Kaggle Playground Season 3 Episode 5 - Wine Quality Data Competition

Part 1 - Data Exploration, Preparation and Model Spot-Checking

The Kaggle Playground competitions provide monthly opportunities to practice skills on realistic, real world type data sets.

This is my first try. The challenge is to predict wine quality ratings (between 3 and 8), based on wine chemistry data. This version was done in R.

Load Packages

In [1]:

```
#load packages

package_names <- c("tidyverse", "caret", "Amelia", "UBL", "corrplot")

check_install_load_packages <- function(package_names){
    for (i in package_names) {
           # if(!i %in% installed.packages()){
           #     install.packages(i, dependencies = c("Depends", "Suggests")))
           #     }
            if(!i %in% (.packages())){
                library(i, character.only = TRUE)
                }
    }
}

check_install_load_packages(package_names)
(.packages())
```

```
── Attaching packages ─────────────────────────────────────── tidyverse
1.3.2 ──
✓ ggplot2 3.4.0      ✓ purrr   1.0.1
✓ tibble  3.1.8      ✓ dplyr   1.0.10
✓ tidyr   1.2.1      ✓ stringr 1.5.0
✓ readr   2.1.3      ✓ forcats 0.5.2
── Conflicts ────────────────────────────────────────── tidyverse_confl
icts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
Loading required package: lattice


Attaching package: 'caret'


The following object is masked from 'package:purrr':

    lift


The following object is masked from 'package:httr':

    progress


Loading required package: Rcpp

##
## Amelia II: Multiple Imputation
## (Version 1.8.1, built: 2022-11-18)
## Copyright (C) 2005-2023 James Honaker, Gary King and Matthew Bla
ckwell
## Refer to http://gking.harvard.edu/amelia/ for more information
##

Loading required package: MBA

Loading required package: gstat

Loading required package: automap

Loading required package: sp
```

```
Loading required package: randomForest

randomForest 4.6-14

Type rfNews() to see new features/changes/bug fixes.


Attaching package: ‘randomForest’


The following object is masked from ‘package:dplyr’:

    combine


The following object is masked from ‘package:ggplot2’:

    margin


corrplot 0.92 loaded
```

'corrplot' · 'UBL' · 'randomForest' · 'automap' · 'sp' · 'gstat' · 'MBA' · 'Amelia' · 'Rcpp' · 'caret' · 'lattice' · 'forcats' · 'stringr' · 'dplyr' · 'purrr' · 'readr' · 'tidyr' · 'tibble' · 'ggplot2' · 'tidyverse' · 'stats' · 'graphics' · 'grDevices' · 'utils' · 'datasets' · 'bigrquery' · 'httr' · 'methods' · 'base'

Import Data

In [2]:

```
#import data
train <- read.csv(file = "/kaggle/input/playground-series-season-3-episode-5/train.csv")
test <- read.csv(file = "/kaggle/input/playground-series-season-3-episode-5/test.csv")
head(train, 3)
dim(train)
table(train$quality)
dim(test)
head(test, 3)
```

A data.frame: 3 × 13

	Id	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxid
	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	0	8.0	0.50	0.39	2.2	0.073	30
2	1	9.3	0.30	0.73	2.3	0.092	30
3	2	7.1	0.51	0.03	2.1	0.059	3

2056 · 13

3	4	5	6	7	8
12	55	839	778	333	39

1372 · 12

A data.frame: 3 × 12

	Id	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxid
	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	2056	7.2	0.510	0.01	2	0.077	31
2	2057	7.2	0.755	0.15	2	0.102	14
3	2058	8.4	0.460	0.40	2	0.065	21

We can see that the data is fairly unbalanced between the different quality categories.

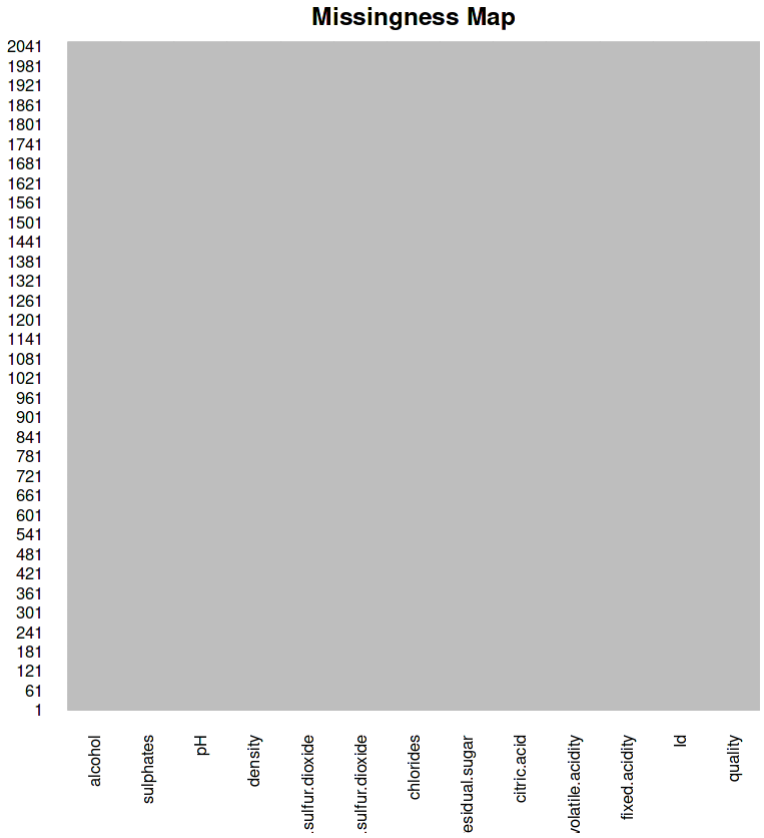
In [3]:

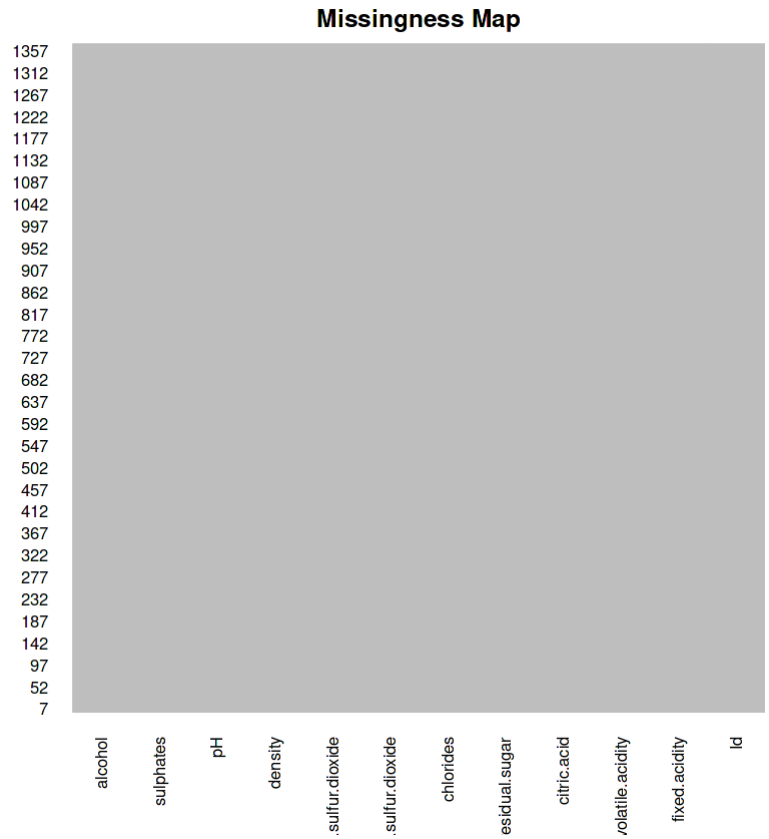
```
#change some variable classes, rearrrange columuns and check for missing values
train$quality <- factor(train$quality)
str(train$quality)
train <- train %>%
    relocate(quality)


#check for duplicates
clean <- unique(train)
dim(train) - dim(clean)
clean <- unique(test)
dim(test) - dim(clean)

#check for missing values
missmap(train, col=c("black", "grey"), legend=FALSE)
missmap(test, col=c("black", "grey"), legend=FALSE)
```

```
 Factor w/ 6 levels "3","4","5","6",..: 4 4 5 3 4 3 4 4 4 3 ...
```

```
0· 0
```

```
0· 0
```



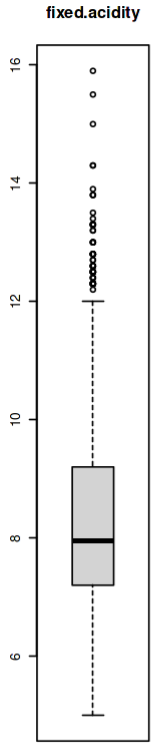


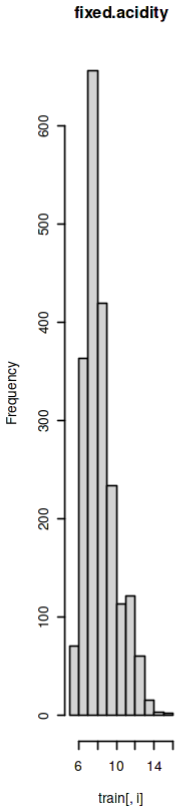
No duplicates. No missing values.

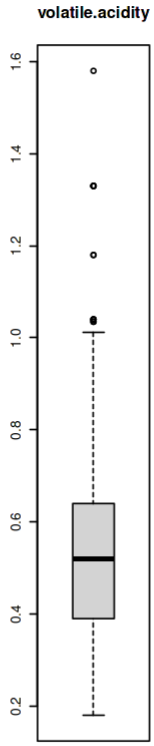
Data Visualisation

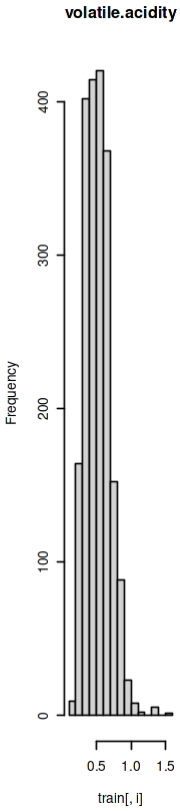
In [4]:

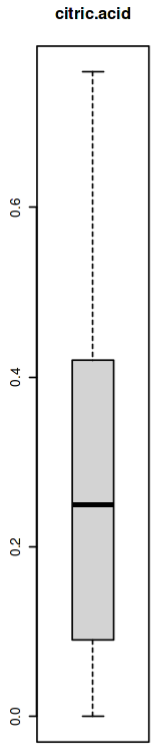
```
#Boxplot and density plot of each attribute
col_first_var <- 3
col_last_var <- dim(train)[2]

nrows<- 1
ncols<- 4

num_plotsets <- ceiling((col_last_var - col_first_var)/(nrows*ncols))
num_plotsets

first <- col_first_var
last <- first + nrows*ncols - 1

for (j in 1:num_plotsets) {

        par(mfrow=c(nrows, ncols))

        for(i in first:last) {
                boxplot(train[,i], main=names(train)[i])
            #densityplot(train[,i], main=names(train)[i])
            hist(train[,i], main=names(train)[i])
        }

        first <- last+1
        last <- first + nrows*ncols - 1
        if (last > dim(train)[2]) { last = dim(train)[2] }
}


# correlation plot
par(mfrow=c(1,1))
correlations <- cor(train[,col_first_var:col_last_var])
corrplot(correlations, method="number", tl.cex = 0.5)
```

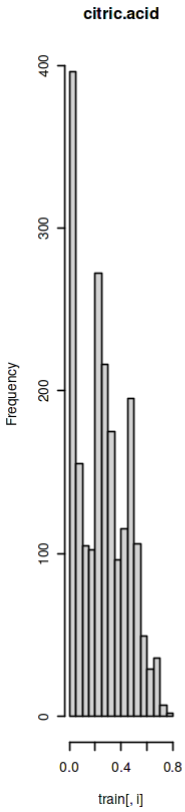

fixed.acidity
6
8
10
12
14
16

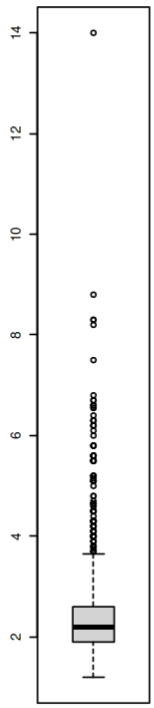

fixed.acidity
Frequency
0
100
200
300
400
500
600
6
10
14
train[, i]

