HarvardX: PH125.9x Data Science Beer Recipe Project

Tran Phu Hoa

May 7, 2019

Contents

1 Introduction

Beer seem to be one of the most favorite drink of all time. Recently, there are many different types of beer occurred in the Vietnam market, and we seem to be interested in challenging each other to correctly name those beers without seeing their labels. In fact, the project is conducted because of the curiosity about beer and the urge to find answer for the challenge.

The project arms to predict the 10 most popular types of beer using the "Brewer's Friend Beer Recipes" dataset in Kaggle. It is a dataset of 75,000 homemade brewed beers with over 176 different styles. Beer records were reported individually by each user, and those beers were classified according to one of the 176 different styles.

The report is expected to follow part of the structure which is recommended by Dr. Roger D.Peng in his book called "Report Writing for Data Science in R". Thus, those steps are.

1. Defining the question
2. Obtaining the data
3. Cleaning the data
4. Exploratory data analysis
5. Statistical prediction/modelling
6. Interpretation of results
7. Conclusion

2 Defining the question

There are two main questions the project expects to answer. The first one is what model will be the good one to produce the highest accuracy. The second one is what is the most important characteristic of beers that help us in choicing the right beer style.

3 Data Cleaning

3.1 Remove and free up working space

```
# Remove and free up working space
rm(list = ls(all.names = TRUE))
gc()
```

```
##           used (Mb) gc trigger (Mb) max used (Mb)
## Ncells  520927 27.9    1183787 63.3   609151 32.6
## Vcells 1024494  7.9    8388608 64.0  1597850 12.2
```

3.2 Load necessary packages for exploration and wrangling purposes

```
# To import, tidy, wrangle, visualize, model and communicate the data
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.r-project.org")

# To create summary statistics of variables
if(!require(skimr)) install.packages("skimr", repos = "http://cran.r-project.org")

# To create grid display for graphs
if(!require(gridExtra)) install.packages("gridExtra", repos = "http://cran.r-project.org")
```

3.3 Load the raw dataset downloaded on Kaggle website

```
# Load the raw dataset
recipedata <- read_csv("dataset/recipeData.csv")
```

3.4 Explore the raw dataset

3.4.1 Check the names of all columns in the raw dataset

Those names seems to be easily read and written while we recall them in different steps of the project, except "Size(L)". In fact, "Size(L)" will be renamed in the next step.

```
# Names of variables in the raw dataset
names(recipedata)
```

```
##  [1] "BeerID"        "Name"          "URL"           "Style"
##  [5] "StyleID"       "Size(L)"       "OG"            "FG"
##  [9] "ABV"           "IBU"           "Color"         "BoilSize"
## [13] "BoilTime"      "BoilGravity"   "Efficiency"    "MashThickness"
## [17] "SugarScale"    "BrewMethod"    "PitchRate"     "PrimaryTemp"
## [21] "PrimingMethod" "PrimingAmount" "UserId"
```

3.4.2 Define the variables

- The "names" function had shown that the dataset had 23 variables, and their definitions are described below.

1. BeerID : Record ID of each user
2. Name: A beer name made by an user
3. URL: Location of recipe webpage
4. Style: Beer Style
5. StyleID: Numeric ID of a beer style
6. Size(L): the batch size of the listed recipe in liter (L)
7. OG: the original gravity of wort before fermentation in Degree Plato (P) or Specific Gravity (SG)
8. FG: the final gravity of wort after fermentation in Degree Plato (P) or Specific Gravity (SG)
9. ABV: Alcohol By Volume in percentage (%)
10. IBU: International Bittering Units (IBU)
11. Color: Standard Reference Method (SRM)
12. BoilSize: Fluid at beginning of boil in liter (L)
13. BoilTime: time to boil the wort in minutes (min)
14. BoilGravity: the gravity of wort before the boil in Degree Plato (P) or Specific Gravity (SG)
15. Efficiency: Beer mash extraction efficiency in percentage (%)
16. MashThickness: Amount of water per pound of grain in liters per kilogram (L/kg)
17. SugarScale: Scale to determine the concentration of dissolved solids in wort support in both Degree Plato (P) and Specific Gravity (SG)
18. BrewMethod: Various techniques for brewing
19. PitchRate: Yeast added to the fermentor per gravity unit (million cells/ml/P)
20. PrimaryTemp: Temperature at the fermenting stage (C)
21. PrimingMethod: to add other ingredients
22. PrimingAmount: Amount of priming ingredients used (g)
23. UserId: ID of an user

- The definition of those variables are supported by the owner of the dataset called "Brewer's Friend Beer Recipes", and other websites which are listed below.

1. Frequently Asked Questions in Brewer' Friend Beer Recipes
2. Box Brew Kits
3. Craft Beer and Brewing Magazine

3.4.3 Look for missing values in the raw dataset

Using "glimpse" function would provide general information of the dataset, especially what we expect about the types of variables and how missing values were represented in the dataset.

```r
# Have a look at the raw dataset
glimpse(recipedata)
```

```
## Observations: 73,861
## Variables: 23
## $ BeerID        <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1...
## $ Name          <chr> "Vanilla Cream Ale", "Southern Tier Pumking clon...
## $ URL           <chr> "/homebrew/recipe/view/1633/vanilla-cream-ale", ...
## $ Style         <chr> "Cream Ale", "Holiday/Winter Special Spiced Beer...
## $ StyleID       <dbl> 45, 85, 7, 7, 20, 10, 86, 45, 129, 86, 7, 7, 7, ...
## $ `Size(L)`     <dbl> 21.77, 20.82, 18.93, 22.71, 50.00, 24.61, 22.71,...
## $ OG            <dbl> 1.055, 1.083, 1.063, 1.061, 1.060, 1.055, 1.072,...
## $ FG            <dbl> 1.013, 1.021, 1.018, 1.017, 1.010, 1.013, 1.018,...
## $ ABV           <dbl> 5.48, 8.16, 5.91, 5.80, 6.48, 5.58, 7.09, 5.36, ...
## $ IBU           <dbl> 17.65, 60.65, 59.25, 54.48, 17.84, 40.12, 268.71...
## $ Color         <dbl> 4.83, 15.64, 8.98, 8.50, 4.57, 8.00, 6.33, 5.94,...
## $ BoilSize      <dbl> 28.39, 24.61, 22.71, 26.50, 60.00, 29.34, 30.28,...
## $ BoilTime      <dbl> 75, 60, 60, 60, 90, 70, 90, 75, 75, 60, 90, 90, ...
## $ BoilGravity   <chr> "1.038", "1.07", "N/A", "N/A", "1.05", "1.047", ...
## $ Efficiency    <dbl> 70, 70, 70, 70, 72, 79, 75, 70, 73, 70, 74, 70, ...
## $ MashThickness <chr> "N/A", "N/A", "N/A", "N/A", "N/A", "N/A", "N/A",...
## $ SugarScale    <chr> "Specific Gravity", "Specific Gravity", "Specifi...
## $ BrewMethod    <chr> "All Grain", "All Grain", "extract", "All Grain"...
## $ PitchRate     <chr> "N/A", "N/A", "N/A", "N/A", "N/A", "1", "N/A", "...
## $ PrimaryTemp   <chr> "17.78", "N/A", "N/A", "N/A", "19", "N/A", "N/A"...
## $ PrimingMethod <chr> "corn sugar", "N/A", "N/A", "N/A", "Sukkerlake",...
## $ PrimingAmount <chr> "4.5 oz", "N/A", "N/A", "N/A", "6-7 g sukker/l",...
## $ UserId        <dbl> 116, 955, NA, NA, 18325, 5889, 1051, 116, 116, N...
```

3.4.4 Re-load the raw dataset with descriptions

- Add "N/A" to the description of missing values in the function "read_csv"
- Specify variable types

```r
# Load dataset with an update in description
recipedata <- read_csv("dataset/recipeData.csv",
                       na = c("", "NA", "N/A"),
                       col_types = cols(Style = col_factor(levels = NULL),
                                        `Size(L)` = col_number(),
                                        OG = col_number(),
                                        FG = col_number(),
                                        ABV = col_number(),
                                        IBU = col_number(),
                                        Color = col_number(),
                                        BoilSize = col_number(),
                                        BoilTime = col_number(),
                                        BoilGravity = col_number(),
                                        Efficiency = col_number(),
                                        MashThickness = col_number(),
                                        SugarScale = col_factor(levels = NULL),
                                        BrewMethod = col_factor(levels = NULL),
                                        PitchRate = col_number(),
                                        PrimaryTemp = col_number(),
                                        PrimingMethod = col_factor(levels = NULL),
                                        PrimingAmount = col_factor(levels = NULL),
```

```
                                        UserId = col_number()))
```

- In addition, variable "`Size(L)`" was named in an odd way which might create confusion in the modelling steps, so it will be changed to "sizeL".

```
# Rename Size(L) to sizeL
recipedata <- recipedata %>% rename(sizeL = `Size(L)`)
```

3.4.5 Summary statistics of the updated dataset

```
# Summarize the dataset
skim_with(numeric = list(hist = NULL))
skim(recipedata)
```

```
## Skim summary statistics
##  n obs: 73861
##  n variables: 23
##
## -- Variable type:character ------------------------------------------------------------------------
##  variable missing complete     n min max empty n_unique
##      Name       1    73860 73861   1  83     0    59140
##       URL       0    73861 73861  26 118     0    73861
##
## -- Variable type:factor ---------------------------------------------------------------------------
##       variable missing complete     n n_unique
##    BrewMethod       0    73861 73861        4
##  PrimingAmount   69085     4776 73861     1892
##  PrimingMethod   67095     6766 73861      871
##          Style     596    73265 73861      175
##    SugarScale       0    73861 73861        2
##                                top_counts ordered
##  All: 49692, BIA: 12016, ext: 8626, Par: 3527   FALSE
##       NA: 69085, 5 o: 205, 3/4: 110, 4 o: 106   FALSE
##       NA: 67095, Cor: 717, Dex: 503, cor: 360   FALSE
##   Ame: 11940, Ame: 7581, Sai: 2617, Ame: 2277   FALSE
##                 Spe: 71959, Pla: 1902, NA: 0   FALSE
##
## -- Variable type:numeric --------------------------------------------------------------------------
##       variable missing complete     n     mean       sd      p0      p25
##            ABV       0    73861 73861     6.14     1.88   0         5.08
##         BeerID       0    73861 73861 36931    21321.98   1     18466
##    BoilGravity    2990    70871 73861     1.35     1.93   0         1.04
##       BoilSize       0    73861 73861    49.72   193.25   1        20.82
##       BoilTime       0    73861 73861    65.07    15.02   0        60
##          Color       0    73861 73861    13.4     11.94   0         5.17
##     Efficiency       0    73861 73861    66.35    14.09   0        65
##             FG       0    73861 73861     1.08     0.43  -0.003     1.01
##            IBU       0    73861 73861    44.28    42.95   0        23.37
##  MashThickness   29864    43997 73861     2.13     1.68   0         1.5
##             OG       0    73861 73861     1.41     2.2    1         1.05
##      PitchRate   39252    34609 73861     0.75     0.39   0         0.35
##    PrimaryTemp   22662    51199 73861    19.18     4.22 -17.78     18
##          sizeL       0    73861 73861    43.93   180.37   1        18.93
##        StyleID       0    73861 73861    60.18    56.81   1        10
```

```
##          UserId   50490    23371 73861 43078.07 27734.25  49     20984
##       p50      p75      p100
##      5.79     6.83     54.72
##  36931    55396     73861
##      1.05     1.06     52.6
##     27.44    30       9700
##     60       60        240
##      8.44    16.79    186
##     70       75       100
##      1.01     1.02     23.42
##     35.77    56.38   3409.3
##      1.5      3       100
##      1.06     1.07     34.03
##      0.75     1         2
##     20       20       114
##     20.82    23.66   9200
##     35      111       176
##  42897    57841    134362
```

3.4.6 Standardizing the measurement unit

- The site Brewer's Friend allowed users to fill their beer's recipes in to different scale "Degree Plato" and "Specific Gravity", and it could be seen in "SugarScale" variable. The project will convert all values recored in "Degree Plato" to "Specific Gravity".
- Convert all Plato units to specific gravity (SG) The function could be found in the dataset owner website

SG = 1+ (plato / (258.6 – ((plato/258.2) x 227.1)))

```
# Set a function to convert Plato to Specific Gravity
plato_to_sg <- function(x) {
  1 + (x / (258.6 - ((x/258.2) * 227.1)))
  }

# Nest the dataset by SugarScale
recipedata_nest_sugarscale <- recipedata %>%
  nest(-SugarScale)

# Apply the plato_to_sg function to the nested dataset
recipedata_nest_sugarscale$data[[2]] <- recipedata_nest_sugarscale$data[[2]] %>%
  mutate_at(vars(OG, FG, BoilGravity), plato_to_sg)

# Unnest the datasets
recipedata_sg_scale <- recipedata_nest_sugarscale %>%
  unnest(data) %>%
  mutate(SugarScale = as.factor("Specific_Gravity"))
```

- "SugarScale" indicated what measurement was used by users, so it should be dropped after all variables were calculated in "Specific Gravity".

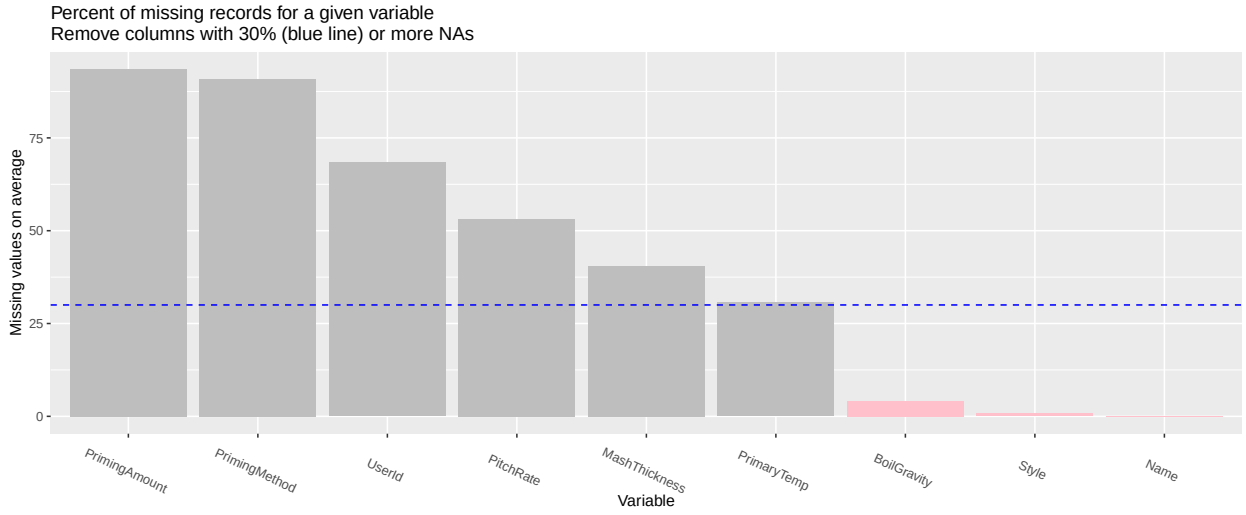
```
# Drop SugarScale
recipedata_sg_scale <- recipedata_sg_scale %>% select(- SugarScale)
```

3.4.7 Handling missing values

3.4.7.1 Visualize missing values of variables

There are six varibles that have more than thirty percent of missing values, such as "PrimingAmount", "PrimingMethod", "UserId", "PitchRate", "MashThickness" and "PrimaryTemp." Those variables that got the missing rate above thirty percent will be dropped out, and the others will be left out for further consideration. In fact, there is special treatment for "Style", "BoilGravity" and "Name" in the following section.

```
# Create a function to calculate proportion of missing values
mean_missingvalue_func <- function(x) {
  mean(is.na(x)) * 100
}

# Show variables with NA values
mean_na <- recipedata_sg_scale %>%
  summarise_all(mean_missingvalue_func) %>%
  gather("Variable", "NA_average") %>%
  filter(NA_average > 0) # Return columns with missing values

# Plot a barchart for the proportion of missing records in each variable
mean_na %>%
  mutate(na_vars = if_else(NA_average > 30, "gray", "pink")) %>%
  ggplot(aes(x = reorder(Variable, -NA_average), y = NA_average, fill = I(na_vars))) +
  geom_bar(stat = "identity") +
  geom_hline(yintercept = 30, color = "blue", linetype = 2) +
  theme(axis.text.x = element_text(angle = -25)) +
  labs(title =
         "Percent of missing records for a given variable\nRemove columns with 30% (blue line) or more
       x = "Variable",
       y = "Missing values on average")
```



3.4.7.2 Drop those variables that had more than 30 percent of missing values

```
# Drop incomplete variables
complete_var_recipedata_sg_scale <- mean_na$Variable[which(mean_na$NA_average > 30)]

recipedata_sg_scale_dropNAabove30 <- recipedata_sg_scale %>%
  select(-c(which(colnames(.) %in% complete_var_recipedata_sg_scale)))
```

3.4.7.3 Missing values in dependent variable "Style"

The variable "Style" had 596 missing values, while the "StyleID" did not have any missing. It might be caused by the typo of users when they added the beer recipe, so the project will fix this problem by matching the StyleID with missing value in Style.

Firstly, having a glimpse in the missing one will help us to choose an appropriate solution to handle this problem. In fact, the result shows that there is only one Style ID (111) represented for all 595 missings, and they can be replaced by the correct style using "styleData.csv". StyleID (111) standed for "N/A", so the project will drop those rows.

```
# Matching missing Style with Style ID
recipedata_sg_scale_dropNAabove30 %>% filter(is.na(Style)) %>% count(StyleID)
```

```
## # A tibble: 1 x 2
##   StyleID     n
##     <dbl> <int>
## 1     111   596
```

The missing values in Style will be dropped out.

```
# Drop rows with missing values in Style
recipedata_sg_scale_dropNAabove30_and_dropNAstyle <-
  recipedata_sg_scale_dropNAabove30 %>%
  filter(Style != is.na(Style))

# Double check the variable Style
skim(recipedata_sg_scale_dropNAabove30_and_dropNAstyle, Style)
```

```
## Skim summary statistics
##  n obs: 73265
##  n variables: 16
##
## -- Variable type:factor ---------------------------------------------------------------------------
##  variable missing complete     n n_unique
##     Style       0    73265 73265      175
##                                   top_counts ordered
##  Ame: 11940, Ame: 7581, Sai: 2617, Ame: 2277   FALSE
```

3.4.7.4 Missing values in "BoilGravity"

Missing values in a numerial variable are commonly replaced by sample mean or sample median. It is believed that the distribution of "BoilGravity" will help to make a choice between those two numbers.

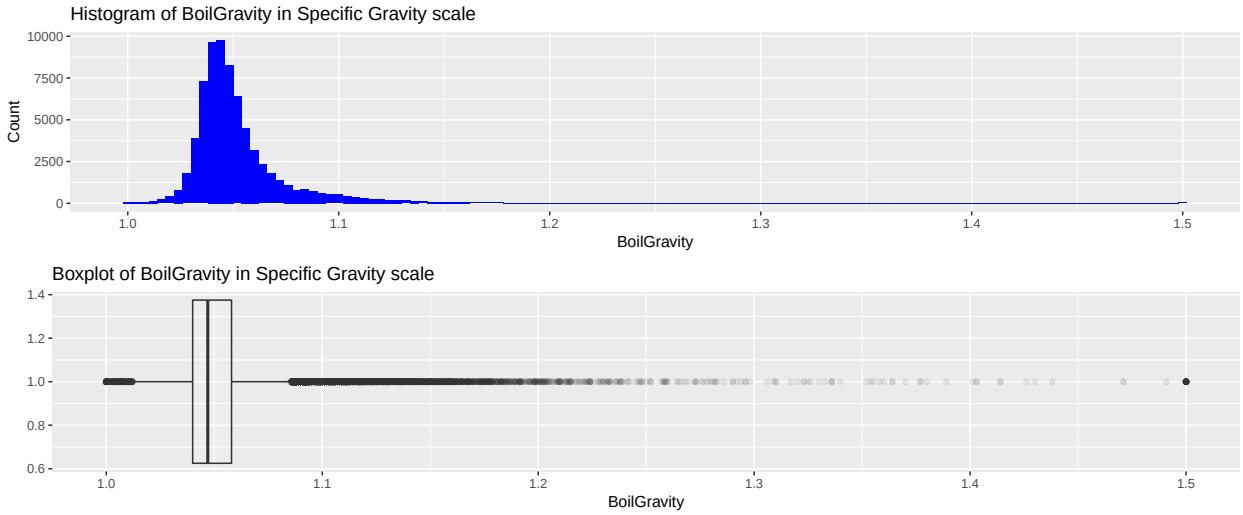
- Plot the histogram of Boil Gravity in Specific Gravity scale

```
# Histogram and boxplot of Boil Gravity
hist_boigravity <- recipedata_sg_scale_dropNAabove30_and_dropNAstyle %>%
  ggplot(aes(BoilGravity)) +
  geom_histogram(binwidth = 0.004, fill = "blue") +
  labs(title = "Histogram of BoilGravity in Specific Gravity scale",
       x = "BoilGravity",
       y = "Count")

boxplot_boilgravity <- recipedata_sg_scale_dropNAabove30_and_dropNAstyle %>%
  ggplot(aes(x = 1, y = BoilGravity)) +
  geom_boxplot(alpha = 0.1) +
  labs(title = "Boxplot of BoilGravity in Specific Gravity scale",
       x = "",
       y = "BoilGravity") +
```

```
  coord_flip()

grid.arrange(hist_boigravity,boxplot_boilgravity, nrow = 2)
```

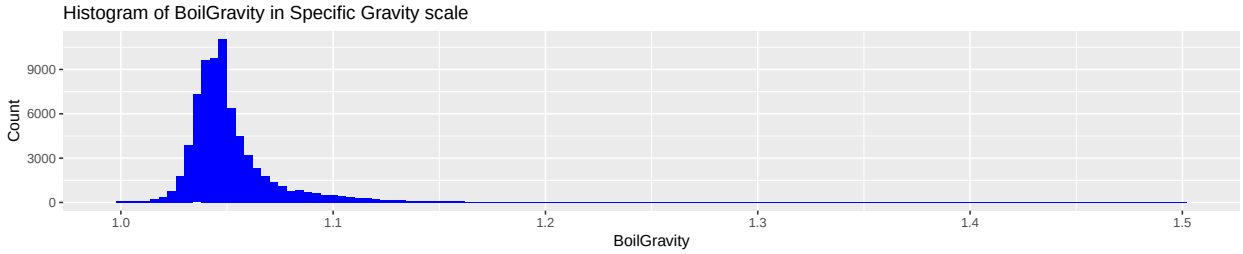
```
## Warning: Removed 2748 rows containing non-finite values (stat_bin).
```

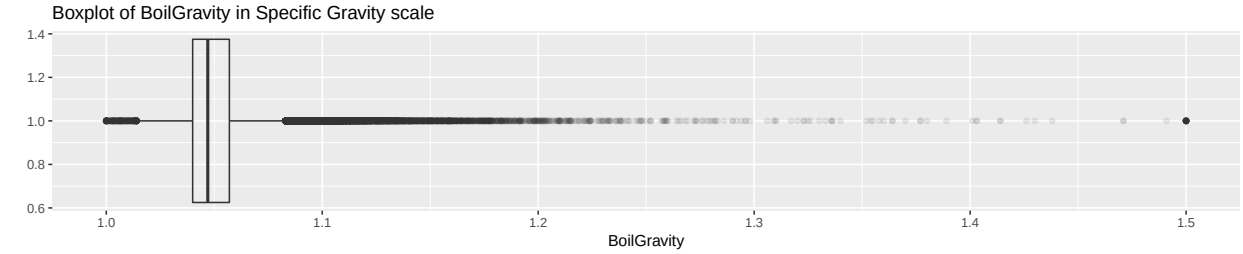
```
## Warning: Removed 2748 rows containing non-finite values (stat_boxplot).
```



