Wine_Quality_Predictions

Group 6

12/9/2018

Introduction

Our goal for this project is to conduct an analysis on Wine Quality data from the University of California, Irvine Machine Learning Repository. The data are the results of a chemical analysis of wines grown in the same region in Italy, but derived from three different cultivars. The data has 13 attributes which are wine type, fixed acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, and quality. We are interested to see if these attributes has an impact on the quality of wine. Therefore, our goal is to use machine learning to see if we can build an accurate model to predict the wine quality. Below you can see the features that were used to build this model.

Description of Data

The data contains 4547 observations with 12 features and 1 outcome variable. Also, the data has two missing values in feature total_sulfur_dioxide.

Features :

- **wine type -** 1096 Red and 3451 White wine
- **fixed acidity -** Most acids involved with wine or fixed or nonvolatile
- **volatile acidity -** The amount of acetic acid in wine, which at too high of levels can lead to an unpleasant, vinegar taste
- **citric acid -** the amount of acetic acid in wine, which at too high of levels can lead to an unpleasant, vinegar taste
- **residual sugar -** The amount of sugar remaining after fermentation stops, it's rare to find wines with less than 1 gram/liter and wines with greater than 45 grams/liter are considered sweet
- **chlorides -** The amount of salt in the wine
- **free sulfur dioxide -** The free form of SO2 exists in equilibrium between molecular SO2 (as a dissolved gas) and bisulfite ion; it prevents microbial growth and the oxidation of wine
- **total sulfur dioxide -** Amount of free and bound forms of S02; in low concentrations, SO2 is mostly undetectable in wine, but at free SO2 concentrations over 50 ppm, SO2 becomes evident in the nose and taste of wine
- **density -** the density of water is close to that of water depending on the percent alcohol and sugar content
- **pH -** Describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic); most wines are between 3-4 on the pH scale
- **sulphates -** a wine additive which can contribute to sulfur dioxide gas (S02) levels, wich acts as an antimicrobial and antioxidant
- **alcohol -** the percent alcohol content of the wine

Outcome Variable:

