Prediction of Sales Volumes

G. L.

15/10/2018

1. The Dataset: Products and Attributes

A company selling electronic items is interested in estimating the market potential of a series of new products that it plans to launch to the market. The sales potential of these new products can be predicted using information of similar products already in the market. This information is available as a dataset containing various types of products and their attributes. Specifically, the table contains:

- 245 electronic products of various types (computers, displays, consoles, etc.).
- 22 variables describing various product attributes.

```
wd <- file.path('~',
                'GitRepos',
                'r-ds-projects',
                'sales_predictions')
setwd(dir= wd)
```

```
library(readr)
library(stringr)
library(ggplot2)
library(ggpubr)
library(dplyr)
library(corrplot)
library(caret)
library(gbm)
library(doMC)

# parallelization
registerDoMC(cores = 4)
```

```
# read dataset of existing products
input_file <- file.path('.',
                        'existing_products.csv')
if( !file.exists(input_file) ) {
  print('File:')
  print(input_file)
  print('not found!. Current dir:')
  getwd()
}
prod_exist <- read.csv(input_file, dec= ',', sep= ';')
```

A look at the structure of the dataset.

```
str(prod_exist)
```

```
## 'data.frame':    245 obs. of  22 variables:
##  $ X                            : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Product_type                 : Factor w/ 12 levels "Accessories",..: 7 7 7 5 5 1 1 1 1 1 ...
##  $ Product_ID                   : int  101 102 103 104 105 106 107 108 109 110 ...
##  $ Prices                       : num  949 2250 399 410 1080 ...
##  $ X5Stars                        : int  3 2 3 49 58 83 11 33 16 10 ...
##  $ X4Stars                        : int  3 1 0 19 31 30 3 19 9 1 ...
##  $ X3Stars                        : int  2 0 0 8 11 10 0 12 2 1 ...
##  $ X2Stars                        : int  0 0 0 3 7 9 0 5 0 0 ...
##  $ X1Stars                        : int  0 0 0 9 36 40 1 9 2 0 ...
##  $ Positive_service_review        : int  2 1 1 7 7 12 3 5 2 2 ...
##  $ Negative_service_review        : int  0 0 0 8 20 5 0 3 1 0 ...
##  $ Would_consumer_recomend__product: num  0.9 0.9 0.9 0.8 0.7 0.3 0.9 0.7 0.8 0.9 ...
##  $ Best_seller_rank               : num  1967 4806 12076 109 268 ...
##  $ Weigth                         : num  25.8 50 17.4 5.7 7 1.6 7.3 12 1.8 0.75 ...
##  $ Depth                          : Factor w/ 138 levels "0","0.04","0.07",..: 97 108 47 66 58 115
##  $ Width                          : num  6.62 31.75 8.3 9.9 0.3 ...
##  $ Heigth                         : num  16.9 19 10.2 1.3 8.9 ...
##  $ Profit_margin                  : num  0.15 0.25 0.08 0.08 0.09 0.05 0.05 0.05 0.05 0.05 ...
##  $ Volume                         : int  12 8 12 196 232 332 44 132 64 40 ...
##  $ Competitors                    : int  3 3 5 1 3 2 1 2 3 5 ...
##  $ Professional                   : int  0 0 0 0 1 0 1 1 1 1 ...
##  $ Age                            : int  2 3 3 2 2 3 3 2 2 3 ...
```

Our target variable is the sales volume `Volume`. We notice that two predictors are just identifiers and can be removed: `X` and `Product_ID`. The description of the remaining 20 variables is the following:

- `Product_type`: type of electronic product (categorical).
- `Prices`: price of product (numeric).
- `X5Stars` - `X1Stars`: number of n-star product reviews (integer).
- `PositiveServiceReview`, `NegativeServiceReview`: number of positive and negative reviews of product service (integer).
- `Would_consumer_recommend_product`: score (from 0 to 1) assigned by user to the product (numeric).
- `Best_Seller_Rank`: position of product in sales ranking (integer).
- `Weight`: product weight (lbs., numeric).
- `Depth`: product depth (in., numeric).
- `Height`: product height (in., numeric).
- `Profit_margin`: profit (fraction of price, numeric).
- `Volume`: sales volume (units, integer).
- `Competitors`: number of competitor products in the market (integer).
- `Professional`: professional or business products (integer 0 or 1).
- `Age`: time of product since launch in the market (integer).

We start by simplifying the feature names:

```r
# remove useless columns (index, product id),
# rename features
prod_exist %>% subset(., select= -c(X, Product_ID)) -> prod_exist
new_colnames <- c('Type', 'Price', 'x5s', 'x4s', 'x3s',
                  'x2s', 'x1s', 'PosServ', 'NegServ',
                  'Recommend', 'BestSeller', 'Weight',
                  'Depth', 'Width', 'Height', 'Profit',
                  'Vol', 'Comp', 'Prfsn', 'Age')
names(prod_exist) <- new_colnames
```

Some predictor data types need to be changed to reflect their meaning: `Professional` should be a factor with two levels ("No", "Yes"), whereas `Depth` is a numeric variable.

```r
# update predictors data types
prod_exist$Prfsn %>%
  factor(levels = c(0, 1),
         labels = c("No", "Yes")) -> prod_exist$Prfsn
```

