Data 622 Assignment 4

Keith DeNivo

Determining water potability

For my job I use analytical chemistry techniques to identify and quantify toxic substances that are in drinking water in New York State. Examples of these substances are industrial waste products and poisons like pesticides. Federal and state laws regulate how often water needs to be tested and the maximum concentration of the chemicals that are allowed for human consumption. If a drinking water supply has one of these toxic substances above the threshold, then the water is deemed unsafe to drink and the supplier must correct the problem or face legal repercussions.

The dataset selected has features of water quality indicators. The target variable is a class of whether the water is potable, or it is not potable. It would be helpful to have simple indicators that would help determine whether water is safe to drink or not. Currently there are several costly methods that require specialized and expensive equipment in order to determine if the not dangerous. Many methods require the use of mass spectrometers which are expensive and need laborious maintenance and specialized skills. Many of the tests listed on the dataset require relatively inexpensive measures, such as pH, turbidity, and conductivity. So, if there were a few quick tests that could indicate that water is safe with good certainty, that potentially could save a lot of communities especially in resource limited areas.

In general, these types of models could be a good flag for quality control. If companies are selling food or liquids to consume, it would be good to determine from a only some tests whether a batch is good or not. It may be utilized to raise some flags for the product from a few tests even if those few tests are in compliance with regulatory standards. So building a model to help interpret the quality of the consumable from easier and quicker tests may help protect people and save money.

For the dataset there are 9 water quality indicators and the target class. Each variable/feature has very different units and are on very different scales. Features such as solids, hardness,

and sulfates, their values were in the hundreds to thousands. The rest had values that were significantly less than that. The there were several features with missing values. For some methods such as decision trees rescaling is not as critical. For the data was rescaled in order to work with other machine learning algorithms such as SVM neural networks. Though with the exception of neural networks the data that was not scaled often produced models that had better or similar performance to models where the data was scaled.

Roughly 1/3 of the data had missing values. Most of the models run into issues when there are missing values. To prevent this, tthe main way I decided to fix it was to impute the missing values with the median of each feature, specifically from the training data set. The features are not well correlated, so they are largely independent of each other.

Outliers were determined using the Mahalanobis distances, which is usually used for more correlated data. Ultimately, for analysis, the outliers were not removed. The outliers are good indicators of when water will not be potable since they would be outside the acceptable thresholds. Using a Chi-square of 97.5% there were ~140 outliers out of the 3276 observations. Some models were trained with those outliers removed and without them being removed for comparison. For most of the features The classes are not separated well. They have similar means and standard deviations. One notiable exception is pH which the potable water is more concentrated around the mean than the nonpotable water is. So visually there is at least a little separation between the classes with the nonpotable water surrounding the other.

Models were built from the modifieddatasets below

Data tested:

Train_data = not scaled median imputed train data

Test_data = not scaled median imputed train data

Train_scaled= scaled median imputed train data

Test_scaled= scaled median imputed train data

clean_train_data= scaled, MICE imputed, outlier removed train data

clean_trest_data= scaled, MICE imputed, outlier removed test data

Target Variable is water potability class 1 , 0

DT, RF, ADABOOST, SVM, Neural Network

Read in Data

```r
file_url <- "https://raw.githubusercontent.com/division-zero/Data-622/refs/heads/main/Assignm
# Download the file to a temporary location
temp_file <- tempfile(fileext = ".csv")
download.file(file_url, destfile = temp_file, mode = "wb")
# Read the csv file
fulldata <- read.delim(file_url, sep = ",", header = TRUE, stringsAsFactors = FALSE)
# View the data
head(fulldata)
```

```
        ph Hardness   Solids Chloramines  Sulfate Conductivity Organic_carbon
1       NA 204.8905 20791.32    7.300212 368.5164     564.3087      10.379783
2 3.716080 129.4229 18630.06    6.635246       NA     592.8854      15.180013
3 8.099124 224.2363 19909.54    9.275884       NA     418.6062      16.868637
4 8.316766 214.3734 22018.42    8.059332 356.8861     363.2665      18.436524
5 9.092223 181.1015 17978.99    6.546600 310.1357     398.4108      11.558279
6 5.584087 188.3133 28748.69    7.544869 326.6784     280.4679       8.399735
  Trihalomethanes Turbidity Potability
1        86.99097  2.963135          0
2        56.32908  4.500656          0
3        66.42009  3.055934          0
4       100.34167  4.628771          0
5        31.99799  4.075075          0
6        54.91786  2.559708          0
```

```r
# Clean up the temporary file
unlink(temp_file)
```

Potability is the target class variable. Features include: Sulfate, Conductivity, Total Organic Carbon, hardness, solids, chloramines, pH, trihalomethanes, turbidity.

Basic data info

```r
summary(fulldata)
```

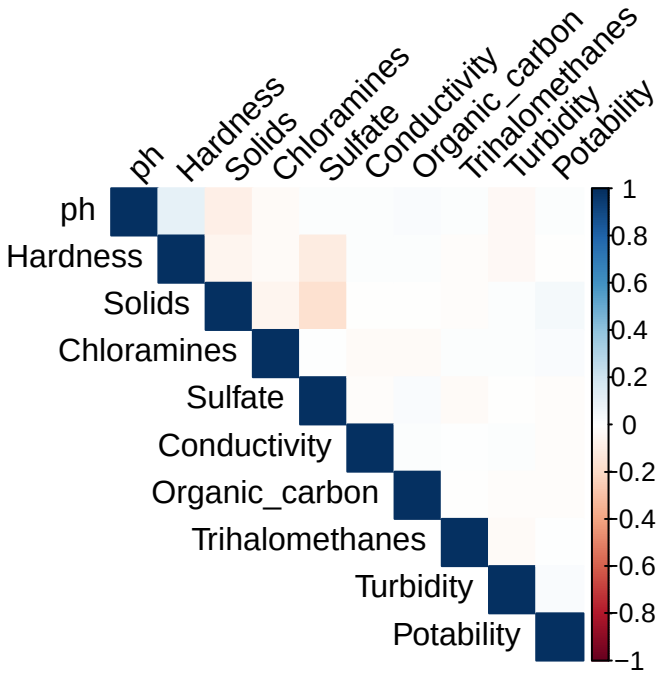
```
       ph            Hardness          Solids         Chloramines    
 Min.   : 0.000   Min.   : 47.43   Min.   :  320.9   Min.   : 0.352  
 1st Qu.: 6.093   1st Qu.:176.85   1st Qu.:15666.7   1st Qu.: 6.127  
 Median : 7.037   Median :196.97   Median :20927.8   Median : 7.130  
 Mean   : 7.081   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
```

```r
colSums(is.na(fulldata)) #see how many NA values
```

```
           ph        Hardness          Solids     Chloramines         Sulfate
          491               0               0               0             781
  Conductivity  Organic_carbon Trihalomethanes       Turbidity      Potability
            0               0             162               0               0
```

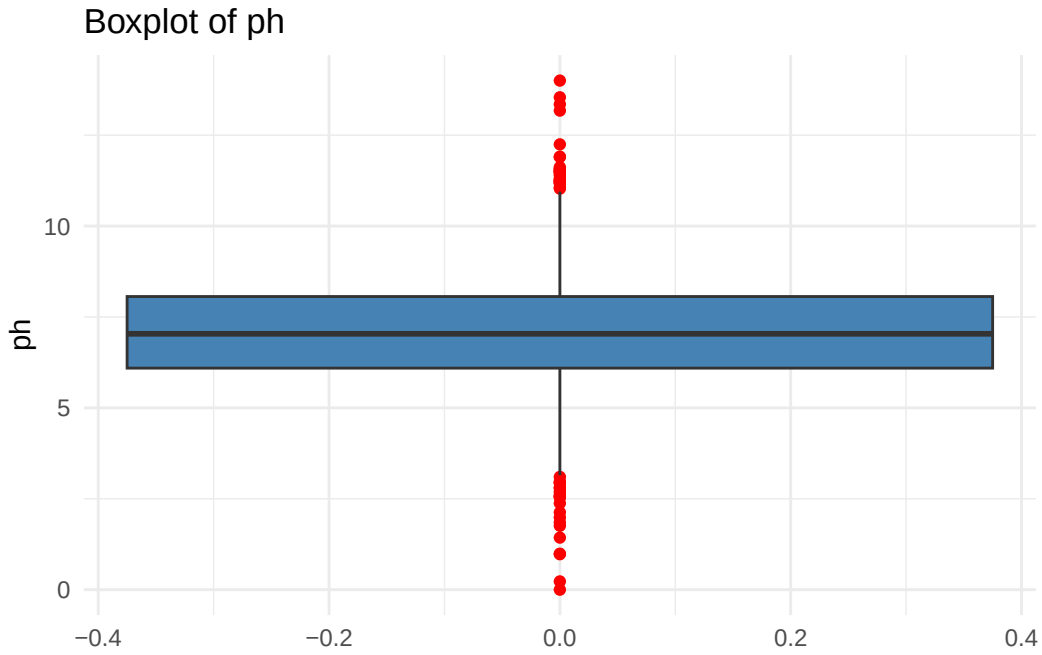
```r
cor_matrix <- cor(fulldata, use = "complete.obs")

#round(cor_matrix, 3)

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

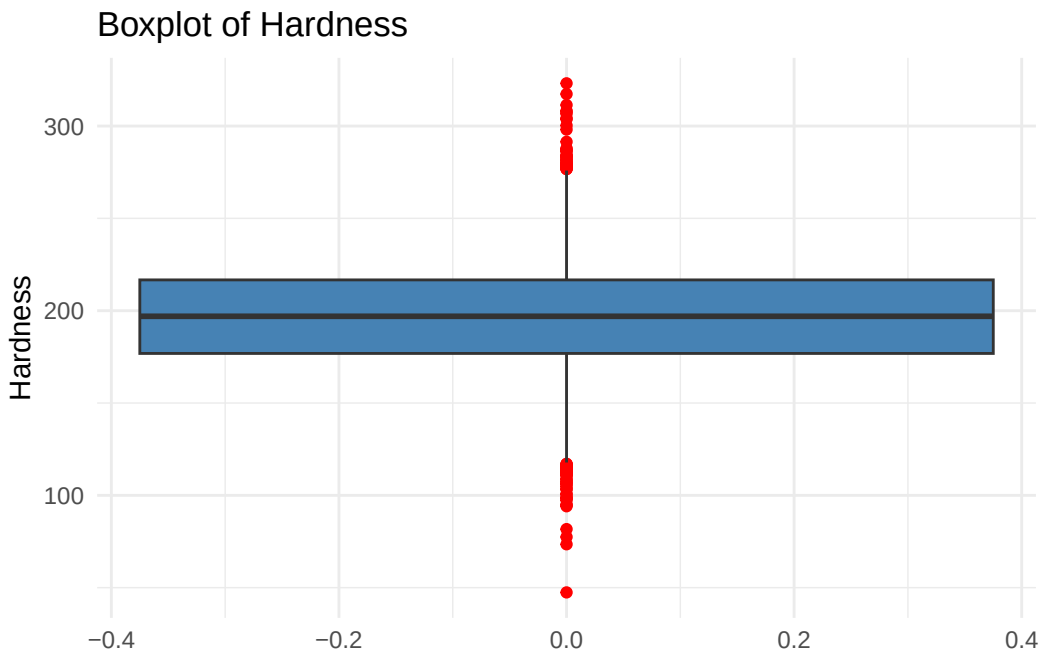


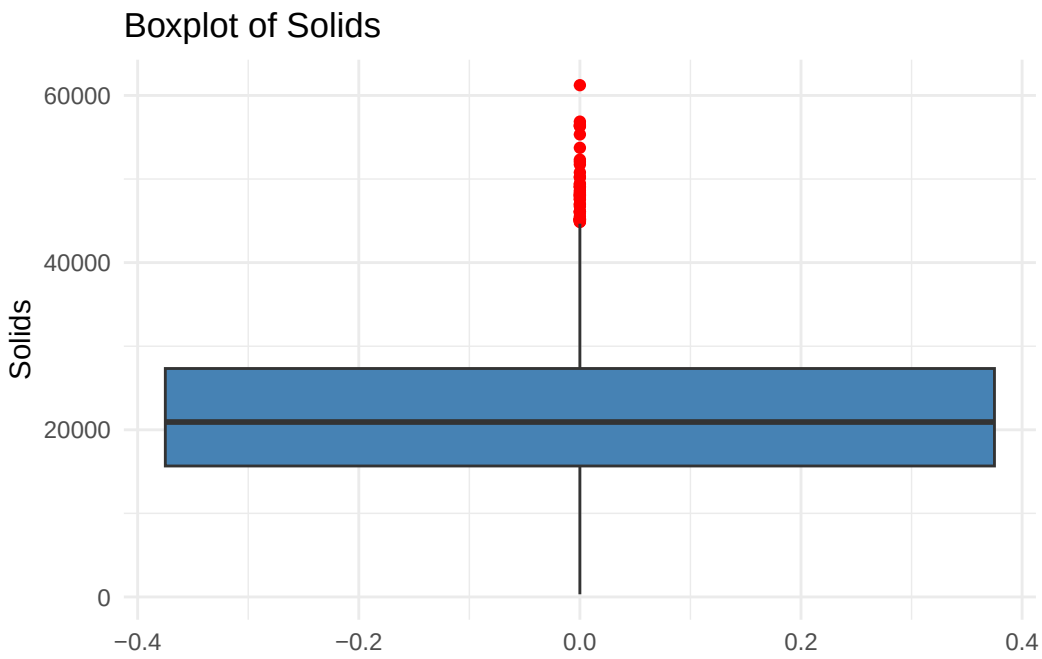
Mean, median, IQR displayed. pH has 491 missing values, sulfate has 781 missing values, trihalomethanes has 162 missing values. Chloramines and Solids have a large spread (0.352, to 13.127) and (321 to 61227) respectfully. Features are spread relatively symmetrically around their means despite there being two classes.

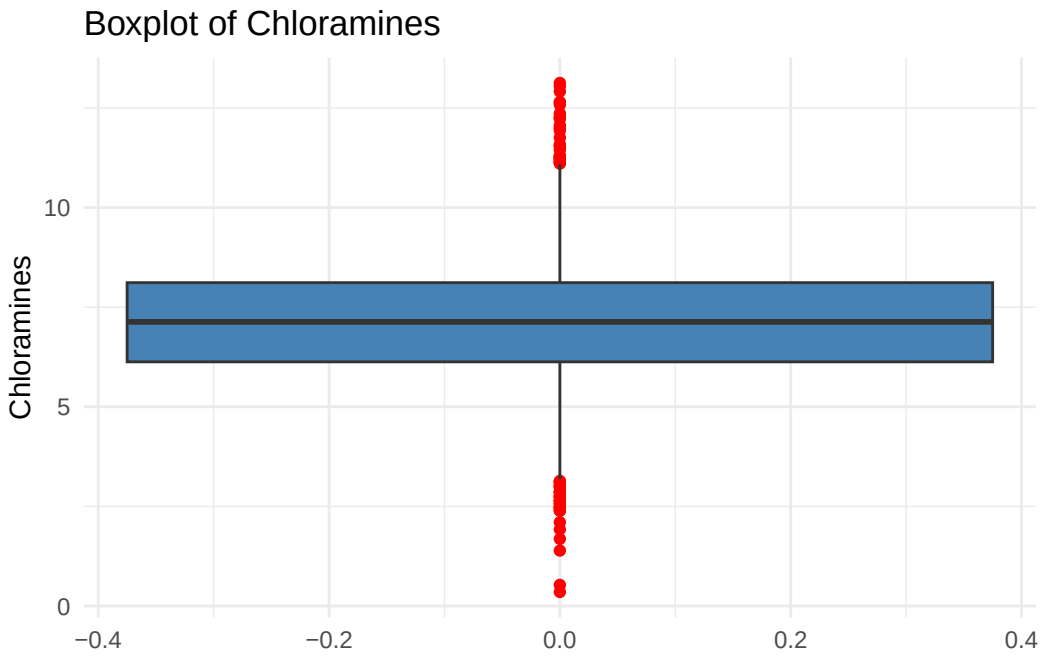
```
#Boxplots of Columns


for (col in names(fulldata)) {
  p <- ggplot(fulldata, aes(y = .data[[col]])) +
    geom_boxplot(fill = "steelblue", outlier.color = "red") +
    labs(title = paste("Boxplot of", col), y = col) +
    theme_minimal()

  print(p)  # prints each plot in the viewer
}
```

```
Warning: Removed 491 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```

