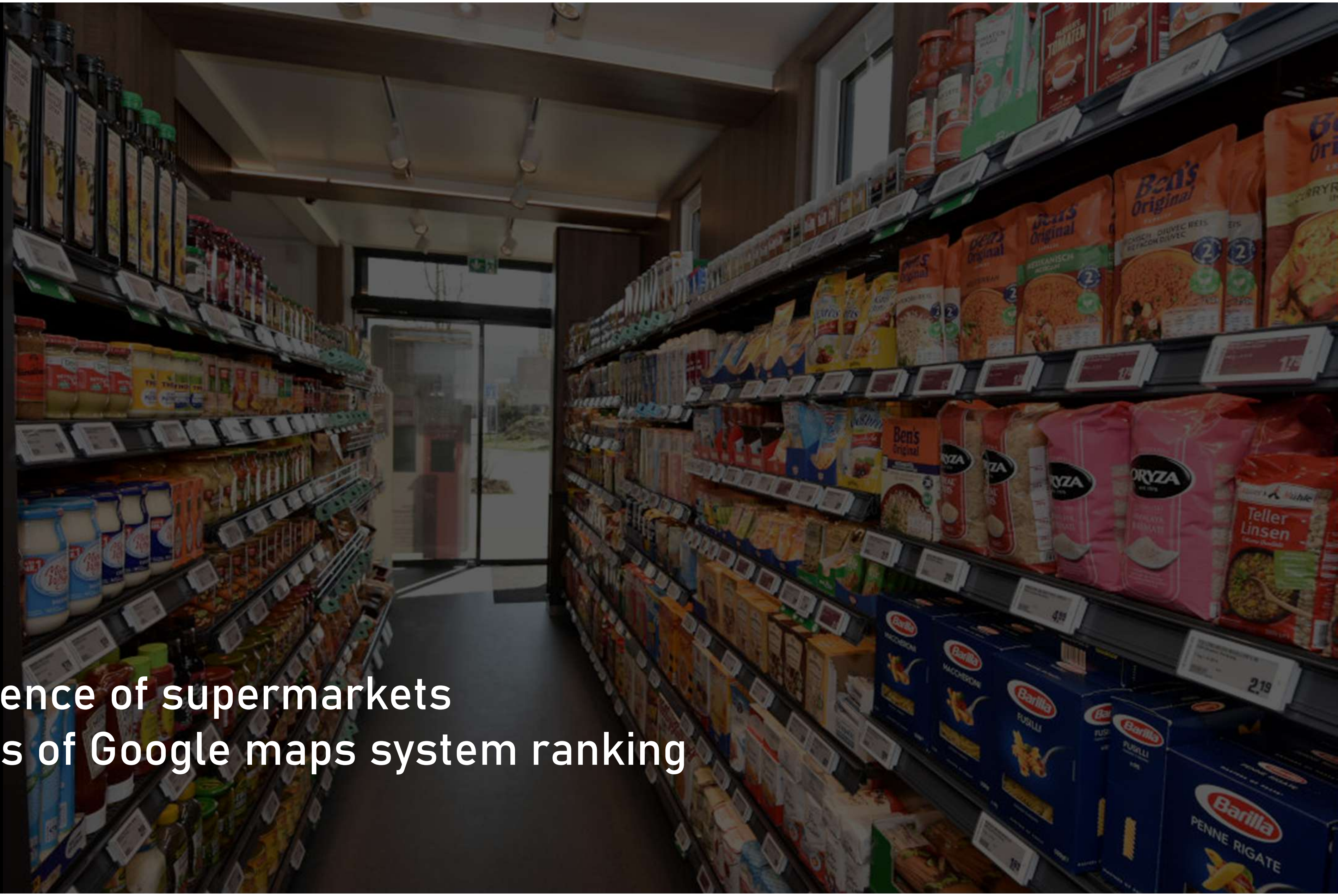
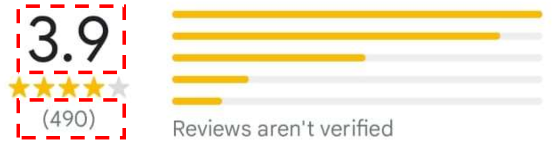


The preference of supermarkets
An analysis of Google maps system ranking

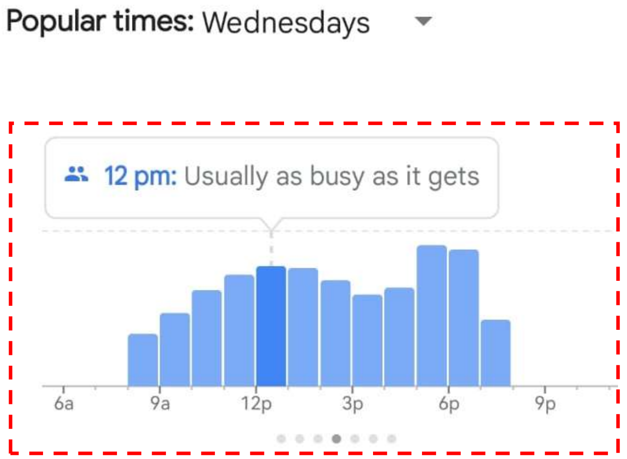
The preference of supermarkets

We gathered data from Google Maps of 496 supermarkets in Berlin (around 75% of the supermarkets listed in Google) including:

The ranking of each supermarket

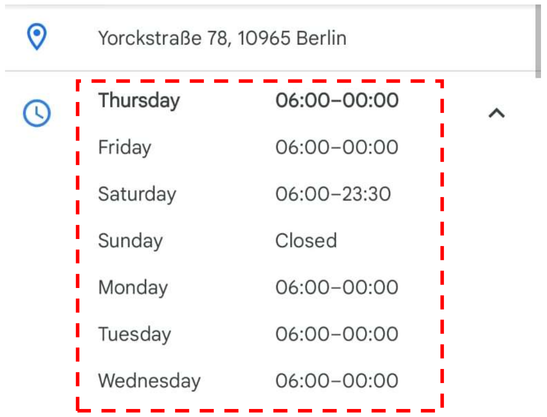


The Number of reviews of each supermarket
Including how many people rate 1 star, 2, 3, 4 and 5 stars



The popular times for each day of the week were
"Usually as busy as it gets" is the highest demand
And "Usually not Busy" is the lowest demand

The average time people spend in the supermarket



Yorckstraße 78, 10965 Berlin

Thursday	06:00–00:00
Friday	06:00–00:00
Saturday	06:00–23:30
Sunday	Closed
Monday	06:00–00:00
Tuesday	06:00–00:00
Wednesday	06:00–00:00

The opening and closing times of the supermarkets

Based on just Google Maps info, we want to predict the Google´s ranking of the supermarkets in Berlin giving by the people

Goal

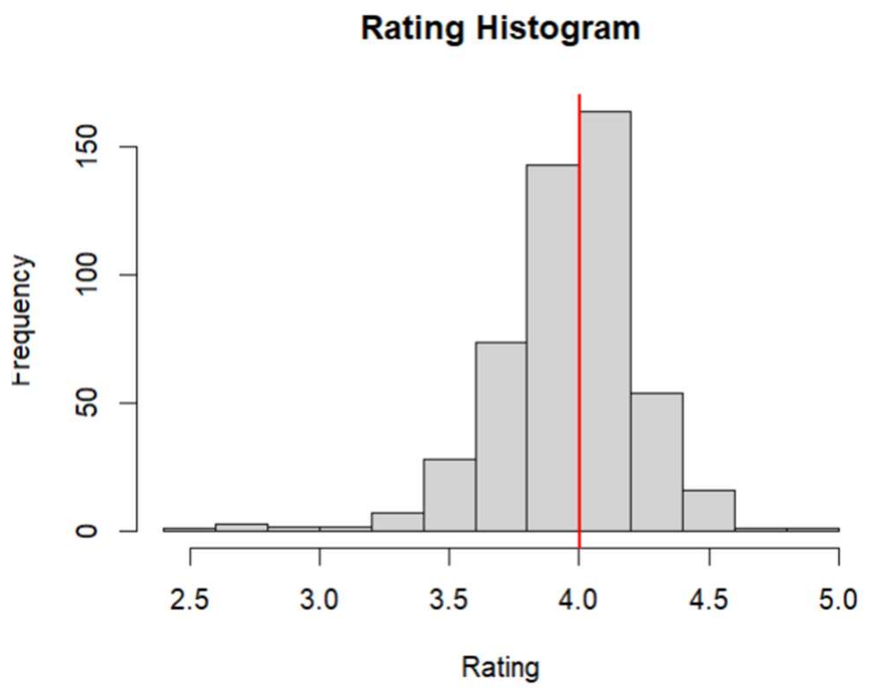
Based on just Google Maps info, we want to predict the Google´s ranking of the supermarkets in Berlin giving by the people.
We are going to use then:

Rating = α + β "AVG TIME"+ β "DISTCENT"+ β "NUM_COMMENTS"+β "NEG_COMMENTPRICE"+β "POS_REVIEWS"+ β "NEG_REVIEWS" + β "perc_BH"+ β "COMCENTER"+β "PARKLOT"+β "OPEN6AM"+ β "OPEN7AM"+ β "OPEN8AM" +β "OPENAFTER8"

RATING:	Rating goes from 1 to 5. Google take the average of all the ratings and presented as the rating for the supermarket
AVGTIME	Average time people spend in the supermarket
DISTCENT	Distance of the supermarket location to the center of Berlin
NUM_COMMENTS	Number of comments the supermarket has
NEG_COMMENTPRICE	Number of negative comments the supermarket has regarding the prices
POS_REVIEWS	Number of positive reviews in the supermarket
NEG_REVIEWS	Number of negative reviews in the supermarket
perc_BH	Number of hours the supermarket is fully busy in the week / Number of hours the supermarket is open in the week
COMCENTER	If supermarket is in a commercial center (mall) or not (1,0)
PARKLOT	If supermarket has parking lot or not (1,0)
OPEN6AM	Supermarkets that open at 6am
OPEN7AM	Supermarkets that open at 7am
OPEN8AM	Supermarkets that open at 8am
OPENAFTER8	Supermarkets that open after 8am

Note: We assumed that the negative reviews (reviews with 1 and 2 stars) is the variable that captures some of the data we don´t observed directly like the food quality or preferences of consumers

Checking the frequency of ratings

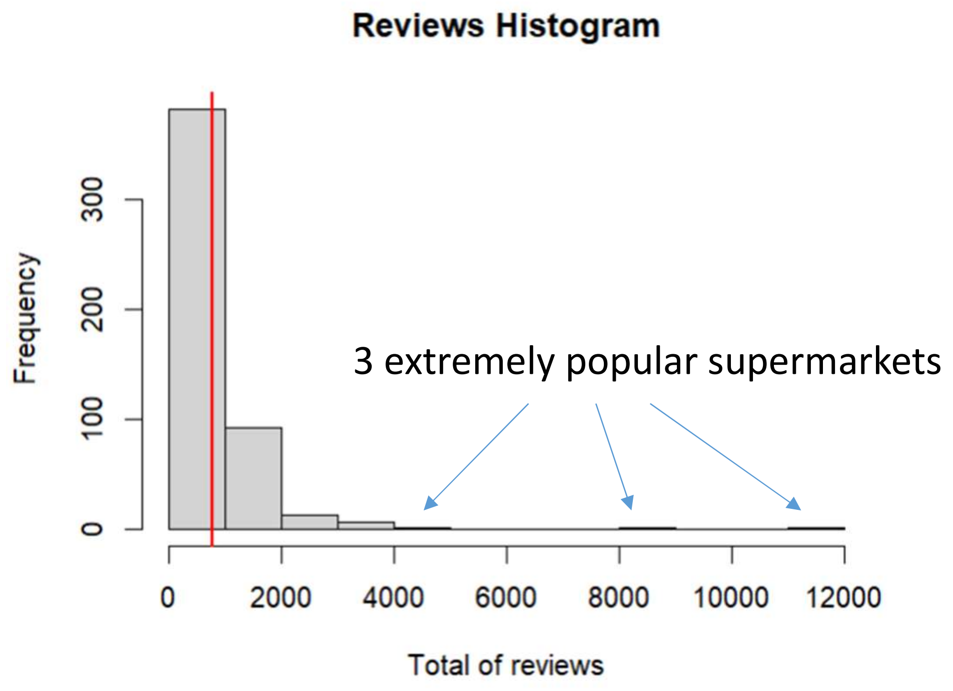


```
rating_sd <- sd(dt_supermarkets$RATING)
print(rating_sd)
[1] 0.2842573
```

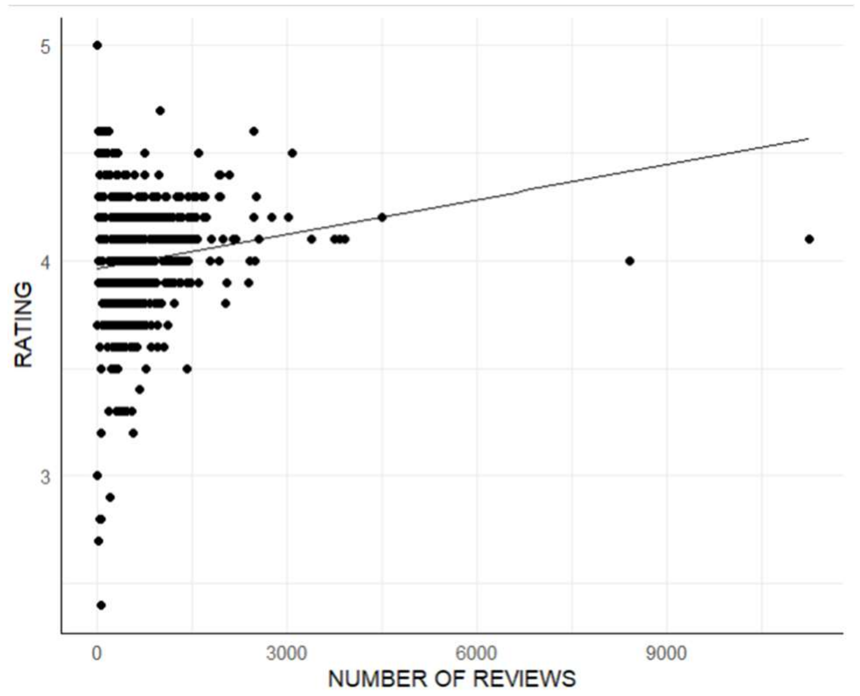
- If is zero then there is no review since the minimium to review is 1
- The Google´s Rating is the weighted average of the reviews by the people
- There is only ratings as integers (1,2,3,4,5) is not possible to rate in decimals