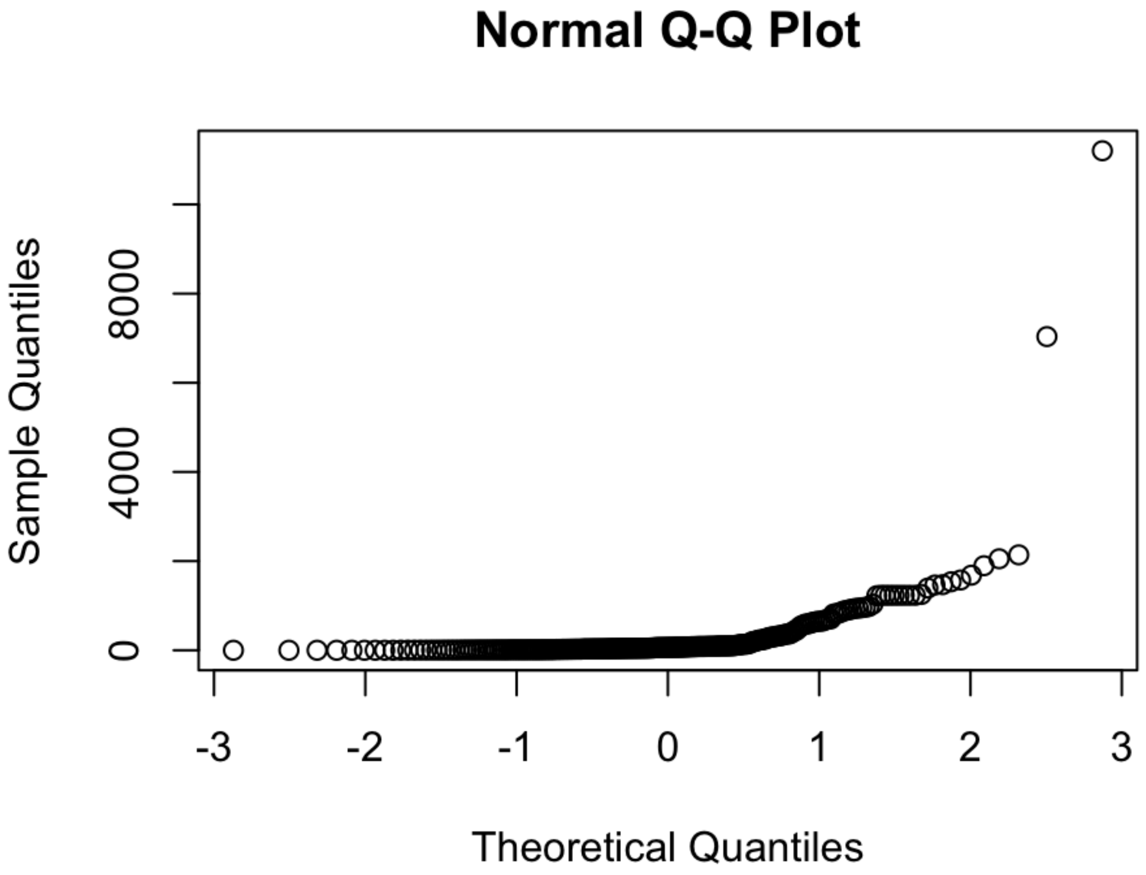
```
prod_exist$Depth %>%
  as.character(.) %>%
  as.numeric(.) -> prod_exist$Depth

head(prod_exist)
```

```
##          Type   Price x5s x4s x3s x2s x1s PosServ NegServ Recommend
## 1          PC  949.00   3   3   2   0   0       2       0       0.9
## 2          PC 2249.99   2   1   0   0   0       1       0       0.9
## 3          PC  399.00   3   0   0   0   0       1       0       0.9
## 4      Laptop  409.99  49  19   8   3   9       7       8       0.8
## 5      Laptop 1079.99  58  31  11   7  36       7      20       0.7
## 6 Accessories  114.22  83  30  10   9  40      12       5       0.3
##   BestSeller Weight Depth Width Height Profit Vol Comp Prfsn Age
## 1       1967   25.8 23.94  6.62  16.89   0.15  12    3    No   2
## 2       4806   50.0 35.00 31.75  19.00   0.25   8    3    No   3
## 3      12076   17.4 10.50  8.30  10.20   0.08  12    5    No   3
## 4        109    5.7 15.00  9.90   1.30   0.08 196    1    No   2
## 5        268    7.0 12.90  0.30   8.90   0.09 232    3   Yes   2
## 6         64    1.6  5.80  4.00   1.00   0.05 332    2    No   3
```

2. Cleaning and Exploration of the Dataset

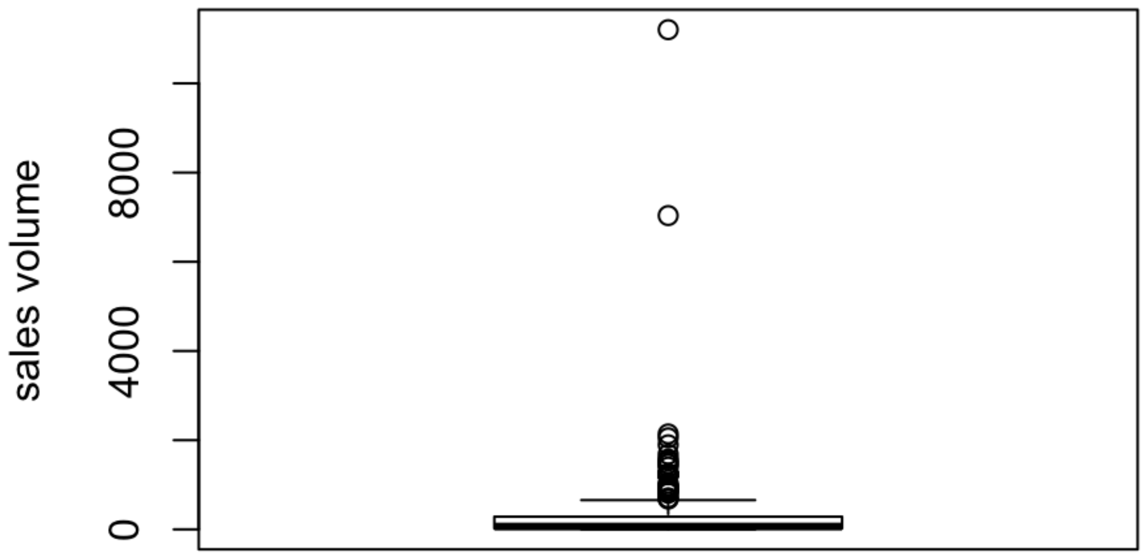
We may first have a look at the distribution of the dependent variable to check for the presence of outliers that may have an outsized effect on the predictive models.

```
qqnorm(prod_exist$Vol)
```



A normalized quantile-quantile plot shows that the `Volume` target variable differs significantly from a normally distributed random variable. In particular, there are at least two points which stand out due to their huge volumes as is also apparent by looking at the boxplot below.

```r
boxplot(x = prod_exist$Vol, ylab = 'sales volume')
```



These points are removed by taking the observations having sales volume < 5000 units.

```
# remove outliers
prod_exist <- filter(prod_exist, Vol < 5000)
```

Secondly, observation with NA values may also be present in the dataset.

```
# find and store NAs on separate data frame
nas <- prod_exist[!complete.cases(prod_exist), ]
dim(nas)
```

```
## [1] 16 20
```

```
summary(nas)
```

