UNIVERSITÀ degli STUDI di CATANIA | DIPARTIMENTO di ECONOMIA e IMPRESA

Data Science – University of Catania

WATER POTABILITY

Statistical Learning | Professor Ingrassia

Asad Aslam | 1000053142
Sameer Afzal | 1000053143
Harsh Mehta | 1000055155
Usama Ali | 1000053161
Khizar Alam | 1000055853

Contents

1. Introduction

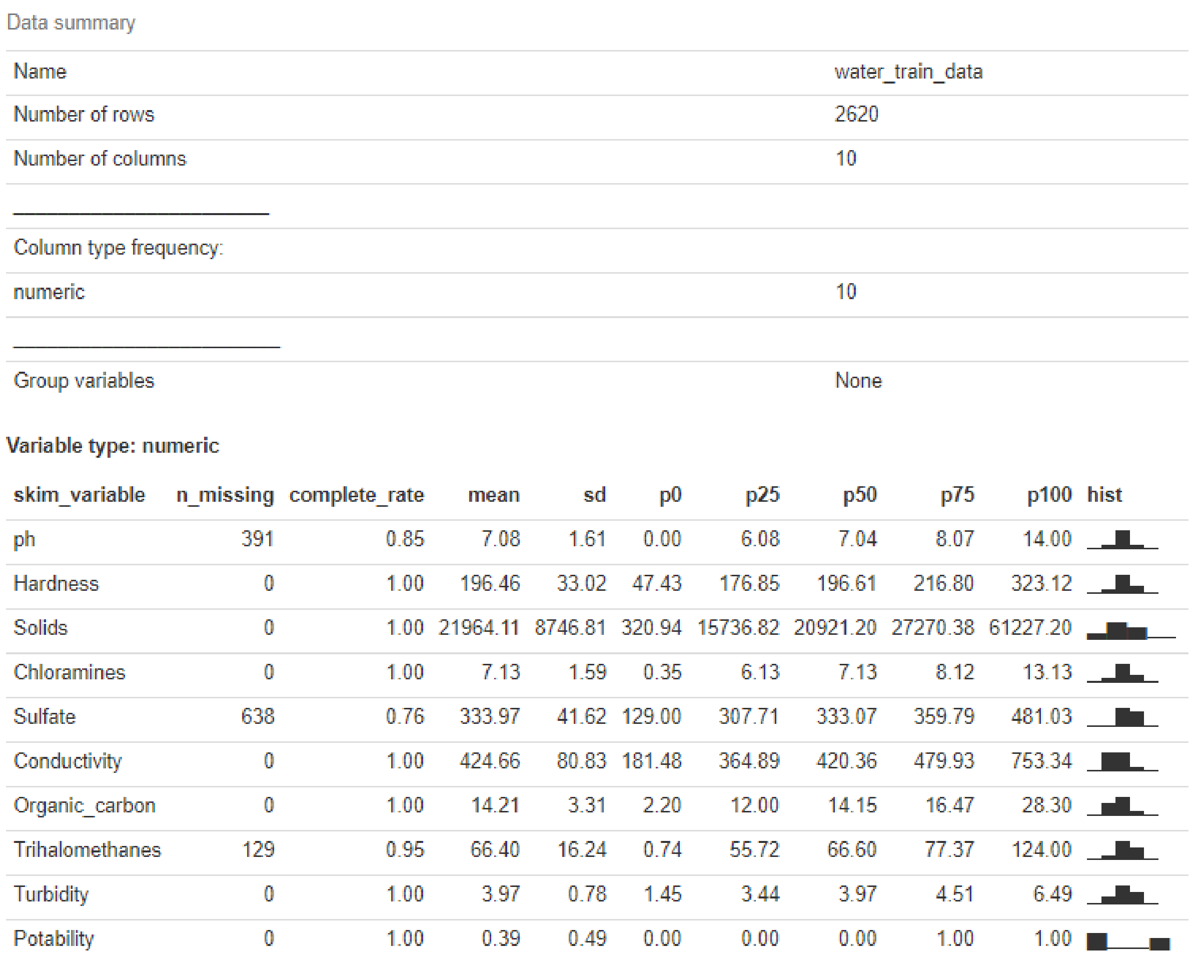
The dataset for this report analysis is "Water Potability". This report we are doing EDA - Exploratory Data Analysis, Model the train data according to three different approaches, which includes logistic regression, random forests and neutral networks. Furthermore, we will compare the results and choose the best model. After that, we will apply the model to the test data and predict our target variable **"Portability"**.

2. Question to Analysis

The aim of this study is to predict the water pot ability and the factors affecting the water quality.

1. Which model best fit the data? How precise is it to describe our data?
2. Is there a variable (or more than one) that is not relevant for our analysis?
3. What can we observe analyzing the relationships between the target variable and the predictors?

3. EDA – Exploratory Data Analysis

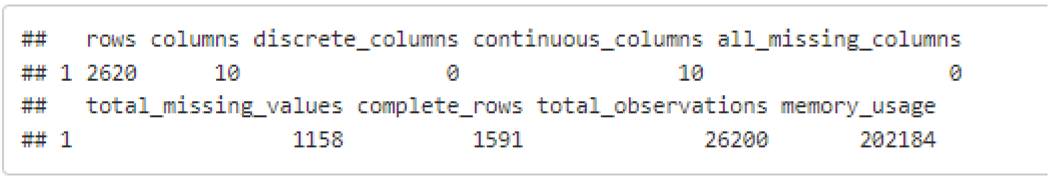


Data summary

Name	water_train_data
Number of rows	2620
Number of columns	10
_______________________	
Column type frequency:	
numeric	10
________________________	
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
ph	391	0.85	7.08	1.61	0.00	6.08	7.04	8.07	14.00	
Hardness	0	1.00	196.46	33.02	47.43	176.85	196.61	216.80	323.12	
Solids	0	1.00	21964.11	8746.81	320.94	15736.82	20921.20	27270.38	61227.20	
Chloramines	0	1.00	7.13	1.59	0.35	6.13	7.13	8.12	13.13	
Sulfate	638	0.76	333.97	41.62	129.00	307.71	333.07	359.79	481.03	
Conductivity	0	1.00	424.66	80.83	181.48	364.89	420.36	479.93	753.34	
Organic_carbon	0	1.00	14.21	3.31	2.20	12.00	14.15	16.47	28.30	
Trihalomethanes	129	0.95	66.40	16.24	0.74	55.72	66.60	77.37	124.00	
Turbidity	0	1.00	3.97	0.78	1.45	3.44	3.97	4.51	6.49	
Potability	0	1.00	0.39	0.49	0.00	0.00	0.00	1.00	1.00	



```
##   rows columns discrete_columns continuous_columns all_missing_columns
## 1 2620      10                0                 10                   0
##   total_missing_values complete_rows total_observations memory_usage
## 1                 1158          1591              26200       202184
```

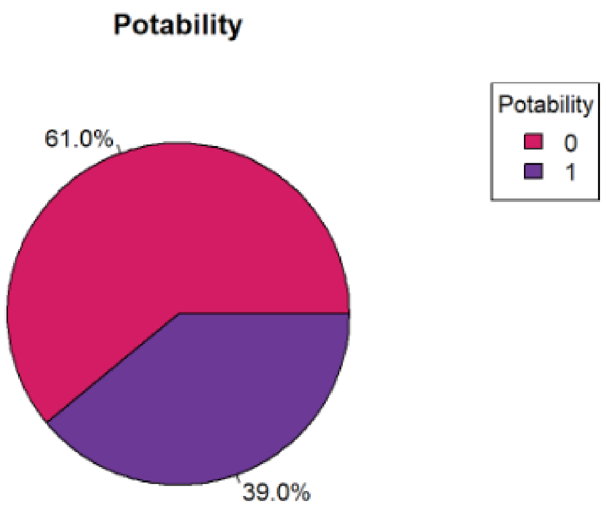
Data set Demonstrate:

1. It is difficult to determine if portability is an integer/binary type or not.
2. Target variable is coded as 1 for indicating safe for human consumption and 0 for not safe.

