Mathematical Methods for Artificial Intelligence Lab 1 and 2

Vytautas Kraujalis

2021-10-10

Contents

1 Lab 1

1.1 Reading Data

```
set.seed(123)

data <- read.csv("superconductor_dataset.csv")
```

1.2 Required packages

```
library(SmartEDA)
library(dplyr)
library(purrr)
library(ggplot2)
library(corrplot)
library(caret)
library(glmnet)
```

1.3 EDA, first look at the dataset

```
ExpData(data,type=1)
```

```
##                                          Descriptions      Value
## 1                                  Sample size (nrow)      21263
## 2                             No. of variables (ncol)        169
## 3                     No. of numeric/interger variables        168
## 4                               No. of factor variables          0
## 5                                 No. of text variables          1
## 6                              No. of logical variables          0
## 7                           No. of identifier variables          0
## 8                                 No. of date variables          0
## 9              No. of zero variance variables (uniform)          9
## 10                %. of variables having complete cases 100% (169)
## 11   %. of variables having >0% and <50% missing cases     0% (0)
## 12 %. of variables having >=50% and <90% missing cases     0% (0)
## 13          %. of variables having >=90% missing cases     0% (0)
```

We have 168 numeric variables and only 1 text variable, no missing values. There are 9 columns with zero variance, we'll look at it later.

Let's look at the text variable:

```
head(data$material)
```

```
## [1] "Ba0.2La1.8Cu1O4"        "Ba0.1La1.9Ag0.1Cu0.9O4" "Ba0.1La1.9Cu1O4"
## [4] "Ba0.15La1.85Cu1O4"      "Ba0.3La1.7Cu1O4"        "Ba0.5La1.5Cu1O4"
```

Out of total 21263 observations, we have 15542 unique record in this text column. We won't be using this variable in our analysis.

We should look at the variables with zero variance:

```
data <- data %>% select(-material)

# Return a character vector of variable names which have 0 variance
variables_with_zero_var <- names(data)[which(map_dbl(data, var) == 0)]

summary(data[,variables_with_zero_var])
```

```
##        He          Ne          Ar          Kr          Xe          Pm
##  Min.   :0   Min.   :0   Min.   :0   Min.   :0   Min.   :0   Min.   :0
##  1st Qu.:0   1st Qu.:0   1st Qu.:0   1st Qu.:0   1st Qu.:0   1st Qu.:0
##  Median :0   Median :0   Median :0   Median :0   Median :0   Median :0
##  Mean   :0   Mean   :0   Mean   :0   Mean   :0   Mean   :0   Mean   :0
##  3rd Qu.:0   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0   3rd Qu.:0
##  Max.   :0   Max.   :0   Max.   :0   Max.   :0   Max.   :0   Max.   :0
##        Po          At          Rn
##  Min.   :0   Min.   :0   Min.   :0
##  1st Qu.:0   1st Qu.:0   1st Qu.:0
##  Median :0   Median :0   Median :0
##  Mean   :0   Mean   :0   Mean   :0
##  3rd Qu.:0   3rd Qu.:0   3rd Qu.:0
##  Max.   :0   Max.   :0   Max.   :0
```

```
data <- data %>% select(-all_of(variables_with_zero_var))
```

Those 9 variables have zero variance, we'll exclude them

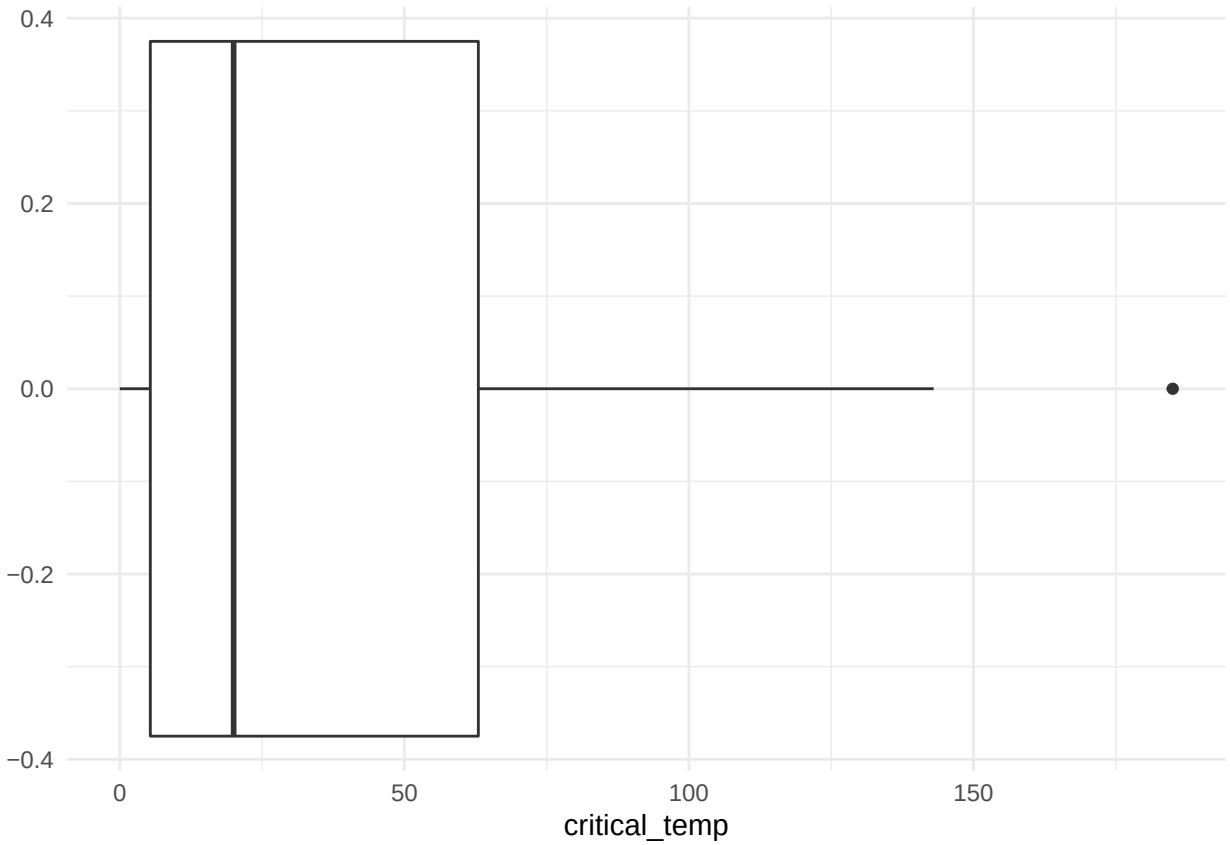
We should look into our target variable - critical_temp

```
summary( data %>% select(critical_temp) )
```

```
##  critical_temp
##  Min.   :  0.00021
##  1st Qu.:  5.36500
##  Median : 20.00000
##  Mean   : 34.42122
##  3rd Qu.: 63.00000
##  Max.   :185.00000
```

We see a maximum value of critical_temp to be 185, while 3rd Quantile - 63. Let's look at the boxplot:

```
ggplot(data, aes(x=critical_temp)) +
  geom_boxplot() +
  theme_minimal()
```



Seems like we have one outlier at >150. We will remove this observation

```
data <- data %>% filter(critical_temp < 150)
```

Let's look at descriptive statistics of each variable:

```
ExpNumStat(data,by ="A",round= 2, gp = "critical_temp") %>%
  select(Vname, min, max, mean, median, SD)
```

```
##                                  Vname    min      max    mean  median      SD
## 123                                 Ag   0.00     7.00    0.01    0.00    0.17
## 91                                  Al   0.00    99.92    0.06    0.00    1.13
## 110                                 As   0.00    18.00    0.16    0.00    1.08
## 153                                 Au   0.00    64.00    0.02    0.00    0.72
## 84                                   B   0.00   105.00    0.14    0.00    1.04
## 131                                 Ba   0.00    24.00    0.57    0.00    0.98
## 83                                  Be   0.00    40.00    0.03    0.00    0.85
## 157                                 Bi   0.00    14.00    0.20    0.00    0.66
## 112                                 Br   0.00     5.00    0.00    0.00    0.08
## 85                                   C   0.00   120.00    0.38    0.00    4.41
## 97                                  Ca   0.00    24.00    0.26    0.00    0.90
## 124                                 Cd   0.00   100.00    0.01    0.00    0.69
## 133                                 Ce   0.00     5.00    0.03    0.00    0.17
## 95                                  Cl   0.00     3.00    0.01    0.00    0.12
## 104                                 Co   0.00    35.38    0.04    0.00    0.58
```