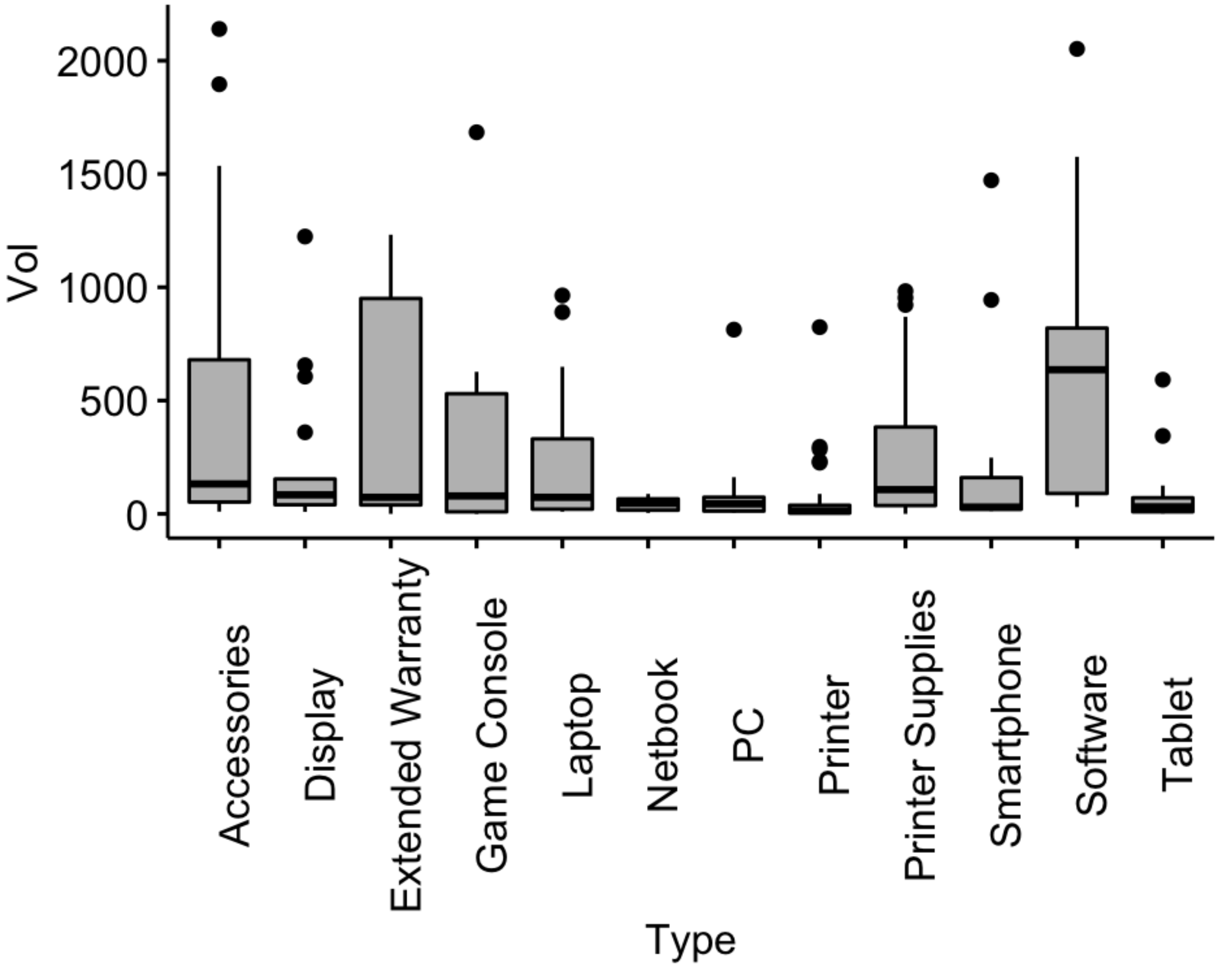
```
##                 Type        Price             x5s              x4s
##  Accessories     :6   Min.   :  6.55   Min.   :  0.00   Min.   :  0.00
##  Printer         :5   1st Qu.: 39.11   1st Qu.:  5.75   1st Qu.:  1.75
##  Printer Supplies:2   Median :132.72   Median : 13.00   Median :  5.00
##  Laptop          :1   Mean   :186.40   Mean   : 38.56   Mean   : 14.75
##  PC              :1   3rd Qu.:221.94   3rd Qu.: 21.00   3rd Qu.: 12.25
##  Software        :1   Max.   :609.99   Max.   :349.00   Max.   :118.00
##  (Other)         :0
##       x3s              x2s              x1s            PosServ
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.: 0.750   1st Qu.: 1.000
##  Median : 2.000   Median : 0.000   Median : 2.000   Median : 2.500
##  Mean   : 4.625   Mean   : 2.062   Mean   : 5.875   Mean   : 9.875
##  3rd Qu.: 4.250   3rd Qu.: 2.500   3rd Qu.:10.500   3rd Qu.: 5.000
##  Max.   :27.000   Max.   :11.000   Max.   :21.000   Max.   :64.000
##
##     NegServ         Recommend       BestSeller      Weight
##  Min.   : 0.000   Min.   :0.500   Min.   :559   Min.   : 0.400
##  1st Qu.: 0.000   1st Qu.:0.675   1st Qu.:559   1st Qu.: 1.000
##  Median : 1.000   Median :0.800   Median :559   Median : 3.805
##  Mean   : 3.625   Mean   :0.750   Mean   :559   Mean   :13.632
##  3rd Qu.: 3.250   3rd Qu.:0.900   3rd Qu.:559   3rd Qu.:30.400
##  Max.   :24.000   Max.   :1.000   Max.   :559   Max.   :39.000
##                                   NA's   :15
##      Depth           Width           Height          Profit
##  Min.   : 1.50   Min.   : 1.60   Min.   : 0.50   Min.   :0.0500
##  1st Qu.: 6.20   1st Qu.: 6.25   1st Qu.: 4.70   1st Qu.:0.0500
##  Median :10.40   Median : 9.40   Median :11.19   Median :0.1300
##  Mean   :11.26   Mean   :10.12   Mean   :10.48   Mean   :0.1412
##  3rd Qu.:16.93   3rd Qu.:14.45   3rd Qu.:14.70   3rd Qu.:0.1850
##  Max.   :22.10   Max.   :20.90   Max.   :20.71   Max.   :0.3000
##                  NA's   :1
##       Vol              Comp       Prfsn         Age
##  Min.   :   0.0   Min.   :0.000   No :12   Min.   :1.000
##  1st Qu.:  30.0   1st Qu.:1.000   Yes: 4   1st Qu.:2.000
##  Median :  52.0   Median :3.000            Median :3.000
##  Mean   : 156.6   Mean   :2.438            Mean   :2.625
##  3rd Qu.:  84.0   3rd Qu.:3.250            3rd Qu.:3.000
##  Max.   :1396.0   Max.   :5.000            Max.   :4.000
##
```

There are 16 NAs, 15 of which in the `BestSeller` column, and 1 in the `Width` column. They're a relatively small number so we'll remove them from the dataset.

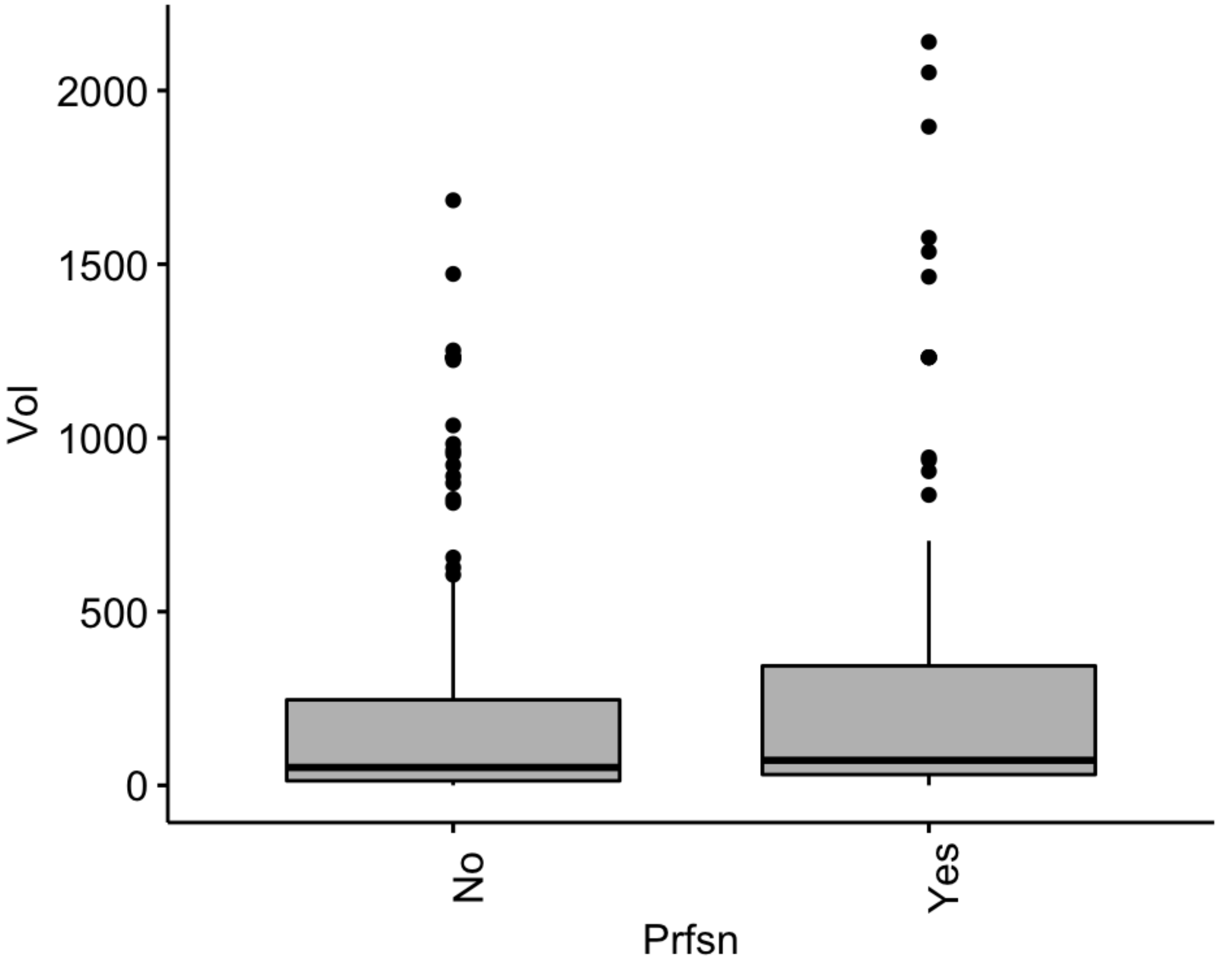
```
# remove NAs
prod_exist %>%
  .[complete.cases(.), ] -> prod_exist
```