- We can observe that there is less data in less than 3.5 to 2.5 there is not less than 2
- Consumers rate more positively the ranking
- The average rating is at 4.00 with a standard deviation of 0.28 so is close to the mean

Checking the rated reviews

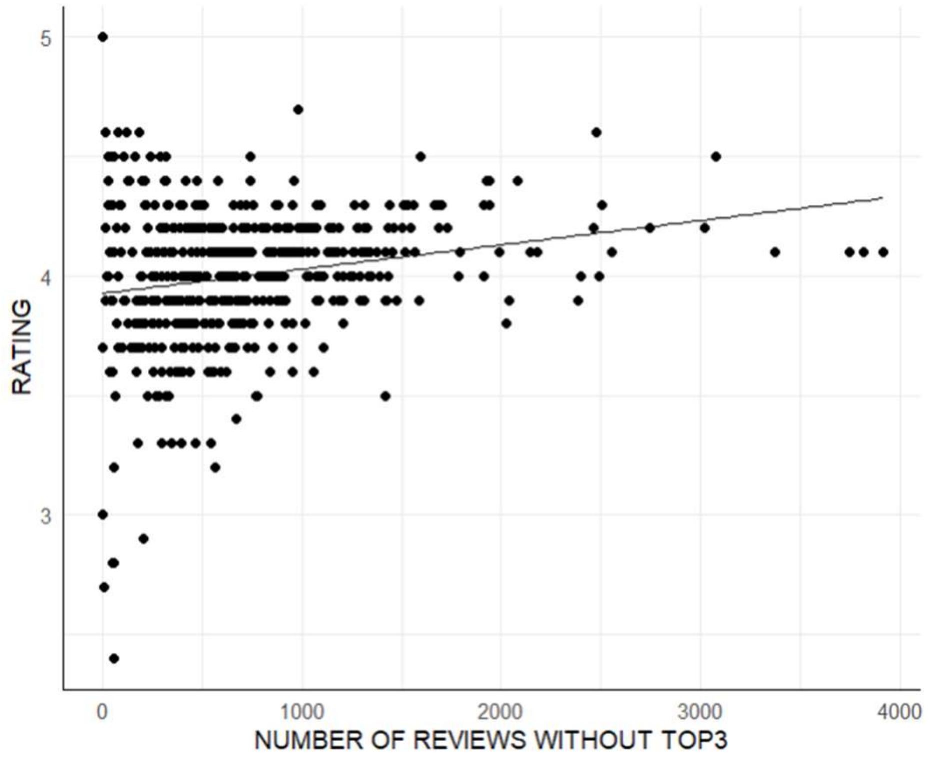


Average of 773.4 reviews, and sd. of 844



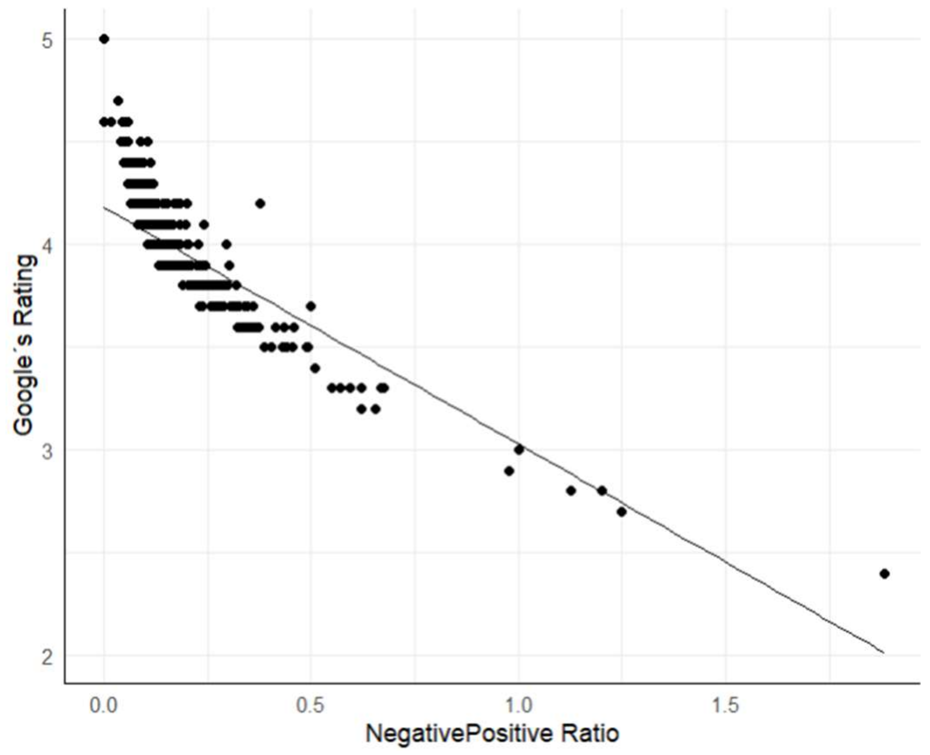
The high positive comments of the top 3 moves (just 3-4% of negative reviews they have) positively the coefficient

Checking the rated reviews



Still a positive relationship between Ratings and Reviews without the Top 3 most rated

Negative/Positive Review Ratio

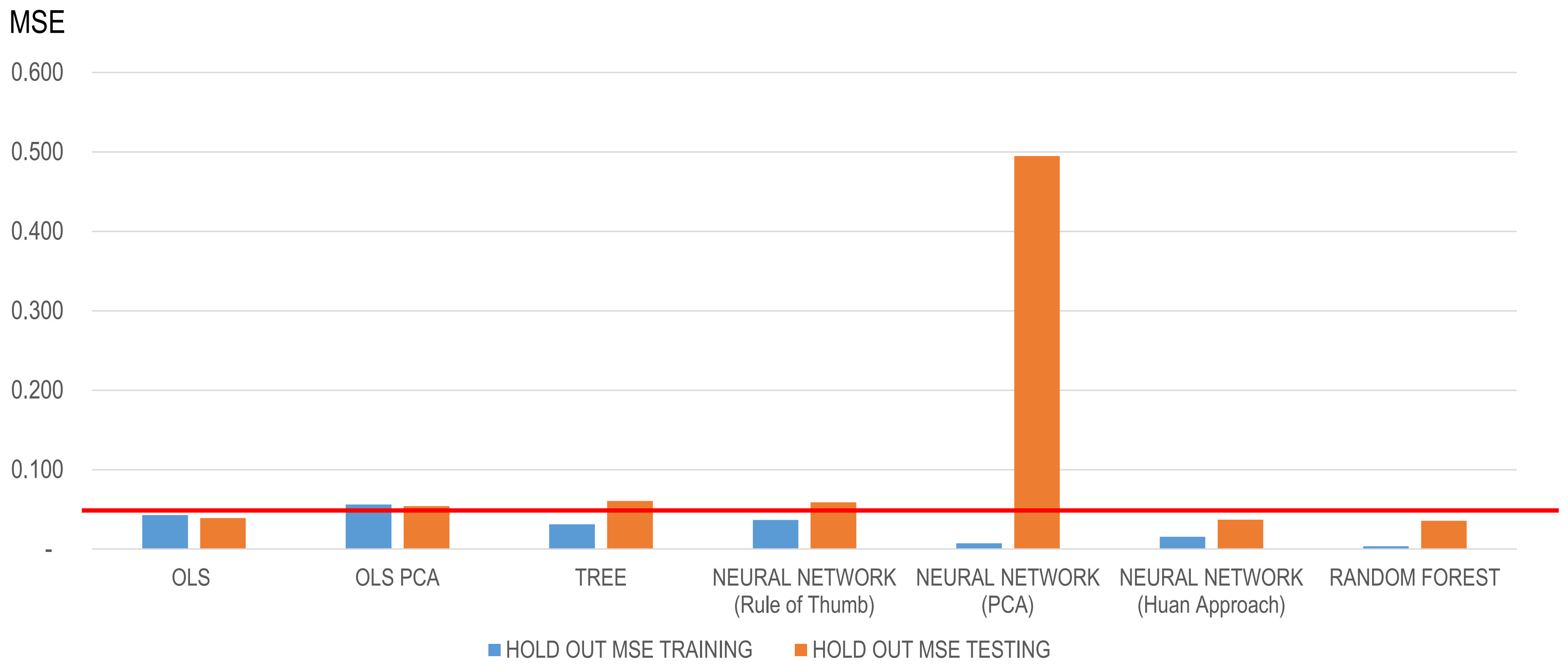


Due to the nature of the rating is obvious that if the supermarket has more negative reviews than positive the increase in one more negative review while the other variables remain constant has a negative effect in average

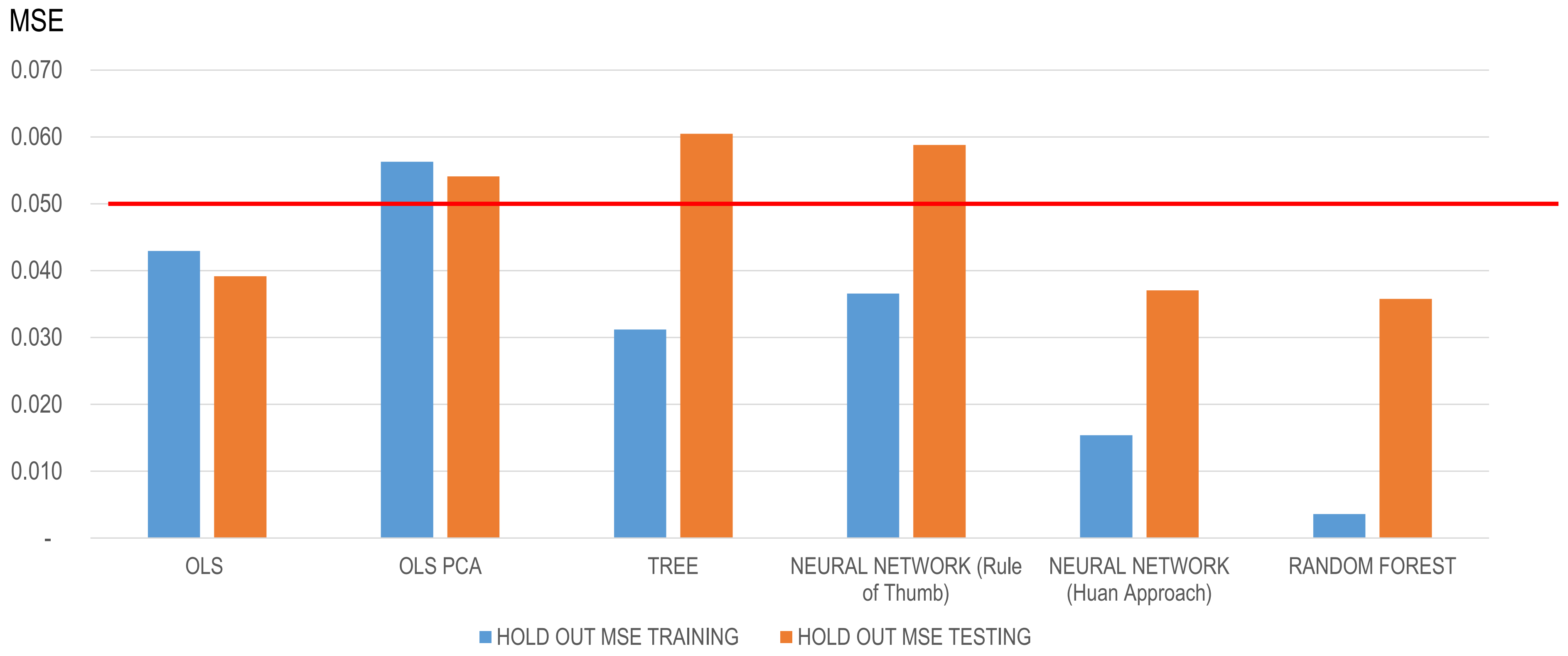
And also it work viceversa

So keeping a track of how many positive or negative reviews the supermarket has is important to predict the rating