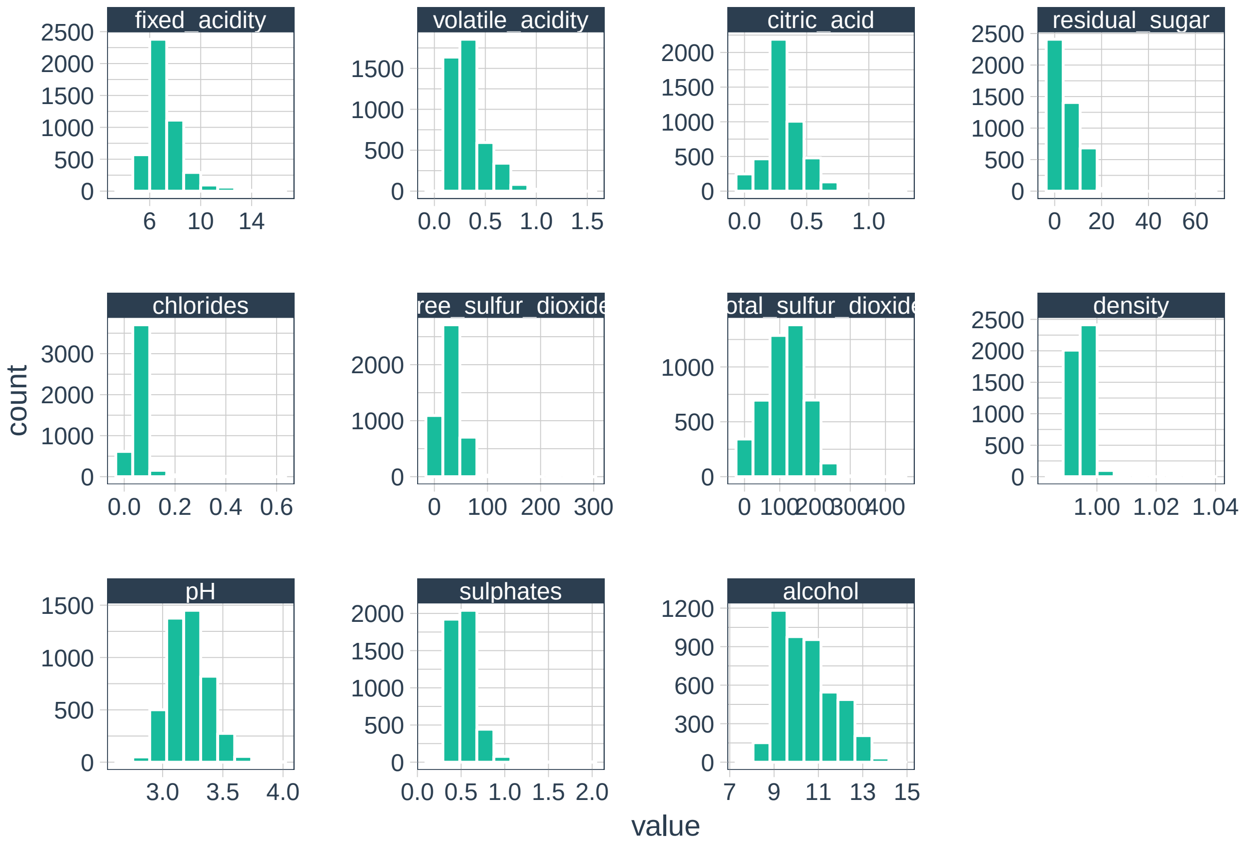
- **quality -** score between 0 and 10

Table 1: The first 5 rows

wine_type	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality
red	7.9	0.34	0.42	2.0	0.086	8	19	0.99546	3.35	0.60	11.4	6
white	7.2	0.33	0.22	4.5	0.031	10	73	0.99076	2.97	0.52	12.2	7
white	8.0	0.29	0.29	13.2	0.046	26	113	0.99830	3.25	0.37	9.7	6
white	5.6	0.19	0.47	4.5	0.030	19	112	0.99220	3.56	0.45	11.2	6
white	6.8	0.21	0.55	14.6	0.053	34	159	0.99805	2.93	0.44	9.2	5

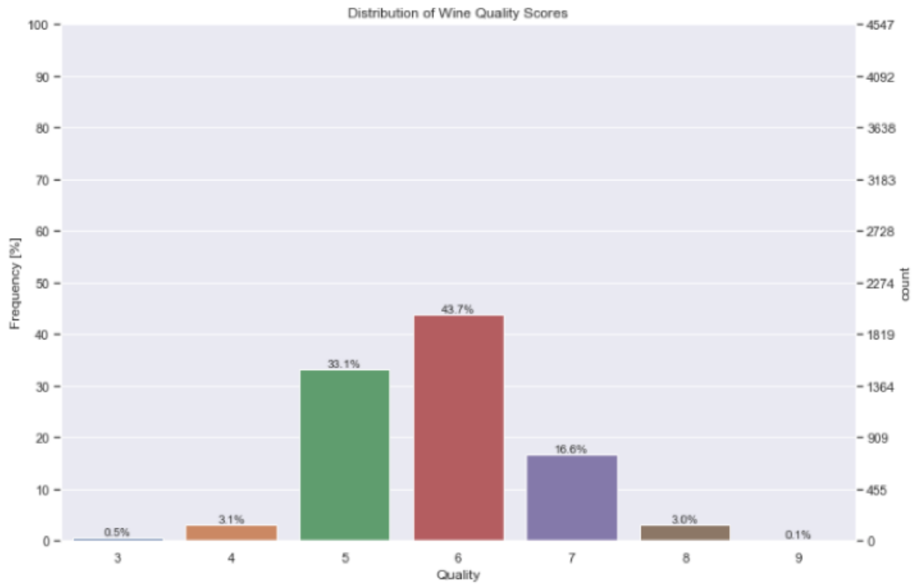
Histogram of features and outcome variable

```r
wine_train_orginal %>%
  select(-wine_type, -quality) %>%
  na.omit() %>% #remove the missing values
  plot_hist_facet(ncol = 4)
```



The distributions of citric acid, total sulfur dioxide, and pH are nearly normally distributed. The distributions of fixed acidity, volatile acidity, residual sugar, sulphates, and alcohol are right skewed. The distributions for chlorides, free sulfur dioxide, density, and quality have unknown distributions. Based on the histograms, we might need to normalize or standardize the features that have right skewed or unknown distributions for prediction.