Finally, to gain insight into how the observations are distributed in our dataset, let's examine the distribution of the sales volume variable against some predictors.

```
ggboxplot(data= prod_exist,
          x = 'Type',
          y = 'Vol',
          fill= 'grey') +
  theme(axis.text.x = element_text(angle = 90))
```

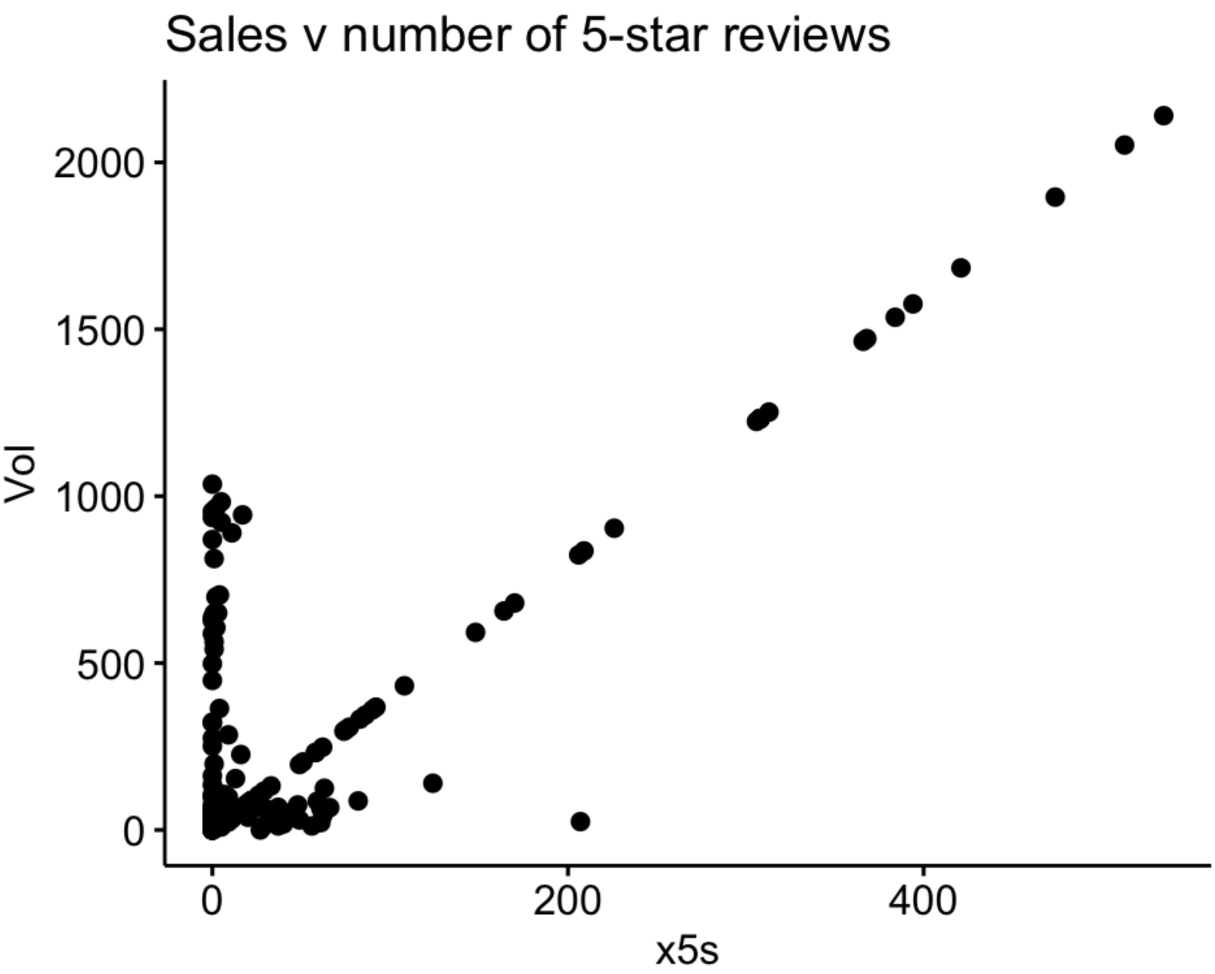


```
ggboxplot(data= prod_exist,
          x = 'Prfsn',
          y = 'Vol',
          fill= 'grey') +
  theme(axis.text.x = element_text(angle = 90))
```