3.1. Factoring the Variable

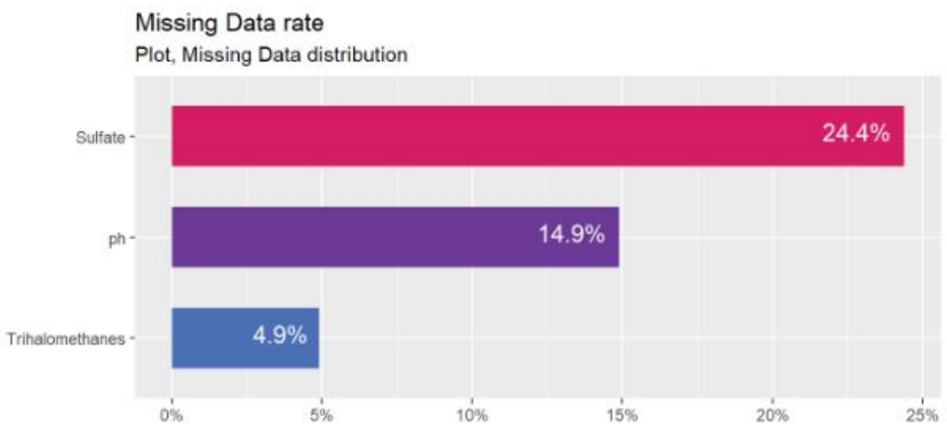
```
Potability <- as.factor(water_train_data$Potability)
summary(Potability)
```

```
##    0    1
## 1598 1022
```

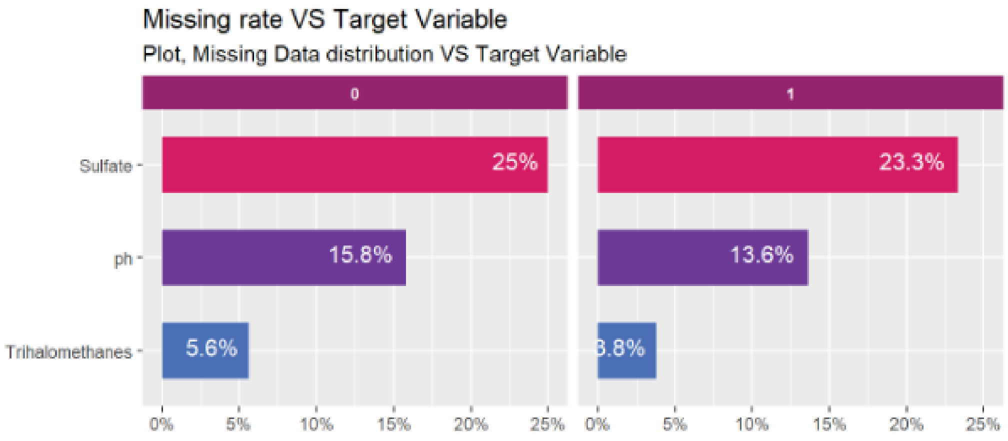


3.2. Missing Data Count

```
##              ph        Hardness          Solids     Chloramines         Sulfate
##             391               0               0               0             638
##    Conductivity  Organic_carbon Trihalomethanes       Turbidity      Potability
##               0               0             129               0               0
```



3.3. Missing Data VS Target Variable

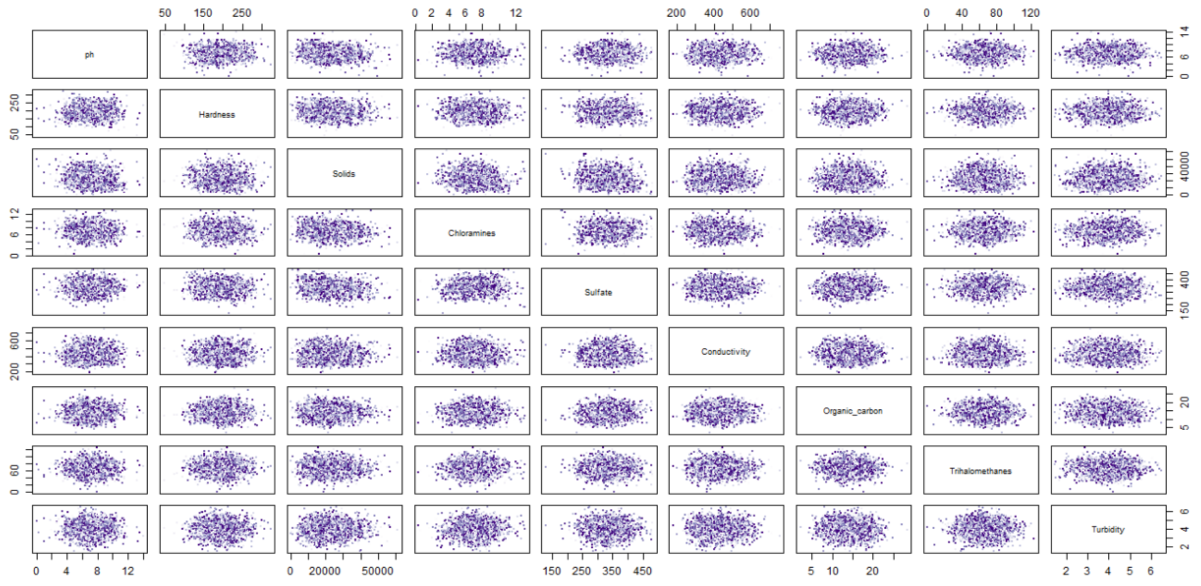


We have three options:

1. Remove missing data records but we will lose significant number of records, and this may harm the model.
2. Replace them with the mean.
3. Use mice function in R to impute missing values.

We are going with impute missing value with using mice package in R. MICE stands for *Multivariate Imputation via Chained Equations*, and it is one of the most common packages for R users. It assumes the missing values are missing at random (MAR). The basic idea behind the algorithm is to treat each variable that has missing values as a dependent variable in regression and treat the others as independent (predictors). There are three MICE imputation method. Predictive mean matching, classification & regression tree, and lasso linear regression. We are using predictive mean matching.

3.4. Scatter-Plot



We think, No multi collinearity; the predictors are not strongly correlated, so let us keep them all as they are!

4. Modeling of Training Data.

4.1. Logistic Regression.

We use glm function for creating logistic regression models. We will use all variables except potability for predictors.