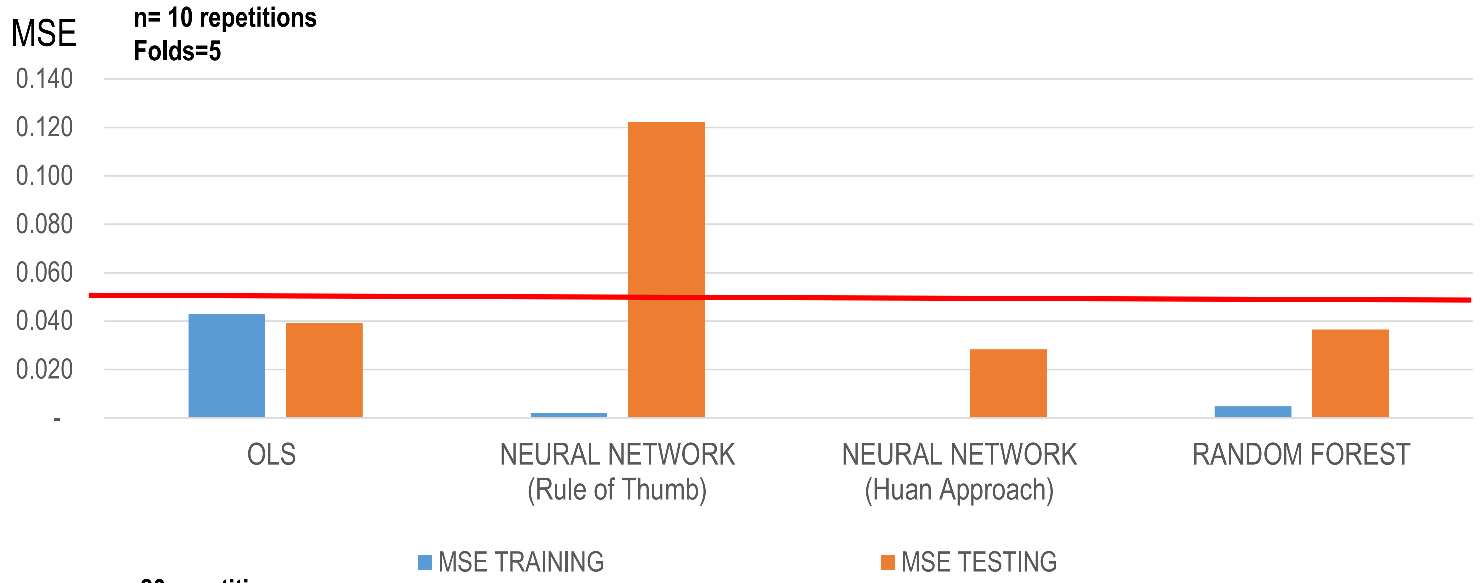
Models with hold out

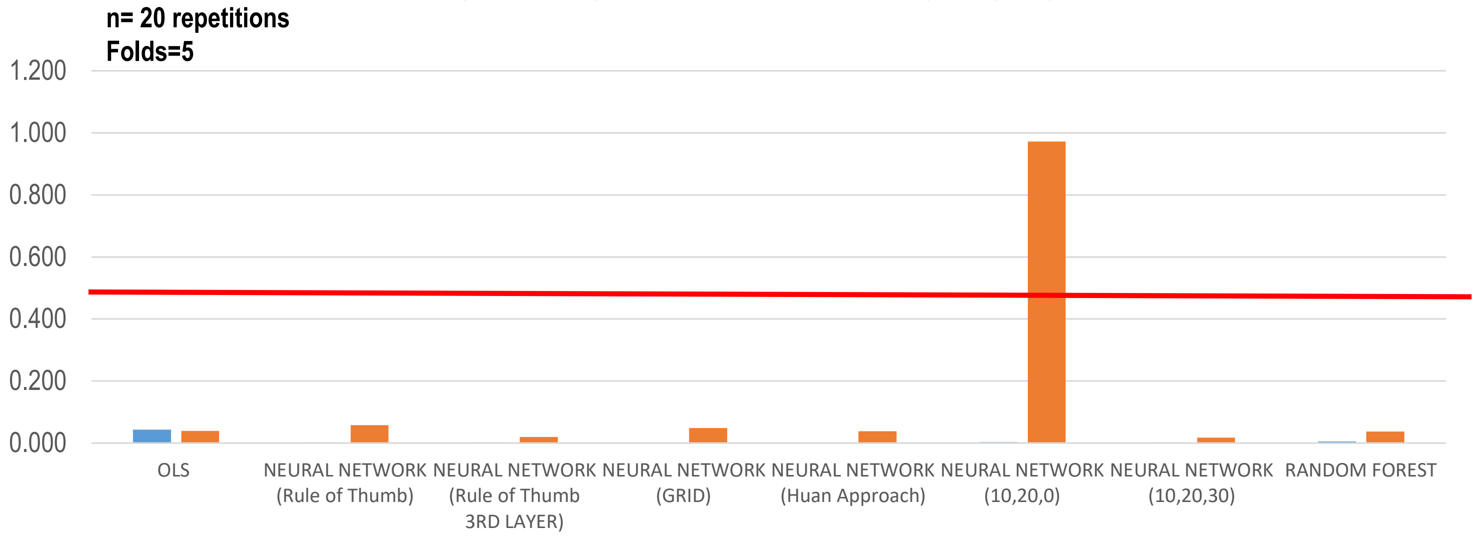


Models with hold out



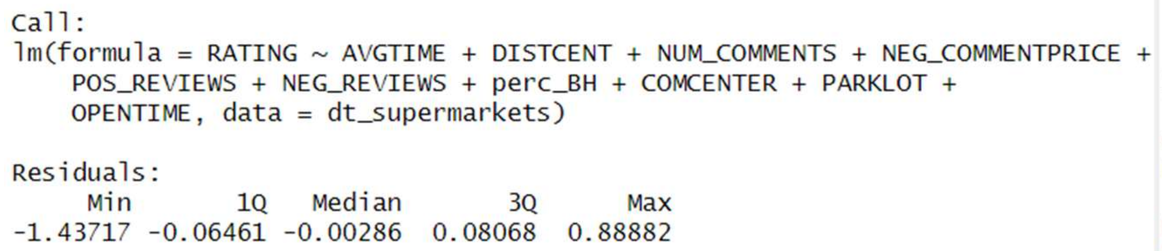
Models with repeated cross validation



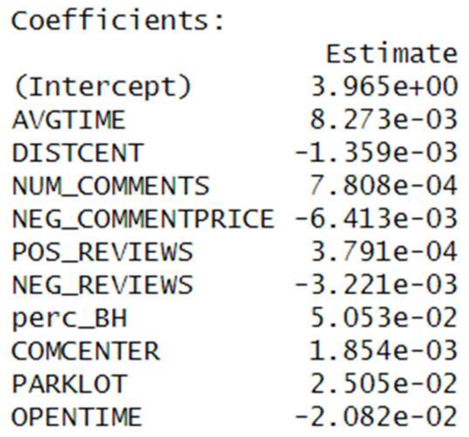


The neural network for Rule of Thumb has an overfiting

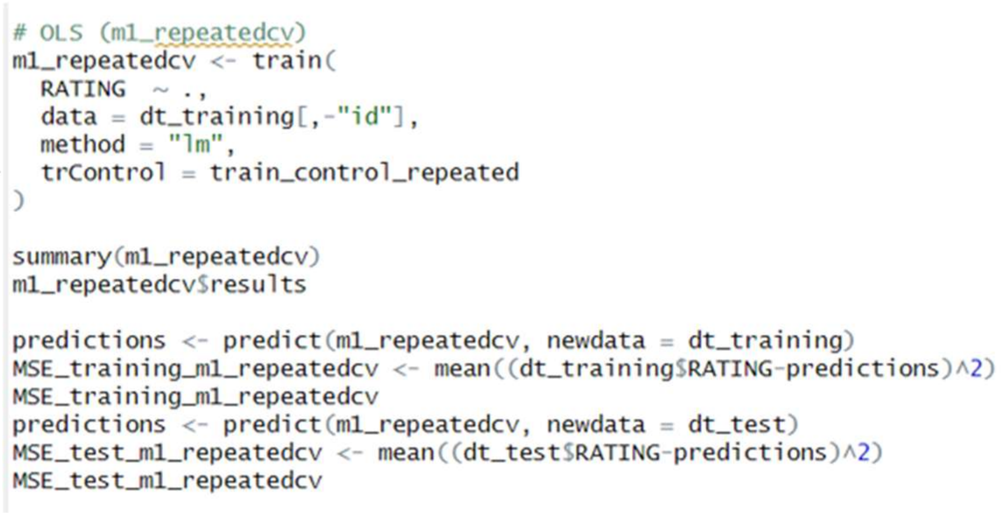
Estimation of the linear model OLS



```
Call:
lm(formula = RATING ~ AVGTIME + DISTCENT + NUM_COMMENTS + NEG_COMMENTPRICE +
    POS_REVIEWS + NEG_REVIEWS + perc_BH + COMCENTER + PARKLOT +
    OPENTIME, data = dt_supermarkets)

Residuals:
     Min       1Q   Median       3Q      Max
-1.43717 -0.06461 -0.00286  0.08068  0.88882
```



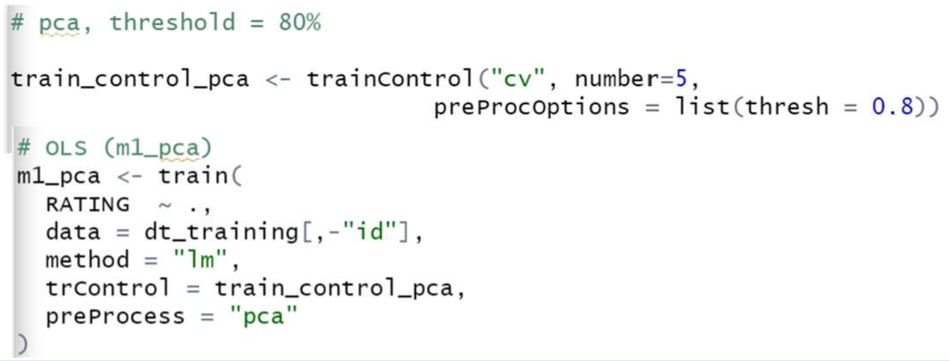
```
Coefficients:
                  Estimate
(Intercept)      3.965e+00
AVGTIME          8.273e-03
DISTCENT        -1.359e-03
NUM_COMMENTS     7.808e-04
NEG_COMMENTPRICE -6.413e-03
POS_REVIEWS      3.791e-04
NEG_REVIEWS     -3.221e-03
perc_BH          5.053e-02
COMCENTER        1.854e-03
PARKLOT          2.505e-02
OPENTIME        -2.082e-02
```



```
# OLS (m1_repeatedcv)
m1_repeatedcv <- train(
  RATING  ~ .,
  data = dt_training[,-"id"],
  method = "lm",
  trControl = train_control_repeated
)

summary(m1_repeatedcv)
m1_repeatedcv$results

predictions <- predict(m1_repeatedcv, newdata = dt_training)
MSE_training_m1_repeatedcv <- mean((dt_training$RATING-predictions)^2)
MSE_training_m1_repeatedcv
predictions <- predict(m1_repeatedcv, newdata = dt_test)
MSE_test_m1_repeatedcv <- mean((dt_test$RATING-predictions)^2)
MSE_test_m1_repeatedcv
```

A MSE training of 0.0429386354

A MSE testing of 0.03917835



```
# pca, threshold = 80%

train_control_pca <- trainControl("cv", number=5,
                                  preProcOptions = list(thresh = 0.8))
 # OLS (m1_pca)
 m1_pca <- train(
   RATING  ~ .,
   data = dt_training[,-"id"],
   method = "lm",
   trControl = train_control_pca,
   preProcess = "pca"
 )
```

- DIST CENT, NEG COMMENTPRICE, NEG REVIEWS and OPENTIME have a negative effect on the rating
- Is not expected that AVG TIME, perc_BH (the ratio between UABhours/#Open hours) have a positive effect on the rating

Neural Networks

We decided to create various neural networks for training and testing

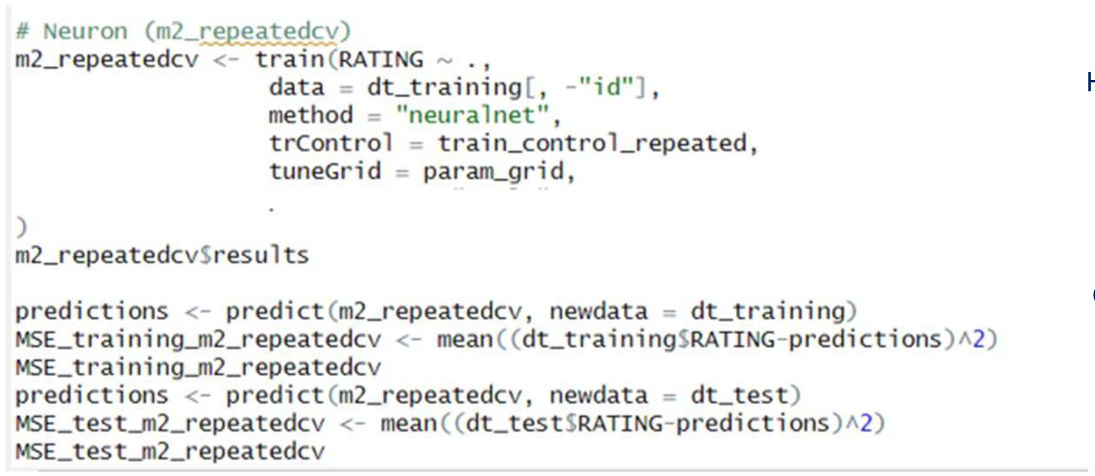
At first we decide to make the standard repeated cross validation with 5 folds and 10 repetitions but our data has only 496 observations so it is consider a small dataset that has high variablity so to improve in tunning we tested in 20 repetitions and 5 folds.

