Water Potability

Rodrigo Ponce de Leon

2022-09-06

Contents:

INTRODUCTION:

As part of the PH125.9x:Data Science: Capstone, in this project the water_potability.csv data set, obtained from kaggle, and whose author is Aditya Kadiwal, is analyzed to fit an ML model or algorithm to predict water potability (1 means potable and 0 otherwise).

- Steps that took place to achieve the goal:

1. Data import.
2. Analysis of the variables in the data set.
3. Data splitting into train set and test set, data preparation-by scaling and by either imputing null values with the mean (train_data_1, test_data_1) or discarding NA-containing rows (train_data_2, test_data_2).
4. Training of eight different models. All 8 models were trained and tested with train_data_1 and test_data_1, while 3 were trained and tested for train_data_2 and test_data_2.

The variables in the data set are pH value, hardness, solids, chloramines, sulfate, conductivity, organic carbon, trihalomethanes, turbidity, and potability. In summary, the data set has 3276 observations and 10 variables.

The following are the descriptions of the variables the dataset has:

1. pH value:

PH is an important parameter in evaluating the acid–base balance of water. It is also the indicator of acidic or alkaline condition of water status. WHO has recommended maximum permissible limit of pH from 6.5 to 8.5. The current investigation ranges were 6.52–6.83 which are in the range of WHO standards.

2. Hardness:

Hardness is mainly caused by calcium and magnesium salts. These salts are dissolved from geologic deposits through which water travels. The length of time water is in contact with hardness producing material helps determine how much hardness there is in raw water. Hardness was originally defined as the capacity of water to precipitate soap caused by Calcium and Magnesium.

3. Solids (Total dissolved solids - TDS):

Water has the ability to dissolve a wide range of inorganic and some organic minerals or salts such as potassium, calcium, sodium, bicarbonates, chlorides, magnesium, sulfates etc. These minerals produced un-wanted taste and diluted color in appearance of water. This is the important parameter for the use of water. The water with high TDS value indicates that water is highly mineralized. Desirable limit for TDS is 500 mg/l and maximum limit is 1000 mg/l which prescribed for drinking purpose.

4. Chloramines:

Chlorine and chloramine are the major disinfectants used in public water systems. Chloramines are most commonly formed when ammonia is added to chlorine to treat drinking water. Chlorine levels up to 4 milligrams per liter (mg/L or 4 parts per million (ppm)) are considered safe in drinking water.

5. Sulfate:

Sulfates are naturally occurring substances that are found in minerals, soil, and rocks. They are present in ambient air, groundwater, plants, and food. The principal commercial use of sulfate is in the chemical industry. Sulfate concentration in seawater is about 2,700 milligrams per liter (mg/L). It ranges from 3 to 30 mg/L in most freshwater supplies, although much higher concentrations (1000 mg/L) are found in some geographic locations.

6. Conductivity:

Pure water is not a good conductor of electric current rather's a good insulator. Increase in ions concentration enhances the electrical conductivity of water. Generally, the amount of dissolved solids in water determines the electrical conductivity. Electrical conductivity (EC) actually measures the ionic process of a solution that enables it to transmit current. According to WHO standards, EC value should not exceeded 400 muS/cm.

7. Organic_carbon:

Total Organic Carbon (TOC) in source waters comes from decaying natural organic matter (NOM) as well as synthetic sources. TOC is a measure of the total amount of carbon in organic compounds in pure water. According to US EPA < 2 mg/L as TOC in treated / drinking water, and < 4 mg/Lit in source water which is use for treatment.

8. Trihalomethanes:

THMs are chemicals which may be found in water treated with chlorine. The concentration of THMs in drinking water varies according to the level of organic material in the water, the amount of chlorine required to treat the water, and the temperature of the water that is being treated. THM levels up to 80 ppm is considered safe in drinking water.

9. Turbidity:

The turbidity of water depends on the quantity of solid matter present in the suspended state. It is a measure of light emitting properties of water and the test is used to indicate the quality of waste discharge with respect to colloidal matter. The mean turbidity value obtained for Wondo Genet Campus (0.98 NTU) is lower than the WHO recommended value of 5.00 NTU.

10. Potability:

Indicates if water is safe for human consumption where 1 means Potable and 0 means Not potable.

The data can be obtained from the following url: https://www.kaggle.com/datasets/adityakadiwal/water-potability?resource=download

PACKAGE LOADING:

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

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

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

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

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

```
## Loading required package: data.table
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last

## The following object is masked from 'package:purrr':
##
##     transpose
```

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

```
## Loading required package: xgboost

##
## Attaching package: 'xgboost'

## The following object is masked from 'package:dplyr':
##
##     slice
```

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

```
## Loading required package: naivebayes

## naivebayes 0.9.7 loaded

##
## Attaching package: 'naivebayes'

## The following object is masked from 'package:data.table':
##
##     tables
```

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

```
## Loading required package: gbm

## Loaded gbm 2.1.8.1
```

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

```
## Loading required package: kernlab

##
## Attaching package: 'kernlab'
```

```
## The following object is masked from 'package:purrr':
##
##     cross

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```
library(tidyverse)
library(caret)
library(data.table)
```

DATA IMPORT:

The data is stored in the same working directory. Therefore, it can be imported using the name of the csv file:

```
wp <- read_csv(url("https://raw.githubusercontent.com/JRPoncedeLeonC/Water-potability/main/water_potabi
```

```
## Rows: 3276 Columns: 10

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## dbl (10): ph, Hardness, Solids, Chloramines, Sulfate, Conductivity, Organic_...

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

VIEW AT DATA:

The data is analyzed to see its structure and distribution of the variables:

```
str(wp) #structure of dataset
```

```
## spec_tbl_df [3,276 x 10] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ ph             : num [1:3276] NA 3.72 8.1 8.32 9.09 ...
##  $ Hardness       : num [1:3276] 205 129 224 214 181 ...
##  $ Solids         : num [1:3276] 20791 18630 19910 22018 17979 ...
##  $ Chloramines    : num [1:3276] 7.3 6.64 9.28 8.06 6.55 ...
##  $ Sulfate        : num [1:3276] 369 NA NA 357 310 ...
##  $ Conductivity   : num [1:3276] 564 593 419 363 398 ...
##  $ Organic_carbon : num [1:3276] 10.4 15.2 16.9 18.4 11.6 ...
##  $ Trihalomethanes: num [1:3276] 87 56.3 66.4 100.3 32 ...
##  $ Turbidity      : num [1:3276] 2.96 4.5 3.06 4.63 4.08 ...
##  $ Potability     : num [1:3276] 0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   ph = col_double(),
##   ..   Hardness = col_double(),
##   ..   Solids = col_double(),
##   ..   Chloramines = col_double(),
```

```
##   ..   Sulfate = col_double(),
##   ..   Conductivity = col_double(),
##   ..   Organic_carbon = col_double(),
##   ..   Trihalomethanes = col_double(),
##   ..   Turbidity = col_double(),
##   ..   Potability = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

```
head(wp) #first 6 observations
```

```
## # A tibble: 6 x 10
##      ph Hardness Solids Chloramines Sulfate Conductivity Organic_carbon
##   <dbl>    <dbl>  <dbl>       <dbl>   <dbl>        <dbl>          <dbl>
## 1 NA        205. 20791.        7.30    369.         564.           10.4
## 2  3.72     129. 18630.        6.64     NA          593.           15.2
## 3  8.10     224. 19910.        9.28     NA          419.           16.9
## 4  8.32     214. 22018.        8.06    357.         363.           18.4
## 5  9.09     181. 17979.        6.55    310.         398.           11.6
## 6  5.58     188. 28749.        7.54    327.         280.            8.40
## # ... with 3 more variables: Trihalomethanes <dbl>, Turbidity <dbl>,
## #   Potability <dbl>
```

```
summary(wp) #summary of numeric variables
```