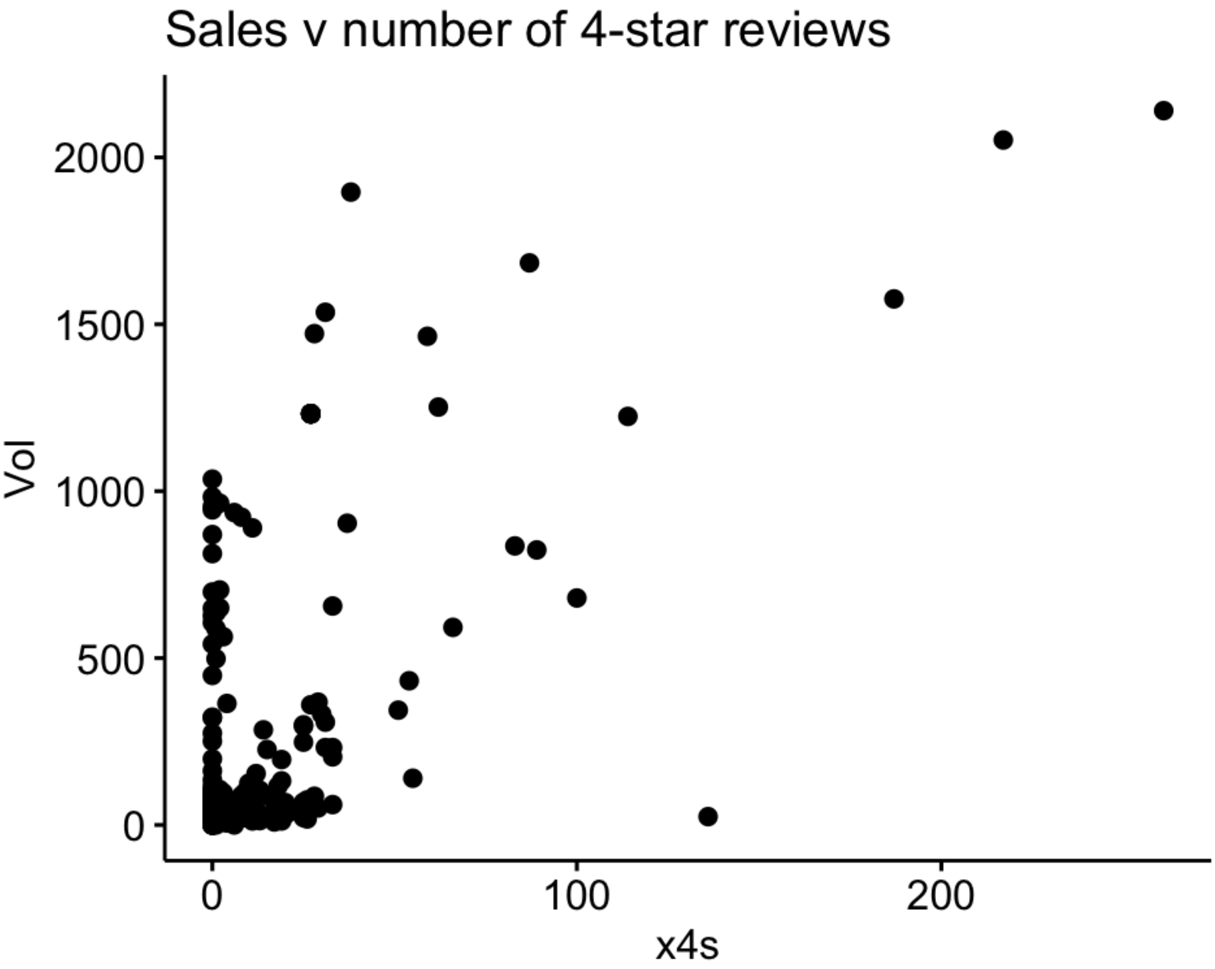


We may expect that the sales volume be influenced by the amount of positive product reviews. The correlation between the variables volume and number of n-star reviews can be displayed using scatterplots.

```
ggscatter(data= prod_exist,
          x = 'x5s',
          y = 'Vol',
          fill= 'grey') +
  labs(title = 'Sales v number of 5-star reviews')
```



```
ggscatter(data= prod_exist,
          x = 'x4s',
          y = 'Vol',
          fill= 'grey') +
  labs(title = 'Sales v number of 4-star reviews')
```



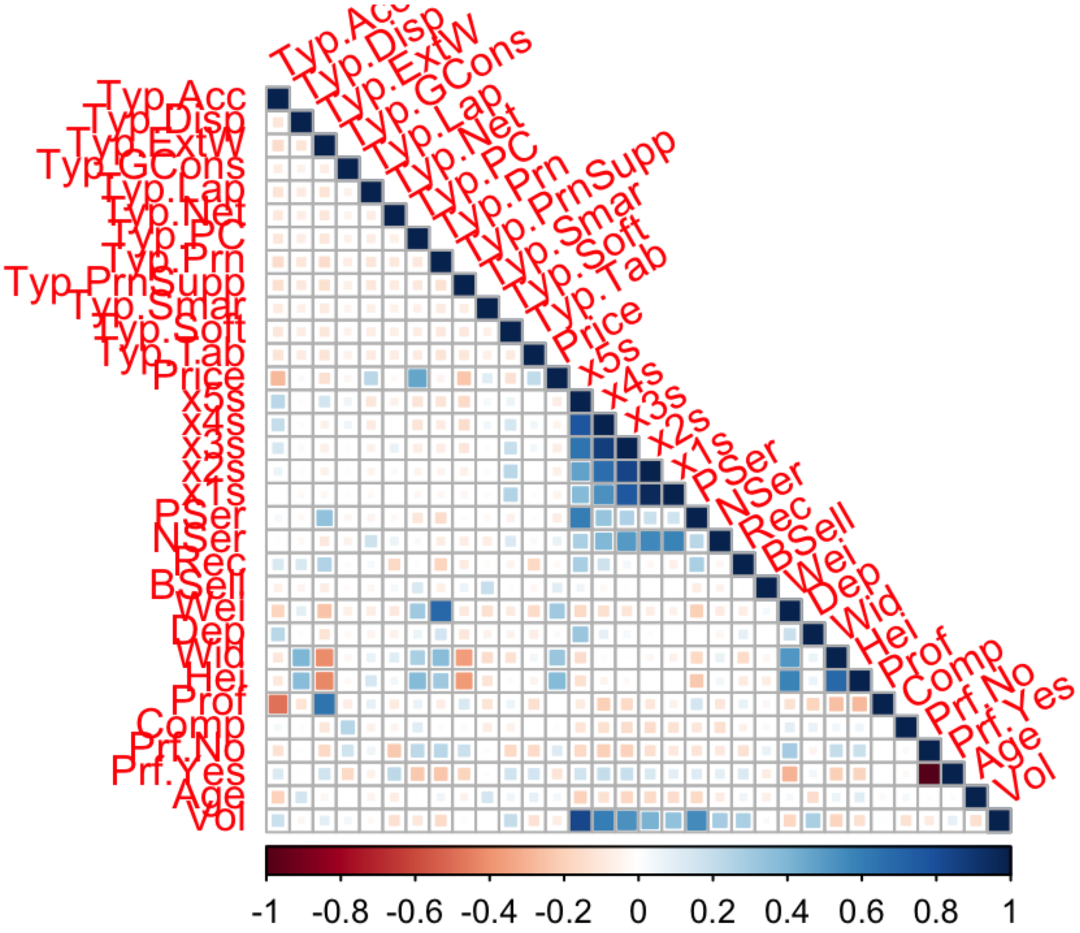
The correlation is really strong for the 5-star reviews, less so for the 4-star reviews. Correlations between numerical features can be examined in more detail by calculating Pearson's correlation coefficient between feature pairs.

2.1 Quantifying and Visualizing Correlations between Variables

Determining correlations between variables is useful if we need to get rid of highly correlated predictors in order to fit the dataset with a linear model, for instance.

```
# set volume as last column in dataset
prod_exist <- prod_exist[ c(1:16, 18, 19, 20, 17) ]
```

```
# generate dummy variables for factors
dmy <- dummyVars('~ .', data = prod_exist)
prod_dmy <- prod_exist %>%
            predict(dmy, .) %>%
            data.frame()

# calculate correlations
corrData <- cor(prod_dmy)
new_colnames <-
  c('Typ.Acc', 'Typ.Disp', 'Typ.ExtW', 'Typ.GCons',
    'Typ.Lap', 'Typ.Net', 'Typ.PC', 'Typ.Prn',
    'Typ.PrnSupp', 'Typ.Smar', 'Typ.Soft',
    'Typ.Tab', 'Price', 'x5s', 'x4s',
    'x3s', 'x2s', 'x1s', 'PSer', 'NSer', 'Rec',
    'BSell', 'Wei', 'Dep', 'Wid', 'Hei', 'Prof',
    'Comp', 'Prf.No', 'Prf.Yes', 'Age', 'Vol')

# shorten column and row names for plotting
colnames(corrData) <- new_colnames
rownames(corrData) <- new_colnames

# display correlogram
corrplot(corrData,
         method = 'square',
         type = 'lower',
         na.label= 'o',
         tl.srt= 30)
```



Highly correlated (collinear) predictors include:

- `x5s` and `Vol`,

- all pairs of n-star reviews predictors (x5s to x1s).