With MICE

```
Call:
glm(formula = Potability ~ ., family = "binomial", data = water_train)

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)     -1.057e+00  7.262e-01  -1.456   0.1454
ph               2.777e-02  2.785e-02   0.997   0.3188
Hardness        -4.286e-04  1.360e-03  -0.315   0.7527
Solids           1.061e-05  5.248e-06   2.021   0.0433 *
Chloramines      3.522e-02  2.825e-02   1.247   0.2125
Sulfate          3.540e-04  1.091e-03   0.324   0.7457
Conductivity    -2.285e-04  5.626e-04  -0.406   0.6846
Organic_carbon  -2.367e-02  1.361e-02  -1.740   0.0819 .
Trihalomethanes  1.454e-03  2.753e-03   0.528   0.5975
Turbidity        5.415e-02  5.768e-02   0.939   0.3478
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2797.5  on 2095  degrees of freedom
Residual deviance: 2787.2  on 2086  degrees of freedom
AIC: 2807.2

Number of Fisher Scoring iterations: 4
```

With Removing the Columns

```
Call:
glm(formula = Potability ~ ., family = "binomial", data =
water_train_removal_column)

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)    -6.346e-01  5.300e-01  -1.197   0.2312
Hardness       -3.613e-04  1.347e-03  -0.268   0.7885
Solids          9.963e-06  5.150e-06   1.935   0.0530 .
Chloramines     3.482e-02  2.822e-02   1.234   0.2171
Conductivity   -2.415e-04  5.623e-04  -0.430   0.6675
Organic_carbon -2.308e-02  1.356e-02  -1.702   0.0888 .
Turbidity       5.153e-02  5.756e-02   0.895   0.3707
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2797.5  on 2095  degrees of freedom
Residual deviance: 2788.6  on 2089  degrees of freedom
AIC: 2802.6

Number of Fisher Scoring iterations: 4
```

Using the Step Function with direction backward.

```
Call:
glm(formula = Potability ~ Solids + Organic_carbon, family = "binomial",
    data = water_train)

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)    -3.436e-01  2.247e-01  -1.529   0.1262
Solids          9.582e-06  5.114e-06   1.874   0.0610 .
Organic_carbon -2.323e-02  1.354e-02  -1.716   0.0862 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2797.5  on 2095  degrees of freedom
Residual deviance: 2791.2  on 2093  degrees of freedom
AIC: 2797.2

Number of Fisher Scoring iterations: 4
```

```
Call:
glm(formula = Potability ~ Solids + Organic_carbon, family = "binomial",
    data = water_train_removal_column)

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)    -3.436e-01  2.247e-01  -1.529   0.1262
Solids          9.582e-06  5.114e-06   1.874   0.0610 .
Organic_carbon -2.323e-02  1.354e-02  -1.716   0.0862 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 2797.5  on 2095  degrees of freedom
Residual deviance: 2791.2  on 2093  degrees of freedom
AIC: 2797.2

Number of Fisher Scoring iterations: 4
```

From the summary of model, we see Solid and Organic_Carbon are near to significance of target variable.

Now let us do the validation of the data and see how well our model is train.

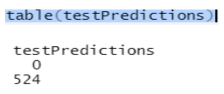
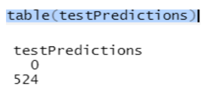
```
validationPredictions   0   1 Sum
                  0   313 211 524
                  Sum 313 211 524
```

```
validationPredictions   0   1 Sum
                  0   313 211 524
                  Sum 313 211 524
```

Confusion Matrix and Statistics

```
            Reference
Prediction   0   1
         0 313 211
         1   0   0

               Accuracy : 0.5973
                 95% CI : (0.5539, 0.6396)
    No Information Rate : 0.5973
    P-Value [Acc > NIR] : 0.5189

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.0000
            Specificity : 1.0000
         Pos Pred Value :    NaN
         Neg Pred Value : 0.5973
             Prevalence : 0.4027
         Detection Rate : 0.0000
   Detection Prevalence : 0.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 1
```

```
            Reference
Prediction   0   1
         0 313 211
         1   0   0

               Accuracy : 0.5973
                 95% CI : (0.5539, 0.6396)
    No Information Rate : 0.5973
    P-Value [Acc > NIR] : 0.5189

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.0000
            Specificity : 1.0000
         Pos Pred Value :    NaN
         Neg Pred Value : 0.5973
             Prevalence : 0.4027
         Detection Rate : 0.0000
   Detection Prevalence : 0.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 1
```

```
table(testPredictions)

testPredictions
  0
524
```

```
table(testPredictions)

testPredictions
  0
524
```

We have done the logistic regression with two types of data. In first, we use mice function to impute the missing values. On the other hand, we removed the columns having missing value because the percentage of missing value was so high. After performing the logistic regression, we see that Solid and Organic_Carbon are near to significance of target variable. Moreover, the p value are also same with having imputed values and removal of columns in data sets. After that, we run our model on validation data to predict the target variable and compare it with the actual value of target variable so we can find the accuracy of our model. The confusion matrix and statistics show us that:

True negatives (TN): 313

False positives (FP): 211

False negatives (FN): 0

True positives (TP): 0

Accuracy: The accuracy of the model calculated as the proportion of correct predictions out of the total predictions. In this case, the accuracy is 0.5973, meaning that approximately 59.73% of the predictions are correct.

Confidence Interval (CI): The confidence interval provides a range of values within which the true accuracy of the model is likely to fall. The 95% CI for the accuracy is between 0.5539 and 0.6396.

No Information Rate (NIR): The no information rate is the accuracy that would be achieve by predicting the majority class in the data. In this case, the NIR is 0.5973, which is the same as the accuracy of the model.

P-Value: The p-value compares the model's accuracy to the no information rate to determine if the model's accuracy is statistically significantly different. In this case, the p-value is 0.5189, indicating that there is no significant difference between the model's accuracy and the no information rate.

Kappa: Kappa is a statistic that measures the agreement between the predicted values and the actual values, taking into account the possibility of agreement by chance. In this case, the kappa value is 0, indicating no agreement beyond what would be expected by chance.

Test P-Value: McNemar's test is a statistical test that compares the differences in errors between two models. In this case, the p-value is less than 2e-16, suggesting a significant difference in errors between the two models.

Sensitivity: Sensitivity, also known as true positive rate or recall, measures the proportion of actual positive cases correctly identified by the model. In this case, the sensitivity is 0.0000, indicating that the model did not correctly identify any positive cases.

Specificity: Specificity measures the proportion of actual negative cases correctly identified by the model. In this case, the specificity is 1.0000, indicating that the model correctly identified all negative cases.

Positive Predictive Value: Positive predictive value, also known as precision, measures the proportion of positive predictions that are correct. In this case, the positive predictive value is NaN (not a number) because there are no true positive cases.

Negative Predictive Value: Negative predictive value measures the proportion of negative predictions that are correct. In this case, the negative predictive value is 0.5973, indicating that approximately 59.73% of the negative predictions are correct.

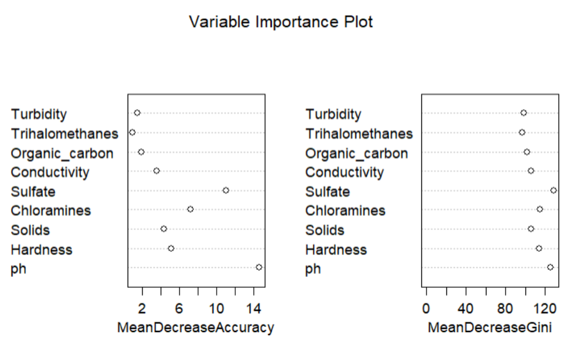
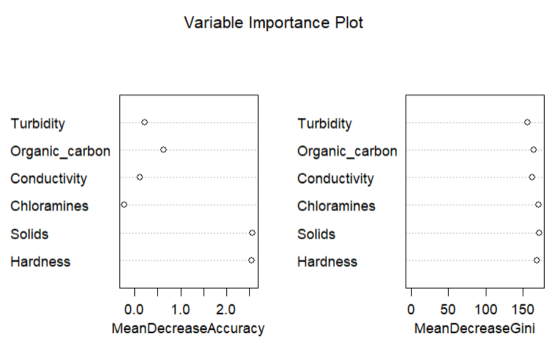
Prevalence: Prevalence represents the proportion of positive cases in the data. In this case, the prevalence is 0.4027, indicating that approximately 40.27% of the cases are positive.