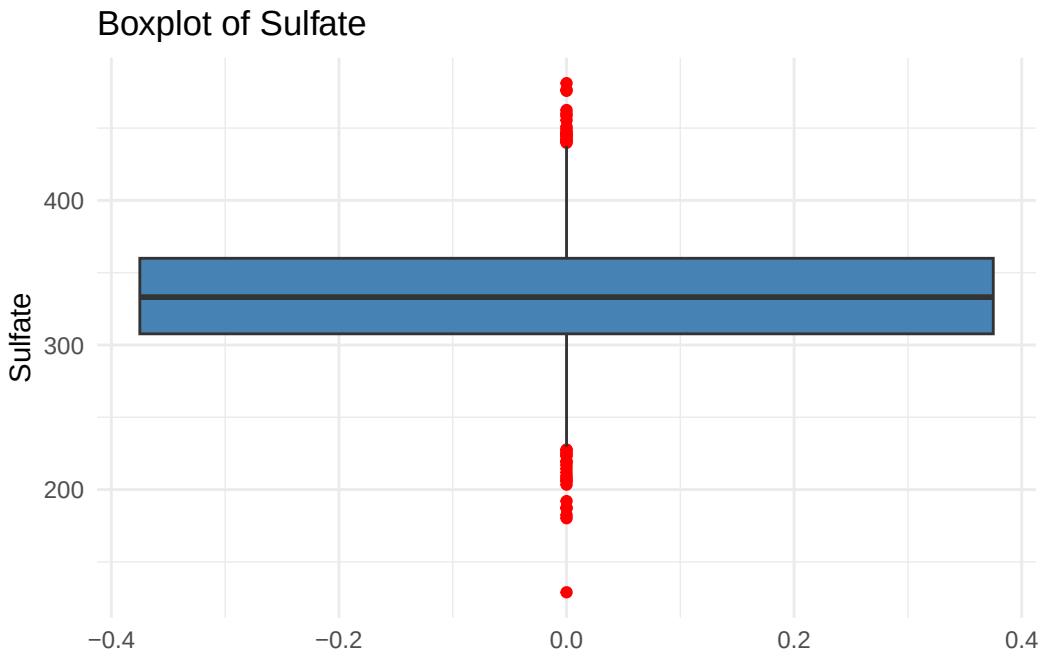


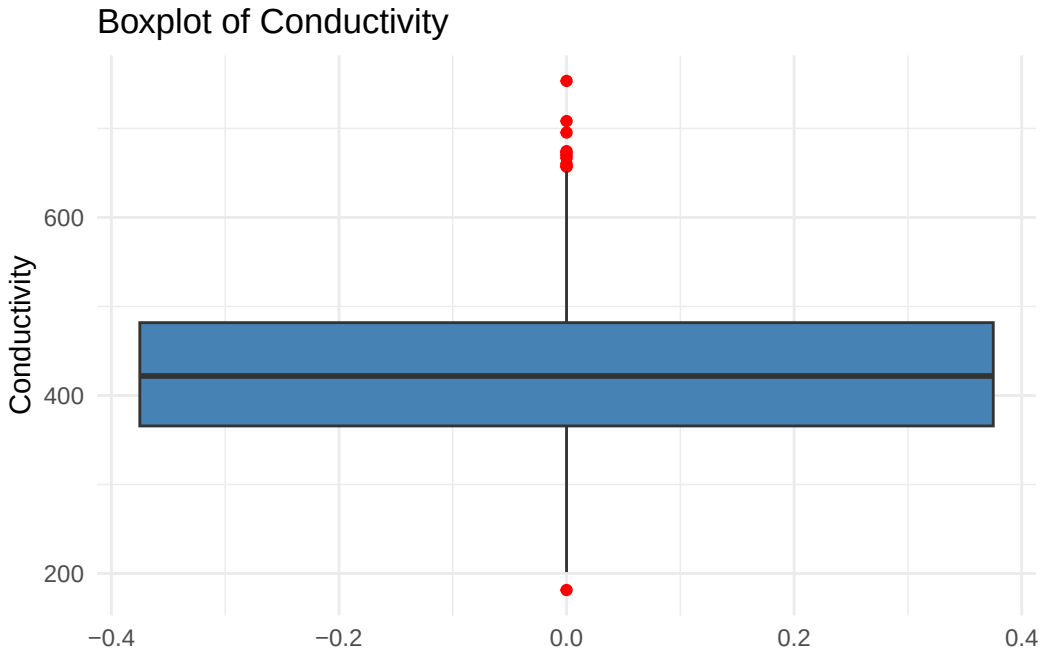


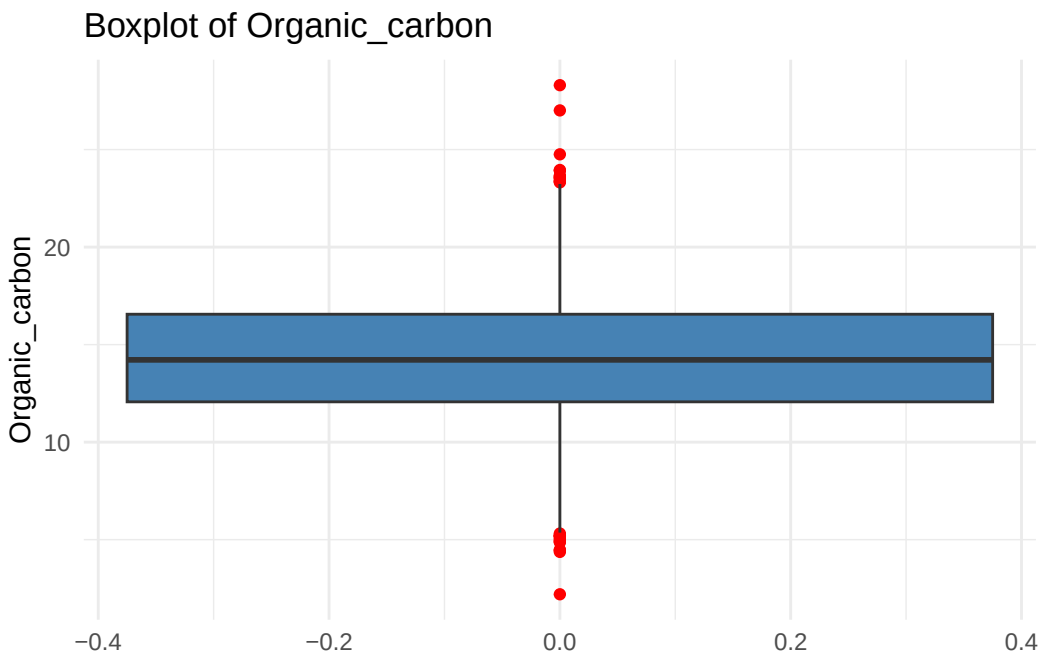




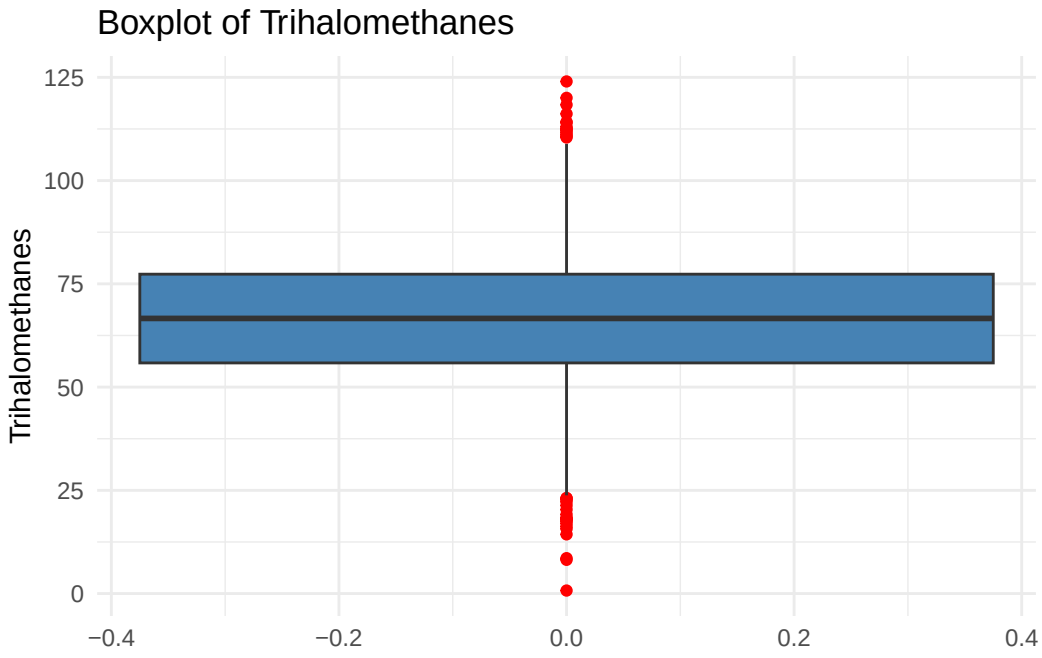

```
Warning: Removed 781 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```

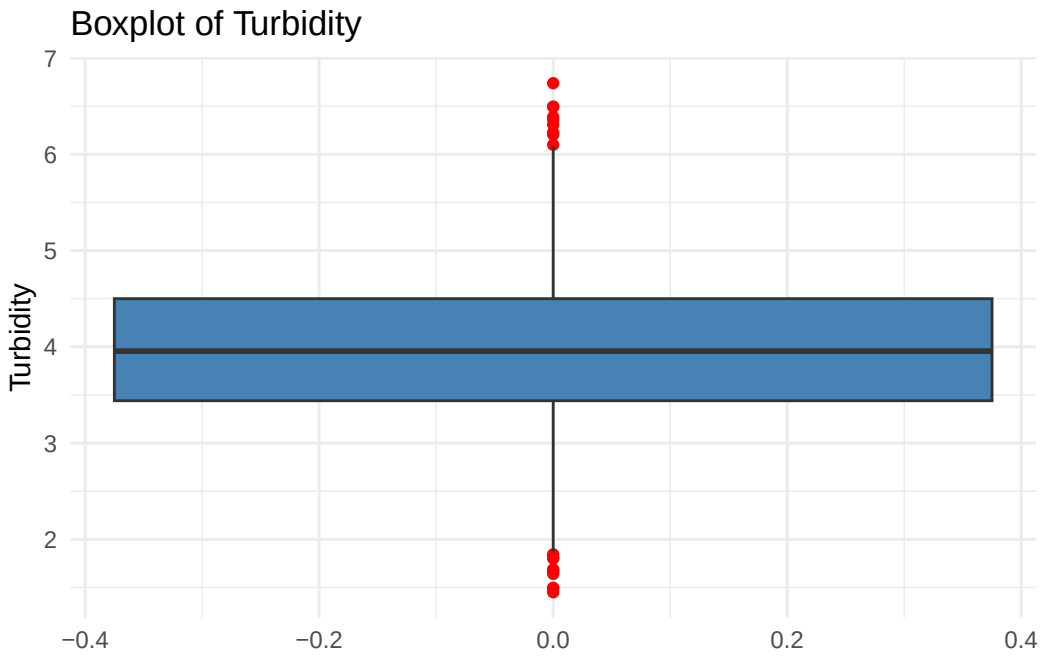

Boxplot of Sulfate
Sulfate
400
300
200
−0.4
−0.2
0.0
0.2
0.4

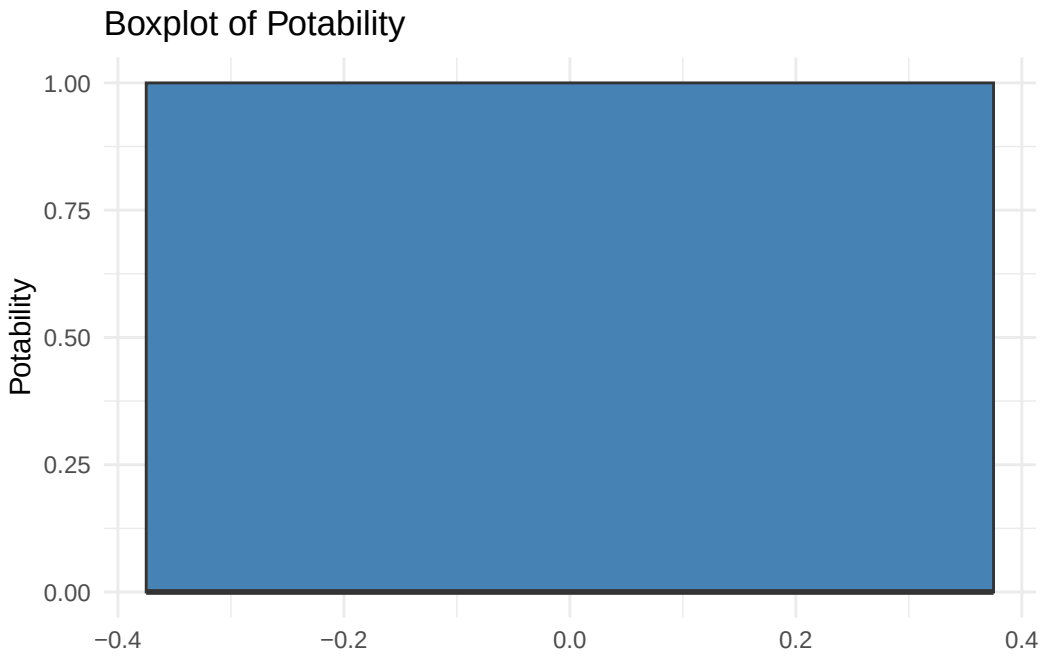

Boxplot of Conductivity
Conductivity
600
400
200
−0.4
−0.2
0.0
0.2
0.4



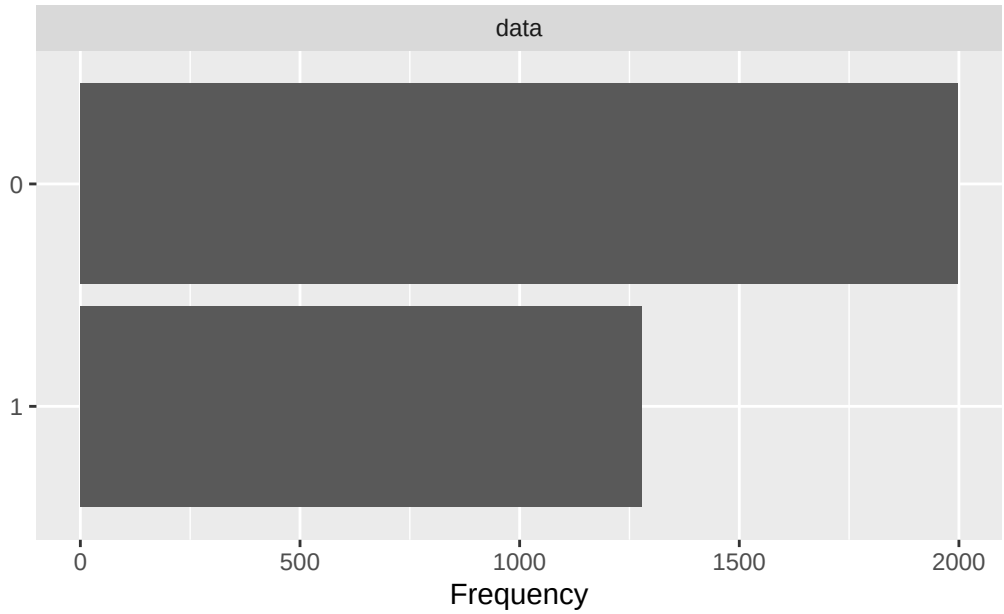

```
Warning: Removed 162 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```








```
#compare the counts of potable vs non-potable
plot_bar(fulldata$Potability)
```



pH has samples at their physically possible max and min of 0 and 14.

This is a slightly imbalanced dataset with around 40% of the water sampled is potable and 60% being non-potable.

Outliers

```
set.seed(42)

#impute the original dataset with predicitve mean matching
X <- fulldata[sapply(fulldata, is.numeric)]
imputed <- mice(X, m = 1, method = "pmm", maxit = 5)
```

```
 iter imp variable
  1   1  ph  Sulfate  Trihalomethanes
  2   1  ph  Sulfate  Trihalomethanes
  3   1  ph  Sulfate  Trihalomethanes
  4   1  ph  Sulfate  Trihalomethanes
  5   1  ph  Sulfate  Trihalomethanes
```

```r
X <- complete(imputed)

#data scaled
X_scaled <- scale(X)
#back to data frame
X <- data.frame(X_scaled)
X$Potability = fulldata$Potability
#remove outliers
md <-  mahalanobis(X_scaled, colMeans(X_scaled), cov(X_scaled))
outliers <- which(md > qchisq(0.975, df=ncol(X_scaled)))
outliers
```

```
  [1]   52   67   68   89  119  200  205  228  254  264  273  276  279  284  286
 [16]  288  291  314  318  331  334  343  346  347  348  352  355  358  364  366
 [31]  367  375  380  384  386  406  409  493  510  532  667  693  699  727  783
 [46]  786  787  811 1026 1032 1069 1076 1078 1097 1102 1107 1124 1156 1187 1241
 [61] 1303 1344 1362 1367 1488 1491 1503 1536 1537 1538 1543 1555 1606 1624 1643
 [76] 1650 1670 1744 1747 1766 1767 1768 1773 1774 1785 1793 1799 1816 1837 1859
 [91] 1869 1893 1909 2013 2052 2061 2076 2077 2083 2097 2099 2213 2231 2237 2250
[106] 2291 2301 2303 2319 2337 2344 2351 2371 2429 2447 2479 2555 2587 2603 2631
[121] 2632 2646 2681 2682 2695 2700 2705 2718 2743 2797 2812 2896 2929 3015 3143
[136] 3151 3163 3191 3198 3219 3222 3237 3270
```

```r
length(outliers)
```

```
[1] 143
```

```r
#data without outliers, and is scaled and imputed.

clean_data <- X[-outliers, ]
clean_data <- na.omit(clean_data)
clean_data$Potability <- factor(clean_data$Potability, levels = c(0, 1))
clean_data$Potability <- relevel(clean_data$Potability, ref = "1")
```

created "clean_data". It was imputed using multiple imputation by chained equations with method predictive mean matching. This data was then scaled and outliers were removed using the Mahalanobis distances, which is usually used for more correlated data. Using a Chi-square of 97.5% there were ~140 outliers out of the 3276 observations. So "clean_data" has outliers removed and has been imputed.

Train/Test Split

```
set.seed(42)
#full dataset split 70%, 30% training and testing
train_index <- sample(seq_len(nrow(fulldata)), size = 0.7 * nrow(fulldata))
train_data <- fulldata[train_index, ]
test_data  <- fulldata[-train_index, ]

#same indexes were used to create the clean test data set
clean_train_data <- clean_data[train_index, ]
clean_test_data <- clean_data[-train_index, ]

#some index were mismatched outliers removed.
clean_train_data <- na.omit(clean_train_data)
clean_test_data <- na.omit(clean_test_data)



#convert target class as a factor which is required for classification in certain models
fulldata$Potability <- factor(fulldata$Potability, levels = c(0, 1))
train_data$Potability <- factor(train_data$Potability, levels = c(0, 1))
test_data$Potability <- factor(test_data$Potability, levels = c(0, 1))
clean_data$Potability <- factor(clean_data$Potability, levels = c(0, 1))
clean_train_data$Potability <- factor(clean_train_data$Potability, levels = c(0, 1))

clean_test_data$Potability <- factor(clean_test_data$Potability, levels = c(0, 1))


#making 1  the positive class
train_data$Potability <- relevel(train_data$Potability, ref = "1")
test_data$Potability <- relevel(test_data$Potability, ref = "1")


#making 1  the positive class
clean_data$Potability <- relevel(clean_data$Potability, ref = "1")
clean_train_data$Potability <- relevel(clean_train_data$Potability, ref = "1")
clean_test_data$Potability <- relevel(clean_test_data$Potability, ref = "1")

#checking to see if clean_train_data could potentially be used with test_data
same_rows <- intersect(clean_train_data, test_data)
n_same <- nrow(same_rows)
n_same
```

```
[1] 0
```

Two sets of data frames: one that was previously imputed and outliers removed: clean_train_data and clean_test_data.