The distribution of "Boilgravity" is right skew and the high number of outliers presents. Consequently, it seems that missing values should be replaced by the median.

```
# Replace missing values by the median
recipedata_sg_scale_dropNAabove30_and_dropNAstyle$BoilGravity[which(is.na(recipedata_sg_scale_dropNAabo

# Histogram and boxplot of Boil Gravity
hist_boigravity <- recipedata_sg_scale_dropNAabove30_and_dropNAstyle %>%
  ggplot(aes(BoilGravity)) +
  geom_histogram(binwidth = 0.004, fill = "blue") +
  labs(title = "Histogram of BoilGravity in Specific Gravity scale",
       x = "BoilGravity",
       y = "Count")

boxplot_boilgravity <- recipedata_sg_scale_dropNAabove30_and_dropNAstyle %>%
  ggplot(aes(x = 1, y = BoilGravity)) +
  geom_boxplot(alpha = 0.1) +
  labs(title = "Boxplot of BoilGravity in Specific Gravity scale",
       x = "",
       y = "BoilGravity") +
  coord_flip()

grid.arrange(hist_boigravity,boxplot_boilgravity, nrow = 2)
```





3.4.8 Create tidy dataset for the project

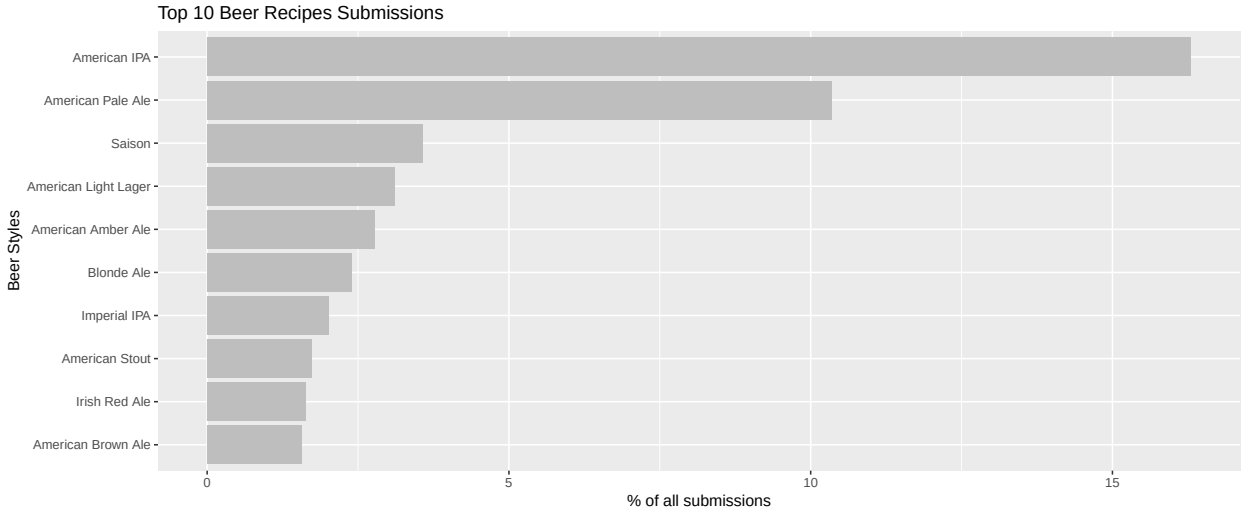
3.4.8.1 Drop variables such as "BeerID", "Name", "URL", and "StyleID"

```
# Drop unnecessary variables
recipedata_tidy <- recipedata_sg_scale_dropNAabove30_and_dropNAstyle %>%
  select(-c("BeerID", "Name", "URL", "StyleID"))
```

3.4.8.2 Create and save the dataset that will be used in this project

- Show the top ten beer styles based off number of submissions

```
top10_beer <- recipedata_tidy %>%
  group_by(Style) %>%
  summarise(n = n(),
            prop = round(100*(n/nrow(recipedata_tidy)),
            digits = 2)) %>%
  top_n(10, prop)

top10_beer %>%
  ggplot(aes(x = reorder(Style, prop), y = prop)) +
  geom_bar(stat = "identity", fill = "grey") +
  labs(title = "Top 10 Beer Recipes Submissions",
       x = "Beer Styles",
       y = "% of all submissions") +
  coord_flip()
```



- Create "recipe_top10" dataset and save it

```
# Create "recipe_top10" dataset
recipe_top10 <- recipedata_tidy %>%
  filter(Style %in% top10_beer$Style) %>%
  droplevels() # Important step to for createDataPartition to work

# Save "recipe_top10" dataset
save(recipe_top10, file = "recipe_top10.RData")
```

4 Exploratory data analysis

4.1 Glimpse at the dataset

The dataset that is used for predicting the top ten popular beer styles contains 33,308 observations and 13 variables. It had three categorical variables including "Style", "SugarScale" and "BrewMethod", and 10 numerical variable as shown below.

```
# Look at the recipe_top10 dataset
glimpse(recipe_top10)
```

```
## Observations: 33,308
## Variables: 12
## $ Style       <fct> American IPA, American IPA, American Pale Ale, Imp...
## $ sizeL       <dbl> 18.93, 22.71, 24.61, 22.71, 20.82, 25.00, 15.14, 2...
## $ OG          <dbl> 1.063, 1.061, 1.055, 1.072, 1.080, 1.064, 1.066, 1...
## $ FG          <dbl> 1.018, 1.017, 1.013, 1.018, 1.017, 1.014, 1.015, 1...
## $ ABV         <dbl> 5.91, 5.80, 5.58, 7.09, 8.22, 6.63, 6.62, 7.07, 6....
## $ IBU         <dbl> 59.25, 54.48, 40.12, 268.71, 93.02, 64.26, 111.00,...
## $ Color       <dbl> 8.98, 8.50, 8.00, 6.33, 8.29, 7.78, 14.26, 6.28, 1...
## $ BoilSize    <dbl> 22.71, 26.50, 29.34, 30.28, 28.39, 29.00, 11.36, 2...
## $ BoilTime    <dbl> 60, 60, 70, 90, 60, 90, 90, 60, 60, 60, 60, 60, 60...
## $ BoilGravity <dbl> 1.047, 1.047, 1.047, 1.047, 1.058, 1.055, 1.047, 1...
## $ Efficiency  <dbl> 70, 70, 79, 75, 70, 74, 70, 70, 30, 70, 70, 75, 70...
## $ BrewMethod  <fct> extract, All Grain, All Grain, All Grain, All Grai...
```

4.2 Summary statistics of the dataset

- There is no missing value in the dataset.
- "BoilSize" and "sizeL" have high standard Deviation 193.82 and 182.13 respectively. Moreover, there are extreme outliers in both variables. For example, the maximum value of "BoilSize" is 6454, while its minumum one is 1.
- The outliers would be observed clearly through the boxplot.

```
# Summary statistics of the dataset
skim(recipe_top10)
```

```
## Skim summary statistics
##  n obs: 33308
##  n variables: 12
##
## -- Variable type:factor ---------------------------------------------------------------------------
##    variable missing complete     n n_unique
##  BrewMethod       0    33308 33308        4
##       Style       0    33308 33308       10
##                                  top_counts ordered
##  All: 22190, BIA: 5463, ext: 4106, Par: 1549   FALSE
##  Ame: 11940, Ame: 7581, Sai: 2617, Ame: 2277   FALSE
##
## -- Variable type:numeric --------------------------------------------------------------------------
##     variable missing complete     n  mean       sd p0   p25   p50   p75
##          ABV       0    33308 33308  6.04   1.43    0  5.22  5.91  6.71
##  BoilGravity       0    33308 33308  1.05   0.024   1  1.04  1.05  1.05
##     BoilSize       0    33308 33308 50.86 193.82    1 21    28    30
##     BoilTime       0    33308 33308 63.63  12.07    0 60    60    60
##        Color       0    33308 33308 10.35   8.74    0  5.41  7.42 11.35
##   Efficiency       0    33308 33308 66.04  14.15    0 65    70    75
##           FG       0    33308 33308  1.01   0.0046  1  1.01  1.01  1.02
##          IBU       0    33308 33308 53.33  44.73    0 29.9  44.26 66.62
##           OG       0    33308 33308  1.06   0.014   1  1.05  1.06  1.07
##        sizeL       0    33308 33308 45.04 182.13    1 18.93 20.82 24
##     p100
##    54.72
##     1.5
##  6454.13
##   240
##   108.65
##   100
##     1.16
##  2197.07
##     1.5
##  6102.08
```

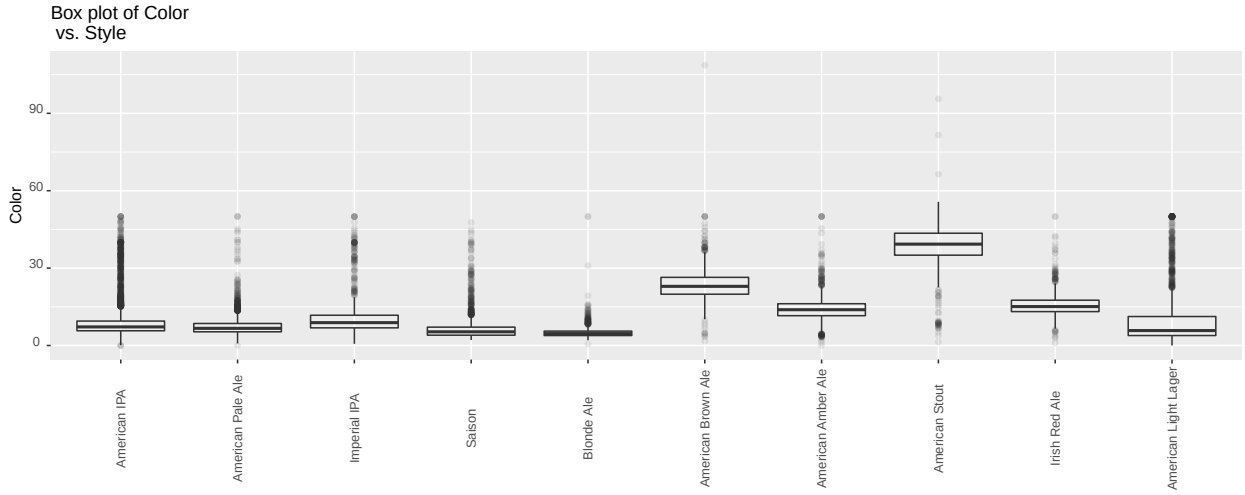
4.3 Make boxplots of every numerical variables in the recipe_top10 again Style

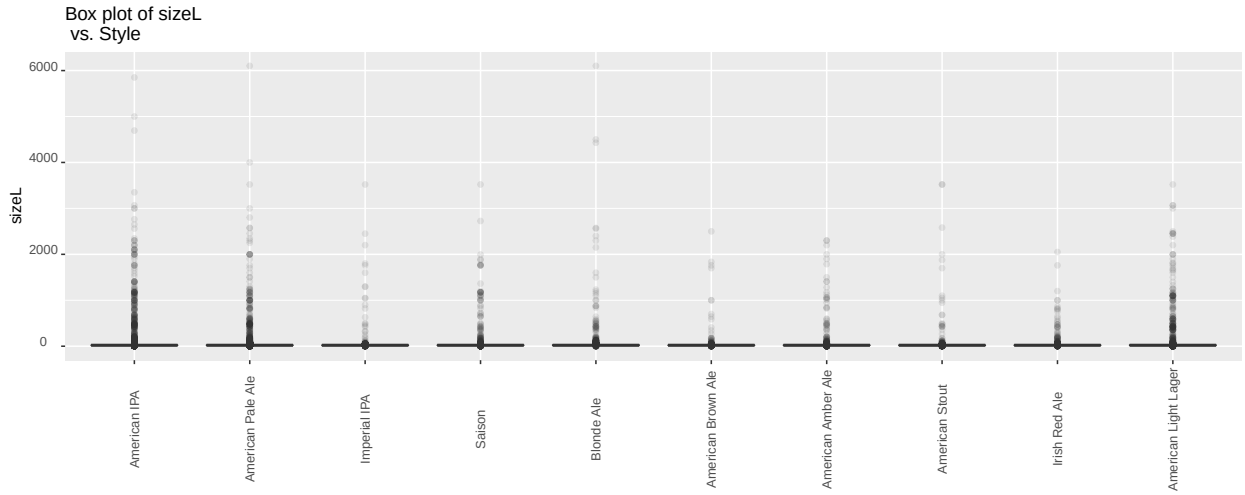
Looking at the boxplot of all numerical variables, it is not only "BoilSize" and "sizeL", that experienced high amount of outliers, but also "IBU", "ABV" and "BoilTime".

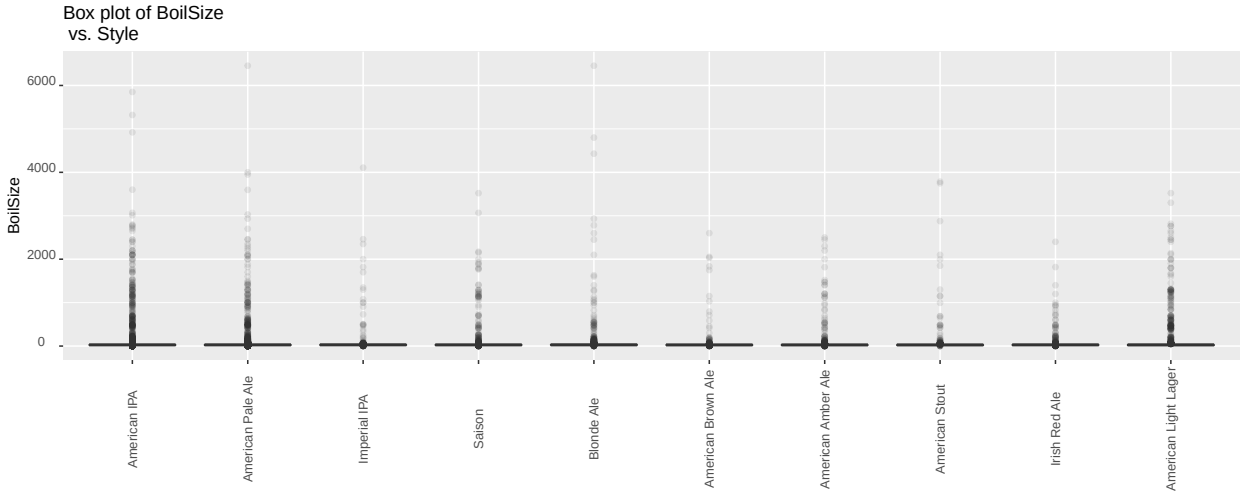
The graph would be improved if we take logarithm base 10 for all numerical variables.

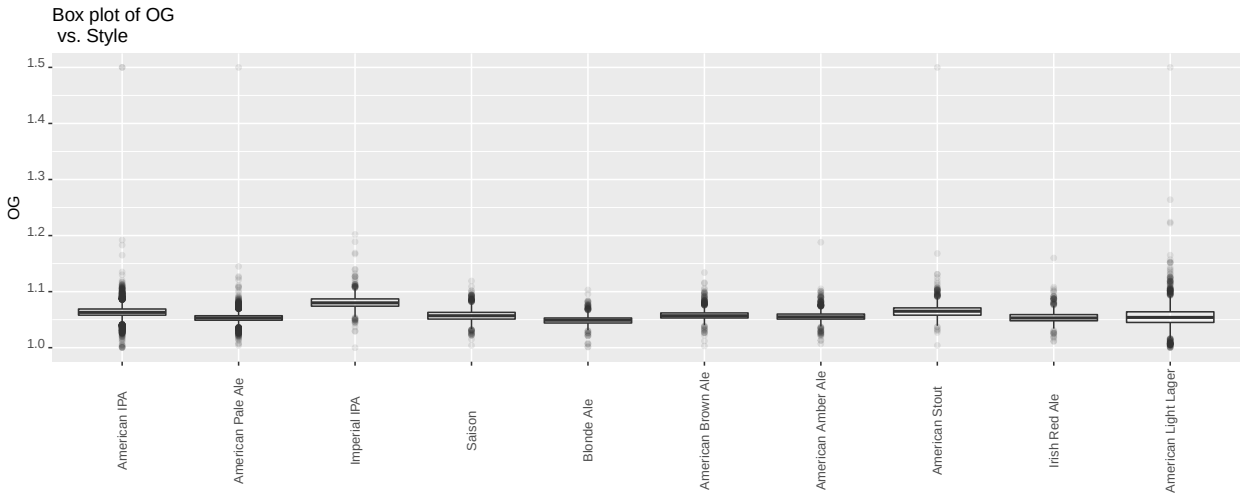
```
# Make boxplots of every numerical variables in the recipe_top10
box_plot_numeric_vars <- function(df, cols, col_x = "Style"){
```

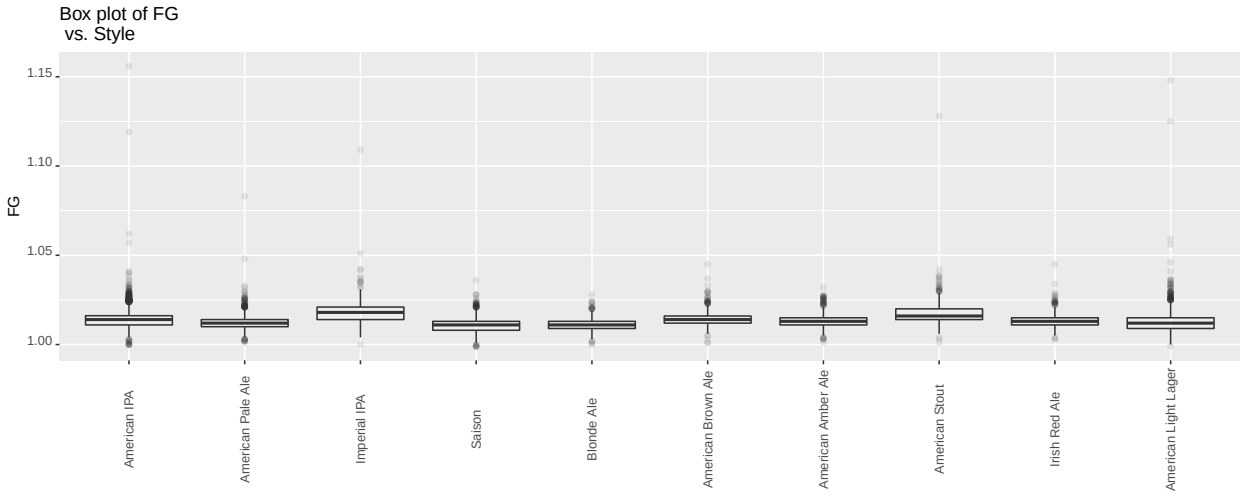
```
    options(repr.plot.width = 4, repr.plot.height = 3.5) # Set the initial plot area dimensions
    for(col in cols){
        p <- ggplot(df, aes_string(col_x, col)) +
                    geom_boxplot(alpha = 0.1) +
                    theme(axis.text.x = element_text(angle = 90, hjust = 0.5, vjust = 0.5),
                          axis.text.y = element_text(angle = 0, hjust = 0.5, vjust = 0)) +
                    labs(title = paste('Box plot of', col, '\n vs.', col_x),
                         x = "",
                         y = col)

        print(p)
    }
}

num_cols <- c("Color", "sizeL", "BoilSize", "OG", "FG", "ABV", "IBU", "BoilTime",
              "BoilGravity", "Efficiency")

box_plot_numeric_vars(recipe_top10, num_cols)
```

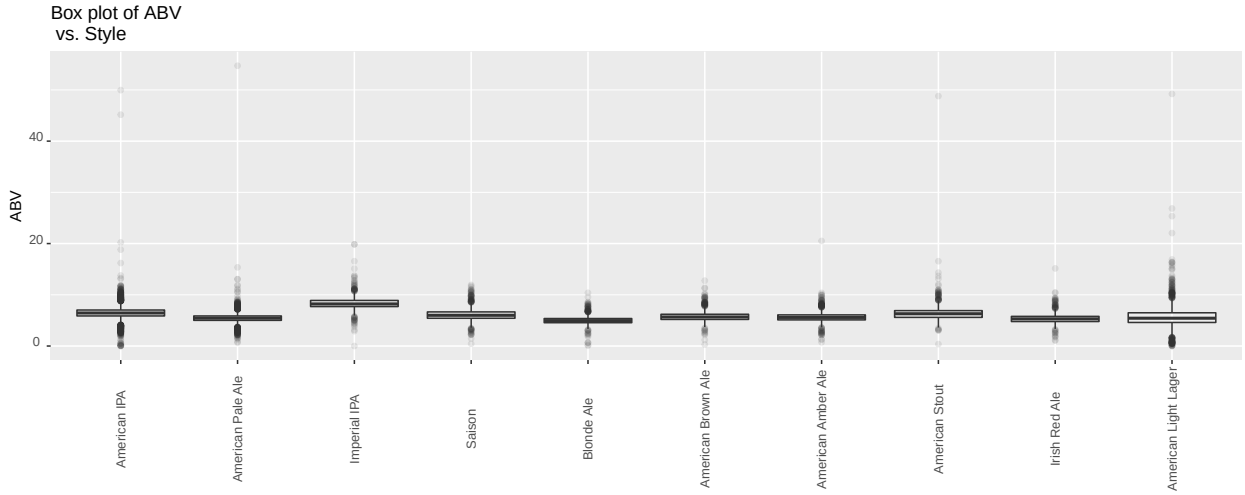


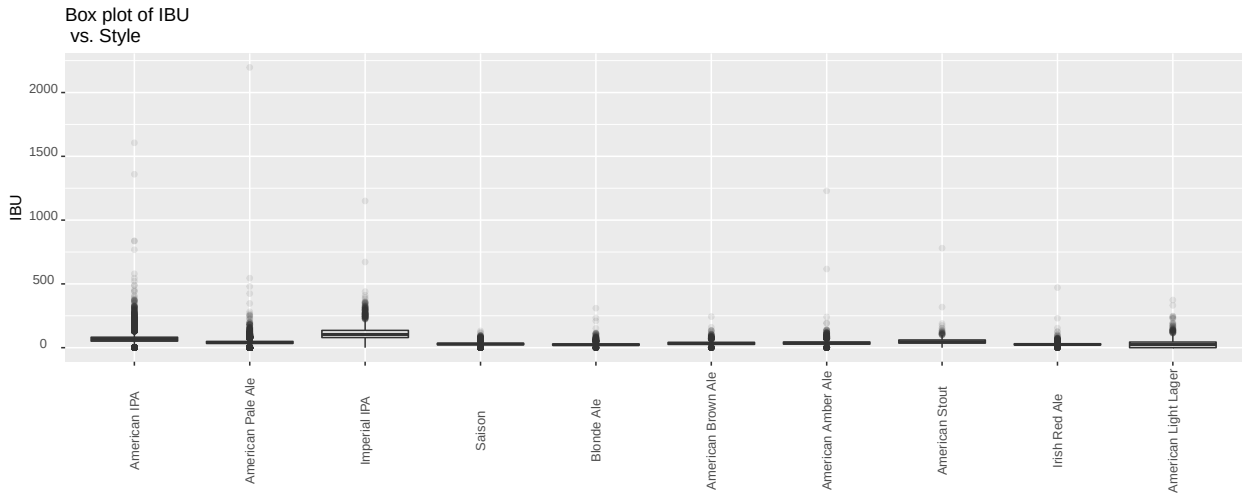


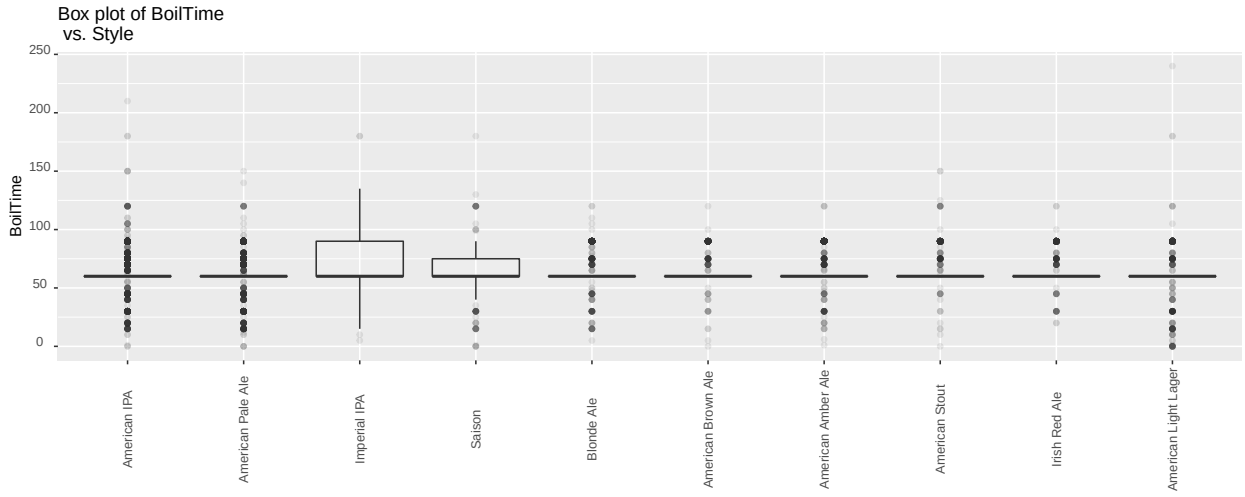

Box plot of BoilSize
vs. Style
BoilSize
6000
4000
2000
0
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager

