *gamLoess*
- Neural Networks as (averaged) *avNNet*, Multilayer Perceptron *mlp*
- k-Nearest Neighbors *knn*
- Random Forest *rf*
- Generalized Boosted Regression Modeling *gbm*

Illustratively, the algorithms *rf* and *knn* are tuned.

```
models <-  c("glm", "svmLinear", "gamboost", "gamLoess", "avNNet", "mlp", "gbm", "svmRadial")

set.seed(100, sample.kind = "Rounding")
control <-  trainControl(method="boot", number=25,  savePredictions="final",  allowParallel = TRUE)
fits_list <- caretList(train_x_imp_sub, train_y_sub,  trControl=control,  methodList= models,
                       tuneList=list(
                         knn=caretModelSpec(method="knn", tuneGrid=data.frame(k=k)),
                         rf=caretModelSpec(method="rf", tuneGrid=data.frame(mtry=mtry))))
```

After training the different ML algorithms with *caretList* one can observe that the ML *gamboost* has the best RMSE on train data followed by *gamLoess* and *glm*. *avNNet* and *mlp* perform the worst on train data.



The correlation between models indicate that the models are highly correlated with each other except *mlp*. For a good performing ensemble the base models should be ideally uncorrelated.

```
# Model correlation matrix
modelCor(results)
```

```
##                 knn        rf       glm svmLinear  gamboost  gamLoess    avNNet       mlp
## knn       1.0000000 0.8322952 0.7407830 0.7374118 0.7137262 0.7248563 0.8724331 0.4089050
## rf        0.8322952 1.0000000 0.8214417 0.7682974 0.8019821 0.8353642 0.6984782 0.5538394
## glm       0.7407830 0.8214417 1.0000000 0.9383413 0.9117205 0.8419402 0.6909469 0.2404520
## svmLinear 0.7374118 0.7682974 0.9383413 1.0000000 0.9315789 0.7595130 0.7043057 0.1504017
## gamboost  0.7137262 0.8019821 0.9117205 0.9315789 1.0000000 0.7978958 0.7052713 0.3614490
## gamLoess  0.7248563 0.8353642 0.8419402 0.7595130 0.7978958 1.0000000 0.6135478 0.3238164
## avNNet    0.8724331 0.6984782 0.6909469 0.7043057 0.7052713 0.6135478 1.0000000 0.3476229
## mlp       0.4089050 0.5538394 0.2404520 0.1504017 0.3614490 0.3238164 0.3476229 1.0000000
## gbm       0.8585528 0.8627965 0.7990421 0.7064464 0.6721481 0.7879134 0.8027682 0.4212875
## svmRadial 0.8524613 0.7035250 0.6030814 0.5549658 0.5780205 0.5327929 0.8652348 0.4369388
##                 gbm svmRadial
## knn       0.8585528 0.8524613
```

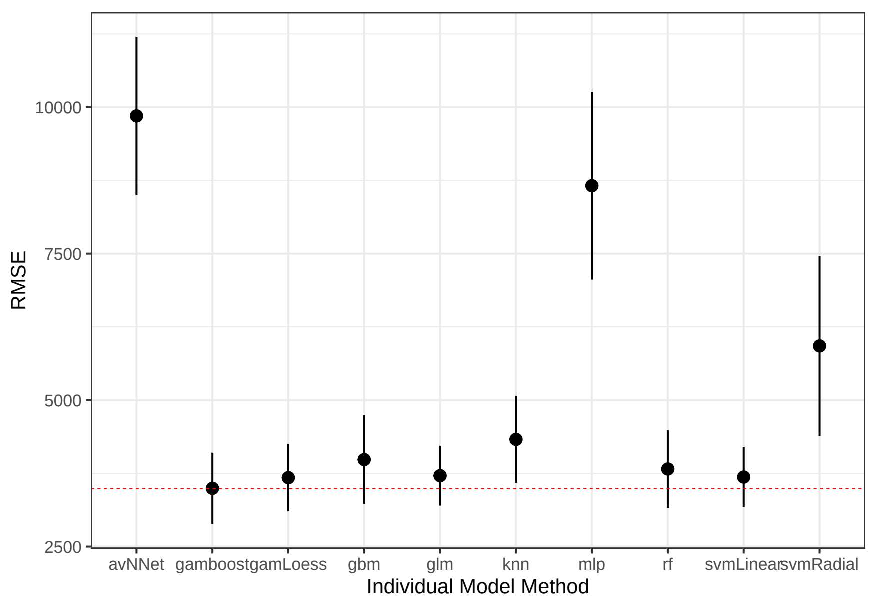
```
## rf        0.8627965 0.7035250
## glm       0.7990421 0.6030814
## svmLinear 0.7064464 0.5549658
## gamboost  0.6721481 0.5780205
## gamLoess  0.7879134 0.5327929
## avNNet    0.8027682 0.8652348
## mlp       0.4212875 0.4369388
## gbm       1.0000000 0.8074756
## svmRadial 0.8074756 1.0000000
```