Below the neural networks with is characteristics and the MSE with holdout and with repeated Cross Validations

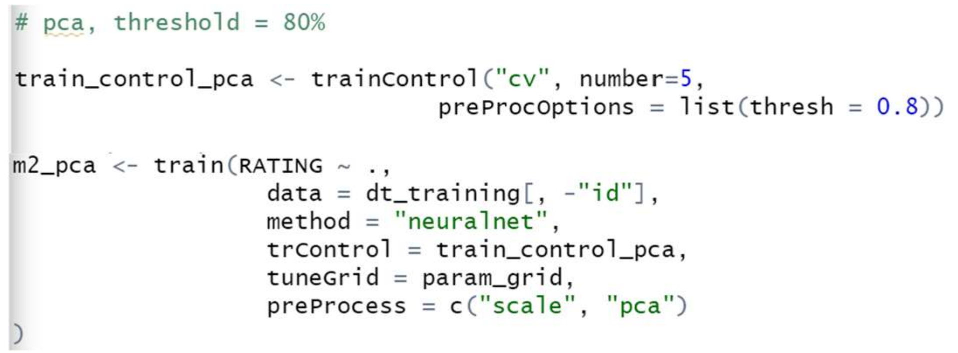
Neural Network	#LAYER	Units in Layers	EPOCH	L.Rate	Holdout	MSE Training Hold put	MSE Test Hold out	RCV (n,folds)	MSE Training RCV	MSE Test RCV	RCV (n,folds)	MSE Training RCV	MSE Test RCV
Rule of Thumb	2	8,16	Default 10000	Default 0.001	Yes	0.037	0.059	Yes, 10,5	0.002	0.122	Yes,20,5	0.002	0.058
PCA	2	8,16	Default	Default	Yes	0.007	**0.495**	No			No		
Huan Approach	2	51,13	Default	0.005	Yes	0.015	0.037	Yes, 10,5	0.0003	0.028	Yes,20,5	0.00023	0.038
Rule of Thumb	3	8.16,32	Default	Default							Yes,20,5	0.001468	0.0193
Grid Search	2	10,20	Default	0.005							Yes,20,5	0.003	0.973
Grid Search	3	10,20,30	Default	0.005							Yes,20,5	0.001637	0.017

Huan: The first layer $\sqrt{(m+2)N} + \sqrt{N/(m+2)}$ and the second $m \sqrt{N/(m+2)}$.

Neural Networks

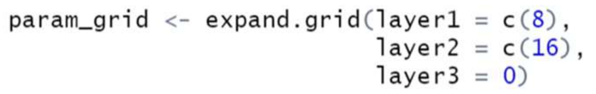


```
# Neuron (m2_repeatedcv)
m2_repeatedcv <- train(RATING ~ .,
                   data = dt_training[, -"id"],
                   method = "neuralnet",
                   trControl = train_control_repeated,
                   tuneGrid = param_grid,
                   .
)
m2_repeatedcv$results

predictions <- predict(m2_repeatedcv, newdata = dt_training)
MSE_training_m2_repeatedcv <- mean((dt_training$RATING-predictions)^2)
MSE_training_m2_repeatedcv
predictions <- predict(m2_repeatedcv, newdata = dt_test)
MSE_test_m2_repeatedcv <- mean((dt_test$RATING-predictions)^2)
MSE_test_m2_repeatedcv
```



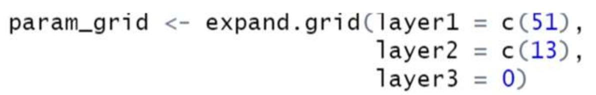
```
# pca, threshold = 80%

train_control_pca <- trainControl("cv", number=5,
                              preProcOptions = list(thresh = 0.8))

m2_pca <- train(RATING ~ .,
                  data = dt_training[, -"id"],
                  method = "neuralnet",
                  trControl = train_control_pca,
                  tuneGrid = param_grid,
                  preProcess = c("scale", "pca")
)
```

Rule of Thumb



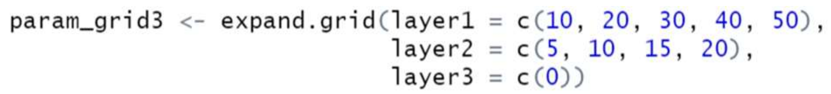
```
param_grid <- expand.grid(layer1 = c(8),
                          layer2 = c(16),
                          layer3 = 0)
```

Huan Approach

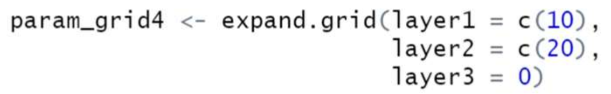


```
param_grid <- expand.grid(layer1 = c(51),
                          layer2 = c(13),
                          layer3 = 0)
```

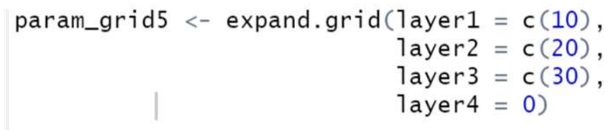
Grid Search



```
param_grid3 <- expand.grid(layer1 = c(10, 20, 30, 40, 50),
                           layer2 = c(5, 10, 15, 20),
                           layer3 = c(0))
```



```
param_grid4 <- expand.grid(layer1 = c(10),
                           layer2 = c(20),
                           layer3 = 0)
```



```
param_grid5 <- expand.grid(layer1 = c(10),
                           layer2 = c(20),
                           layer3 = c(30),
                           layer4 = 0)
```

Heuristic search.

Several heuristics in the literature gained from previous experiments on where a near-optimal topology might exist.

The objective to devise a formula that estimates the # of nodes in the hidden layers as a function of the number of input and output nodes.

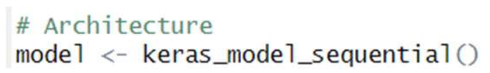
- The estimate can take the form of a single exact topology to be adopted:
  - Hecht-Nielsen 1987: 2n + 1 rule is not for any class of activation functions but for a specific one
  - Hush 1989, Wang 1994
  - Ripley 1993
- or of a range of topologies that should be searched
  - Fletcher and Goss 1993
  - Kanellopoulos and Wilkinson 1997
- others have been introduced by experimenting with spatial data (
  Paola 1994, Wang 1994,
  Kanellopoulos and Wilkinson 1997:

In practice these heuristics are frequently used as points of departure for subsequent search by trial and error

Huang 2003: proved that in 2 hidden-layer case, with m output neurons, the number of hidden nodes that are enough to learn N samples with negligibly small error is given by $2\sqrt{(m+2)N}$
The number of hidden nodes in the first layer is $\sqrt{(m+2)N} + \sqrt{N/(m+2)}$ and the second $m\sqrt{N/(m+2)}$

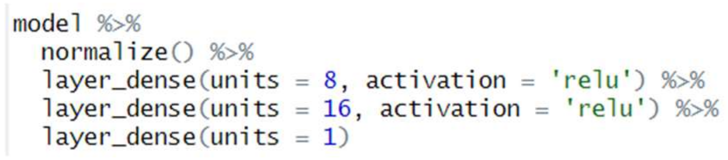
(Yao 1993), i.e. the fact that neural networks produce different results due to different initialization conditions even when everything else is kept fixed. This is why a single run is actually not enough to evaluate a topology. Multiple runs are in fact needed. In this experiment, it has been found that the side-effects of the noisy fitness evaluation problem are reduced by increasing the number of samples. It is known that in remote sensing, the availability of samples varies widely with circumstances. If, however, samples are available, the number of samples used should be progressively increased while observing the variance of the classification results. There is a point where the inclusion of additional samples yields no benefit towards the stabilization of results.

Neural Networks in Keras



```
# Architecture
model <- keras_model_sequential()
```

is used to initialize an empty sequential model. This function sets up a container for the layers to be added subsequently. The model can then be customized by adding layers to it, configuring their parameters, and specifying the desired network architecture.



```
model %>%
  normalize() %>%
  layer_dense(units = 8, activation = 'relu') %>%
  layer_dense(units = 16, activation = 'relu') %>%
  layer_dense(units = 1)
```

model %>% indicates that the subsequent operations will be applied to the model object
normalize() is a function used to normalize the input data

layer_dense(units=8) : adds a dense (fully connected) layer to the model. is used to create the layer. It specifies that the layer will have 8 units/neurons and will use the ReLU activation function

layer dense(units=16): another dense layer to the model, with 16 units and a ReLU activation function

layer_dense(units = 1) This line adds the final dense layer to the model. It specifies that the layer will have 1 unit/neuron, which is often used in regression tasks. No activation function is specified, which means it will default to a linear activation

ReLU (Rectified Linear Unit) ; is a non-linear activation function,

- introduces non-linearity into the network,
- enabling to learn complex patterns.
- is computationally efficient and straightforward to implement.
- can promote sparse activation in nn.
- (only a subset of neurons will be activated for a given input, < redundancy)
- Addressing vanishing gradient problem: has a derivative of 1 for positive values,

allowing gradients to flow more easily during backpropagation and facilitating faster convergence.

- dying ReLU: allowing a small, non-zero output for negative inputs.

For any given input x, ReLU returns x if x is positive, and it returns 0 otherwise. This means that ReLU "rectifies" negative values to zero, while keeping positive values unchanged.

tanh ; is hyperbolic tangent (tanh) function.,

- non-linear transformation applied to the output of each neuron in nn layer
- maps the input values to the range [-1, 1].
- has an S-shaped curve and is symmetric around the origin
- useful in capturing both positive and negative relationships in the data

EPOCH

During each epoch,

- the model makes predictions on the input data,
- calculates the loss (error) between the predicted output and the actual output,
- and adjusts the model's weights using an optimization algorithm (e.g., backpropagation) to minimize the loss.
- The process of passing the data through the network, calculating the loss, and updating the weights is repeated for each epoch.

The number of epochs is a hyperparameter that you need to specify when training a neural network.
Choosing the appropriate # of epochs depends on the complexity of the problem, the size of the dataset, and the convergence behavior of the model.

If you train the network for too few epochs, it may not have sufficient time to learn the underlying patterns in the data, resulting in underfitting.
if you train for too many epochs, the model may start to memorize the training examples excessively, leading to overfitting, where the model performs well on the training data but fails to generalize to new, unseen data.

Finding the right number of epochs often involves monitoring the model's performance on a separate validation dataset. You can observe the validation loss or accuracy during training and stop training when the performance on the validation set starts to degrade, indicating that the model is starting to overfit.
It's worth noting that the number of epochs is a hyperparameter that requires experimentation and tuning.

It depends on various factors such as the complexity of the problem, the amount of training data, the model architecture, and the optimization algorithm used. It's common to try different values for the number of epochs and observe the model's performance on the validation set to find the optimal number of epochs for your specific task.

CARET, Neural Networks with repeated CV

CARET

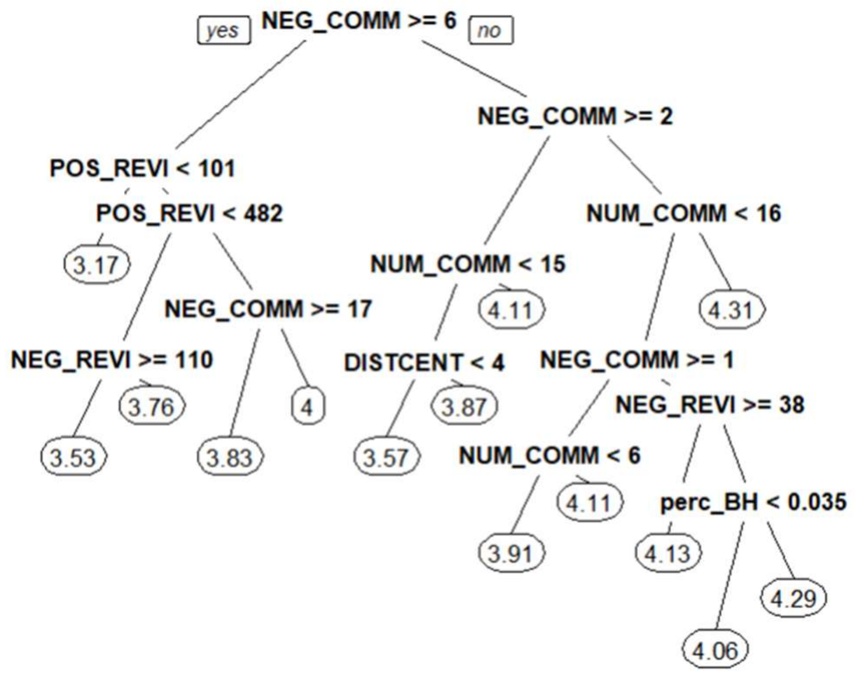
Caret (Classification And REgression Training) is an R package that provides a unified interface for various machine learning algorithms. It’s particularly useful for data scientists who want to compare the performance of different algorithms on a given problem or dataset.

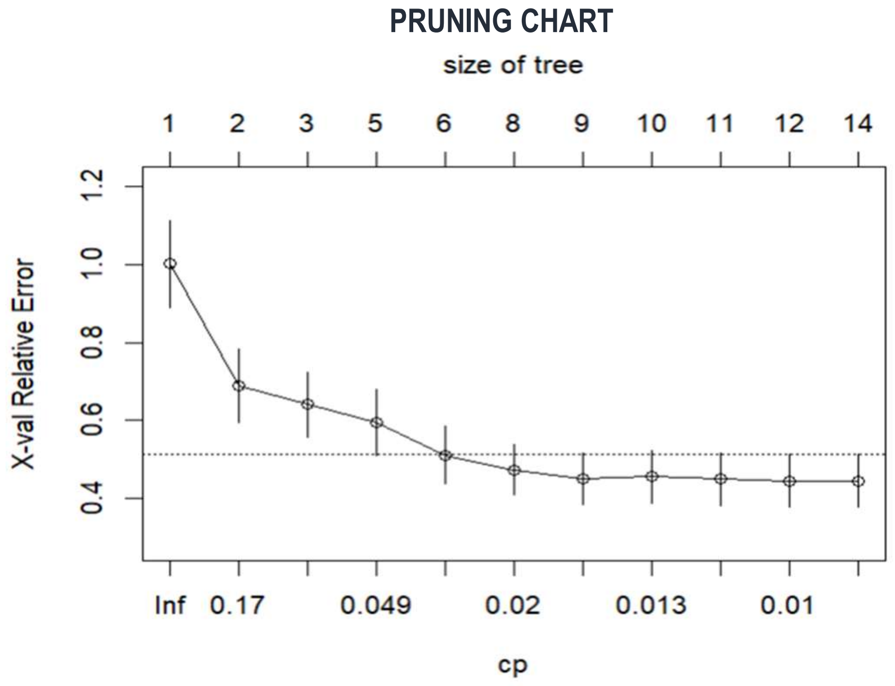
Cross validation is a technique for evaluating the performance of a machine learning model. The idea is to divide your dataset into two parts: a training set and a validation set. You train your model on the training set, and then evaluate its performance on the validation set.
However, there’s a risk of overfitting if you use the same validation set repeatedly to tune your hyperparameters. This is where cross validation comes in. Cross validation involves dividing your dataset into multiple folds, and training your model on each fold while using the other folds for validation. This ensures that you’re evaluating your model on data that it hasn’t seen before, which can help prevent overfitting.