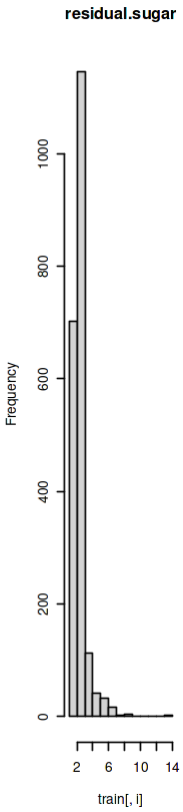

volatile.acidity
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6

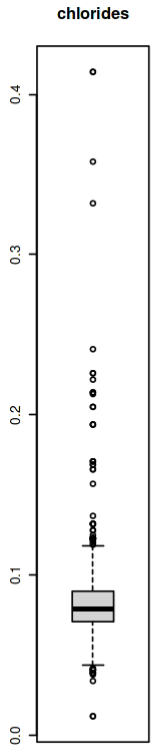

volatile.acidity
Frequency
0
100
200
300
400
0.5
1.0
1.5
train[, i]

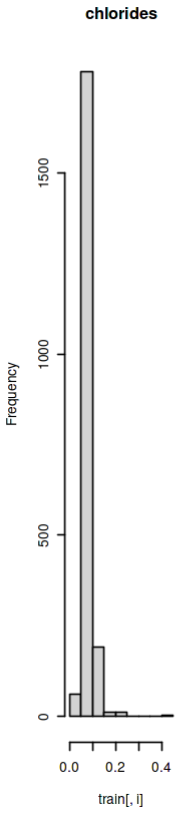

citric.acid
0.0
0.2
0.4
0.6

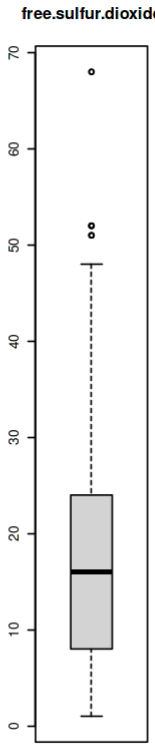

citric.acid
Frequency
0
100
200
300
400
0.0
0.4
0.8
train[, i]

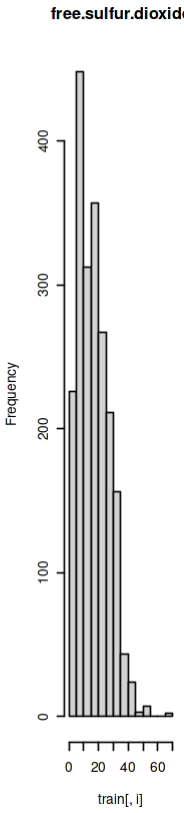

residual.sugar
2
4
6
8
10
12
14

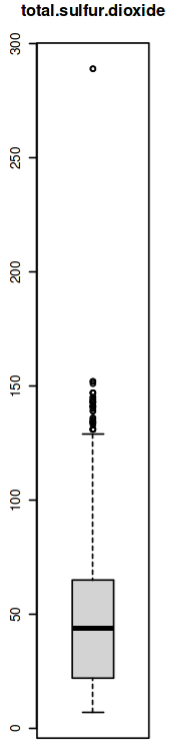

residual.sugar
Frequency
0
200
400
600
800
1000
2
6
10
14
train[, i]

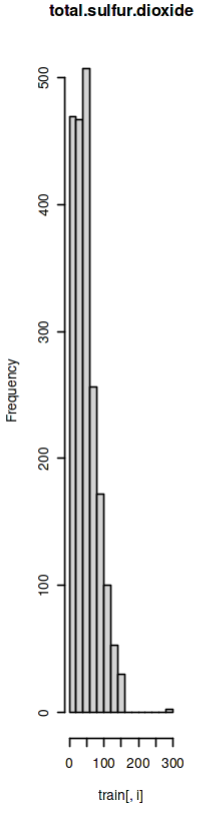

chlorides
0.0
0.1
0.2
0.3
0.4

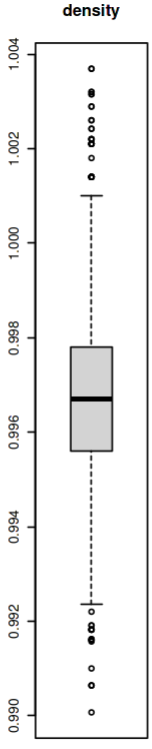

chlorides
Frequency
0
500
1000
1500
0.0
0.2
0.4
train[, i]

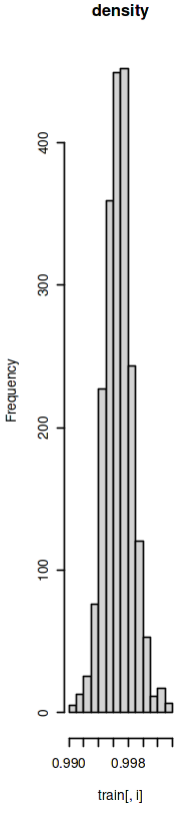

free.sulfur.dioxide
0
10
20
30
40
50
60
70

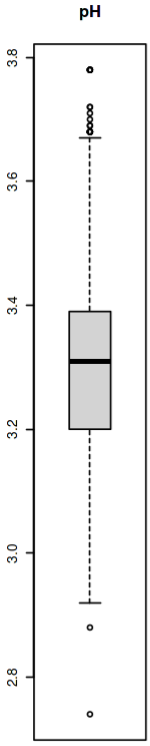

free.sulfur.dioxide
Frequency
0
100
200
300
400
0
20
40
60
train[, i]

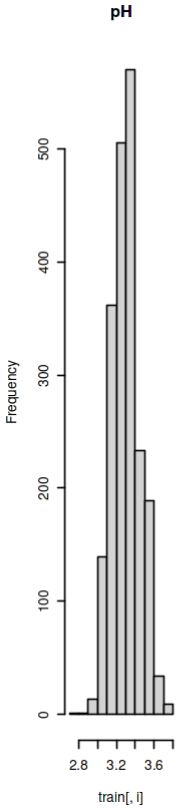

total.sulfur.dioxide
0
50
100
150
200
250
300

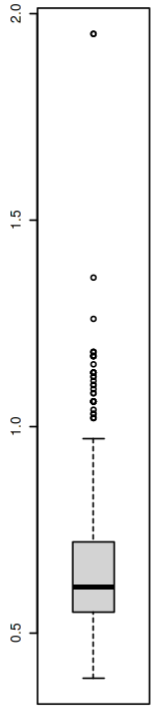

total.sulfur.dioxide
Frequency
0
100
200
300
400
500
0
100
200
300
train[, i]