101	Cr	0.00	34.90	0.01	0.00	0.25
80	critical_temp	0.00	143.00	34.41	20.00	34.24
130	Cs	0.00	3.00	0.00	0.00	0.08
106	Cu	0.00	98.00	1.28	0.90	2.08
140	Dy	0.00	5.00	0.01	0.00	0.10
5	entropy_atomic_mass	0.00	1.98	1.17	1.20	0.36
25	entropy_atomic_radius	0.00	2.14	1.27	1.33	0.38
35	entropy_Density	0.00	1.95	1.07	1.09	0.34
45	entropy_ElectronAffinity	0.00	1.77	1.07	1.14	0.34
15	entropy_fie	0.00	2.16	1.30	1.36	0.38
55	entropy_FusionHeat	0.00	2.03	1.09	1.11	0.38
65	entropy_ThermalConductivity	0.00	1.63	0.73	0.74	0.33
75	entropy_Valence	0.00	2.14	1.30	1.37	0.39
142	Er	0.00	5.00	0.01	0.00	0.13
137	Eu	0.00	6.00	0.02	0.00	0.15
88	F	0.00	4.00	0.01	0.00	0.13
103	Fe	0.00	30.00	0.15	0.00	0.71
108	Ga	0.00	41.00	0.07	0.00	1.12
138	Gd	0.00	4.00	0.02	0.00	0.16
109	Ge	0.00	46.00	0.08	0.00	1.02
3	gmean_atomic_mass	5.32	208.98	71.29	66.36	31.03
23	gmean_atomic_radius	48.00	298.00	144.45	142.81	22.09
33	gmean_Density	1.43	22590.00	3460.85	1339.97	3703.27
43	gmean_ElectronAffinity	1.50	326.10	54.36	51.47	29.00
13	gmean_fie	375.50	1313.10	737.46	727.96	78.28
53	gmean_FusionHeat	0.22	105.00	10.14	5.25	10.07
63	gmean_ThermalConductivity	0.03	317.88	29.84	14.29	34.06
73	gmean_Valence	1.00	7.00	3.06	2.62	1.05
81	H	0.00	14.00	0.02	0.00	0.27
146	Hf	0.00	25.00	0.01	0.00	0.21
154	Hg	0.00	8.00	0.04	0.00	0.21
141	Ho	0.00	5.00	0.01	0.00	0.10
129	I	0.00	4.00	0.00	0.00	0.09
115	Y	0.00	9.00	0.18	0.00	0.43
144	Yb	0.00	16.00	0.01	0.00	0.21
125	In	0.00	31.50	0.05	0.00	0.52
151	Ir	0.00	45.00	0.06	0.00	0.86
96	K	0.00	3.30	0.02	0.00	0.14
132	La	0.00	98.00	0.26	0.00	2.32
82	Li	0.00	3.00	0.01	0.00	0.13
145	Lu	0.00	7.00	0.03	0.00	0.28
1	mean_atomic_mass	6.94	208.98	87.56	84.92	29.67
21	mean_atomic_radius	48.00	298.00	157.99	160.25	20.14
31	mean_Density	1.43	22590.00	6111.71	5329.09	2846.63
41	mean_ElectronAffinity	1.50	326.10	76.88	73.10	27.70
11	mean_fie	375.50	1313.10	769.60	764.90	87.45
51	mean_FusionHeat	0.22	105.00	14.30	9.30	11.30
61	mean_ThermalConductivity	0.03	332.50	89.71	96.50	38.51
71	mean_Valence	1.00	7.00	3.20	2.83	1.04
90	Mg	0.00	12.00	0.03	0.00	0.27
102	Mn	0.00	14.00	0.00	0.00	0.13
118	Mo	0.00	99.99	0.15	0.00	2.08
86	N	0.00	12.80	0.01	0.00	0.15
89	Na	0.00	4.00	0.01	0.00	0.10

```
## 117                             Nb   0.00    99.98    0.44    0.00    4.85
## 135                             Nd   0.00     6.00    0.04    0.00    0.22
## 105                             Ni   0.00    45.00    0.09    0.00    0.98
## 87                               O   0.00    66.00    3.01    1.00    3.81
## 150                             Os   0.00    10.00    0.02    0.00    0.28
## 93                               P   0.00    20.00    0.03    0.00    0.47
## 156                             Pb   0.00    19.00    0.04    0.00    0.27
## 122                             Pd   0.00    51.00    0.09    0.00    1.55
## 134                             Pr   0.00   185.00    0.04    0.00    1.28
## 152                             Pt   0.00     5.80    0.03    0.00    0.31
## 7               range_atomic_mass   0.00   207.97  115.61  122.91   54.63
## 27             range_atomic_radius   0.00   256.00  139.33  171.00   67.27
## 37                  range_Density   0.00 22588.57 8665.75 8958.57 4096.97
## 47          range_ElectronAffinity   0.00   349.00  120.73  127.05   58.70
## 17                       range_fie   0.00  1304.50  572.23  764.10  309.62
## 57               range_FusionHeat   0.00   104.78   21.14   12.88   20.37
## 67       range_ThermalConductivity   0.00   429.97  250.91  399.80  158.70
## 113                             Rb   0.00     4.00    0.01    0.00    0.12
## 149                             Re   0.00    97.24    0.04    0.00    1.18
## 121                             Rh   0.00    45.00    0.07    0.00    1.01
## 120                             Ru   0.00    64.00    0.06    0.00    0.77
## 94                               S   0.00    15.00    0.11    0.00    0.76
## 127                             Sb   0.00    83.50    0.10    0.00    1.84
## 98                              Sc   0.00     5.00    0.01    0.00    0.19
## 111                             Se   0.00    19.00    0.08    0.00    0.68
## 92                              Si   0.00   100.00    0.19    0.00    2.22
## 136                             Sm   0.00    12.00    0.02    0.00    0.18
## 126                             Sn   0.00    99.20    0.12    0.00    1.89
## 114                             Sr   0.00    16.70    0.33    0.00    0.76
## 9                std_atomic_mass   0.00   101.02   44.39   45.12   20.03
## 29              std_atomic_radius   0.00   115.50   51.60   58.66   22.90
## 39                   std_Density   0.00 10724.37 3417.03 3301.89 1673.58
## 49           std_ElectronAffinity   0.00   162.90   48.91   51.13   21.74
## 19                       std_fie   0.00   499.67  215.63  266.37  109.97
## 59                std_FusionHeat   0.00    51.63    8.32    4.95    8.67
## 69        std_ThermalConductivity   0.00   214.99   98.95  135.76   60.14
## 78                    std_Valence   0.00     3.00    0.84    0.80    0.48
## 147                             Ta   0.00    55.00    0.04    0.00    0.85
## 139                             Tb   0.00     5.00    0.00    0.00    0.06
## 119                             Tc   0.00     6.00    0.00    0.00    0.06
## 128                             Te   0.00    66.70    0.04    0.00    0.72
## 99                              Ti   0.00    75.00    0.16    0.00    2.73
## 155                             Tl   0.00     7.00    0.05    0.00    0.27
## 143                             Tm   0.00     5.00    0.01    0.00    0.13
## 100                              V   0.00    79.50    0.22    0.00    3.41
## 148                              W   0.00    14.00    0.01    0.00    0.16
## 6          wtd_entropy_atomic_mass   0.00     1.96    1.06    1.15    0.40
## 26        wtd_entropy_atomic_radius   0.00     1.90    1.13    1.24    0.41
## 36             wtd_entropy_Density   0.00     1.70    0.86    0.88    0.32
## 46     wtd_entropy_ElectronAffinity   0.00     1.68    0.77    0.78    0.29
## 16                 wtd_entropy_fie   0.00     2.04    0.93    0.92    0.33
## 56           wtd_entropy_FusionHeat   0.00     1.75    0.91    0.99    0.37
## 66  wtd_entropy_ThermalConductivity   0.00     1.61    0.54    0.55    0.32
## 76              wtd_entropy_Valence   0.00     1.95    1.05    1.17    0.38
```

```
## 4             wtd_gmean_atomic_mass   1.96   208.98   58.54   39.92   36.65
## 24          wtd_gmean_atomic_radius  48.00   298.00  120.99  113.18   35.84
## 34                wtd_gmean_Density   0.69 22590.00 3117.39 1515.53 3975.16
## 44       wtd_gmean_ElectronAffinity   1.50   326.10   72.41   73.17   31.65
## 14                     wtd_gmean_fie 375.50  1327.59  832.75  856.19  119.75
## 54              wtd_gmean_FusionHeat   0.22   105.00   10.14    4.93   13.13
## 64    wtd_gmean_ThermalConductivity   0.02   376.03   27.31    6.10   40.19
## 74                wtd_gmean_Valence   1.00     7.00    3.06    2.43    1.17
## 2              wtd_mean_atomic_mass   6.42   208.98   72.99   60.70   33.49
## 22           wtd_mean_atomic_radius  48.00   298.00  134.72  125.97   28.80
## 32                wtd_mean_Density   1.43 22590.00 5267.41 4303.71 3221.23
## 42       wtd_mean_ElectronAffinity   1.50   326.10   92.72  102.86   32.28
## 12                     wtd_mean_fie 375.50  1348.03  870.43  889.96  143.26
## 52              wtd_mean_FusionHeat   0.22   105.00   13.85    8.33   14.28
## 62     wtd_mean_ThermalConductivity   0.03   406.96   81.55   73.33   45.52
## 72                wtd_mean_Valence   1.00     7.00    3.15    2.62    1.19
## 8              wtd_range_atomic_mass   0.00   205.59   33.23   26.64   26.97
## 28           wtd_range_atomic_radius   0.00   240.16   51.37   43.00   35.02
## 38                wtd_range_Density   0.00 22434.16 2902.84 2082.96 2398.48
## 48       wtd_range_ElectronAffinity   0.00   218.70   59.33   71.16   28.62
## 18                     wtd_range_fie   0.00  1251.86  483.51  510.44  224.05
## 58              wtd_range_FusionHeat   0.00   102.67    8.22    3.44   11.41
## 68    wtd_range_ThermalConductivity   0.00   401.44   62.04   56.56   43.12
## 77                wtd_range_Valence   0.00     6.99    1.48    1.06    0.98
## 10              wtd_std_atomic_mass   0.00   101.02   41.45   44.29   19.98
## 30            wtd_std_atomic_radius   0.00    97.14   52.34   59.94   25.29
## 40                 wtd_std_Density   0.00 10410.93 3319.28 3625.63 1611.75
## 50         wtd_std_ElectronAffinity   0.00   169.08   44.41   48.03   20.43
## 20                      wtd_std_fie   0.00   479.16  224.05  258.46  127.93
## 60              wtd_std_FusionHeat   0.00    51.68    7.72    5.50    7.29
## 70      wtd_std_ThermalConductivity   0.00   213.30   96.24  113.56   63.71
## 79                wtd_std_Valence   0.00     3.00    0.67    0.50    0.46
## 107                            Zn   0.00    20.00    0.01    0.00    0.40
## 116                            Zr   0.00    96.71    0.37    0.00    4.85
```