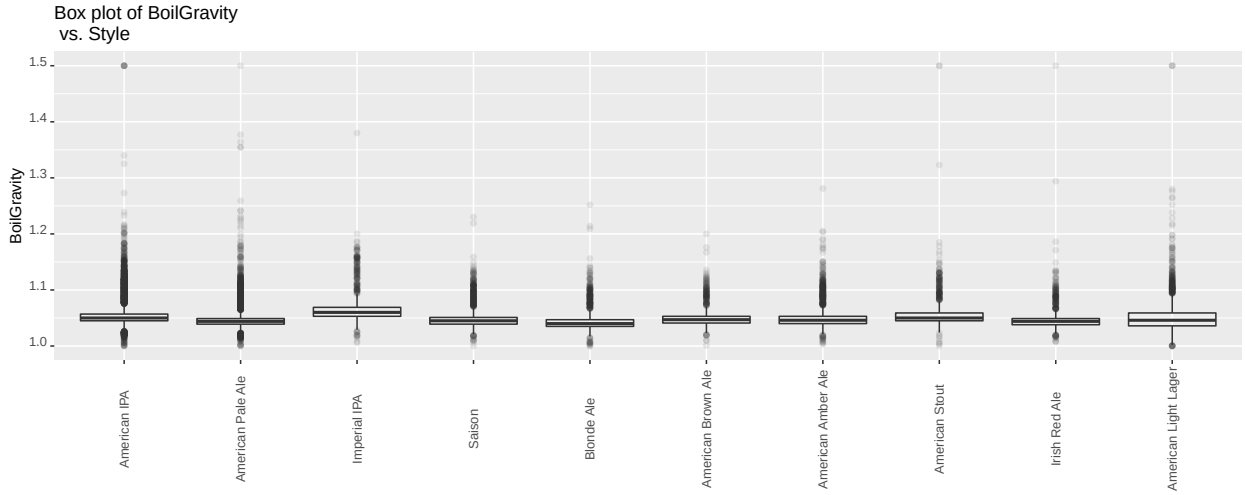

Box plot of OG
vs. Style
OG
1.5
1.4
1.3
1.2
1.1
1.0
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager

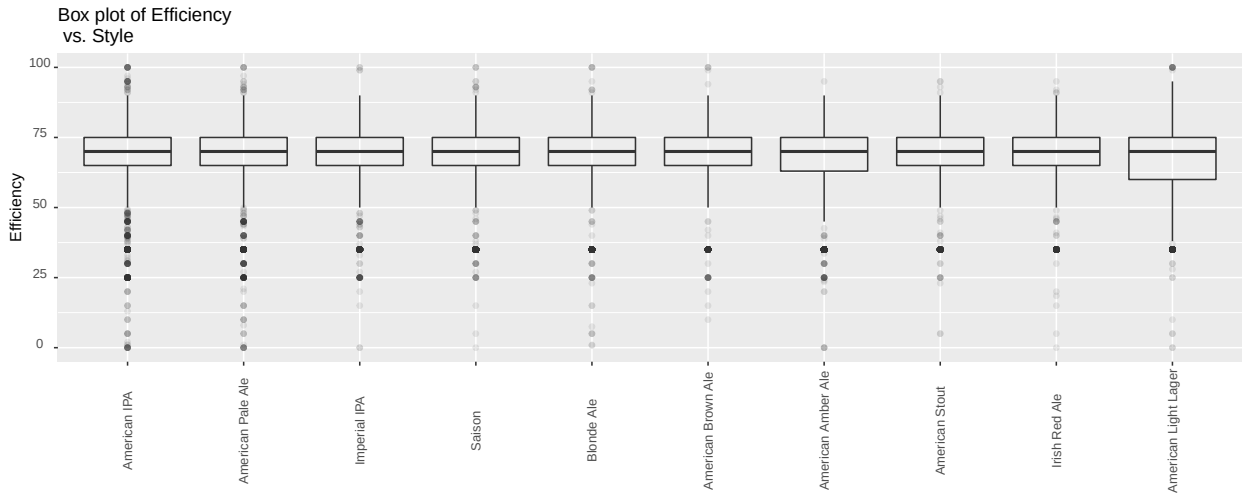

Box plot of FG
vs. Style
FG
1.15
1.10
1.05
1.00
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager


Box plot of ABV
vs. Style
ABV
40
20
0
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager


Box plot of IBU
vs. Style
IBU
2000
1500
1000
500
0
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager


Box plot of BoilTime
vs. Style
BoilTime
250
200
150
100
50
0
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager





4.4 Transform all numerical variables in the dataset with log10 to duel with extreme values

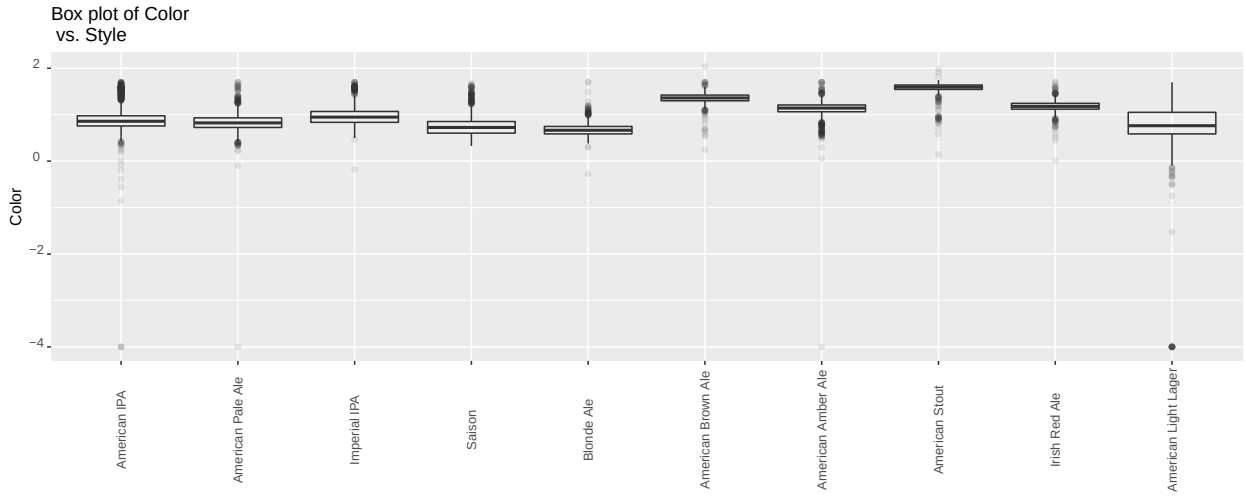
```
# log10 function with addional small value to handle zero
func_log10 <- function(x) (log10(x+0.0001))

# New dataset with log10 transformation for all numerical variables
recipe_top10_log10 <- recipe_top10 %>%
  mutate_if(is.numeric, func_log10)
```

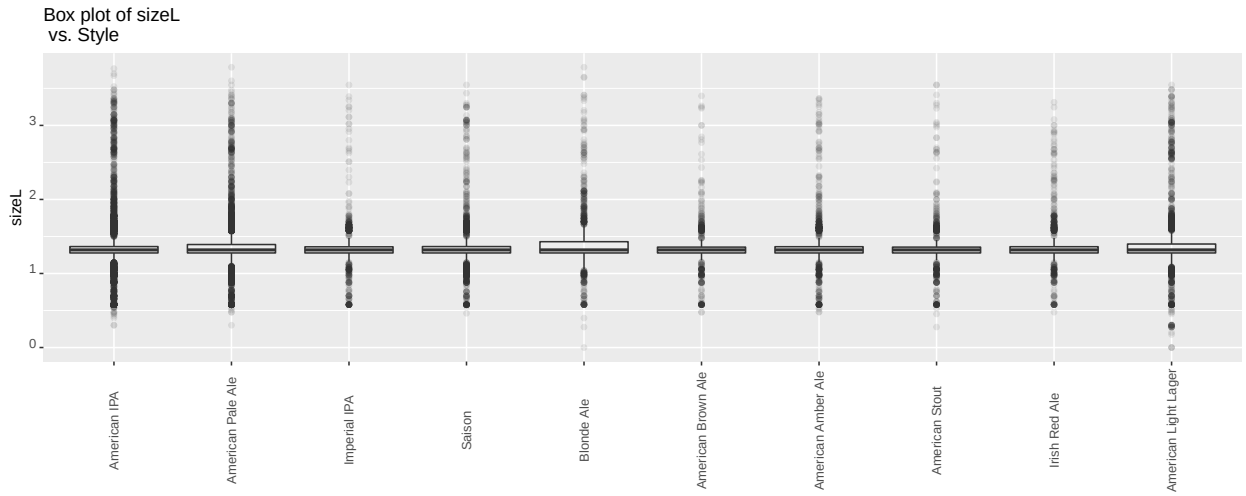
4.5 Boxplot of transformed dataset

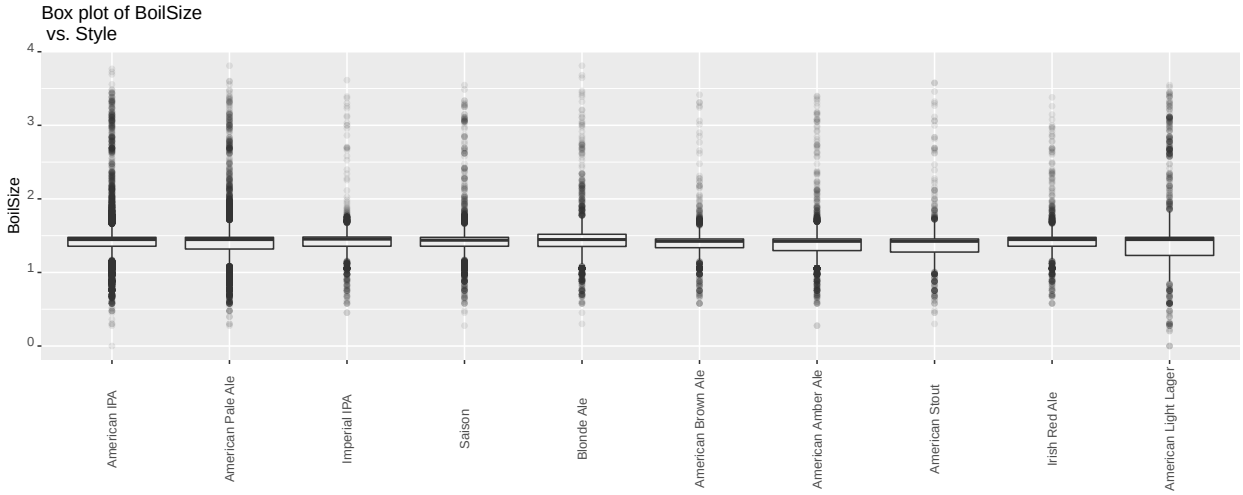
The boxplot had shown there would have significant explanation power for "Style" using "Color", "OG", "FG" and "IBU".

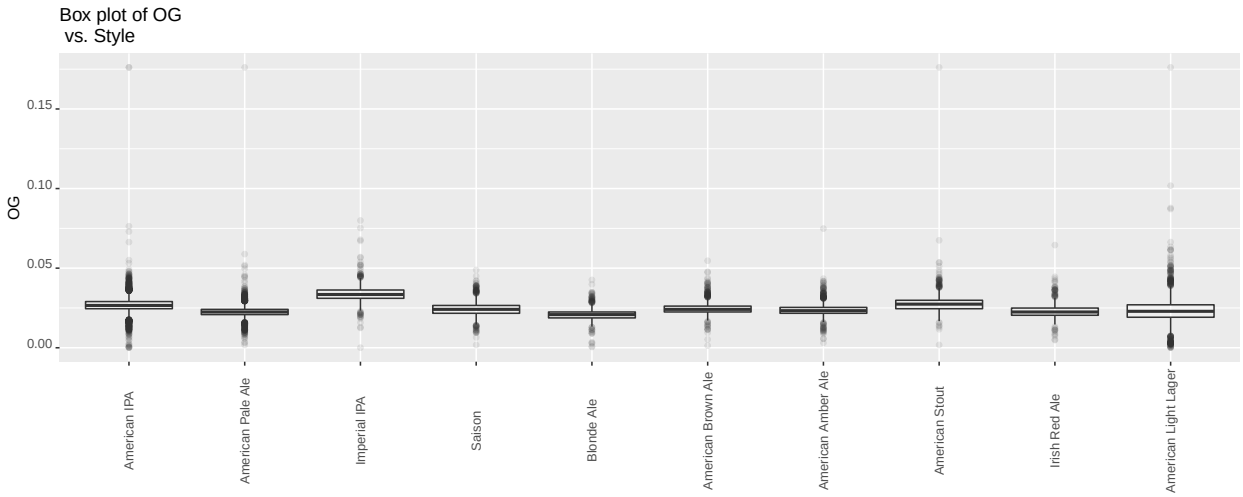
```
# Make boxplots of every numerical variables in the recipe_top10_log10
box_plot_numeric_vars <- function(df, cols, col_x = "Style"){
    options(repr.plot.width = 4, repr.plot.height = 3.5) # Set the initial plot area dimensions
    for(col in cols){
        p <- ggplot(df, aes_string(col_x, col)) +
```

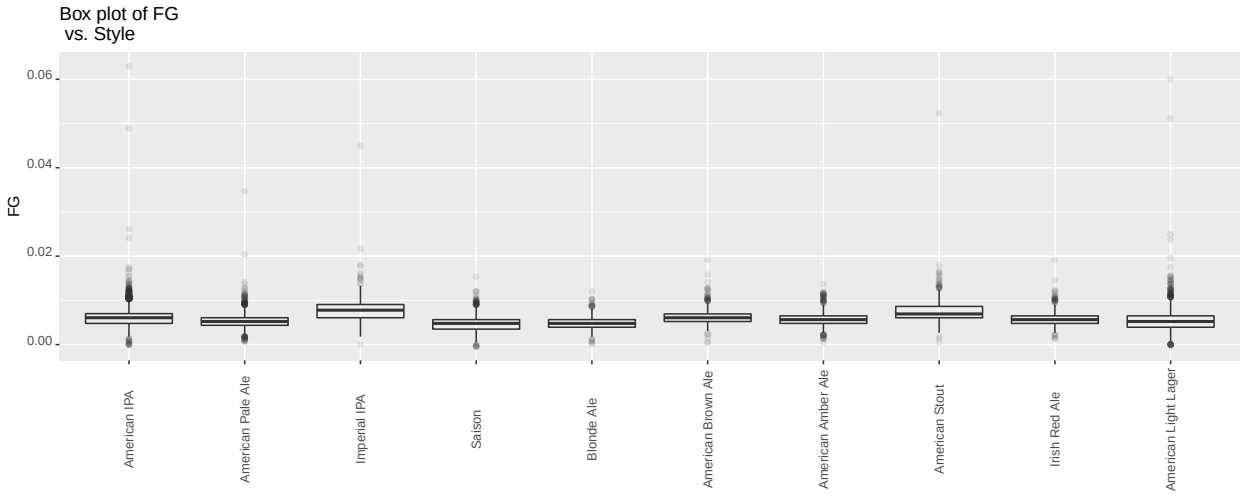
```
                    geom_boxplot(alpha = 0.1) +
                    theme(axis.text.x = element_text(angle = 90, hjust = 0.5, vjust = 0.5),
                          axis.text.y = element_text(angle = 0, hjust = 0.5, vjust = 0)) +
                    labs(title = paste('Box plot of', col, '\n vs.', col_x),
                         x = "",
                         y = col)

        print(p)
    }
}

num_cols <- c("Color", "sizeL", "BoilSize", "OG", "FG", "ABV", "IBU", "BoilTime",
              "BoilGravity", "Efficiency")

box_plot_numeric_vars(recipe_top10_log10, num_cols)
```

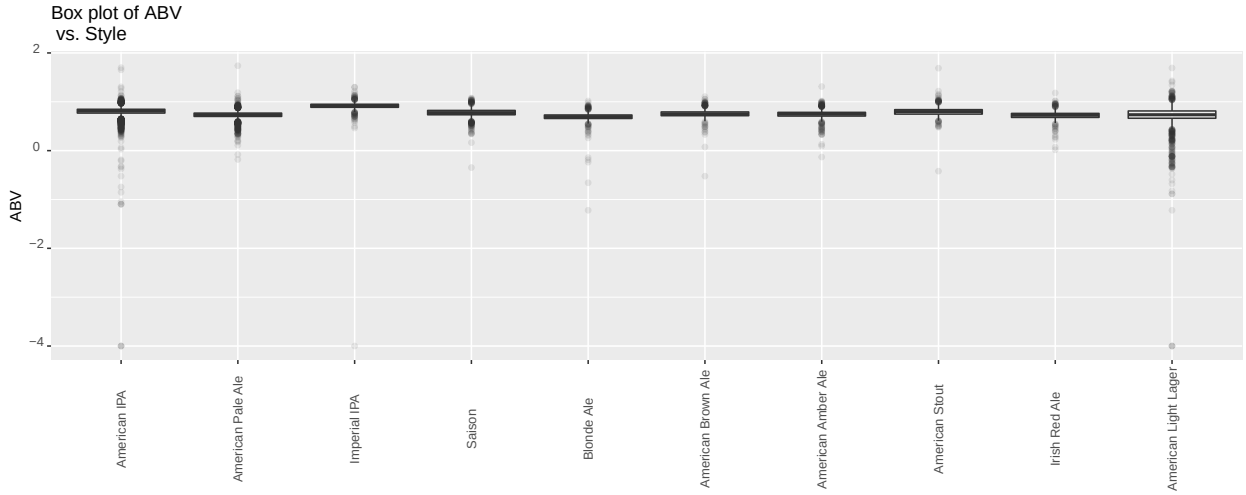


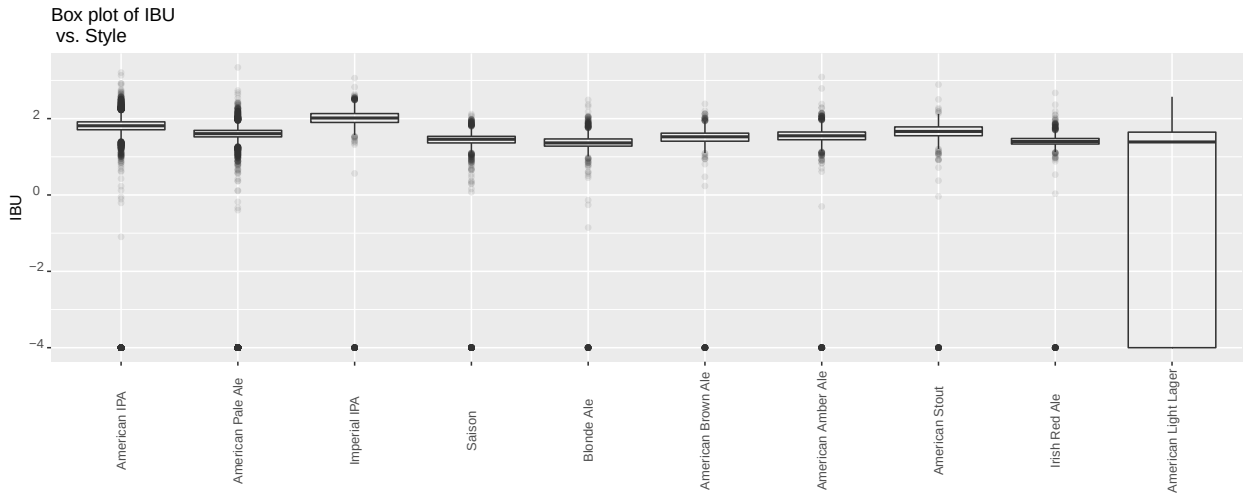


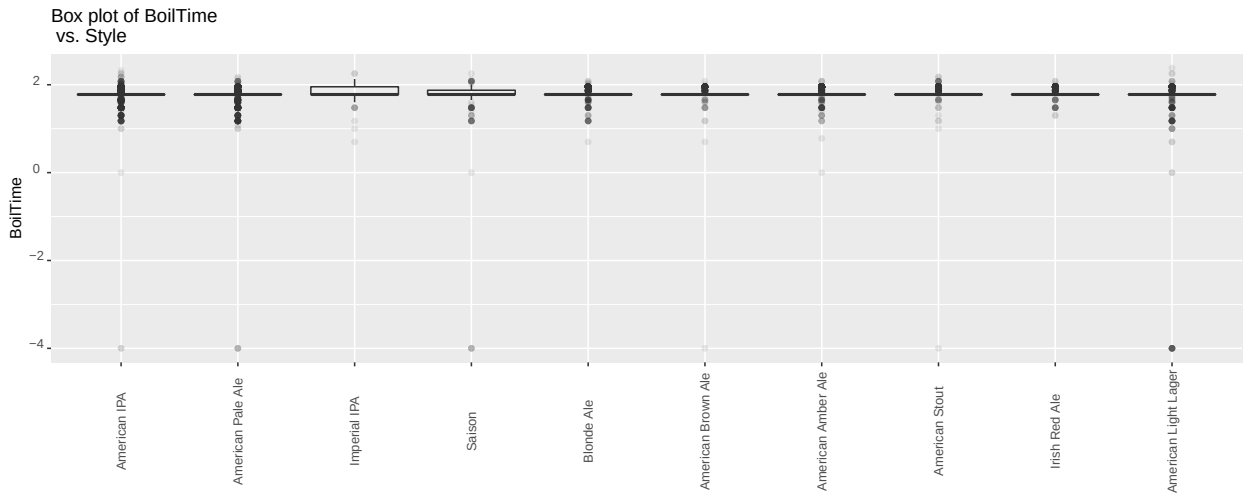

Box plot of BoilSize
vs. Style
BoilSize
4
3
2
1
0
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager