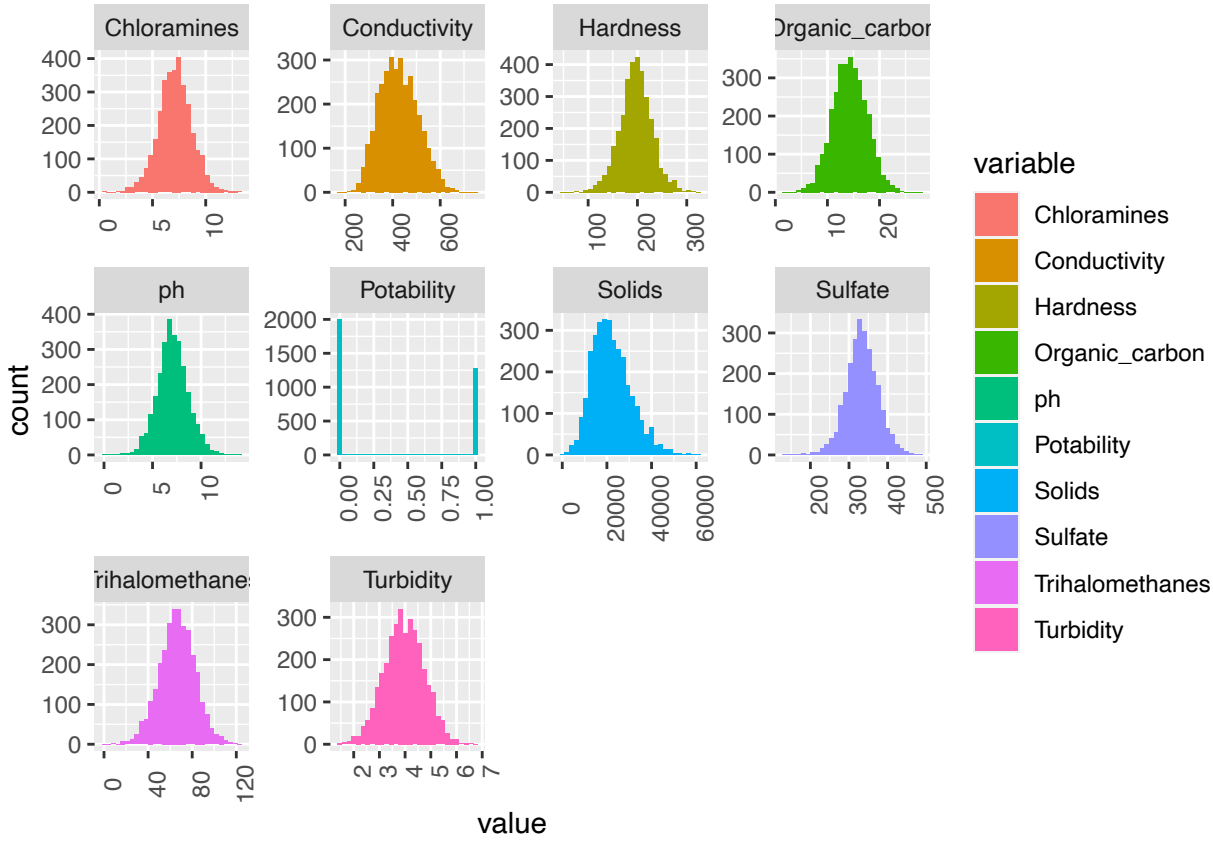
```
##        ph            Hardness          Solids         Chloramines
##  Min.   : 0.000   Min.   : 47.43   Min.   :  320.9   Min.   : 0.352
##  1st Qu.: 6.093   1st Qu.:176.85   1st Qu.:15666.7   1st Qu.: 6.127
##  Median : 7.037   Median :196.97   Median :20927.8   Median : 7.130
##  Mean   : 7.081   Mean   :196.37   Mean   :22014.1   Mean   : 7.122
##  3rd Qu.: 8.062   3rd Qu.:216.67   3rd Qu.:27332.8   3rd Qu.: 8.115
##  Max.   :14.000   Max.   :323.12   Max.   :61227.2   Max.   :13.127
##  NA's   :491
##     Sulfate       Conductivity   Organic_carbon  Trihalomethanes
##  Min.   :129.0   Min.   :181.5   Min.   : 2.20   Min.   :  0.738
##  1st Qu.:307.7   1st Qu.:365.7   1st Qu.:12.07   1st Qu.: 55.845
##  Median :333.1   Median :421.9   Median :14.22   Median : 66.622
##  Mean   :333.8   Mean   :426.2   Mean   :14.28   Mean   : 66.396
##  3rd Qu.:360.0   3rd Qu.:481.8   3rd Qu.:16.56   3rd Qu.: 77.337
##  Max.   :481.0   Max.   :753.3   Max.   :28.30   Max.   :124.000
##  NA's   :781                                     NA's   :162
##    Turbidity       Potability
##  Min.   :1.450   Min.   :0.0000
##  1st Qu.:3.440   1st Qu.:0.0000
##  Median :3.955   Median :0.0000
##  Mean   :3.967   Mean   :0.3901
##  3rd Qu.:4.500   3rd Qu.:1.0000
##  Max.   :6.739   Max.   :1.0000
##
```

```
wp %>% gather(key = "variable", value = "value") %>%
  ggplot(aes(value, fill = variable)) + geom_histogram() +
```

```
  theme(axis.text.x = element_text(angle = 90)) +
  facet_wrap(~ variable, scales = "free") #histogram for all variables
```

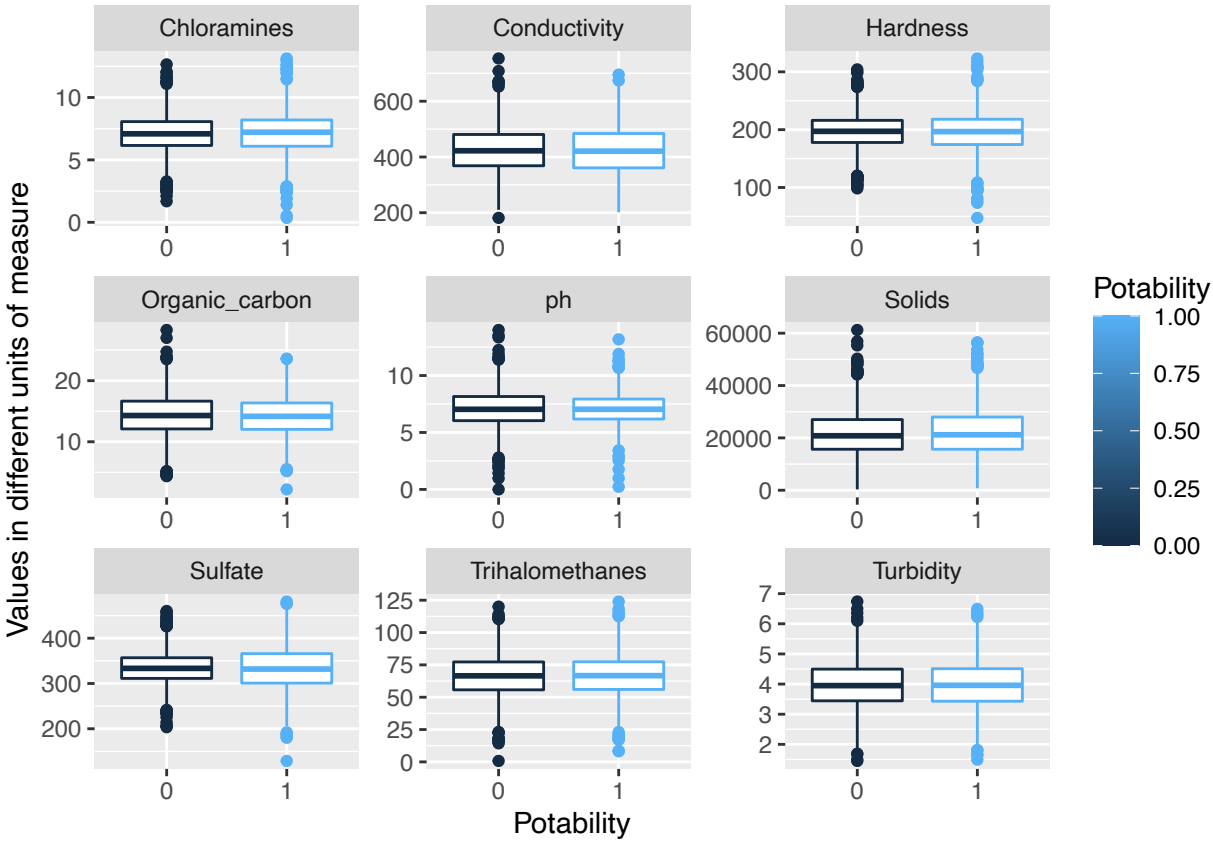
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 1434 rows containing non-finite values (stat_bin).
```