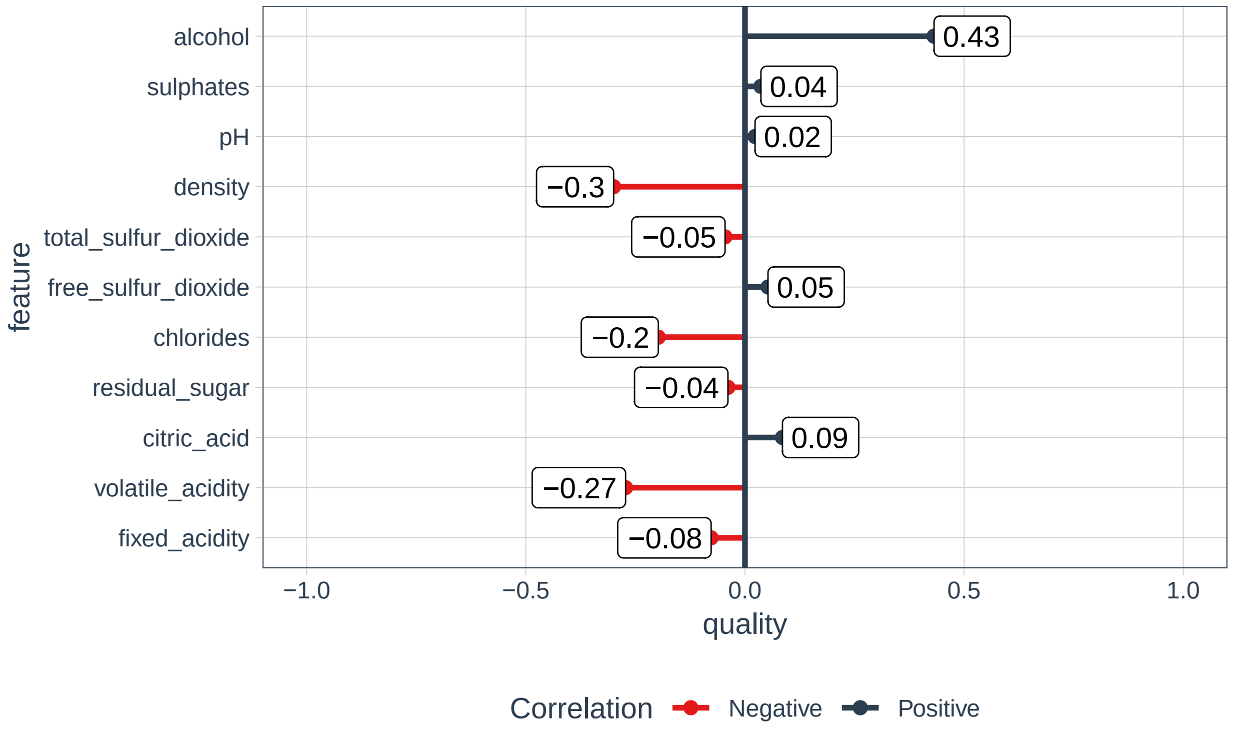
Histogram of the outcome variable quality