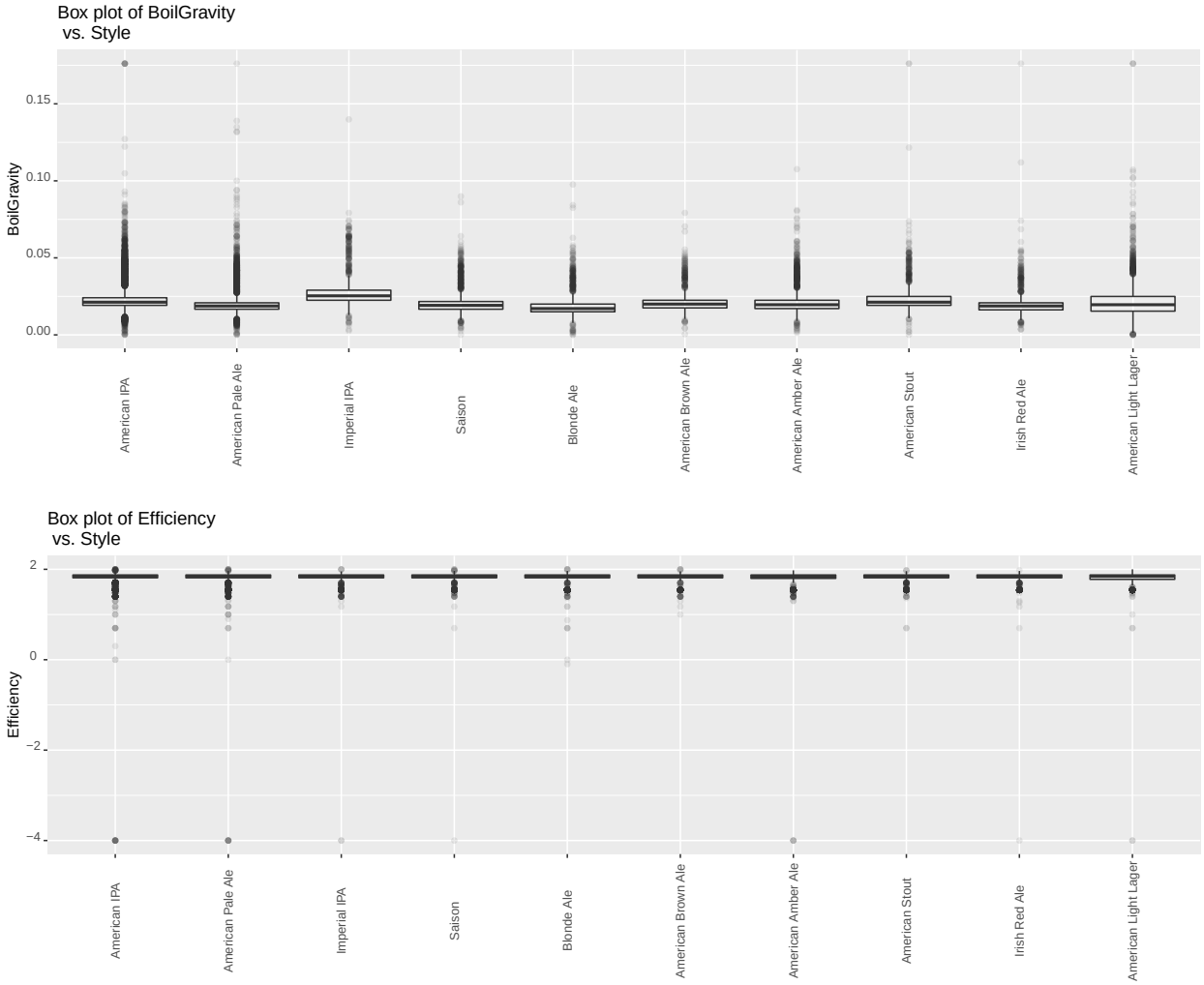

Box plot of OG
vs. Style
OG
0.15
0.10
0.05
0.00
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager


Box plot of FG
vs. Style
FG
0.06
0.04
0.02
0.00
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager


Box plot of ABV
vs. Style
ABV
2
0
−2
−4
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager


Box plot of IBU
vs. Style
IBU
2
0
−2
−4
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager


Box plot of BoilTime
vs. Style
BoilTime
2
0
−2
−4
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager

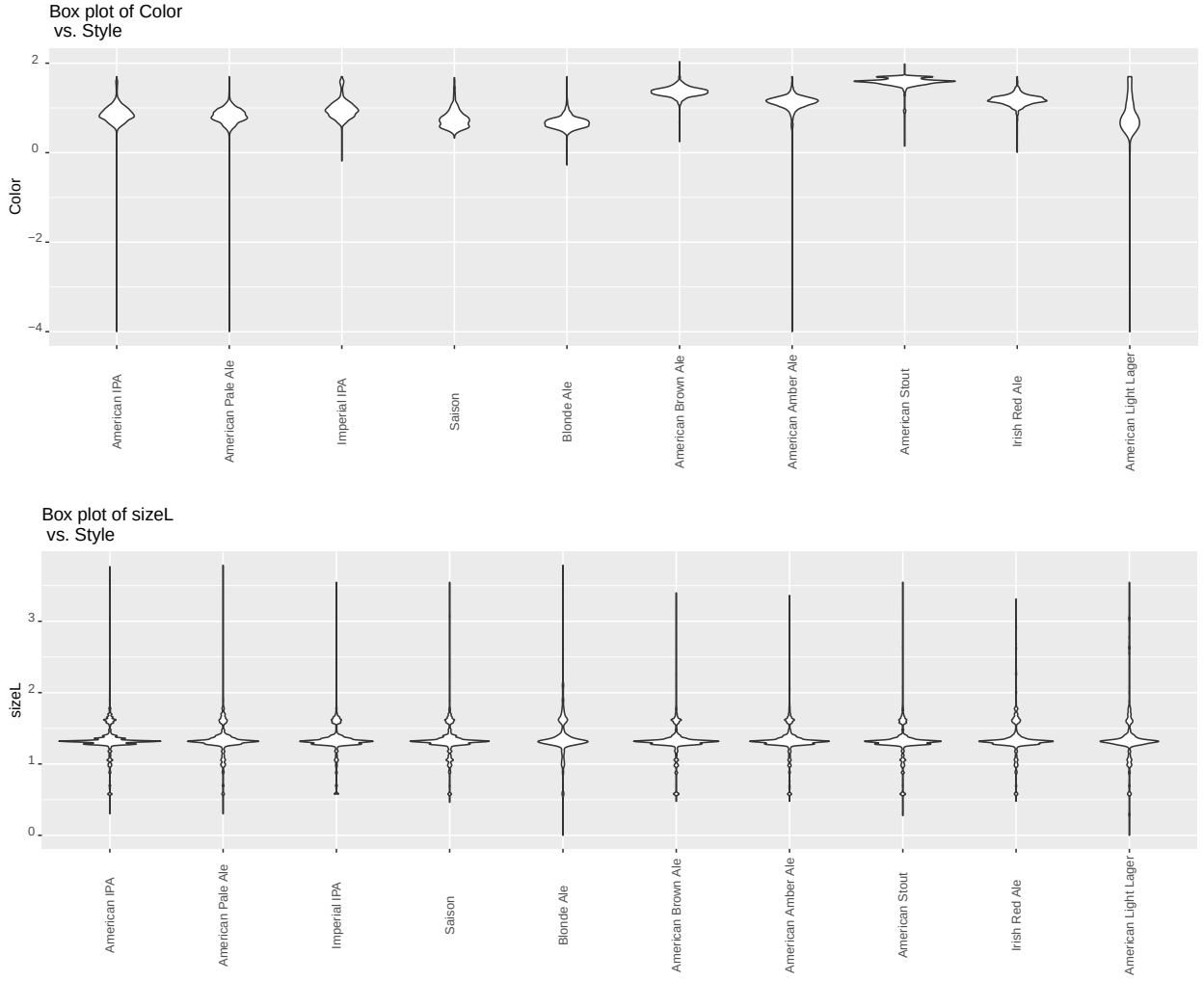


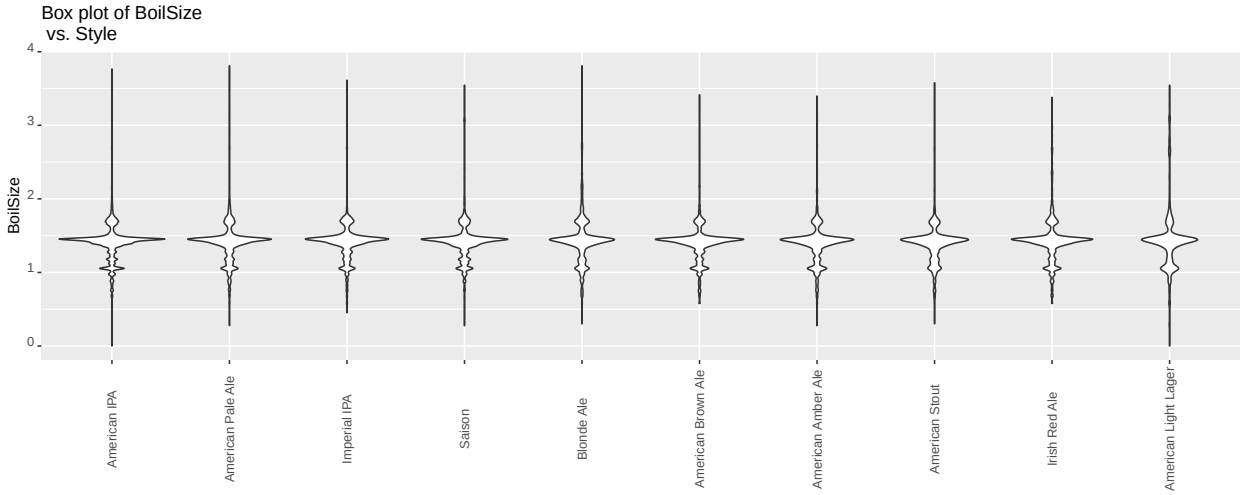
4.6 Violin plot of transformed dataset

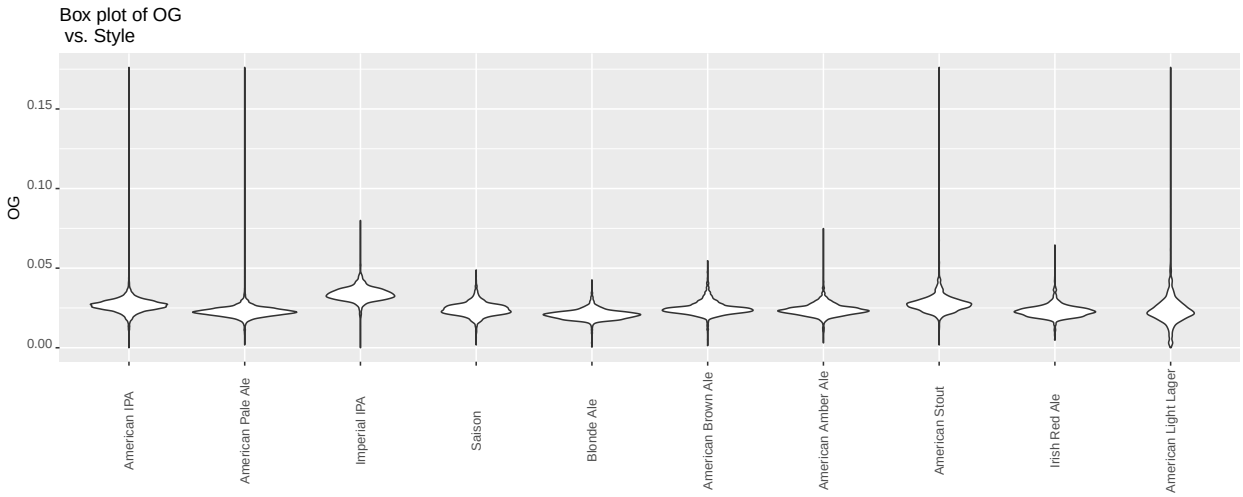
There are groups of variables that seem to have identical shapes, for example "BoilSize" and "sizeL", "OG" and "FG".

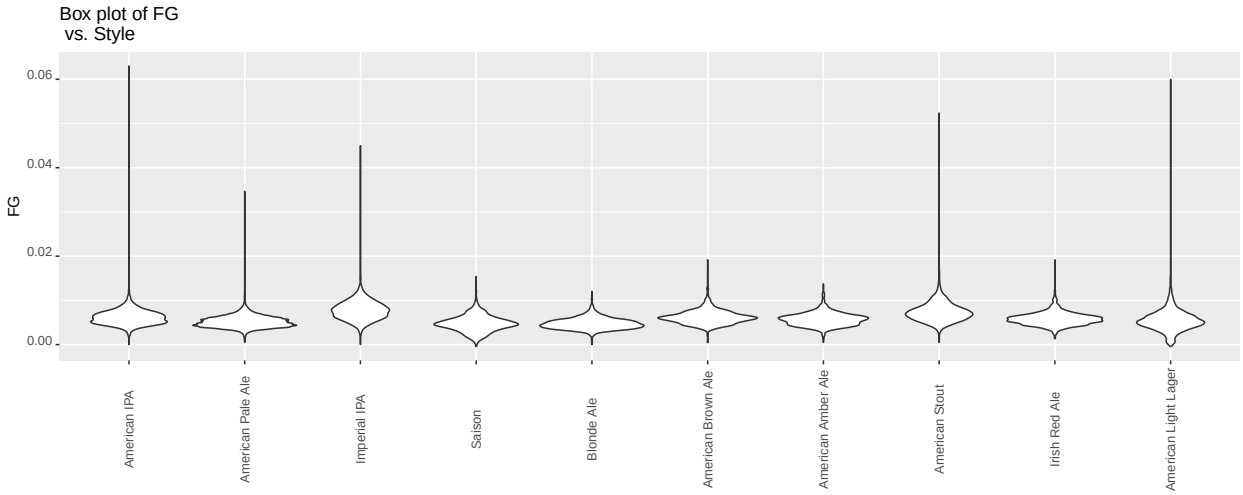
```
# Violin plot of transformed dataset
violin_plot_numeric_vars <- function(df, cols, col_x = "Style"){
    options(repr.plot.width=4, repr.plot.height=3.5) # Set the initial plot area dimensions
    for(col in cols){
        p <- ggplot(df, aes_string(col_x, col)) +
                  geom_violin() +
                  theme(axis.text.x = element_text(angle = 90, hjust = 0.5, vjust = 0.5),
                        axis.text.y = element_text(angle = 0, hjust = 0.5, vjust = 0)) +
                  labs(title = paste('Box plot of', col, '\n vs.', col_x),
                       x = "",
                       y = col)

        print(p)
    }
}
```

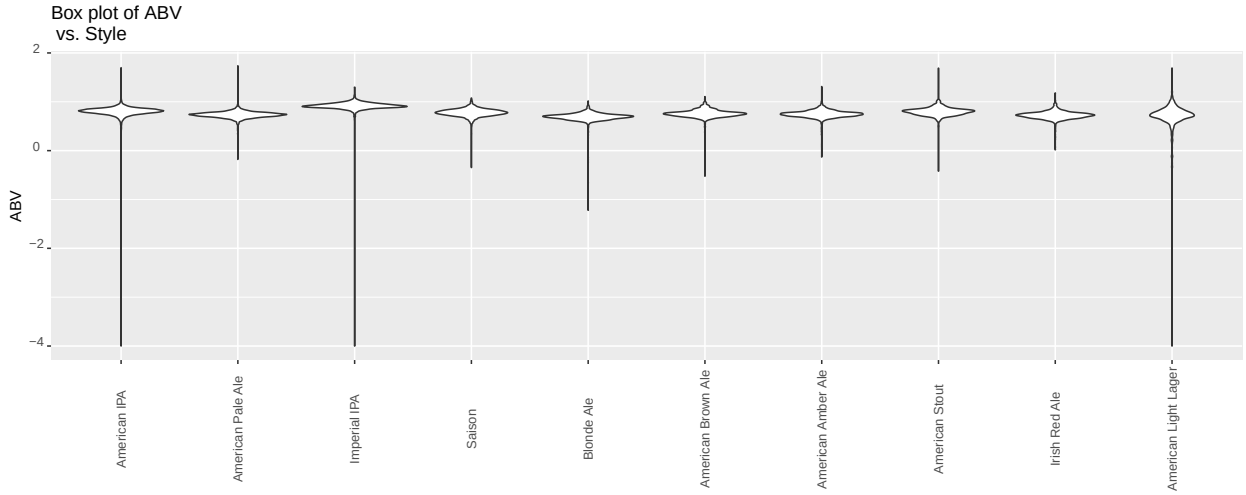
```
num_cols <- c("Color", "sizeL", "BoilSize", "OG", "FG", "ABV", "IBU", "BoilTime",
              "BoilGravity", "Efficiency")

violin_plot_numeric_vars(recipe_top10_log10, num_cols)
```

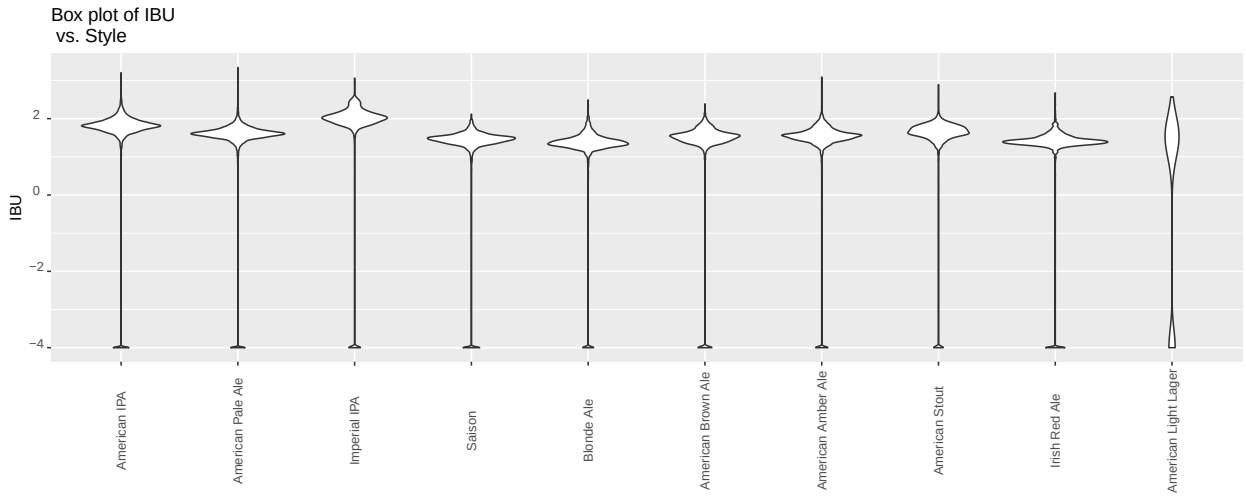


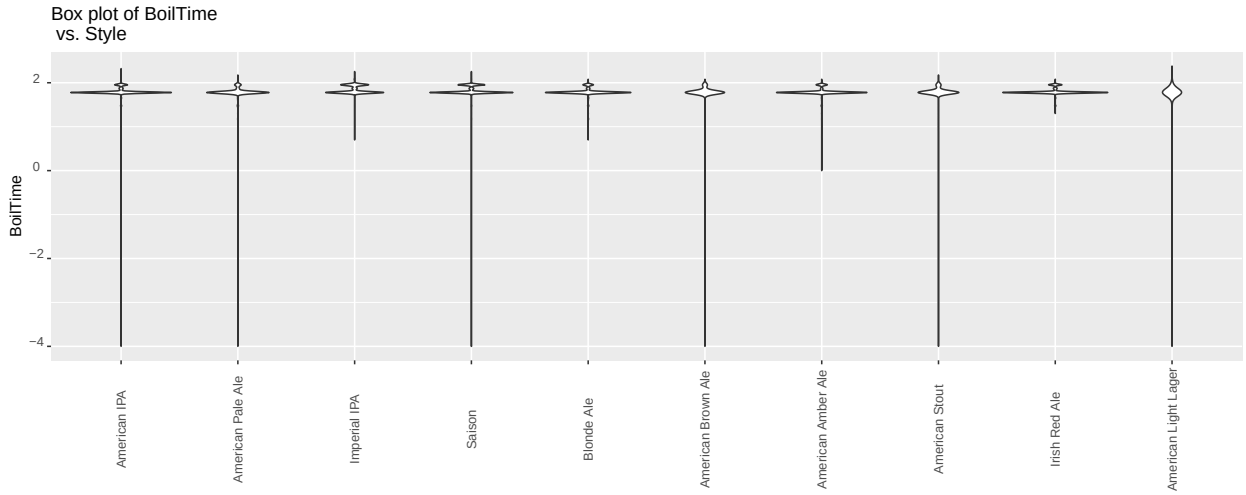

Box plot of BoilSize
vs. Style
BoilSize
4
3
2
1
0
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager

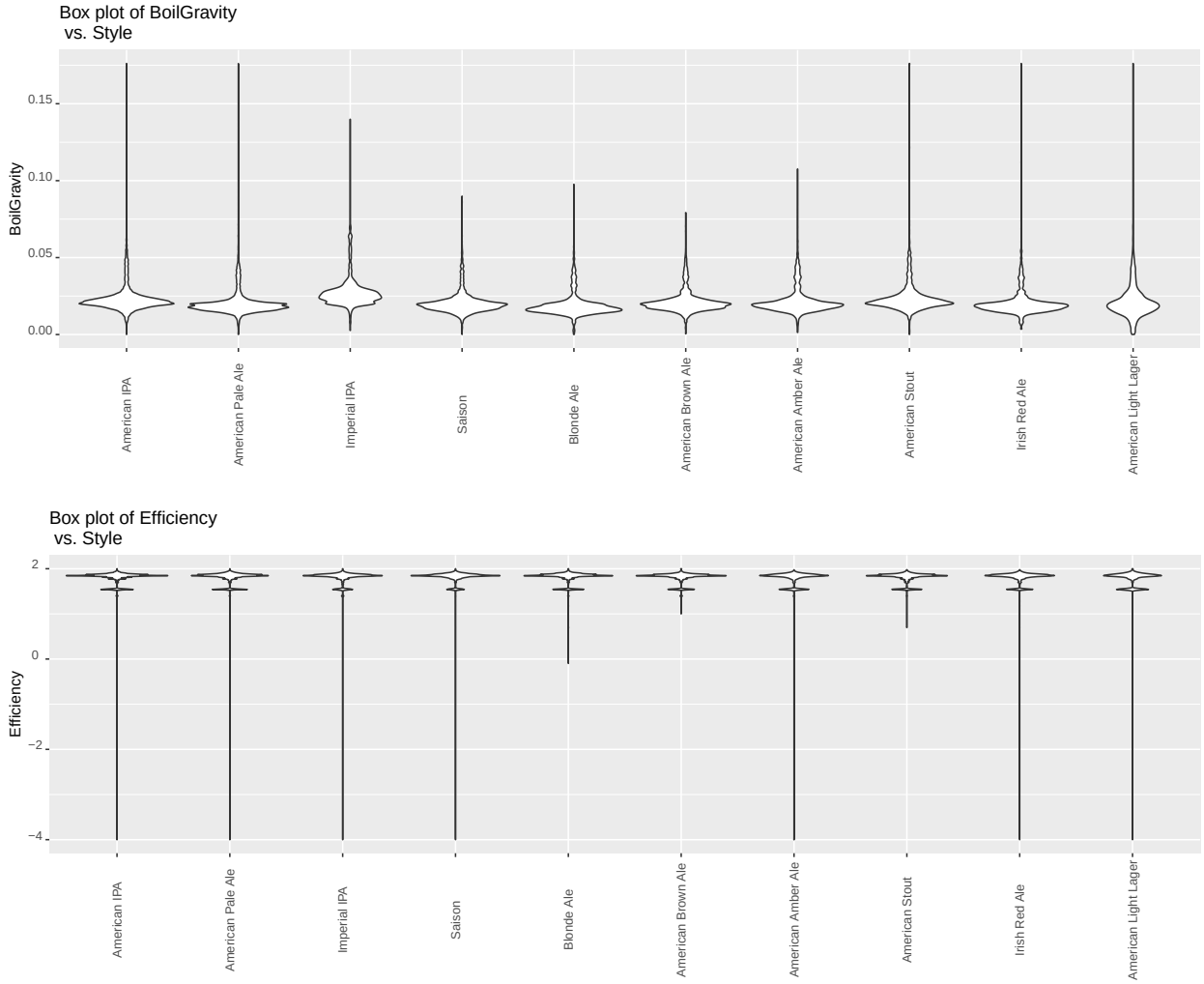

Box plot of OG
vs. Style
OG
0.15
0.10
0.05
0.00
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager