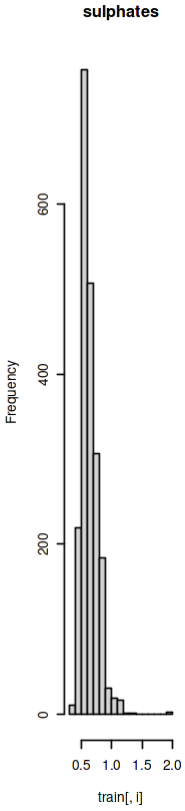

density
0.990
0.992
0.994
0.996
0.998
1.000
1.002
1.004

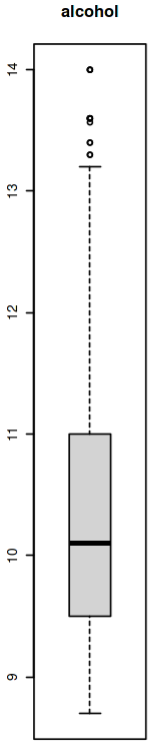

density
Frequency
0
100
200
300
400
0.990
0.998
train[, i]

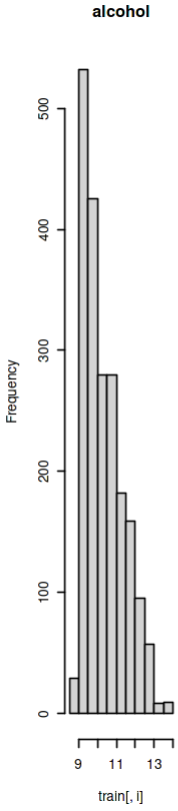

pH
2.8
3.0
3.2
3.4
3.6
3.8

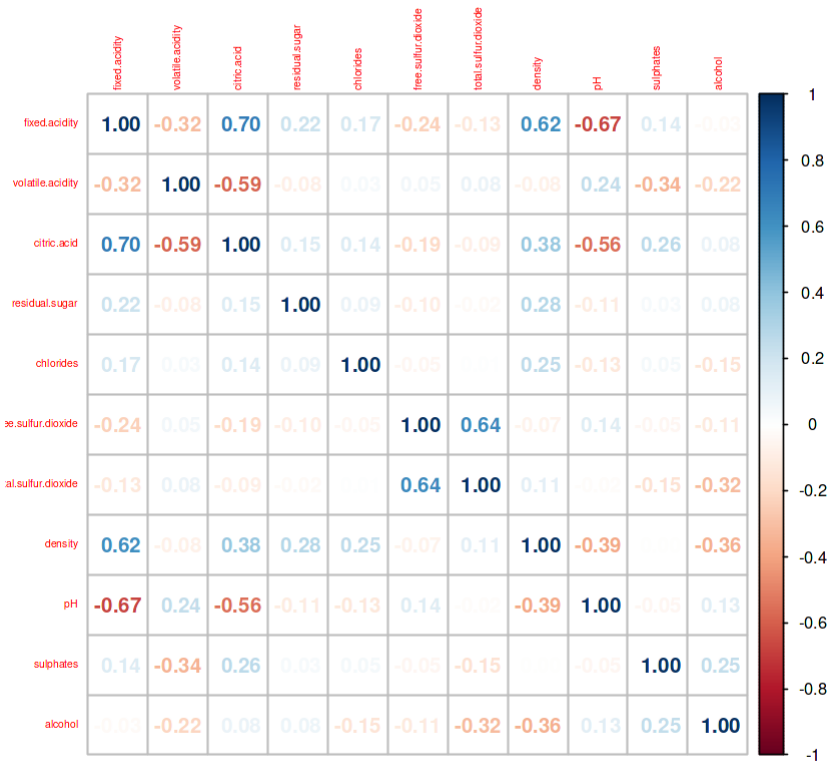

pH
Frequency
0
100
200
300
400
500
2.8
3.2
3.6
train[, i]


sulphates
0.5
1.0
1.5
2.0


sulphates
Frequency
0
200
400
600
0.5
1.0
1.5
2.0
train[, i]


alcohol
9
10
11
12
13
14


alcohol
Frequency
0
100
200
300
400
500
9
11
13
train[, i]



We note quite a lot of outliers on many of the variables.

Let's split off some validation data and build some classification models. We'll compare test and compare the balanced and imbalanced data.

Split Data into Training and Validation Sets

In [5]:

```
#split data into training and validation sets
set.seed(100)
part <- createDataPartition(train$quality, p = 3/4, list = FALSE)
training <- train[part, ]
validate <- train[-part, ]
table(training$quality)
table(validate$quality)
```

```
  3   4   5   6   7   8 
  9  42 630 584 250  30 
```

```
  3   4   5   6   7   8 
  3  13 209 194  83   9 
```

Identify and Remove Outliers

In [6]:

```
out_ind <- 0
missing <- 0
train_no_outliers <- training

for (i in col_first_var:col_last_var) {
        below <- quantile(train_no_outliers[, i], 0.01) #remove bottom 2%
        above <- quantile(train_no_outliers[, i], 0.99) #remove top 2%

        out_ind <- which(train_no_outliers[, i] < below | train_no_outliers[, i]
> above)

        train_no_outliers[out_ind, i] = NA
        missing[i-1] = sum(is.na(train_no_outliers[, i])) #count the amount of N
A's for that variable

        #replace values by KNN imputation
        #imp <- preProcess(train_no_outliers, method = "knnImpute", k = 8)
        #train_no_outliers <- predict(imp, train_no_outliers)

        #replace values with averages by group (courtesy of)[https://stackoverflo
w.com/questions/55345593/impute-missing-data-with-mean-by-group]
        train_no_outliers[i] <- ave(train_no_outliers[, i],
                                   #train_no_outliers$user_name,train_no_outlier
s$classe,
                                   FUN = function(x) ifelse(is.na(x), mean(x,n
a.rm=TRUE), x))
}

print("Amount of values that were identified as outliers, removed, and imputed -
per variable:")
missing
print("Check if any NA's are left after imputation command:")
ouliers_remaining <- sapply(train_no_outliers, function(x) sum(is.na(x)))
ouliers_remaining[ouliers_remaining>0]
```

```
[1] "Amount of values that were identified as outliers, removed, an
d imputed - per variable:"
```

```
0· 31· 23· 8· 16· 30· 16· 20· 31· 29· 28· 15
```

```
[1] "Check if any NA's are left after imputation command:"
```