We should look at the correlation between variables

```
# Correlation

corr_simple <- function(df,sig=0.5){
  corr <- cor(df)
  #prepare to drop duplicates and correlations of 1
  corr[lower.tri(corr,diag=TRUE)] <- NA
  #drop perfect correlations
  corr[corr == 1] <- NA
  #turn into a 3-column table
  corr <- as.data.frame(as.table(corr))
  #remove the NA values from above
  corr <- na.omit(corr)
  #select significant values
  corr <- subset(corr, abs(Freq) > sig)
  #sort by highest correlation
  corr <- corr[order(-abs(corr$Freq)),]
  return(corr)
```

```
}

correlation_matrix = cor(data)

length(findCorrelation(correlation_matrix, cutoff = 0.99))
```

```
## [1] 4
```

```
length(findCorrelation(correlation_matrix, cutoff = 0.95))
```

```
## [1] 22
```

```
length(findCorrelation(correlation_matrix, cutoff = 0.9))
```

```
## [1] 35
```

We have 4 variables with higher than .99 correlation. We have 22 variables with higher than .95 correlation. We have 35 variables with higher than .9 correlation.

1.4 Pre-process

1.4.1 Data split

```
indices <- createDataPartition(data$critical_temp, p = 0.8, list = FALSE)
train <- data[indices,]
test <- data[-indices,]
```

Splitted a dataset by 80% / 20% rule. Created a training dataset with 17010 (80.0018813%) observations and testing dataset with 4252 (19.9981187%) observations.

1.4.2 Data transformation

```
preProcValues <- preProcess(train, method = c("center", "scale"))
train_transformed <- predict(preProcValues, train)
test_transformed <- predict(preProcValues, test)

label_index <- which(colnames(train_transformed) == "critical_temp")
```

We also centered and scaled our data. We transformed the testing dataset based on the pre process of training dataset.

1.5 Linear Regression

1.5.1 Linear Regression - no changes in data

```r
# linear regression
train_control_cv <- trainControl(method = "cv", number = 10)
fit_lm <- train(critical_temp ~ ., data = train_transformed, method = "lm", trControl = train_control_cv
```

We are performing linear regression model on the whole training dataset with 10 fold cross-validation. The model:

```r
print(fit_lm)
```

```
## Linear Regression
##
## 17010 samples
##   158 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 15309, 15309, 15310, 15309, 15309, 15307, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.5476384  0.7229608  0.3632821
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

Our RMSE = 0.55, R^2 = 72%, MAE = 0.36

Using fitted linear regression model on the testing dataset:

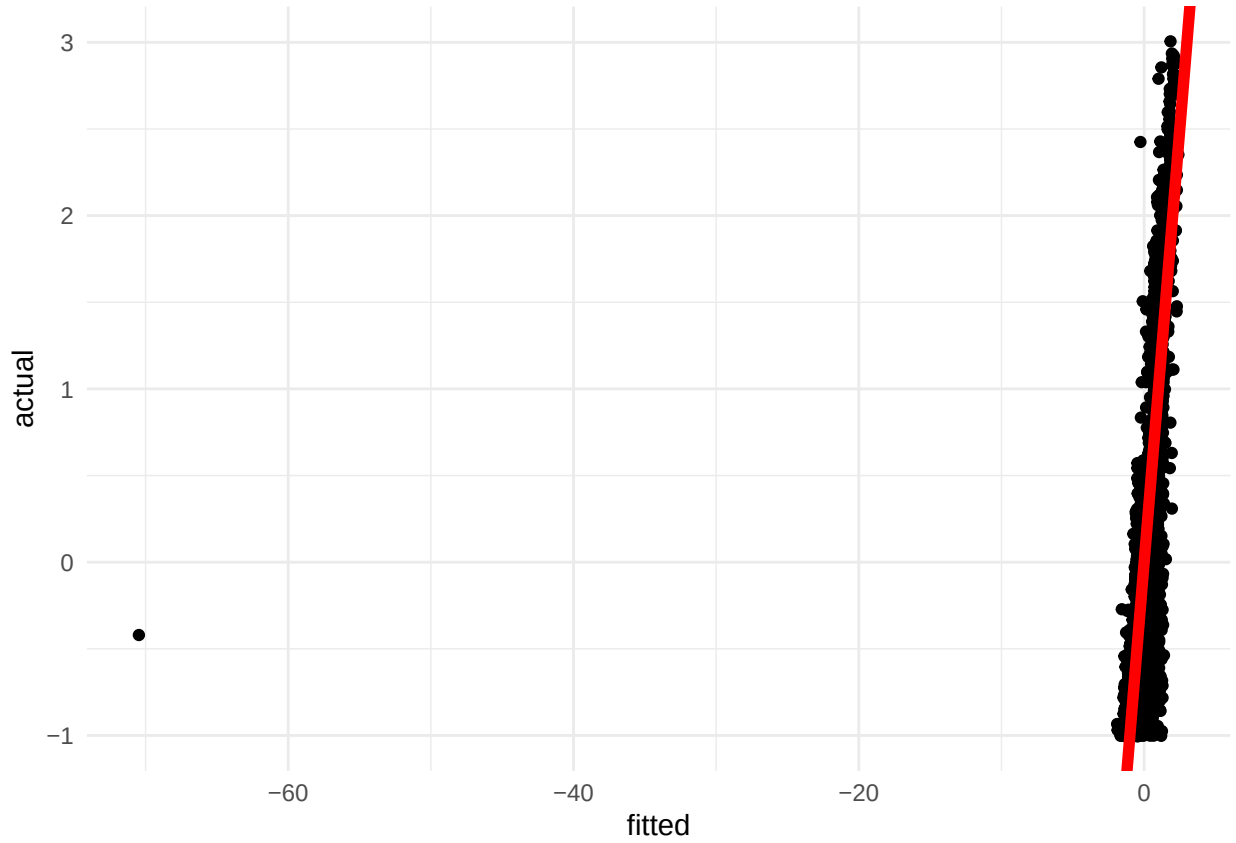
```r
test_lm <- predict(fit_lm, newdata = test_transformed)
print(round(postResample(pred = test_lm, obs = test_transformed$critical_temp), 3))
```

```
##     RMSE Rsquared      MAE
##    1.180    0.306    0.380
```

Our model, on the testing dataset, only explains ~31% of variance, while on a training set the rsquared was 72%. But the MAE was 0.36 for the training set and 0.38 for the testing dataset, it could mean an outlier influences the RMSE and rqsuared.

Let's look at the fitted vs actual plot:

```r
fitted_actual_lm <- data.frame(fitted = test_lm, actual = test_transformed$critical_temp)

ggplot(fitted_actual_lm,
       aes(x = fitted,
           y = actual)) +
  geom_point() +
  geom_abline(intercept = 0,
              slope = 1,
              color = "red",
              size = 2) +
  theme_minimal()
```



We clearly see one value, with an actual value of ~-0.5, while the fitted value is ~-70.

Let's look at that observation before transformation was applied:

```
test[test_lm < -60,] %>%
  tidyr::pivot_longer(cols = everything())
```

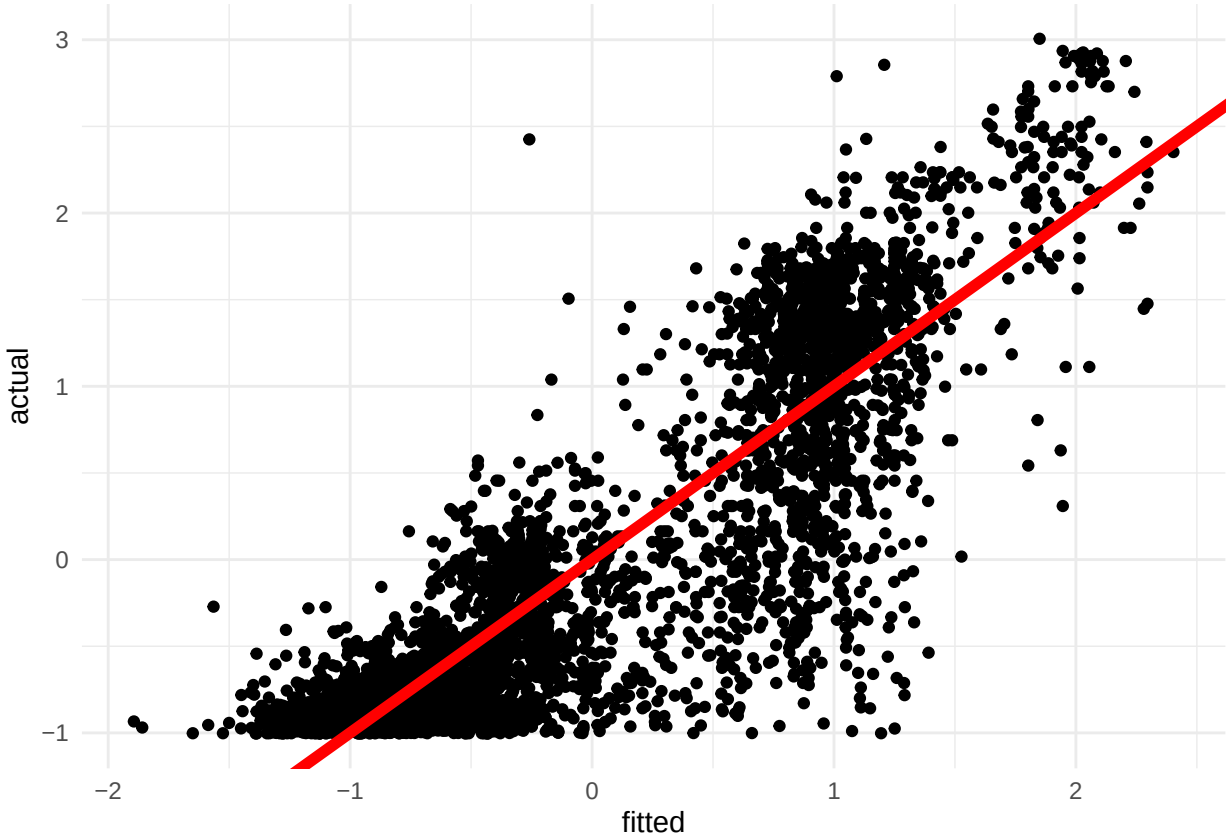
```
## # A tibble: 159 x 2
##    name                        value
##    <chr>                       <dbl>
##  1 number_of_elements        4
##  2 mean_atomic_mass         90.1
##  3 wtd_mean_atomic_mass    138.
##  4 gmean_atomic_mass        66.9
##  5 wtd_gmean_atomic_mass   134.
##  6 entropy_atomic_mass       1.18
##  7 wtd_entropy_atomic_mass   0.0406
##  8 range_atomic_mass       125.
##  9 wtd_range_atomic_mass   137.
## 10 std_atomic_mass          53.1
## # ... with 149 more rows
```

If we would exclude this observation from our testing dataset:

```r
# without the outlier
print(round(postResample(pred = test_lm[test_lm > -60], obs = test_transformed$critical_temp[test_lm > -
```

```
##      RMSE Rsquared      MAE
##     0.489    0.759    0.364
```

```r
ggplot(fitted_actual_lm %>% filter(fitted > -60),
       aes(x = fitted,
           y = actual)) +
  geom_point() +
  geom_abline(intercept = 0,
              slope = 1,
              color = "red",
              size = 2) +
  theme_minimal()
```



Our rsquared would be 76%, the MAE is 0.36 which is a reasonable result.

1.5.2 Linear Regression - removed correlated columns

```r
# Removed >.9 correlations
correlation_matrix = cor(train_transformed)
correlated_columns = findCorrelation(correlation_matrix, cutoff = 0.9)
```

```
correlated_columns = sort(correlated_columns)

colnames(train_transformed[, correlated_columns])
```

```
##  [1] "number_of_elements"          "gmean_atomic_mass"
##  [3] "wtd_gmean_atomic_mass"       "entropy_atomic_mass"
##  [5] "range_atomic_mass"           "wtd_std_atomic_mass"
##  [7] "mean_fie"                    "wtd_mean_fie"
##  [9] "entropy_fie"                 "range_fie"
## [11] "wtd_std_fie"                 "gmean_atomic_radius"
## [13] "wtd_gmean_atomic_radius"     "entropy_atomic_radius"
## [15] "wtd_entropy_atomic_radius"   "range_atomic_radius"
## [17] "wtd_std_atomic_radius"       "wtd_gmean_Density"
## [19] "range_Density"               "wtd_std_Density"
## [21] "range_ElectronAffinity"      "wtd_mean_FusionHeat"
## [23] "gmean_FusionHeat"            "wtd_gmean_FusionHeat"
## [25] "entropy_FusionHeat"          "std_FusionHeat"
## [27] "wtd_std_FusionHeat"          "range_ThermalConductivity"
## [29] "wtd_std_ThermalConductivity" "mean_Valence"
## [31] "wtd_mean_Valence"            "wtd_gmean_Valence"
## [33] "entropy_Valence"             "wtd_entropy_Valence"
## [35] "range_Valence"
```

We are going to remove these columns from our training dataset, those columns have $>.9$ correlation

```
train_transformed_no_correlated = train_transformed[,-c(correlated_columns)]
label_index_noCorr <- which(colnames(train_transformed_no_correlated) == "critical_temp")
fit_lm_no_correlated <- train(critical_temp ~ ., data = train_transformed_no_correlated, method = "lm",
```

Performing Linear Regression with 10 fold cross-validation

```
print(fit_lm_no_correlated)
```

```
## Linear Regression
##
## 17010 samples
##   123 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 15310, 15310, 15309, 15309, 15309, 15309, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.5921099  0.6957537  0.379391
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

We see, that our rsquared on training dataset dropped to 70% and MAE increased to 0.38.

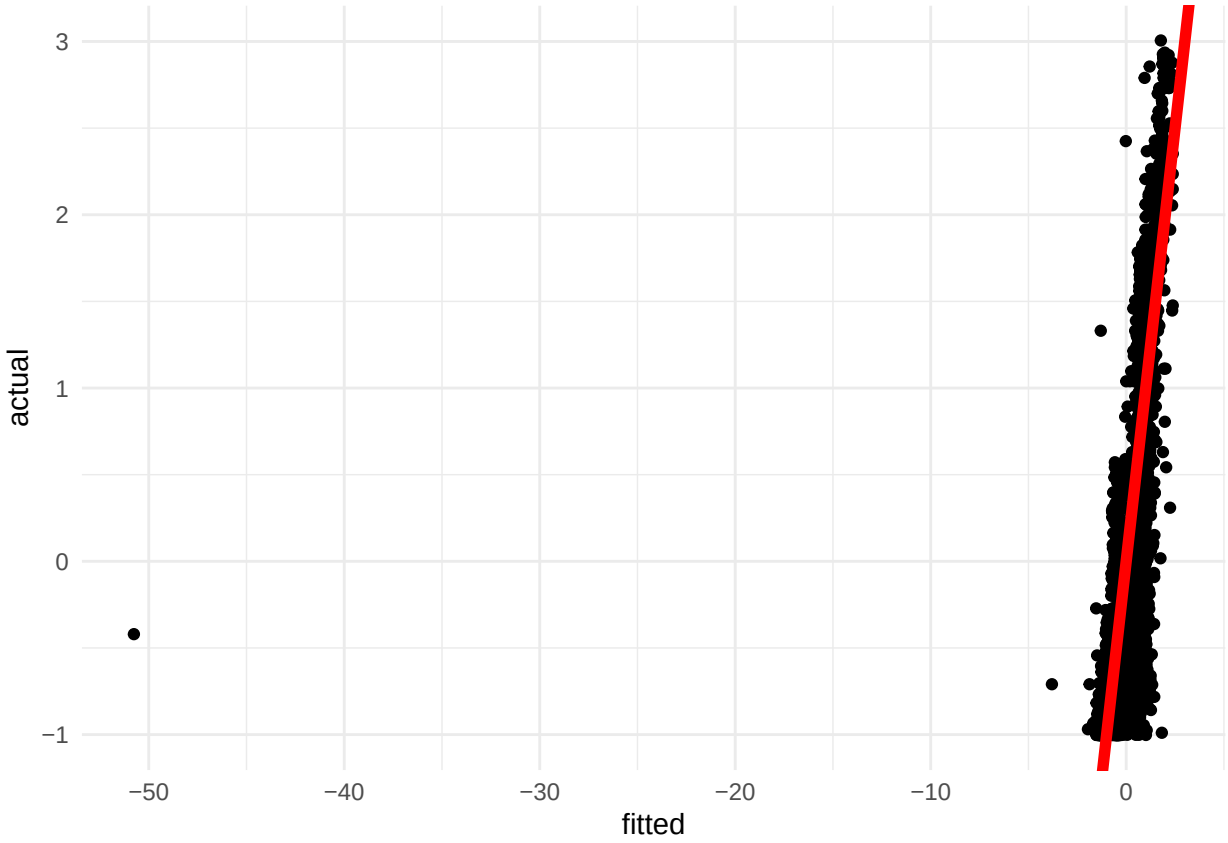
Testing on a test dataset

```
test_lm_noCorr <- predict(fit_lm_no_correlated, newdata = test_transformed[, -c(correlated_columns, lab
round(postResample(pred = test_lm_noCorr, obs = test_transformed$critical_temp), 3)
```

```
##      RMSE Rsquared      MAE
##     0.925    0.416    0.391
```