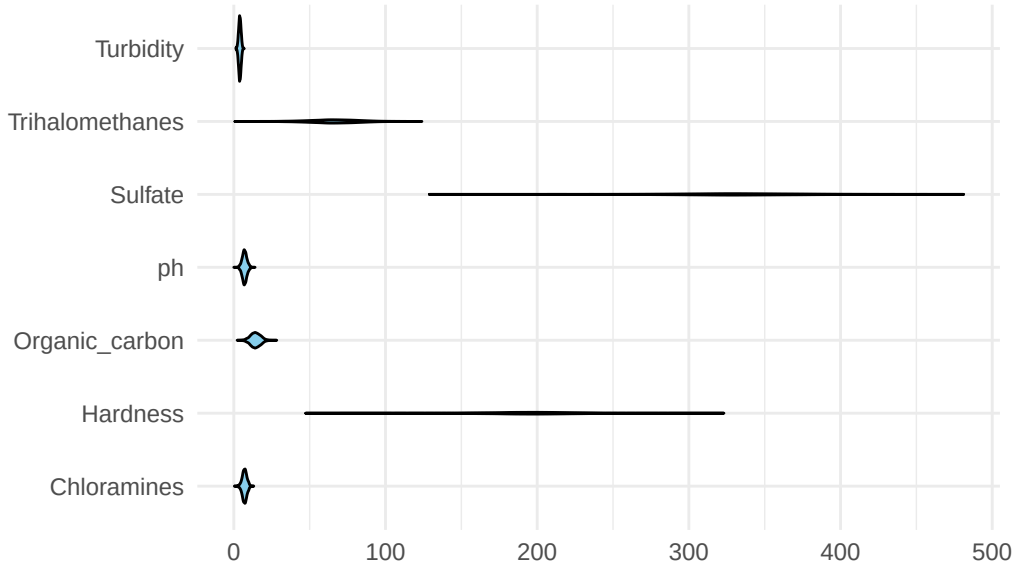
The other set of data frames: train_data, test_data, were split before any pre-processing.

```
#testing removal of features
#removal of features
#clean_test_data <- clean_test_data |> dplyr::select(-Trihalomethanes, -Conductivity, - Turbi

#clean_train_data <- clean_train_data |> dplyr::select(-Trihalomethanes, -Conductivity, - Tur
```

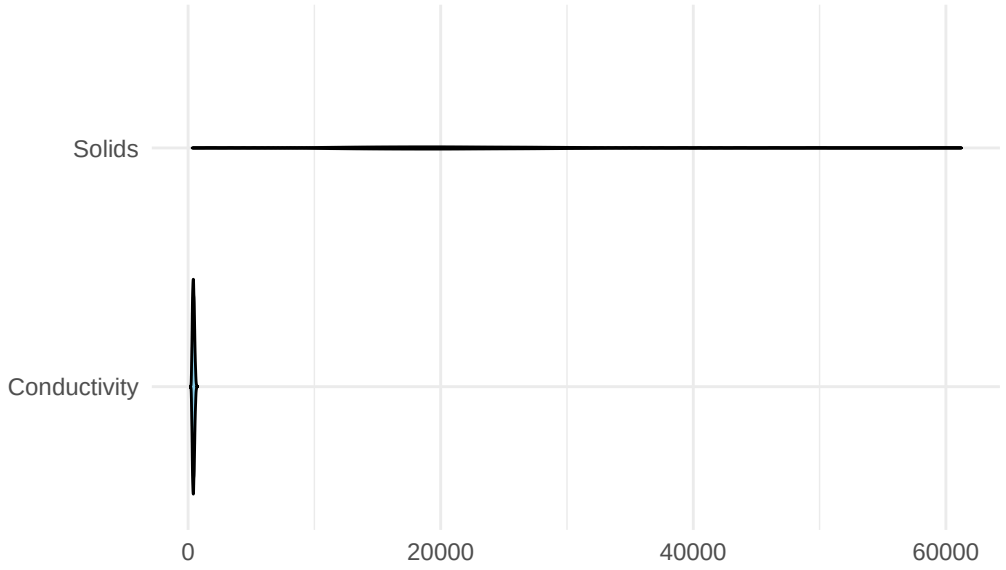
Violin plot visualization

```
#violin plots of the original data
numeric_long <- fulldata  |> dplyr::select(-Potability, -Solids, -Conductivity) |>
  pivot_longer(cols = everything(), names_to = "Variable", values_to = "Value")

ggplot(numeric_long, aes(x = Variable, y = Value)) +
geom_violin(fill = "skyblue", color = "black") +
  #scale_y_continuous(limits = c(0, 500)) +
labs(title = "",
x = "",
y = "") +
theme_minimal()+
  coord_flip()
```

```
Warning: Removed 1434 rows containing non-finite outside the scale range
(`stat_ydensity()`).
```



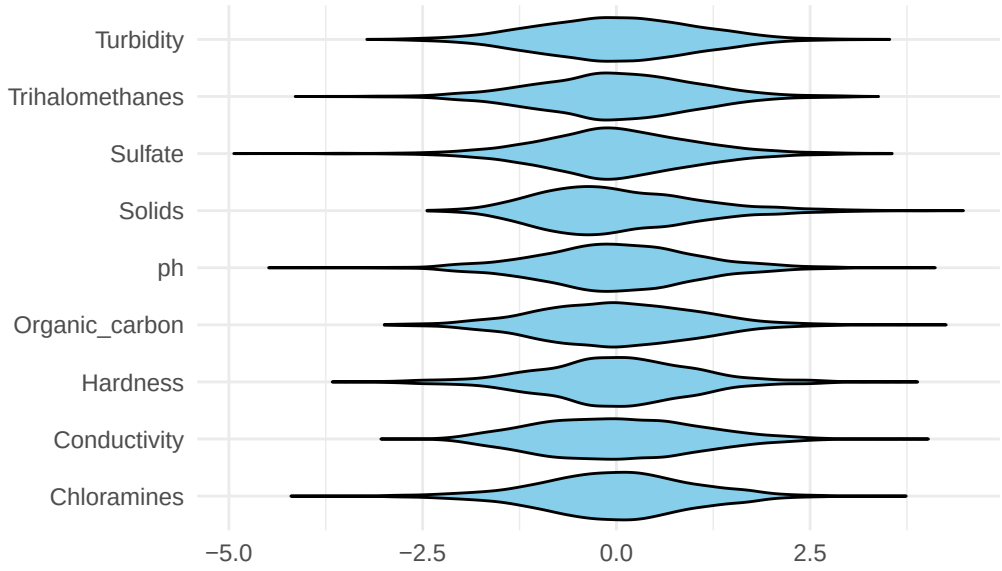

```
numeric_long <- fulldata  |> dplyr::select( Solids, Conductivity) |>
  pivot_longer(cols = everything(), names_to = "Variable", values_to = "Value")

ggplot(numeric_long, aes(x = Variable, y = Value)) +
geom_violin(fill = "skyblue", color = "black") +
  #scale_y_continuous(limits = c(0, 500)) +
labs(title = "",
x = "",
y = "") +
theme_minimal()+
  coord_flip()
```




```
#Potability should not be scaled
num_cols <- sapply(train_data, is.numeric)
num_cols["Potability"] <- FALSE

#scale all the columns except potability
train_scaled_vals <- scale(train_data[, num_cols])

#apply the scale from the training data to the test data
train_center <- attr(train_scaled_vals, "scaled:center") #pull out the center of the scaled c
train_scale <- attr(train_scaled_vals, "scaled:scale") #apply the scaling factor

#scale the test data using the scaling from the train data
test_scaled_vals <- scale(test_data[, num_cols], center = train_center, scale = train_scale)

#add the potability column back to training data and scale data
train_scaled <- data.frame(train_scaled_vals, Potability = train_data$Potability)

test_scaled <- data.frame(test_scaled_vals, Potability = test_data$Potability)

#checking the distributions to see that they are on the same scale.

numeric_scaled_long <- train_scaled  |> dplyr::select(-Potability) |>
  pivot_longer(cols = everything(), names_to = "Variable", values_to = "Value")
```

```
#distributions on the same scale
ggplot(numeric_scaled_long, aes(x = Variable, y = Value)) +
geom_violin(fill = "skyblue", color = "black") +
  #scale_y_continuous(limits = c(0, 500)) +
labs(title = "",
x = "",
y = "") +
theme_minimal()+
  coord_flip()
```

```
Warning: Removed 1016 rows containing non-finite outside the scale range
(`stat_ydensity()`).
```



From the scaled data, it becomes clearer to see that pH. sulfate trihalomethanes are concentrated around their mean. Solids is right skewed. The features are uni-modal.

```
# number of rows with missing values
sum(rowSums(is.na(fulldata)) > 0)
```

```
[1] 1265
```

Around one third of the dataset has rows with missing values. To avoid large data loss, the rows were kept and the "not clean" training and test data set with imputed.

Imputation

Medians from training data set used to impute the training (dirty) and test data set

```
#selecting the features that are not a factor (potability)
num_cols <- names(train_data)[
  sapply(train_data, is.numeric)
]
#determine the median of each column and imput the column
datamedians <- train_data |> select(-Potability) |> sapply( function(x) {
  if (is.numeric(x)) median(x, na.rm = TRUE)
})

for (num_cols in names(datamedians)) {
  train_data[[num_cols]][is.na(train_data[[num_cols]])] <- datamedians[num_cols]
  test_data[[num_cols]][is.na(test_data[[num_cols]])]  <- datamedians[num_cols]
}

#check if there are any more missing values
colSums(is.na(train_data))
```

```
             ph        Hardness          Solids     Chloramines         Sulfate
              0               0               0               0               0
   Conductivity  Organic_carbon Trihalomethanes       Turbidity      Potability
              0               0               0               0               0
```

```
colSums(is.na(test_data))
```

```
             ph        Hardness          Solids     Chloramines         Sulfate
              0               0               0               0               0
   Conductivity  Organic_carbon Trihalomethanes       Turbidity      Potability
              0               0               0               0               0
```

```
#insure that the potability column remains a factor.  with 1 being the positive level
fulldata$Potability <- factor(fulldata$Potability, levels = c(0, 1))
train_data$Potability <- factor(train_data$Potability, levels = c(0, 1))
test_data$Potability <- factor(test_data$Potability, levels = c(0, 1))
```

```
#making 1  the positive class
train_data$Potability <- relevel(train_data$Potability, ref = "1")
test_data$Potability <- relevel(test_data$Potability, ref = "1")
```

columns successfully imputed.

Visualizing the full data set

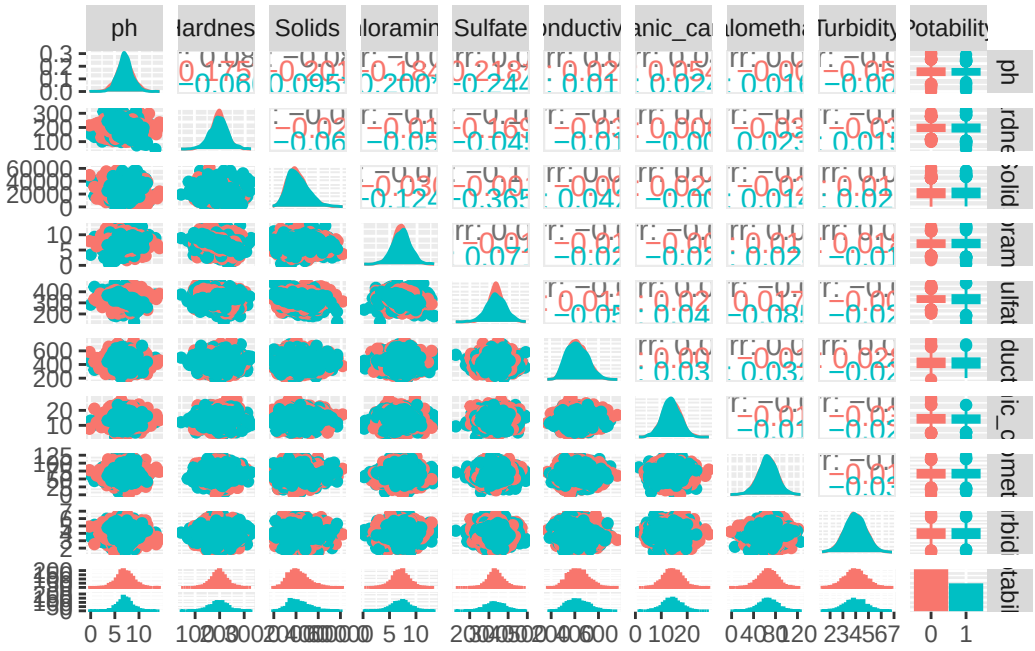
```
datapair <- fulldata |> ggpairs(
  aes(color = Potability, fill = Potability)
)

#visulize the data spread of the features of the classes
datapair
```

```
Warning: Removed 491 rows containing non-finite outside the scale range
(`stat_density()`).

Warning: Removed 491 rows containing missing values
Removed 491 rows containing missing values
Removed 491 rows containing missing values

Warning: Removed 1160 rows containing missing values

Warning: Removed 491 rows containing missing values
Removed 491 rows containing missing values

Warning: Removed 627 rows containing missing values

Warning: Removed 491 rows containing missing values

Warning: Removed 491 rows containing non-finite outside the scale range
(`stat_boxplot()`).

Warning: Removed 491 rows containing missing values or values outside the scale range
(`geom_point()`).
```

```
Warning: Removed 781 rows containing missing values

Warning: Removed 162 rows containing missing values

Warning: Removed 491 rows containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 781 rows containing missing values

Warning: Removed 162 rows containing missing values

Warning: Removed 491 rows containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 781 rows containing missing values

Warning: Removed 162 rows containing missing values

Warning: Removed 1160 rows containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 781 rows containing missing values or values outside the scale range
(`geom_point()`).
Removed 781 rows containing missing values or values outside the scale range
(`geom_point()`).
Removed 781 rows containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 781 rows containing non-finite outside the scale range
(`stat_density()`).

Warning: Removed 781 rows containing missing values
Removed 781 rows containing missing values

Warning: Removed 903 rows containing missing values

Warning: Removed 781 rows containing missing values
```

```
Warning: Removed 781 rows containing non-finite outside the scale range
(`stat_boxplot()`).

Warning: Removed 491 rows containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 781 rows containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 162 rows containing missing values

Warning: Removed 491 rows containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 781 rows containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 162 rows containing missing values

Warning: Removed 627 rows containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 162 rows containing missing values or values outside the scale range
(`geom_point()`).
Removed 162 rows containing missing values or values outside the scale range
(`geom_point()`).
Removed 162 rows containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 903 rows containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 162 rows containing missing values or values outside the scale range
(`geom_point()`).
Removed 162 rows containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 162 rows containing non-finite outside the scale range
(`stat_density()`).
```

```
Warning: Removed 162 rows containing missing values

Warning: Removed 162 rows containing non-finite outside the scale range
(`stat_boxplot()`).

Warning: Removed 491 rows containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 781 rows containing missing values or values outside the scale range
(`geom_point()`).

Warning: Removed 162 rows containing missing values or values outside the scale range
(`geom_point()`).

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

Warning: Removed 491 rows containing non-finite outside the scale range
(`stat_bin()`).

`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`.

Warning: Removed 781 rows containing non-finite outside the scale range
(`stat_bin()`).

`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`.

Warning: Removed 162 rows containing non-finite outside the scale range
(`stat_bin()`).

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




```
#describe(fulldata)
skim(fulldata)
```

Table 1: Data summary

Name	fulldata
Number of rows	3276
Number of columns	10
_____	
Column type frequency:	
factor	1
numeric	9
_____	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Potability	0	1	FALSE	2	0: 1998, 1: 1278

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
ph	491	0.85	7.08	1.59	0.00	6.09	7.04	8.06	14.00	
Hardness	0	1.00	196.37	32.88	47.43	176.85	196.97	216.67	323.12	
Solids	0	1.00	22014.09	8768.57	320.94	15666.69	20927.83	27332.76	61227.20	
Chloramines	0	1.00	7.12	1.58	0.35	6.13	7.13	8.11	13.13	
Sulfate	781	0.76	333.78	41.42	129.00	307.70	333.07	359.95	481.03	
Conductivity	0	1.00	426.21	80.82	181.48	365.73	421.88	481.79	753.34	
Organic_carbon	0	1.00	14.28	3.31	2.20	12.07	14.22	16.56	28.30	
Trihalomethanes	162	0.95	66.40	16.18	0.74	55.84	66.62	77.34	124.00	
Turbidity	0	1.00	3.97	0.78	1.45	3.44	3.96	4.50	6.74	

```
ggpairs was used to visualize the distribution of the features of both classes.  It becomes c
```

Train and evaluate the Models

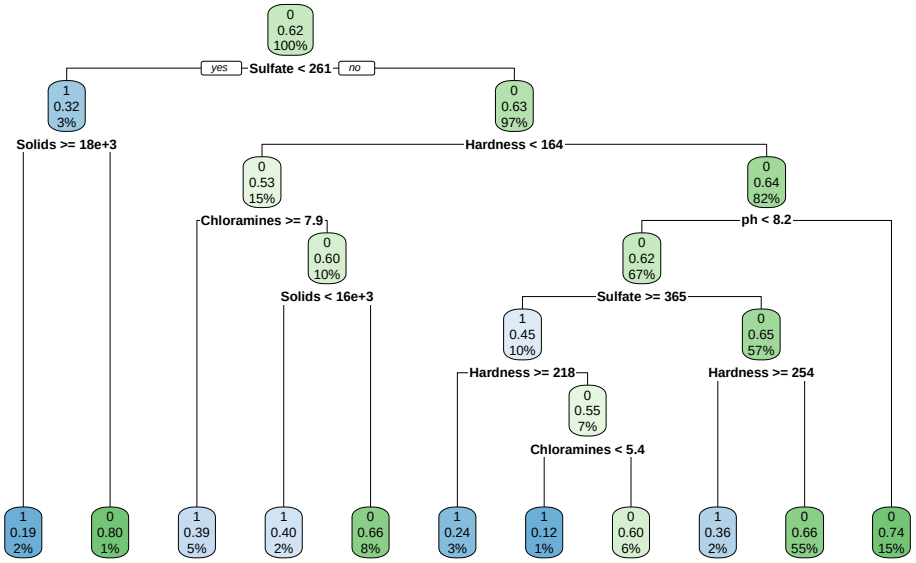
Several models were tested before settling on a few that may work. Summary of models at end.

Decision Trees

```r
set.seed(42)
#decision tree model

tree_model <- rpart(
  Potability ~ .,
  data = train_data, #not scaled
  method = "class",
  control = rpart.control(
    cp = 0.01,
    minsplit = 30,
    maxdepth = 20
  )
)

tree_model
```

```
n= 2293

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 2293 878 0 (0.3829045 0.6170955)
    2) Sulfate< 261.2405 68  22 1 (0.6764706 0.3235294)
      4) Solids>=18346.62 53  10 1 (0.8113208 0.1886792) *
      5) Solids< 18346.62 15   3 0 (0.2000000 0.8000000) *
    3) Sulfate>=261.2405 2225 832 0 (0.3739326 0.6260674)
      6) Hardness< 164.3612 339 160 0 (0.4719764 0.5280236)
       12) Chloramines>=7.911695 117  46 1 (0.6068376 0.3931624) *
       13) Chloramines< 7.911695 222  89 0 (0.4009009 0.5990991)
         26) Solids< 16280.58 50  20 1 (0.6000000 0.4000000) *
         27) Solids>=16280.58 172  59 0 (0.3430233 0.6569767) *
      7) Hardness>=164.3612 1886 672 0 (0.3563097 0.6436903)
       14) ph< 8.206025 1538 583 0 (0.3790637 0.6209363)
         28) Sulfate>=365.4012 222 100 1 (0.5495495 0.4504505)
           56) Hardness>=217.7926 70  17 1 (0.7571429 0.2428571) *
           57) Hardness< 217.7926 152  69 0 (0.4539474 0.5460526)
            114) Chloramines< 5.44537 17   2 1 (0.8823529 0.1176471) *
            115) Chloramines>=5.44537 135  54 0 (0.4000000 0.6000000) *
         29) Sulfate< 365.4012 1316 461 0 (0.3503040 0.6496960)
           58) Hardness>=254.399 44  16 1 (0.6363636 0.3636364) *
           59) Hardness< 254.399 1272 433 0 (0.3404088 0.6595912) *
       15) ph>=8.206025 348  89 0 (0.2557471 0.7442529) *
```

```
rpart.plot(tree_model)
```




```
pred_class <- predict(tree_model, test_data, type = "class")
pred_prob <- predict(tree_model, test_data, type = "prob")



#table(Predicted = pred_class, Actual = eqtest_data$term_deposit_factor)

#mean(pred_class == eqtest_data$term_deposit_factor)




#confusion matrix contains most metrics for classifiers
cm <- confusionMatrix(pred_class, test_data$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
```

```
Prediction   1   0
         1  86  80
         0 314 503

               Accuracy : 0.5992
                 95% CI : (0.5678, 0.63)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 0.3612

                  Kappa : 0.0856

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

            Sensitivity : 0.21500
            Specificity : 0.86278
         Pos Pred Value : 0.51807
         Neg Pred Value : 0.61567
             Prevalence : 0.40692
         Detection Rate : 0.08749
   Detection Prevalence : 0.16887
      Balanced Accuracy : 0.53889

       'Positive' Class : 1
```

```r
# calculate the F1 Score which is more useful than accuracy due to the focus on precision and
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1 <- 2 * (precision * recall) / (precision + recall)
f1
```