Detection Rate: Detection rate, also known as true positive rate or recall, measures the proportion of actual positive cases correctly identified by the model. In this case, the detection rate is 0.0000, indicating that the model did not correctly identify any positive cases.

Detection Prevalence: Detection prevalence represents the proportion of cases correctly identified as positive by the model. In this case, the detection prevalence is 0.0000

4.2. Random Forests.

The second method applied to fit our data is the so-called Random forest. It is an ensemble method (learning techniques with the aim of constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. Below, it shows the importance of the variables according to two indices, the Mean Decrease Accuracy and the Mean Decrease Gini index. The Mean Decrease Accuracy refers to the mean decrease of accuracy predictions on the OOB (out of bag) samples, when a given variable is excluded by the model; the Mean Decrease in Gini coefficient is a measure of how each variable contributes to the homogeneity of the nodes and leaves (foglie) in the resulting random forest.

With Mice	With Column Removal
	

On the left side plot, it is clear that across all of the trees considered in the random forest, Sulfate and ph are the most relevant predictors. On the other hand, Solids and Hardness are the most relevant predictors.

Using Random Forests yields us to the following results:

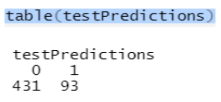
OOB 0.3640267	OOB 0.4031489

Now let us do the validation of the data and see how well our model is train. Confusion Matrix and Statistics.

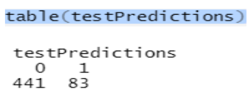
```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 269 143
         1  44  68

               Accuracy : 0.6431
                 95% CI : (0.6004, 0.6842)
    No Information Rate : 0.5973
    P-Value [Acc > NIR] : 0.01767

                  Kappa : 0.1967

 Mcnemar's Test P-Value : 7.696e-13

            Sensitivity : 0.3223
            Specificity : 0.8594
         Pos Pred Value : 0.6071
         Neg Pred Value : 0.6529
             Prevalence : 0.4027
         Detection Rate : 0.1298
   Detection Prevalence : 0.2137
      Balanced Accuracy : 0.5908

       'Positive' Class : 1
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 260 159
         1  53  52

               Accuracy : 0.5954
                 95% CI : (0.552, 0.6378)
    No Information Rate : 0.5973
    P-Value [Acc > NIR] : 0.5543

                  Kappa : 0.084

 Mcnemar's Test P-Value : 5.537e-13

            Sensitivity : 0.24645
            Specificity : 0.83067
         Pos Pred Value : 0.49524
         Neg Pred Value : 0.62053
             Prevalence : 0.40267
         Detection Rate : 0.09924
   Detection Prevalence : 0.20038
      Balanced Accuracy : 0.53856

       'Positive' Class : 1
```

Prediction on Test data



```
table(testPredictions)

 testPredictions
   0   1
 431  93
```



```
table(testPredictions)

 testPredictions
   0   1
 441  83
```

As we select, here data with mice imputed value because the accuracy of this data is more than the other one.

Confusion Matrix:

True negatives (0, 0): 269

False positives (0, 1): 143

False negatives (1, 0): 44

True positives (1, 1): 68

Accuracy: The accuracy of the model is 0.6431, which means that 64.31% of the predictions were correct.

Confidence Interval (CI): The accuracy has a 95% confidence interval of (0.6004, 0.6842). This means that we are 95% confident that the true accuracy of the model lies within this range.

No Information Rate (NIR): The no information rate is 0.5973, which represents the accuracy that could be achieve by always predicting the majority class. In this case, the no information rate is the same as the accuracy, indicating that the model is not performing significantly better than randomly guessing.

P-Value: The p-value for accuracy being greater than the no information rate is 0.01767. This suggests that the model's accuracy is significantly better than randomly guessing.

Kappa: The kappa coefficient is 0.1967, which measures the agreement between the actual and predicted classes. A kappa value less than 0.2 indicates poor agreement.

McNemar's Test: The p-value for McNemar's test is 7.696e-13, which indicates a significant difference in performance between the model's predictions for different classes.

Sensitivity: The sensitivity, also known as the true positive rate, is 0.3223. It represents the proportion of actual positives correctly identified by the model.

Specificity: The specificity, also known as the true negative rate, is 0.8594. It represents the proportion of actual negatives correctly identified by the model.

Positive Predictive Value: The positive predictive value is 0.6071, which indicates the proportion of positive predictions that are correct.

Negative Predictive Value: The negative predictive value is 0.6529, which indicates the proportion of negative predictions that are correct.

Prevalence: The prevalence is 0.4027, which represents the proportion of positive instances in the dataset.

Detection Rate: The detection rate is 0.1298, which indicates the proportion of actual positive instances that were correctly identify by the model.

Detection Prevalence: The detection prevalence is 0.2137, which represents the proportion of instances predicted as positive by the model.

Balanced Accuracy: The balanced accuracy is 0.5908, which takes into account both sensitivity and specificity and provides an overall measure of classification performance.

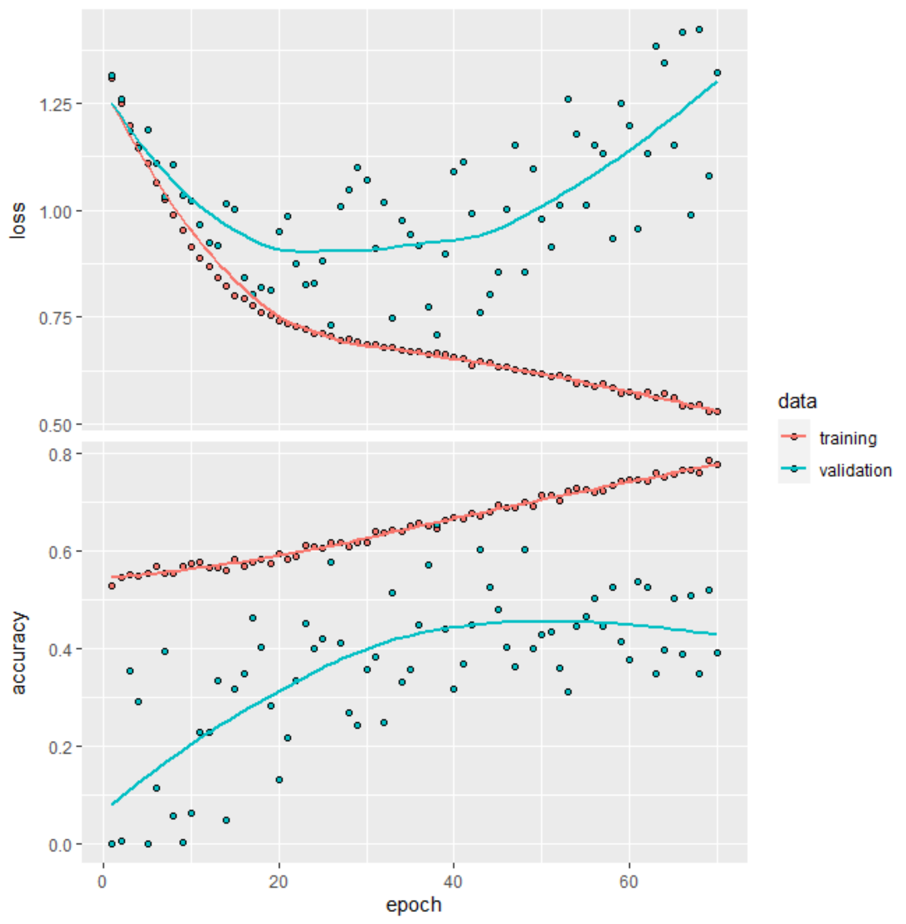
'Positive' Class: The positive class labeled as one in the confusion matrix, indicating the class of interest for the analysis.