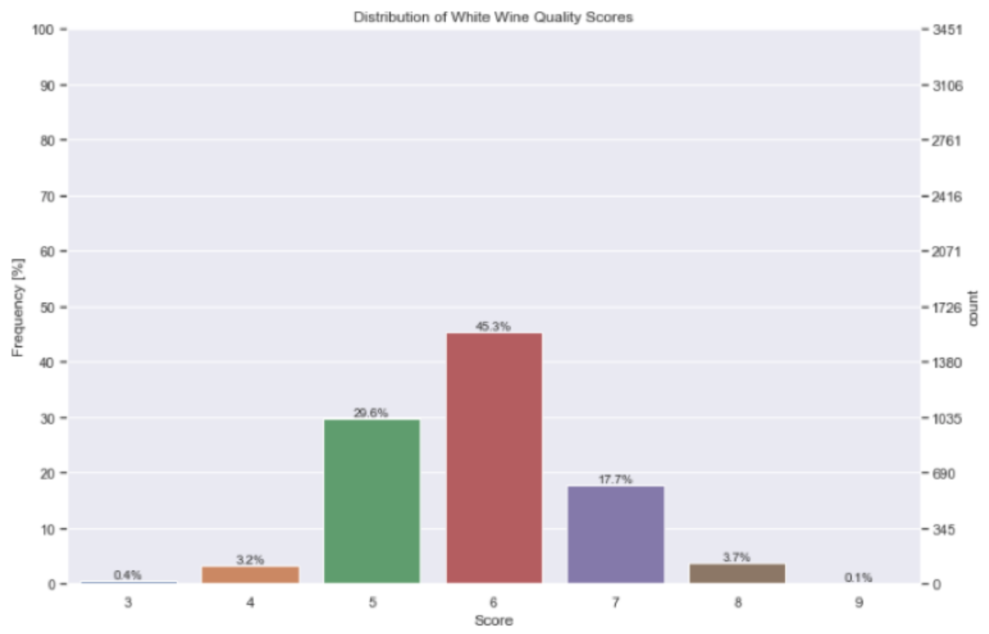
We can see that most of the scores are between 5 and 7 with a small percentage of the wines getting a score higher than 8.

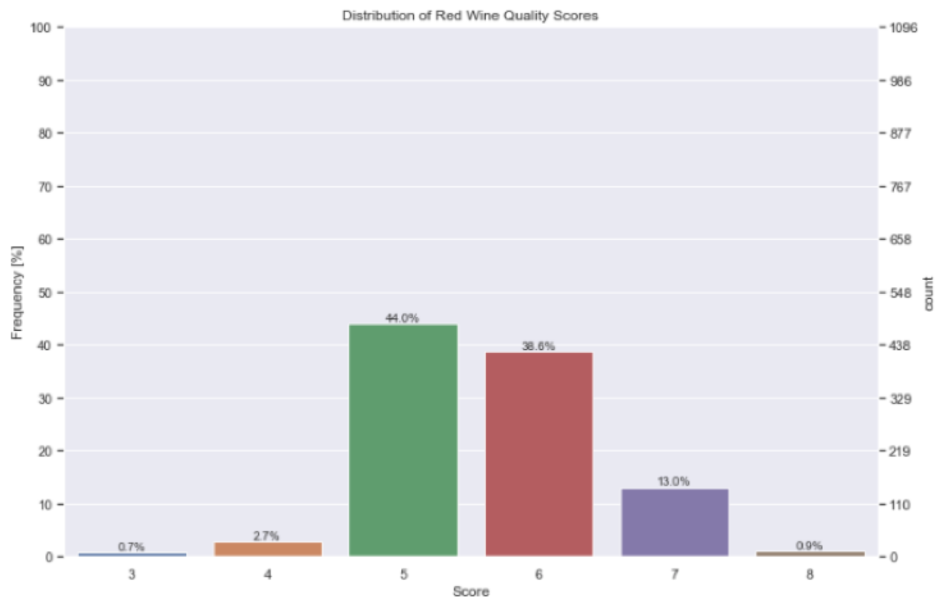
The correlation between features and outcome variable quality

```
wine_train_orginal %>%
  select(-wine_type) %>%
  plot_cor(target = quality)
```



We can see that quality doesn't have a strong correlation with almost all of the variables. Alcohol is the only variable that has a moderate positive correlation. Quality has a weak negative correlation with density and volatile acidity. I also split the data between red and white to see if there was any significant difference with the distribution of scores.

