Appendix

```r
# Reading in the libraries
library(tidyverse)
library(dplyr)
library(caret)
library(readr)
library(knitr)
library(regclass)
library(formatR)
library(e1071)
library(kernlab)
library(ROCR)
library(gridExtra)
library(corrplot)
```

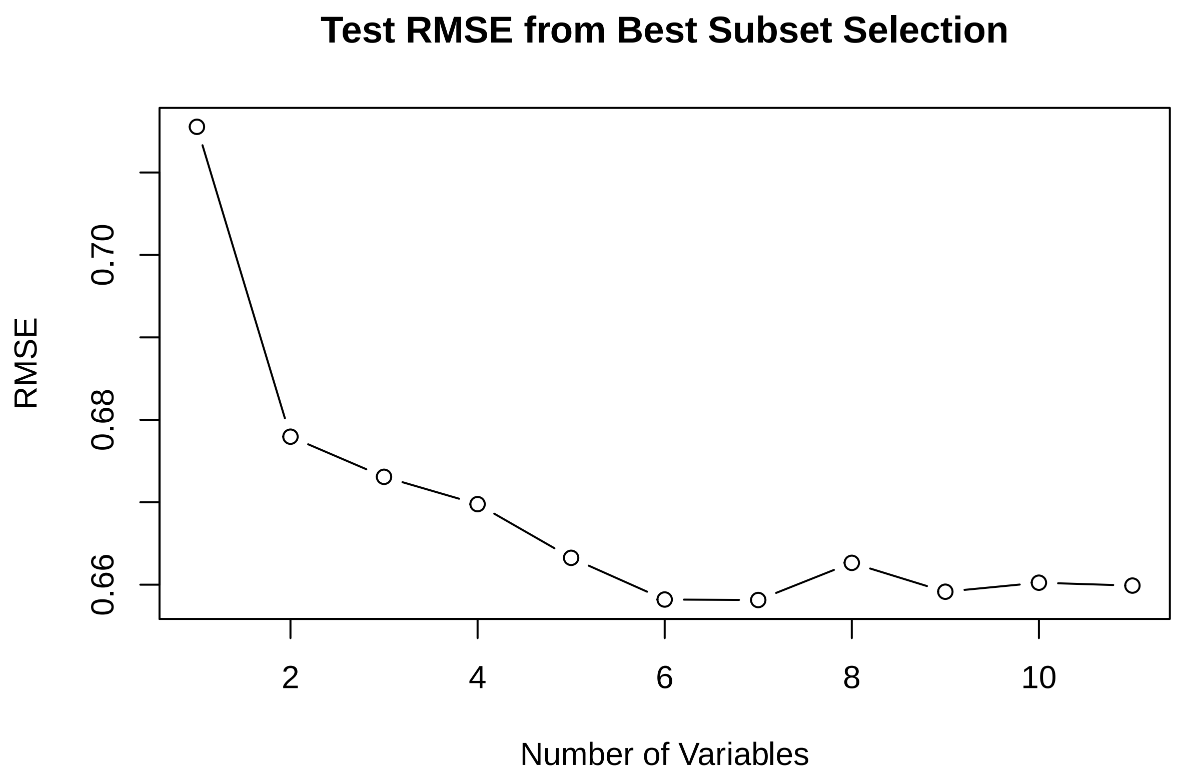
```r
# Reading in the dataset
red <- read.csv("winequality-red.csv", sep=";")
str(red)
```

```
## 'data.frame':    1599 obs. of  12 variables:
##  $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
##  $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
##  $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
##  $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
##  $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...
##  $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
##  $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
##  $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
##  $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
##  $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
##  $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
##  $ quality             : int  5 5 5 6 5 5 5 7 7 5 ...
```

```r
# Setting seed for reproducibility
set.seed(123)
# wine_data$quality <- as.factor(wine_data$quality)
# Dividing the data randomly into two sets
# A training set that I will use to fit the models
# A test set that will be used to evaluate the methods.
trainIndex <- createDataPartition(red$quality, p = 0.7,
                                  list = FALSE)
train.red <- red[trainIndex, ]
test.red <- red[-trainIndex, ]
```