4.3. Neural Networks.

Neural networks are a type of machine learning algorithm that are inspired by the human brain. They are made up of layers of interconnected nodes, each of which performs a simple mathematical operation. The nodes are arranged in a hierarchy, with the nodes in the first layer receiving input data, the nodes in the last layer producing output data, and the nodes in the middle layers performing intermediate calculations.

With Mice			**Without Mice**		
	Reference			Reference	
Prediction	0	1	Prediction	0	1
0	290	130	0	282	138
1	63	41	1	60	44

```
> evaluation
     loss  accuracy
0.7454050 0.6815578
```



The loss during training data is 74% and the accuracy of this model is 68%

5. Results.

Model Name	Accuracy	Specificity	Sensitivity
Logistic Regression with MICE	59.73%	1.00	0.00
Logistic Regression without MICE	59.73%	1.00	0.00
Random forests with MICE	64.31%	0.8594	0.3223
Random forests without MICE	59.54%	0.83067	0.24645
Neural Networks with MICE	68.15578%	0.86	0.34
Neural Networks without MICE	68.15578%	0.86	0.34

6. Conclusion.

In the start of report, we write some questions, which we are going to analysis in this report. Below are the final answers of the those questions

Which model best fit the data?

According to the accuracy, Neural networks model fits best to our data with 68.15% of accuracy.

Is there a variable (or more than one) that is not relevant for our analysis?

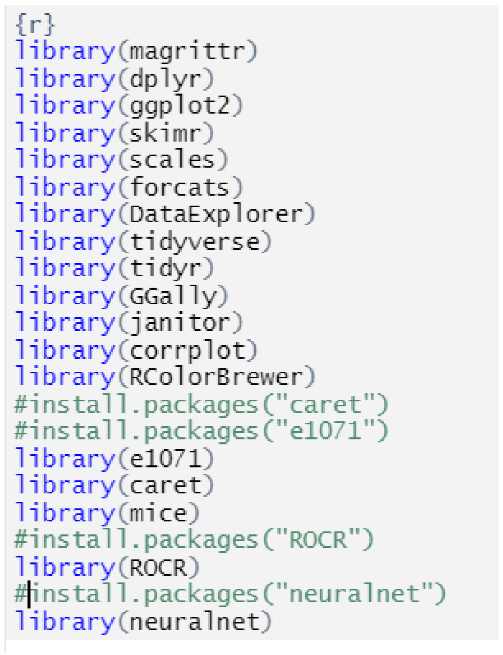
According to the analysis and the data, no variable is much relevant for the prediction but we found only two variables with were near to the threshold value of 0.05 which are Solids and Organic_Carbon

What can we observe analyzing the relationships between the target variable and the predictors?

According to the correlation of the data with targeted variable, no predictors are have strong relationship.

7. Appendix.

Libraries used in this analysis

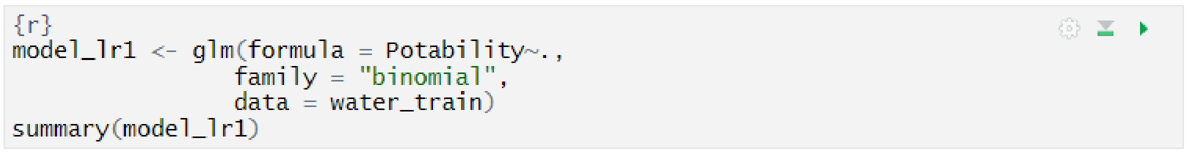


```
{r}
library(magrittr)
library(dplyr)
library(ggplot2)
library(skimr)
library(scales)
library(forcats)
library(DataExplorer)
library(tidyverse)
library(tidyr)
library(GGally)
library(janitor)
library(corrplot)
library(RColorBrewer)
#install.packages("caret")
#install.packages("e1071")
library(e1071)
library(caret)
library(mice)
#install.packages("ROCR")
library(ROCR)
#install.packages("neuralnet")
library(neuralnet)
```

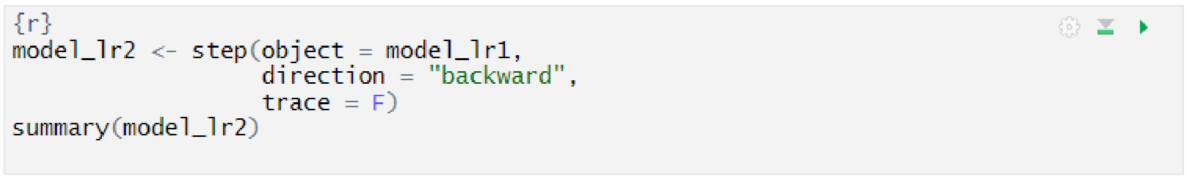
Modeling Code

Logistic Regression with mice

Train The Model

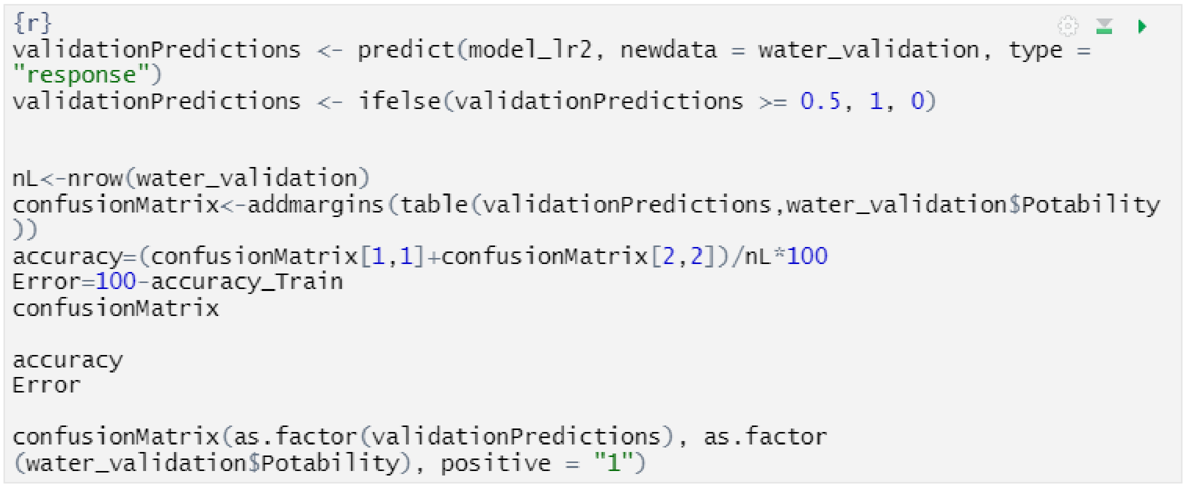


```
{r}
model_lr1 <- glm(formula = Potability~.,
                 family = "binomial",
                 data = water_train)
summary(model_lr1)
```