Tree Regression

```
#ESTABLISH A TREE REGRESION WITH ANOVA (ANALISIS OF VARIANCE)

 model_tree_sup <- rpart(data=dt_supermarkets, formula = RATING~AVGTIME+DISTCENT
                         +NUM_COMMENTS+NEG_COMMENTPRICE
                         +POS_REVIEWS +NEG_REVIEWS+perc_BH+COMCENTER+PARKLOT+
                           OPEN6AM+OPEN7AM+
                           OPEN8AM+OPENAFTER8 , method="anova", )
 prp(model_tree_sup, digits = -3)

 summary(model_tree_sup)
 plotcp(model_tree_sup)
```

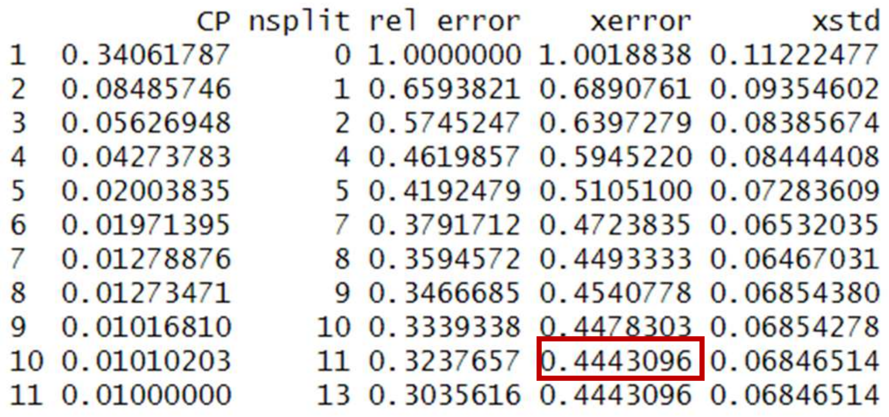
```
          CP nsplit rel error    xerror       xstd
1  0.34061787      0 1.0000000 1.0018838 0.11222477
2  0.08485746      1 0.6593821 0.6890761 0.09354602
3  0.05626948      2 0.5745247 0.6397279 0.08385674
4  0.04273783      4 0.4619857 0.5945220 0.08444408
5  0.02003835      5 0.4192479 0.5105100 0.07283609
6  0.01971395      7 0.3791712 0.4723835 0.06532035
7  0.01278876      8 0.3594572 0.4493333 0.06467031
8  0.01273471      9 0.3466685 0.4540778 0.06854380
9  0.01016810     10 0.3339338 0.4478303 0.06854278
10 0.01010203     11 0.3237657 0.4443096 0.06846514
11 0.01000000     13 0.3035616 0.4443096 0.06846514
```





highest cross-validated error minus the min cross-validatec error

Tree Regression



```
          CP nsplit rel error    xerror       xstd
1  0.34061787      0 1.0000000 1.0018838 0.11222477
2  0.08485746      1 0.6593821 0.6890761 0.09354602
3  0.05626948      2 0.5745247 0.6397279 0.08385674
4  0.04273783      4 0.4619857 0.5945220 0.08444408
5  0.02003835      5 0.4192479 0.5105100 0.07283609
6  0.01971395      7 0.3791712 0.4723835 0.06532035
7  0.01278876      8 0.3594572 0.4493333 0.06467031
8  0.01273471      9 0.3466685 0.4540778 0.06854380
9  0.01016810     10 0.3339338 0.4478303 0.06854278
10 0.01010203     11 0.3237657 0.4443096 0.06846514
11 0.01000000     13 0.3035616 0.4443096 0.06846514
```

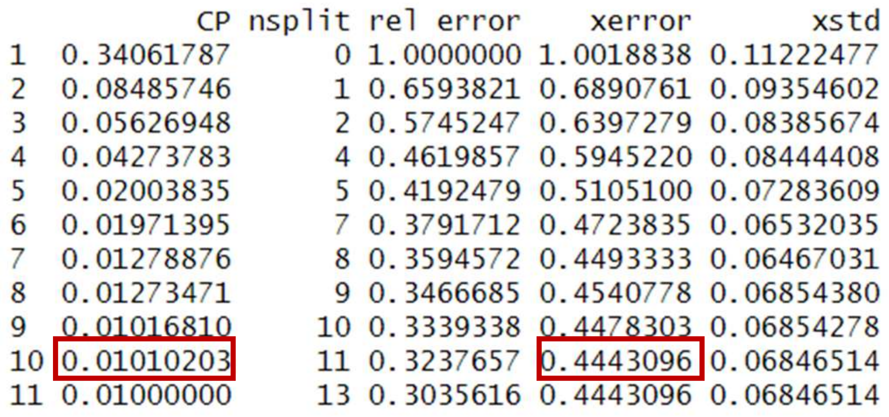
The CP values the higher the value, the smaller the tree (#splits)
A too small value of CP leads overfitting and too large will result to a small tree. Both cases decrease the predictive performance of the model

is 1−R2 Root mean squared error, similar to linear regression. This is the error on the observations used to estimate the model

This is the error on the observations from cross validation data.
In this regression the data use where all for the model and no testing

- An optimal cp value can be estimated by testing different cp values and using cross-validation approaches to determine the corresponding prediction accuracy of the model.
- The best cp is then defined as the one that maximize the cross-validation accuracy

Tree Regression



```
          CP nsplit rel error    xerror       xstd
1  0.34061787      0 1.0000000 1.0018838 0.11222477
2  0.08485746      1 0.6593821 0.6890761 0.09354602
3  0.05626948      2 0.5745247 0.6397279 0.08385674
4  0.04273783      4 0.4619857 0.5945220 0.08444408
5  0.02003835      5 0.4192479 0.5105100 0.07283609
6  0.01971395      7 0.3791712 0.4723835 0.06532035
7  0.01278876      8 0.3594572 0.4493333 0.06467031
8  0.01273471      9 0.3466685 0.4540778 0.06854380
9  0.01016810     10 0.3339338 0.4478303 0.06854278
10 0.01010203     11 0.3237657 0.4443096 0.06846514
11 0.01000000     13 0.3035616 0.4443096 0.06846514
```

The CP values the higher the value, the smaller the tree (#splits)
A too small value of CP leads overfitting and too large will result to a small tree. Both cases decrease the predictive performance of the model

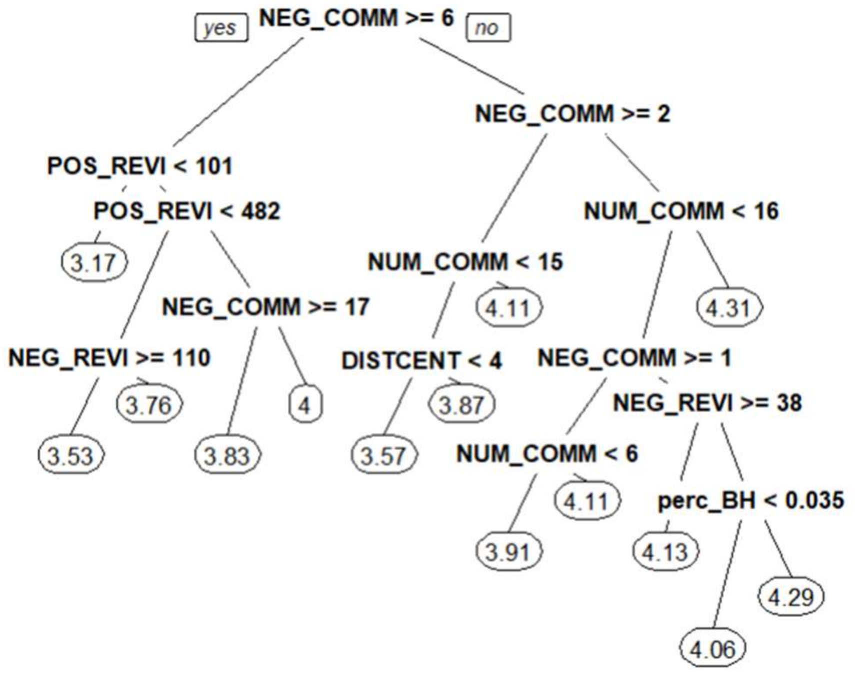
is 1−R2 Root mean squared error, similar to linear regression. This is the error on the observations used to estimate the model

This is the error on the observations from cross validation data.
In this regression the data use where all for the model and no testing

- An optimal cp value can be estimated by testing different cp values and using cross-validation approaches to determine the corresponding prediction accuracy of the model.
- The best cp is then defined as the one that maximize the cross-validation accuracy

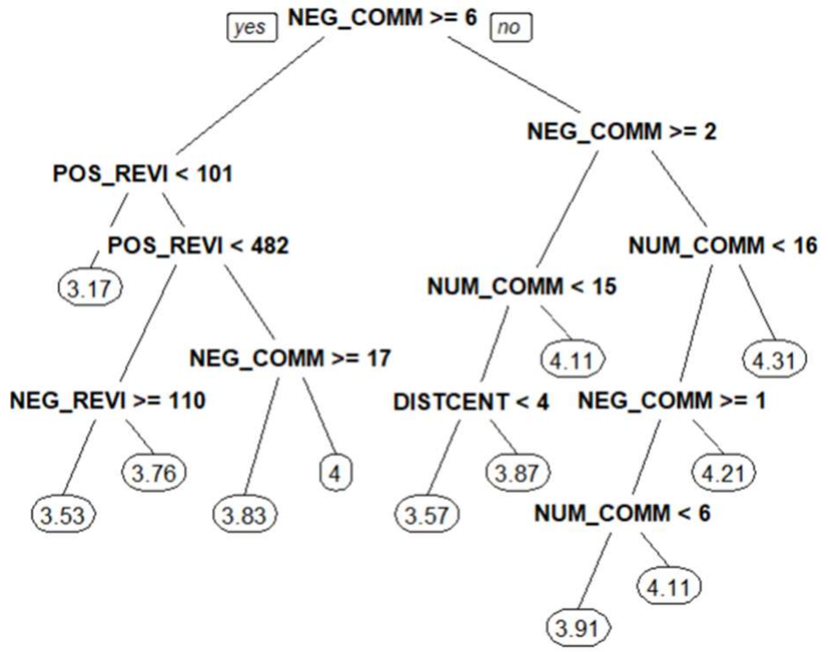
Tree Regression

PREVIOUS TREE WITH NO CPVALUE no PRUNNING



To perform repeated cross-validation with decision trees, the caret package provides a convenient interface and integration with various machine learning algorithms, including decision trees implemented in the rpart package. The train function from caret allows you to easily specify the desired cross-

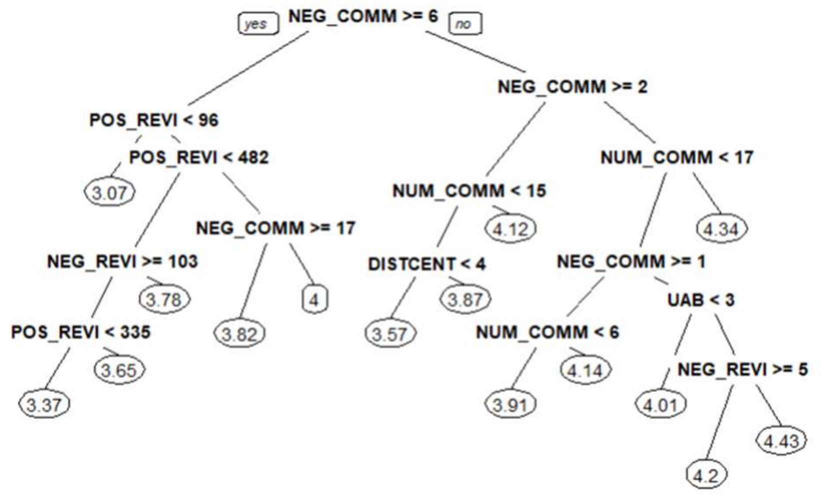
CP VALUE MIN XERROR

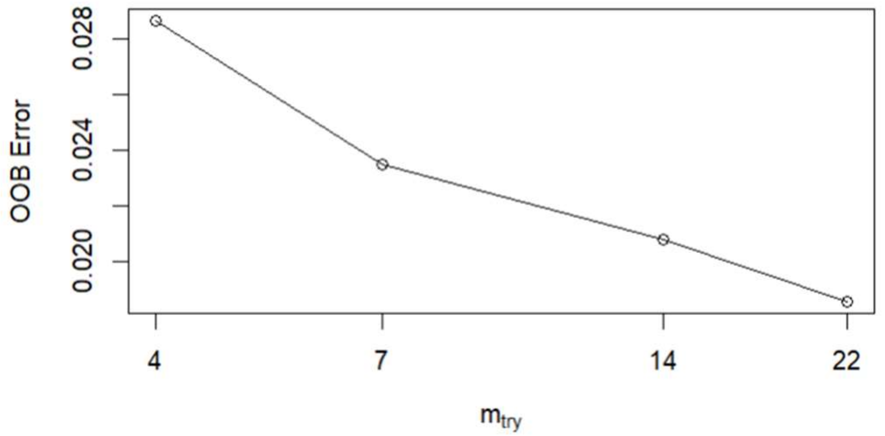


```
>   MSE_TREE
[1] 0.02610824
```

Random Forest

- In the random forest we use the next variables to predict the RATING: UAB, UABSAT, PARKLOT, OPENTIME, OPEN6AM, OPEN7AM, OPEN8AM, OPENAFTER8, REVIEWS, AVGTIME, WORK_24HRS ,SUNDAYS, WORSTREVIEWS, DISTCENT, Open_hourSUM, COMCENTER, NUM_COMMENTS, NEG_COMMENTPRICE, NEG_COMMENTFOOD, NEG_REVIEWS ,POS_REVIEWS, perc_BH