```
Pos Pred Value
     0.3038869
```

Deeper Decision Tree

cp = 0.001 overfit

```r
set.seed(42)
tree_model <- rpart(
  Potability ~ .,
  data = train_data,
  method = "class",
  control = rpart.control(
    cp = 0.001,
    minsplit = 30,
    maxdepth = 20
  )
)

tree_model
```

```
n= 2293

node), split, n, loss, yval, (yprob)
      * denotes terminal node

     1) root 2293 878 0 (0.38290449 0.61709551)
       2) Sulfate< 261.2405 68  22 1 (0.67647059 0.32352941)
         4) Solids>=18346.62 53  10 1 (0.81132075 0.18867925) *
         5) Solids< 18346.62 15   3 0 (0.20000000 0.80000000) *
       3) Sulfate>=261.2405 2225 832 0 (0.37393258 0.62606742)
         6) Hardness< 164.3612 339 160 0 (0.47197640 0.52802360)
          12) Chloramines>=7.911695 117  46 1 (0.60683761 0.39316239)
            24) ph>=6.257209 77  23 1 (0.70129870 0.29870130)
              48) Sulfate< 329.3489 35   4 1 (0.88571429 0.11428571) *
              49) Sulfate>=329.3489 42  19 1 (0.54761905 0.45238095)
                98) Turbidity< 3.439771 11   2 1 (0.81818182 0.18181818) *
                99) Turbidity>=3.439771 31  14 0 (0.45161290 0.54838710)
                 198) ph< 7.121211 12   4 1 (0.66666667 0.33333333) *
                 199) ph>=7.121211 19   6 0 (0.31578947 0.68421053) *
            25) ph< 6.257209 40  17 0 (0.42500000 0.57500000)
              50) Sulfate>=330.6019 25   9 1 (0.64000000 0.36000000) *
              51) Sulfate< 330.6019 15   1 0 (0.06666667 0.93333333) *
          13) Chloramines< 7.911695 222  89 0 (0.40090090 0.59909910)
            26) Solids< 16280.58 50  20 1 (0.60000000 0.40000000)
              52) Solids>=13665.68 23   4 1 (0.82608696 0.17391304) *
              53) Solids< 13665.68 27  11 0 (0.40740741 0.59259259) *
            27) Solids>=16280.58 172  59 0 (0.34302326 0.65697674)
              54) Solids>=19579.79 138  55 0 (0.39855072 0.60144928)
```

```
      108) Sulfate< 376.8662 111  51 0 (0.45945946 0.54054054)
        216) Hardness>=150.5074 63  27 1 (0.57142857 0.42857143)
          432) ph>=5.50641 51  18 1 (0.64705882 0.35294118) *
          433) ph< 5.50641 12   3 0 (0.25000000 0.75000000) *
        217) Hardness< 150.5074 48  15 0 (0.31250000 0.68750000) *
      109) Sulfate>=376.8662 27   4 0 (0.14814815 0.85185185) *
     55) Solids< 19579.79 34   4 0 (0.11764706 0.88235294) *
7) Hardness>=164.3612 1886 672 0 (0.35630965 0.64369035)
 14) ph< 8.206025 1538 583 0 (0.37906372 0.62093628)
   28) Sulfate>=365.4012 222 100 1 (0.54954955 0.45045045)
     56) Hardness>=217.7926 70  17 1 (0.75714286 0.24285714)
      112) Solids< 25287.28 50   4 1 (0.92000000 0.08000000) *
      113) Solids>=25287.28 20   7 0 (0.35000000 0.65000000) *
     57) Hardness< 217.7926 152  69 0 (0.45394737 0.54605263)
      114) Chloramines< 5.44537 17   2 1 (0.88235294 0.11764706) *
      115) Chloramines>=5.44537 135  54 0 (0.40000000 0.60000000)
        230) Chloramines>=9.344868 11   2 1 (0.81818182 0.18181818) *
        231) Chloramines< 9.344868 124  45 0 (0.36290323 0.63709677)
          462) Organic_carbon>=11.24699 101  42 0 (0.41584158 0.58415842)
            924) Organic_carbon< 13.9326 36  14 1 (0.61111111 0.38888889)
             1848) Chloramines< 8.04623 23   6 1 (0.73913043 0.26086957) *
             1849) Chloramines>=8.04623 13   5 0 (0.38461538 0.61538462) *
            925) Organic_carbon>=13.9326 65  20 0 (0.30769231 0.69230769)
             1850) Trihalomethanes< 71.89747 41  17 0 (0.41463415 0.58536585)
               3700) Hardness< 188.8751 19   8 1 (0.57894737 0.42105263) *
               3701) Hardness>=188.8751 22   6 0 (0.27272727 0.72727273) *
             1851) Trihalomethanes>=71.89747 24   3 0 (0.12500000 0.87500000) *
          463) Organic_carbon< 11.24699 23   3 0 (0.13043478 0.86956522) *
   29) Sulfate< 365.4012 1316 461 0 (0.35030395 0.64969605)
     58) Hardness>=254.399 44  16 1 (0.63636364 0.36363636)
      116) Sulfate>=314.7838 33   8 1 (0.75757576 0.24242424) *
      117) Sulfate< 314.7838 11   3 0 (0.27272727 0.72727273) *
     59) Hardness< 254.399 1272 433 0 (0.34040881 0.65959119)
      118) ph>=4.594908 1209 426 0 (0.35235732 0.64764268)
        236) ph>=7.055863 396 161 0 (0.40656566 0.59343434)
          472) Chloramines>=6.985978 218 107 0 (0.49082569 0.50917431)
            944) Conductivity< 328.9354 27   6 1 (0.77777778 0.22222222) *
            945) Conductivity>=328.9354 191  86 0 (0.45026178 0.54973822)
             1890) Solids>=29267.16 32  10 1 (0.68750000 0.31250000) *
             1891) Solids< 29267.16 159  64 0 (0.40251572 0.59748428)
               3782) Hardness>=187.906 122  57 0 (0.46721311 0.53278689)
                 7564) Hardness< 198.2794 23   7 1 (0.69565217 0.30434783) *
                 7565) Hardness>=198.2794 99  41 0 (0.41414141 0.58585859)
```

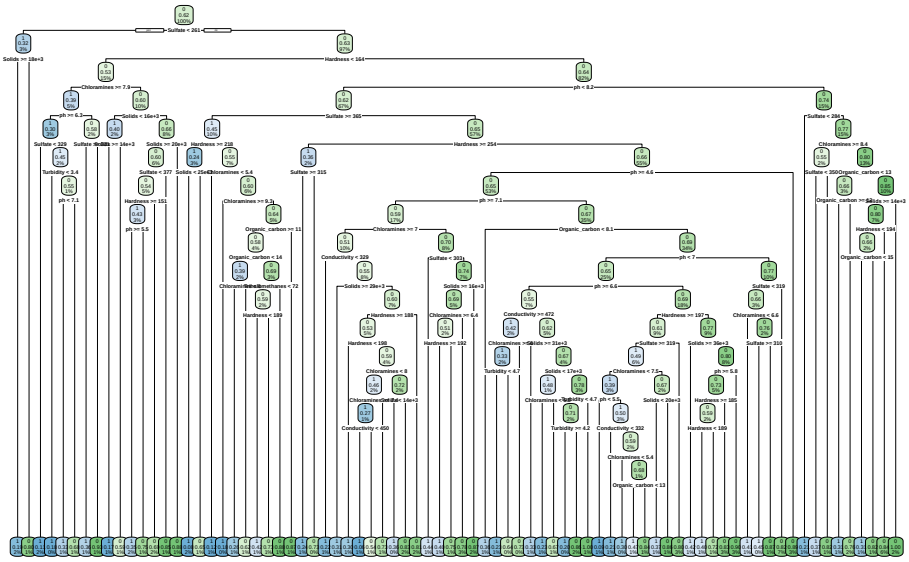
```
          15130) Chloramines< 8.038136 52  24 1 (0.53846154 0.46153846)
            30260) Chloramines>=7.391049 30   8 1 (0.73333333 0.26666667)
              60520) Conductivity< 450.4485 17   1 1 (0.94117647 0.05882353)
              60521) Conductivity>=450.4485 13   6 0 (0.46153846 0.53846154)
            30261) Chloramines< 7.391049 22   6 0 (0.27272727 0.72727273) *
          15131) Chloramines>=8.038136 47  13 0 (0.27659574 0.72340426)
            30262) Solids< 13813.33 11   4 1 (0.63636364 0.36363636) *
            30263) Solids>=13813.33 36   6 0 (0.16666667 0.83333333) *
        3783) Hardness< 187.906 37   7 0 (0.18918919 0.81081081) *
    473) Chloramines< 6.985978 178  54 0 (0.30337079 0.69662921)
      946) Sulfate< 303.4462 27  12 1 (0.55555556 0.44444444) *
      947) Sulfate>=303.4462 151  39 0 (0.25827815 0.74172185)
       1894) Solids>=16496.34 106  33 0 (0.31132075 0.68867925)
         3788) Chloramines>=6.365501 37  18 0 (0.48648649 0.51351351)
           7576) Hardness>=192.3607 25  10 1 (0.60000000 0.40000000) *
           7577) Hardness< 192.3607 12   3 0 (0.25000000 0.75000000) *
         3789) Chloramines< 6.365501 69  15 0 (0.21739130 0.78260870) *
       1895) Solids< 16496.34 45   6 0 (0.13333333 0.86666667) *
  237) ph< 7.055863 813 265 0 (0.32595326 0.67404674)
    474) Organic_carbon< 8.13401 28  10 1 (0.64285714 0.35714286) *
    475) Organic_carbon>=8.13401 785 247 0 (0.31464968 0.68535032)
      950) ph< 7.04223 563 197 0 (0.34991119 0.65008881)
       1900) ph>=6.634442 160  72 0 (0.45000000 0.55000000)
         3800) Conductivity>=472.4072 53  22 1 (0.58490566 0.41509434)
           7600) Chloramines>=6.00645 42  14 1 (0.66666667 0.33333333)
            15200) Turbidity< 4.708821 31   7 1 (0.77419355 0.22580645) *
            15201) Turbidity>=4.708821 11   4 0 (0.36363636 0.63636364) *
           7601) Chloramines< 6.00645 11   3 0 (0.27272727 0.72727273) *
         3801) Conductivity< 472.4072 107  41 0 (0.38317757 0.61682243)
           7602) Solids>=31070.29 18   6 1 (0.66666667 0.33333333) *
           7603) Solids< 31070.29 89  29 0 (0.32584270 0.67415730)
            15206) Solids< 17469.33 31  15 1 (0.51612903 0.48387097)
              30412) Chloramines< 6.847432 13   3 1 (0.76923077 0.23076923) *
              30413) Chloramines>=6.847432 18   6 0 (0.33333333 0.66666667) *
            15207) Solids>=17469.33 58  13 0 (0.22413793 0.77586207)
              30414) Turbidity< 4.733707 45  13 0 (0.28888889 0.71111111)
                60828) Turbidity>=4.239223 10   2 1 (0.80000000 0.20000000) *
                60829) Turbidity< 4.239223 35   5 0 (0.14285714 0.85714286) *
              30415) Turbidity>=4.733707 13   0 0 (0.00000000 1.00000000) *
       1901) ph< 6.634442 403 125 0 (0.31017370 0.68982630)
         3802) Hardness>=197.1672 204  80 0 (0.39215686 0.60784314)
           7604) Sulfate>=318.9145 128  63 1 (0.50781250 0.49218750)
            15208) Chloramines< 7.485071 80  31 1 (0.61250000 0.38750000)
```

```
                    30416) ph< 5.53909 22   2 1 (0.90909091 0.09090909) *
                    30417) ph>=5.53909 58  29 1 (0.50000000 0.50000000)
                      60834) Conductivity< 332.1286 14   3 1 (0.78571429 0.21428571)
                      60835) Conductivity>=332.1286 44  18 0 (0.40909091 0.59090909)
                       121670) Chloramines< 5.370559 10   3 1 (0.70000000 0.3000000
                       121671) Chloramines>=5.370559 34  11 0 (0.32352941 0.6764705
                         243342) Organic_carbon< 12.78914 15   7 1 (0.53333333 0.46
                         243343) Organic_carbon>=12.78914 19   3 0 (0.15789474 0.84
                  15209) Chloramines>=7.485071 48  16 0 (0.33333333 0.66666667)
                    30418) Solids< 19568.21 19   7 1 (0.63157895 0.36842105) *
                    30419) Solids>=19568.21 29   4 0 (0.13793103 0.86206897) *
                 7605) Sulfate< 318.9145 76  15 0 (0.19736842 0.80263158) *
               3803) Hardness< 197.1672 199  45 0 (0.22613065 0.77386935)
                 7606) Solids>=35661.62 12   5 1 (0.58333333 0.41666667) *
                 7607) Solids< 35661.62 187  38 0 (0.20320856 0.79679144)
                  15214) ph>=5.751502 115  31 0 (0.26956522 0.73043478)
                    30428) Hardness>=185.4236 49  20 0 (0.40816327 0.59183673)
                      60856) Hardness< 189.2005 20   8 1 (0.60000000 0.40000000) *
                      60857) Hardness>=189.2005 29   8 0 (0.27586207 0.72413793) *
                    30429) Hardness< 185.4236 66  11 0 (0.16666667 0.83333333) *
                  15215) ph< 5.751502 72   7 0 (0.09722222 0.90277778) *
            951) ph>=7.04223 222  50 0 (0.22522523 0.77477477)
             1902) Sulfate< 319.1 58  20 0 (0.34482759 0.65517241)
               3804) Chloramines< 6.590676 17   7 1 (0.58823529 0.41176471) *
               3805) Chloramines>=6.590676 41  10 0 (0.24390244 0.75609756)
                 7610) Sulfate>=310.1858 11   5 1 (0.54545455 0.45454545) *
                 7611) Sulfate< 310.1858 30   4 0 (0.13333333 0.86666667) *
             1903) Sulfate>=319.1 164  30 0 (0.18292683 0.81707317) *
      119) ph< 4.594908 63   7 0 (0.11111111 0.88888889) *
  15) ph>=8.206025 348  89 0 (0.25574713 0.74425287)
    30) Sulfate< 284.1368 14   3 1 (0.78571429 0.21428571) *
    31) Sulfate>=284.1368 334  78 0 (0.23353293 0.76646707)
      62) Chloramines>=8.423856 44  20 0 (0.45454545 0.54545455)
       124) Sulfate< 349.7264 27  10 1 (0.62962963 0.37037037) *
       125) Sulfate>=349.7264 17   3 0 (0.17647059 0.82352941) *
      63) Chloramines< 8.423856 290  58 0 (0.20000000 0.80000000)
       126) Organic_carbon< 12.58131 70  24 0 (0.34285714 0.65714286)
         252) Organic_carbon>=12.02412 16   5 1 (0.68750000 0.31250000) *
         253) Organic_carbon< 12.02412 54  13 0 (0.24074074 0.75925926) *
       127) Organic_carbon>=12.58131 220  34 0 (0.15454545 0.84545455)
         254) Solids>=13653.4 169  34 0 (0.20118343 0.79881657)
           508) Hardness< 194.1059 41  14 0 (0.34146341 0.65853659)
            1016) Organic_carbon< 14.52538 13   4 1 (0.69230769 0.30769231) *
```

```
      1017) Organic_carbon>=14.52538 28    5 0 (0.17857143 0.82142857) *
     509) Hardness>=194.1059 128   20 0 (0.15625000 0.84375000) *
   255) Solids< 13653.4 51    0 0 (0.00000000 1.00000000) *
```

```
rpart.plot(tree_model)
```

```
Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```
pred_class <- predict(tree_model, test_data, type = "class")
```

```
table(Predicted = pred_class, Actual = test_data$Potability)
```

```
         Actual
Predicted   1   0
        1 185 193
        0 215 390
```

```r
mean(pred_class == test_data$Potability)
```

```
[1] 0.584944
```

```r
#confusionMatrix(pred_class$class, actual, positive = "1")
cm <- confusionMatrix(pred_class, test_data$Potability)
cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 185 193
         0 215 390

               Accuracy : 0.5849
                 95% CI : (0.5534, 0.616)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 0.7099

                  Kappa : 0.1326

 Mcnemar's Test P-Value : 0.2985

            Sensitivity : 0.4625
            Specificity : 0.6690
         Pos Pred Value : 0.4894
         Neg Pred Value : 0.6446
             Prevalence : 0.4069
         Detection Rate : 0.1882
   Detection Prevalence : 0.3845
      Balanced Accuracy : 0.5657

       'Positive' Class : 1
```

```r
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]
```

```
f1_tree <- 2 * (precision * recall) / (precision + recall)
f1_tree
```

```
Pos Pred Value
     0.4755784
```

adding deeper decision tree complexity improves the performance.

```
#had issues below with data not being set up correctly repeated code to insure that the data
num_cols <- names(train_data)[
  sapply(train_data, is.numeric)
]

datamedians <- train_data |> select(-Potability) |> sapply( function(x) {
  if (is.numeric(x)) median(x, na.rm = TRUE)
})

for (num_cols in names(datamedians)) {
  train_data[[num_cols]][is.na(train_data[[num_cols]])] <- datamedians[num_cols]
  test_data[[num_cols]][is.na(test_data[[num_cols]])]  <- datamedians[num_cols]
}

colSums(is.na(train_data))
```

```
             ph        Hardness          Solids     Chloramines         Sulfate
              0               0               0               0               0
   Conductivity  Organic_carbon Trihalomethanes       Turbidity      Potability
              0               0               0               0               0
```

```
colSums(is.na(test_data))
```

```
             ph        Hardness          Solids     Chloramines         Sulfate
              0               0               0               0               0
   Conductivity  Organic_carbon Trihalomethanes       Turbidity      Potability
              0               0               0               0               0
```

```
fulldata$Potability <- factor(fulldata$Potability, levels = c(0, 1))
train_data$Potability <- factor(train_data$Potability, levels = c(0, 1))
test_data$Potability <- factor(test_data$Potability, levels = c(0, 1))
```

```r
#making 1  the positive class
train_data$Potability <- relevel(train_data$Potability, ref = "1")
test_data$Potability <- relevel(test_data$Potability, ref = "1")
```

Random forest

```r
set.seed(42)
#random forest model

rf_model <- randomForest(
  Potability ~ .,
  data = train_data,

  ntree = 500,
  importance = TRUE
)

forrest_pred_class <- predict(rf_model, test_data, type = "class")
forrest_pred_prob <- predict(rf_model, test_data, type = "prob")

rf_model
```

```
Call:
 randomForest(formula = Potability ~ ., data = train_data, ntree = 500,      importance = TRU
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 32.53%
Confusion matrix:
    1    0 class.error
1 291  587   0.6685649
0 159 1256   0.1123675
```