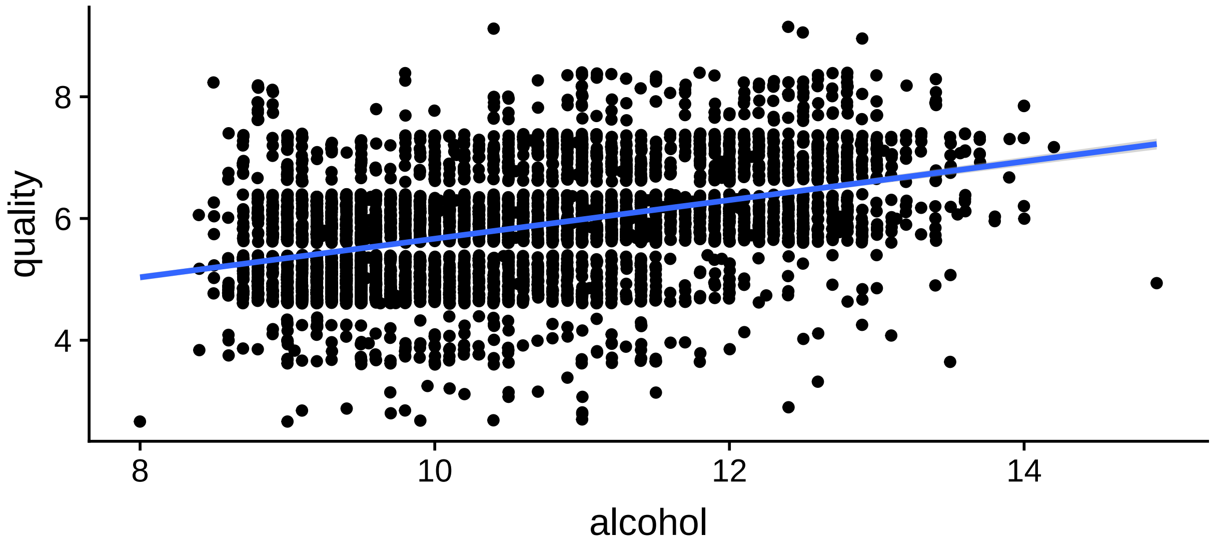




We can see that white wine had a bigger share of high quality (8+) scores than red wine. In addition, red wine seems to have more wines that are scored 5 or below than white wines.

Since alcohol has the strongest correction to wine quality, we plotted a scatter plot below.

```
wine_train_orginal %>%
  ggplot(aes(x = alcohol, y = quality)) +
  geom_jitter() + #add some variations to outcome variables.
  geom_smooth(method = "lm")
```



The scatter plot above did not show a very strong correlation. Our assumption is linear regression which would not be best model to predict wine quality.

Methods and Results

Our outcome variable quality is numerical, we are going to use the following Machine Learning Algorithms(Regression) to predict the wine quality:

- Linear Regression
- Ridge Regression
- Lasso Regression
- Ensemble Methods - Random Forest and Gradient Boosting Machines

We will split the data into 70% training and 30% validation. The validation set is used for model selection.

Pre Processing

```
pre_processing <- function(data){
  data %>%
    mutate(wine_type = as.factor(wine_type)) %>%
    na.omit() #remove the missing values

}

wine_train_orginal <- pre_processing(wine_train_orginal)


split_h2o <- h2o.splitFrame(as.h2o(wine_train_orginal), ratios = c(0.70), seed = 1234)

wine_train_h2o <- split_h2o[[1]]
wine_valid_h2o <- split_h2o[[2]]
```

linear Regression

```
y <- "quality"
x <- setdiff(names(wine_train_h2o), y)

wine_lm <- h2o.glm(
  training_frame = wine_train_h2o,
  validation_frame = wine_valid_h2o,
  x = x,
  y = y,
  lambda = 0, #no regualation
  seed = 123
  )


get_metrics_lm(model = wine_lm)
```

```
## Lambda: 0
## Alpha: 0.5
##
## Training MSE: 0.5186
## Training R_Squared: 0.29373
## -----------------------------------
## Validation MSE: 0.59614
## Validation R_Squared: 0.27598
```