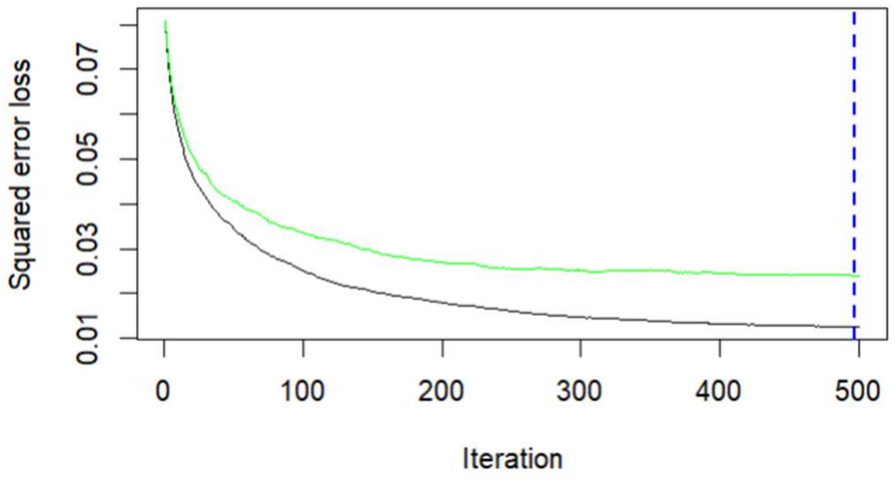




```
Call:
 randomForest(formula = RATING ~ ., data = dt_training, SingleBaggingtry = (ncol(dt_training) -      1), ntree = 500)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 7

          Mean of squared residuals: 0.03046418
                    % Var explained: 64.49
```

Random Forest GRADIENT BOOSTING

```
model_gForest <- gbm(RATING ~ .,
                     data=dt_training,
                     distribution="gaussian",
                     interaction.depth=1,
                     n.trees=500,
                     shrinkage=0.1,
                     cv.folds = 5
)

gbm.perf(model_gForest, method = "cv")
```



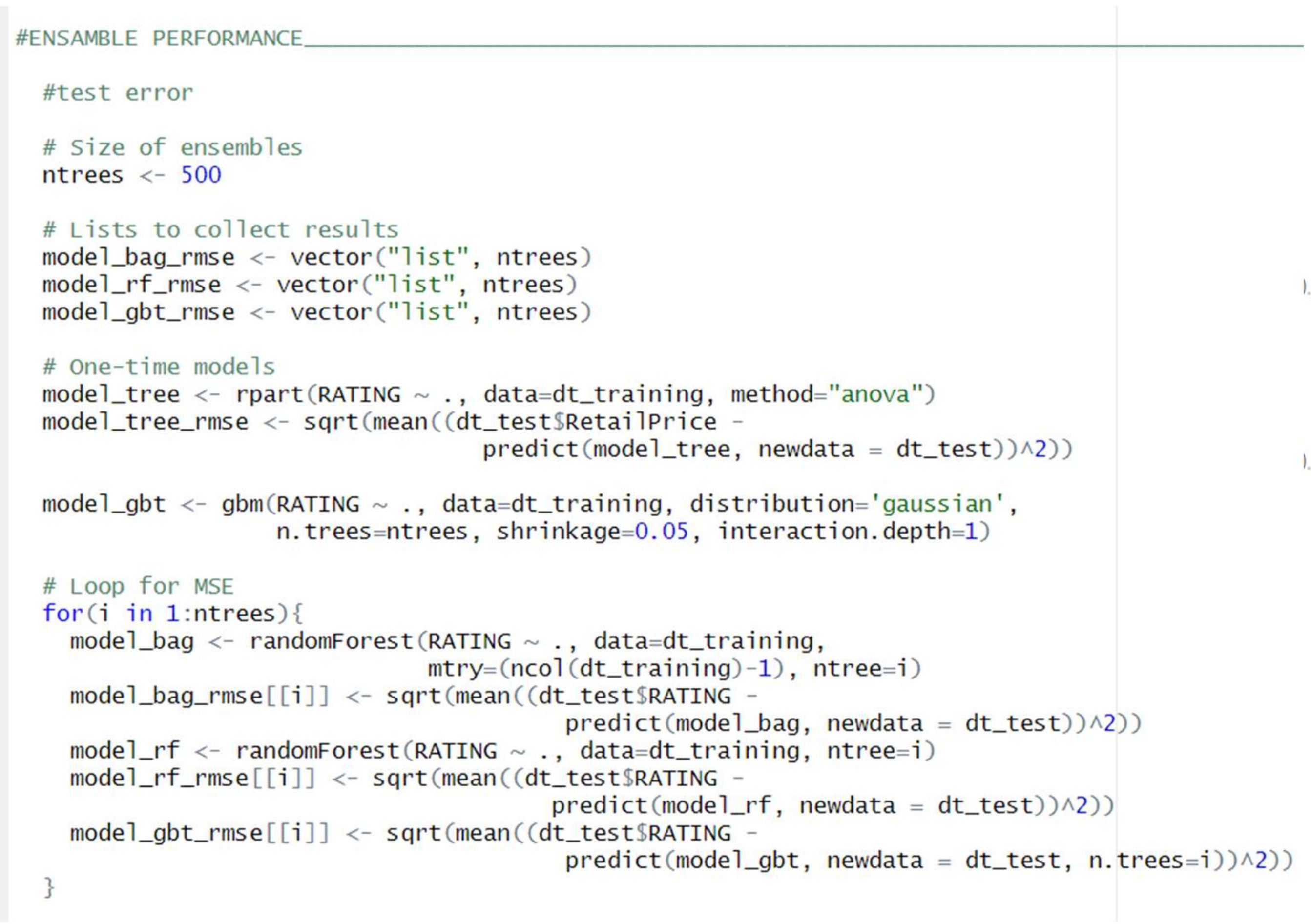
we are boosting tree stumps

The ensemble size is set to 500 to compare the results to the previous models

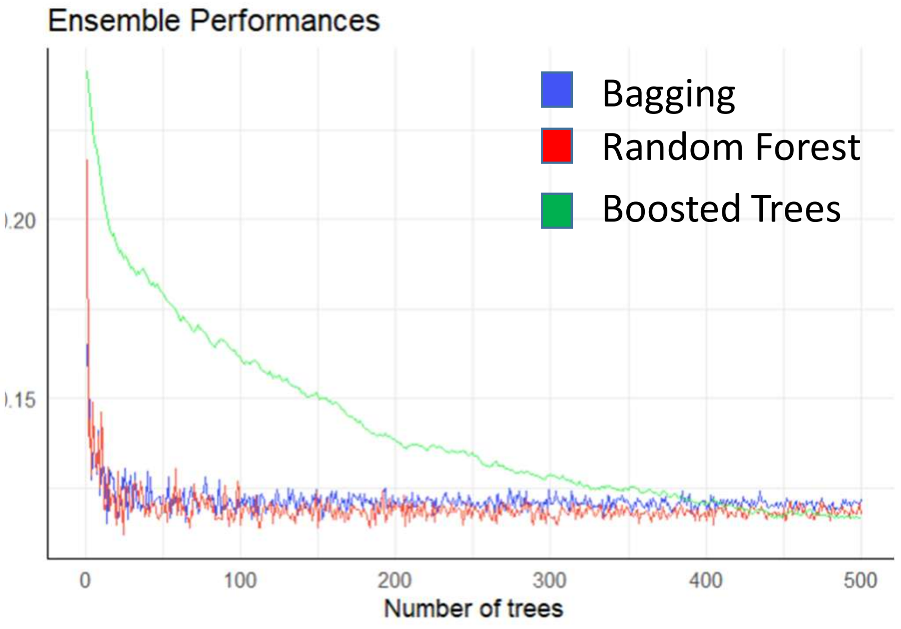
learning rate in *shrinkage* which is as crucial for our gradient descent algorithm as it has been for neural networks.

A larger learning rate speeds up the descent but may not find optimal solution(s).

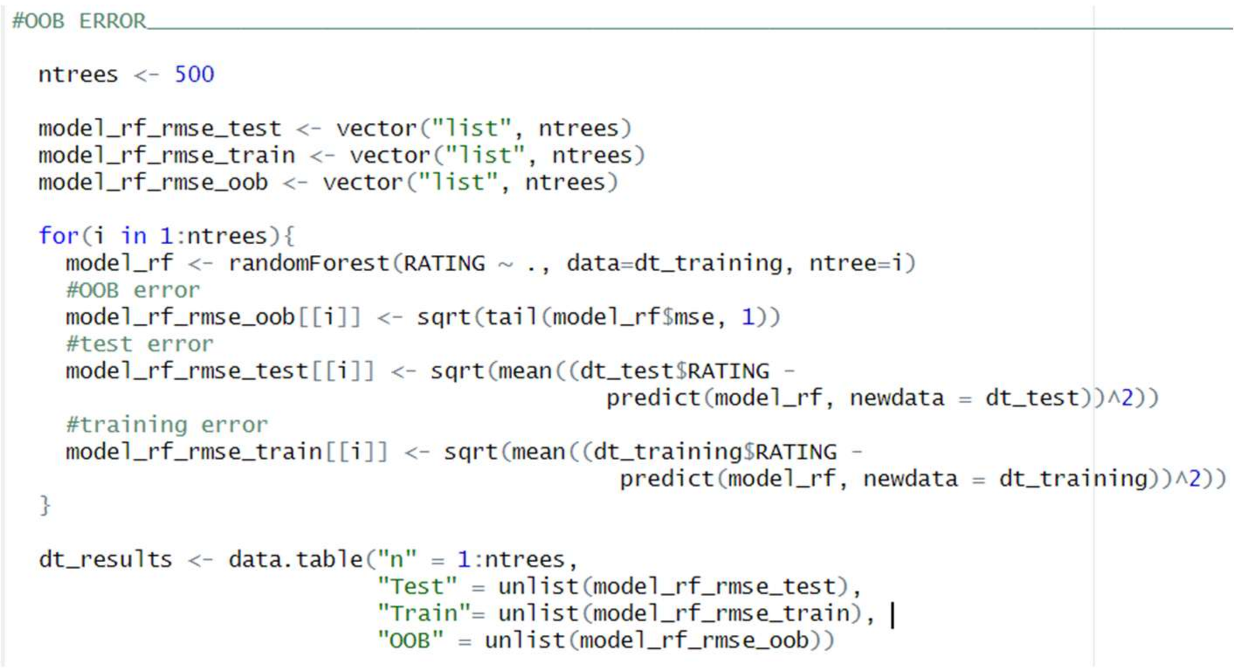
Random Forest TEST ERROR



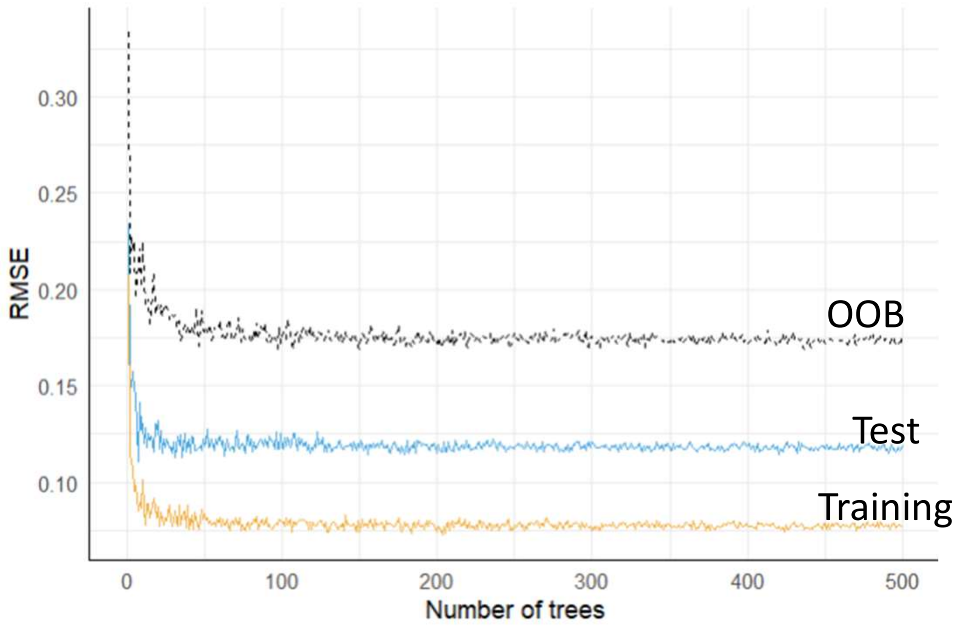
```
#ENSAMBLE PERFORMANCE______________________________________________________________

  #test error

  # Size of ensembles
  ntrees <- 500

  # Lists to collect results
  model_bag_rmse <- vector("list", ntrees)
  model_rf_rmse <- vector("list", ntrees)
  model_gbt_rmse <- vector("list", ntrees)

  # One-time models
  model_tree <- rpart(RATING ~ ., data=dt_training, method="anova")
  model_tree_rmse <- sqrt(mean((dt_test$RetailPrice -
                                predict(model_tree, newdata = dt_test))^2))

  model_gbt <- gbm(RATING ~ ., data=dt_training, distribution='gaussian',
                   n.trees=ntrees, shrinkage=0.05, interaction.depth=1)

  # Loop for MSE
  for(i in 1:ntrees){
    model_bag <- randomForest(RATING ~ ., data=dt_training,
                              mtry=(ncol(dt_training)-1), ntree=i)
    model_bag_rmse[[i]] <- sqrt(mean((dt_test$RATING -
                                      predict(model_bag, newdata = dt_test))^2))
    model_rf <- randomForest(RATING ~ ., data=dt_training, ntree=i)
    model_rf_rmse[[i]] <- sqrt(mean((dt_test$RATING -
                                     predict(model_rf, newdata = dt_test))^2))
    model_gbt_rmse[[i]] <- sqrt(mean((dt_test$RATING -
                                      predict(model_gbt, newdata = dt_test, n.trees=i))^2))
  }
```