Multiple Linear Regression

```r
# select the model using best subset selection
regfit_best = regsubsets(quality~., data=train.red, nvmax=11)

# create a test matrix
test_mat = model.matrix(quality~., data=test.red)

# create a vector to contain all test MSE
val_errors = rep(NA,11)

# calculate the test MSE for the best model of each size
for(i in 1:11){
  coefi=coef(regfit_best, id=i)
  pred=test_mat[,names(coefi)]%*%coefi
  val_errors[i]=sqrt(mean((test.red$quality-pred)^2))
}

# plot the test MSE by variable number
plot(seq(1:11), val_errors, type='b',
     ylab='RMSE',
     xlab='Number of Variables',
     main='Test RMSE from Best Subset Selection')
```



```
# get the coefficient estimates for the 6-variable model
coef(regfit_best, 6)
```

```
##          (Intercept)     volatile.acidity            chlorides
##          3.992762657         -1.092690945         -2.125881455
## total.sulfur.dioxide                   pH            sulphates
##         -0.001995661         -0.325031921          0.935400833
##              alcohol
##          0.284116039
```

```
# obtain the test RMSE for the final model
val_errors[6]
```

```
## [1] 0.6582011
```

Logistic Regression

```
# Setting seed for reproducibility
set.seed(123)
red2 <- red
red2$quality <- ifelse(red$quality >= 6, 1, 0)
red2$quality <- as.factor(red2$quality)

# Dividing the data randomly into two sets
# A training set that I will use to fit the models
# A test set that will be used to evaluate the methods.
trainIndex2 <- createDataPartition(red2$quality, p = 0.7,
                                   list = FALSE)
train.red2 <- red2[trainIndex2, ]
test.red2 <- red2[-trainIndex2, ]
```

```
# Fitting a binary logistic regresison model
model <- glm(quality ~ alcohol + volatile.acidity + sulphates +
               I(volatile.acidity^2), data = train.red2, family = "binomial")
# Model Summary
summary(model)
```

```
##
## Call:
## glm(formula = quality ~ alcohol + volatile.acidity + sulphates +
##     I(volatile.acidity^2), family = "binomial", data = train.red2)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4928  -0.8481   0.2852   0.8729   2.4408
##
## Coefficients:
##                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)             -9.5068     1.1101  -8.564  < 2e-16 ***
## alcohol                  1.0061     0.0832  12.092  < 2e-16 ***
## volatile.acidity       -4.2111     1.9170  -2.197    0.028 *
## sulphates               2.0037     0.4633   4.325 1.53e-05 ***
## I(volatile.acidity^2)   0.6345     1.6096   0.394    0.693
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1547.2  on 1119  degrees of freedom
## Residual deviance: 1179.5  on 1115  degrees of freedom
## AIC: 1189.5
##
## Number of Fisher Scoring iterations: 4
```

```
# Response: This has a very low AIC
```

```
# Logistic regression model fitting
logfit <- train(quality ~ alcohol + volatile.acidity + sulphates +
    I(volatile.acidity^2), data = train.red2, method = "glm",
    family = "binomial", preProcess = c("center", "scale"), trControl = trainControl(method = "cv",
        number = 5))
logfit
```

```
## Generalized Linear Model
##
## 1120 samples
##    3 predictor
##    2 classes: '0', '1'
##
## Pre-processing: centered (4), scaled (4)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 896, 895, 896, 897, 896
## Resampling results:
##
##   Accuracy   Kappa
##   0.7366768  0.4725162
```

```
# Seeing how well Marcus model performs on the test set
# using accuracy for the logistic modeling
log_conf = confusionMatrix(data = test.red2$quality, reference = predict(logfit,
    newdata = test.red2))

# produce a kable
kable(log_conf$table, caption = "Confusion Matrix\nfor Logistic Regression ")
```

Table 1: Confusion Matrix for Logistic Regression

	0	1
0	160	63
1	76	180

```
misclass0 <- 1- ((160+180)/(160+63+76+180))
misclass0
```