Box plot of FG
vs. Style
FG
0.06
0.04
0.02
0.00
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager


Box plot of ABV
vs. Style
ABV
2
0
−2
−4
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager


Box plot of IBU
vs. Style
IBU
2
0
−2
−4
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager


Box plot of BoilTime
vs. Style
BoilTime
2
0
−2
−4
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager

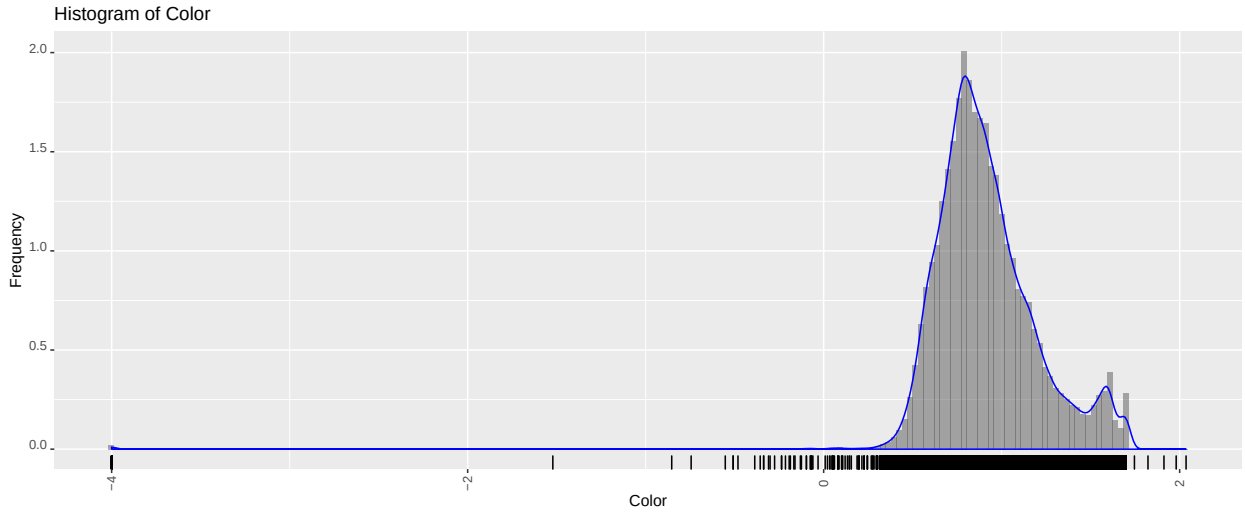


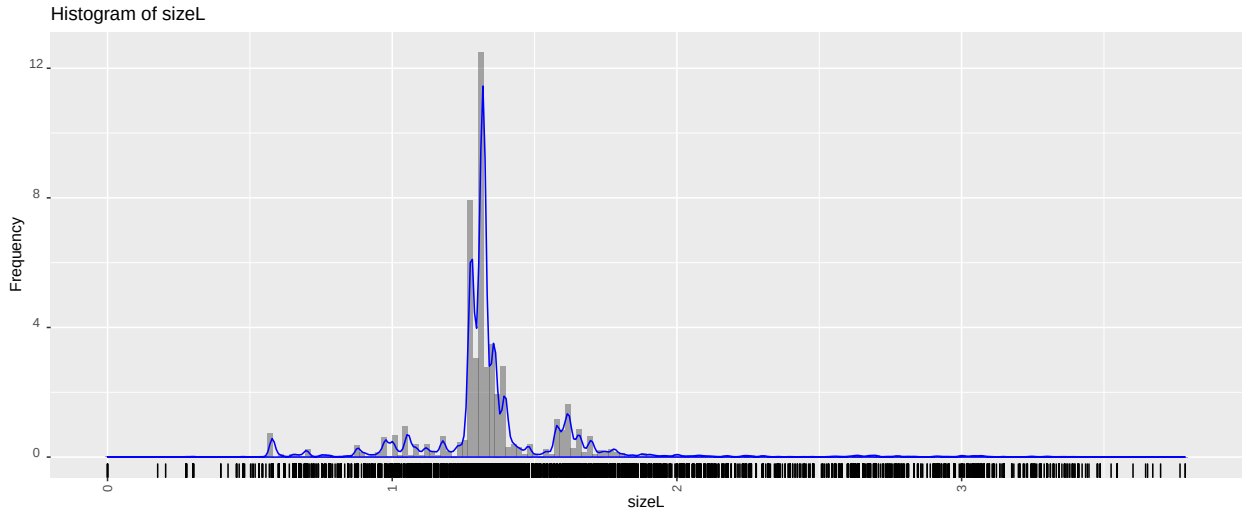
4.7 Histogram of the transformed data

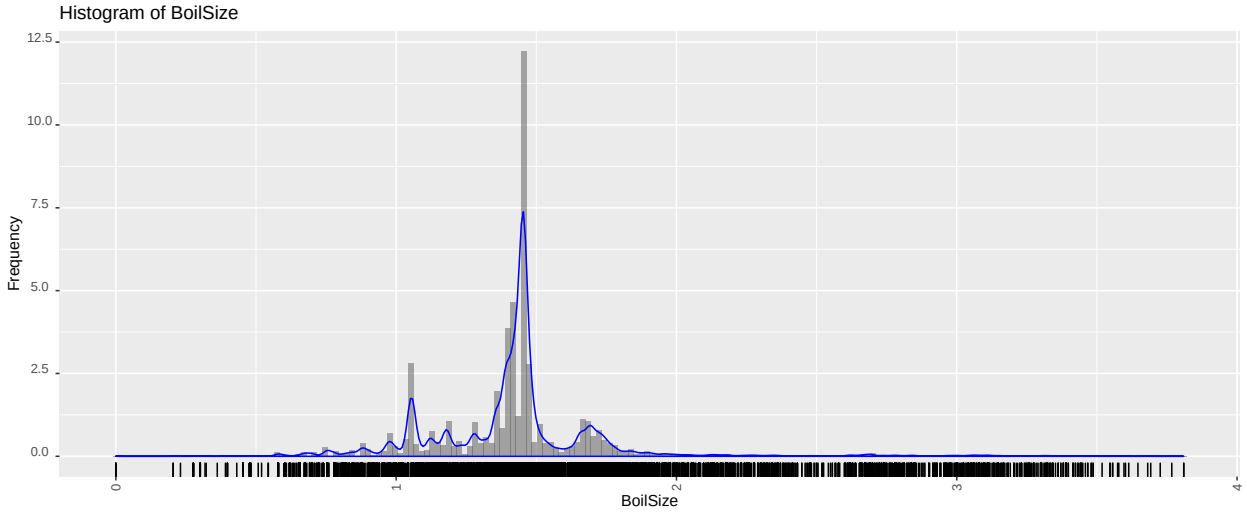
"OG" and "ABV" look approximately like the normal distribution, while "Color" and "IBU" are skew to the left. "FG" and "BoilGravity" are right skew.

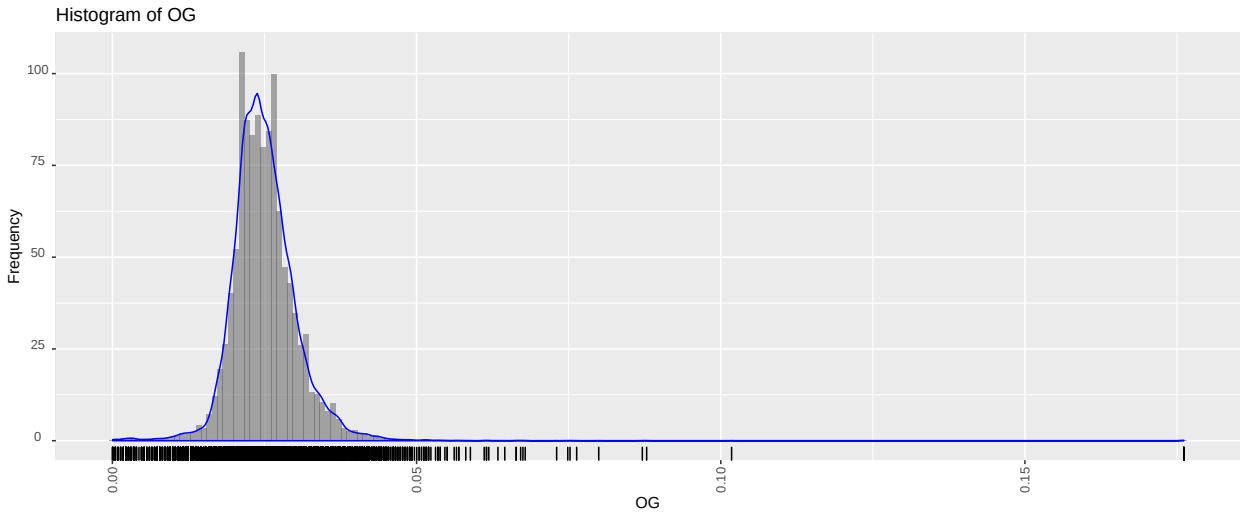
```
# Histogram of the transformed data
hist_plot_numeric_vars <- function(df, cols, col_x = "Style"){
    options(repr.plot.width=4, repr.plot.height=3.5) # Set the initial plot area dimensions
    for(col in cols){
        p <- ggplot(df, aes_string(col)) +
                geom_histogram(aes(y=..density..),
                               alpha = 0.5,
                               bins = 200) +
                geom_density(aes(y=..density..),
                             color = 'blue') +
                geom_rug() +
                theme(axis.text.x = element_text(angle = 90, hjust = 0.5, vjust = 0.5),
                      axis.text.y = element_text(angle = 0, hjust = 0.5, vjust = 0)) +
                labs(title = paste('Histogram of', col),
                     x = col,
                     y = "Frequency")
```

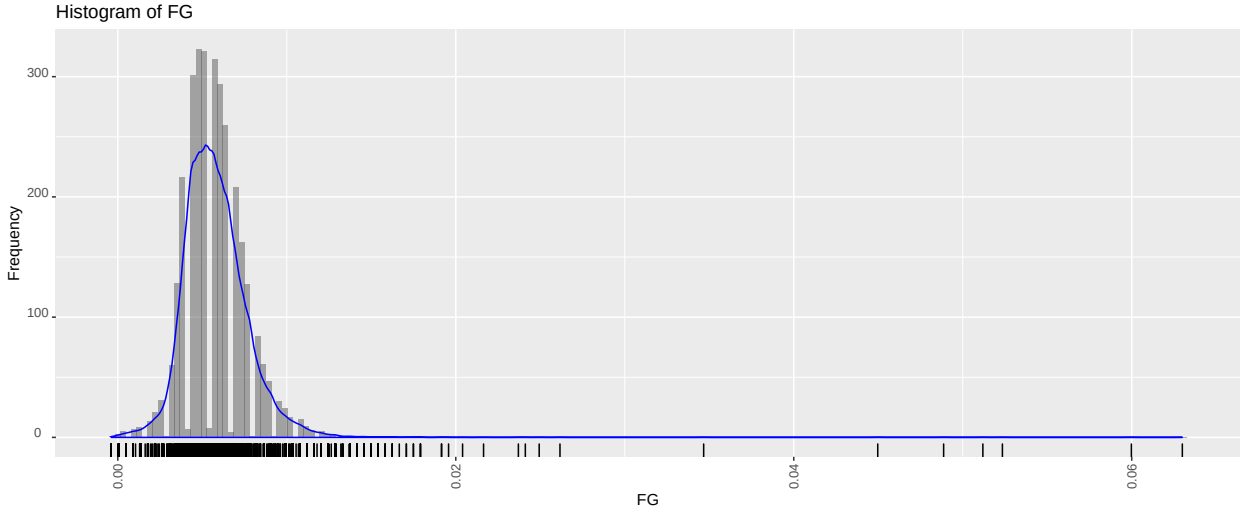
```
        print(p)
    }
}

num_cols <- c("Color", "sizeL", "BoilSize", "OG", "FG", "ABV", "IBU", "BoilTime",
              "BoilGravity", "Efficiency")

hist_plot_numeric_vars(recipe_top10_log10, num_cols)
```