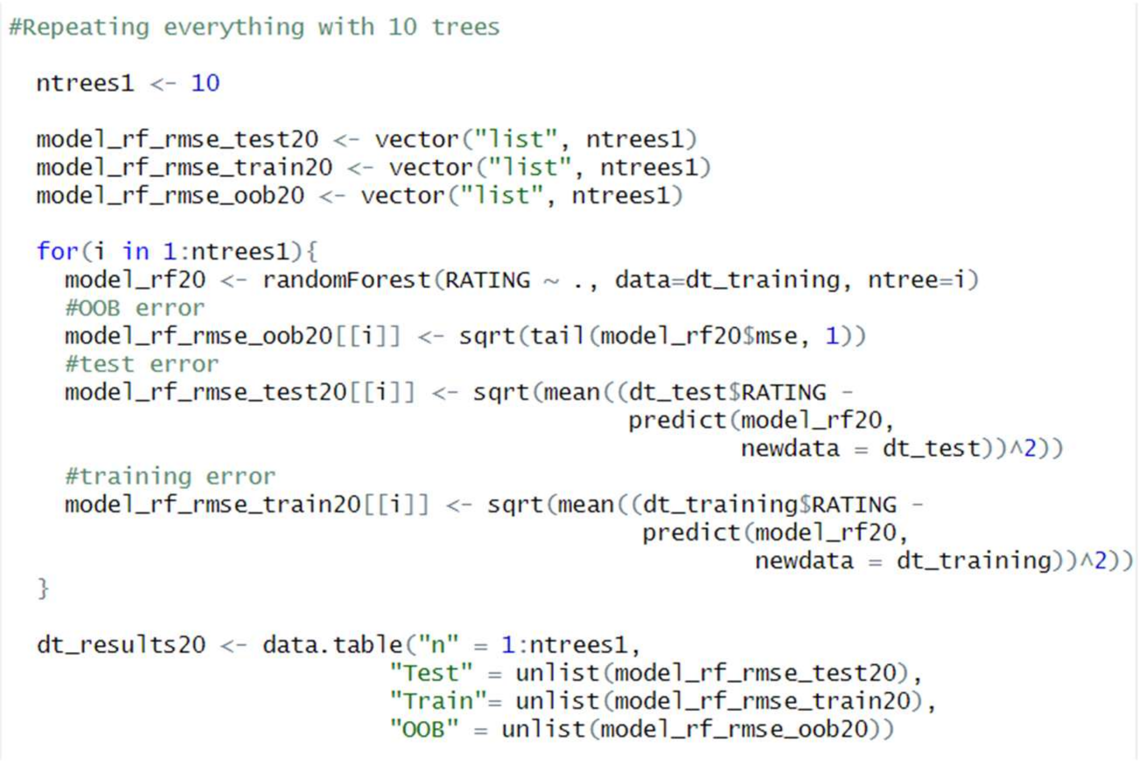
Random Forest OOB



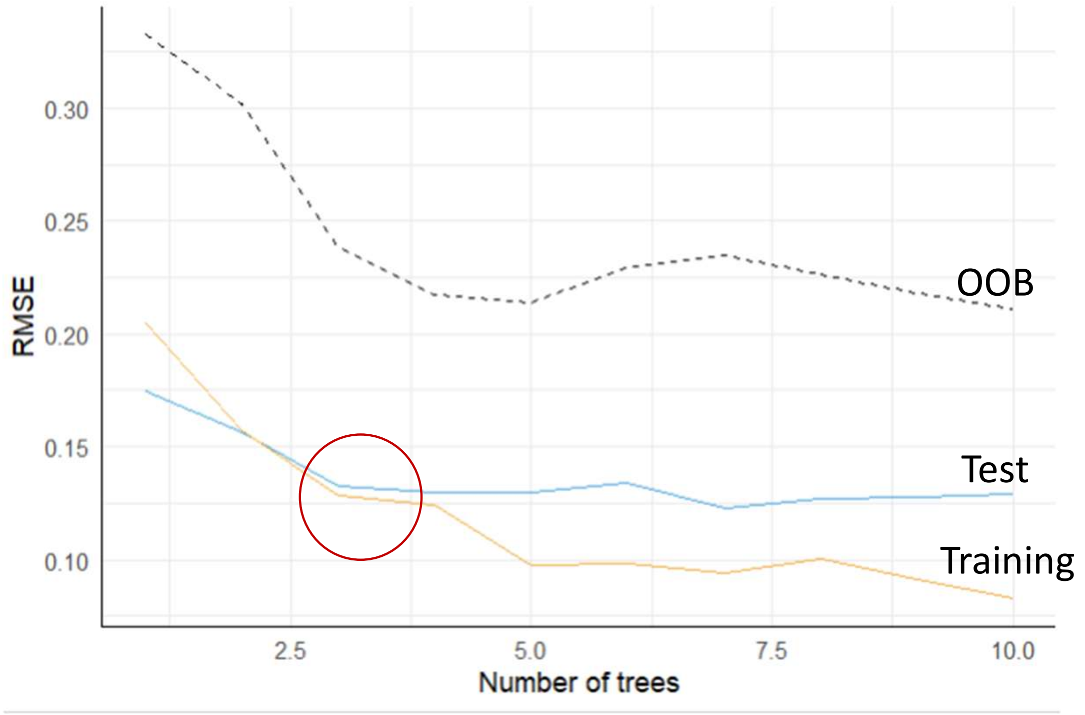
```
#OOB ERROR

  ntrees <- 500

  model_rf_rmse_test <- vector("list", ntrees)
  model_rf_rmse_train <- vector("list", ntrees)
  model_rf_rmse_oob <- vector("list", ntrees)

  for(i in 1:ntrees){
    model_rf <- randomForest(RATING ~ ., data=dt_training, ntree=i)
    #OOB error
    model_rf_rmse_oob[[i]] <- sqrt(tail(model_rf$mse, 1))
    #test error
    model_rf_rmse_test[[i]] <- sqrt(mean((dt_test$RATING -
                                          predict(model_rf, newdata = dt_test))^2))
    #training error
    model_rf_rmse_train[[i]] <- sqrt(mean((dt_training$RATING -
                                           predict(model_rf, newdata = dt_training))^2))
  }

  dt_results <- data.table("n" = 1:ntrees,
                           "Test" = unlist(model_rf_rmse_test),
                           "Train"= unlist(model_rf_rmse_train),
                           "OOB" = unlist(model_rf_rmse_oob))
```

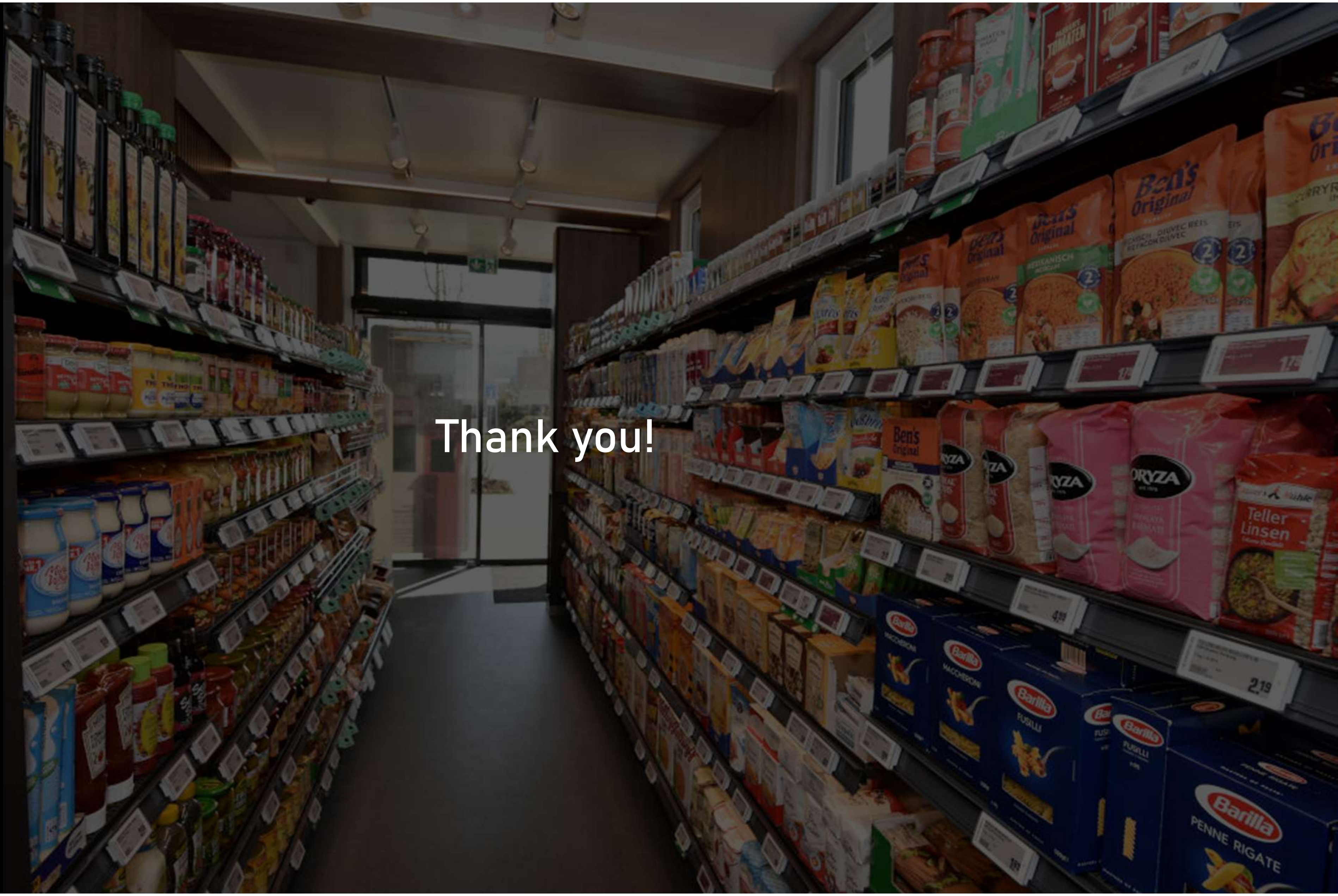


Random Forest OOB With 10 trees



```
#Repeating everything with 10 trees

 ntrees1 <- 10

 model_rf_rmse_test20 <- vector("list", ntrees1)
 model_rf_rmse_train20 <- vector("list", ntrees1)
 model_rf_rmse_oob20 <- vector("list", ntrees1)

 for(i in 1:ntrees1){
   model_rf20 <- randomForest(RATING ~ ., data=dt_training, ntree=i)
   #OOB error
   model_rf_rmse_oob20[[i]] <- sqrt(tail(model_rf20$mse, 1))
   #test error
   model_rf_rmse_test20[[i]] <- sqrt(mean((dt_test$RATING -
                                         predict(model_rf20,
                                                 newdata = dt_test))^2))

   #training error
   model_rf_rmse_train20[[i]] <- sqrt(mean((dt_training$RATING -
                                          predict(model_rf20,
                                                  newdata = dt_training))^2))
 }

 dt_results20 <- data.table("n" = 1:ntrees1,
                            "Test" = unlist(model_rf_rmse_test20),
                            "Train"= unlist(model_rf_rmse_train20),
                            "OOB" = unlist(model_rf_rmse_oob20))
```



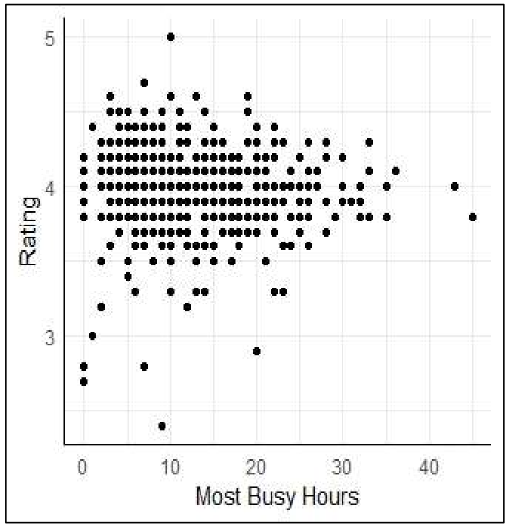


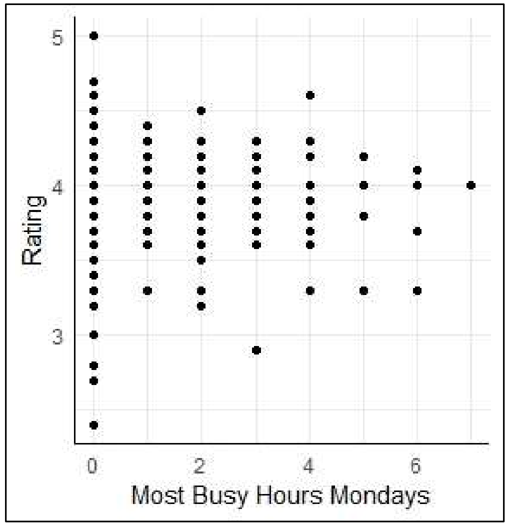
Thank you!