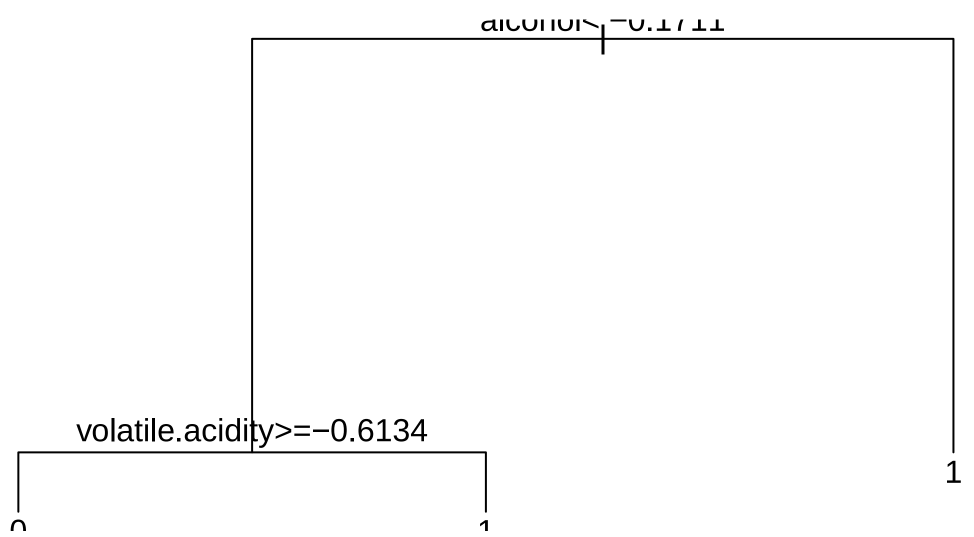
```
## [1] 0.2901879
```

Classification tree

```
trctrl <- trainControl(method= "repeatedcv", number = 10, repeats = 3)
# Create a classification tree
cTree <- train(quality ~ ., method = "rpart", trControl = trctrl, data = train.red2,
               preProcess = c("center", "scale"))
cTree
```

```
## CART
##
## 1120 samples
##   11 predictor
##    2 classes: '0', '1'
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1008, 1008, 1008, 1008, 1008, 1008, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.02879079  0.7199865  0.4351482
##   0.05566219  0.7009385  0.4023184
##   0.38771593  0.6076122  0.1812726
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.02879079.
```

```
# plot the final tree model
plot(cTree$finalModel)
text(cTree$finalModel)
```



```
# predict the values for our respone variable and compare it to our\ testing data
cTree_pred <- predict(cTree, newdata=select(test.red2,-quality))
# a frequency of how many of each response there is.
cTreepred <- table(cTree_pred, test.red2$quality)
#cTreepred
misclass1 <- 1- (sum(diag(cTreepred))/sum(cTreepred))
misclass1
```

```
## [1] 0.3465553
```

```
# produce a kable
kable(cTreepred, caption="Confusion Matrix for Classification Tree")
```

Table 2: Confusion Matrix for Classification Tree

	0	1
0	126	69
1	97	187

Random Forest model

```r
# Create a random forest model
rforest <- train(quality ~ ., method = "rf", trControl = trctrl,
    data = train.red2, preProcess = c("center", "scale"))
rforest
```

```
## Random Forest
##
## 1120 samples
##   11 predictor
##    2 classes: '0', '1'
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1008, 1008, 1008, 1007, 1008, 1008, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.8103978  0.6192520
##    6    0.8080088  0.6144588
##   11    0.8008658  0.6000804
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```r
# Predict the values for our response variable and compare it to our testing data.
rforest_pred <- predict(rforest, newdata = select(test.red2,-quality))

# a frequency of how many of each respons]e there is.
rfpred <- table(rforest_pred, test.red2$quality)
misclass2 <- 1- (sum(diag(rfpred))/sum(rfpred))
misclass2
```

```
## [1] 0.2212944
```

```r
# produce a kable
kable(rfpred, caption="Confusion Matrix\nfor Random Forest")
```