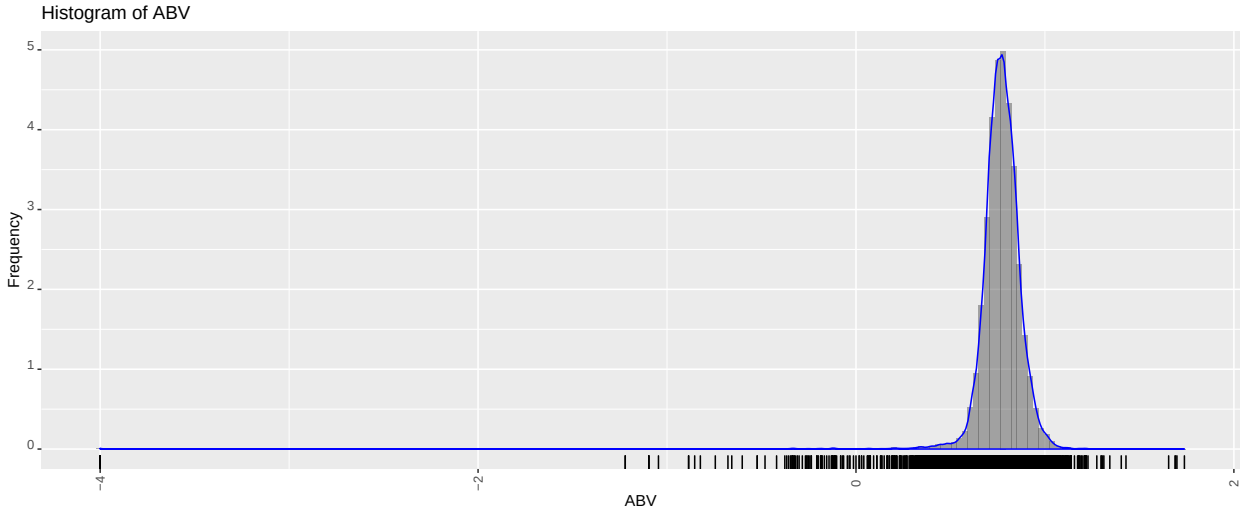


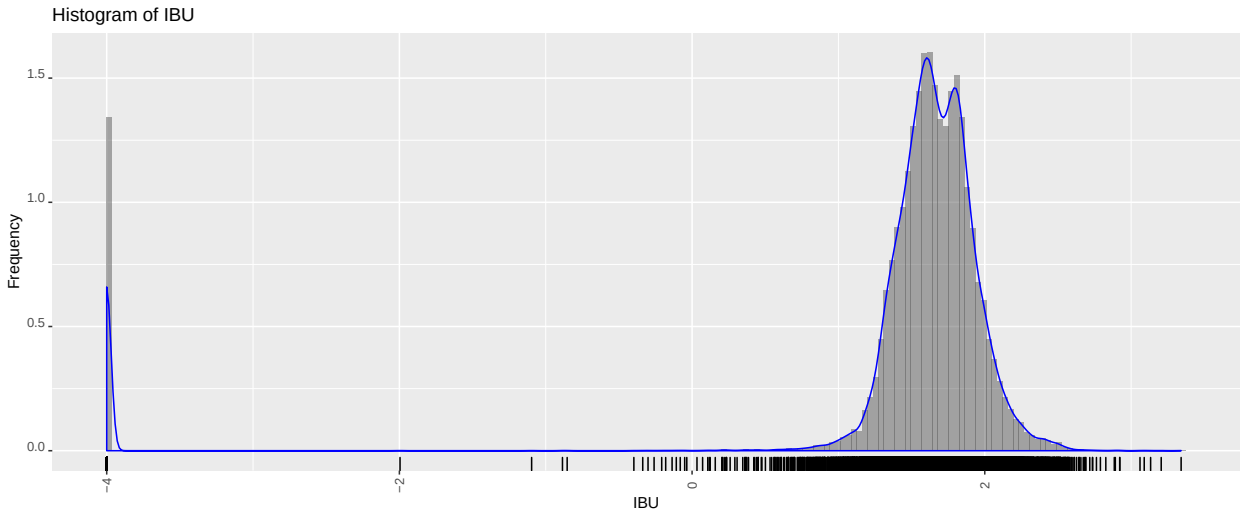


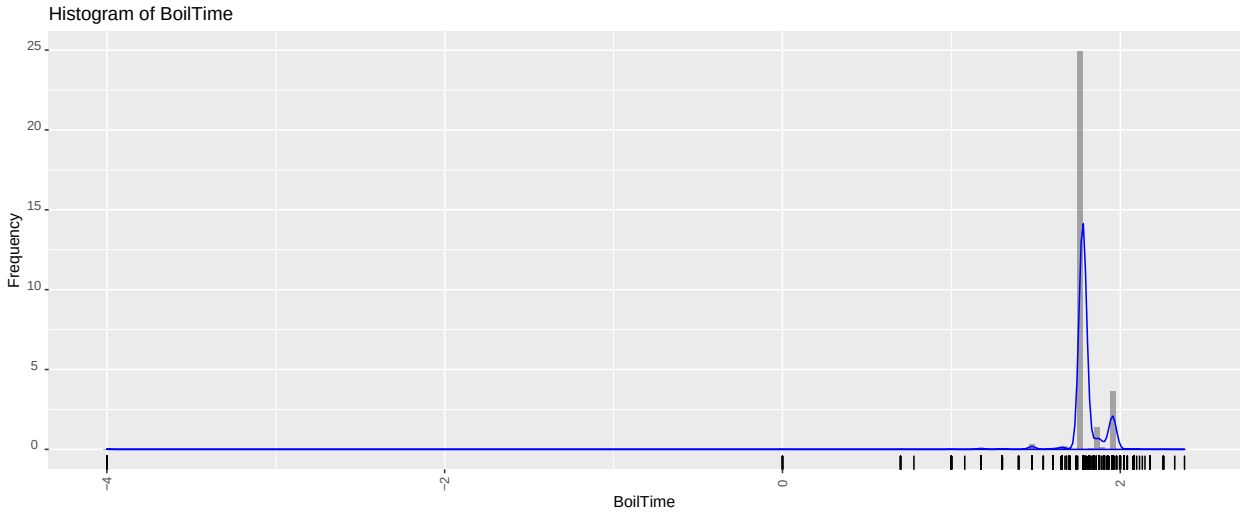

Histogram of BoilSize
Frequency
12.5
10.0
7.5
5.0
2.5
0.0
0
1
2
3
4
BoilSize

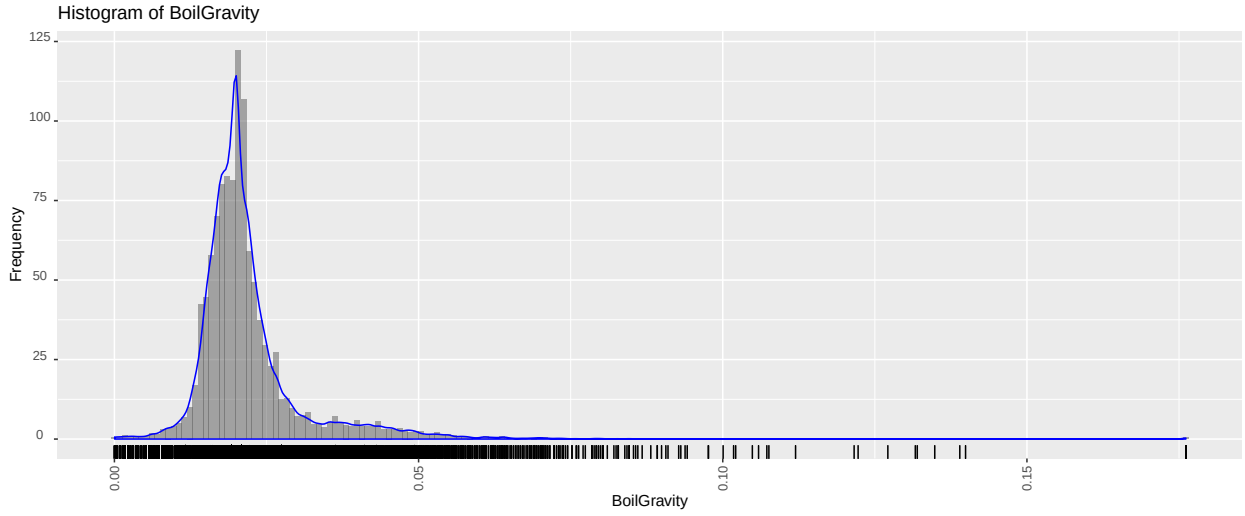

Histogram of OG
Frequency
100
75
50
25
0
0.00
0.05
0.10
0.15
OG

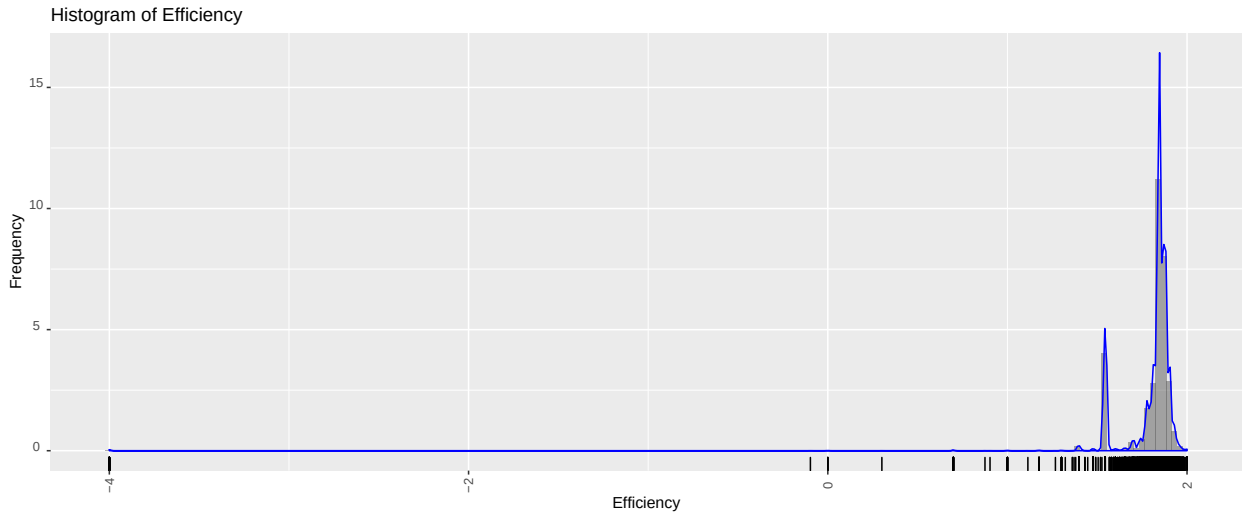

Histogram of FG
Frequency
300
200
100
0
0.00
0.02
0.04
0.06
FG


Histogram of ABV
Frequency
ABV


Histogram of IBU
Frequency
IBU


Histogram of BoilTime
Frequency
BoilTime



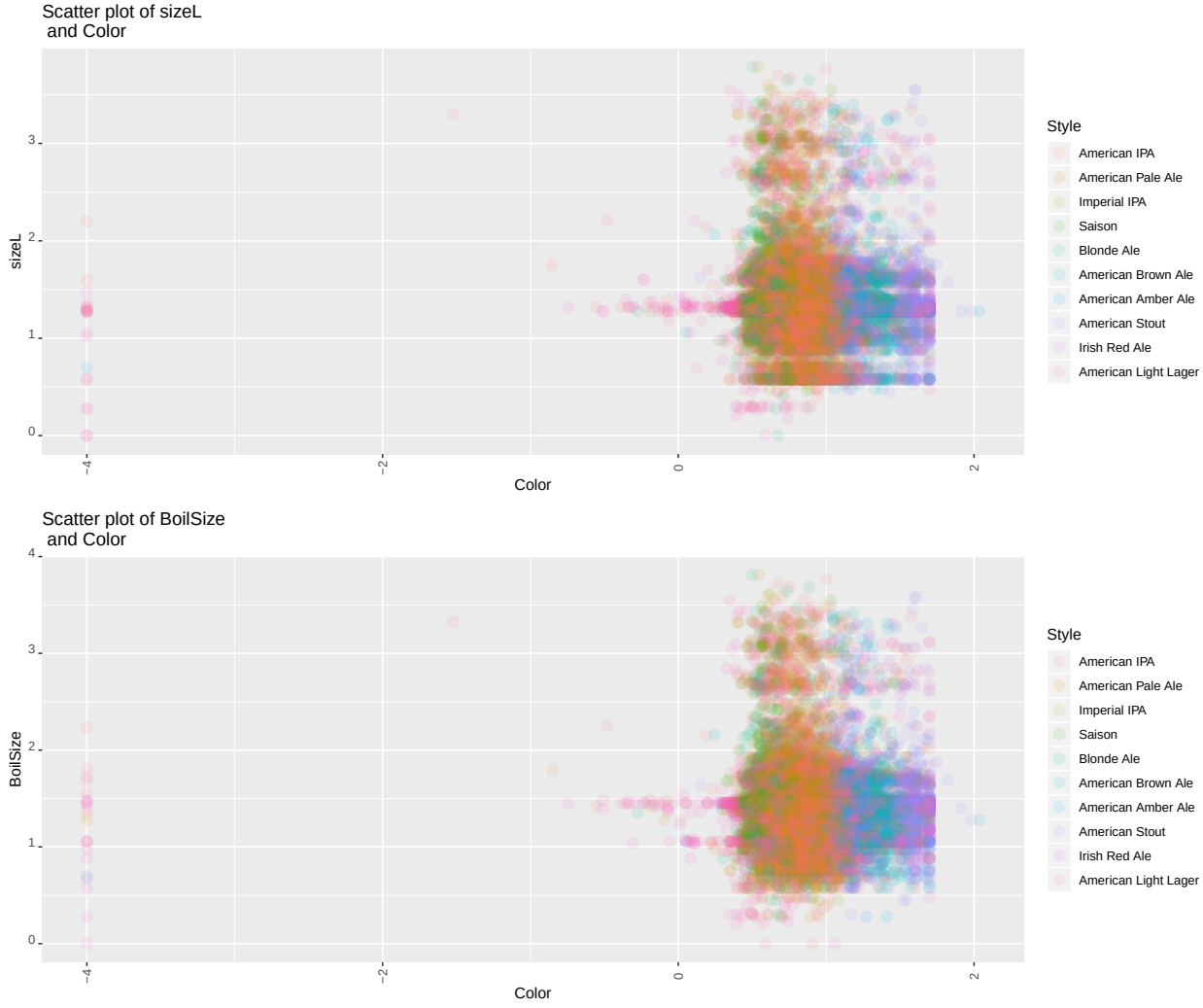


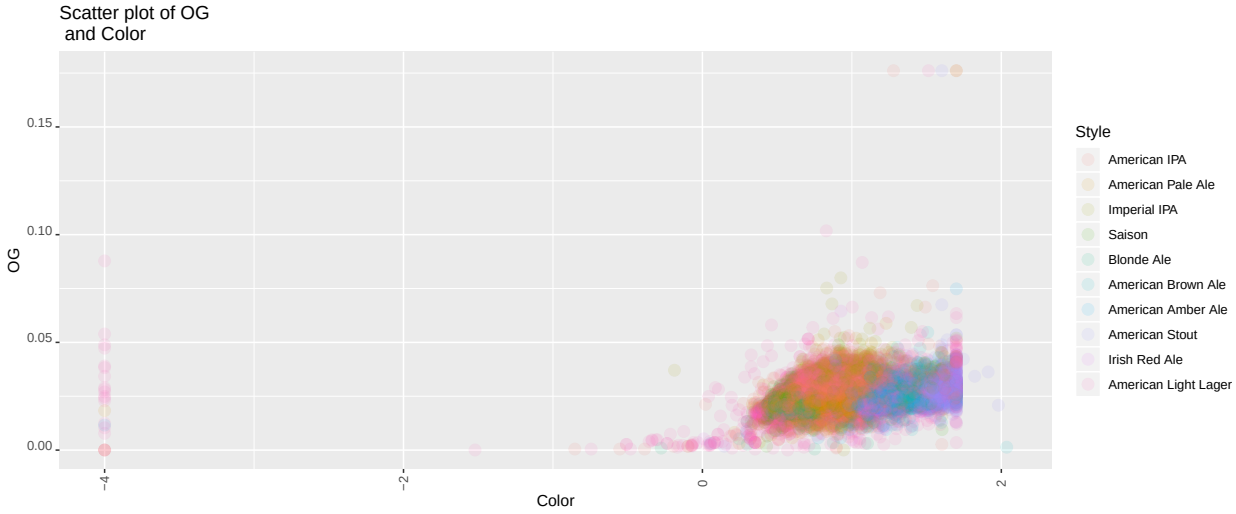
4.8 Scatter plot of transformed dataset

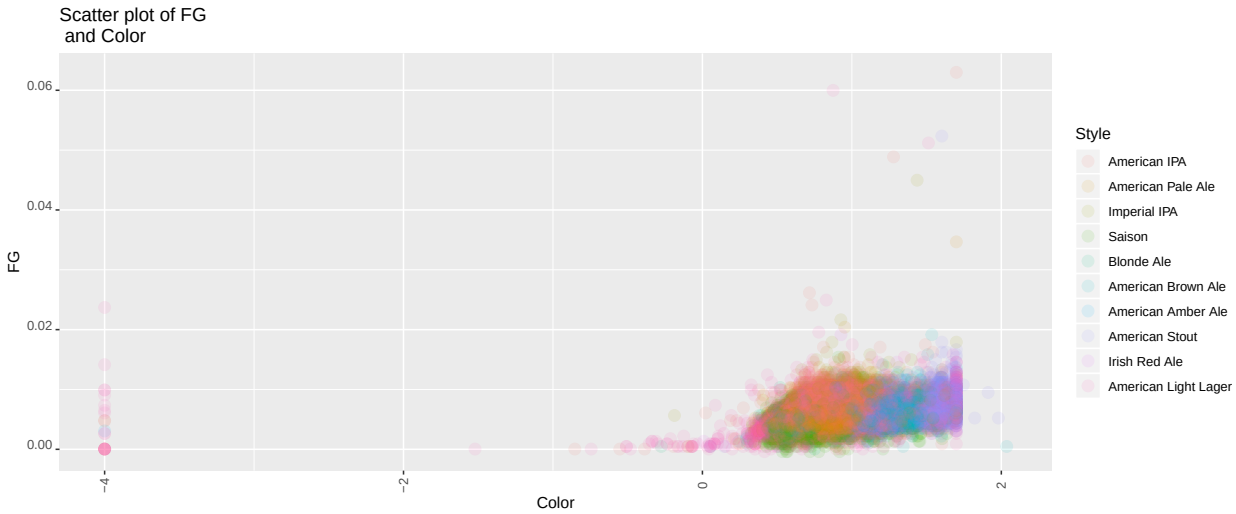
Scatter plots have shown how "Color" and other variables help to separate ten popular beer styles. The result of those scatter plots point out that Style can not be classified by these simple indicators.

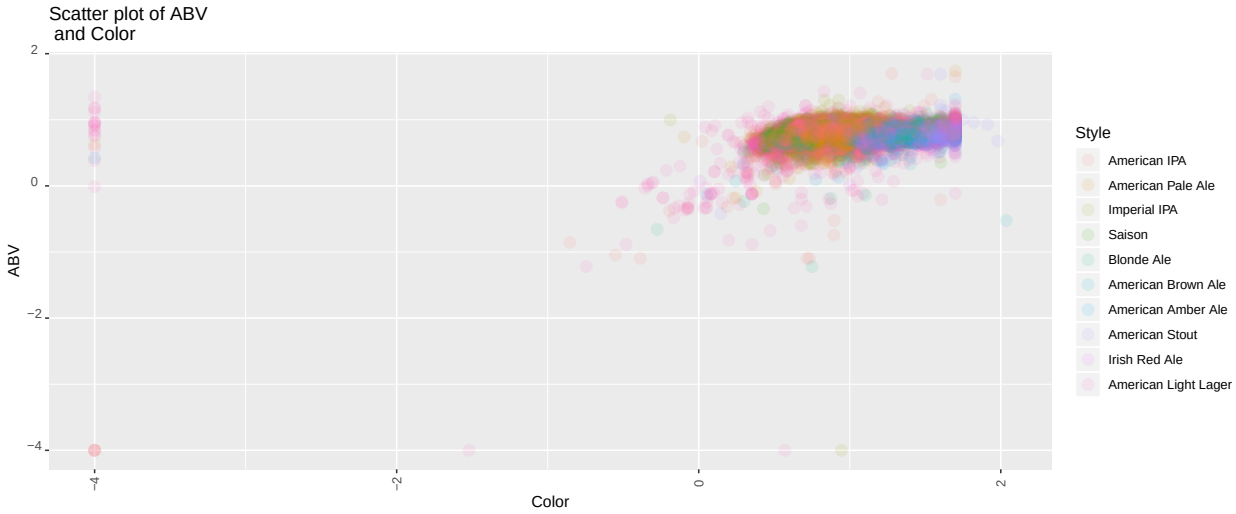
```
# Scatter plot of transformed dataset
scatter_plot_numeric_vars <- function(df, cols, col_x = "Color"){
    options(repr.plot.width=4, repr.plot.height=3.5) # Set the initial plot area dimensions
    for(col in cols){
        p <- ggplot(df, aes_string(col_x, col)) +
                  geom_jitter(aes(col = Style), alpha = 0.1, size = 3.5) +
                  theme(axis.text.x = element_text(angle = 90, hjust = 0.5, vjust = 0.5),
                        axis.text.y = element_text(angle = 0, hjust = 0.5, vjust = 0)) +
                  labs(title = paste('Scatter plot of', col, '\n and', col_x),
                       x = "Color",
                       y = col)

        print(p)
    }
}
```

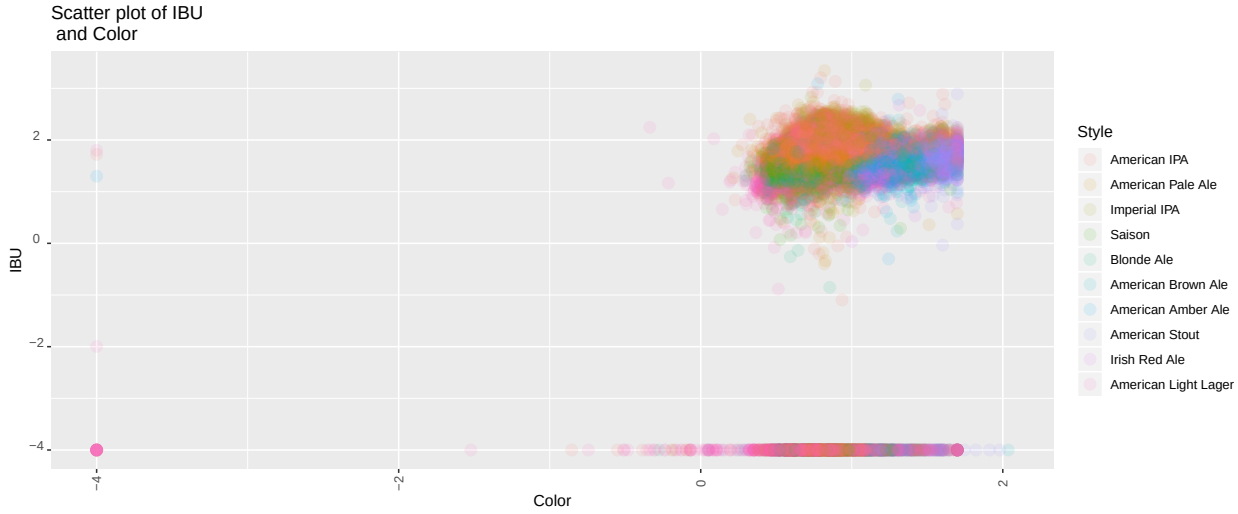
```
num_cols <- c("sizeL", "BoilSize", "OG", "FG", "ABV", "IBU", "BoilTime",
              "BoilGravity", "Efficiency")

scatter_plot_numeric_vars(recipe_top10_log10, num_cols)
```

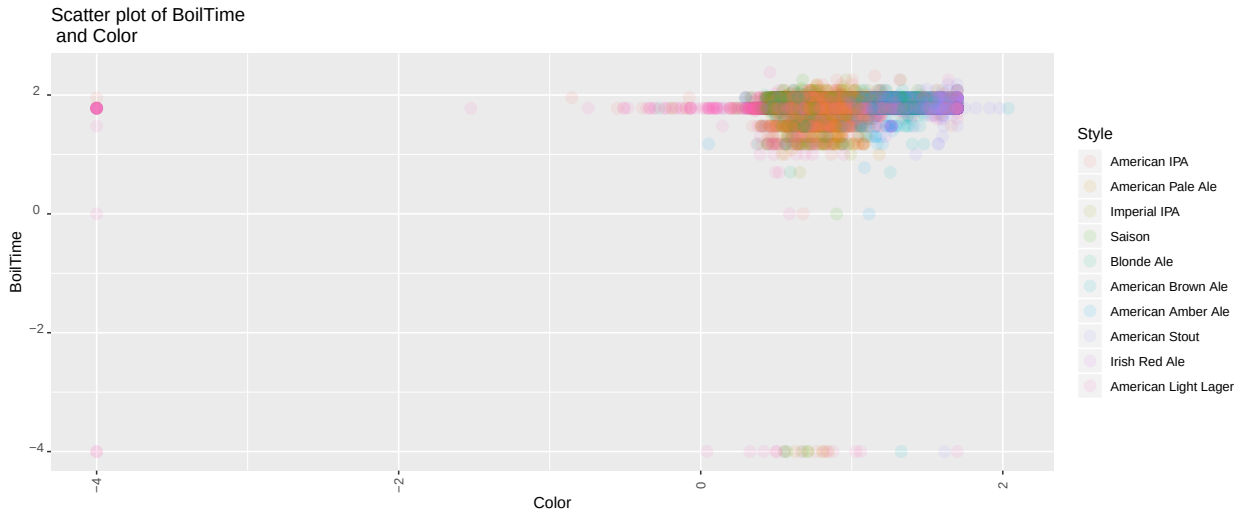


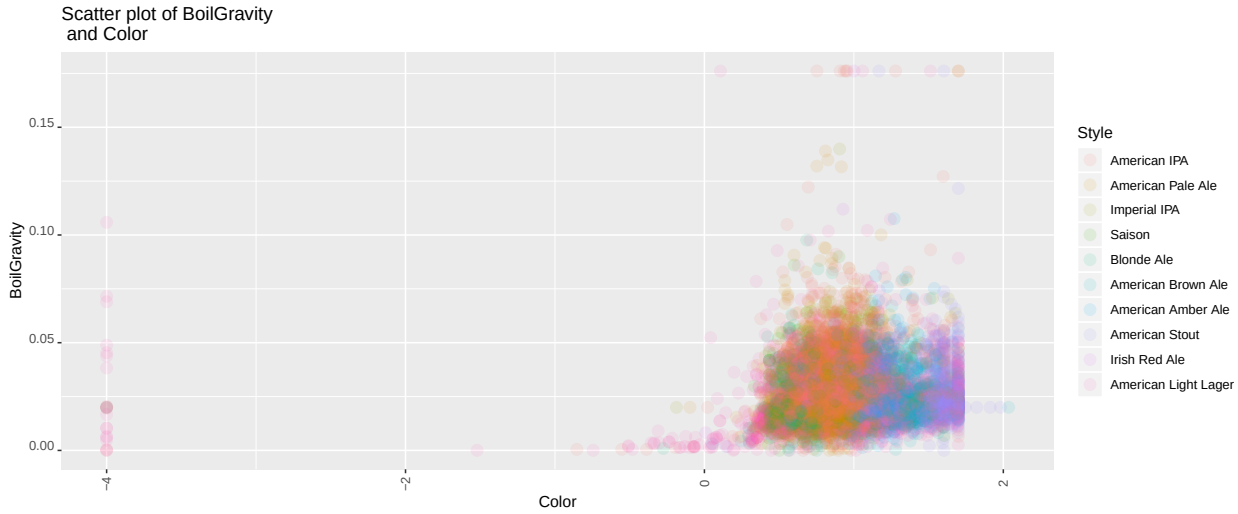

Scatter plot of OG and Color
OG
0.15
0.10
0.05
0.00
−4
−2
0
2
Color
Style
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager

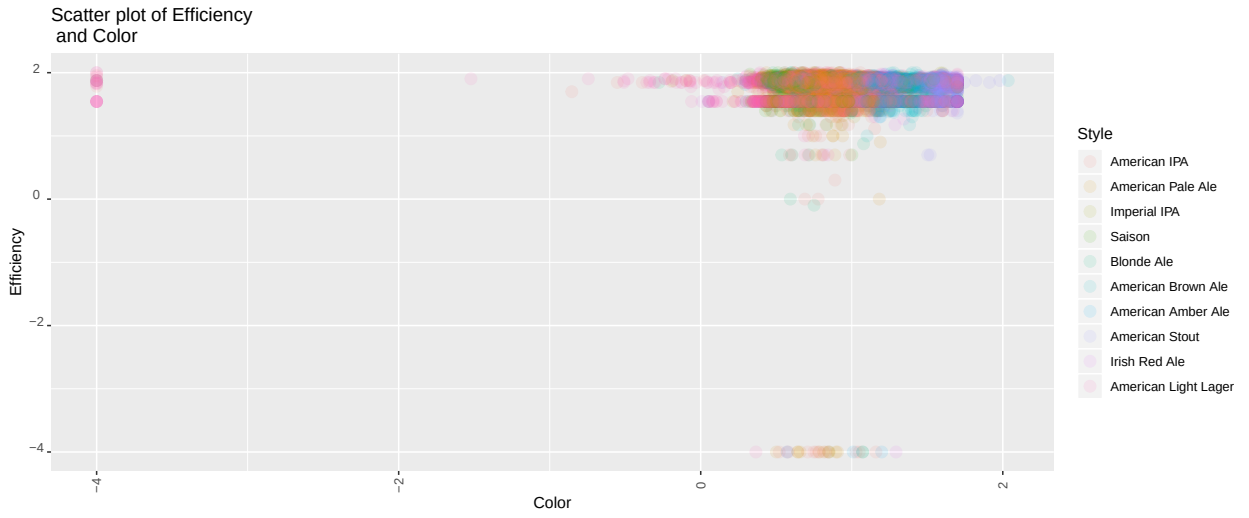

Scatter plot of FG and Color
FG
0.06
0.04
0.02
0.00
−4
−2
0
2
Color
Style
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager


Scatter plot of ABV and Color
ABV
2
0
−2
−4
−4
−2
0
2
Color
Style
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager


Scatter plot of IBU
and Color
IBU
2
0
−2
−4
−4
−2
0
2
Color
Style
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager


Scatter plot of BoilTime
and Color
BoilTime
2
0
−2
−4
−4
−2
0
2
Color
Style
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager

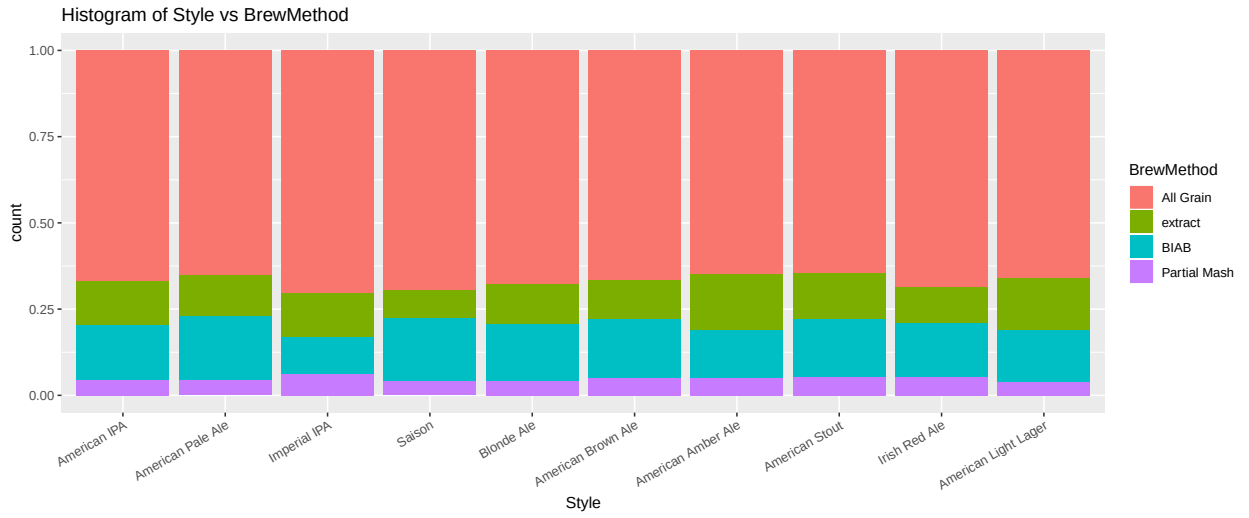

Scatter plot of BoilGravity
and Color
BoilGravity
0.15
0.10
0.05
0.00
−4
−2
0
2
Color
Style
American IPA
American Pale Ale
Imperial IPA
Saison
Blonde Ale
American Brown Ale
American Amber Ale
American Stout
Irish Red Ale
American Light Lager



4.9 Proportion of different brewing methods on Style

The unique categorical variable in the dataset is unbalance. There are more than fifty percent of all Beer styles brewed by All Grain method. Partial Mash method is not prefered by homebrews where it has only about 2-3 percent in total.

```
# Proportion of different brewing methods on Style
recipe_top10_log10 %>%
  ggplot(aes(x = Style, fill = BrewMethod)) +
   geom_bar(position = "fill") +
   ggtitle("Histogram of Style vs BrewMethod") +
   theme(axis.text.x = element_text(angle = 30, hjust = 1))
```