```r
table(Predicted = forrest_pred_class, Actual = test_data$Potability)
```

```
         Actual
```

```
Predicted   1   0
        1 151  70
        0 249 513
```

```r
mean(forrest_pred_class == test_data$Potability)
```

```
[1] 0.6754832
```

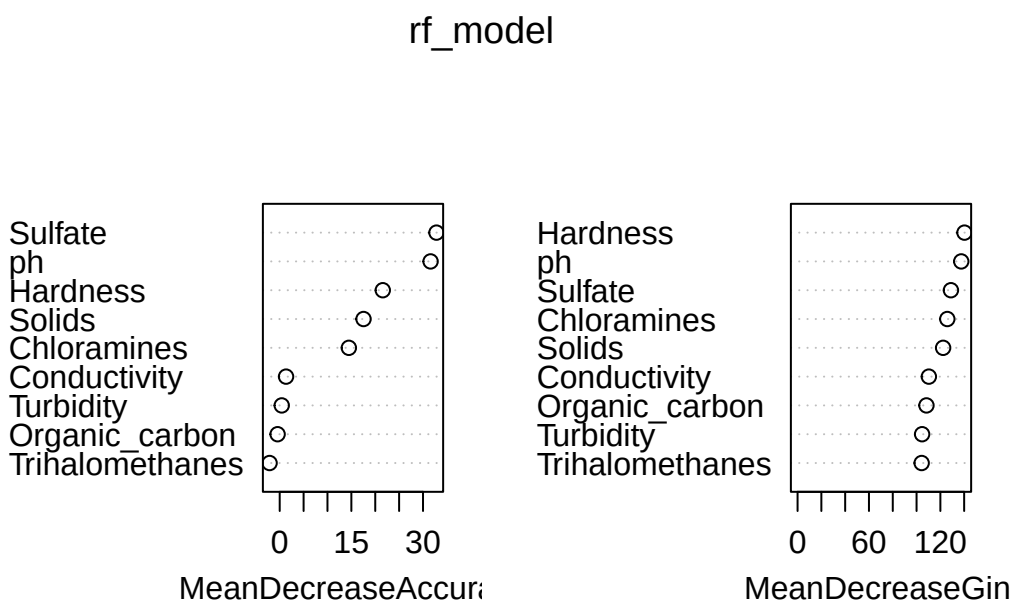
```r
cm <- confusionMatrix(forrest_pred_class, test_data$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 151  70
         0 249 513

               Accuracy : 0.6755
                 95% CI : (0.6452, 0.7047)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 5.997e-08

                  Kappa : 0.2769

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.3775
            Specificity : 0.8799
         Pos Pred Value : 0.6833
         Neg Pred Value : 0.6732
             Prevalence : 0.4069
         Detection Rate : 0.1536
   Detection Prevalence : 0.2248
      Balanced Accuracy : 0.6287

       'Positive' Class : 1
```

```r
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1_rf <- 2 * (precision * recall) / (precision + recall)
f1_rf
```

```
Pos Pred Value
     0.4863124
```

```r
varImpPlot(rf_model)
```



Ensemble methods like Random forest perform better than single decision trees.

Random Forest with cleaner data

```r
set.seed(42)
#random forest model
```

```r
rf_model <- randomForest(
  Potability ~ .,
  data = clean_train_data,

  ntree = 500,
  importance = TRUE
)

forrest_pred_class <- predict(rf_model, clean_test_data, type = "class")
forrest_pred_prob <- predict(rf_model, clean_test_data, type = "prob")

rf_model
```

```
Call:
 randomForest(formula = Potability ~ ., data = clean_train_data,      ntree = 500, importanc
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 35.75%
Confusion matrix:
    1    0 class.error
1 225  615   0.7321429
0 169 1184   0.1249076
```

```r
#table(Predicted = forrest_pred_class, Actual = test_data$Potability)

#mean(forrest_pred_class == test_data$Potability)

cm <- confusionMatrix(forrest_pred_class, clean_test_data$Potability)

cm
```

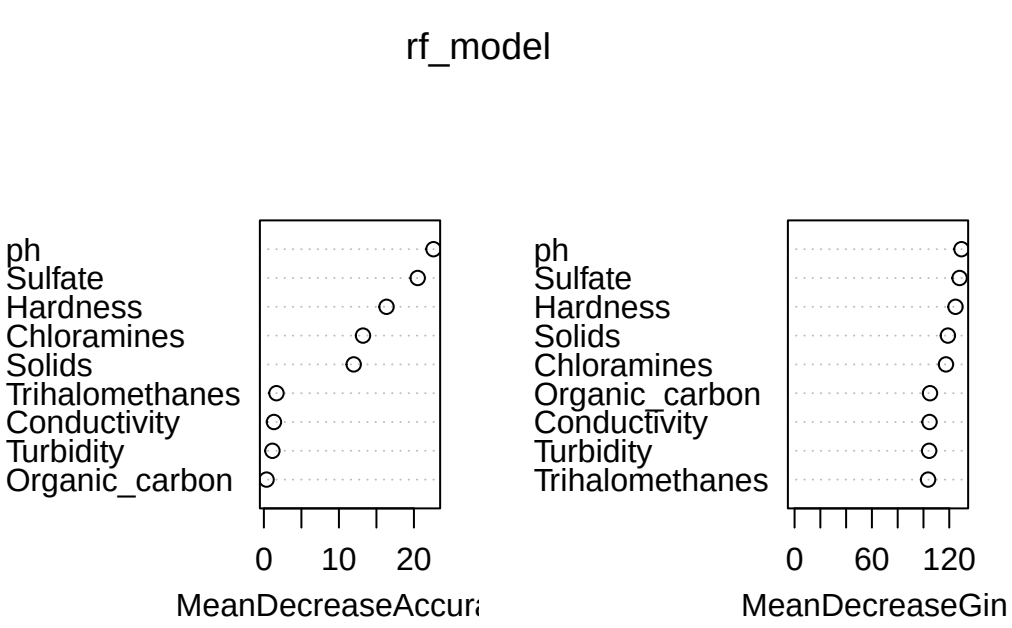
```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1  95  67
         0 263 515
```

```
               Accuracy : 0.6489
                 95% CI : (0.6175, 0.6795)
    No Information Rate : 0.6191
    P-Value [Acc > NIR] : 0.03187

                  Kappa : 0.1679

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

            Sensitivity : 0.2654
            Specificity : 0.8849
         Pos Pred Value : 0.5864
         Neg Pred Value : 0.6620
             Prevalence : 0.3809
         Detection Rate : 0.1011
   Detection Prevalence : 0.1723
      Balanced Accuracy : 0.5751

       'Positive' Class : 1
```

```r
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1_rf <- 2 * (precision * recall) / (precision + recall)
f1_rf
```

```
Pos Pred Value
     0.3653846
```

```r
varImpPlot(rf_model)
```



training with data without outliers decreases accuracy, kappa, and F1 for both ensemble me

Adaboost

```
#adaptive boosting model


#ada_partdata <-partdata |>  dplyr::select(-y, -term_deposit, -default)
#ada_partdata[] <- lapply(ada_partdata, function(x) {
#  if (is.character(x)) factor(x) else x
#})

##ada_partdata$default <- factor(ada_partdata$default, levels = c("no", "yes", "unknown"))

set.seed(42)



ada_model <- boosting(
  Potability ~ .,
```

```
  data = train_data,
  mfinal = 100,     # number of boosting iterations
  boos = TRUE
)

#ada_model

ada_pred <- predict(ada_model, test_data)

ada_pred$class <- factor(
  ada_pred$class,
  levels = levels(test_data$Potability)
)

cm <- confusionMatrix(ada_pred$class, test_data$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 169 111
         0 231 472

               Accuracy : 0.6521
                 95% CI : (0.6214, 0.6819)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 8.357e-05

                  Kappa : 0.2436

 Mcnemar's Test P-Value : 1.236e-10

            Sensitivity : 0.4225
            Specificity : 0.8096
         Pos Pred Value : 0.6036
         Neg Pred Value : 0.6714
             Prevalence : 0.4069
         Detection Rate : 0.1719
   Detection Prevalence : 0.2848
      Balanced Accuracy : 0.6161
```

```
       'Positive' Class : 1
```

```
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1_ada <- 2 * (precision * recall) / (precision + recall)
f1_ada
```

```
Pos Pred Value
     0.4970588
```

```
sort(ada_model$importance, decreasing = TRUE)
```

```
             ph         Sulfate        Hardness          Solids     Chloramines
      16.769479       14.418847       13.255398       11.705347       11.301376
   Conductivity Trihalomethanes  Organic_carbon       Turbidity
       8.890219        8.409151        7.949138        7.301046
```

Adaboost with "clean" data

```
set.seed(42)



ada_model <- boosting(
  Potability ~ .,
  data = clean_train_data,
  mfinal = 100,      # number of boosting iterations
  boos = TRUE
)

#ada_model

ada_pred <- predict(ada_model, clean_test_data)

ada_pred$class <- factor(
```

```
  ada_pred$class,
  levels = levels(clean_test_data$Potability)
)

cm <- confusionMatrix(ada_pred$class, clean_test_data$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 133 130
         0 225 452

               Accuracy : 0.6223
                 95% CI : (0.5905, 0.6534)
    No Information Rate : 0.6191
    P-Value [Acc > NIR] : 0.4343

                  Kappa : 0.1561

 Mcnemar's Test P-Value : 6.069e-07

            Sensitivity : 0.3715
            Specificity : 0.7766
         Pos Pred Value : 0.5057
         Neg Pred Value : 0.6677
             Prevalence : 0.3809
         Detection Rate : 0.1415
   Detection Prevalence : 0.2798
      Balanced Accuracy : 0.5741

       'Positive' Class : 1
```

```
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1_ada <- 2 * (precision * recall) / (precision + recall)
f1_ada
```

```
Pos Pred Value
     0.4283414
```

```r
sort(ada_model$importance, decreasing = TRUE)
```

```
             ph        Hardness     Chloramines         Sulfate          Solids
      13.589622       13.132811       12.762566       11.902679       11.610060
Trihalomethanes  Organic_carbon    Conductivity       Turbidity
      10.752570        9.341308        8.625785        8.282599
```

SVM: Radial Basis kernel

```r
set.seed(42)
#SVM model radial basis kernel



SVM_rbf_model <- ksvm(
  Potability ~ .,
  data = train_data,
  kernel = "rbfdot"
)

SVM_rbf_model
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0836255427875869

Number of Support Vectors : 1711

Objective Function Value : -1500.678
Training error : 0.280855
```

```r
rbf_pred_class <- predict(SVM_rbf_model, test_data, type = "response")



#table(Predicted = rbf_pred_class, Actual = eqtest_data$term_deposit_factor)


cm <- confusionMatrix(rbf_pred_class, test_data$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 121  34
         0 279 549

               Accuracy : 0.6816
                 95% CI : (0.6514, 0.7106)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 6.16e-09

                  Kappa : 0.2702

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.3025
            Specificity : 0.9417
         Pos Pred Value : 0.7806
         Neg Pred Value : 0.6630
             Prevalence : 0.4069
         Detection Rate : 0.1231
   Detection Prevalence : 0.1577
      Balanced Accuracy : 0.6221

       'Positive' Class : 1
```

```r
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1 <- 2 * (precision * recall) / (precision + recall)
f1
```

```
Pos Pred Value
      0.436036
```

SVM radial performance is similar to the ensemble methods adaboost and RF.

```r
num_cols <- sapply(train_data, is.numeric)
num_cols["Potability"] <- FALSE

train_scaled_vals <- scale(train_data[, num_cols])

train_center <- attr(train_scaled_vals, "scaled:center")
train_scale <- attr(train_scaled_vals, "scaled:scale")

test_scaled_vals <- scale(test_data[, num_cols], center = train_center, scale = train_scale)

train_scaled <- data.frame(train_scaled_vals, Potability = train_data$Potability)

test_scaled <- data.frame(test_scaled_vals, Potability = test_data$Potability)




set.seed(42)
#SVM model radial basis kernel




SVM_rbf_model <- ksvm(
  Potability ~ .,
  data = train_scaled,
  kernel = "rbfdot"
)
```

```
SVM_rbf_model
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0836255427875869

Number of Support Vectors : 1711

Objective Function Value : -1500.678
Training error : 0.280855
```

```
rbf_pred_class <- predict(SVM_rbf_model, test_scaled, type = "response")



#table(Predicted = rbf_pred_class, Actual = eqtest_data$term_deposit_factor)



cm <- confusionMatrix(rbf_pred_class, test_scaled$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 121  34
         0 279 549

               Accuracy : 0.6816
                 95% CI : (0.6514, 0.7106)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 6.16e-09

                  Kappa : 0.2702
```

```
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.3025
            Specificity : 0.9417
         Pos Pred Value : 0.7806
         Neg Pred Value : 0.6630
             Prevalence : 0.4069
         Detection Rate : 0.1231
   Detection Prevalence : 0.1577
      Balanced Accuracy : 0.6221

       'Positive' Class : 1
```

```r
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1 <- 2 * (precision * recall) / (precision + recall)
f1
```

```
Pos Pred Value
      0.436036
```

There was no noticable difference between using scaled data to train and non scaled data to train SVM

```r
set.seed(42)
#SVM model radial basis kernel




SVM_rbf_model <- ksvm(
  Potability ~ .,
  data = clean_train_data,
  kernel = "rbfdot"
)

SVM_rbf_model
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.078170040490051

Number of Support Vectors : 1699

Objective Function Value : -1517.98
Training error : 0.309166
```

```
rbf_pred_class <- predict(SVM_rbf_model, clean_test_data, type = "response")



#table(Predicted = rbf_pred_class, Actual = eqtest_data$term_deposit_factor)



cm <- confusionMatrix(rbf_pred_class, clean_test_data$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1  75  34
         0 283 548

               Accuracy : 0.6628
                 95% CI : (0.6315, 0.693)
    No Information Rate : 0.6191
    P-Value [Acc > NIR] : 0.003078

                  Kappa : 0.1744

 Mcnemar's Test P-Value : < 2.2e-16
```

```
            Sensitivity : 0.20950
            Specificity : 0.94158
         Pos Pred Value : 0.68807
         Neg Pred Value : 0.65945
             Prevalence : 0.38085
         Detection Rate : 0.07979
   Detection Prevalence : 0.11596
      Balanced Accuracy : 0.57554

       'Positive' Class : 1
```

```
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1 <- 2 * (precision * recall) / (precision + recall)
f1
```

```
Pos Pred Value
     0.3211991
```

performance of SVM decreased with outliers removed.

SVM: Linear kernel

```
set.seed(42)
#SVM model



SVM_lin_model <- ksvm(
  Potability ~ .,
  data = train_data,
  kernel = "vanilladot"
)
```

```
 Setting default kernel parameters
```

```
SVM_lin_model
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Linear (vanilla) kernel function.

Number of Support Vectors : 1899

Objective Function Value : -1756
Training error : 0.382904
```

```
lin_pred_class <- predict(SVM_lin_model, test_data, type = "response")



#table(Predicted = rbf_pred_class, Actual = eqtest_data$term_deposit_factor)



cm <- confusionMatrix(lin_pred_class, test_data$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1   0   0
         0 400 583

               Accuracy : 0.5931
                 95% CI : (0.5616, 0.624)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 0.5137

                  Kappa : 0
```

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

            Sensitivity : 0.0000
            Specificity : 1.0000
         Pos Pred Value :    NaN
         Neg Pred Value : 0.5931
             Prevalence : 0.4069
         Detection Rate : 0.0000
   Detection Prevalence : 0.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 1
```

```r
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1 <- 2 * (precision * recall) / (precision + recall)
f1
```

```
Pos Pred Value
           NaN
```

SVM: hyperbolic tangent sigmoid kernel

```r
set.seed(42)
#SVM model



SVM_tanh_model <- ksvm(
  Potability ~ .,
  data = train_data,
  kernel = "tanhdot"
)
```

```
 Setting default kernel parameters
```

```
SVM_tanh_model
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 1

Hyperbolic Tangent kernel function.
 Hyperparameters : scale =  1  offset =  1

Number of Support Vectors : 1315

Objective Function Value : -27507.11
Training error : 0.57174
```

```
tanh_pred_class <- predict(SVM_tanh_model, test_data, type = "response")



#table(Predicted = rbf_pred_class, Actual = eqtest_data$term_deposit_factor)



cm <- confusionMatrix(tanh_pred_class, test_data$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 105 262
         0 295 321

               Accuracy : 0.4334
                 95% CI : (0.4021, 0.465)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 1.0000
```

```
                 Kappa : -0.1894

 Mcnemar's Test P-Value : 0.1751

            Sensitivity : 0.2625
            Specificity : 0.5506
         Pos Pred Value : 0.2861
         Neg Pred Value : 0.5211
             Prevalence : 0.4069
         Detection Rate : 0.1068
   Detection Prevalence : 0.3733
      Balanced Accuracy : 0.4066

       'Positive' Class : 1
```

```r
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1 <- 2 * (precision * recall) / (precision + recall)
f1
```

```
Pos Pred Value
      0.273794
```

both sigmoid and linear kernels performed worse than radial

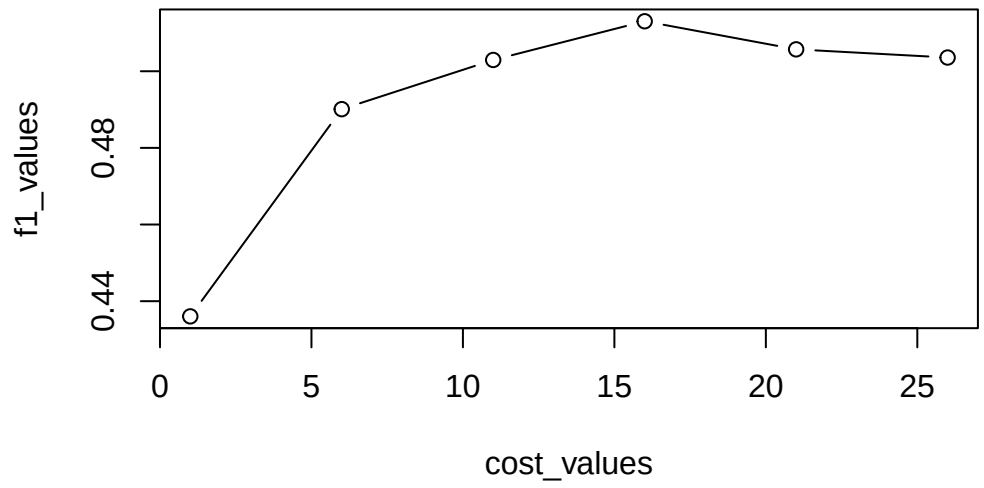
radial is the best

Cost Parameters SVM

Cost Parameter based on f1 score

```r
#loop and plot the cost values vs f1 score
cost_values <- c(seq(from=1, to  = 26, by = 5))
f1_values <- sapply(cost_values, function(x){
  SVM_rbf_model <- ksvm(
  Potability ~ .,
```

```
  data = train_data,
  kernel = "rbfdot", C = x
)
  rbf_pred <- predict(SVM_rbf_model, test_data, type = "response")
  #agree <- ifelse(rbf_pred == eqtest_data$term_deposit_factor, 1, 0)
  #accuracy <-  sum(agree) / nrow(eqtest_data)

   cm <- confusionMatrix(rbf_pred, test_data$Potability)

#cm

#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1 <- 2 * (precision * recall) / (precision + recall)

  return (f1)
})

plot(cost_values, f1_values, type = "b")
```



cost parameter of 16 had the highest F1 score. This was used as the best SVM model to compare to the other models.

```
set.seed(42)
#SVM model



SVM_rbf_model <- ksvm(
  Potability ~ .,
  data = train_data,
  kernel = "rbfdot", C = 16
)

SVM_rbf_model
```

```
Support Vector Machine object of class "ksvm"

SV type: C-svc  (classification)
 parameter : cost C = 16

Gaussian Radial Basis kernel function.
 Hyperparameter : sigma =  0.0836255427875869

Number of Support Vectors : 1568

Objective Function Value : -17313.53
Training error : 0.175752
```

```
rbf_pred_class <- predict(SVM_rbf_model, test_data, type = "response")



#table(Predicted = rbf_pred_class, Actual = eqtest_data$term_deposit_factor)



cm <- confusionMatrix(rbf_pred_class, test_data$Potability)

cm
```