Model Training - Imbalanced Data

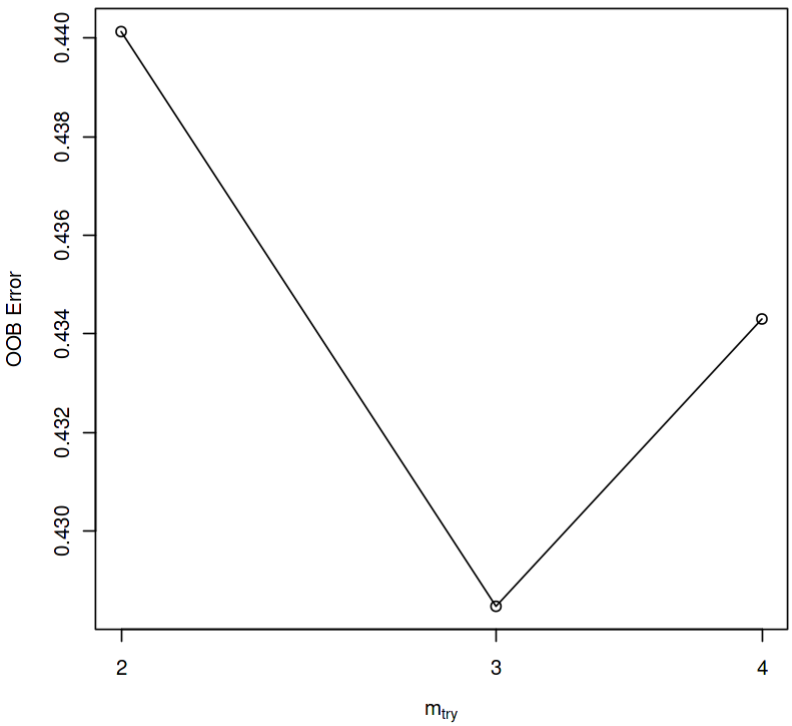
For ideas of available models in CARET, see: [https://topepo.github.io/caret/available-models.html]

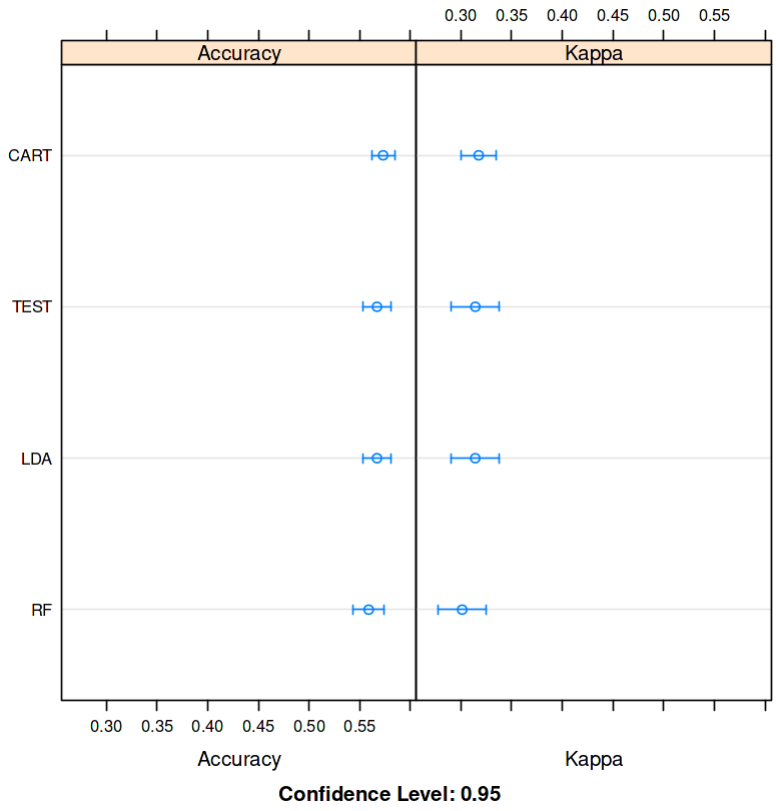
In [7]:

```r
x <- training

#train models
tcont <- trainControl(method = "repeatedcv", number=5, repeats=3)

#CART model
set.seed(100)

grid <- expand.grid(.cp=c(0.01,0.05,0.1))
model_cart <- train(quality ~ ., method="rpart", data = subset(x, select = -c( I
d)), tuneGrid=grid,
                    preProcess = c("center", "scale"), trControl=tcont)

#print(model_cart)

library(randomForest)
#use tuneRF() function from the randomForest package to find the optimal value for
mtry
tune_mtry <- tuneRF(subset(x, select = -c(quality, Id)), x$quality,
                    mtryStart = 2, stepFactor=1.5, ntreeTry = 100, improve = 0.0
1)
#print(tune_mtry)

#set mtry to value corresponding to minimum OOBerror
tunegrid <- expand.grid(.mtry = subset(tune_mtry, tune_mtry[,2] == min(tune_mtry
[,2], na.rm=TRUE))[1])

#manual iterations were performed to find the lowest value for ntree without havin
g a noticeable effect on the resulting accuracy.
set.seed(100)

rf_tuned <- train(quality ~ ., method = "rf", data = subset(x, select = -c(Id)),
ntree = 20,
                  tuneGrid = tunegrid, preProcess = c("center", "scale"), trCont
rol = tcont )
#print(rf_tuned)

# Linear Discriminant Analysis
set.seed(100)

model_lda <- train(quality ~ ., method = "lda", data = subset(x, select = -c(I
d)), metric = "Accuracy",
                   preProcess = c("center", "scale"), trControl = tcont)
```

```r
#Test - nnet,pda
set.seed(100)
start.time4 <- Sys.time()

model_test <- train(quality ~ ., method = "pda", data = subset(x, select = -c(I
d)), metric = "Accuracy",
                    preProcess = c("center", "scale"), trControl = tcont)
end.time4 <- Sys.time()
time.taken4 <- end.time4 - start.time4

time.taken4
#print(model_lda)

#show model comparison based on resampling accuracy results:
results <- resamples(list(CART = model_cart, RF=rf_tuned, LDA = model_lda, TEST
= model_test))
summary(results)
dotplot(results)
```

```
mtry = 2  OOB error = 44.01%
Searching left ...
Searching right ...
mtry = 3        OOB error = 42.85%
0.02647059 0.01
mtry = 4        OOB error = 43.43%
-0.01359517 0.01


Time difference of 2.827233 secs


Call:
summary.resamples(object = results)

Models: CART, RF, LDA, TEST
Number of resamples: 15

Accuracy
          Min.   1st Qu.    Median      Mean   3rd Qu.     Max. N
A's
CART 0.5419355 0.5593511 0.5649351 0.5730283 0.5896649 0.605178
0
RF   0.5096774 0.5444201 0.5548387 0.5588146 0.5672729 0.631068
0
LDA  0.4903226 0.5582530 0.5714286 0.5670076 0.5799718 0.605178
0
TEST 0.4903226 0.5582530 0.5714286 0.5670076 0.5799718 0.605178
0

Kappa
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. N
A's
CART 0.2649367 0.2916379 0.3159072 0.3176116 0.3398006 0.3630911
0
RF   0.2306436 0.2833575 0.2983326 0.3013348 0.3144233 0.4145296
0
LDA  0.1904534 0.2968883 0.3267196 0.3142422 0.3365145 0.3788331
0
TEST 0.1904534 0.2968883 0.3268076 0.3143233 0.3365145 0.3788331
0
```