Rsquared on the testing dataset is 42% while MAE is 0.39 and RMSE = 0.93, let's look at the plot

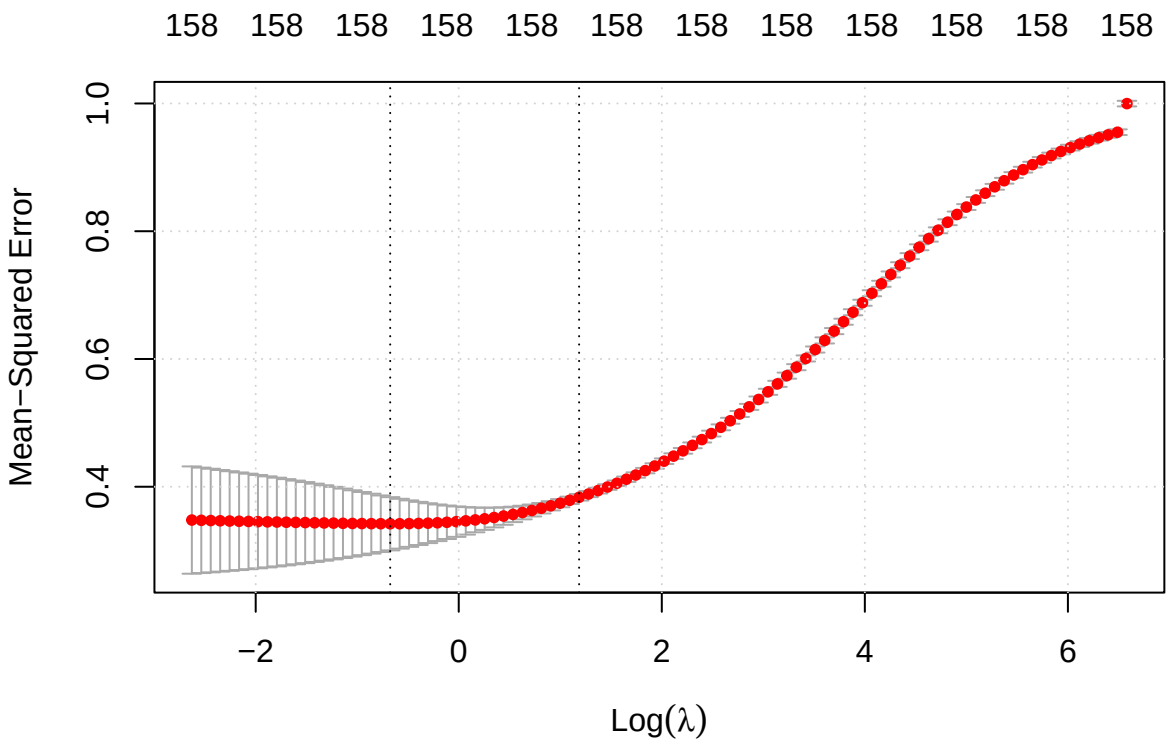
```
fitted_actual_lm_noCorr <- data.frame(fitted = test_lm_noCorr, actual = test_transformed$critical_temp)

ggplot(fitted_actual_lm_noCorr,
       aes(x = fitted,
           y = actual)) +
  geom_point() +
  geom_abline(intercept = 0,
              slope = 1,
              color = "red",
              size = 2) +
  theme_minimal()
```



Same result with that one outlier.

1.5.3 Ridge Linear Regression

```
fit_lmridge_cv = cv.glmnet(as.matrix(train_transformed[, -label_index]), train_transformed[, label_inde
plot(fit_lmridge_cv)
grid()
```



Best minimum lambda is:

```
fit_lmridge_cv$lambda.min
```

```
## [1] 0.5091288
```

Let's fit a Ridge Linear Regression using best minimum lambda:

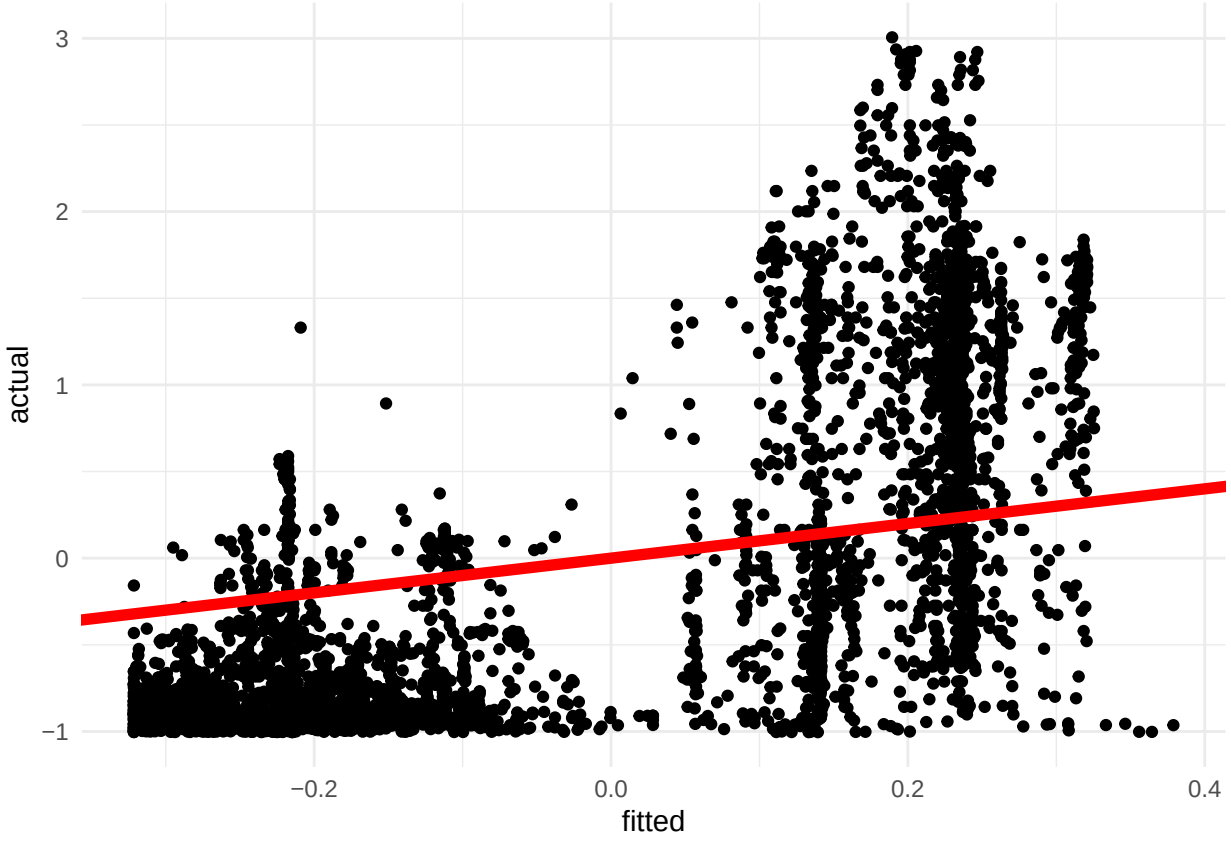
```
fit_lmridge = glmnet(as.matrix(train_transformed[, -label_index]), train_transformed[, label_index], la
test_lmridge <- predict(fit_lmridge, as.matrix(test_transformed[, -label_index]))
round(postResample(pred = test_lmridge, obs = test_transformed$critical_temp), 3)
```

```
##     RMSE Rsquared      MAE
##    0.854    0.523    0.721
```

Using Ridge Linear Regression we obtained 54% Rsquared, 0.67 MAE and 0.8 RMSE on testing dataset.