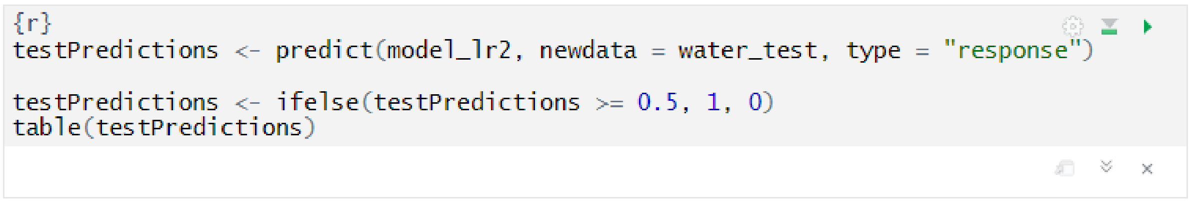
```
{r}
model_lr2 <- step(object = model_lr1,
                  direction = "backward",
                  trace = F)
summary(model_lr2)
```

Validation Model



```
{r}
validationPredictions <- predict(model_lr2, newdata = water_validation, type =
"response")
validationPredictions <- ifelse(validationPredictions >= 0.5, 1, 0)


nL<-nrow(water_validation)
confusionMatrix<-addmargins(table(validationPredictions,water_validation$Potability
))
accuracy=(confusionMatrix[1,1]+confusionMatrix[2,2])/nL*100
Error=100-accuracy_Train
confusionMatrix

accuracy
Error

confusionMatrix(as.factor(validationPredictions), as.factor
(water_validation$Potability), positive = "1")
```

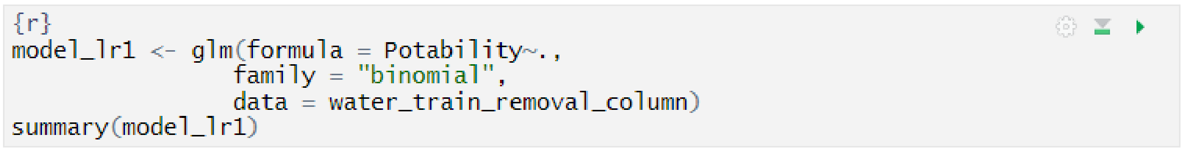
Test Model



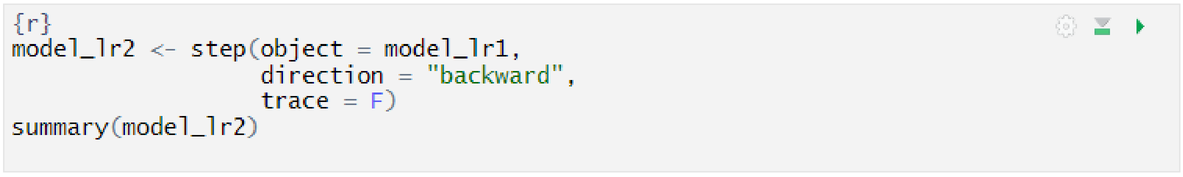
```
{r}
testPredictions <- predict(model_lr2, newdata = water_test, type = "response")

testPredictions <- ifelse(testPredictions >= 0.5, 1, 0)
table(testPredictions)
```

Logistic Regression with Removal of column

Train The Model

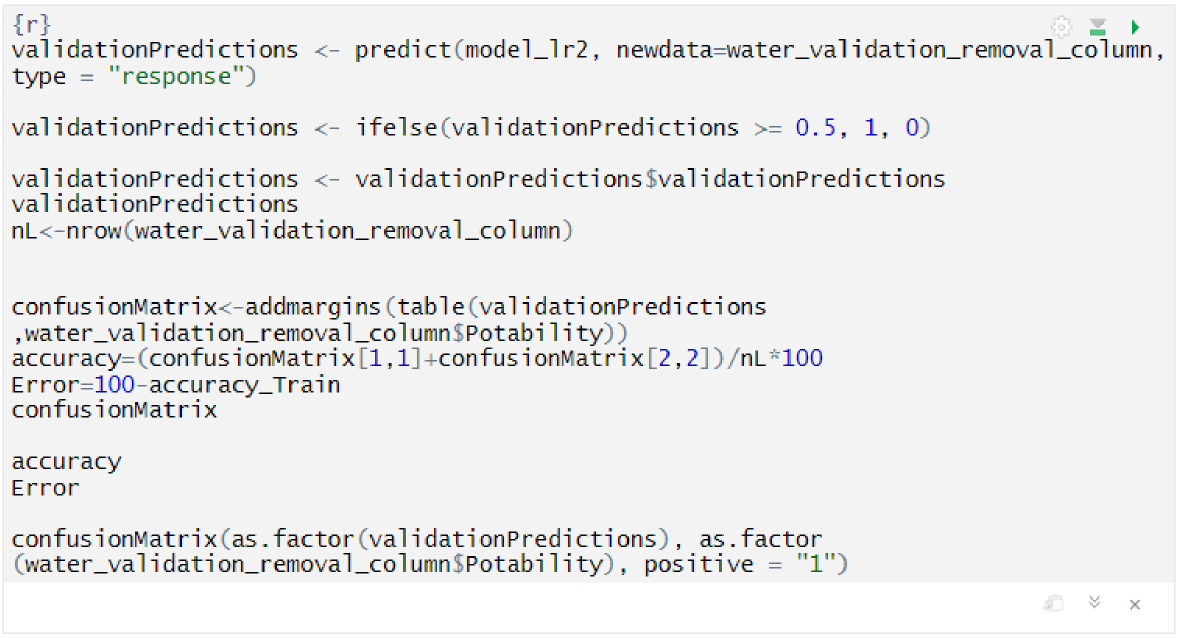


```
{r}
model_lr1 <- glm(formula = Potability~.,
                family = "binomial",
                data = water_train_removal_column)
summary(model_lr1)
```



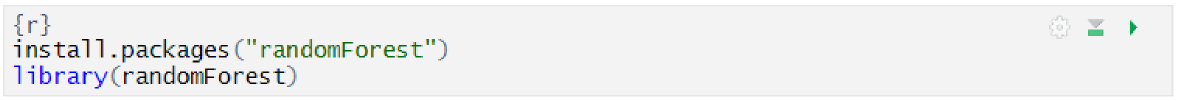
```
{r}
model_lr2 <- step(object = model_lr1,
                  direction = "backward",
                  trace = F)
summary(model_lr2)
```