```
gather(wp, key = "k", value = "v", -Potability) %>%
  ggplot(aes(x=factor(Potability), y=v, col = Potability)) +
  geom_boxplot() +
  facet_wrap(~ k, scales="free") +
  xlab("Potability") +
  ylab("Values in different units of measure") #boxplot of all variables by potability
```

```
## Warning: Removed 1434 rows containing non-finite values (stat_boxplot).
```



The dataset contains only numerical data, has 3276 and 10 variables, and all the predictors have some sort of symmetry. Also, there aren't histograms that are tail-heavy, which makes it easier for some ML algorithms to detect patterns. On top of that, there are more observations for potability equals 0 than 1.

When looking at the boxplots for each variable all of them have outliers. Moreover, their interquartile ranges don't vary much for potable and non-potable water.

The missing values in the entire dataset are analyzed:

```
sum(is.na(wp)) #total number of NA
```

```
## [1] 1434
```

```
wp %>% gather(key = "variable", value = "value") %>%
  group_by(variable) %>% summarise(na_num = sum(is.na(value)))
```

```
## # A tibble: 10 x 2
##    variable        na_num
##    <chr>            <int>
##  1 Chloramines          0
##  2 Conductivity         0
##  3 Hardness             0
##  4 Organic_carbon       0
##  5 ph                 491
##  6 Potability           0
##  7 Solids               0
```

```
##  8 Sulfate            781
##  9 Trihalomethanes    162
## 10 Turbidity            0
```

From the tibble above, we can see that pH, Sulfate, and Trihalomethanes are the only variables with missing values, with Sulfate being the one with most missing values. In total, we have 1434 NAs.

DATA SPLITTING AND TRAINING SET ANALYSIS:

A test set and train set are created. The train set is analyzed to see if there are any correlations between the variables:

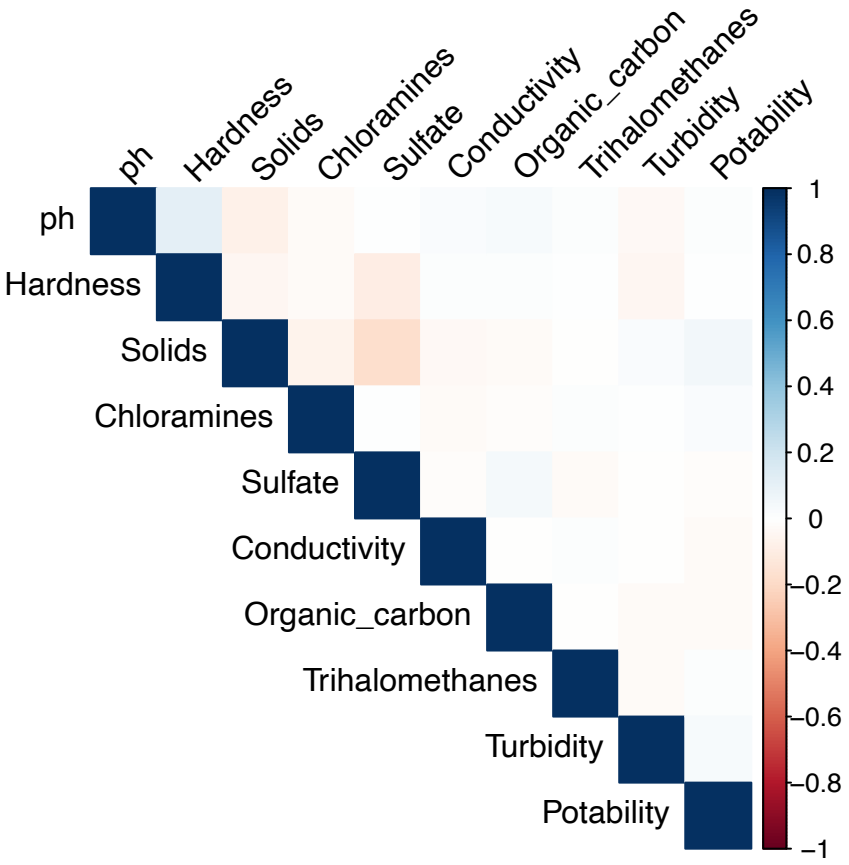
```
y <- wp$Potability
set.seed(42, sample.kind = "default")
test_index <- createDataPartition(y, times=1, p=0.2, list=F)
train_data <- wp[-test_index,]
test_data <- wp[test_index,]
```

```
cor_mat <- cor(train_data, use = "complete.obs")
```

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

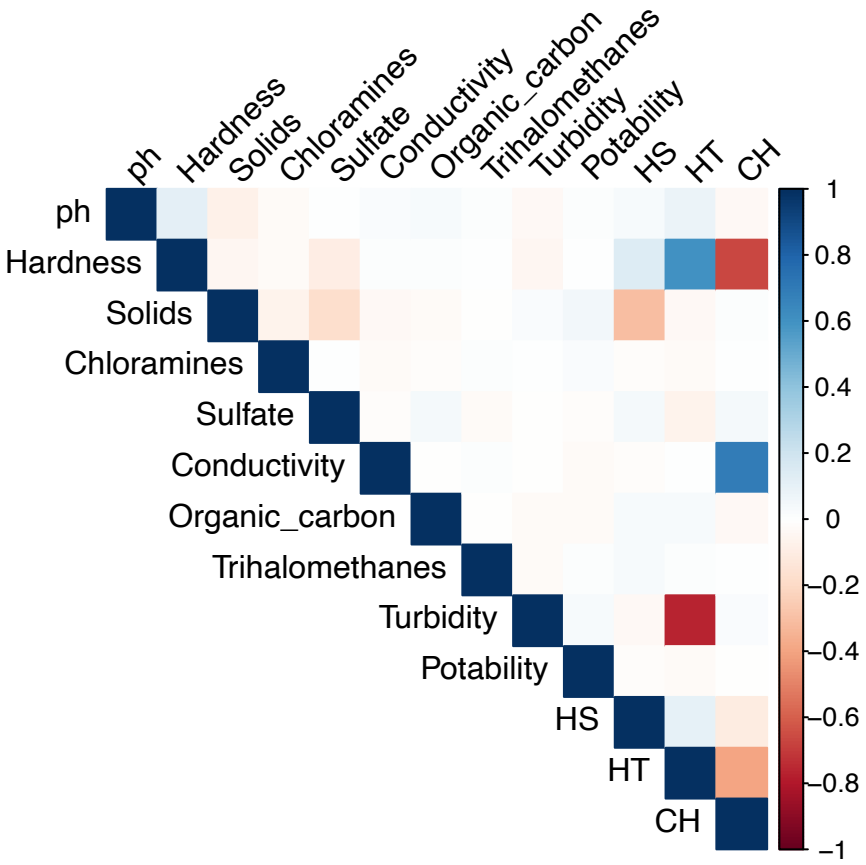
```
corrplot(cor_mat, type = "upper",
         tl.col = "black", tl.srt = 45, method="color")
```



From the plot above, we can see that there is no strong correlations between the variables, except when they are compared with themselves. We see that the strongest correlation for Potability is when this variable is compared against Solids. However, still, the correlation is not strong.

Let's see if we can get stronger correlations for Potability by experimenting with a few attribute combinations. Let's experiment with Hardness/Solids, Hardness/Turbidity and Conductivity/Hardness:

```
cor_mat_1 <- cor(train_data %>% mutate("HS"=Hardness/Solids,
                       "HT"=Hardness/Turbidity,
                       "CH"=Conductivity/Hardness), use = "complete.obs")
corrplot(cor_mat_1, type = "upper",
         tl.col = "black", tl.srt = 45, method="color")
```



Looking at both previous plots, there are no strong correlations with Potability and the rest of the variables, and this makes sense since Potability is a nominal-categorical variable. Therefore, we won't consider the attribute combinations to train the ML algorithms.

DATA PREPARATION:

For the data preparation process, we first need to replace the missing values with a statistical measure like the mean. Afterwards, scaling of the data can be performed for better comparison.