Let's plot fitted vs actual values:

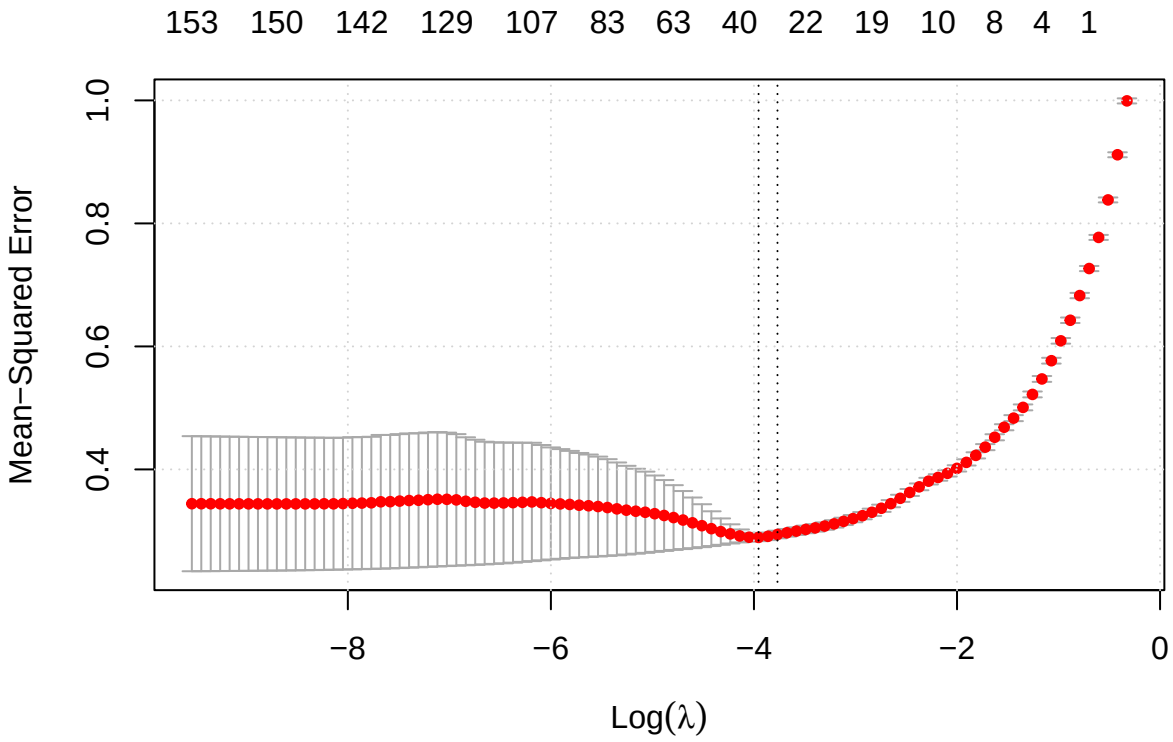
```
fitted_actual_lmridge <- data.frame(fitted = test_lmridge, actual = test_transformed$critical_temp) %>%
  rename(fitted = s0)

ggplot(fitted_actual_lmridge,
       aes(x = fitted,
           y = actual)) +
  geom_point() +
  geom_abline(intercept = 0,
              slope = 1,
              color = "red",
              size = 2) +
  theme_minimal()
```



While our model is better at predicting than a random guess, but the model is not that great when you look at the graph.

1.5.4 Lasso Linear Regression

```
fit_lasso_cv = cv.glmnet(as.matrix(train_transformed[, -label_index]), train_transformed[, label_index]
plot(fit_lasso_cv)
grid()
```



Best minimum lambda is:

```
fit_lasso_cv$lambda.min
```

```
## [1] 0.01916837
```

Let's fit a Lasso Linear Regression using best minimum lambda:

```
fit_lasso = glmnet(as.matrix(train_transformed[, -label_index]), train_transformed[, label_index], lambd
test_lasso <- predict(fit_lasso, as.matrix(test_transformed[, -label_index]))
round(postResample(pred = test_lasso, obs = test_transformed$critical_temp), 3)
```

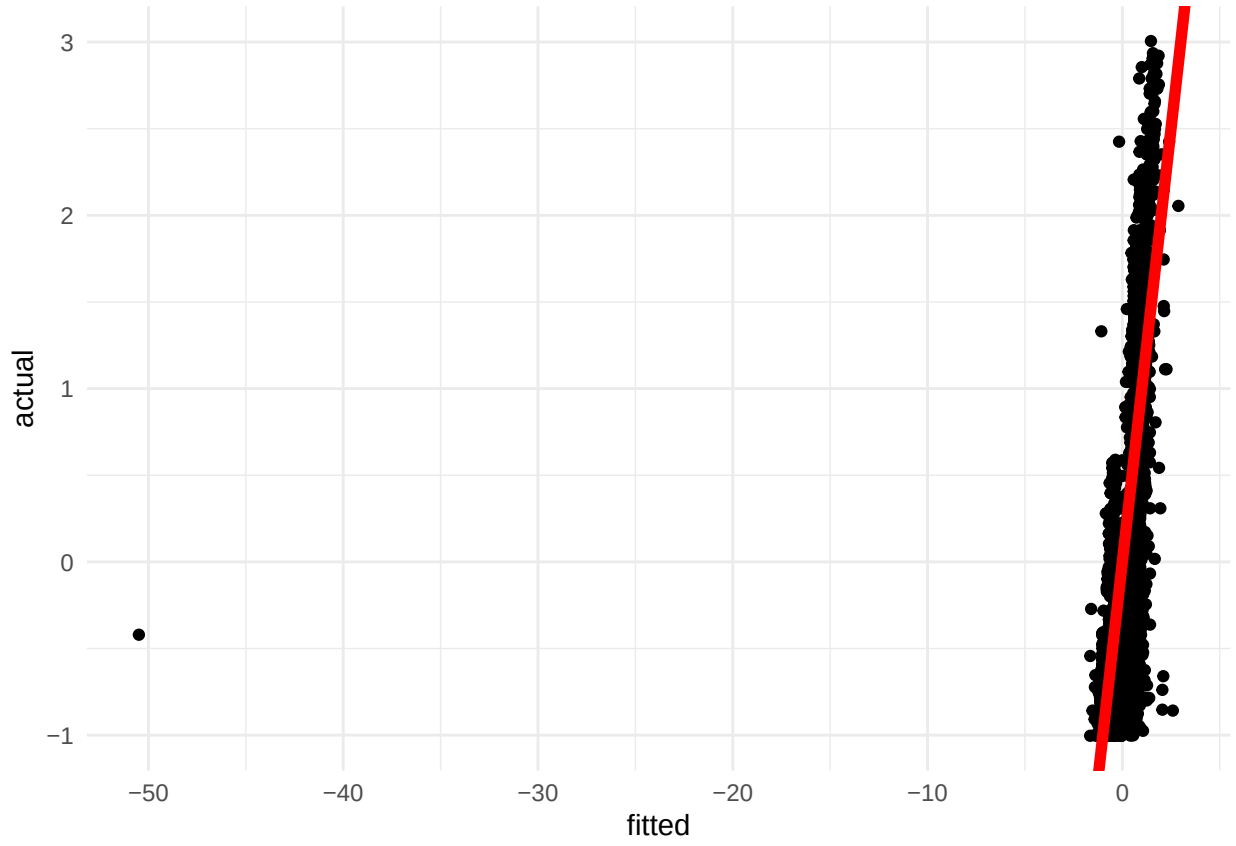
```
##     RMSE Rsquared      MAE
##    0.939    0.371    0.418
```

Using Lasso Linear Regression we obtained 37% Rsquared, 0.41 MAE and 0.94 RMSE on testing dataset.

Let's plot fitted vs actual values:

```
fitted_actual_lasso <- data.frame(fitted = test_lasso, actual = test_transformed$critical_temp) %>%
  rename(fitted = s0)

ggplot(fitted_actual_lasso,
       aes(x = fitted,
           y = actual)) +
```

```
  geom_point() +
  geom_abline(intercept = 0,
              slope = 1,
              color = "red",
              size = 2) +
  theme_minimal()
```