Table 3: Confusion Matrix for Random Forest

	0	1
0	165	48
1	58	208

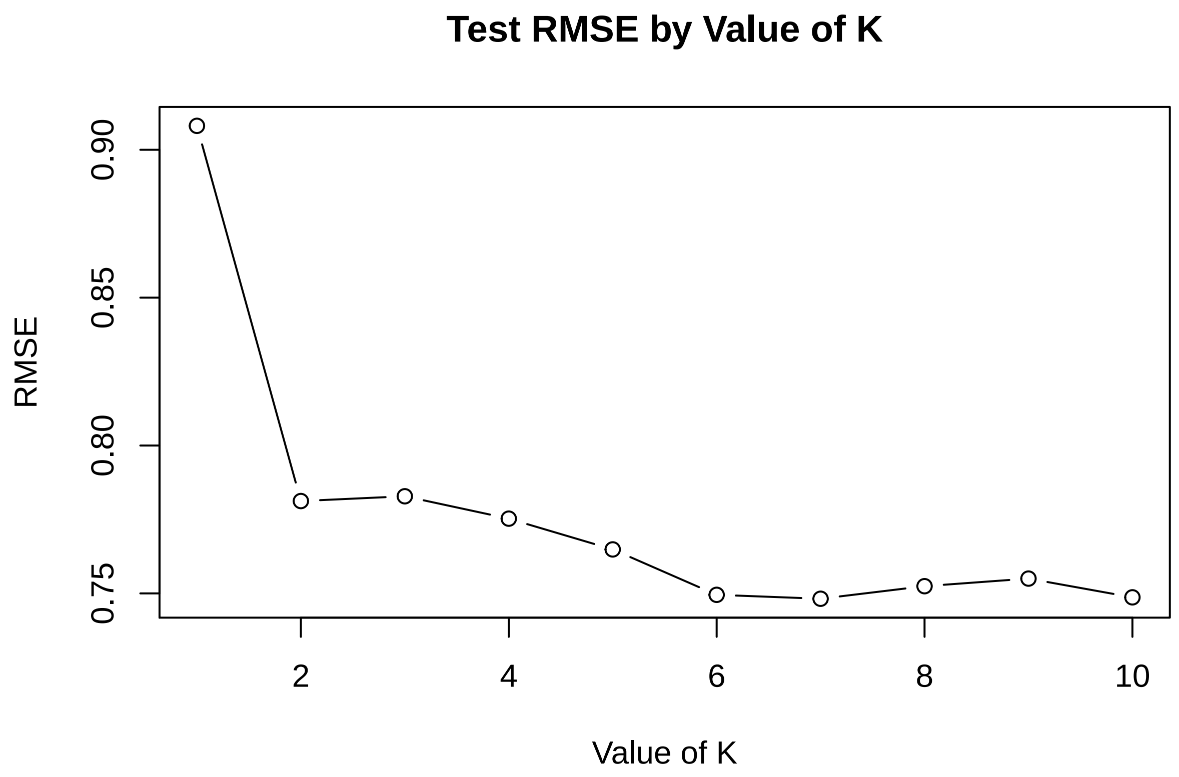
K-Nearest Neighbors

```
### KNN - Classification

# set global training control options
train_control = trainControl(method='cv', number=5)

# fit the model
knn_class_fit <- train(quality ~ .,
                       method='knn',
                       tuneGrid=expand.grid(k=1:10),
                       trControl=train_control,
                       metric="Accuracy",
                       data=train.red2)

# predict on the KNN classification model
knn_conf = confusionMatrix(data=test.red2$quality,
                           reference=predict(knn_class_fit,newdata=test.red2))

# produce a kable
kable(knn_conf$table, caption="Confusion Matrix\nfor KNN")
```

Table 4: Confusion Matrix for KNN

	0	1
0	143	80
1	63	193

```
### KNN - Regression

# Calculate test MSE for K values
knn_RMSE = rep(NA,10)
for(i in 1:10){
  knn_reg_fit <- train(quality ~ .,
                       method='knn',
                       tuneGrid=expand.grid(k=i),
                       trControl=train_control,
                       metric="RMSE",
                       data=train.red)
  knn_preds = predict(knn_reg_fit, newdata=test.red)
  knn_RMSE[i] = RMSE(knn_preds, test.red$quality)
}

# plot the RMSE by K
plot(seq(1:10), knn_RMSE, type='b',
     xlab='Value of K',
     ylab='RMSE',
     main='Test RMSE by Value of K')
```