The ensemble model is built on all 10 base models (see above) despite the high correlations.

```r
set.seed(100, sample.kind = "Rounding")
ensemble <- caretEnsemble(fits_list, metric = "RMSE",
    trControl = control)
summary(ensemble)
```

```
## The following models were ensembled: knn, rf, glm, svmLinear, gamboost, gamLoess, avNNet, mlp, gbm, s
## They were weighted:
## 146.6647 -0.0723 0.11 -1.3255 1.2884 0.7775 -0.0497 NA -0.0082 0.3479 -0.1035
## The resulting RMSE is: 3491.6097
## The fit for each individual model on the RMSE is:
##     method     RMSE    RMSESD
##        knn 4329.347  740.9806
##         rf 3824.001  664.1886
##        glm 3710.066  510.8684
##  svmLinear 3686.873  511.7914
##   gamboost 3494.060  608.8010
##   gamLoess 3676.293  572.0054
##     avNNet 9850.614 1349.9903
##        mlp 8659.152 1601.8838
##        gbm 3984.500  756.7827
##  svmRadial 5924.685 1536.9543
```

```r
plot(ensemble)
```



The RMSE value (red line) of the ensemble model on train data is 3492 which is better than the RMSE value of the individual models.

```
y_hat_ensemble <- predict(ensemble, newdata = test_x_imp)
RMSE_ensemble<- RMSE(y_hat_ensemble, test_y)
RMSE_ensemble
```

```
## [1] 4150.034
```

The RMSE value of the ensemble model is evaluated on test set and is equal to 4150.
Surprisingly, the RMSE value of the ensemble model on **test set** is worse than the RMSE value of *rf* on test set (see section above):

$$\frac{RMSE(rf)}{RMSE(ensemble)} = 0.61$$

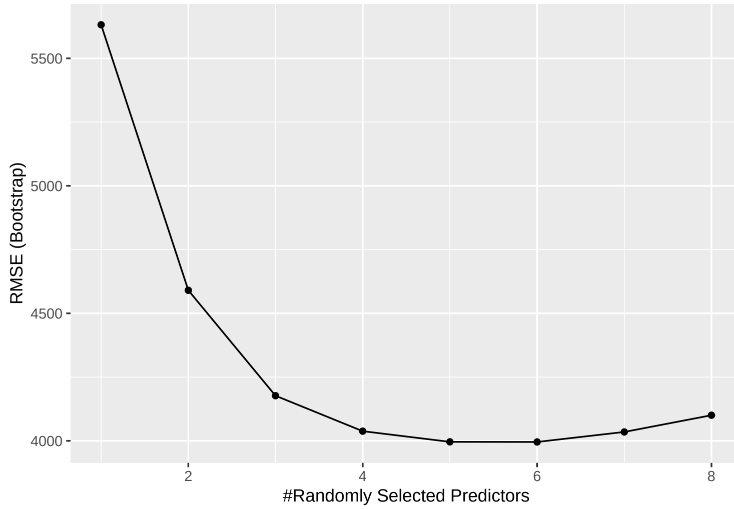
As stated above the worse performance of the ensemble model on test set probably is due to the 10 **highly correlated** base models, which could have a negative impact on model performance.

Final model:

The ensemble and *rf* (due to best performance on test set) are chosen as final models. Before their performances are evaluated on **validation set**, they are trained on whole train set:

```
set.seed(100, sample.kind = "Rounding")
control <-  trainControl(method="boot", number=25,  savePredictions="final",  allowParallel = TRUE)
fits_list_final <- caretList(train_x_imp, train_y,  trControl=control,  methodList= models,
                       tuneList=list(
                         knn=caretModelSpec(method="knn", tuneGrid=data.frame(k=k)),
                         rf=caretModelSpec(method="rf", tuneGrid=data.frame(mtry=mtry))))
ensemble_final<- caretEnsemble(fits_list_final, metric = "RMSE", trControl = control)
y_hat_ensemble_final <- predict(ensemble_final, newdata = validation_x_imp)
RMSE_ensemble_final<- RMSE(validation_y, y_hat_ensemble_final)
RMSE_ensemble_final
```

```
set.seed(100, sample.kind = "Rounding")
mtry<-seq(1,8,1)
fit_rf_final<-caret::train(train_x_imp, train_y, method = "rf", tuneGrid=data.frame(mtry=mtry))
ggplot(fit_rf_final)
```