We see the same predicted outlier

1.5.5 Elastic Net Regression

```
fit_elasticNet <- train(critical_temp ~ ., data = train_transformed, method = "glmnet", trControl = trai

print(fit_elasticNet)
```

```
## glmnet
##
## 17010 samples
##   158 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 15309, 15308, 15310, 15309, 15308, 15309, ...
```

```
## Resampling results across tuning parameters:
##
##   alpha  lambda        RMSE       Rsquared   MAE
##   0.1    0.0003334334  0.5521809  0.7193294  0.3649076
##   0.1    0.0007702743  0.5522658  0.7192238  0.3649629
##   0.1    0.0017794336  0.5545260  0.7169534  0.3661600
##   0.1    0.0041107225  0.5590509  0.7127377  0.3685153
##   0.1    0.0094963024  0.5597657  0.7090862  0.3719233
##   0.1    0.0219376907  0.5604106  0.7040964  0.3778729
##   0.1    0.0506789121  0.5592158  0.6997338  0.3863189
##   0.1    0.1170748628  0.5498485  0.7009994  0.4030787
##   0.1    0.2704581245  0.5607061  0.6927732  0.4273265
##   0.1    0.6247933616  0.6056380  0.6537335  0.4700689
##   0.2    0.0003334334  0.5519383  0.7198446  0.3646780
##   0.2    0.0007702743  0.5527009  0.7188167  0.3652199
##   0.2    0.0017794336  0.5563327  0.7155850  0.3668772
##   0.2    0.0041107225  0.5590409  0.7114180  0.3698067
##   0.2    0.0094963024  0.5603548  0.7065805  0.3747921
##   0.2    0.0219376907  0.5607752  0.7013897  0.3816803
##   0.2    0.0506789121  0.5515384  0.7012231  0.3950047
##   0.2    0.1170748628  0.5474042  0.7046765  0.4131781
##   0.2    0.2704581245  0.5831226  0.6737630  0.4477594
##   0.2    0.6247933616  0.6484289  0.6219819  0.5084315
##   0.3    0.0003334334  0.5513567  0.7200799  0.3646204
##   0.3    0.0007702743  0.5527408  0.7185293  0.3654303
##   0.3    0.0017794336  0.5575213  0.7143876  0.3676158
##   0.3    0.0041107225  0.5580076  0.7108241  0.3707994
##   0.3    0.0094963024  0.5612338  0.7041247  0.3774650
##   0.3    0.0219376907  0.5601104  0.6993409  0.3860775
##   0.3    0.0506789121  0.5424114  0.7078329  0.4006299
##   0.3    0.1170748628  0.5566857  0.6968250  0.4213747
##   0.3    0.2704581245  0.6077080  0.6510514  0.4694203
##   0.3    0.6247933616  0.6857001  0.6008747  0.5441989
##   0.4    0.0003334334  0.5516928  0.7198124  0.3647089
##   0.4    0.0007702743  0.5537761  0.7178002  0.3656745
##   0.4    0.0017794336  0.5585365  0.7132842  0.3682473
##   0.4    0.0041107225  0.5591652  0.7091728  0.3722974
##   0.4    0.0094963024  0.5620595  0.7024931  0.3792569
##   0.4    0.0219376907  0.5566752  0.6995428  0.3900958
##   0.4    0.0506789121  0.5405213  0.7103291  0.4055180
##   0.4    0.1170748628  0.5671035  0.6876283  0.4310280
##   0.4    0.2704581245  0.6273664  0.6345659  0.4871116
##   0.4    0.6247933616  0.7195125  0.5903604  0.5804922
##   0.5    0.0003334334  0.5514335  0.7198256  0.3648245
##   0.5    0.0007702743  0.5542251  0.7172830  0.3660783
##   0.5    0.0017794336  0.5582399  0.7127030  0.3688447
##   0.5    0.0041107225  0.5608447  0.7071781  0.3740072
##   0.5    0.0094963024  0.5621964  0.7012753  0.3810729
##   0.5    0.0219376907  0.5513977  0.7018831  0.3928875
##   0.5    0.0506789121  0.5453154  0.7059041  0.4100737
##   0.5    0.1170748628  0.5775074  0.6782821  0.4407695
##   0.5    0.2704581245  0.6420213  0.6256123  0.5006355
##   0.5    0.6247933616  0.7539155  0.5839453  0.6182037
##   0.6    0.0003334334  0.5510896  0.7199308  0.3648557
```

```
##   0.6    0.0007702743  0.5552037  0.7165790  0.3664450
##   0.6    0.0017794336  0.5575531  0.7124646  0.3692785
##   0.6    0.0041107225  0.5613511  0.7057024  0.3755873
##   0.6    0.0094963024  0.5615538  0.7005314  0.3828374
##   0.6    0.0219376907  0.5463272  0.7048921  0.3954241
##   0.6    0.0506789121  0.5495900  0.7020660  0.4138821
##   0.6    0.1170748628  0.5885958  0.6677457  0.4505049
##   0.6    0.2704581245  0.6564471  0.6173091  0.5142887
##   0.6    0.6247933616  0.7919268  0.5680272  0.6577118
##   0.7    0.0003334334  0.5515030  0.7196136  0.3649986
##   0.7    0.0007702743  0.5561384  0.7158043  0.3669028
##   0.7    0.0017794336  0.5575125  0.7119862  0.3698123
##   0.7    0.0041107225  0.5626984  0.7041787  0.3768379
##   0.7    0.0094963024  0.5609514  0.6998210  0.3846362
##   0.7    0.0219376907  0.5421341  0.7080582  0.3979136
##   0.7    0.0506789121  0.5542239  0.6977795  0.4180351
##   0.7    0.1170748628  0.6006550  0.6553029  0.4612744
##   0.7    0.2704581245  0.6721532  0.6066712  0.5293529
##   0.7    0.6247933616  0.8304477  0.5477207  0.6961553
##   0.8    0.0003334334  0.5514697  0.7194395  0.3651443
##   0.8    0.0007702743  0.5564705  0.7152770  0.3672868
##   0.8    0.0017794336  0.5577991  0.7113232  0.3704589
##   0.8    0.0041107225  0.5630112  0.7033940  0.3776723
##   0.8    0.0094963024  0.5599868  0.6994105  0.3863209
##   0.8    0.0219376907  0.5393873  0.7106559  0.4003750
##   0.8    0.0506789121  0.5585991  0.6937711  0.4218935
##   0.8    0.1170748628  0.6108651  0.6449047  0.4707964
##   0.8    0.2704581245  0.6875495  0.5960897  0.5444564
##   0.8    0.6247933616  0.8655400  0.5333669  0.7298585
##   0.9    0.0003334334  0.5517517  0.7191661  0.3653083
##   0.9    0.0007702743  0.5572359  0.7145975  0.3676006
##   0.9    0.0017794336  0.5587896  0.7103251  0.3711990
##   0.9    0.0041107225  0.5630367  0.7028598  0.3783587
##   0.9    0.0094963024  0.5582791  0.6995977  0.3879573
##   0.9    0.0219376907  0.5389630  0.7113308  0.4028289
##   0.9    0.0506789121  0.5631410  0.6895308  0.4259500
##   0.9    0.1170748628  0.6191725  0.6369462  0.4785580
##   0.9    0.2704581245  0.7003066  0.5898256  0.5584944
##   0.9    0.6247933616  0.8976460  0.5211803  0.7602418
##   1.0    0.0003334334  0.5519935  0.7189174  0.3654389
##   1.0    0.0007702743  0.5578343  0.7140175  0.3679050
##   1.0    0.0017794336  0.5598592  0.7092525  0.3720341
##   1.0    0.0041107225  0.5629040  0.7024430  0.3790233
##   1.0    0.0094963024  0.5562527  0.7002069  0.3892984
##   1.0    0.0219376907  0.5409921  0.7094397  0.4052670
##   1.0    0.0506789121  0.5676839  0.6852420  0.4301046
##   1.0    0.1170748628  0.6243707  0.6334658  0.4833076
##   1.0    0.2704581245  0.7140457  0.5809629  0.5736239
##   1.0    0.6247933616  0.9324051  0.5211803  0.7931457
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 0.9 and lambda = 0.02193769.
```

Best parameters:

```
fit_elasticNet$bestTune
```

```
##    alpha     lambda
## 86   0.9 0.02193769
```

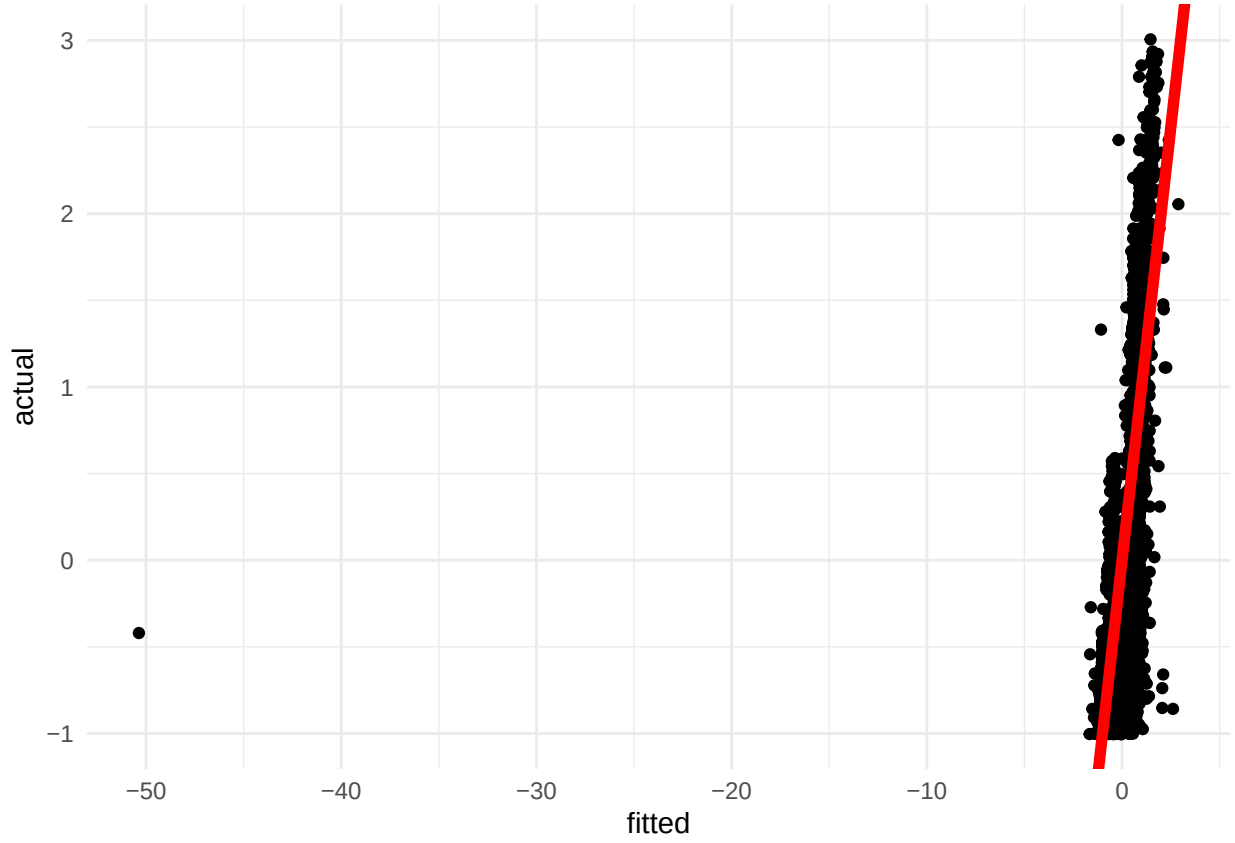
Testing on a test dataset

```
test_elasticNet <- predict(fit_elasticNet, newdata = test_transformed[, -label_index])
round(postResample(pred = test_elasticNet, obs = test_transformed$critical_temp), 3)
```

```
##     RMSE Rsquared      MAE
##    0.938    0.371    0.419
```