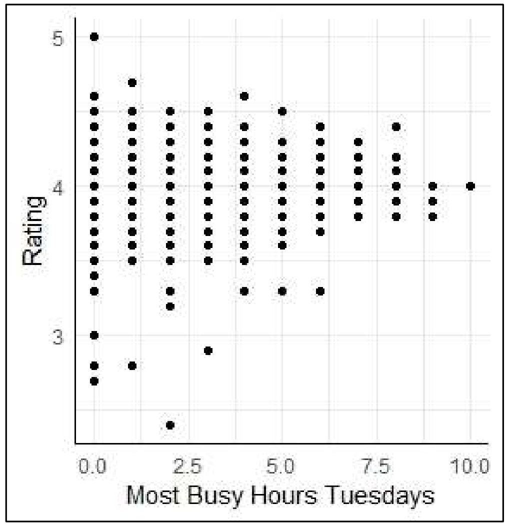
Checking the distribution and scale of the variables

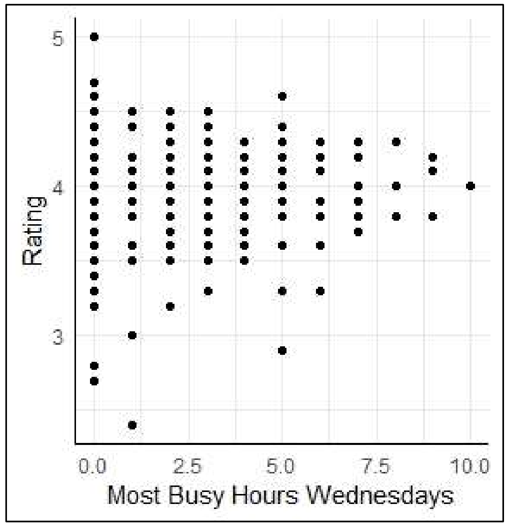
Variable	MIN	MAX	AVG	RANGE
RATING	2.4	5	4.002	[1,5]
UAB	0	45	12.79	[0,126]
REVIEWS	1	11,233	773.5	[1,
NEGREVIEWS	1	401	49.4	[1,
AVGTIME	0	42.5	17.32	[0,
DISTCENT	0.006	0.309	0.094	[0,

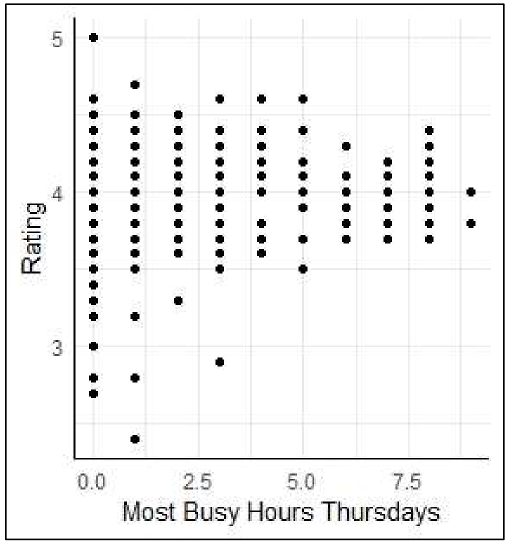
Checking the relationship between Rating and UAB

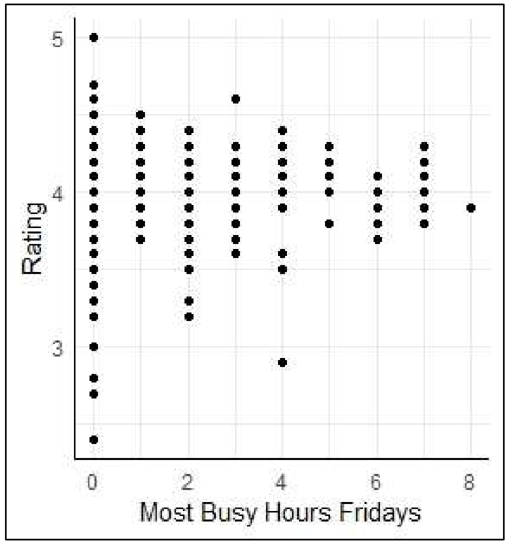


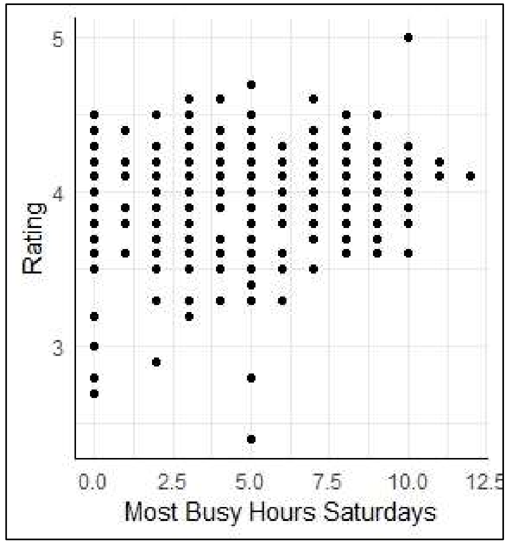




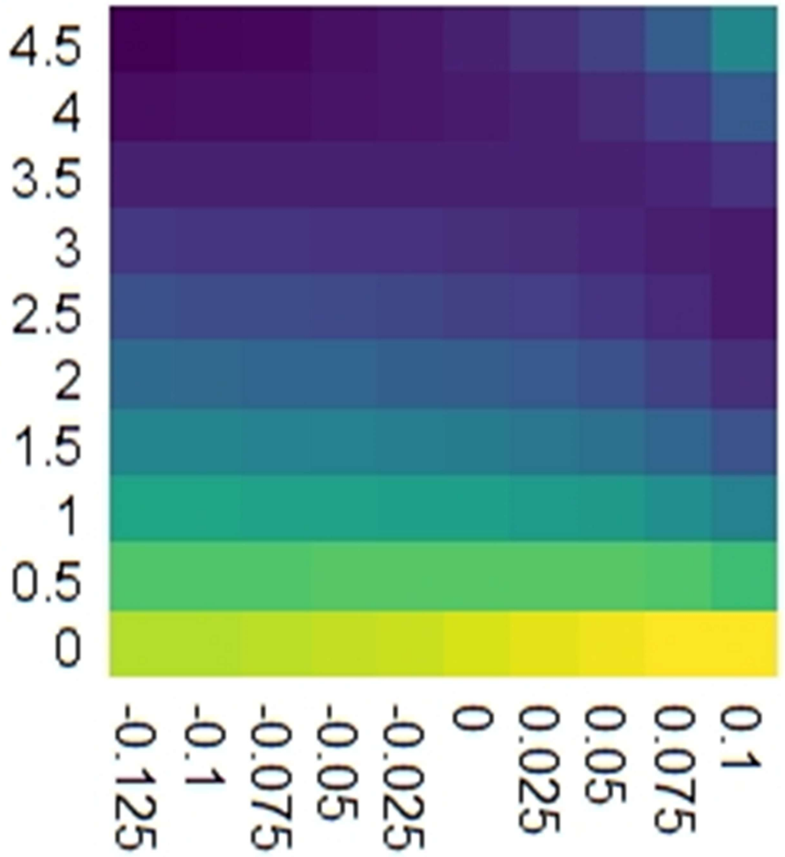






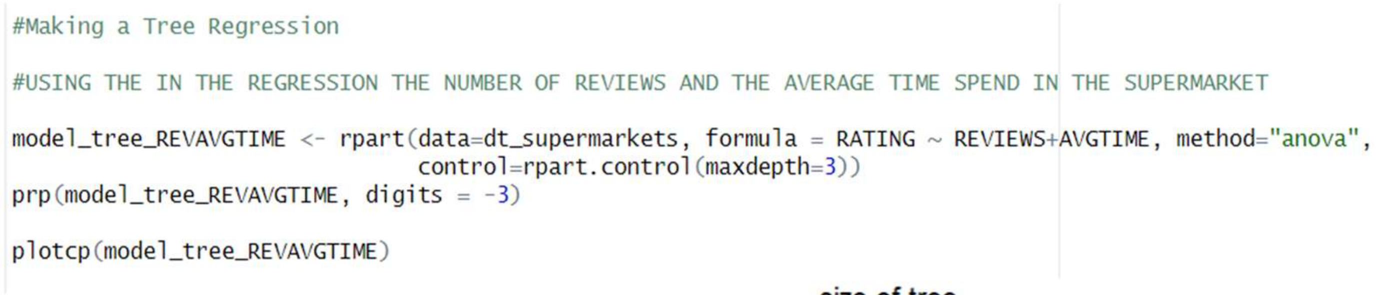


MSE of UAB and MSE of Reviews

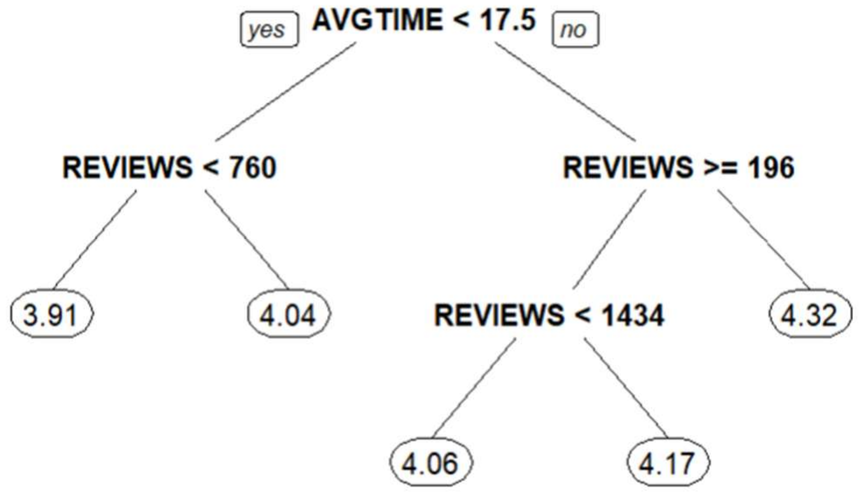


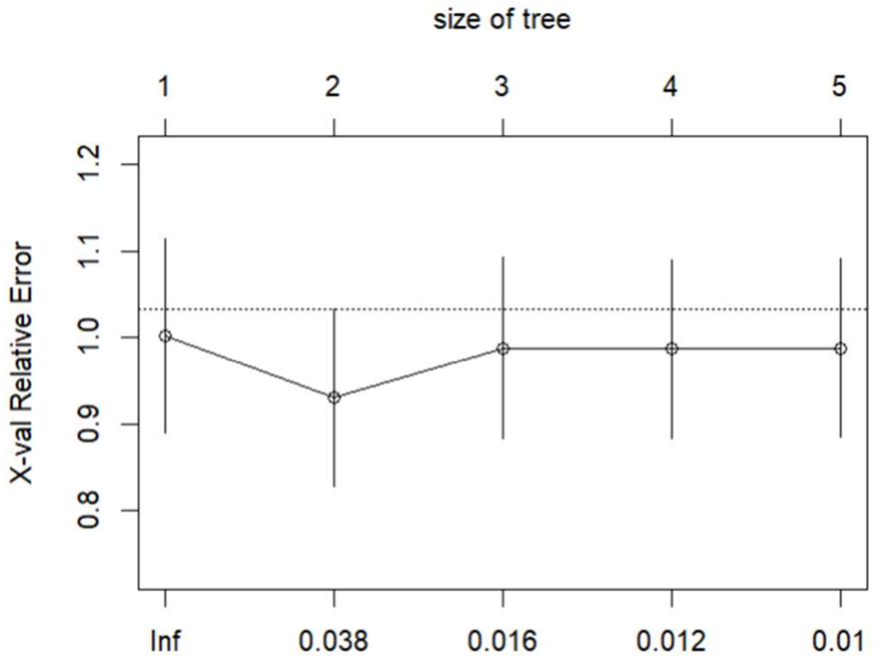
```
       4.5 4 3.5 3 2.5  2 1.5  1 0.5  0
0.1      4 2   1 1   1  1   2  4   6  8
0.075    3 1   1 0   1  2   3  5   7 10
0.05     2 1   0 0   1  2   4  6   8 12
0.025    1 0   0 1   2  3   5  7  10 14
0        0 0   0 1   2  4   6  9  12 16
-0.025   0 0   1 2   3  6   8 11  15 19
-0.05    0 1   2 3   5  7  10 13  17 22
-0.075   1 1   3 4   6  9  12 16  20 25
-0.1     1 2   4 6   8 11  15 19  24 29
-0.125   2 4   5 8  11 14  18 22  27 32
```

Decision Tree



```
#Making a Tree Regression

#USING THE IN THE REGRESSION THE NUMBER OF REVIEWS AND THE AVERAGE TIME SPEND IN THE SUPERMARKET

model_tree_REVAVGTIME <- rpart(data=dt_supermarkets, formula = RATING ~ REVIEWS+AVGTIME, method="anova",
                               control=rpart.control(maxdepth=3))
prp(model_tree_REVAVGTIME, digits = -3)

plotcp(model_tree_REVAVGTIME)
```





We can observed that the rating goes higher if the avg.time spend is higher than the most frequent and the reviews gets between 196 and 760
But if the threeshold of 760 is pass then the rating gets lower (clearly because it can get more negative reviews)