```
ReplaceNa <- function(df){ #function to replace the missing values
  df_1 <- df %>%
    replace_na(list(ph=mean(df$ph,na.rm=T),
                    Sulfate=mean(df$Sulfate,na.rm=T),
                    Trihalomethanes=mean(df$Trihalomethanes,na.rm=T)))
```

```
  return(df_1)
}
```

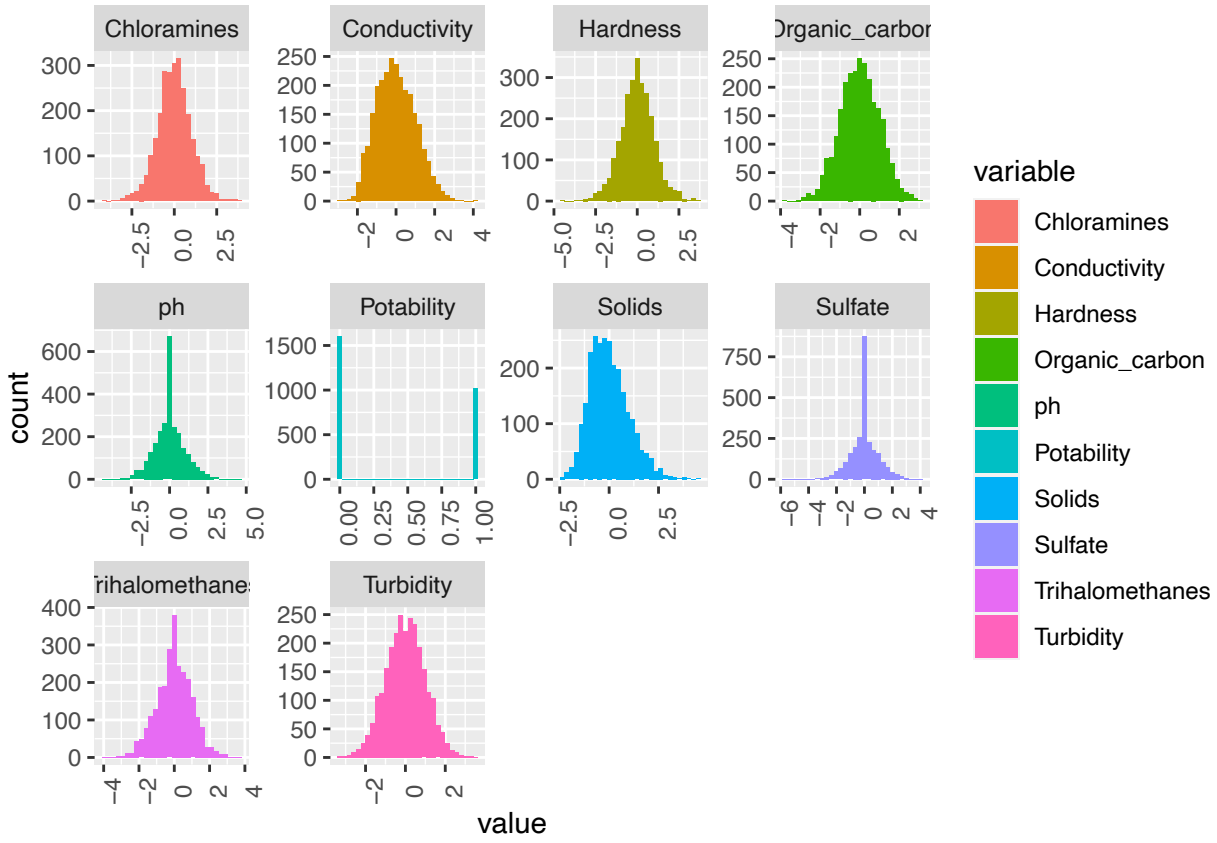
```
ScaleData <- function(df){ #function to scale the data
  df_pred <- df %>% select(-Potability) %>% scale() #dataframe with scaled predictors
  df_y <- df %>% select(Potability) #dataframe with Potability
  return(cbind(df_pred, df_y))
}
```

```
train_data_1 <- ReplaceNa(train_data) %>% ScaleData #clean training set
```

```
test_data_1 <- ReplaceNa(test_data) %>% ScaleData #clean test set
```

```
train_data_1 %>% gather(key = "variable", value = "value") %>%
  ggplot(aes(value, fill = variable)) + geom_histogram() +
  theme(axis.text.x = element_text(angle = 90)) +
  facet_wrap(~ variable, scales = "free") #histogram for all variables
```

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



MODEL TRAINING AND EVALUATION:

Now that the data is prepared, different models can be trained to see which one performs the best. In this case, classification Machine Learning algorithms are used. On top of that, k-fold cross-validation and parameter tuning on some algorithms are considered:

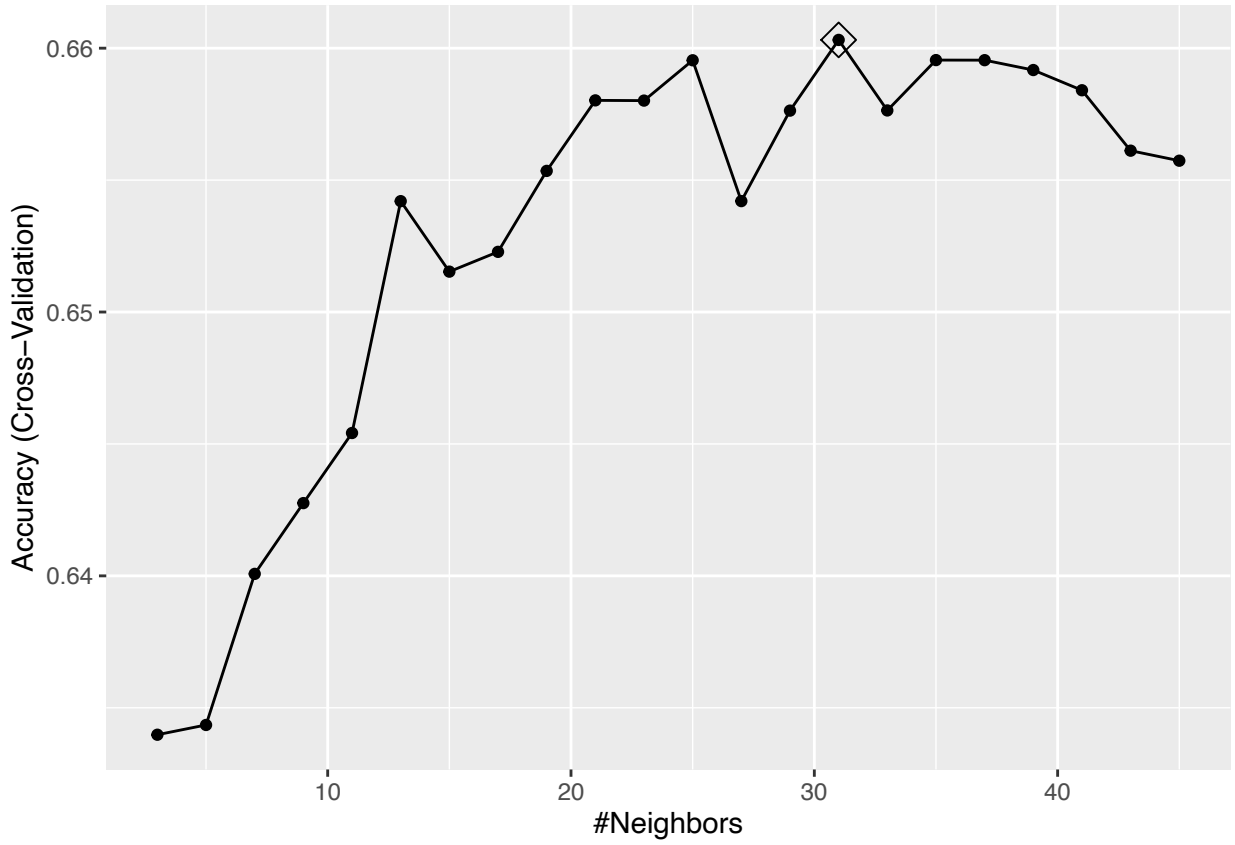
```
train_control <- trainControl(method = "cv", number = 10) #for algorithms to perform cross validation
```