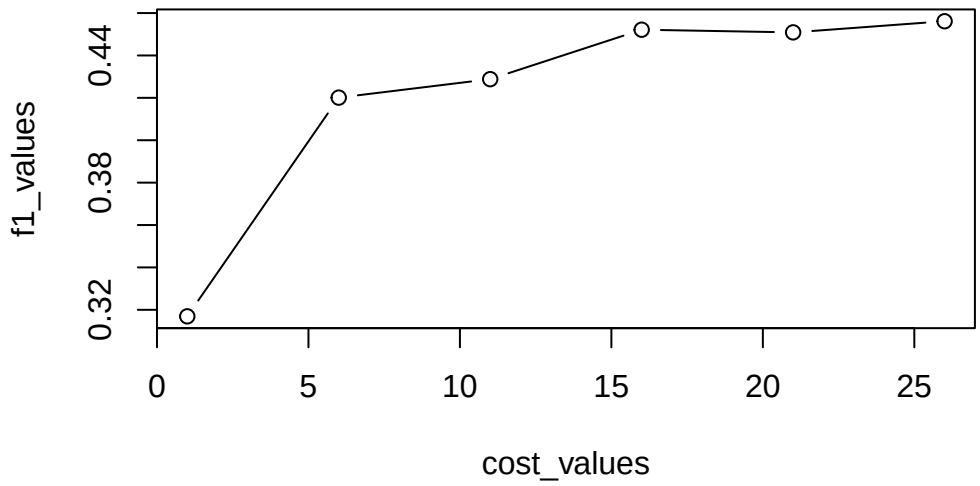
```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 177 117
         0 223 466

               Accuracy : 0.6541
                 95% CI : (0.6234, 0.6839)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 4.880e-05

                  Kappa : 0.2523

 Mcnemar's Test P-Value : 1.238e-08

            Sensitivity : 0.4425
            Specificity : 0.7993
         Pos Pred Value : 0.6020
         Neg Pred Value : 0.6763
             Prevalence : 0.4069
         Detection Rate : 0.1801
   Detection Prevalence : 0.2991
      Balanced Accuracy : 0.6209

       'Positive' Class : 1
```

```r
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1_svm <- 2 * (precision * recall) / (precision + recall)
f1_svm
```

```
Pos Pred Value
     0.5100865
```

SVM radial kernel cost value 16 meaning that there is a higher penalty for misclassifcation and will have a narrower margin for the hyperplanes. results in better predictions in this case.

```
cost_values <- c(seq(from=1, to  = 26, by = 5))
f1_values <- sapply(cost_values, function(x){
  SVM_rbf_model <- ksvm(
  Potability ~ .,
  data = clean_train_data,
  kernel = "rbfdot", C = x
)
  rbf_pred <- predict(SVM_rbf_model, clean_test_data, type = "response")
  #agree <- ifelse(rbf_pred == eqtest_data$term_deposit_factor, 1, 0)
  #accuracy <-  sum(agree) / nrow(eqtest_data)

   cm <- confusionMatrix(rbf_pred, clean_test_data$Potability)

#cm

#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1 <- 2 * (precision * recall) / (precision + recall)

  return (f1)
})

plot(cost_values, f1_values, type = "b")
```



Logistic Regression

```
lmodel <- glm(
  Potability ~.,
  data = train_scaled,
  family = binomial
)

summary(lmodel)
```

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

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)      0.47929    0.04307  11.128   <2e-16 ***
ph               0.01465    0.04365   0.336   0.7372
Hardness         0.01931    0.04351   0.444   0.6572
Solids          -0.08432    0.04383  -1.924   0.0544 .
Chloramines     -0.04804    0.04322  -1.111   0.2664
```

```
Sulfate          0.03224    0.04375   0.737   0.4612
Conductivity    -0.02672    0.04317  -0.619   0.5359
Organic_carbon   0.07072    0.04327   1.634   0.1022
Trihalomethanes  0.01253    0.04316   0.290   0.7716
Turbidity        0.02573    0.04313   0.597   0.5507
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3051.8  on 2292  degrees of freedom
Residual deviance: 3042.1  on 2283  degrees of freedom
AIC: 3062.1

Number of Fisher Scoring iterations: 4
```

```r
log_pred_class <- predict(lmodel, test_scaled)

log_pred_class <- ifelse(pred_prob >= 0.5, 1, 0)
log_pred_class <- factor(pred_class, levels = c(0, 1))


#table(Predicted = rbf_pred_class, Actual = eqtest_data$term_deposit_factor)



cm <- confusionMatrix(log_pred_class, test_scaled$Potability)
```

```
Warning in confusionMatrix.default(log_pred_class, test_scaled$Potability):
Levels are not in the same order for reference and data. Refactoring data to
match.
```

```r
cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 185 193
         0 215 390
```

```
               Accuracy : 0.5849
                 95% CI : (0.5534, 0.616)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 0.7099

                  Kappa : 0.1326

 Mcnemar's Test P-Value : 0.2985

            Sensitivity : 0.4625
            Specificity : 0.6690
         Pos Pred Value : 0.4894
         Neg Pred Value : 0.6446
             Prevalence : 0.4069
         Detection Rate : 0.1882
   Detection Prevalence : 0.3845
      Balanced Accuracy : 0.5657

       'Positive' Class : 1
```

```r
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1_svm <- 2 * (precision * recall) / (precision + recall)
f1_svm
```

```
Pos Pred Value
     0.4755784
```

```r
lmodel <- glm(
  Potability ~.,
  data = train_data,
  family = binomial,
)



summary(lmodel)
```

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

Coefficients:
                  Estimate Std. Error z value Pr(>|z|)
(Intercept)      6.647e-02  7.403e-01   0.090   0.9285
ph               1.012e-02  3.015e-02   0.336   0.7372
Hardness         5.937e-04  1.338e-03   0.444   0.6572
Solids          -9.583e-06  4.981e-06  -1.924   0.0544 .
Chloramines     -2.985e-02  2.686e-02  -1.111   0.2664
Sulfate          8.918e-04  1.210e-03   0.737   0.4612
Conductivity    -3.302e-04  5.333e-04  -0.619   0.5359
Organic_carbon   2.143e-02  1.311e-02   1.634   0.1022
Trihalomethanes  8.108e-04  2.794e-03   0.290   0.7716
Turbidity        3.283e-02  5.502e-02   0.597   0.5507
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 3051.8  on 2292  degrees of freedom
Residual deviance: 3042.1  on 2283  degrees of freedom
AIC: 3062.1

Number of Fisher Scoring iterations: 4
```

```r
log_pred_class <- predict(lmodel, test_data)

log_pred_class <- ifelse(pred_prob >= 0.5, 1, 0)
log_pred_class <- factor(pred_class, levels = c(0, 1))


#table(Predicted = rbf_pred_class, Actual = eqtest_data$term_deposit_factor)



cm <- confusionMatrix(log_pred_class, test_data$Potability)
```

```
Warning in confusionMatrix.default(log_pred_class, test_data$Potability):
Levels are not in the same order for reference and data. Refactoring data to
match.
```

```
cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 185 193
         0 215 390

               Accuracy : 0.5849
                 95% CI : (0.5534, 0.616)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 0.7099

                  Kappa : 0.1326

 Mcnemar's Test P-Value : 0.2985

            Sensitivity : 0.4625
            Specificity : 0.6690
         Pos Pred Value : 0.4894
         Neg Pred Value : 0.6446
             Prevalence : 0.4069
         Detection Rate : 0.1882
   Detection Prevalence : 0.3845
      Balanced Accuracy : 0.5657

       'Positive' Class : 1
```

```
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1_svm <- 2 * (precision * recall) / (precision + recall)
f1_svm
```

```
Pos Pred Value
     0.4755784
```

```r
set.seed(42)

log_train <- train_data
log_test <- test_data

log_train$Potability <- factor(log_train$Potability, levels = c(1, 0), labels = c("Yes", "No
log_test$Potability <- factor(log_test$Potability, levels = c(1, 0), labels = c("Yes", "No"))


#optimizing logistic regression with caret
x <- log_train[, -which(names(log_train) == "Potability")]
y <- log_train$Potability

ctrl <- trainControl(
  method = "cv",
  number = 20,
  classProbs = TRUE,
  summaryFunction = twoClassSummary ,
  sampling = 'down' #
)

# finding alpha and lambda
glmnet_fit <- train(
  Potability ~ .,
  data = log_train,
  method = "glmnet",
  trControl = ctrl,
  tuneLength = 20,
  metric = "ROC"
)

glmnet_fit
```

```
glmnet

2293 samples
   9 predictor
   2 classes: 'Yes', 'No'

No pre-processing
Resampling: Cross-Validated (20 fold)
Summary of sample sizes: 2178, 2178, 2179, 2178, 2178, 2178, ...
```

Addtional sampling using down-sampling

Resampling results across tuning parameters:

alpha	lambda	ROC	Sens	Spec
0.1000000	0.001238637	0.4980884	0.4748943	0.5222736
0.1000000	0.001920526	0.4980884	0.4748943	0.5222736
0.1000000	0.002977806	0.4980884	0.4748943	0.5222736
0.1000000	0.004617136	0.4980884	0.4748943	0.5222736
0.1000000	0.007158942	0.4980884	0.4748943	0.5222736
0.1000000	0.011100053	0.4982037	0.4748943	0.5215694
0.1000000	0.017210807	0.4981530	0.4783298	0.5187425
0.1000000	0.026685628	0.4967976	0.4737844	0.5187324
0.1473684	0.001238637	0.5040959	0.4682347	0.5173843
0.1473684	0.001920526	0.5040959	0.4682347	0.5173843
0.1473684	0.002977806	0.5040959	0.4682347	0.5173843
0.1473684	0.004617136	0.5040959	0.4682347	0.5173843
0.1473684	0.007158942	0.5041279	0.4682347	0.5173843
0.1473684	0.011100053	0.5041652	0.4705074	0.5223642
0.1473684	0.017210807	0.5057794	0.4716702	0.5237827
0.1473684	0.026685628	0.5068357	0.4841966	0.5251811
0.1947368	0.001238637	0.5025932	0.4613108	0.5137726
0.1947368	0.001920526	0.5025932	0.4613108	0.5137726
0.1947368	0.002977806	0.5025932	0.4613108	0.5137726
0.1947368	0.004617136	0.5025932	0.4613108	0.5137726
0.1947368	0.007158942	0.5025921	0.4601744	0.5123742
0.1947368	0.011100053	0.5028824	0.4566860	0.5144970
0.1947368	0.017210807	0.5029268	0.4635307	0.5137928
0.1947368	0.026685628	0.5031401	0.4703488	0.5159054
0.2421053	0.001238637	0.5027535	0.4910148	0.5158954
0.2421053	0.001920526	0.5027535	0.4910148	0.5158954
0.2421053	0.002977806	0.5027535	0.4910148	0.5158954
0.2421053	0.004617136	0.5027535	0.4910148	0.5158954
0.2421053	0.007158942	0.5028059	0.4921512	0.5159054
0.2421053	0.011100053	0.5021523	0.4817918	0.5180181
0.2421053	0.017210807	0.5022328	0.4784091	0.5194467
0.2421053	0.026685628	0.5006211	0.4795719	0.5180684
0.2894737	0.001238637	0.5061936	0.4783298	0.5208350
0.2894737	0.001920526	0.5061936	0.4783298	0.5208350
0.2894737	0.002977806	0.5061936	0.4783298	0.5208350
0.2894737	0.004617136	0.5063057	0.4783298	0.5201107
0.2894737	0.007158942	0.5074441	0.4760571	0.5165694
0.2894737	0.011100053	0.5080969	0.4851744	0.5229477

0.2894737	0.017210807	0.5098367	0.4908827	0.5236620
0.2894737	0.026685628	0.5112034	0.4920455	0.5208249
0.3368421	0.001238637	0.4934056	0.4795719	0.5103622
0.3368421	0.001920526	0.4934056	0.4795719	0.5103622
0.3368421	0.002977806	0.4934056	0.4795719	0.5103622
0.3368421	0.004617136	0.4937744	0.4784355	0.5096479
0.3368421	0.007158942	0.4943801	0.4807082	0.5145976
0.3368421	0.011100053	0.4940829	0.4818182	0.5138632
0.3368421	0.017210807	0.4937207	0.4806554	0.5060765
0.3368421	0.026685628	0.4949761	0.4692918	0.5131590
0.3842105	0.001238637	0.5072666	0.4898784	0.5123742
0.3842105	0.001920526	0.5072666	0.4898784	0.5123742
0.3842105	0.002977806	0.5072666	0.4898784	0.5123742
0.3842105	0.004617136	0.5075902	0.4853594	0.5102616
0.3842105	0.007158942	0.5079891	0.4877114	0.5117002
0.3842105	0.011100053	0.5105666	0.4865486	0.5180382
0.3842105	0.017210807	0.5109772	0.4830338	0.5216097
0.3842105	0.026685628	0.5110668	0.4762156	0.5329477
0.4315789	0.001238637	0.5015042	0.4795983	0.5236620
0.4315789	0.001920526	0.5015042	0.4795983	0.5236620
0.4315789	0.002977806	0.5015524	0.4795983	0.5236620
0.4315789	0.004617136	0.5012989	0.4818710	0.5293461
0.4315789	0.007158942	0.5006120	0.4750529	0.5215795
0.4315789	0.011100053	0.4985340	0.4705603	0.5187525
0.4315789	0.017210807	0.4958789	0.4694767	0.5152314
0.4315789	0.026685628	0.4936367	0.4842495	0.5166398
0.4789474	0.001238637	0.5043833	0.4749207	0.5229980
0.4789474	0.001920526	0.5043833	0.4749207	0.5229980
0.4789474	0.002977806	0.5040625	0.4737844	0.5237022
0.4789474	0.004617136	0.5049652	0.4760835	0.5244165
0.4789474	0.007158942	0.5043324	0.4806290	0.5223139
0.4789474	0.011100053	0.5017849	0.4702960	0.5152616
0.4789474	0.017210807	0.4972815	0.4589588	0.5237726
0.4789474	0.026685628	0.5007844	0.4624736	0.5266197
0.5263158	0.001238637	0.5049850	0.4728594	0.5160262
0.5263158	0.001920526	0.5049850	0.4728594	0.5160262
0.5263158	0.002977806	0.5051305	0.4705867	0.5167304
0.5263158	0.004617136	0.5053653	0.4682875	0.5188934
0.5263158	0.007158942	0.5055509	0.4637421	0.5224245
0.5263158	0.011100053	0.5062657	0.4658827	0.5266700
0.5263158	0.017210807	0.5031033	0.4670455	0.5273843
0.5263158	0.026685628	0.5003343	0.4635571	0.5372133
0.5736842	0.001238637	0.5015217	0.4773256	0.5131388

0.5736842	0.001920526	0.5015217	0.4773256	0.5131388
0.5736842	0.002977806	0.5019552	0.4772992	0.5124044
0.5736842	0.004617136	0.5041984	0.4704810	0.5096076
0.5736842	0.007158942	0.5052487	0.4739165	0.5152515
0.5736842	0.011100053	0.5069663	0.4875000	0.5216197
0.5736842	0.017210807	0.5081078	0.4807082	0.5216398
0.5736842	0.026685628	0.5095674	0.5171247	0.4929074
0.6210526	0.001238637	0.5023061	0.4807347	0.5081891
0.6210526	0.001920526	0.5023061	0.4807347	0.5081891
0.6210526	0.002977806	0.5028727	0.4807347	0.5117203
0.6210526	0.004617136	0.5021948	0.4830338	0.5089235
0.6210526	0.007158942	0.5022860	0.4841702	0.5139135
0.6210526	0.011100053	0.4987518	0.4785412	0.5153622
0.6210526	0.017210807	0.4943205	0.4796512	0.4970121
0.6210526	0.026685628	0.4948592	0.5274577	0.4724044
0.6684211	0.001238637	0.5035328	0.4694239	0.5144366
0.6684211	0.001920526	0.5035808	0.4694239	0.5144366
0.6684211	0.002977806	0.5047122	0.4706131	0.5109054
0.6684211	0.004617136	0.5056940	0.4717495	0.5109256
0.6684211	0.007158942	0.5057324	0.4728858	0.5194467
0.6684211	0.011100053	0.5054867	0.4728858	0.5194366
0.6684211	0.017210807	0.5096462	0.4762685	0.5314487
0.6684211	0.026685628	0.5109331	0.5364429	0.4718913
0.7157895	0.001238637	0.5101235	0.4785148	0.5194970
0.7157895	0.001920526	0.5100600	0.4796512	0.5194970
0.7157895	0.002977806	0.5097803	0.4807611	0.5152314
0.7157895	0.004617136	0.5106113	0.4784355	0.5110262
0.7157895	0.007158942	0.5099457	0.4852008	0.5188028
0.7157895	0.011100053	0.5088896	0.4818446	0.5294366
0.7157895	0.017210807	0.5078119	0.4772727	0.5253219
0.7157895	0.026685628	0.5118315	0.5085888	0.5161871
0.7631579	0.001238637	0.5010693	0.4851216	0.5088330
0.7631579	0.001920526	0.5010375	0.4873943	0.5074145
0.7631579	0.002977806	0.5016375	0.4907770	0.5116700
0.7631579	0.004617136	0.5000114	0.4895877	0.5109557
0.7631579	0.007158942	0.5006943	0.4828224	0.5073843
0.7631579	0.011100053	0.5019294	0.4783298	0.5017404
0.7631579	0.017210807	0.5066556	0.4966173	0.5151610
0.7631579	0.026685628	0.5052897	0.5414641	0.4591650
0.8105263	0.001238637	0.5046380	0.4839852	0.5181187
0.8105263	0.001920526	0.5044759	0.4851216	0.5167103
0.8105263	0.002977806	0.5043714	0.4828488	0.5209557
0.8105263	0.004617136	0.5036517	0.4749207	0.5279980

```
  0.8105263  0.007158942  0.5022351  0.4716173  0.5322435
  0.8105263  0.011100053  0.5031416  0.4671247  0.5315694
  0.8105263  0.017210807  0.5067449  0.4648520  0.5153320
  0.8105263  0.026685628  0.5023660  0.6603066  0.3484809
  0.8578947  0.001238637  0.4975425  0.4783562  0.5025050
  0.8578947  0.001920526  0.4982128  0.4783562  0.5025050
  0.8578947  0.002977806  0.4983166  0.4748943  0.5025151
  0.8578947  0.004617136  0.4990904  0.4738108  0.5131187
  0.8578947  0.007158942  0.4986748  0.4646406  0.5201710
  0.8578947  0.011100053  0.4973704  0.4645613  0.5172032
  0.8578947  0.017210807  0.4927344  0.4933140  0.4785211
  0.8578947  0.026685628  0.4908003  0.7056818  0.2724748
  0.9052632  0.001238637  0.4955057  0.4624207  0.5166801
  0.9052632  0.001920526  0.4960710  0.4612844  0.5166801
  0.9052632  0.002977806  0.4964066  0.4658562  0.5201911
  0.9052632  0.004617136  0.4961163  0.4670455  0.5258451
  0.9052632  0.007158942  0.4977696  0.4659091  0.5272736
  0.9052632  0.011100053  0.4998224  0.4545190  0.5195674
  0.9052632  0.017210807  0.5068360  0.5261099  0.4876559
  0.9052632  0.026685628  0.4939133  0.7397727  0.2613984
  0.9526316  0.001238637  0.5030094  0.4897199  0.5272334
  0.9526316  0.001920526  0.5034601  0.4919926  0.5300704
  0.9526316  0.002977806  0.5036722  0.4863108  0.5286720
  0.9526316  0.004617136  0.5055464  0.4828753  0.5279577
  0.9526316  0.007158942  0.5077374  0.4852537  0.5202515
  0.9526316  0.011100053  0.5107806  0.4898256  0.5187928
  0.9526316  0.017210807  0.5088930  0.5058140  0.5047384
  0.9526316  0.026685628  0.4905922  0.7102273  0.2746076
  1.0000000  0.001238637  0.5142096  0.4955603  0.5280382
  1.0000000  0.001920526  0.5140499  0.4944239  0.5329779
  1.0000000  0.002977806  0.5136087  0.4921247  0.5308652
  1.0000000  0.004617136  0.5151153  0.4955603  0.5302213
  1.0000000  0.007158942  0.5154838  0.4875529  0.5224145
  1.0000000  0.011100053  0.5127179  0.4887421  0.5245171
  1.0000000  0.017210807  0.5154478  0.5477801  0.4769718
  1.0000000  0.026685628  0.4926611  0.8152748  0.1781690

ROC was used to select the optimal model using the largest value.
The final values used for the model were alpha = 1 and lambda = 0.007158942.
```

```
glmnet_fit$bestTune
```

```
    alpha      lambda
157     1 0.007158942
```

```
log_pred_class <- predict(glmnet_fit,  log_test)
```

```
cm <- confusionMatrix(log_pred_class, log_test$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction Yes  No
       Yes 205 289
       No  195 294

               Accuracy : 0.5076
                 95% CI : (0.4759, 0.5393)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0162

 Mcnemar's Test P-Value : 2.365e-05

            Sensitivity : 0.5125
            Specificity : 0.5043
         Pos Pred Value : 0.4150
         Neg Pred Value : 0.6012
             Prevalence : 0.4069
         Detection Rate : 0.2085
   Detection Prevalence : 0.5025
      Balanced Accuracy : 0.5084

       'Positive' Class : Yes
```

```r
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1_svm <- 2 * (precision * recall) / (precision + recall)
f1_svm
```

```
Pos Pred Value
      0.458613
```