Support Vector Machines

```
### SVM - Linear Classification

# fit the model
svm_class_linear <- train(quality ~ .,
                          method='svmLinear',
                          preProcess=c('center','scale'),
                          trControl=train_control,
                          metric="Accuracy",
                          data=train.red2)

# predict on the SVM linear classification model
svm_lin_conf = confusionMatrix(data=test.red2$quality,
                               reference=predict(svm_class_linear,newdata=test.red2))

# produce a kable
kable(svm_lin_conf$table, caption="Confusion Matrix\nfor SVM - Linear")
```

Table 5: Confusion Matrix for SVM - Linear

	0	1
0	167	56
1	76	180

```
### SVM - Linear Regression
svm_reg_linear <- train(quality ~ .,
                        method='svmLinear',
                        preProcess=c('center','scale'),
                        trControl=train_control,
                        metric="RMSE",
                        data=train.red)

# predict on the svm linear regression model
svm_linear_preds <- predict(svm_reg_linear, newdata=test.red)
RMSE(svm_linear_preds, test.red$quality)
```

```
## [1] 0.6620388
```

```
### SVM - Poly Classification

# fit the model
svm_class_poly <- train(quality ~ .,
                        method='svmPoly',
                        preProcess=c('center','scale'),
                        trControl=train_control,
                        metric="Accuracy",
                        data=train.red2)

# predict on the SVM poly classification model
svm_poly_conf = confusionMatrix(data=test.red2$quality,
                                reference=predict(svm_class_poly,newdata=test.red2))

# produce a kable
kable(svm_poly_conf$table, caption="Confusion Matrix\nfor SVM - Polynomial")
```

Table 6: Confusion Matrix for SVM - Polynomial

	0	1
0	168	55
1	66	190

```
### SVM - Poly Regression

# fit the model
svm_reg_poly <- train(quality ~ .,
                      method='svmPoly',
                      preProcess=c('center','scale'),
                      trControl=train_control,
```

```r
                      metric="RMSE",
                      data=train.red)

# predict on the svm poly regression model
svm_poly_preds <- predict(svm_reg_poly, newdata=test.red)
RMSE(svm_poly_preds, test.red$quality)
```

```
## [1] 0.6520513
```

```r
### SVM - Radial Classification

# fit the model
svm_class_radial <- train(quality ~ .,
                          method='svmRadial',
                          preProcess=c('center','scale'),
                          trControl=train_control,
                          metric="Accuracy",
                          data=train.red2)

# predict on the SVM radial classification model
svm_radial_conf = confusionMatrix(data=test.red2$quality,
                                  reference=predict(svm_class_radial,newdata=test.red2))

# produce a kable
kable(svm_radial_conf$table, caption="Confusion Matrix\nfor SVM - Radial")
```