Validation Model



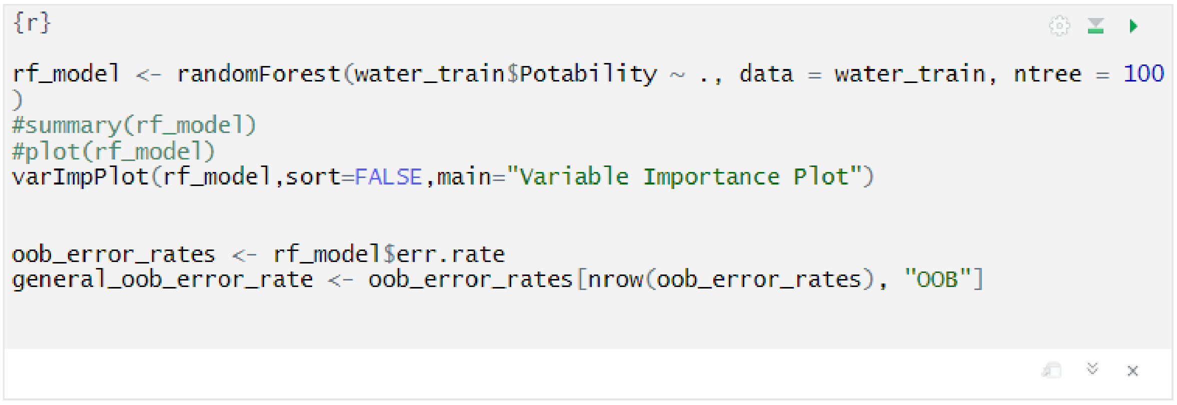
```
{r}
validationPredictions <- predict(model_lr2, newdata=water_validation_removal_column,
type = "response")

validationPredictions <- ifelse(validationPredictions >= 0.5, 1, 0)

validationPredictions <- validationPredictions$validationPredictions
validationPredictions
nL<-nrow(water_validation_removal_column)


confusionMatrix<-addmargins(table(validationPredictions
,water_validation_removal_column$Potability))
accuracy=(confusionMatrix[1,1]+confusionMatrix[2,2])/nL*100
Error=100-accuracy_Train
confusionMatrix

accuracy
Error

confusionMatrix(as.factor(validationPredictions), as.factor
(water_validation_removal_column$Potability), positive = "1")
```

Random forests with MICE



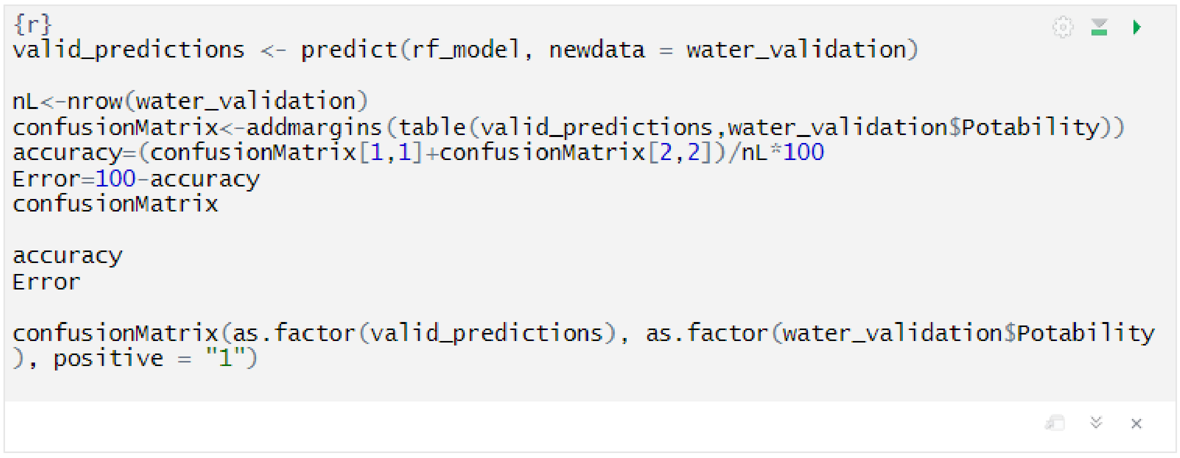
```
{r}
install.packages("randomForest")
library(randomForest)
```

Training



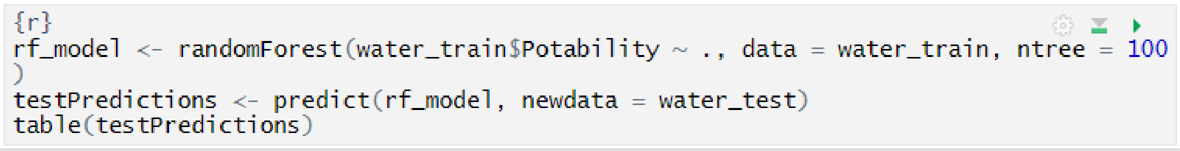
```
{r}

rf_model <- randomForest(water_train$Potability ~ ., data = water_train, ntree = 100
)
#summary(rf_model)
#plot(rf_model)
varImpPlot(rf_model,sort=FALSE,main="Variable Importance Plot")


oob_error_rates <- rf_model$err.rate
general_oob_error_rate <- oob_error_rates[nrow(oob_error_rates), "OOB"]
```

Validation



```
{r}
valid_predictions <- predict(rf_model, newdata = water_validation)

nL<-nrow(water_validation)
confusionMatrix<-addmargins(table(valid_predictions,water_validation$Potability))
accuracy=(confusionMatrix[1,1]+confusionMatrix[2,2])/nL*100
Error=100-accuracy
confusionMatrix

accuracy
Error

confusionMatrix(as.factor(valid_predictions), as.factor(water_validation$Potability
), positive = "1")
```