The correlation coefficients for a select pair of variables can be displayed via the following:

```r
corrData["Vol", "x5s"]
```

```
## [1] 0.8342784
```

```r
corrData["x5s", "x4s"]
```

```
## [1] 0.7707467
```

```r
corrData["x4s", "x3s"]
```

```
## [1] 0.8740702
```

```r
corrData["x3s", "x2s"]
```

```
## [1] 0.8495147
```

Some of these predictors will be removed prior to fitting a linear model to the data.

The next steps to build a predictive model for the sales volume involve:

- the selection of a machine learning algorithm to apply to the dataset
- a validation step to optimize the parameters of the algorithm so as the risk of overfitting is minimized, and
- a test step to see how well the model fares when predicting unseen data.

3. Model: Gradient Boosted Machines

We will first try to fit a tree-based model, namely a gradient-boosted tree model from the gbm package.

```r
# split dataset into train and test set
set.seed(1987)
train_indices <-
  createDataPartition(prod_exist$Vol,
                      p= 0.75,
                      list= F,
                      times= 1)
data_train <- prod_exist[ train_indices, ]
data_test <- prod_exist[ -train_indices, ]
```

10-fold cross validation helps avoid overfitting, i.e. reproducing too closely the patterns in the dataset, which often decreases the predictive performances of the model on new, unseen data.

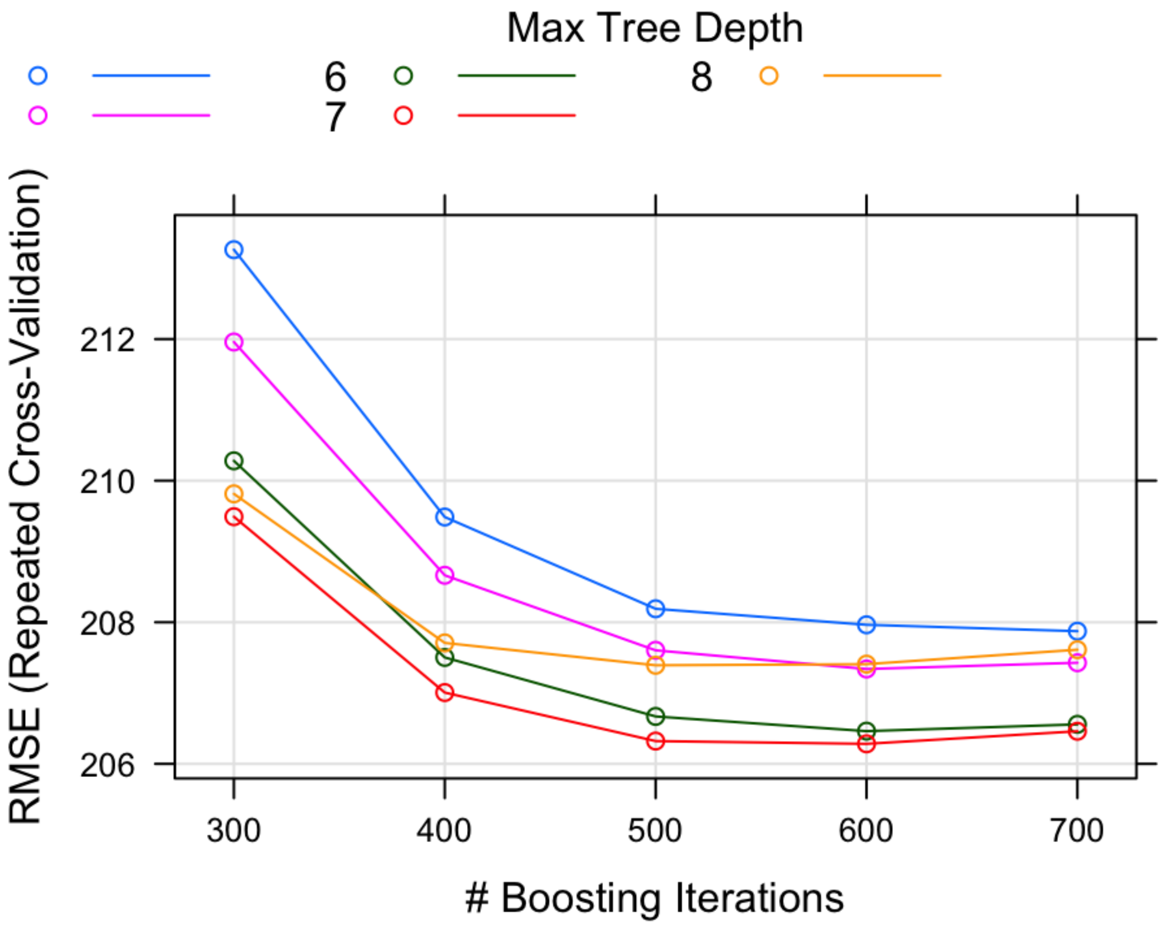
```r
# model validation: 10-fold cross-validation
ctrl <-
  trainControl(method= 'repeatedcv',
               number= 10,
               repeats= 30,
               summaryFunction= defaultSummary)
```

Fit a GBM model to the dataset.

```r
# gbm model fit
# set search grid for parameters
gbmGrid <-
  expand.grid(n.trees= c(200, 300, 400, 500),
              interaction.depth= c(5, 6, 7, 8),
              shrinkage= 0.01,
              n.minobsinnode= 2)

# fit without scaling
gbm <-
  train(Vol ~ .,
        data= data_train,
        method= 'gbm',
        trControl= ctrl,
        tuneGrid= gbmGrid,
        preProc= NULL,
        metric= 'RMSE',
        verbose= F)
```

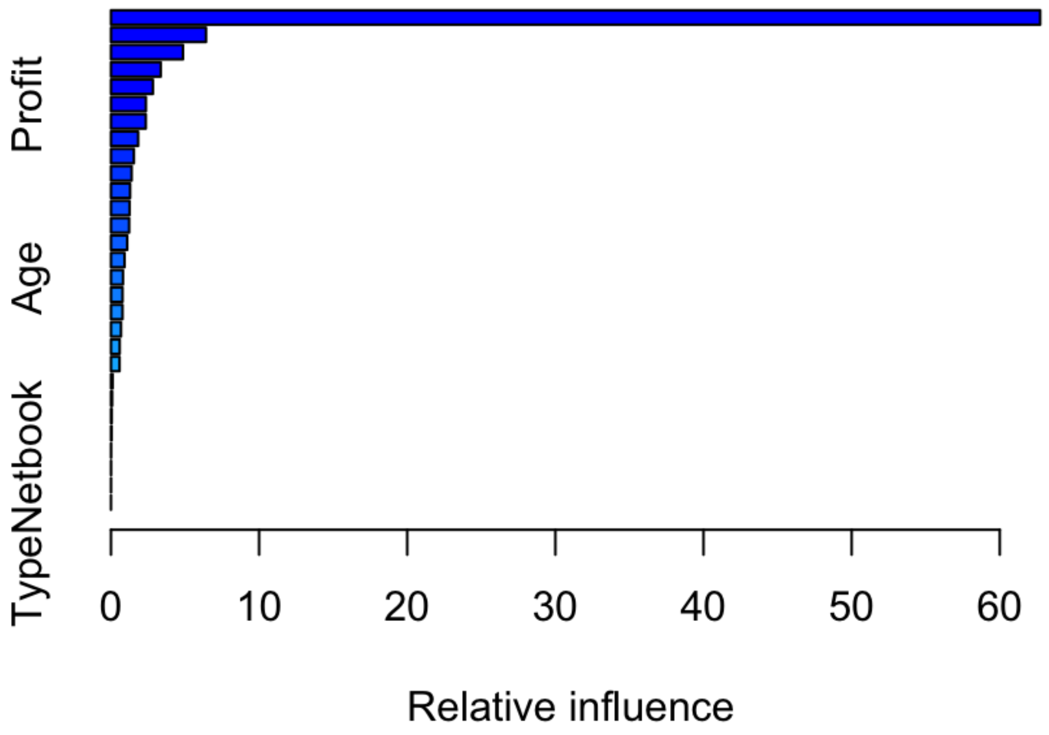
The optimal model parameters are `n.trees` = 600 and `interaction.depth` = 8; the RMSE curve as a function of the number of trees and tree depth can be displayed with the `plot` command:

```
plot(gbm)
```