Table 7: Confusion Matrix for SVM - Radial

	0	1
0	173	50
1	67	189

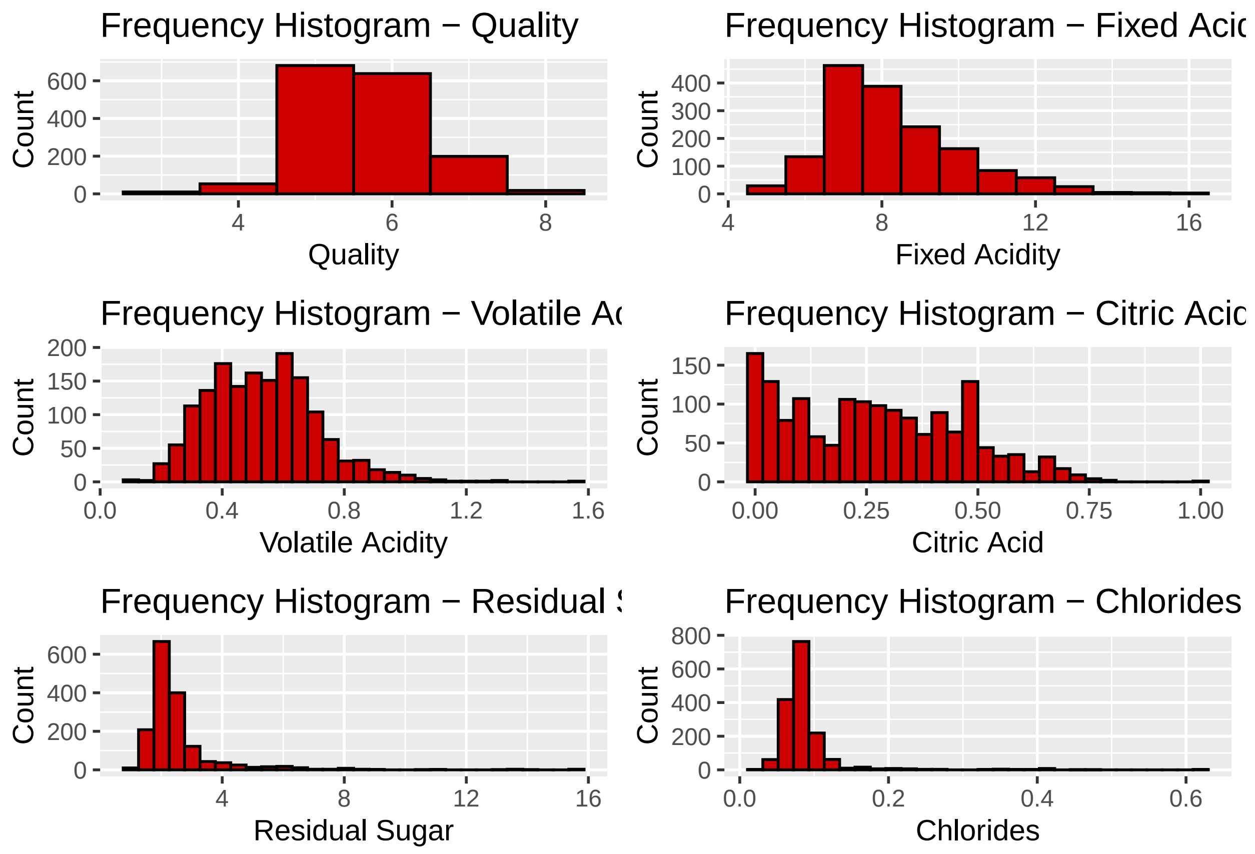
```r
### SVM - Radial Regression

# fit the model
svm_reg_radial <- train(quality ~ .,
                        method='svmRadial',
                        preProcess=c('center','scale'),
                        trControl=train_control,
                        metric="RMSE",
                        data=train.red)

# predict on the SVM radial regression model
svm_radial_preds <- predict(svm_reg_radial, newdata=test.red)
RMSE(svm_radial_preds, test.red$quality)
```

```
## [1] 0.6170703
```

```r
# plot histograms for all variables
quality = ggplot(red, aes(x=red$quality)) +
  geom_histogram(binwidth=1, fill='#CC0000', color='#000000') +
  xlab('Quality') +
  ylab('Count') +
  ggtitle('Frequency Histogram - Quality')

fixed_acidity = ggplot(red, aes(x=red$fixed.acidity)) +
  geom_histogram(binwidth=1, fill='#CC0000', color='#000000') +
  xlab('Fixed Acidity') +
  ylab('Count') +
  ggtitle('Frequency Histogram - Fixed Acidity')

volatile_acidity = ggplot(red, aes(x=red$volatile.acidity)) +
  geom_histogram(fill='#CC0000', color='#000000') +
  xlab('Volatile Acidity') +
  ylab('Count') +
  ggtitle('Frequency Histogram - Volatile Acidity')

citric_acid = ggplot(red, aes(x=red$citric.acid)) +
  geom_histogram(fill='#CC0000', color='#000000') +
  xlab('Citric Acid') +
  ylab('Count') +
  ggtitle('Frequency Histogram - Citric Acid')

residual_sugar = ggplot(red, aes(x=red$residual.sugar)) +
  geom_histogram(fill='#CC0000', color='#000000') +
  xlab('Residual Sugar') +
  ylab('Count') +
  ggtitle('Frequency Histogram - Residual Sugar')

chorides = ggplot(red, aes(x=red$chlorides)) +
  geom_histogram(fill='#CC0000', color='#000000') +
  xlab('Chlorides') +
  ylab('Count') +
  ggtitle('Frequency Histogram - Chlorides')

free_sulfur = ggplot(red, aes(x=red$free.sulfur.dioxide)) +
  geom_histogram(binwidth=1, fill='#CC0000', color='#000000') +
  xlab('Free Sulfur Dioxide') +
  ylab('Count') +
  ggtitle('Frequency Histogram - Free Sulfur Dioxide')

total_sulfur = ggplot(red, aes(x=red$total.sulfur.dioxide)) +
  geom_histogram(binwidth=10, fill='#CC0000', color='#000000') +
  xlab('Total Sulfur Dioxide') +
  ylab('Count') +
  ggtitle('Frequency Histogram - Total Sulfur Dioxide')

density = ggplot(red, aes(x=red$density)) +
  geom_histogram( fill='#CC0000', color='#000000') +
  xlab('Density') +
  ylab('Count') +
  ggtitle('Frequency Histogram - Density')
```