Confidence Level: 0.95

The results look poor, considering that a 50% accuracy would be just as good as a flip of a coin and we are sitting around 57 with all of the models.

Let's see how predicting on our validation set goes.

Model Validation - Trained on Imbalanced Data

In [8]:

```
set.seed(100)
print("Results on test set prediction for CART: single decision tree")
predQ1 <- predict(object = model_cart, newdata = subset(validate, select = -c(qu
ality)))
result1 <- confusionMatrix(validate$quality, predQ1)
result1$table

set.seed(100)
print("Results on test set prediction for rf_tuned: tuned random forest with out
liers included")
predQ2 <- predict(object = rf_tuned, newdata = subset(validate, select = -c(qual
ity)))
result2 <- confusionMatrix(validate$quality, predQ2)
result2$table

set.seed(100)
print("Results on test set prediction for model_lda: linear discriminant analyis
model")
predQ3 <- predict(object = model_lda, newdata = subset(validate, select = -c(qua
lity)))
result3 <- confusionMatrix(validate$quality, predQ3)
result3$table

set.seed(100)
print("Results on test set prediction for model_lda: TEST model")
predQ4 <- predict(object = model_test, newdata = subset(validate, select = -c(qu
ality)))
result4 <- confusionMatrix(validate$quality, predQ4)
result4$table


compare_validation_result <- data.frame(Model = c("CART", "RF", "LDA", "TEST"),
                                        Accuracy = c(result1$overall[1], result2
$overall[1],
                                                     result3$overall[1], result4
$overall[1] ))
print("Out of sample accuracies compared:")
compare_validation_result
```

```
[1] "Results on test set prediction for CART: single decision tree"


          Reference
Prediction   3   4   5   6   7   8
         3   0   0   2   1   0   0
         4   0   0  12   1   0   0
         5   0   0 158  51   0   0
         6   0   0  64 118  12   0
         7   0   0  14  57  12   0
         8   0   0   2   4   3   0


[1] "Results on test set prediction for rf_tuned: tuned random fore
st with outliers included"


          Reference
Prediction   3   4   5   6   7   8
         3   0   0   3   0   0   0
         4   0   0  11   2   0   0
         5   0   1 148  54   6   0
         6   0   0  56 113  25   0
         7   0   0  12  44  26   1
         8   0   0   2   5   2   0


[1] "Results on test set prediction for model_lda: linear discrimin
ant analyis model"


          Reference
Prediction   3   4   5   6   7   8
         3   0   0   3   0   0   0
         4   0   0   9   4   0   0
         5   2   0 152  49   6   0
         6   0   0  57 120  17   0
         7   0   0   9  48  25   1
         8   0   0   1   6   2   0


[1] "Results on test set prediction for model_lda: TEST model"
```

```
          Reference
Prediction   3   4   5   6   7   8
         3   0   0   3   0   0   0
         4   0   0   9   4   0   0
         5   2   0 152  49   6   0
         6   0   0  57 119  18   0
         7   0   0   9  48  25   1
         8   0   0   1   6   2   0
```

```
[1] "Out of sample accuracies compared:"
```

A data.frame: 4 × 2

Model	Accuracy
<chr>	<dbl>
CART	0.5636008
RF	0.5616438
LDA	0.5812133
TEST	0.5792564

The results are equally unimpressive. Let's look at the variable importance.

In [9]:

```
i_scores1 <- varImp(rf_tuned, scale = TRUE)$importance %>%
        as.data.frame() %>%
        arrange(desc(Overall))
print("Variable Imporantance for RF")
i_scores1

i_scores2 <- varImp(model_cart)$importance %>%
        as.data.frame() %>%
        arrange(desc(Overall))
print("Variable Imporantance for CART model")
i_scores2
```

```
[1] "Variable Imporantance for RF"
```

A data.frame: 11 × 1

	Overall
	<dbl>
sulphates	100.00000000
alcohol	83.72262306
total.sulfur.dioxide	39.23826040
density	33.15882276
volatile.acidity	17.79559890
fixed.acidity	16.84326368
chlorides	10.70318153
citric.acid	6.65155401
pH	4.98328101
residual.sugar	0.05447381
free.sulfur.dioxide	0.00000000

```
[1] "Variable Imporantance for CART model"
```

A data.frame: 11 × 1

	Overall
	<dbl>
sulphates	100.0000000
alcohol	78.4020582
total.sulfur.dioxide	43.6273108
volatile.acidity	39.0899918
citric.acid	26.0223489
fixed.acidity	11.5924071
residual.sugar	4.9987303
density	1.8073669
free.sulfur.dioxide	1.0284015
pH	0.6706679
chlorides	0.0000000

Model Training on Balanced Data

Balancing Data for multi-class problem

We'll try to rebalance the data with SMOTE.