Information about the model performance on each resample, including variable importance, is displayed with the `summary` function.

```
summary(gbm)
```



```
##                                        var      rel.inf
## x5s                                    x5s 60.742914828
## Price                                Price  6.875675083
## PosServ                            PosServ  4.848300472
## Weight                              Weight  3.294094894
## BestSeller                      BestSeller  3.141454009
## Profit                              Profit  3.040617866
## TypeSoftware                  TypeSoftware  2.798701814
## Recommend                        Recommend  1.967087397
## TypePrinter Supplies   TypePrinter Supplies  1.521949311
## Comp                                  Comp  1.432628319
## TypeLaptop                      TypeLaptop  1.247307212
## x2s                                    x2s  1.122869429
## x4s                                    x4s  1.100079772
## NegServ                            NegServ  1.088472761
## Age                                    Age  0.926005784
## Depth                                Depth  0.861446139
## x3s                                    x3s  0.858103767
## TypeGame Console           TypeGame Console  0.774419334
## x1s                                           x1s  0.688410854
## Width                                       Width  0.659762421
## Height                                     Height  0.529121067
## PrfsnYes                                 PrfsnYes  0.283014355
## TypePC                                     TypePC  0.116028425
## TypeExtended Warranty TypeExtended Warranty  0.028795787
## TypeTablet                             TypeTablet  0.026783493
## TypeSmartphone                     TypeSmartphone  0.015628456
## TypePrinter                           TypePrinter  0.005165923
## TypeNetbook                           TypeNetbook  0.005161028
## TypeDisplay                           TypeDisplay  0.000000000
```

The optimal model has an R squared of ~0.71, and an RMSE ~200. Interestingly, many variables have little relevance to the quality of the fit as can be seen from the variable importance bar plot. We could refit the model after removing all variables with importance parameter less than 2, for instance, without significant losses of predictive performance.

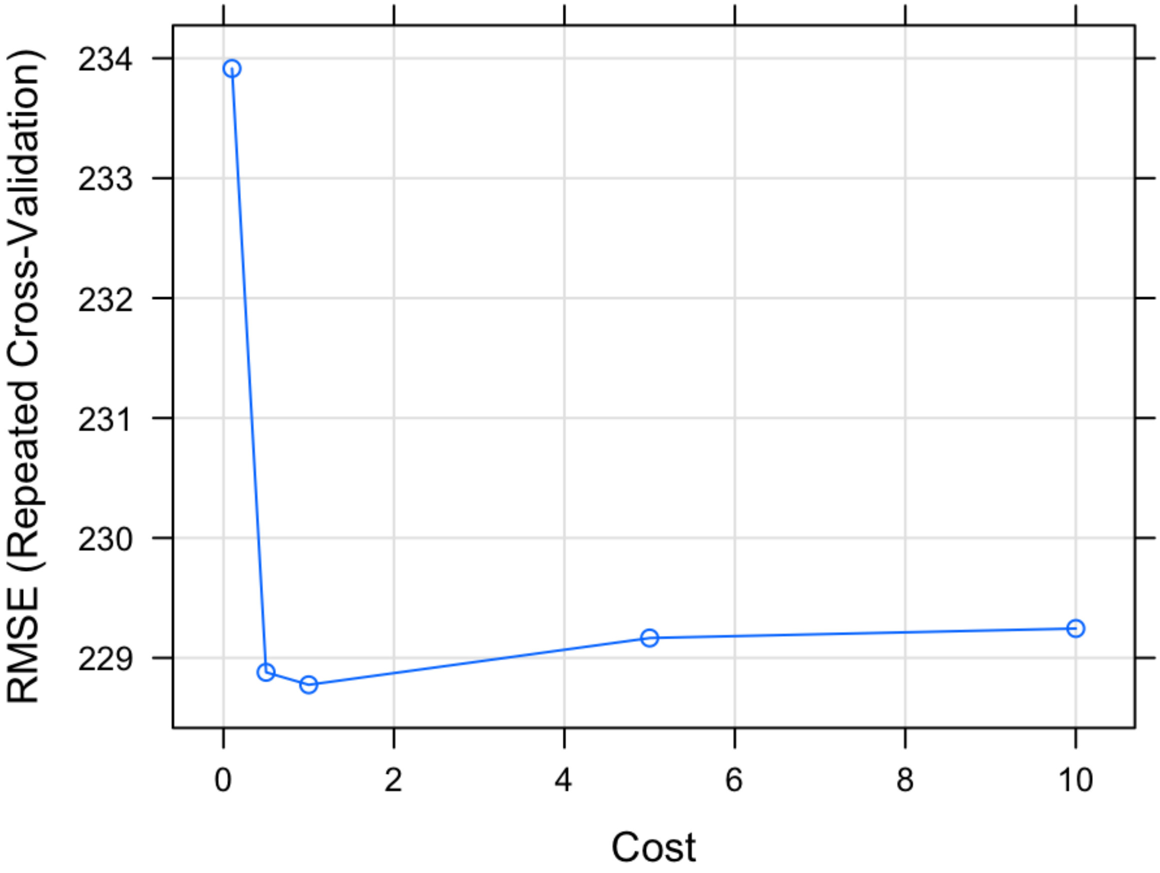
4. Model: Support Vector Machines with Linear Kernel

Next, we fit a predictive model that uses the SVM algorithm with linear kernel function.

```
# SVM fit
# set search grid for parameters
svmGrid <-
  expand.grid(C= c(0.1, 0.5, 1, 5, 10))

# fit without scaling
svm_lin <-
  train(Vol ~ .,
        data= data_train,
        method= 'svmLinear',
        trControl= ctrl,
        tuneGrid= svmGrid,
        preProc= NULL,
        metric= 'RMSE',
        verbose= F)
```

A look at the RMSE against cost parameter.

```
plot(svm_lin)
```



And model summary.

```
print(svm_lin)
```

```
## Support Vector Machines with Linear Kernel
##
## 172 samples
##  19 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 30 times)
## Summary of sample sizes: 153, 155, 155, 156, 155, 155, ...
## Resampling results across tuning parameters:
##
##   C     RMSE      Rsquared   MAE
##    0.1  233.1642  0.6633830  133.7456
##    0.5  227.9703  0.6671308  128.9273
##    1.0  227.3311  0.6654434  128.8330
##    5.0  227.3023  0.6634290  129.1928
##   10.0  227.0253  0.6635151  129.1173
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was C = 10.
```