```
#dataframe of results
res <- data.frame(matrix(nrow = 0, ncol = 4))
colnames(res) <- c("model", "data_set", "accuracy", "f_1")
```

- KNN:

```
#knn algorithm
set.seed(42, sample.kind = "default")
train_knn <- train(factor(Potability) ~ ., method = "knn",
                   tuneGrid = data.frame(k = seq(3, 45, 2)),
                   data = train_data_1,
                   trControl = train_control)
```

```
ggplot(train_knn, highlight = T)
```




```
train_knn$bestTune #best k for the algorithm
```

```
##     k
## 15 31
```

```
pred_knn <- predict(train_knn, test_data_1)
```

```
confusionMatrix(pred_knn,
                factor(test_data_1$Potability))$overall["Accuracy"]
```

```
##  Accuracy
## 0.6280488
```

```
#_1
F_meas(data = pred_knn, reference = factor(test_data_1$Potability))
```

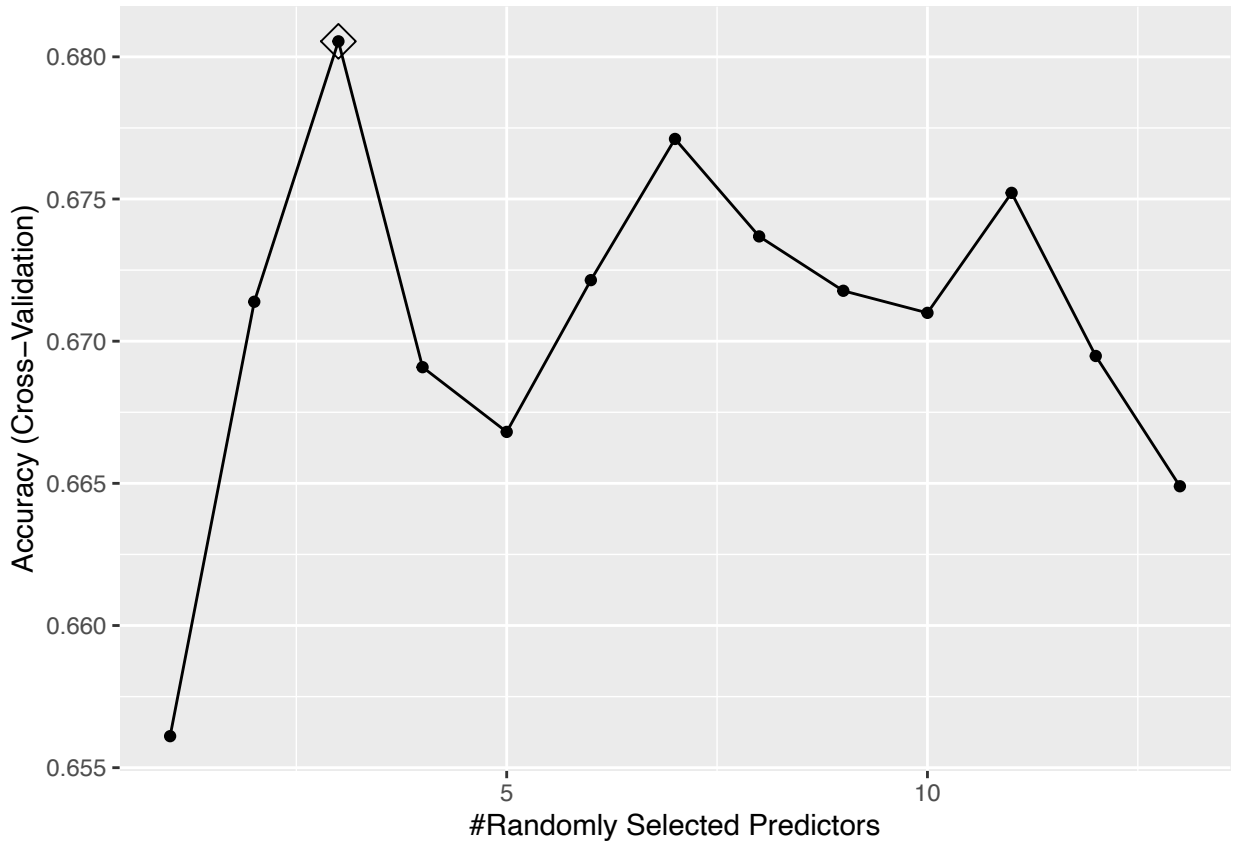
```
## [1] 0.7494867
```

```
#data stored
res[1,] <- c("KNN", 1, as.numeric(confusionMatrix(pred_knn,
                factor(test_data_1$Potability))$overall["Accuracy"]),
             F_meas(data = pred_knn, reference = factor(test_data_1$Potability)))
```

- Random Forest:

```
#random forest
set.seed(42, sample.kind = "default")
train_rf <- train(factor(Potability) ~., method = "rf",
                  data = train_data_1,
                  tuneGrid = data.frame(mtry = seq(3:15)), #3
                  trControl = train_control, #cross-validation
                  ntree = 200) #number of trees
```

```
ggplot(train_rf, highlight = T)
```




```
train_rf$bestTune
```

```
##   mtry
## 3    3
```

```
pred_rf <- predict(train_rf, test_data_1)
```

```
confusionMatrix(pred_rf,
                factor(test_data_1$Potability))$overall["Accuracy"]
```

```
##  Accuracy
## 0.6692073
```

```
#we store the data
res[nrow(res)+1,] <- c("rf", 1, as.numeric(confusionMatrix(pred_rf,
                factor(test_data_1$Potability))$overall["Accuracy"]),
             F_meas(data = pred_rf, reference = factor(test_data_1$Potability)))
```

- Logistic Regression:

```
set.seed(42, sample.kind = "default")
train_glm <- train(factor(Potability) ~., method = "glm",
                   data = train_data_1)
```

```r
pred_glm <- predict(train_glm, test_data_1)
```

```r
confusionMatrix(pred_glm,
                factor(test_data_1$Potability))$overall["Accuracy"]
```

```
##  Accuracy
## 0.6051829
```

```r
res[nrow(res)+1,] <- c("glm", 1, as.numeric(confusionMatrix(pred_glm,
                factor(test_data_1$Potability))$overall["Accuracy"]),
             F_meas(data = pred_glm, reference = factor(test_data_1$Potability)))
```

- XGBoost: The algorithm used is a variation of the xgbtree model. Since the aim is to perform parameter tuning, train_control1 is created to reduce the amount of iterations for the k-fold cross-validation and, thus, optimize time.

If the one who is seeing this project wants the training to be quicker, please substitute the parameters in expand.grid with the ones that are commented.