4.10 The proportions of Beer styles

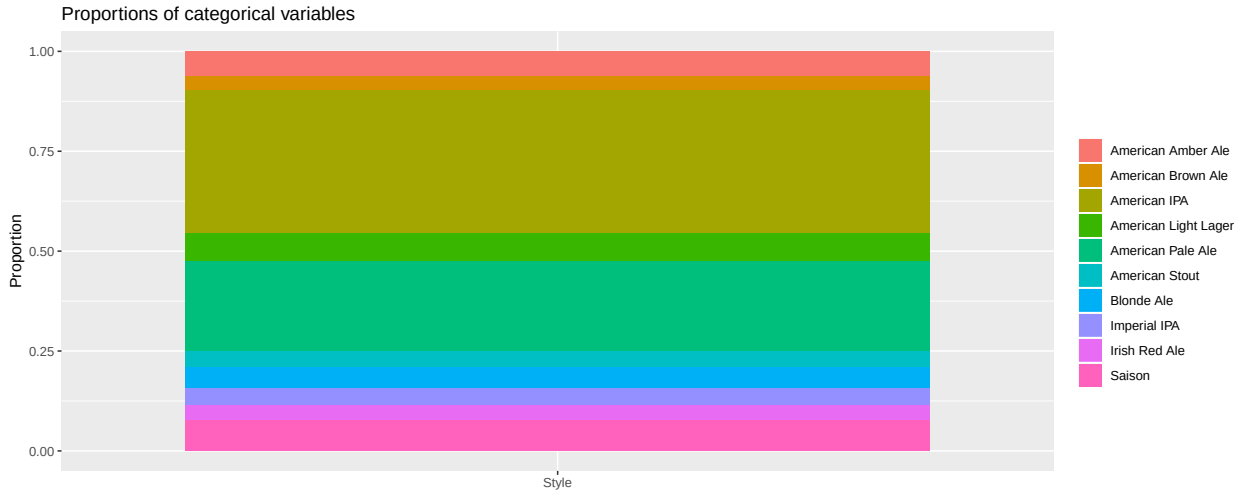
The graph shows that the project is facing classification problem with the unbalanced dataset. For example, American IPA and American PAle Ale contributed to approximately 60 percent of the total observation of 10 beer styles, and it lefts only about 40 percent to other eight styles.

```
# Proportions of Beer Styles
recipe_top10_log10 %>% group_by(Style) %>%
```

```
  summarize(n_style = n(), prop_style = n_style/nrow(recipe_top10_log10)) %>%
  arrange(desc(prop_style))
```

```
## # A tibble: 10 x 3
##    Style                n_style prop_style
##    <fct>                  <int>      <dbl>
##  1 American IPA           11940     0.358
##  2 American Pale Ale       7581     0.228
##  3 Saison                  2617     0.0786
##  4 American Light Lager    2277     0.0684
##  5 American Amber Ale      2038     0.0612
##  6 Blonde Ale              1753     0.0526
##  7 Imperial IPA            1478     0.0444
##  8 American Stout          1268     0.0381
##  9 Irish Red Ale           1204     0.0361
## 10 American Brown Ale      1152     0.0346
```

```
# Visualize proportions of beer styles
recipe_top10_log10 %>%
  select(Style) %>%
  gather("cols", "sets") %>%
  ggplot(aes(x = cols, fill = sets)) +
  geom_bar(position = "fill") +
  labs(title = "Proportions of categorical variables",
       x = "",
       y = "Proportion") +
  guides(fill = guide_legend(title=""))
```



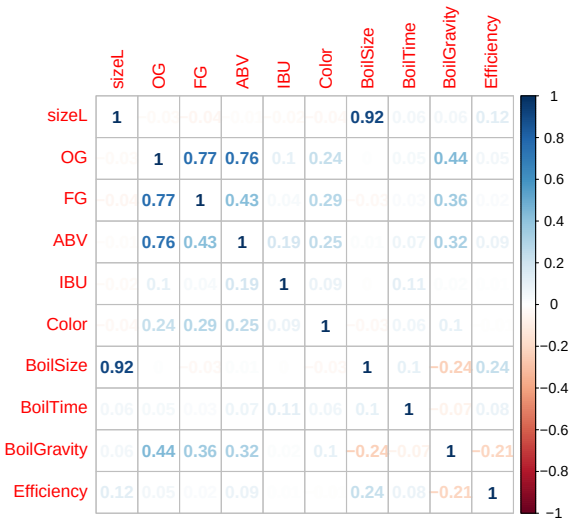
4.11 Correlation matrices of all numerical variables

There was a strong positive correlation (0.92) between "sizeL" and "BoilSize", so it would be reasonable to drop out "sizeL" variable. In addition, 0.77 and 0.76 show the moderate positive correlation results in both cases including OG versuses FG and OG versuses ABV, but the project decided to to keep them in the dataset for modelling step.

```
# visualize correlation matrices
if(!require(corrplot))
  install.packages("corrplot", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: corrplot

## Warning: package 'corrplot' was built under R version 3.5.3

## corrplot 0.84 loaded
```

```
# Explore numeric features
numeric_vars <- recipe_top10_log10 %>%
  select(which(sapply(., class) != "factor"))

corrplot(cor(numeric_vars, use = "complete.obs"), method = "number")
```



- Drop "sizeL"

```
# Drop "sizeL"
recipe_top10_log10_tidy <- recipe_top10_log10 %>% select(- sizeL)

# Save "recipe_top10_log10_tidy" dataset
save(recipe_top10_log10_tidy, file = "recipe_top10_log10_tidy.RData")
```

5 Modelling approaches

The project will run four different machine learning algorithms including decision trees, random forest, support vector machine, neuron network, naivebayes and xtreme boosted tree gradient.

The expectation is to find the best model that have the highest accuracy rate.

The dataset used in the modelling step would be "recipe_top10_log10_tidy". It contains 33,308 observations and 11 variables. The explanatory varibles have 9 numerical and 1 categorical ones.

```
# Glance at the tidy dataset
glimpse(recipe_top10_log10_tidy)
```

```
## Observations: 33,308
## Variables: 11
## $ Style       <fct> American IPA, American IPA, American Pale Ale, Imp...
## $ OG          <dbl> 0.02657412, 0.02575631, 0.02329362, 0.03023530, 0....
## $ FG          <dbl> 0.007790437, 0.007363654, 0.005652315, 0.007790437...
## $ ABV         <dbl> 0.7715948, 0.7634355, 0.7466420, 0.8506524, 0.9148...
## $ IBU         <dbl> 1.772689, 1.736238, 1.603362, 2.429284, 1.968577, ...
```

```
## $ Color       <dbl> 0.9532812, 0.9294240, 0.9030954, 0.8014106, 0.9185...
## $ BoilSize    <dbl> 1.356219, 1.423248, 1.467462, 1.481157, 1.453167, ...
## $ BoilTime    <dbl> 1.778152, 1.778152, 1.845099, 1.954243, 1.778152, ...
## $ BoilGravity <dbl> 0.01998816, 0.01998816, 0.01998816, 0.01998816, 0....
## $ Efficiency  <dbl> 1.845099, 1.845099, 1.897628, 1.875062, 1.845099, ...
## $ BrewMethod  <fct> extract, All Grain, All Grain, All Grain, All Grai...
```

5.1 Create training data for modelling approaches and testing data for validation step

```
# Load caret package
library(caret)

# Create train_data and test_data
set.seed(2019)

test_index <- createDataPartition(recipe_top10_log10_tidy$Style,
                                  time = 1,
                                  p = 0.5,
                                  list = FALSE)

train_data <- recipe_top10_log10_tidy %>% slice(test_index)
test_data <- recipe_top10_log10_tidy %>% slice(-test_index)

# Check the dimension of train_data and test_data
dim(train_data)
```

```
## [1] 16656    11
```

```
dim(test_data)
```

```
## [1] 16652    11
```

5.2 Scaling the numeric variables

```
# Scaling the numeric variables
num_cols <- c("Color", "BoilSize", "OG",
              "FG", "ABV", "IBU",
              "BoilTime", "BoilGravity", "Efficiency")

preprocess_values <- preProcess(train_data[,num_cols], method = c("center", "scale"))

train_data[,num_cols] <- predict(preprocess_values, train_data[,num_cols])


test_data[,num_cols] <- predict(preprocess_values, test_data[,num_cols])

# Save the training data and testing data for the modelling step
save(train_data, file = "train_data_beerrecipe.RData")
save(test_data, file = "test_data_beerrecipe.RData")

# Glance at the scaled train_data
head(train_data[,num_cols])
```

```
## # A tibble: 6 x 9
```

```
##     Color BoilSize     OG      FG     ABV   IBU BoilTime BoilGravity
##     <dbl>    <dbl>  <dbl>   <dbl>   <dbl> <dbl>    <dbl>       <dbl>
## 1  0.130  -0.210    0.275  1.03    0.0254 0.308  -0.0607      -0.228
## 2  0.0515 -0.00625  0.130  0.813  -0.0392 0.279  -0.0607      -0.228
## 3 -0.0355  0.128   -0.307 -0.0686 -0.172  0.174   0.280       -0.228
## 4 -0.372   0.170    0.924  1.03    0.652  0.826   0.835       -0.228
## 5  0.0156  0.0849   1.50   0.813   1.16   0.463  -0.0607       0.245
## 6 -0.0756  0.113    0.347  0.152   0.421  0.336   0.835        0.116
## # ... with 1 more variable: Efficiency <dbl>
```

5.3 Naive Bayes approach

5.3.1 Naive Bayes approach using naive_bayes method in caret package with imbalanced dataset

```
# Remove and free up working space
rm(list = ls(all.names = TRUE))
gc()
```

```
##           used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 2268802 121.2    3519178 188.0  3519178 188.0
## Vcells 4225351  32.3   17175091 131.1 21468864 163.8
```

```
# Load necessary packages
if(!require(tidyverse))install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(tictoc))install.packages("tictoc", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tictoc
```

```
if(!require(caret))install.packages("caret", repos = "http://cran.us.r-project.org")

# Load the datasets
tic()
load("train_data_beerrecipe.RData")


# Set up cross validation method for the Naive Bayes model
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,  ## 5-fold CV
                           repeats = 3, ## repeated three times
                           p = 0.8)

# Run the Naive Bayes model
set.seed(201955)

model_nb <- train(Style ~ .,
           data = train_data,
           method = "naive_bayes",
           trControl = fitControl,
           verbose= FALSE)


# Result of Naive Bayes approach
max_accuracy_model_nb <- model_nb$results %>% filter(Accuracy == max(Accuracy))
max_accuracy_model_nb
```

```
##   usekernel laplace adjust Accuracy     Kappa  AccuracySD     KappaSD
## 1      TRUE       0      1 0.570025 0.4416909 0.004611291 0.006731889
```

```
# Save the maximum of accuracy to conduct Table of results
nb_accuracy <- max(model_nb$results[4])
nb_accuracy
```

```
## [1] 0.570025
```

```
save(nb_accuracy, file = "nb_accuracy.RData")

# Memory usage of running Naive Bayes model
memory.size()
```

```
## [1] 276.05
```

```
# Time spending in running Naive Bayes model
runtime <- toc()
```

```
## 5.77 sec elapsed
```

```
nb_runtime <- runtime$toc - runtime$tic
save(nb_runtime, file = "nb_runtime.RData")
```

5.3.2 Naive Bayes approach using naive_bayes method and "up" sampling in caret package

```
# Remove and free up working space
rm(list = ls(all.names = TRUE))
gc()
```

```
##           used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 2314483 123.7    3519178 188.0  3519178 188.0
## Vcells 4321845  33.0   17176795 131.1 21468864 163.8
```

```
# Load necessary packages
if(!require(tidyverse))install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(tictoc))install.packages("tictoc", repos = "http://cran.us.r-project.org")
if(!require(caret))install.packages("caret", repos = "http://cran.us.r-project.org")

# Load the datasets
tic()
load("train_data_beerrecipe.RData")


# Set up cross validation method for the Naive Bayes model
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,  ## 5-fold CV
                           repeats = 3, ## repeated three times
                           p = 0.8,
                           sampling = "up" ## randomly subset all the classes to match the
                                           ## frequencies of the higest prevalent class
                           )

# Run the Naive Bayes model with "up" sampling method
set.seed(201955)

model_nb_upsampling <- train(Style ~ .,
```

```
            data = train_data,
            method = "naive_bayes",
            trControl = fitControl,
            verbose= FALSE)

# Result of Naive Bayes approach with "up" sampling method
max_accuracy_model_nb_upsampling <- model_nb_upsampling$results %>%
  filter(Accuracy == max(Accuracy))

max_accuracy_model_nb_upsampling
```

```
##   usekernel laplace adjust  Accuracy     Kappa AccuracySD    KappaSD
## 1      TRUE       0      1 0.5068842 0.4146242 0.01252885 0.01256412
```

```
# Save the maximum of accuracy to conduct Table of results
nb_upsampling_accuracy <- max(model_nb_upsampling$results[4])
nb_upsampling_accuracy
```

```
## [1] 0.5068842
```

```
save(nb_upsampling_accuracy, file = "nb_upsampling_accuracy.RData")

# Memory usage of running Naive Bayes model with "up" sampling method
memory.size()
```

```
## [1] 272.67
```

```
# Time spending in running Naive Bayes model with "up" sampling method
runtime <- toc()
```

```
## 9.82 sec elapsed
```

```
nb_upsampling_runtime <- runtime$toc - runtime$tic
save(nb_upsampling_runtime, file = "nb_upsampling_runtime.RData")
```

5.4 Decision tree approach

5.4.1 Decision tree approach using rpart method in caret package with imbalanced dataset

```
# Remove and free up working space
rm(list = ls(all.names = TRUE))
gc()
```

```
##           used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 2315413 123.7    3519178 188.0  3519178 188.0
## Vcells 4326583  33.1   17183161 131.1 21468864 163.8
```

```
# Load necessary packages
if(!require(tidyverse))install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(tictoc))install.packages("tictoc", repos = "http://cran.us.r-project.org")
if(!require(caret))install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(rpart))install.packages("rpart", repos = "http://cran.us.r-project.org")
```

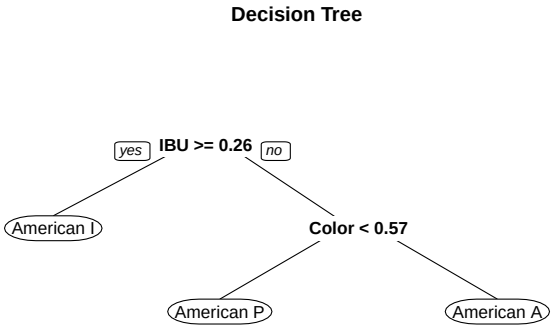
```
## Loading required package: rpart
```

```
if(!require(rpart.plot))install.packages("rpart.plot", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: rpart.plot
```

```
## Warning: package 'rpart.plot' was built under R version 3.5.3
```

```
# Load the datasets
tic()
load("train_data_beerrecipe.RData")


# Set up cross validation method for the Decision tree model
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,  ## 5-fold CV
                           repeats = 3, ## repeated three times
                           p = 0.8,
                           returnResamp="all")

# Run the Decision tree model
set.seed(201955)

model_rpart <- train(Style ~ .,
           data = train_data,
           method = "rpart",
           trControl = fitControl)

# Plot the decision tree
rpart.plot.version1(model_rpart$finalModel)
title("Decision Tree")
```




```
# Result of Decision tree approach
max_accuracy_model_rpart <- model_rpart$results %>% filter(Accuracy == max(Accuracy))
max_accuracy_model_rpart
```

```
##           cp  Accuracy     Kappa  AccuracySD    KappaSD
## 1 0.03256597 0.4850697 0.3070498 0.009102972 0.01429092
```

```
# Save the maximum of accuracy to conduct Table of results
rpart_accuracy <- max(model_rpart$results[2])
save(rpart_accuracy, file = "rpart_accuracy.RData")
rpart_accuracy
```

```
## [1] 0.4850697
```

```
# Memory usage of running Decision tree model
memory.size()
```

```
## [1] 249.96
```

```
# Time spending in running Decision tree model
runtime <- toc()
```

```
## 3.64 sec elapsed
```

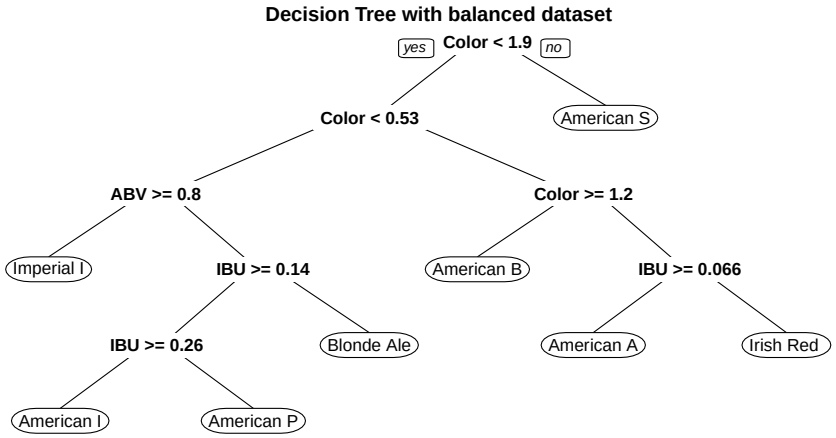
```
rpart_runtime <- runtime$toc - runtime$tic
save(rpart_runtime, file = "rpart_runtime.RData")
```

5.4.2 Decision tree approach using rpart method in caret package with balanced dataset

```
# Remove and free up working space
rm(list = ls(all.names = TRUE))
gc()
```

```
##           used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells 2375778 126.9    3519178 188.0  3519178 188.0
## Vcells 4452627  34.0   17183161 131.1 21468864 163.8
```

```
# Load necessary packages
if(!require(tidyverse))install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(tictoc))install.packages("tictoc", repos = "http://cran.us.r-project.org")
if(!require(caret))install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(rpart))install.packages("rpart", repos = "http://cran.us.r-project.org")
if(!require(rpart.plot))install.packages("rpart.plot", repos = "http://cran.us.r-project.org")

# Load the datasets
tic()
load("train_data_beerrecipe.RData")

# Set up cross validation method for the Decision tree model
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,  ## 5-fold CV
                           repeats = 3, ## repeated three times
                           p = 0.8,
                           sampling = "up", ## randomly subset all the classes to match the
                                            ## frequencies of the higest prevalent class
                           returnResamp="all")

# Run the Decision tree model  with "up" sampling method
set.seed(201955)

model_rpart_upsampling <- train(Style ~ .,
           data = train_data,
           method = "rpart",
           trControl = fitControl)

# Plot the decision tree
rpart.plot.version1(model_rpart_upsampling$finalModel)
title("Decision Tree with balanced dataset")
```




```
# Result of Decision tree approach with "up" sampling method
max_accuracy_model_rpart_upsampling <- model_rpart_upsampling$results %>%
  filter(Accuracy == max(Accuracy))

max_accuracy_model_rpart_upsampling
```

```
##           cp  Accuracy     Kappa AccuracySD    KappaSD
## 1 0.03256597 0.4882931 0.3875907 0.01738334 0.02408783
```

```
# Save the maximun of accuracy to conduct Table of results
rpart_upsampling_accuracy <- max(model_rpart_upsampling$results[2])
save(rpart_upsampling_accuracy, file = "rpart_upsampling_accuracy.RData")
rpart_upsampling_accuracy
```

```
## [1] 0.4882931
```

```
# Memory usage of running Decision tree model with "up" sampling method
memory.size()
```

```
## [1] 324.5
```

```
# Time spending in running Decision tree model with "up" sampling method
runtime <- toc()
```

```
## 9.03 sec elapsed
```

```
rpart_upsampling_runtime <- runtime$toc - runtime$tic
save(rpart_upsampling_runtime, file = "rpart_upsampling_runtime.RData")
```

5.5 Ensemble method for decision trees

There are two main techniques in ensemble method including Bagging and Boosting, and the project will use Randomforest and AdaBoost Classification Trees methods to test those techniques respectively.

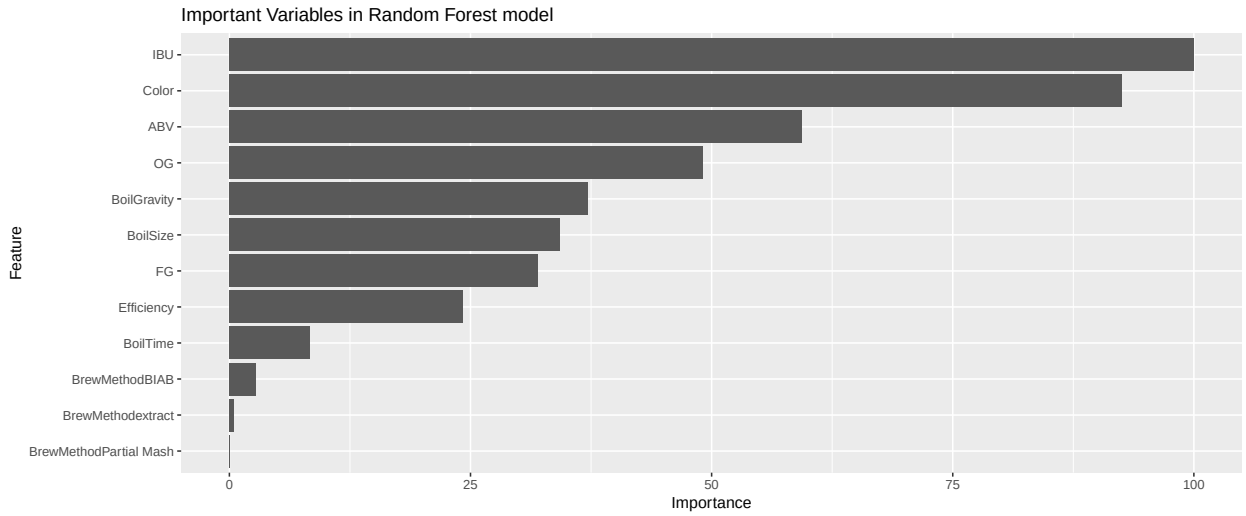
5.5.1 Bagging technique

5.5.1.1 Random forest using “rf” method in caret package

```
# Remove and free up working space
rm(list = ls(all.names = TRUE))
gc()
```

```
##            used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells  2375808 126.9    4263013 227.7  4141960 221.3
## Vcells  4454861  34.0   17184685 131.2 21468864 163.8
```

```r
# Load necessary packages
if(!require(tidyverse))install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(tictoc))install.packages("tictoc", repos = "http://cran.us.r-project.org")
if(!require(caret))install.packages("caret", repos = "http://cran.us.r-project.org")

# Load the datasets
tic()
load("train_data_beerrecipe.RData")

# Set up cross validation method for the random forest model
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,  ## 5-fold CV
                           repeats = 3, ## repeated three times
                           p = 0.8,
                           returnResamp="all"
                           )

# Run the random forest model
set.seed(201955)
model_rf <- train(Style ~ .,
           data = train_data,
           method = "rf",
           trControl = fitControl,
           verbose= FALSE)

# Show variable important score in the random forest model
var_imp_rf <- varImp(model_rf)

# plot the random forest model
ggplot(var_imp_rf) +
  ggtitle("Important Variables in Random Forest model")
```




```r
# Result of random forest approach
max_accuracy_model_rf <- model_rf$results %>% filter(Accuracy == max(Accuracy))
```

```
max_accuracy_model_rf
```

```
##   mtry  Accuracy     Kappa  AccuracySD    KappaSD
## 1    2 0.6259412 0.5180277 0.008763206 0.01116293
```

```
# Save the maximum of accuracy to conduct Table of results
rf_accuracy <- max(model_rf$results[2])
save(rf_accuracy, file = "rf_accuracy.RData")
rf_accuracy
```

```
## [1] 0.6259412
```

```
# Memory usage of running random forest model
memory.size()
```

```
## [1] 1077.11
```

```
# Time spending in running random forest model
runtime <- toc()
```

```
## 802.97 sec elapsed
```

```
rf_runtime <- runtime$toc - runtime$tic
save(rf_runtime, file = "rf_runtime.RData")
```