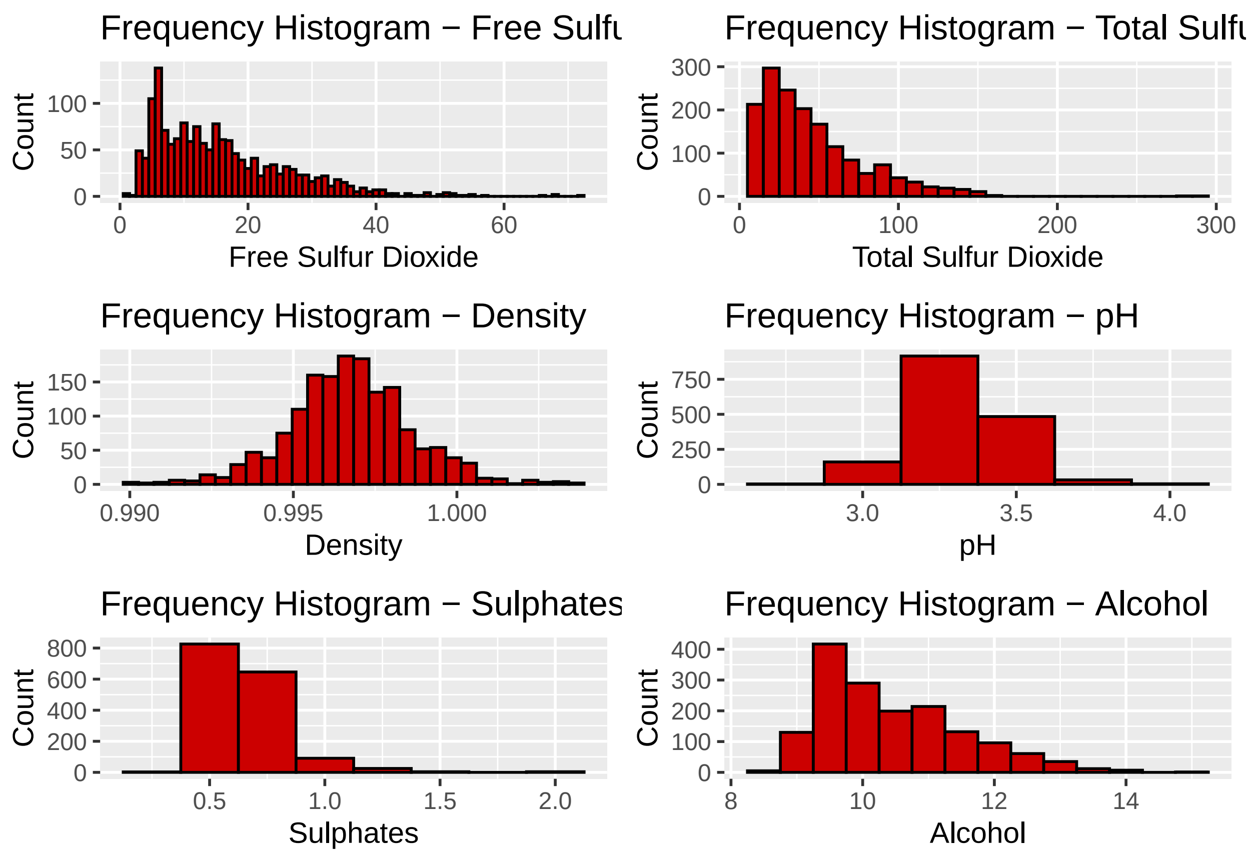
```
ph = ggplot(red, aes(x=red$pH)) +
  geom_histogram(binwidth=0.25, fill='#CC0000', color='#000000') +
  xlab('pH') +
  ylab('Count') +
  ggtitle('Frequency Histogram - pH')

sulphates = ggplot(red, aes(x=red$sulphates)) +
  geom_histogram(binwidth=0.25,fill='#CC0000', color='#000000') +
  xlab('Sulphates') +
  ylab('Count') +
  ggtitle('Frequency Histogram - Sulphates')

alcohol = ggplot(red, aes(x=red$alcohol)) +
  geom_histogram(binwidth=0.5, fill='#CC0000', color='#000000') +
  xlab('Alcohol') +
  ylab('Count') +
  ggtitle('Frequency Histogram - Alcohol')