checked logistic regression and tuning the alpha (regularization) and lambda (penalty). with alpha being 0 for Ridge and 1 for lasso. model performed best with lasso and a low alpha of 0.007.

```r
train_data$Potability <- factor(train_data$Potability, levels = c(0, 1))
test_data$Potability <- factor(test_data$Potability, levels = c(0, 1))
train_data$Potability <- relevel(train_data$Potability, ref = "1")
test_data$Potability <- relevel(test_data$Potability, ref = "1")


train_data$Potability <- as.numeric(as.character(train_data$Potability))
test_data$Potability  <- as.numeric(as.character(test_data$Potability))


num_cols <- sapply(train_data, is.numeric)

train_scaled <- train_data
test_scaled  <- test_data

train_scaled <- scale(train_data[,num_cols])

train_center <- attr(train_scaled, "scaled:center")
train_scale <- attr(train_scaled, "scaled:scale")

test_scaled <- scale(test_data,
  center = train_center,
  scale  = train_scale
)
```

```r
set.seed(42)
#neural network



neural_model <- neuralnet(
  Potability ~ . ,
  data = train_scaled,
  act.fct = "logistic", hidden = c(1),
  linear.output = FALSE
)

plot(neural_model)
#neural_model




neural_prob <- predict(neural_model, test_scaled) [,1]

neural_class <- ifelse(neural_prob >= 0.5, 1, 0)

neural_class <- factor(neural_class, levels = c(0,1))
neural_class <- relevel(neural_class, ref = "1")


train_data$Potability <- factor(train_data$Potability, levels = c(0, 1))
test_data$Potability <- factor(test_data$Potability, levels = c(0, 1))
train_data$Potability <- relevel(train_data$Potability, ref = "1")
test_data$Potability <- relevel(test_data$Potability, ref = "1")


cm <- confusionMatrix(neural_class, test_data$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1  50  23
```

```
         0 350 560

               Accuracy : 0.6205
                 95% CI : (0.5894, 0.651)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 0.04228

                  Kappa : 0.0981

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

            Sensitivity : 0.12500
            Specificity : 0.96055
         Pos Pred Value : 0.68493
         Neg Pred Value : 0.61538
             Prevalence : 0.40692
         Detection Rate : 0.05086
   Detection Prevalence : 0.07426
      Balanced Accuracy : 0.54277

       'Positive' Class : 1
```

```r
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1_svm <- 2 * (precision * recall) / (precision + recall)
f1_svm
```

```
Pos Pred Value
     0.2114165
```

neural net hidden layer 1 poor kappa and f1

```r
clean_train_data$Potability <- factor(clean_train_data$Potability, levels = c(0, 1))
clean_test_data$Potability <- factor(clean_test_data$Potability, levels = c(0, 1))
clean_train_data$Potability <- relevel(clean_train_data$Potability, ref = "1")
clean_test_data$Potability <- relevel(clean_test_data$Potability, ref = "1")
```

```r
clean_train_data$Potability <- as.numeric(as.character(clean_train_data$Potability))
clean_test_data$Potability  <- as.numeric(as.character(clean_test_data$Potability))


#num_cols <- sapply(train_data, is.numeric)

#train_scaled <- train_data
#test_scaled  <- test_data

#train_scaled <- scale(train_data[,num_cols])

#train_center <- attr(train_scaled, "scaled:center")
#train_scale <- attr(train_scaled, "scaled:scale")

#test_scaled <- scale(test_data,
#  center = train_center,
#  scale  = train_scale
#)


set.seed(42)
#neural network



neural_model <- neuralnet(
  Potability ~ . ,
  data = clean_train_data,
  act.fct = "logistic", hidden = c(1),
  linear.output = FALSE
)

plot(neural_model)
#neural_model




neural_prob <- predict(neural_model, clean_test_data) [,1]

neural_class <- ifelse(neural_prob >= 0.5, 1, 0)
```

```r
neural_class <- factor(neural_class, levels = c(0,1))
neural_class <- relevel(neural_class, ref = "1")


clean_train_data$Potability <- factor(clean_train_data$Potability, levels = c(0, 1))
clean_test_data$Potability <- factor(clean_test_data$Potability, levels = c(0, 1))
clean_train_data$Potability <- relevel(clean_train_data$Potability, ref = "1")
clean_test_data$Potability <- relevel(clean_test_data$Potability, ref = "1")


cm <- confusionMatrix(neural_class, clean_test_data$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1   0   0
         0 358 582

               Accuracy : 0.6191
                 95% CI : (0.5872, 0.6503)
    No Information Rate : 0.6191
    P-Value [Acc > NIR] : 0.5145

                  Kappa : 0

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

            Sensitivity : 0.0000
            Specificity : 1.0000
         Pos Pred Value :    NaN
         Neg Pred Value : 0.6191
             Prevalence : 0.3809
         Detection Rate : 0.0000
   Detection Prevalence : 0.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 1
```

```r
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1 <- 2 * (precision * recall) / (precision + recall)
f1
```

```
Pos Pred Value
           NaN
```

data with outliers removed or not scaled produced poor neural networks.

```r
set.seed(42)
#neural network



neural_model <- neuralnet(
  Potability ~ . ,
  data = train_data,
  act.fct = "logistic", hidden = c(1),
  linear.output = FALSE
)

plot(neural_model)
#neural_model




neural_prob <- predict(neural_model, test_data) [,1]

neural_class <- ifelse(neural_prob >= 0.5, 1, 0)

neural_class <- factor(neural_class, levels = c(0,1))
neural_class <- relevel(neural_class, ref = "1")


clean_train_data$Potability <- factor(clean_train_data$Potability, levels = c(0, 1))
```

```r
clean_test_data$Potability <- factor(clean_test_data$Potability, levels = c(0, 1))
clean_train_data$Potability <- relevel(clean_train_data$Potability, ref = "1")
clean_test_data$Potability <- relevel(clean_test_data$Potability, ref = "1")


cm <- confusionMatrix(neural_class, test_data$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 400 583
         0   0   0

               Accuracy : 0.4069
                 95% CI : (0.376, 0.4384)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 1

                  Kappa : 0

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

            Sensitivity : 1.0000
            Specificity : 0.0000
         Pos Pred Value : 0.4069
         Neg Pred Value :    NaN
             Prevalence : 0.4069
         Detection Rate : 0.4069
   Detection Prevalence : 1.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 1
```

```r
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]
```

```
f1_svm <- 2 * (precision * recall) / (precision + recall)
f1_svm
```

```
Pos Pred Value
     0.5784526
```

```
num_cols <- sapply(train_data, is.numeric)
num_cols["Potability"] <- FALSE

train_scaled_vals <- scale(train_data[, num_cols])

train_center <- attr(train_scaled_vals, "scaled:center")
train_scale <- attr(train_scaled_vals, "scaled:scale")

test_scaled_vals <- scale(test_data[, num_cols], center = train_center, scale = train_scale)

train_scaled <- data.frame(train_scaled_vals, Potability = train_data$Potability)

test_scaled <- data.frame(test_scaled_vals, Potability = test_data$Potability)
```

```
set.seed(42)



#hidden_layers <- c(seq(from=2, to  = 9, by = 1))
#f1_values <- sapply(hidden_layers, function(x){
# neural_model <- neuralnet(
 # Potability ~ . ,
#  data = train_scaled,
#  act.fct = "logistic", hidden = x,
 # linear.output = FALSE
#)
 # neural_prob <- predict(neural_model, test_scaled) [,1]

#neural_class <- ifelse(neural_prob >= 0.5, 1, 0)

#neural_class <- factor(neural_class, levels = c(0,1))
#neural_class <- relevel(neural_class, ref = "1")
```

```
#cm <- confusionMatrix(neural_class, test_data$Potability)

#cm

#calculate f1 from precision and recall
#precision <- cm$byClass["Pos Pred Value"]
#recall    <- cm$byClass["Sensitivity"]




#f1 <- 2 * (precision * recall) / (precision + recall)

 # return (f1)
#})

#plot(hidden_layers, f1_values, type = "b")
```

finding the optimal amount of nodes in a layer

```
set.seed(42)
#train_data$Potability <- as.numeric(as.character(train_data$Potability))
#test_data$Potability  <- as.numeric(as.character(test_data$Potability))

#hidden_layers <- c(seq(from=0, to  = 2, by = 1))
#f1_values <- sapply(hidden_layers, function(x){
 #neural_model <- neuralnet(
  #Potability ~ . ,
  #data = train_scaled,
  #act.fct = "logistic",
  #linear.output = FALSE,
  #if (x == 0) {
   #  hidden = 9

  #}
  #else{
   # hidden = c(9,x)
  #}


#)
```

```
 # neural_prob <- predict(neural_model, test_scaled) [,1]

#neural_class <- ifelse(neural_prob >= 0.5, 1, 0)

#neural_class <- factor(neural_class, levels = c(0,1))
#neural_class <- relevel(neural_class, ref = "1")


#cm <- confusionMatrix(neural_class, test_data$Potability)

#cm

#calculate f1 from precision and recall
#precision <- cm$byClass["Pos Pred Value"]
#recall    <- cm$byClass["Sensitivity"]




#f1 <- 2 * (precision * recall) / (precision + recall)

 # return (f1)
#})

#plot(hidden_layers, f1_values, type = "b")
```

```
set.seed(42)
#neural network



neural_model <- neuralnet(
  Potability ~ . ,
  data = train_scaled,
  act.fct = "logistic", hidden = c(9,1),
  linear.output = FALSE,
  threshold = 0.01
)

plot(neural_model)
#neural_model
```

```r
neural_prob <- predict(neural_model, test_scaled) [,1]

neural_class <- ifelse(neural_prob >= 0.5, 1, 0)

neural_class <- factor(neural_class, levels = c(0,1))
neural_class <- relevel(neural_class, ref = "1")



cm <- confusionMatrix(neural_class, test_data$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 333 532
         0  67  51

               Accuracy : 0.3906
                 95% CI : (0.36, 0.4219)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 1

                  Kappa : -0.0677

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

            Sensitivity : 0.83250
            Specificity : 0.08748
         Pos Pred Value : 0.38497
         Neg Pred Value : 0.43220
             Prevalence : 0.40692
         Detection Rate : 0.33876
   Detection Prevalence : 0.87996
      Balanced Accuracy : 0.45999
```

```
       'Positive' Class : 1
```

```
#garson(neural_model)
#olden(neural_model, out_var = 1)
#olden(neural_model, out_var = 0)



#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1_svm <- 2 * (precision * recall) / (precision + recall)
f1_svm
```

```
Pos Pred Value
     0.5264822
```

adding a second layer imporved iperformance slightly.

remove some features

```
#train_neural <- train_scaled |> dplyr::select(-Turbidity, - Sulfate, -Conductivity)
#test_neural <- test_scaled |> dplyr::select(-Turbidity, - Sulfate, -Conductivity)
#set.seed(42)
#neural network




#train_data$Potability <- as.numeric(as.character(train_data$Potability))
#test_data$Potability  <- as.numeric(as.character(test_data$Potability))

#hidden_layers <- c(seq(from=1, to  = 4, by = 1))
#f1_values <- sapply(hidden_layers, function(x){
# neural_model <- neuralnet(
 # Potability ~ . ,
 # data = train_neural,
#  act.fct = "logistic",
```

```
#  linear.output = FALSE,
 # threshold = 0.01,

#   hidden = c(x)



#)
# neural_prob <- predict(neural_model, test_neural) [,1]

#neural_class <- ifelse(neural_prob >= 0.5, 1, 0)

#neural_class <- factor(neural_class, levels = c(0,1))
#neural_class <- relevel(neural_class, ref = "1")


#cm <- confusionMatrix(neural_class, test_neural$Potability)

#cm

#calculate f1 from precision and recall
#precision <- cm$byClass["Pos Pred Value"]
#recall    <- cm$byClass["Sensitivity"]




#f1 <- 2 * (precision * recall) / (precision + recall)

 # return (f1)
#})

#plot(hidden_layers, f1_values, type = "b")
```

```
set.seed(42)
#neural network




#train_data$Potability <- as.numeric(as.character(train_data$Potability))
#test_data$Potability  <- as.numeric(as.character(test_data$Potability))
```

```
#hidden_layers <- c(seq(from=0, to  = 3, by = 1))
#f1_values <- sapply(hidden_layers, function(x){
 #neural_model <- neuralnet(
  #Potability ~ . ,
  #data = train_neural,
  #act.fct = "logistic",
  #linear.output = FALSE,
  #threshold = 0.05,
# if (x == 0){
 #   hidden = 3
# }
 # else{
  #  hidden = c(3,x)
#  }

#)
 # neural_prob <- predict(neural_model, test_neural) [,1]

#neural_class <- ifelse(neural_prob >= 0.5, 1, 0)

#neural_class <- factor(neural_class, levels = c(0,1))
#neural_class <- relevel(neural_class, ref = "1")


#cm <- confusionMatrix(neural_class, test_neural$Potability)

#cm

#calculate f1 from precision and recall
#precision <- cm$byClass["Pos Pred Value"]
#recall    <- cm$byClass["Sensitivity"]




#f1 <- 2 * (precision * recall) / (precision + recall)

 # return (f1)
#})

#plot(hidden_layers, f1_values, type = "b")
```

```
#set.seed(42)
#neural network



#neural_model <- neuralnet(
 # Potability ~ . ,
#  data = train_neural,
 # act.fct = "logistic", hidden = c(3,1),
#  linear.output = FALSE,
 # threshold = 0.05
#)

#plot(neural_model)
#neural_model






#neural_prob <- predict(neural_model, test_neural) [,1]

#neural_class <- ifelse(neural_prob >= 0.5, 1, 0)

#neural_class <- factor(neural_class, levels = c(0,1))
#neural_class <- relevel(neural_class, ref = "1")




#cm <- confusionMatrix(neural_class, test_neural$Potability)

#cm



#garson(neural_model)
#olden(neural_model, out_var = 1)
#olden(neural_model, out_var = 0)
```

```
#calculate f1 from precision and recall
#precision <- cm$byClass["Pos Pred Value"]
#recall    <- cm$byClass["Sensitivity"]


#f1 <- 2 * (precision * recall) / (precision + recall)
#f1
```

different structure of neural network 9, 1 vs 3,1. 3,1 had poorer accuracy but a higher f1.

switched to the nnet package.

```
neural_train <- train_scaled
neural_test <- test_scaled


neural_train$Potability <- factor(train_scaled$Potability, levels = c(0,1), labels = c("No","
neural_test$Potability <- factor(test_scaled$Potability, levels = c(0,1), labels = c("No","Ye

set.seed(42)

hidden_layers <- c(seq(from=1, to  = 10, by = 1))
f1_values <- sapply(hidden_layers, function(x){
nn_model <- nnet(
  Potability ~ .,
  data = neural_train,
  size = x,
  maxit = 200,
  decay = 0.01
)


#nn_model

#plot(nn_model)




neural_prob <- predict(nn_model, neural_test) [,1]
```

```r
neural_class <- ifelse(neural_prob >= 0.5, 1, 0)

neural_class <- factor(neural_class, levels = c(0,1))
neural_class <- relevel(neural_class, ref = "1")



train_data$Potability <- relevel(train_data$Potability, ref = "1")
neural_test$Potability <- relevel(test_data$Potability, ref = "1")

cm <- confusionMatrix(neural_class, neural_test$Potability)

#cm

#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1 <- 2 * (precision * recall) / (precision + recall)
print(cat('layer',x,'Accuracy',cm$overall["Accuracy"],'kappa',cm$overall["Kappa"],'f1',f1,'p
  return (f1)
})
```

```
# weights:  12
initial  value 1629.141561
iter  10 value 1504.430638
iter  20 value 1480.698730
iter  30 value 1480.146499
iter  40 value 1479.763508
iter  50 value 1479.373466
iter  50 value 1479.373466
iter  50 value 1479.373466
final  value 1479.373466
converged
layer 1 Accuracy 0.6276704 kappa 0.1146672 f1 0.2245763 precision 0.7361111NULL
# weights:  23
initial  value 1530.604065
iter  10 value 1504.698132
iter  20 value 1458.868507
iter  30 value 1447.291281
```

```
iter  40 value 1446.212709
iter  50 value 1445.411356
iter  60 value 1445.340097
iter  70 value 1445.325584
iter  80 value 1445.256724
iter  90 value 1443.756037
iter 100 value 1443.727228
final  value 1443.726503
converged
layer 2 Accuracy 0.6236012 kappa 0.1105987 f1 0.2386831 precision 0.6744186NULL
# weights:  34
initial  value 1558.312953
iter  10 value 1511.316686
iter  20 value 1475.518388
iter  30 value 1424.278505
iter  40 value 1406.663365
iter  50 value 1398.824131
iter  60 value 1393.267652
iter  70 value 1391.692911
iter  80 value 1391.223831
iter  90 value 1391.006874
iter 100 value 1390.118523
iter 110 value 1385.436078
iter 120 value 1379.612225
iter 130 value 1377.425760
iter 140 value 1375.244236
iter 150 value 1374.454432
iter 160 value 1373.890449
iter 170 value 1373.381685
iter 180 value 1373.256096
iter 190 value 1372.451125
iter 200 value 1371.762482
final  value 1371.762482
stopped after 200 iterations
layer 3 Accuracy 0.681587 kappa 0.2898792 f1 0.4943457 precision 0.6986301NULL
# weights:  45
initial  value 1535.740973
iter  10 value 1505.814997
iter  20 value 1451.369747
iter  30 value 1419.853006
iter  40 value 1401.582608
iter  50 value 1381.557656
iter  60 value 1369.588705
```