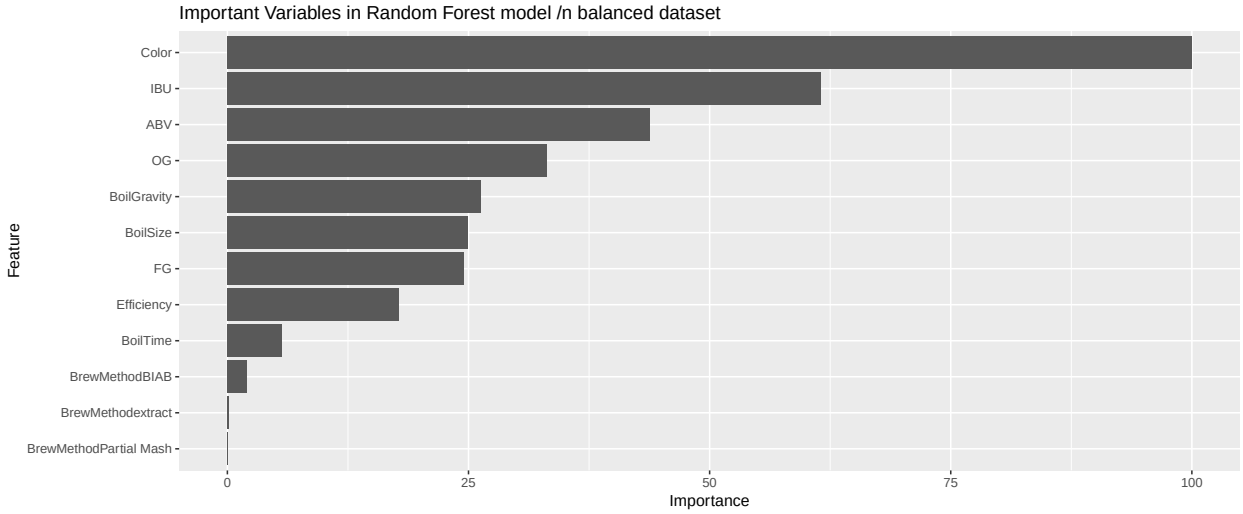
5.5.1.2 Random forest using "rf" method and "up" sampling in caret package

```
# Remove and free up working space
rm(list = ls(all.names = TRUE))
gc()
```

```
##           used  (Mb) gc trigger  (Mb)  max used  (Mb)
## Ncells 2385983 127.5    4263013 227.7   4141960 221.3
## Vcells 4416213  33.7   95219556 726.5 116623107 889.8
```

```
# Load necessary packages
if(!require(tidyverse))install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(tictoc))install.packages("tictoc", repos = "http://cran.us.r-project.org")
if(!require(caret))install.packages("caret", repos = "http://cran.us.r-project.org")

# Load the datasets
tic()
load("train_data_beerrecipe.RData")

# Set up cross validation method for the Random forest model
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,  ## 5-fold CV
                           repeats = 3, ## repeated three times
                           p = 0.8,
                           returnResamp="all",
                           sampling = "up" ## randomly subset all the classes to match the
                                           ## frequencies of the higest prevalent class
                           )

# Run the Random forest model  with "up" sampling method
set.seed(201955)
model_rf_upsampling <- train(Style ~ .,
           data = train_data,
```

```
            method = "rf",
            trControl = fitControl,
            verbose= FALSE)


# Show variable important score in the random forest model
var_imp_rf_upsampling <- varImp(model_rf_upsampling)

# plot the Random forest model
ggplot(var_imp_rf_upsampling) +
  ggtitle("Important Variables in Random Forest model /n balanced dataset")
```




```
# Result of Random forest approach with "up" sampling method
max_accuracy_model_rf_upsampling <- model_rf_upsampling$results %>%
  filter(Accuracy == max(Accuracy))

max_accuracy_model_rf_upsampling
```

```
##   mtry  Accuracy     Kappa  AccuracySD     KappaSD
## 1    2 0.6199171 0.5255325 0.007925267 0.009478505
```

```
# Save the maximum of accuracy to conduct Table of results
rf_upsampling_accuracy <- max(model_rf_upsampling$results[2])
save(rf_upsampling_accuracy, file = "rf_upsampling_accuracy.RData")
rf_upsampling_accuracy
```

```
## [1] 0.6199171
```

```
# Memory usage of running Random forest model with "up" sampling method
memory.size()
```

```
## [1] 2963.65
```

```
# Time spending in running Random forest model with "up" sampling method
runtime <- toc()
```

```
## 2815.11 sec elapsed
```

```
rf_upsampling_runtime <- runtime$toc - runtime$tic
save(rf_upsampling_runtime, file = "rf_upsampling_runtime.RData")
```

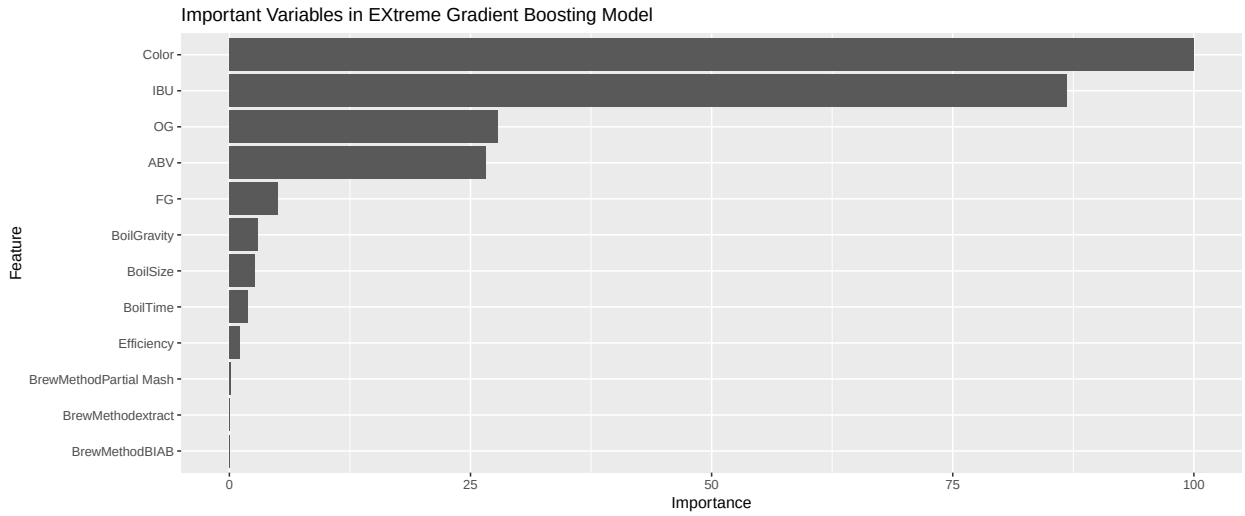
5.5.2 Boosting technique

5.5.2.1 EXtreme Gradient Boosting model using "xgbTree" method in caret package

```
# Remove and free up working space
rm(list = ls(all.names = TRUE))
gc()
```

```
##           used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells 2386231 127.5    4263013  227.7   4141960  221.3
## Vcells 4424408  33.8  323696399 2469.7 380801493 2905.3
```

```
# Load necessary packages
if(!require(tidyverse))install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(tictoc))install.packages("tictoc", repos = "http://cran.us.r-project.org")
if(!require(caret))install.packages("caret", repos = "http://cran.us.r-project.org")

# Load the datasets
tic()
load("train_data_beerrecipe.RData")

# Set up cross validation method for the EXtreme Gradient Boosting model
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,  ## 5-fold CV
                           repeats = 3, ## repeated three times
                           p = 0.8,
                           returnResamp="all"
                           )

# Run the EXtreme Gradient Boosting model
set.seed(201955)
model_xgbTree <- train(Style ~ .,
            data = train_data,
            method = "xgbTree",
            trControl = fitControl,
            verbose= FALSE)


# Show variable important score in the EXtreme Gradient Boosting model
var_imp_xgbTree <- varImp(model_xgbTree)

# plot the EXtreme Gradient Boosting model
ggplot(var_imp_xgbTree) +
  ggtitle("Important Variables in EXtreme Gradient Boosting Model")
```




```
# Result of EXtreme Gradient Boosting approach
max_accuracy_model_xgbTree <- model_xgbTree$results %>% filter(Accuracy == max(Accuracy))
max_accuracy_model_xgbTree
```

```
##   eta max_depth gamma colsample_bytree min_child_weight subsample nrounds
## 1 0.3         2     0              0.6                1         1     100
##    Accuracy     Kappa  AccuracySD    KappaSD
## 1 0.6312246 0.5294517 0.008715443 0.01098674
```

```
# Save the maximum of accuracy to conduct Table of results
xgbTree_accuracy <- max(model_xgbTree$results[8])
save(xgbTree_accuracy, file = "xgbTree_accuracy.RData")
xgbTree_accuracy
```

```
## [1] 0.6312246
```

```
# Memory usage of running EXtreme Gradient Boosting model
memory.size()
```

```
## [1] 262.28
```

```
# Time spending in running EXtreme Gradient Boosting model
runtime <- toc()
```

```
## 3508.83 sec elapsed
```

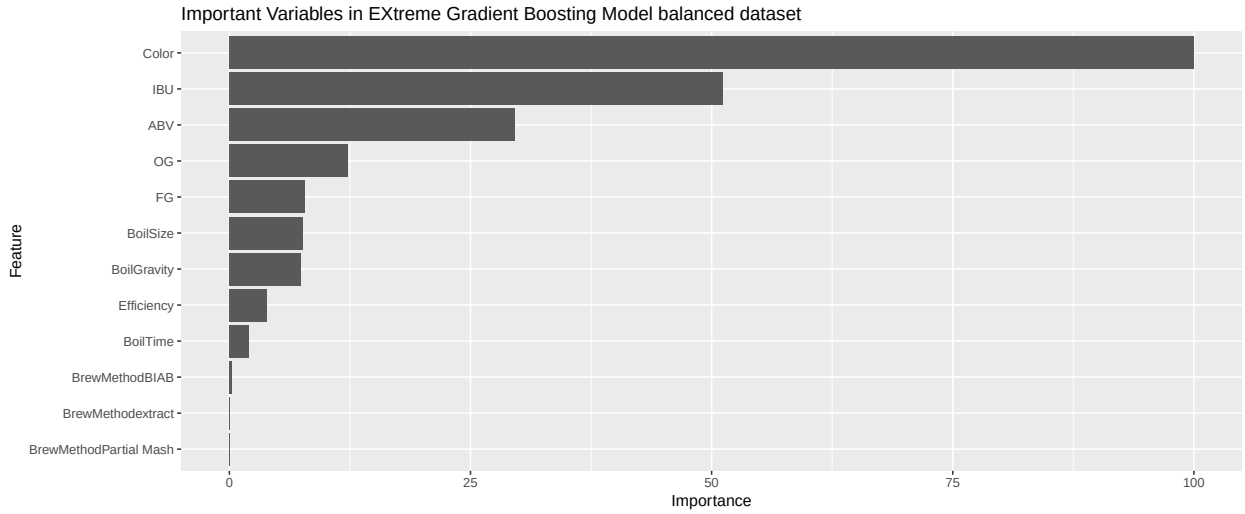
```
xgbTree_runtime <- runtime$toc - runtime$tic
save(xgbTree_runtime, file = "xgbTree_runtime.RData")
```

5.5.2.2 EXtreme Gradient Boosting model using "xgbTree" method and "up" sampling in caret package

```
# Remove and free up working space
rm(list = ls(all.names = TRUE))
gc()
```

```
##           used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells 2432018 129.9    4263013  227.7   4263013  227.7
## Vcells 4636434  35.4  207165695 1580.6 380801493 2905.3
```

```r
# Load necessary packages
if(!require(tidyverse))install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(tictoc))install.packages("tictoc", repos = "http://cran.us.r-project.org")
if(!require(caret))install.packages("caret", repos = "http://cran.us.r-project.org")

# Load the datasets
tic()
load("train_data_beerrecipe.RData")

# Set up cross validation method for the EXtreme Gradient Boosting model
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,  ## 5-fold CV
                           repeats = 3, ## repeated three times
                           p = 0.8,
                           returnResamp="all",
                           sampling = "up" ## randomly subset all the classes to match the
                                           ## frequencies of the higest prevalent class
                           )

# Run the EXtreme Gradient Boosting model  with "up" sampling method
set.seed(201955)
model_xgbTree_upsampling <- train(Style ~ .,
            data = train_data,
            method = "xgbTree",
            trControl = fitControl,
            verbose= FALSE)

# Show variable important score in the EXtreme Gradient Boosting model
var_imp_xgbTree_upsampling <- varImp(model_xgbTree_upsampling)

# plot the EXtreme Gradient Boosting model
ggplot(var_imp_xgbTree_upsampling) +
  ggtitle("Important Variables in EXtreme Gradient Boosting Model balanced dataset")
```




```r
# Result of EXtreme Gradient Boosting approach with "up" sampling method
max_accuracy_model_xgbTree_upsampling <- model_xgbTree_upsampling$results %>%
  filter(Accuracy == max(Accuracy))
```

```
max_accuracy_model_xgbTree_upsampling
```

```
##   eta max_depth gamma colsample_bytree min_child_weight subsample nrounds
## 1 0.3         3     0              0.8                1       0.5     150
##    Accuracy     Kappa  AccuracySD     KappaSD
## 1 0.5891184 0.5023963 0.008690979 0.009166818
```

```
# Save the maximun of accuracy to conduct Table of results
xgbTree_upsampling_accuracy <- max(model_xgbTree_upsampling$results[8])
save(xgbTree_upsampling_accuracy, file = "xgbTree_upsampling_accuracy.RData")
xgbTree_upsampling_accuracy
```

```
## [1] 0.5891184
```

```
# Memory usage of running EXtreme Gradient Boosting model with "up" sampling method
memory.size()
```

```
## [1] 377.25
```

```
# Time spending in running EXtreme Gradient Boosting model with "up" sampling method
runtime <- toc()
```

```
## 8868.52 sec elapsed
```

```
xgbTree_upsampling_runtime <- runtime$toc - runtime$tic
save(xgbTree_upsampling_runtime, file = "xgbTree_upsampling_runtime.RData")
```

6 Choosing a model for validation

The table of Accuracy metric of different models will help in deciding which model should be use in the validation step. This project will choose the model that generates the highest Accuracy in the testing data (test_data).

```
load("nb_accuracy.RData")
load("nb_runtime.RData")
load("nb_upsampling_accuracy.RData")
load("nb_upsampling_runtime.RData")
load("rpart_accuracy.RData")
load("rpart_runtime.RData")
load("rpart_upsampling_accuracy.RData")
load("rpart_upsampling_runtime.RData")
load("rf_upsampling_runtime.RData")
load("rf_accuracy.RData")
load("rf_runtime.RData")
load("rf_upsampling_accuracy.RData")
load("rf_upsampling_runtime.RData")
load("xgbTree_accuracy.RData")
load("xgbTree_runtime.RData")
load("xgbTree_upsampling_accuracy.RData")
load("xgbTree_upsampling_runtime.RData")


# Table of Accuracy of different machine learning approaches
accuracy_results <- data_frame(Model = "Naive Bayes",
                               Accuracy = nb_accuracy,
                               Runtime = nb_runtime)
```

```
accuracy_results <- bind_rows(accuracy_results,
                          data_frame(Model = "Naive Bayes Balanced dataset",
                                     Accuracy = nb_upsampling_accuracy,
                                     Runtime = nb_upsampling_runtime))

accuracy_results <- bind_rows(accuracy_results,
                          data_frame(Model = "Decision Tree",
                                     Accuracy = rpart_accuracy,
                                     Runtime = rpart_runtime))

accuracy_results <- bind_rows(accuracy_results,
                          data_frame(Model = "Decision Tree Balanced dataset",
                                     Accuracy = rpart_upsampling_accuracy,
                                     Runtime = rpart_upsampling_runtime))

accuracy_results <- bind_rows(accuracy_results,
                          data_frame(Model = "Random Forest",
                                     Accuracy = rf_accuracy,
                                     Runtime = rf_runtime))

accuracy_results <- bind_rows(accuracy_results,
                          data_frame(Model = "Random Forest Balanced dataset",
                                     Accuracy = rf_upsampling_accuracy,
                                     Runtime = rf_upsampling_runtime))

accuracy_results <- bind_rows(accuracy_results,
                          data_frame(Model = "EXtreme Gradient Boosting model",
                                     Accuracy = xgbTree_accuracy,
                                     Runtime = xgbTree_runtime))

accuracy_results <- bind_rows(accuracy_results,
                          data_frame(Model = "EXtreme Gradient Boosting model Balanced dataset",
                                     Accuracy = xgbTree_upsampling_accuracy,
                                     Runtime = xgbTree_upsampling_runtime))
accuracy_results %>% knitr::kable()
```

Model	Accuracy	Runtime
Naive Bayes	0.5700250	5.77
Naive Bayes Balanced dataset	0.5068842	9.82
Decision Tree	0.4850697	3.64
Decision Tree Balanced dataset	0.4882931	9.03
Random Forest	0.6259412	802.97
Random Forest Balanced dataset	0.6199171	2815.11
EXtreme Gradient Boosting model	0.6312246	3508.83
EXtreme Gradient Boosting model Balanced dataset	0.5891184	8868.52

7 Validate the highest accuracy predicting model of Beer Styles

```
## Remove and free up working space
rm(list = ls(all.names = TRUE))
gc()
```

```
##           used  (Mb) gc trigger  (Mb)  max used   (Mb)
## Ncells 2435928 130.1    4263013 227.7   4263013  227.7
## Vcells 4710568  36.0  106068835 809.3 380801493 2905.3
```

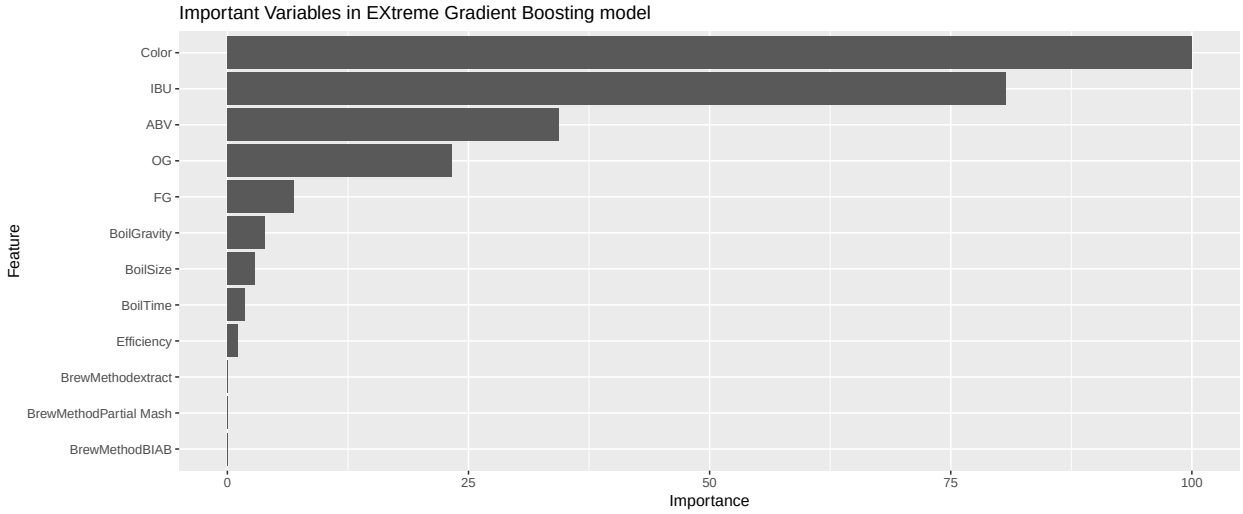
```
## Load the test_data
load("train_data_beerrecipe.RData")
load("test_data_beerrecipe.RData")

## Load necessary packages
if(!require(tidyverse))install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(tictoc))install.packages("tictoc", repos = "http://cran.us.r-project.org")
if(!require(caret))install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(Metrics))install.packages("Metrics", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: Metrics
```

```
##
## Attaching package: 'Metrics'
```

```
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```
## Set up cross validation method for the EXtreme Gradient Boosting model
tic()
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,  ## 5-fold CV
                           repeats = 3, ## repeated three times
                           p = 0.8,
                           returnResamp="all")

## Set up xgbTree parameters
xgbTree_Grid <- expand.grid(subsample = 1,
                            nrounds = 100,
                            min_child_weight = 1,
                            colsample_bytree = 0.6,
                            gamma = 0,
                            max_depth = 2,
                            eta = 0.3)

## Run the EXtreme Gradient Boosting model
set.seed(201955)
final_model_xgbTree <- train(Style ~ .,
                  data = train_data,
                  method = "xgbTree",
                  trControl = fitControl,
                  tuneGrid = xgbTree_Grid, ## Now specify the exact models
                                           ## to evaluate
                  verbose= FALSE)

## Show variable important score in the EXtreme Gradient Boostingt model
var_imp_final_model_xgbTree <- varImp(final_model_xgbTree)

## plot the EXtreme Gradient Boosting model
ggplot(var_imp_final_model_xgbTree) +
  ggtitle("Important Variables in EXtreme Gradient Boosting model")
```




```
## Result of EXtreme Gradient Boosting approach in the test_data
actual_style <- test_data$Style
predict_style <- predict(final_model_xgbTree, test_data)
final_model_xgbTree_accuracy <- accuracy(actual_style,predict_style)
final_model_xgbTree_accuracy
```

```
## [1] 0.6402834
```

```
## Save the maximum of accuracy to conduct Table of results
save(final_model_xgbTree_accuracy, file = "final_model_xgbTree_accuracy.RData")

## Memory usage of running EXtreme Gradient Boosting model
memory.size()
```

```
## [1] 325.13
```

```
## Time spending in running EXtreme Gradient Boosting model
runtime <- toc()
```

```
## 54.76 sec elapsed
```

```
final_model_xgbTree_runtime <- runtime$toc - runtime$tic
save(final_model_xgbTree_runtime, file = "final_model_xgbTree_runtime.RData")
```

8 Report Accuracy of the best model in the validation step

```
# Final result
Final_result <- data_frame(Model = "EXtreme Gradient Boosting",
                           Accuracy = final_model_xgbTree_accuracy,
                           Runtime_second = final_model_xgbTree_runtime)

Final_result %>% knitr::kable()
```

Model	Accuracy	Runtime_second
EXtreme Gradient Boosting	0.6402834	54.76

9 Conclusion

Through the process of doing a data science project including cleaning data, exploratory data analysis and running machine learning methods, the project had shown the final predicting model for Beer Style in this project is the EXtreme Gradient Boosting model. It achieved the highest accuracy rate at about 64%. The project had implemented up-sampling methods to handle unbalanced dataset, however, the results of accuracy score of all the up-sampled dataset for four machine learning algorithms did not increase. Although the maximum accuracy rate found in this project is quite low compared to the percentage of flipping a coin, it help an amateur beer tester like me to know there are about three most important characteristics of a beer that I should know to improve my skills are Beer Colors, IBU and ABV.

It is believed that the accuracy score can be improved if the project invests more time to solve the problem of having high appearance of outliers in the dataset.

Moreover, the author does not fully understand the mathematic behind those models are being used in the caret package, so it leads to the shallow exploration in tuning them to increase the test metric. In fact, lack of knowledge and experience in running those algorithms shortens the ability of the author to defend how he picks up the best model.

By doing the project, it helps the author to discover what he should target in the future of his career path as being Data Scientist. For example, instead of running many different machine learning, the learner can go deep into one algorithm like "Random Forest" and test it in many circumstances, so that he can apply the full power of this particular technique. In addition, he will be able to add the argument about advantages and disadvantages of using different machine learning methods.

Having more experience on machine learning would stop the author to randomly test out many models for his dataset, as a result, he can save more time for improving other tasks in the project. For example, the project would not run the Support Vector Machine, if the error like "Error in model.frame.default(formula = y ~ x, na.action = na.omit, drop.unused.levels = TRUE) : invalid type (list) for variable 'y' " could be known early. In addition, there is another error while running Neuron Network, i.e. "Error in h2o[hidden, output] <- abeta[grep(label, nms, fixed = TRUE)] : number of items to replace is not a multiple of replacement length".

It is not only to learn about the math behind each algorithm but also to learn more techniques in speeding up the running time of each model.