grid.arrange(quality, fixed_acidity, volatile_acidity, citric_acid, residual_sugar, chorides)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

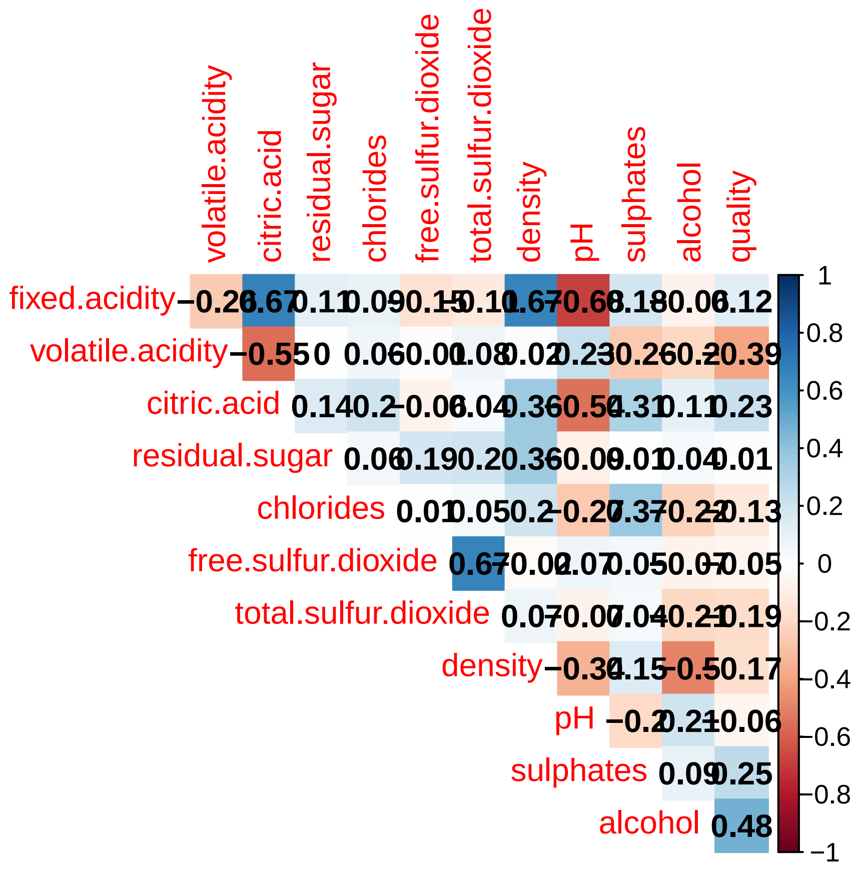


```
grid.arrange(free_sulfur, total_sulfur, density, ph, sulphates, alcohol)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
# create a correlation plot
cor_red = cor(red)
corrplot(cor_red, method='color', type='upper', addCoef.col='black', diag=FALSE)
```


volatile.acidity
citric.acid
residual.sugar
chlorides
free.sulfur.dioxide
total.sulfur.dioxide
density
pH
sulphates
alcohol
quality
fixed.acidity
volatile.acidity
citric.acid
residual.sugar
chlorides
free.sulfur.dioxide
total.sulfur.dioxide
density
pH
sulphates
alcohol