Lasso and Ridge

- Perform the grid search to find best lambda
- Standarized the features

```
#alpha = 0 for ridge regression
#alpha = 1 for lasso

glm_params <- list(lambda = 10^seq(10, -2, length.out  = 100),
                   alpha = c(0,1))

glm_grid <- h2o.grid("glm",
                      x = x,
                      y = y,
                      training_frame = wine_train_h2o,
                      validation_frame = wine_valid_h2o,
                      hyper_params = glm_params,
                      standardize = T,
                      seed = 12)

glm_grid_metrics<- h2o.getGrid(grid_id = glm_grid@grid_id,
                             sort_by = "mse",
                             decreasing = F)
best_model_glm <- h2o.getModel(glm_grid_metrics@model_ids[[1]])
get_metrics_lm(model =  best_model_glm)
```

```
## Lambda: 0.01
## Alpha: 0
##
```

```
## Training MSE: 0.51883
## Training R_Squared: 0.29341
## ------------------------------------
## Validation MSE: 0.59764
## Validation R_Squared: 0.27416
```

Gradient Boosting Machine

- Train the GBM as base model
- Perform grid search to tune parameters for number of trees and maximum of depth

```
wine_gbm <- h2o.gbm(
  x = x,
  y = y,
  training_frame = wine_train_h2o,
  validation_frame = wine_valid_h2o,
  seed = 123,
  model_id = "wine_gbm"
)

get_metrics_tree(wine_gbm)
```

```
## ntrees: 50
## max_depth: 5
##
## Training MSE: 0.31357
## Training R_Squared: 0.57295
## ------------------------------------------
## Validation MSE: 0.48635
## Validation R_Squared: 0.40932
```

```
#grid search
parameters_gbm <- list(
  ntrees = c(50,100,200),
  max_depth = c(5,8,10,15))

grid_gbm <- h2o.grid(grid_id = "gbm_TUNE",
                    "gbm",
                    hyper_params = parameters_gbm,
                    y = y,
                    x = x,
                    training_frame = wine_train_h2o,
                    validation_frame = wine_valid_h2o,
                    seed = 123
                    )

gbm_sorted_grid <- h2o.getGrid(grid_id = "gbm_TUNE", sort_by = "mse")

best_model_gbm <- h2o.getModel(gbm_sorted_grid@model_ids[[1]])

get_metrics_tree(model = best_model_gbm)
```

```
## ntrees: 100
## max_depth: 15
##
```

```
## Training MSE: 0.02782
## Training R_Squared: 0.96211
## ------------------------------------------
## Validation MSE: 0.45457
## Validation R_Squared: 0.44792
```

Random Forest

- Train the RF as base model
- Perform grid search to tune parameters for number of trees and maximum of depth

```r
wine_rf <- h2o.randomForest(
  x = x,
  y = y,
  training_frame = wine_train_h2o,
  validation_frame = wine_valid_h2o,
  seed = 123,
  model_id = "wine_rf"
)

get_metrics_tree(model = wine_rf)
```

```
## ntrees: 50
## max_depth: 20
##
## Training MSE: 0.431
## Training R_Squared: 0.41302
## ------------------------------------------
## Validation MSE: 0.43854
## Validation R_Squared: 0.46738
```

```r
#grid search find best combination for number of tress and
#maximum of depth.

parameters_rf <- list(
  ntrees = c(300,500,600,700),
  max_depth = c(10,15,20,25,30))

grid_rf <- h2o.grid(grid_id = "RF_TUNE",
                    "randomForest",
                    hyper_params = parameters_rf,
                    y = y,
                    x = x,
                    training_frame = wine_train_h2o,
                    validation_frame = wine_valid_h2o,
                    seed = 123
                    )

rf_sorted_grid <- h2o.getGrid(grid_id = "RF_TUNE", sort_by = "mse")

best_model_rf <- h2o.getModel(rf_sorted_grid@model_ids[[1]])

get_metrics_tree(model = best_model_rf)
```

```
## ntrees: 700
## max_depth: 30
##
## Training MSE: 0.40946
## Training R_Squared: 0.44237
## ------------------------------------------
## Validation MSE: 0.42899
## Validation R_Squared: 0.47898
```