```r
library(xgboost)
```

```r
train_control1 <- trainControl(method = "cv", number = 3)
set.seed(42, sample.kind = "default")
train_xgb <- train(factor(Potability) ~., method = "xgbDART",
                  data = train_data_1,
                  trControl = train_control1,
                   tuneGrid = expand.grid(
                     nrounds = c(11),
                     max_depth = c(6, 7, 8),#8
                     eta = c(0, 0.01, 0.1, 0.2), #0.01
                     gamma = c(0, 0.001, 0.01), #0.01
                     subsample = 0.5,
                     colsample_bytree = c(0.5, 0.8, 1),#1
                     rate_drop = c(0, 0.4,0.5), #0
                     skip_drop = c(0, 0.4,0.5), #0.5
                     min_child_weight = c(0,1,2) #1
                   ))
```

```
## [18:17:32] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:17:32] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:17:32] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:17:32] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:17:32] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:17:32] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:17:32] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:17:32] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:17:32] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:17:32] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:17:32] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:17:32] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:17:32] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
```

```
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:47] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
## [18:22:48] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
```

```r
train_xgb$bestTune
```

```
##      nrounds max_depth  eta gamma subsample colsample_bytree rate_drop
## 2259      11         8 0.01  0.01       0.5                1         0
##      skip_drop min_child_weight
## 2259       0.5                1
```

```r
pred_xgb <- predict(train_xgb, test_data_1)
```

```
## [18:22:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: ‘ntree_limit‘ is deprecated, use ‘iterat
```

```
confusionMatrix(pred_xgb,
                factor(test_data_1$Potability))$overall["Accuracy"]
```

```
##  Accuracy
## 0.6615854
```

```
res[nrow(res)+1,] <- c("xgbDART", 1, as.numeric(confusionMatrix(pred_xgb,
                factor(test_data_1$Potability))$overall["Accuracy"]),
             F_meas(data = pred_xgb, reference = factor(test_data_1$Potability)))
```

- Naive Bayes:

```
library(naivebayes)
```

```
set.seed(42, sample.kind = "default")
train_nb <- train(factor(Potability) ~., data = train_data_1, method = "naive_bayes")
```

```
pred_nb <- predict(train_nb, test_data_1)
```

```
confusionMatrix(pred_nb,
                factor(test_data_1$Potability))$overall["Accuracy"]
```

```
##  Accuracy
## 0.6051829
```

```
res[nrow(res)+1,] <- c("naive_bayes", 1, as.numeric(confusionMatrix(pred_nb,
                factor(test_data_1$Potability))$overall["Accuracy"]),
             F_meas(data = pred_nb, reference = factor(test_data_1$Potability)))
```

- xgbTree: If the user wants a faster training, please set the parameters to the ones that are commented.

```
library(gbm)
```

```
set.seed(42, sample.kind = "default")
train_xgbT <- train(factor(Potability) ~., data = train_data_1,
                    method = "xgbTree",
                    trControl = train_control,
                    tuneGrid = expand.grid(
                      nrounds=c(63,65,70), #65
                      eta=c(0.01, 0.05), #0.05
                      max_depth=c(6, 7, 8),#6
                      colsample_bytree=c(0.5, 1), #1
                      subsample=0.5,
                      gamma=c(0.0001, 0.001, 0.01), #0.01
                      min_child_weight=c(0,1,2))) #0
```

```
## [18:22:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterati
## [18:22:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterati
```

```
train_xgbT$bestTune
```

```
##     nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
## 209      65         6 0.05  0.01                1                0       0.5
```

```
pred_xgbT <- predict(train_xgbT, test_data_1)
```

```
confusionMatrix(pred_xgbT,
                factor(test_data_1$Potability))$overall["Accuracy"]
```

```
##  Accuracy
## 0.6707317
```

```
res[nrow(res)+1,] <- c("xgbTree", 1, as.numeric(confusionMatrix(pred_xgbT,
                factor(test_data_1$Potability))$overall["Accuracy"]),
             F_meas(data = pred_xgbT, reference = factor(test_data_1$Potability)))
```

- QDA:

```
set.seed(42, sample.kind = "default")
train_qda <- train(factor(Potability) ~., data = train_data_1,
                   method = "qda")
```

```
pred_qda <- predict(train_qda, test_data_1)
```

```
confusionMatrix(pred_qda,
                factor(test_data_1$Potability))$overall["Accuracy"]
```

```
##  Accuracy
## 0.6615854
```

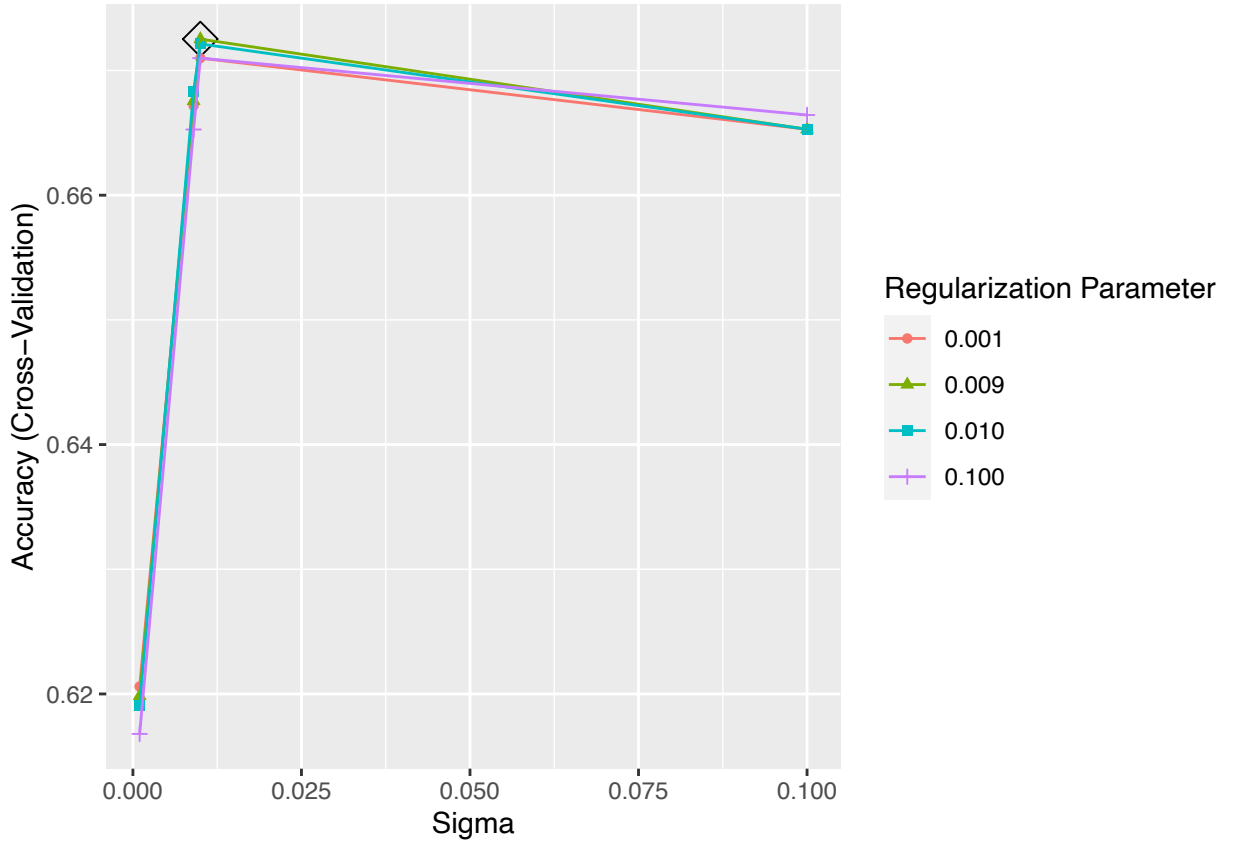
```
res[nrow(res)+1,] <- c("qda", 1, as.numeric(confusionMatrix(pred_qda,
                factor(test_data_1$Potability))$overall["Accuracy"]),
             F_meas(data = pred_qda, reference = factor(test_data_1$Potability)))
```

- Least Squares Support Vector Machine: If the user wants a faster training, please set the parameters to the ones that are commented.