(I found help here)[https://stackoverflow.com/questions/48215299/any-package-in-r-which-can-do-multi-class-oversampling-under-sampling-both-and]

Create a second training set with balanced data:

In [10]:

```
#rebalance data - use synthetic balancing from te UBL package

#?AdasynClassif
#?RandOverClassif
set.seed(100)
training.balanced <- AdasynClassif(quality ~ ., dat = training)
#training.balanced <- RandOverClassif(quality ~ ., dat = training) #This one does even worse
table(training.balanced$quality)
```

```
  3   4   5   6   7   8 
629 628 630 584 674 632 
```

In [11]:

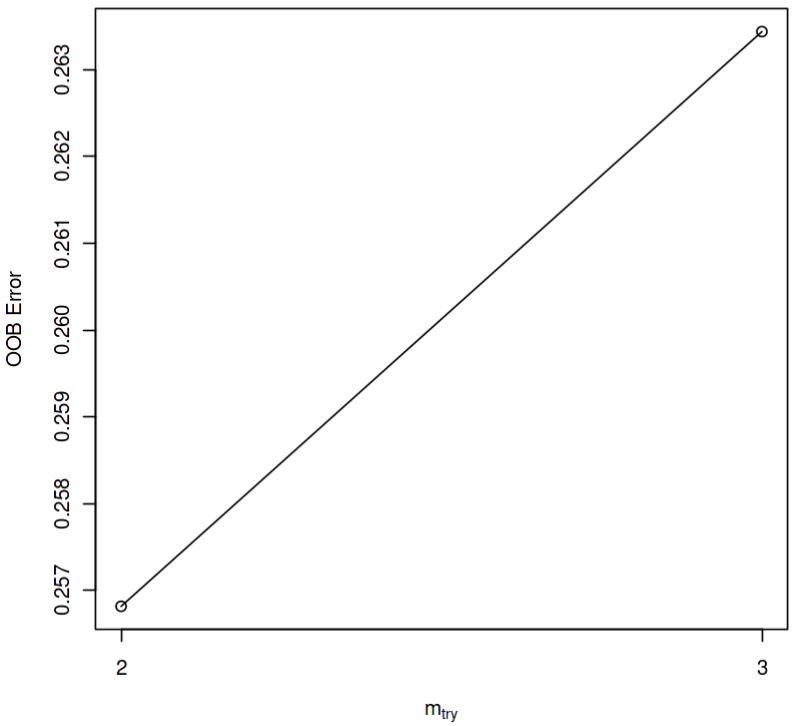
```r
#train models
tcont <- trainControl(method = "repeatedcv", number=5, repeats=3)

#CART model
set.seed(100)
start.time1 <- Sys.time()
grid <- expand.grid(.cp=c(0.01,0.05,0.1))
model_cart.balanced <- train(quality ~ ., method="rpart", data = training.balanc
ed, tuneGrid=grid, trControl=tcont)
end.time1 <- Sys.time()
time.taken1 <- end.time1 - start.time1

time.taken1
print(model_cart.balanced)

library(randomForest)
#use tuneRF() function from the randomForest package to find the optimal value for
mtry
tune_mtry <- tuneRF(subset(training.balanced, select = -quality), training.balan
ced$quality,
                    mtryStart = 2, stepFactor=1.5, ntreeTry = 100, improve = 0.0
1)
print(tune_mtry)

#set mtry to value corresponding to minimum OOBerror
tunegrid <- expand.grid(.mtry = subset(tune_mtry, tune_mtry[,2] == min(tune_mtry
[,2], na.rm=TRUE))[1])

#manual iterations were performed to find the lowest value for ntree without havin
g a noticeable effect on the resulting accuracy.
set.seed(100)
start.time2 <- Sys.time()

rf_tuned.balanced <- train(quality ~ ., method = "rf", data = training.balanced,
ntree = 100,
                   tuneGrid = tunegrid, preProcess = c("center", "scale"),trContr
ol = tcont )

end.time2 <- Sys.time()
time.taken2 <- end.time2 - start.time2

time.taken2
print(rf_tuned.balanced)
```

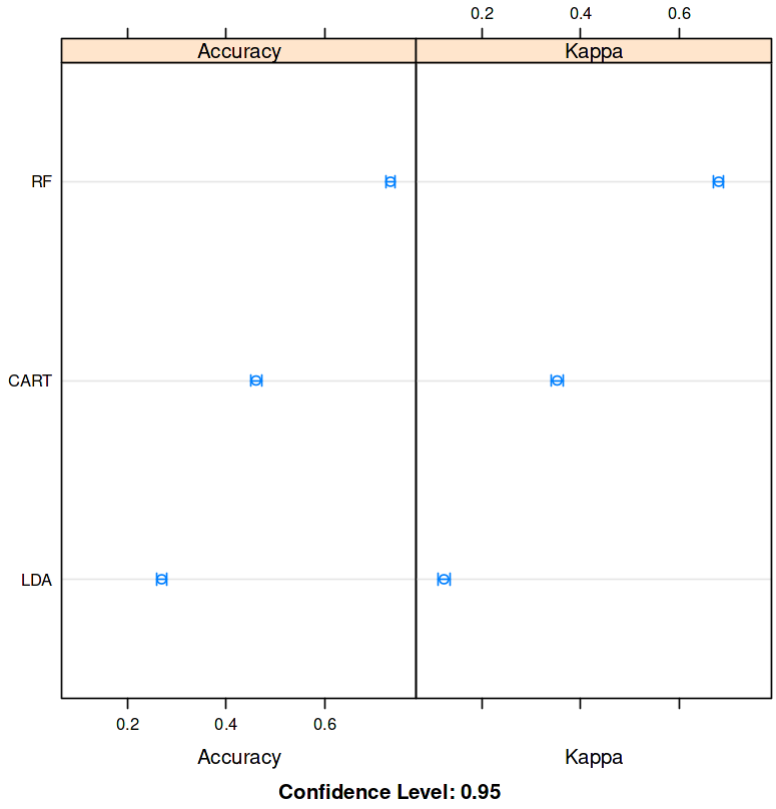
```r
# Linear Discriminant Analysis
set.seed(100)
start.time3 <- Sys.time()

model_lda.balanced <- train(quality ~ ., method = "lda", data = training.balance
d, metric="Accuracy", trControl=tcont)

end.time3 <- Sys.time()
time.taken3 <- end.time3 - start.time3

time.taken3
print(model_lda.balanced)


#show model comparison based on resampling accuracy results:
results.balanced <- resamples(list(CART = model_cart.balanced, RF=rf_tuned.balan
ced, LDA = model_lda.balanced))
summary(results.balanced)
dotplot(results.balanced)
```

```
Time difference of 1.917042 secs


CART

3777 samples
  12 predictor
   6 classes: '3', '4', '5', '6', '7', '8'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 3021, 3021, 3023, 3022, 3021, 3022, ...
Resampling results across tuning parameters:

  cp    Accuracy   Kappa
  0.01  0.4609456  0.3525591
  0.05  0.3636929  0.2345431
  0.10  0.3108272  0.1729886

Accuracy was used to select the optimal model using the largest val
ue.
The final value used for the model was cp = 0.01.
mtry = 2  OOB error = 25.68%
Searching left ...
Searching right ...
mtry = 3        OOB error = 26.34%
-0.0257732 0.01
      mtry  OOBError
2.OOB    2 0.2568176
3.OOB    3 0.2634366


Time difference of 10.7838 secs
```

```
Random Forest

3777 samples
  12 predictor
   6 classes: '3', '4', '5', '6', '7', '8'

Pre-processing: centered (12), scaled (12)
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 3021, 3021, 3023, 3022, 3021, 3022, ...
Resampling results:

  Accuracy   Kappa
  0.7332908  0.6799054

Tuning parameter 'mtry' was held constant at a value of 2


Time difference of 0.8569205 secs


Linear Discriminant Analysis

3777 samples
  12 predictor
   6 classes: '3', '4', '5', '6', '7', '8'

No pre-processing
Resampling: Cross-Validated (5 fold, repeated 3 times)
Summary of sample sizes: 3021, 3021, 3023, 3022, 3021, 3022, ...
Resampling results:

  Accuracy   Kappa
  0.2696162  0.1231001
```

```
Call:
summary.resamples(object = results.balanced)

Models: CART, RF, LDA
Number of resamples: 15

Accuracy
          Min.   1st Qu.    Median      Mean   3rd Qu.      Max. N
A's
CART 0.4291391 0.4510582 0.4668435 0.4609456 0.4765050 0.4854497
0
RF   0.7086093 0.7225166 0.7320955 0.7332908 0.7456967 0.7582563
0
LDA  0.2370861 0.2624256 0.2701987 0.2696162 0.2794672 0.2993377
0

Kappa
           Min.   1st Qu.    Median      Mean   3rd Qu.      Max. N
A's
CART 0.31671902 0.3397053 0.3593802 0.3525591 0.3708770 0.3835531
0
RF   0.65040558 0.6669986 0.6784349 0.6799054 0.6948283 0.7098487
0
LDA  0.08288379 0.1140860 0.1235856 0.1231001 0.1353144 0.1586122
0
```