Dicussion

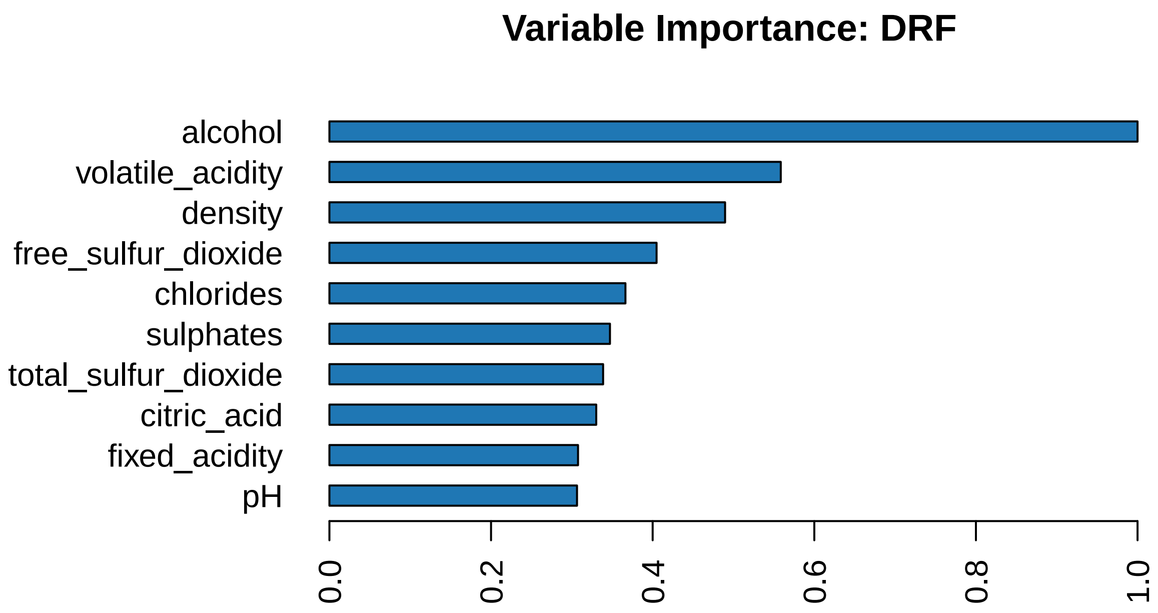
```r
models <- c(best_model_glm, best_model_gbm, best_model_rf)

h2o.mse_valid <- function(model){
  h2o.mse(model, valid = T)
}



df <- tibble(model = factor(c("LM", "GBM", "RF")),
       train_mse = map_dbl(.x = models, .f = h2o.mse),
       validation_mse = map_dbl(.x = models, .f = h2o.mse_valid)
)

knitr::kable(df)
```

model	train_mse	validation_mse
LM	0.5188315	0.5976424
GBM	0.0278216	0.4545675
RF	0.4094558	0.4289930

The best method to predict wine quality is Random Forest. The table above shows that it has the lowest validation mse. The GBM has very low training mse, but high validation mse. It indicates that the GBM is overfitted. The linear model do not have enough flexiablity to find the data. Therefore, Random Forest is the best method.

Variables Importance

```r
h2o.varimp_plot(best_model_rf)
```



According the variable importance plot, the top 3 most important features are alcohol, volatile acidity, and density.

In order to improve the model, we need more data to make our outcome variable quality more evenly distributed. In our training dataset, we only have 4 observation that has quality score 9. Also, the model like GBM and Ranfom Forest tend to work better with larger data, because it won't overfitted the data.