```
library(kernlab)
```

```
set.seed(42, sample.kind = "default")
train_svm <- train(factor(Potability) ~., method = "lssvmRadial",
                   data = train_data_1,
                   tuneGrid = expand.grid(
                     tau = c(0.001,0.009,0.01,0.1),#0.009
                     sigma = c(0.001,0.009,0.01,0.1)#0.01
                   ),
                   trControl = train_control)
```

```
ggplot(train_svm, highlight = T)
```




```
train_svm$bestTune
```

```
##   sigma   tau
## 7  0.01 0.009
```

```
pred_svm <- predict(train_svm, test_data_1)
```

```
confusionMatrix(pred_svm,
                factor(test_data_1$Potability))$overall["Accuracy"]
```

```
##  Accuracy
## 0.6509146
```

```
res[nrow(res)+1,] <- c("lssvmRadial", 1, as.numeric(confusionMatrix(pred_svm,
                factor(test_data_1$Potability))$overall["Accuracy"]),
             F_meas(data = pred_svm, reference = factor(test_data_1$Potability)))
```

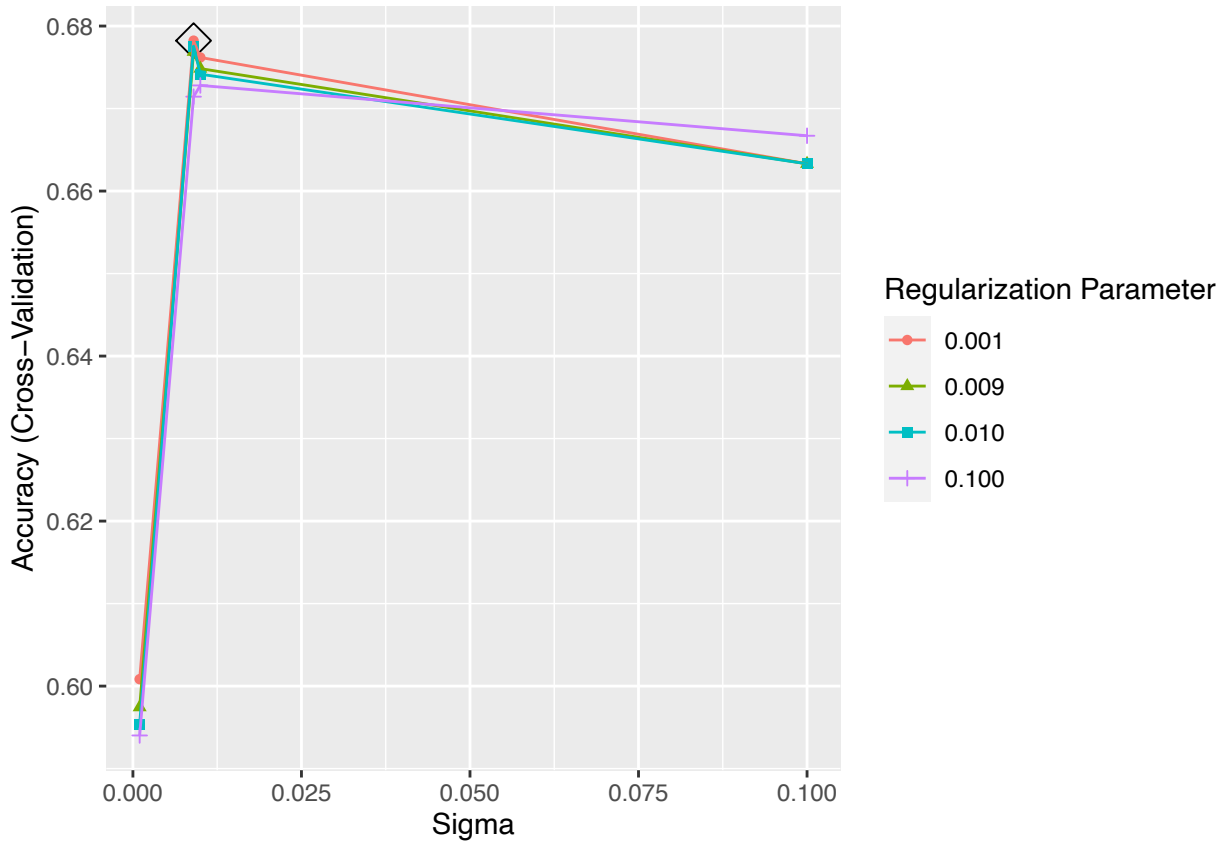
A new train set, and test set, is created to see if the accuracy can improve. In this case, the data sets have NA-containing rows removed:

```
set.seed(42, sample.kind = "default")
#new data sets
test_index1 <- createDataPartition(y, times=1, p=0.27, list=F)
train_data1 <- wp[-test_index1,]
test_data1 <- wp[test_index1,]
train_data_2 <- train_data1[complete.cases(train_data1),] %>% ScaleData() #scaled clean data without NA
test_data_2 <- test_data1[complete.cases(test_data1),] %>% ScaleData() #scaled clean data without NA ro
```

- Least Squares Support Vector Machine: If the user wants a faster training, please set the parameters to the ones that are commented.

```
set.seed(42, sample.kind = "default")
train_svm1 <- train(factor(Potability) ~., method = "lssvmRadial",
                    data = train_data_2,
                    tuneGrid = expand.grid(
                      tau = c(0.001,0.009,0.01,0.1), # 0.001
                      sigma = c(0.001,0.009,0.01,0.1) #0.009
                    ),
                    trControl = train_control)
```

```
ggplot(train_svm1, highlight = T)
```



```
train_svm1$bestTune
```

```
##   sigma   tau
## 2 0.009 0.001
```

```
pred_svm1 <- predict(train_svm1, test_data_2)
```

```
confusionMatrix(pred_svm1,
                factor(test_data_2$Potability))$overall["Accuracy"]
```

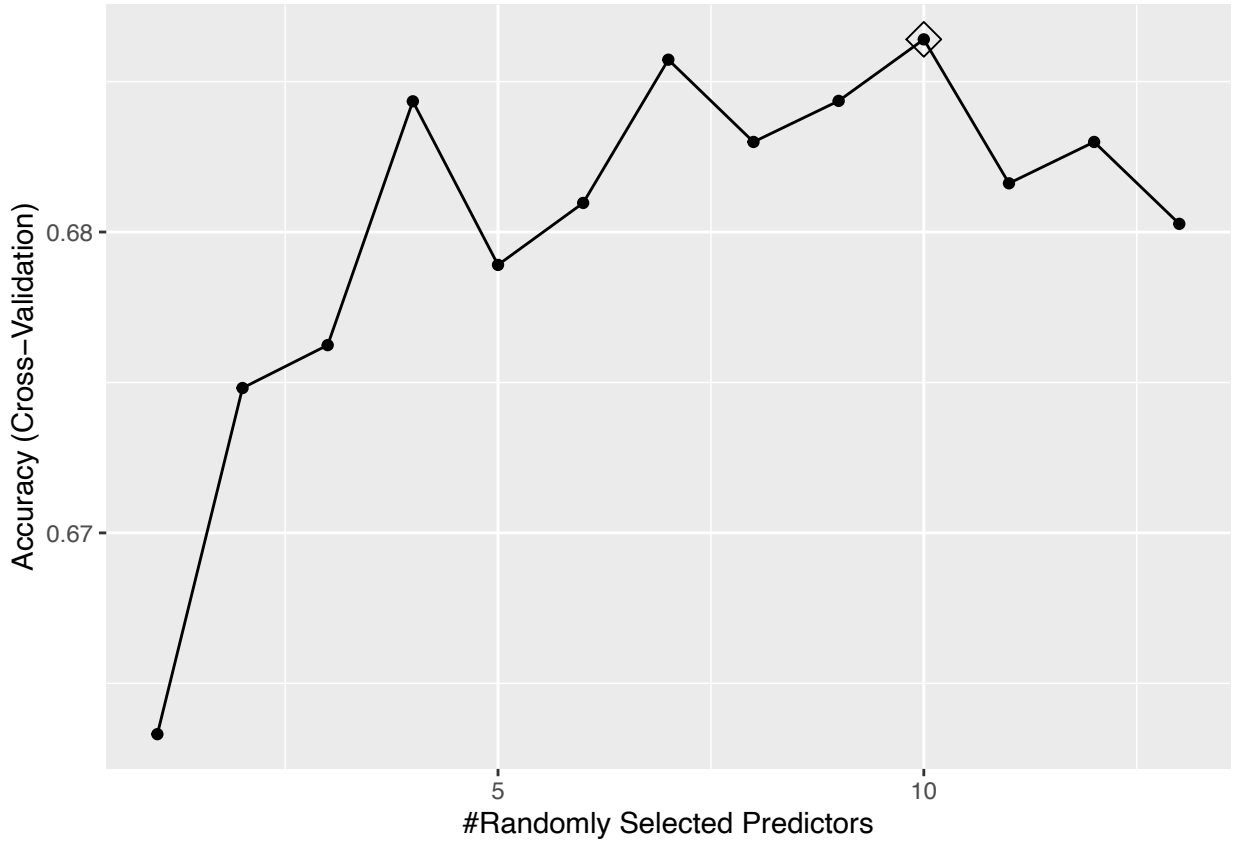
```
##  Accuracy
## 0.6579926
```