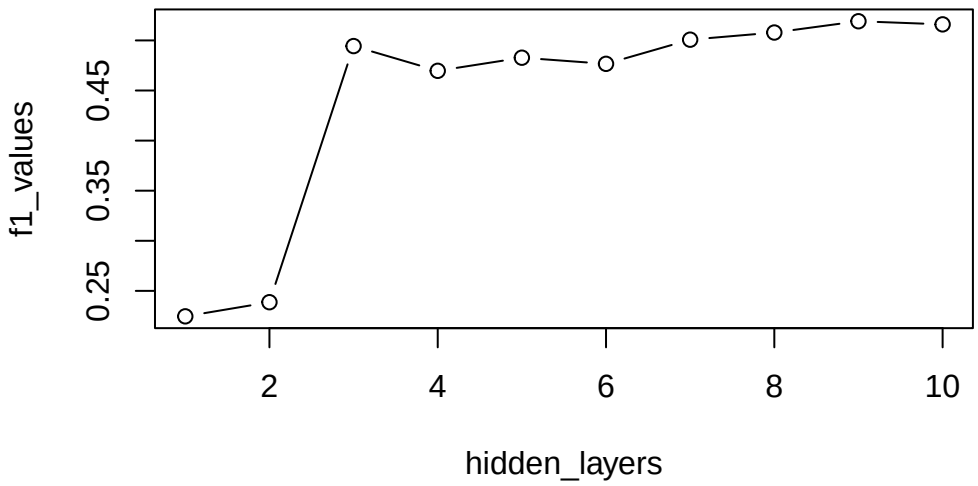
```
iter  70 value 1363.408514
iter  80 value 1361.245954
iter  90 value 1359.669503
iter 100 value 1358.277199
iter 110 value 1357.407663
iter 120 value 1357.024199
iter 130 value 1356.798271
iter 140 value 1356.022428
iter 150 value 1355.689413
iter 160 value 1355.665114
final  value 1355.664313
converged
layer 4 Accuracy 0.6622584 kappa 0.2489932 f1 0.4696486 precision 0.6504425NULL
# weights:  56
initial  value 1579.847786
iter  10 value 1482.445418
iter  20 value 1433.919532
iter  30 value 1407.040313
iter  40 value 1371.915270
iter  50 value 1350.252137
iter  60 value 1340.639311
iter  70 value 1336.684724
iter  80 value 1334.008945
iter  90 value 1331.589611
iter 100 value 1330.306537
iter 110 value 1328.929718
iter 120 value 1328.364836
iter 130 value 1328.157302
iter 140 value 1327.796056
iter 150 value 1327.400406
iter 160 value 1327.156093
iter 170 value 1327.110765
iter 180 value 1327.081318
iter 190 value 1327.058449
iter 190 value 1327.058440
iter 190 value 1327.058440
final  value 1327.058440
converged
layer 5 Accuracy 0.6490336 kappa 0.2328321 f1 0.4827586 precision 0.6029963NULL
# weights:  67
initial  value 1545.904197
iter  10 value 1483.927469
iter  20 value 1387.467395
```

```
iter  30 value 1359.221563
iter  40 value 1346.439376
iter  50 value 1343.100775
iter  60 value 1338.945824
iter  70 value 1335.177823
iter  80 value 1333.027727
iter  90 value 1331.026457
iter 100 value 1329.919180
iter 110 value 1329.198023
iter 120 value 1328.927211
iter 130 value 1328.805190
final  value 1328.795087
converged
layer 6 Accuracy 0.646999 kappa 0.2271049 f1 0.4766214 precision 0.6007605NULL
# weights:  78
initial  value 1677.833601
iter  10 value 1502.349417
iter  20 value 1431.290351
iter  30 value 1375.049519
iter  40 value 1350.102688
iter  50 value 1339.399429
iter  60 value 1334.204951
iter  70 value 1331.311915
iter  80 value 1329.080959
iter  90 value 1327.365208
iter 100 value 1326.300146
iter 110 value 1324.773207
iter 120 value 1323.746445
iter 130 value 1322.456921
iter 140 value 1321.029212
iter 150 value 1320.112210
iter 160 value 1319.763242
iter 170 value 1319.533506
iter 180 value 1319.070475
iter 190 value 1318.853291
iter 200 value 1318.714363
final  value 1318.714363
stopped after 200 iterations
layer 7 Accuracy 0.6551373 kappa 0.2499004 f1 0.5007364 precision 0.609319NULL
# weights:  89
initial  value 1706.260130
iter  10 value 1443.698462
iter  20 value 1365.535712
```

```
iter  30 value 1341.636401
iter  40 value 1332.513574
iter  50 value 1326.856624
iter  60 value 1320.164916
iter  70 value 1310.573933
iter  80 value 1302.943296
iter  90 value 1297.812626
iter 100 value 1296.710893
iter 110 value 1296.112516
iter 120 value 1295.903369
iter 130 value 1295.475553
iter 140 value 1293.689360
iter 150 value 1292.001126
iter 160 value 1291.482343
iter 170 value 1291.243830
iter 180 value 1291.087471
iter 190 value 1291.082657
iter 200 value 1291.078667
final  value 1291.078667
stopped after 200 iterations
layer 8 Accuracy 0.6510682 kappa 0.2466356 f1 0.507891 precision 0.5959596NULL
# weights:  100
initial  value 1609.466192
iter  10 value 1484.287677
iter  20 value 1409.377511
iter  30 value 1359.549165
iter  40 value 1339.253214
iter  50 value 1325.609213
iter  60 value 1317.467090
iter  70 value 1309.843307
iter  80 value 1305.456075
iter  90 value 1303.205572
iter 100 value 1299.838004
iter 110 value 1294.100717
iter 120 value 1293.276490
iter 130 value 1292.915044
iter 140 value 1292.845060
iter 150 value 1292.829392
iter 160 value 1292.822843
iter 170 value 1292.817046
iter 180 value 1292.816031
iter 190 value 1292.815728
final  value 1292.815517
```

```
converged
layer 9 Accuracy 0.6551373 kappa 0.2578488 f1 0.5191489 precision 0.6NULL
# weights:  111
initial  value 1685.956874
iter  10 value 1476.782256
iter  20 value 1377.490520
iter  30 value 1336.203330
iter  40 value 1309.625680
iter  50 value 1291.506652
iter  60 value 1284.521320
iter  70 value 1280.515171
iter  80 value 1277.017343
iter  90 value 1275.615236
iter 100 value 1274.597789
iter 110 value 1274.221998
iter 120 value 1273.600797
iter 130 value 1273.250253
iter 140 value 1273.153939
iter 150 value 1273.121472
iter 160 value 1273.056209
iter 170 value 1272.908203
iter 180 value 1272.791730
iter 190 value 1272.253464
iter 200 value 1270.782654
final  value 1270.782654
stopped after 200 iterations
layer 10 Accuracy 0.646999 kappa 0.2440322 f1 0.5160391 precision 0.5835962NULL
```

```r
plot(hidden_layers, f1_values, type = "b")
```



found that 3 or 9 nodes may perform best.

```
neural_train <- train_scaled
neural_test <- test_scaled


neural_train$Potability <- factor(train_scaled$Potability, levels = c(0,1), labels = c("No","
neural_test$Potability <- factor(test_scaled$Potability, levels = c(0,1), labels = c("No","Ye

set.seed(42)



nn_model <- nnet(
  Potability ~ .,
  data = neural_train,
  size = 3,
  maxit = 200,
  decay = 0.01
)
```

```
# weights:  34
initial  value 1903.854171
```

```
iter  10 value 1531.199877
iter  20 value 1496.174573
iter  30 value 1462.393026
iter  40 value 1452.768202
iter  50 value 1433.137530
iter  60 value 1405.812890
iter  70 value 1394.074865
iter  80 value 1389.883467
iter  90 value 1388.555375
iter 100 value 1386.245184
iter 110 value 1381.867158
iter 120 value 1378.241879
iter 130 value 1377.728331
iter 140 value 1377.114538
iter 150 value 1376.986225
iter 160 value 1376.554173
iter 170 value 1375.970367
iter 180 value 1375.362247
iter 190 value 1375.189021
iter 200 value 1375.178482
final  value 1375.178482
stopped after 200 iterations
```

```
nn_model
```

```
a 9-3-1 network with 34 weights
inputs: ph Hardness Solids Chloramines Sulfate Conductivity Organic_carbon Trihalomethanes Tu
output(s): Potability
options were - entropy fitting  decay=0.01
```

```
#plot(nn_model)
```

```
neural_prob <- predict(nn_model, neural_test) [,1]

neural_class <- ifelse(neural_prob >= 0.5, 1, 0)

neural_class <- factor(neural_class, levels = c(0,1))
```

```r
neural_class <- relevel(neural_class, ref = "1")



train_data$Potability <- relevel(train_data$Potability, ref = "1")
neural_test$Potability <- relevel(test_data$Potability, ref = "1")

cm <- confusionMatrix(neural_class, neural_test$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 144  63
         0 256 520

               Accuracy : 0.6755
                 95% CI : (0.6452, 0.7047)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 5.997e-08

                  Kappa : 0.2726

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.3600
            Specificity : 0.8919
         Pos Pred Value : 0.6957
         Neg Pred Value : 0.6701
             Prevalence : 0.4069
         Detection Rate : 0.1465
   Detection Prevalence : 0.2106
      Balanced Accuracy : 0.6260

       'Positive' Class : 1
```

```r
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]
```

```r
f1 <- 2 * (precision * recall) / (precision + recall)

f1
```

```
Pos Pred Value
     0.4744646
```

```r
neural_train <- train_scaled
neural_test <- test_scaled


neural_train$Potability <- factor(train_scaled$Potability, levels = c(0,1), labels = c("No","
neural_test$Potability <- factor(test_scaled$Potability, levels = c(0,1), labels = c("No","Ye



set.seed(62)

nn_model <- nnet(
  Potability ~ .,
  data = neural_train,
  size = 9,
  maxit = 200,
  decay = 0.01
)
```

```
# weights:  100
initial  value 2394.672258
iter  10 value 1444.646944
iter  20 value 1368.775470
iter  30 value 1334.219405
iter  40 value 1322.997824
iter  50 value 1316.752630
iter  60 value 1310.624708
iter  70 value 1300.031645
iter  80 value 1292.737565
iter  90 value 1288.789169
iter 100 value 1286.858601
iter 110 value 1286.262378
iter 120 value 1285.991334
```

```
iter 130 value 1285.862192
iter 140 value 1285.689169
iter 150 value 1285.487064
iter 160 value 1285.140824
iter 170 value 1284.629840
iter 180 value 1284.219392
iter 190 value 1283.289787
iter 200 value 1282.909216
final  value 1282.909216
stopped after 200 iterations
```

```
nn_model
```

```
a 9-9-1 network with 100 weights
inputs: ph Hardness Solids Chloramines Sulfate Conductivity Organic_carbon Trihalomethanes Tu
output(s): Potability
options were - entropy fitting  decay=0.01
```

```
#plot(nn_model)
```

```
neural_prob <- predict(nn_model, neural_test) [,1]

neural_class <- ifelse(neural_prob >= 0.5, 1, 0)

neural_class <- factor(neural_class, levels = c(0,1))
neural_class <- relevel(neural_class, ref = "1")
```

```
train_data$Potability <- relevel(train_data$Potability, ref = "1")
neural_test$Potability <- relevel(test_data$Potability, ref = "1")

cm <- confusionMatrix(neural_class, neural_test$Potability)

cm
```

```
Confusion Matrix and Statistics
```

```
          Reference
Prediction   1   0
         1 171 120
         0 229 463

               Accuracy : 0.645
                 95% CI : (0.6141, 0.6749)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 0.0004808

                  Kappa : 0.2316

 Mcnemar's Test P-Value : 7.421e-09

            Sensitivity : 0.4275
            Specificity : 0.7942
         Pos Pred Value : 0.5876
         Neg Pred Value : 0.6691
             Prevalence : 0.4069
         Detection Rate : 0.1740
   Detection Prevalence : 0.2960
      Balanced Accuracy : 0.6108

       'Positive' Class : 1
```

```r
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1 <- 2 * (precision * recall) / (precision + recall)

f1
```

```
Pos Pred Value
     0.4949349
```

```r
neural_train <- train_scaled
neural_test <- test_scaled
```

```r
neural_train$Potability <- factor(train_scaled$Potability, levels = c(0,1), labels = c("No","
neural_test$Potability <- factor(test_scaled$Potability, levels = c(0,1), labels = c("No","Y

set.seed(42)



nn_model <- nnet(
  Potability ~ .,
  data = neural_train,
  size = 3,
  maxit = 200,
  decay = 0.001
)
```

```
# weights:  34
initial  value 1903.800073
iter  10 value 1526.480537
iter  20 value 1522.674128
iter  30 value 1519.500603
iter  40 value 1512.128743
iter  50 value 1498.544491
iter  60 value 1490.030053
iter  70 value 1483.938119
iter  80 value 1480.585709
iter  90 value 1477.676224
iter 100 value 1476.591195
iter 110 value 1475.060015
iter 120 value 1435.440106
iter 130 value 1420.152005
iter 140 value 1418.468693
iter 150 value 1418.173735
iter 160 value 1416.483639
iter 170 value 1415.938185
iter 180 value 1414.907132
iter 190 value 1414.820350
iter 200 value 1414.786520
final  value 1414.786520
stopped after 200 iterations
```

```
nn_model
```

```
a 9-3-1 network with 34 weights
inputs: ph Hardness Solids Chloramines Sulfate Conductivity Organic_carbon Trihalomethanes Tu
output(s): Potability
options were - entropy fitting  decay=0.001
```

```
#plot(nn_model)
```

```
neural_prob <- predict(nn_model, neural_test) [,1]

neural_class <- ifelse(neural_prob >= 0.5, 1, 0)

neural_class <- factor(neural_class, levels = c(0,1))
neural_class <- relevel(neural_class, ref = "1")



train_data$Potability <- relevel(train_data$Potability, ref = "1")
neural_test$Potability <- relevel(test_data$Potability, ref = "1")

cm <- confusionMatrix(neural_class, neural_test$Potability)

cm
```

```
Confusion Matrix and Statistics

          Reference
Prediction   1   0
         1 124  51
         0 276 532

               Accuracy : 0.6673
                 95% CI : (0.6369, 0.6968)
    No Information Rate : 0.5931
    P-Value [Acc > NIR] : 9.688e-07
```

```
                 Kappa : 0.2441

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.3100
            Specificity : 0.9125
         Pos Pred Value : 0.7086
         Neg Pred Value : 0.6584
             Prevalence : 0.4069
         Detection Rate : 0.1261
   Detection Prevalence : 0.1780
      Balanced Accuracy : 0.6113

       'Positive' Class : 1
```

```
#calculate f1 from precision and recall
precision <- cm$byClass["Pos Pred Value"]
recall    <- cm$byClass["Sensitivity"]


f1 <- 2 * (precision * recall) / (precision + recall)

f1
```

```
Pos Pred Value
     0.4313043
```

optimal neural net had 3 layers with decay of 0.001.

Model

Parameters

Data

Accuracy

Kappa

F1 score

Decision Tree

Cp=0.01

Minsplit =20, maxdepth =20

Train_data

0.59

0.0856

0.3

Decision tree (deep)

Cp=0.001 Minsplit =20, maxdepth =20

Train_data

0.5849

0.1326

0.475578

Random Forest

Ntree = 500

Train_data

0.6755

0.2769

0.4863124

Random Forest

Ntree= 500

Clean_train_data

0.6489

0.1679

0.3653846

Adaboost

Mfinal = 100

Train_data

0.6521

0.2436

0.4970588

Adaboost

Mfinal = 100

Clean_train_data

0.6223

0.1561

0.4283414

SVM

Kernel Radial

Train_data

0.6816

0.2702

0.436036

SVM

Kernel Radial

Train_scaled

0.6816

0.27

0.436

SVM

Kernel Linear

Train_data

0.5931

0

Nan

SVM

Kernel tanhdot

Train_data

0.4334

-0.1894

0.273794

SVM

Kernel Radial, cost value = 16

Train_data

0.65

0.25

0.5

Logistic Regression

Glm default

Train_scaled

0.5849

0.1326

0.475784

Logistic Regression

Glm default

Train_data

0.5931

0.1429

0.45667

Logistic Regression

Alpha = 1, lambda =0.007159 glmnet trained

Train_data

0.50776

0.162

0.4586

Neural network

Neuralnet, act.function = logistic hidden= 1

Train_data

0.4069

0

0.578426

Neural Network

Neuralnet, act.function = logistic hidden= 1

Train_scaled

0.6205

0.0981

0.2114165

Neural network

Neuralnet, act.function = logistic hidden= 1

Clean_train_data

0.619

0

Nan

Neural network

Neuralnet, act.function = logistic hidden= 9,1

Train_scaled

0.639

0.1603

0.314

Neural network

Neuralnet, act.function = logistic hidden= 3,1

Train_scaled (3 features were removed)

0.4059

0.0661

0.511

Neural Network

Nnet, decay = 0.01, maxit=200, size 9

Train scaled

0.6368

0.218

0.493617

Neural network

Nnet, decay =0.001, iterations =200 , size =3

Train_scaled

0.6931

0.343

0.5758929

One thing to keep in mind is that if a model chose only the majority non potable class, then it would have a 60% accuracy, so checking the f1 score and kappa is crucial for model selection, as it shows that it is actually extrapolating from the data to determine classes.

Some insights from models:

The worse performing model was a simple decision tree. A deep decision tree performed better overall on the test data than the shallow one. Performance of a deep decision tree matched logistic regression with accuracies of 0.6 and f1 of ~0.45. It was notable that switching between scaled and unscaled data did not affect the performance of these models significantly.

Random forest and performed well on this dataset achieving accuracies above the 0.6 and had f1 scores around 0.5. There was a notable dip in performance of the models when the data had outliers removed. This may be due to the fact that the outliers may determine the potability of the water. Especially the values significantly above the mean. When looking at the features potabile water should have a pH from 6.5 to 8.5, total disloved solids less than 1000mg/L, less than 4 ppm for chloramines, <400uS/cm for conductivity,< 2mg/L for total organic carbon, <80 ppm for trihalomethanes, <5 NTU for turbidity.

SVM required careful selection of the kernel and by optimizing the cost value the f1 score improved. The radial kernel was shown to be the best when compared to linear and hyperbolic tangent kernel.

Models

Best Methods

Model

Parameters

Data

Accuracy

Kappa

F1 score

Neural network

Nnet, decay =0.001, iterations =200 , size =3

Train_scaled

0.6931

0.343

0.5758929

Random Forest

Ntree = 500

Train_data

0.6755

0.2769

0.4863124

SVM

Kernel Radial, cost value = 16

Train_data

0.65

0.25

0.5

Adaboost

Mfinal = 100

Train_data

0.6521

0.2436

0.4970588

After testing multiple models, the best performing models had accuracies greater than 0.65, f1 scores at or above 0.5, and kappas above 0.2. The best model had 1 layer of 3 nodes, from nnet, with a decay of 0.001. Random forest, SVM with a tuned cost value and adaboost all had similar performances.

Ultimately all the features that determine waters potability is not within this dataset. Building a model with limited data has it's challenges however some predictions could be made using a neural network, showcasing that machine learning models are useful for datasets that are limited.

This may help with quality control of water and may help raise flags that may help save lives.

reference for features:

”

The water_potability.csv file contains water quality metrics for 3276 different water bodies.

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 S/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.

”

-Water Quality