Confidence Level: 0.95

The results from the cross validation suggest that the random forest model gained a significant jump in accuracy from 56% to just over 69%. The LDA model, by contrast, lost about 20% of accuracy. Let's have a look at predictions on the validation set.

Model Validation after training on Balanced Data

In [12]:

```
set.seed(100)
print("Results on test set prediction for CART: single decision tree")
predQ1.balanced <- predict(object = model_cart.balanced, newdata = subset(valida
te, select = -c(quality)))
result1.balanced <- confusionMatrix(validate$quality, predQ1.balanced)
result1.balanced$table

set.seed(100)
print("Results on test set prediction for rf_tuned: tuned random forest with out
liers included")
predQ2.balanced <- predict(object = rf_tuned.balanced, newdata = subset(validat
e, select = -c(quality)))
result2.balanced <- confusionMatrix(validate$quality, predQ2.balanced)
result2.balanced$table

set.seed(100)
print("Results on test set prediction for model_lda: linear discriminant analyis
model")
predQ3.balanced <- predict(object = model_lda.balanced, newdata = subset(validat
e, select = -c(quality)))
result3.balanced <- confusionMatrix(validate$quality, predQ3.balanced)
result3.balanced$table

compare_validation_result.balanced <- data.frame(Model = c("CART", "RF", "LDA"),
                                        Accuracy = c(result1.balanced$overall
[1], result2.balanced$overall[1],
                                                     result3.balanced$overall
[1]))
print("Out of sample accuracies compared:")
compare_validation_result.balanced
```

```
[1] "Results on test set prediction for CART: single decision tree"


          Reference
Prediction   3   4   5   6   7   8
         3   1   0   1   0   1   0
         4   1   0   6   4   2   0
         5   4   0 123  49  32   1
         6   0   0  28  91  73   2
         7   0   0   4  25  53   1
         8   0   0   0   3   6   0


[1] "Results on test set prediction for rf_tuned: tuned random fore
st with outliers included"


          Reference
Prediction   3   4   5   6   7   8
         3   0   0   2   1   0   0
         4   1   0  10   2   0   0
         5   4   0 144  54   5   2
         6   1   0  56 107  29   1
         7   0   0  14  34  34   1
         8   0   0   1   5   3   0


[1] "Results on test set prediction for model_lda: linear discrimin
ant analyis model"


          Reference
Prediction  3  4  5  6  7  8
         3  0  0  3  0  0  0
         4  2  3  5  1  2  0
         5 35 21 94 47 12  0
         6 47 18 62 53 14  0
         7 18  3 25 26 11  0
         8  3  2  1  0  3  0


[1] "Out of sample accuracies compared:"
```

A data.frame: 3 × 2

Model	Accuracy
<chr>	<dbl>
CART	0.5244618
RF	0.5577299
LDA	0.3150685

Conclusion on Balancing Data

In this case, the out of sample accuracy is worse on all counts compared to using the imbalanced data.

Conclusion on Removing Outliers

We tried removing outliers ranging from 10% to 1% on the top and bottom end of the data and imputing by KNN or means and it didn't do much good!

Conclusion on model selection

The LDA proved to be the best model to use and the data will be output as a prediction on the test set

In [13]:

```
print("Results on test set prediction for model_lda: linear discriminant analyis model")
pred_final <- predict(object = model_lda, newdata = test)
df_final <- data.frame(Id = test$Id, quality = pred_final)
head(df_final, 10)
write.csv(df_final, "/kaggle/working/submission.csv", row.names=FALSE)
```

```
[1] "Results on test set prediction for model_lda: linear discrimin
ant analyis model"
```

A data.frame: 10 × 2

	Id	quality
	<int>	<fct>
1	2056	5
2	2057	5
3	2058	5
4	2059	6
5	2060	6
6	2061	6
7	2062	6
8	2063	6
9	2064	7
10	2065	5