```
res[nrow(res)+1,] <- c("lssvmRadial", 2, as.numeric(confusionMatrix(pred_svm1,
                factor(test_data_2$Potability))$overall["Accuracy"]),
             F_meas(data = pred_svm1, reference = factor(test_data_2$Potability)))
```

*Random Forest:

```
#random forest
set.seed(42, sample.kind = "default")
train_rf1 <- train(factor(Potability) ~., method = "rf",
                   data = train_data_2,
                   tuneGrid = data.frame(mtry = seq(3:15)), #parameter tuning mtry = 5
                   trControl = train_control, #cross-validation
                   ntree = 200) #number of trees
```

```
ggplot(train_rf1, highlight = T)
```




```
pred_rf1 <- predict(train_rf1,
                    test_data_2)
```

```
confusionMatrix(pred_rf1,
                factor(test_data_2$Potability))$overall["Accuracy"]
```

```
##  Accuracy
## 0.6654275
```

```
res[nrow(res)+1,] <- c("rf", 2, as.numeric(confusionMatrix(pred_rf1,
                factor(test_data_2$Potability))$overall["Accuracy"]),
             F_meas(data = pred_rf1, reference = factor(test_data_2$Potability)))
```

- xgbTree: If the user wants a faster training, please set the parameters to the ones that are commented.

```
set.seed(42, sample.kind = "default")
train_xgbT1 <- train(factor(Potability) ~., data = train_data_2,
                     method = "xgbTree",
                     trControl = train_control,
                     tuneGrid = expand.grid(
                       nrounds=c(63,65,70), #70
                       eta=c(0.01, 0.05), #0.01
                       max_depth=c(6, 7, 8),#8
                       colsample_bytree=c(0.5, 1), #1
```

```r
                      subsample=0.5,
                      gamma=c(0.0001, 0.001, 0.01), #0.001
                      min_child_weight=c(0,1,2))) #1
```

```
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:51] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:52] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:53] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
## [18:33:54] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterat
```

```
## [18:36:07] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterati
## [18:36:07] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterati
## [18:36:07] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterati
## [18:36:07] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterati
## [18:36:07] WARNING: amalgamation/../src/c_api/c_api.cc:785: `ntree_limit` is deprecated, use `iterati
```

```
train_xgbT1$bestTune
```

```
##     nrounds max_depth  eta gamma colsample_bytree min_child_weight subsample
## 141      70         8 0.01 0.001                1                1       0.5
```

```
pred_xgbT1 <- predict(train_xgbT1, test_data_2)
```

```
confusionMatrix(pred_xgbT1,
                factor(test_data_2$Potability))$overall["Accuracy"]
```

```
##  Accuracy
## 0.6858736
```

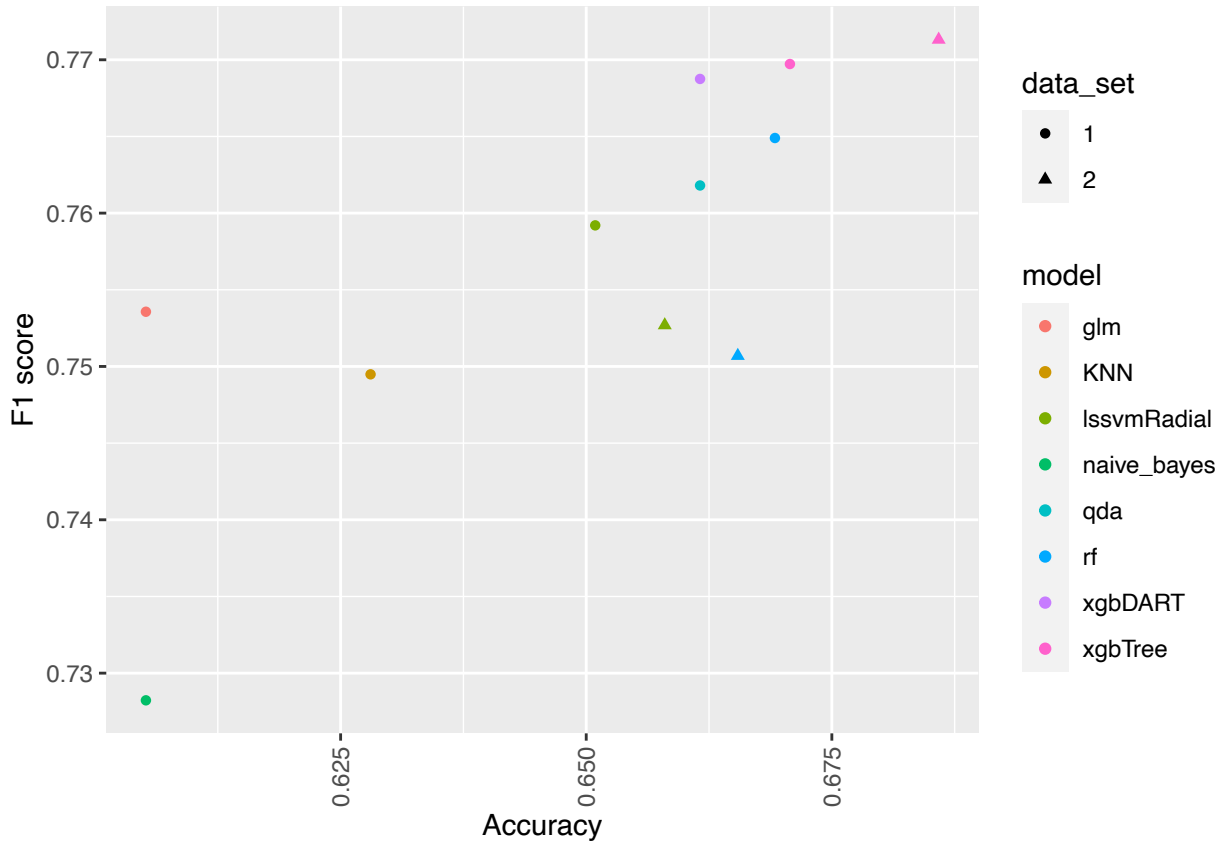
```
res[nrow(res)+1,] <- c("xgbTree", 2, as.numeric(confusionMatrix(pred_xgbT1,
                factor(test_data_2$Potability))$overall["Accuracy"]),
             F_meas(data = pred_xgbT1, reference = factor(test_data_2$Potability)))
```

RESULTS:

From the plot below, we can see that by using the scaled data sets which had their NA-rows discarded (train_data_2 and test_data_2) and by training an XgbTree model, the highest accuracy and F1 score is obtained out of all cases. On the other hand, the Naive Bayes model trained and tested with train_data_1 and test_data_1, respectively, had the lowest results.

We can see that in both data sets, the model that performs the best is xgbTree.

```
res %>% ggplot(aes(as.numeric(accuracy),
                   as.numeric(f_1), col = model, shape = data_set)) +
  geom_point() +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
  xlab("Accuracy") + ylab("F1 score")
```




```
res %>% filter(accuracy == max(accuracy) & f_1 == max(f_1)) #best result
```

```
##     model data_set          accuracy               f_1
## 1 xgbTree        2 0.685873605947955 0.771312584573748
```

CONCLUSIONS:

After following the steps to achieve the aim of this project, it was found that using the data that was scaled and had its NA-rows removed, along with the optimized xgbTree algorithm, the best results were obtained for both accuracy and F1 score.

This algorithm is crucial, since it helps determine the quality of water based on specific variables, which can tell the user if it is safe for drinking or not. However, the algorithm has its limitations, since the accuracy and F1 score are not above 0.95. For the improvement of it, higher computational power might be needed to find better parameters (enhanced parameter tuning), and, perhaps, training other classification models that were not used in this report.