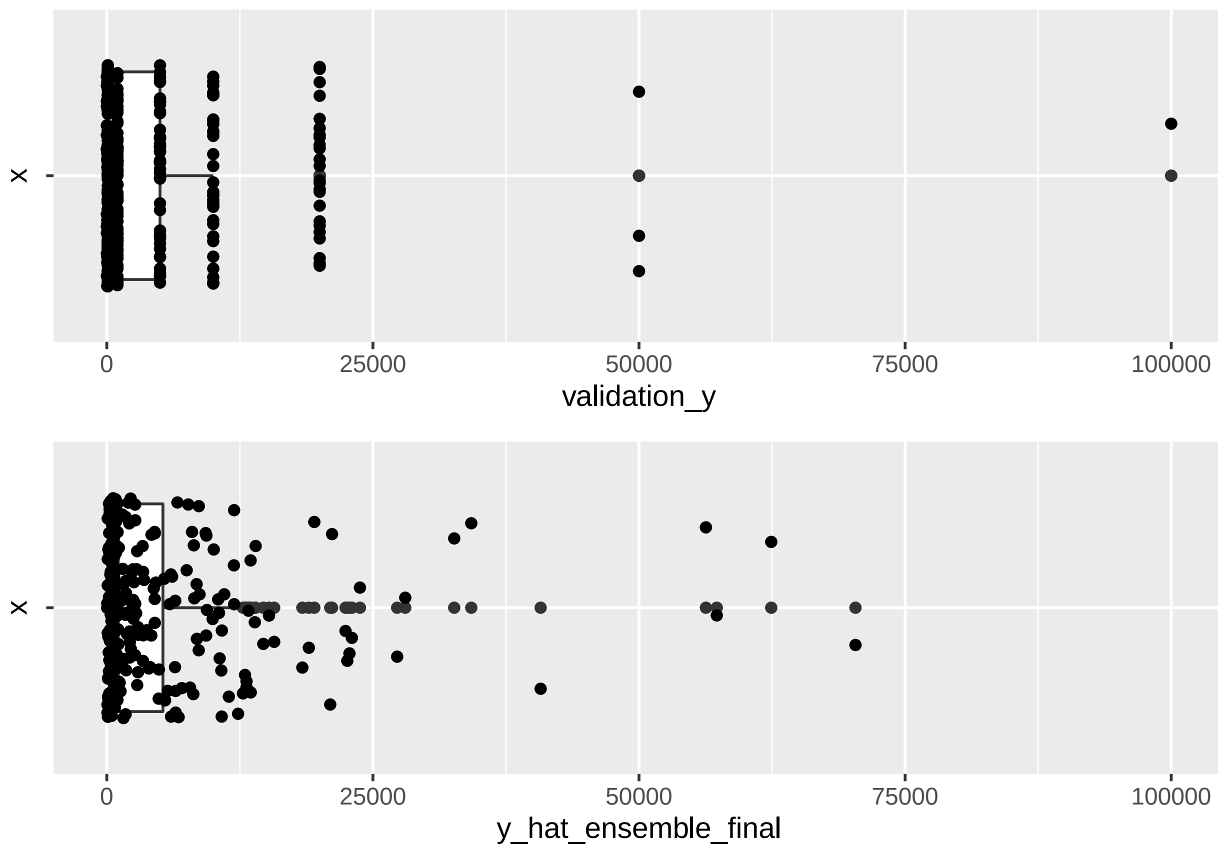



```
fit_rf_final$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##           Mean of squared residuals: 14703183
##                     % Var explained: 82.03
```

```
y_hat_rf_final <- predict(fit_rf_final, validation_x_imp)
RMSE_rf_final<-RMSE(validation_y, y_hat_rf_final)
RMSE_rf_final
```

```
## [1] 4815.677
```



The ensemble model has a RMSE of 4661 and slightly better than the RMSE of *random forest* 4816. Overall, the RMSE of ensemble model is slightly higher than the mean of *units_sold* (=4267) regarding train data. In addition, the median of *units_sold* regarding train data is equal to 1,000 units, thus 50% of units_sold is under this value. In conclusion, the ensemble model prediction is, strictly spoken, not precise enough. As the plots above show, ensemble model underestimate the two highest values equal to 100,000 units and overestimate most of units larger than 1,000 units.

4) Conclusion

The final RMSE value on validation set by ensemble model is 4661 which improves the RMSE value of the baseline model about 56% with an acceptable computation time.

The models developed in this paper must be adjusted for new merchants not having sold any products on the e-commerce platform Wish yet. This applies to new products having no product ratings on the platform, either.

Overall, this data set has (very) less correlated features with the model output. This could probably result in poor ML performance in terms of RMSE value. Further improvement of prediction could be achieved through ensemble models by using of many other *caret package* models which are uncorrelated. Collection of further sound data correlating with *units_sold* and comprehensive utilization of *predictor engineering* which creates features with “more signal and less noise” would help, too.