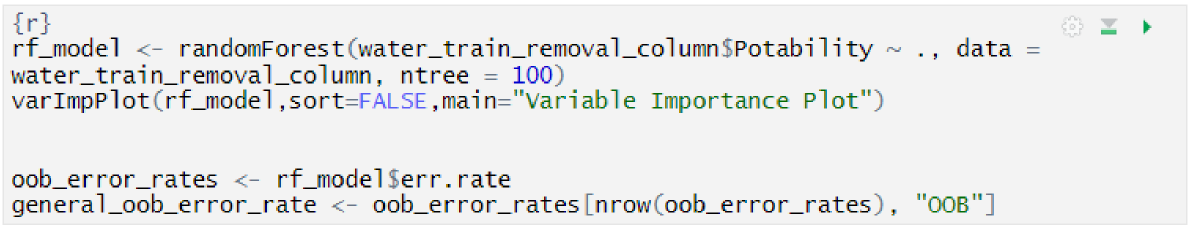
Test Model



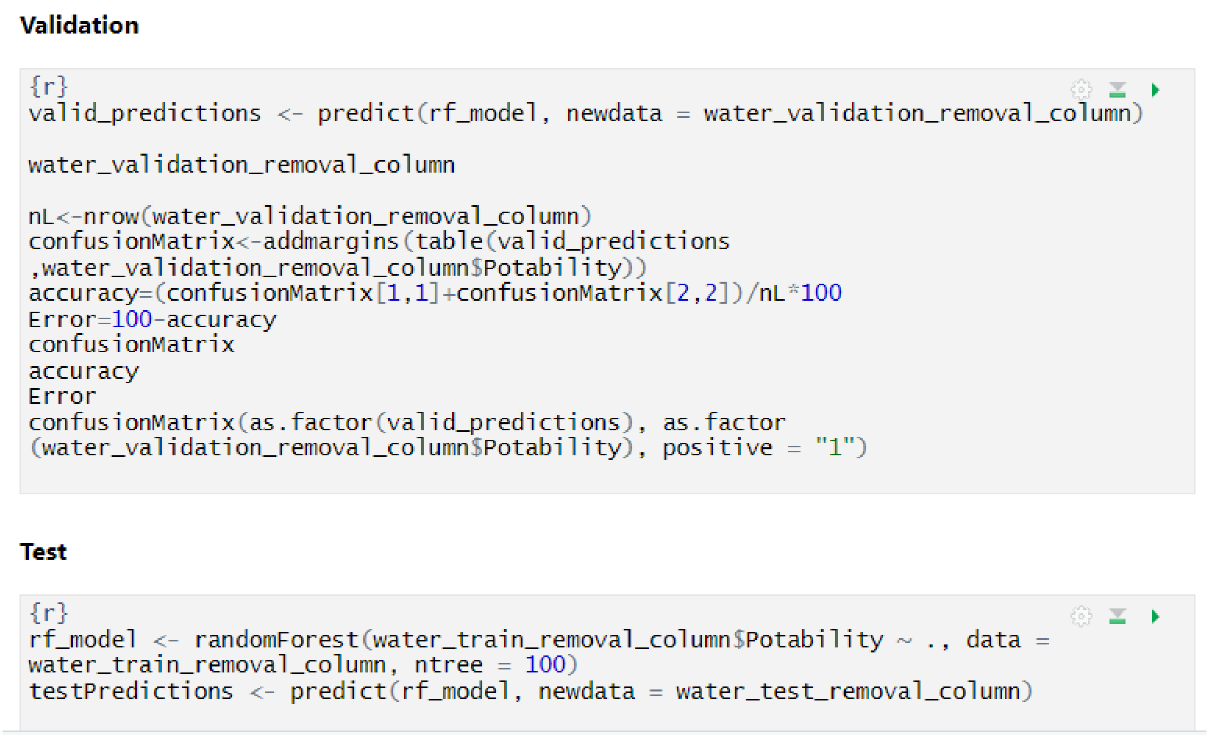
```
{r}
rf_model <- randomForest(water_train$Potability ~ ., data = water_train, ntree = 100
)
testPredictions <- predict(rf_model, newdata = water_test)
table(testPredictions)
```

Random forests with removal of column

Training



```
{r}
rf_model <- randomForest(water_train_removal_column$Potability ~ ., data =
water_train_removal_column, ntree = 100)
varImpPlot(rf_model,sort=FALSE,main="Variable Importance Plot")


oob_error_rates <- rf_model$err.rate
general_oob_error_rate <- oob_error_rates[nrow(oob_error_rates), "OOB"]
```



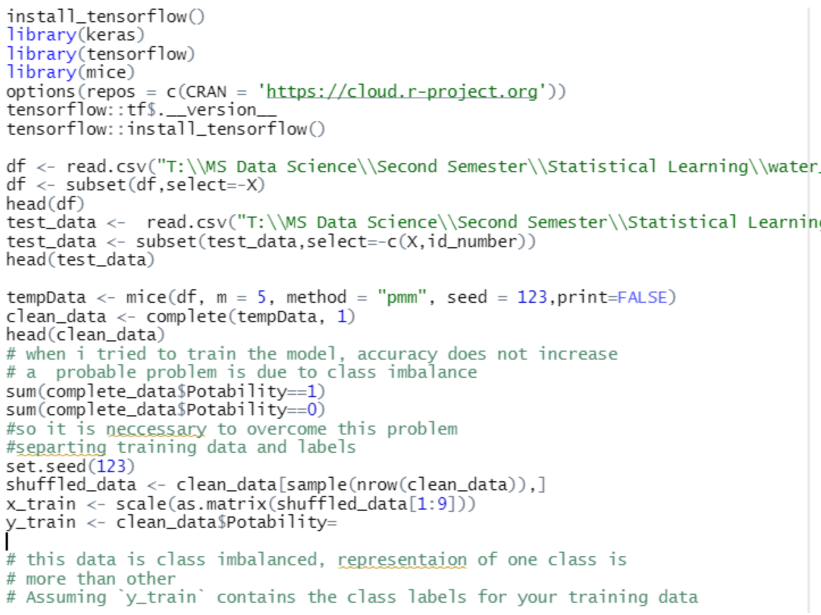
Validation

```
{r}
valid_predictions <- predict(rf_model, newdata = water_validation_removal_column)

water_validation_removal_column

nL<-nrow(water_validation_removal_column)
confusionMatrix<-addmargins(table(valid_predictions
,water_validation_removal_column$Potability))
accuracy=(confusionMatrix[1,1]+confusionMatrix[2,2])/nL*100
Error=100-accuracy
confusionMatrix
accuracy
Error
confusionMatrix(as.factor(valid_predictions), as.factor
(water_validation_removal_column$Potability), positive = "1")
```

Test

```
{r}
rf_model <- randomForest(water_train_removal_column$Potability ~ ., data =
water_train_removal_column, ntree = 100)
testPredictions <- predict(rf_model, newdata = water_test_removal_column)
```

Neural Networks



```
install_tensorflow()
library(keras)
library(tensorflow)
library(mice)
options(repos = c(CRAN = 'https://cloud.r-project.org'))
tensorflow::tf$.__version__
tensorflow::install_tensorflow()

df <- read.csv("T:\\MS Data Science\\Second Semester\\Statistical Learning\\water_potability_train.csv")
df <- subset(df,select=-X)
head(df)
test_data <-  read.csv("T:\\MS Data Science\\Second Semester\\Statistical Learning\\water_potability_test.csv")
test_data <- subset(test_data,select=-c(X,id_number))
head(test_data)

tempData <- mice(df, m = 5, method = "pmm", seed = 123,print=FALSE)
clean_data <- complete(tempData, 1)
head(clean_data)
# when i tried to train the model, accuracy does not increase
# a  probable problem is due to class imbalance
sum(complete_data$Potability==1)
sum(complete_data$Potability==0)
#so it is neccessary to overcome this problem
#separting training data and labels
set.seed(123)
shuffled_data <- clean_data[sample(nrow(clean_data)),]
x_train <- scale(as.matrix(shuffled_data[1:9]))
y_train <- clean_data$Potability=

# this data is class imbalanced, representaion of one class is
# more than other
# Assuming `y_train` contains the class labels for your training data
```

```
class_counts <- table(y_train)
total_samples <- length(y_train)
class_weights <- 1 / (class_counts / total_samples)
class_weights <- setNames(class_weights, names(class_weights))
# as the model does not training properly
# so trying differnt techniques based on trial and error method
# using L2_regularizer, dropout layer and batch normalization layer
# creating the model
modnn <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu",kernel_regularizer = regularizer_l2(0.001),
              input_shape =ncol(x_train)) %>%
  layer_dense(units = 128, activation = "relu",kernel_regularizer = regularizer_l2(0.001))%>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 128, activation = "relu",kernel_regularizer = regularizer_l2(0.001))%>%
  #layer_dropout(rate = 0.2)%>%
  layer_dense(units = 128, activation = "relu",kernel_regularizer = regularizer_l2(0.001)) %>%
  layer_batch_normalization()%>%
  layer_dense(units = 128, activation = "relu",kernel_regularizer = regularizer_l2(0.001)) %>%
  #layer_dropout(rate = 0.2)%>%
  layer_dense(units = 64, activation = "relu",kernel_regularizer = regularizer_l2(0.001)) %>%
  layer_dropout(rate = 0.3)%>%
  layer_dense(units = 64, activation = "relu",kernel_regularizer = regularizer_l2(0.001)) %>%
  layer_dense(units = 1,activation="sigmoid")
# compiling model
modnn %>% compile(loss = "binary_crossentropy",
                  optimizer = optimizer_rmsprop(learning_rate = 0.001),
                  metrics = list("accuracy"),
                  )

history <- modnn %>% fit(
  x_train, y_train, epochs = 70, batch_size = 32,
  validation_split = 0.3,class_weight = list(class_weights))

x11()
plot(history)
# Description
```