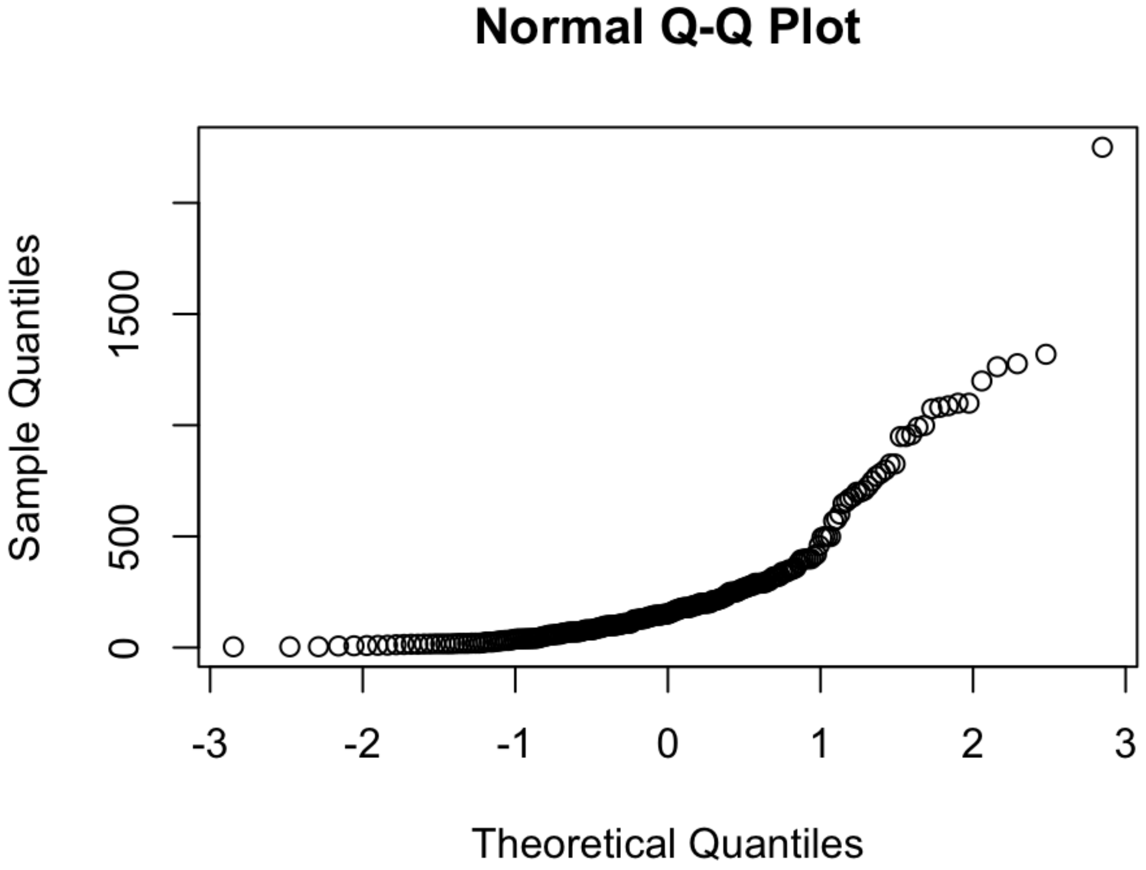
5. Model: Multiple Linear Regression

Highly correlated predictors have to be removed before fitting a linear model as a multiple linear regression model is constructed assuming that predictors are non-collinear. The model can estimate the variation in the response variable (volume) if any of the independent variables is changed, while all the other are kept constant.

However, other assumptions have to be met in order for a linear regression model to be appropriate for the particular dataset under investigation. These include: the independent variables must be normally distributed, the errors must be uncorrelated and normally distributed with the same variance.

We can see that the assumption of normality of the independent variables is violated by visualizing the normalized quantile-quantile plot of some predictors. For example:

```r
qqnorm(prod_exist$Price)
```



`Price` is clearly far from being normally distributed (the quantile points should line up to form a straight line, if the variable were normally distributed). As a consequence, a linear regression model is not appropriate for this data - it may produce unreliable estimates.

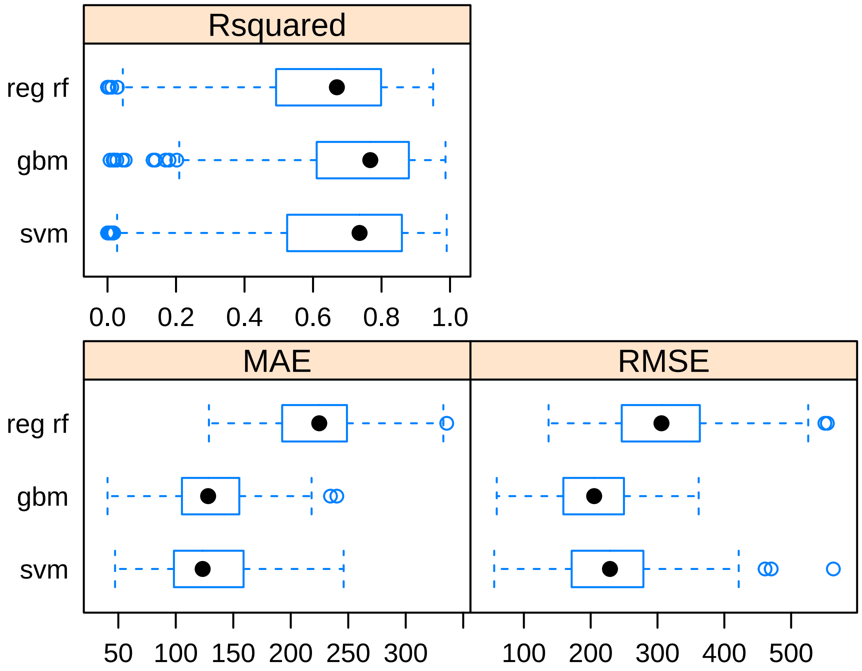
6. Model: Regularized Random Forest

We choose to fit a Regularized Random Forest model instead. The regularization term peforms feature selection by evaluating and comparing the importance (e.g. impurity decrease) of different features while growing the forest. It penalizes features that lead to a low impurity decrease relative to features already used for growing previous trees.

```
# regularized random forest fit
# set search grid for parameters
rrfGrid <-
  expand.grid(mtry = 0.75,
              coefReg = 0.8,
              coefImp = 0.25)

# fit without scaling
rrf <-
  train(Vol ~ .,
        data= data_train,
        method= 'RRF',
        trControl= ctrl,
        tuneGrid= rrfGrid,
        preProc= NULL,
        metric= 'RMSE',
        verbose= F)
```

7. Model Selection

```
# compare models
# significance test on resamples
sig_test <-
  resamples( list('gbm'= gbm,
                  'svm'= svm_lin,
                  'reg rf'= rrf) )
# plot perf. metrics
bwplot(sig_test,
       scales = list(x = list(relation = "free")))
```


Rsquared
reg rf
gbm
svm
0.0
0.2
0.4
0.6
0.8
1.0
MAE
reg rf
gbm
svm
50
100
150
200
250
300
RMSE
100
200
300
400
500