Code Appendix

```
####Functons for wine quality predictions


#Rename
rename_wine <- function(data) {
  data %>%
    rename(fixed_acidity = `fixed acidity`, volatile_acidity = `volatile acidity`,
           citric_acid = `citric acid`,
           residual_sugar = `residual sugar`,
           free_sulfur_dioxide = `free sulfur dioxide`,
           total_sulfur_dioxide = `total sulfur dioxide`)
}

# Correlation ----
get_cor <- function(data, target, use = "pairwise.complete.obs",
                    fct_reorder = FALSE, fct_rev = FALSE) {

  feature_expr <- enquo(target)
  feature_name <- quo_name(feature_expr)

  data_cor <- data %>%
    mutate_if(is.character, as.factor) %>%
    mutate_if(is.factor, as.numeric) %>%
    cor(use = use) %>%
    as.tibble() %>%
    mutate(feature = names(.)) %>%
    select(feature, !! feature_expr) %>%
    filter(!(feature == feature_name)) %>%
    mutate_if(is.character, as_factor)

  if (fct_reorder) {
    data_cor <- data_cor %>%
      mutate(feature = fct_reorder(feature, !! feature_expr)) %>%
      arrange(feature)
  }

  if (fct_rev) {
    data_cor <- data_cor %>%
      mutate(feature = fct_rev(feature)) %>%
      arrange(feature)
  }

  return(data_cor)

}


# Correlation plot----
plot_cor <- function(data, target, fct_reorder = FALSE, fct_rev = FALSE,
                     include_lbl = TRUE, lbl_precision = 2, lbl_position = "outward",
                     size = 2, line_size = 1, vert_size = 1,
```

```r
                      color_pos = palette_light()[[1]],
                      color_neg = palette_light()[[2]]) {

  feature_expr <- enquo(target)
  feature_name <- quo_name(feature_expr)

  data_cor <- data %>%
    get_cor(!! feature_expr, fct_reorder = fct_reorder, fct_rev = fct_rev) %>%
    mutate(feature_name_text = round(!! feature_expr, lbl_precision)) %>%
    mutate(Correlation = case_when(
      (!! feature_expr) >= 0 ~ "Positive",
      TRUE                   ~ "Negative") %>% as.factor())

  g <- data_cor %>%
    ggplot(aes_string(x = feature_name, y = "feature", group = "feature")) +
    geom_point(aes(color = Correlation), size = size) +
    geom_segment(aes(xend = 0, yend = feature, color = Correlation), size = line_size) +
    geom_vline(xintercept = 0, color = palette_light()[[1]], size = vert_size) +
    expand_limits(x = c(-1, 1)) +
    theme_tq() +
    scale_color_manual(values = c(color_neg, color_pos))

  if (include_lbl) g <- g + geom_label(aes(label = feature_name_text), hjust = lbl_position)

  return(g)

}


#histogram ----
plot_hist_facet <- function(data, bins = 10, ncol = 5,
                            fct_reorder = FALSE, fct_rev = FALSE,
                            fill = palette_light()[[3]],
                            color = "white", scale = "free") {

  data_factored <- data %>%
    mutate_if(is.character, as.factor) %>%
    mutate_if(is.factor, as.numeric) %>%
    gather(key = key, value = value, factor_key = TRUE)

  if (fct_reorder) {
    data_factored <- data_factored %>%
      mutate(key = as.character(key) %>% as.factor())
  }

  if (fct_rev) {
    data_factored <- data_factored %>%
      mutate(key = fct_rev(key))
  }

  g <- data_factored %>%
    ggplot(aes(x = value, group = key)) +
    geom_histogram(bins = bins, fill = fill, color = color) +
    facet_wrap(~ key, ncol = ncol, scale = scale) +
    theme_tq()

  return(g)
}


#For GLM model
get_metrics_lm <- function(model){

glue::glue(
"Lambda: {model@parameters$lambda}
Alpha: {model@parameters$alpha}

Training MSE: {round(h2o.mse(model, train = T), 5)}
Training R_Squared: {round(h2o.r2(model, train = T), 5)}
-----------------------------------
Validation MSE: {round(h2o.mse(model, valid = T),5)}
Validation R_Squared: {round(h2o.r2(model, valid = T),5)}")
}


#For tree based model
get_metrics_tree <- function(model){

glue::glue(
"ntrees: {model@allparameters$ntrees}
max_depth: {model@allparameters$max_depth}

Training MSE: {round(h2o.mse(model, train = T), 5)}
Training R_Squared: {round(h2o.r2(model, train = T), 5)}
------------------------------------------
Validation MSE: {round(h2o.mse(model, valid = T),5)}
Validation R_Squared: {round(h2o.r2(model, valid = T),5)}")
}
```