Rsquared on the testing dataset is 37% while MAE is 0.41 and RMSE = 0.95, let's look at the plot

```
fitted_actual_elasticNet <- data.frame(fitted = test_elasticNet, actual = test_transformed$critical_temp

ggplot(fitted_actual_elasticNet,
       aes(x = fitted,
           y = actual)) +
  geom_point() +
  geom_abline(intercept = 0,
              slope = 1,
              color = "red",
              size = 2) +
  theme_minimal()
```



1.6 Lab 1 Conclusion

```
models <- t(data.frame(
  `Linear Regression` = round(postResample(pred = test_lm, obs = test_transformed$critical_temp), 3),
  `Linear Regression Without Correlated Columns` = round(postResample(pred = test_lm_noCorr, obs = test
  `RIDGE` = round(postResample(pred = test_lmridge, obs = test_transformed$critical_temp), 3),
  `LASSO` = round(postResample(pred = test_lasso, obs = test_transformed$critical_temp), 3),
  `Elastic Net Regression` = round(postResample(pred = test_elasticNet, obs = test_transformed$critical
))

print(models)
```

```
##                                              RMSE Rsquared   MAE
## Linear.Regression                           1.180    0.306 0.380
## Linear.Regression.Without.Correlated.Columns 0.925    0.416 0.391
## RIDGE                                       0.854    0.523 0.721
## LASSO                                       0.939    0.371 0.418
## Elastic.Net.Regression                      0.938    0.371 0.419
```

The best result was achieved using RIDGE linear regression, while the MAE of Ridge is the highest. While the Ridge method gave the best result, but the model itself is not that great when you look at the graph, anyways, using Ridge method we did not predicted any outlier, unlike other methods.

---

2 Lab 2

2.1 Reading Data

```
set.seed(123)

data <- read.csv("water_potability.csv")
```

2.2 Required packages

```
library(SmartEDA)
library(dplyr)
library(ggplot2)
library(mice)
library(VIM)
library(glmnet)
library(caret)
library(ROCR)

# Function for binary class accuracy from confusion matrix
classAcc <- function(confusionMatrix) {
```

```
  class0 <- round(confusionMatrix$table[1, 1] / sum(confusionMatrix$table[, 1]) * 100, 1)
  class1 <- round(confusionMatrix$table[2, 2] / sum(confusionMatrix$table[, 2]) * 100, 1)
  acc <- c(class0, class1)
  names(acc) <- c("Acc: 0", "Acc: 1")
  return(acc)
}
```

2.3 EDA, first look at the dataset

```
ExpData(data, type = 1)
```

```
##                                          Descriptions   Value
## 1                                  Sample size (nrow)    3276
## 2                             No. of variables (ncol)      10
## 3                     No. of numeric/interger variables      10
## 4                               No. of factor variables       0
## 5                                 No. of text variables       0
## 6                              No. of logical variables       0
## 7                           No. of identifier variables       6
## 8                                 No. of date variables       0
## 9              No. of zero variance variables (uniform)       0
## 10                %. of variables having complete cases 70% (7)
## 11   %. of variables having >0% and <50% missing cases 30% (3)
## 12 %. of variables having >=50% and <90% missing cases  0% (0)
## 13          %. of variables having >=90% missing cases  0% (0)
```

We can see, that we have 10 numerical variables and 3276 observations, we can also notice, that 3 variables have missing variables.

```
summary(data)
```

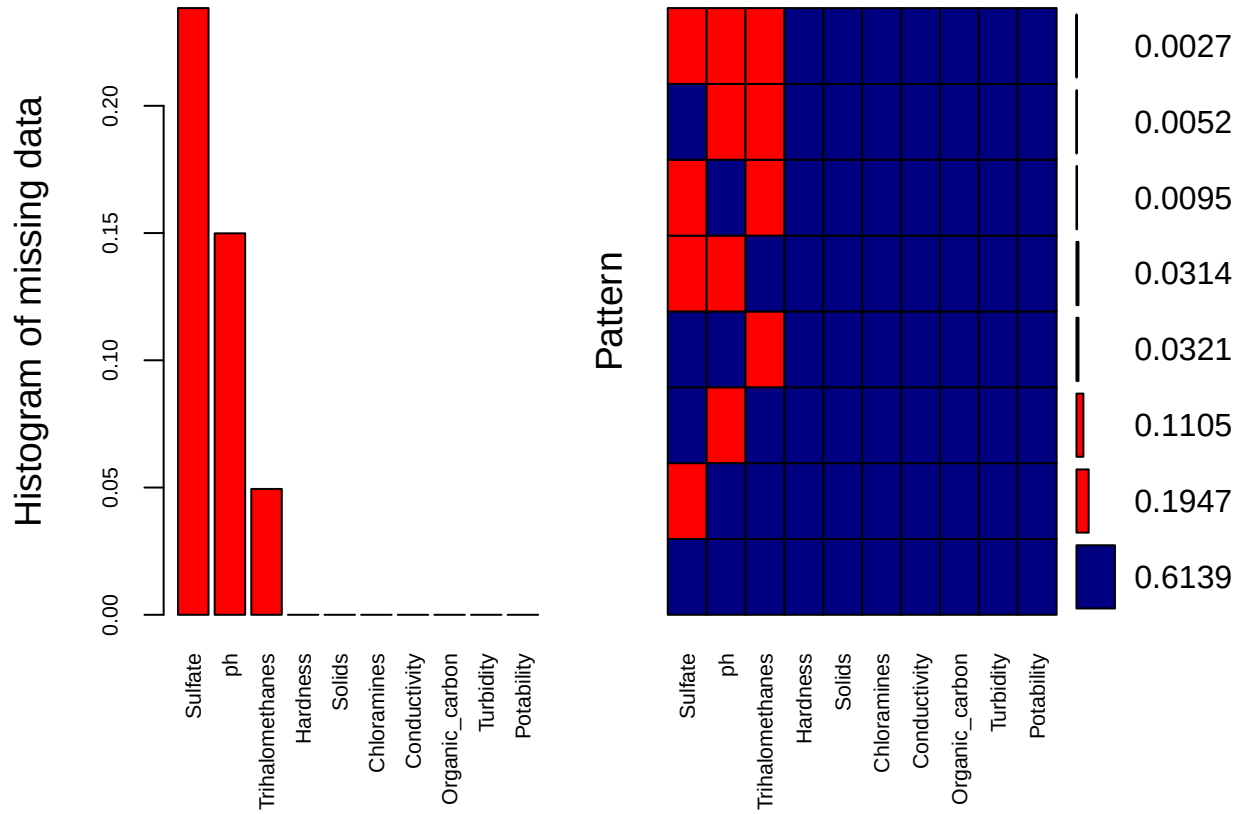
```
##        ph            Hardness          Solids         Chloramines
##  Min.   : 0.000   Min.   : 47.43   Min.   :  320.9   Min.   : 0.352
##  1st Qu.: 6.093   1st Qu.:176.85   1st Qu.:15666.7   1st Qu.: 6.127
##  Median : 7.037   Median :196.97   Median :20927.8   Median : 7.130
##  Mean   : 7.081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

Variables “ph”, “Sulfate” and “Trihalomethanes” have missing values.

Let’s look how our missing values are distributed:

```
aggr(data, col = c("navyblue", "red"), numbers = TRUE, sortVars = TRUE, labels = names(data), cex.axis =
```




```
##
##  Variables sorted by number of missings:
##         Variable      Count
##          Sulfate 0.23840049
##               ph 0.14987790
##  Trihalomethanes 0.04945055
##         Hardness 0.00000000
##           Solids 0.00000000
##      Chloramines 0.00000000
##     Conductivity 0.00000000
##   Organic_carbon 0.00000000
##        Turbidity 0.00000000
##       Potability 0.00000000
```

Out of 3276 observations we have 61% of observations without any missing value. 19% of observations have only one missing variable. Variable "ph" contains almost 15% of missing values, while "Sulfate" has almost 24% of missing values, this may cause problems. "Trihalomethanes" has less than <5% of missing values.

We are going to use "mice" package for missing values imputation.

```
imp <- mice(data)
```

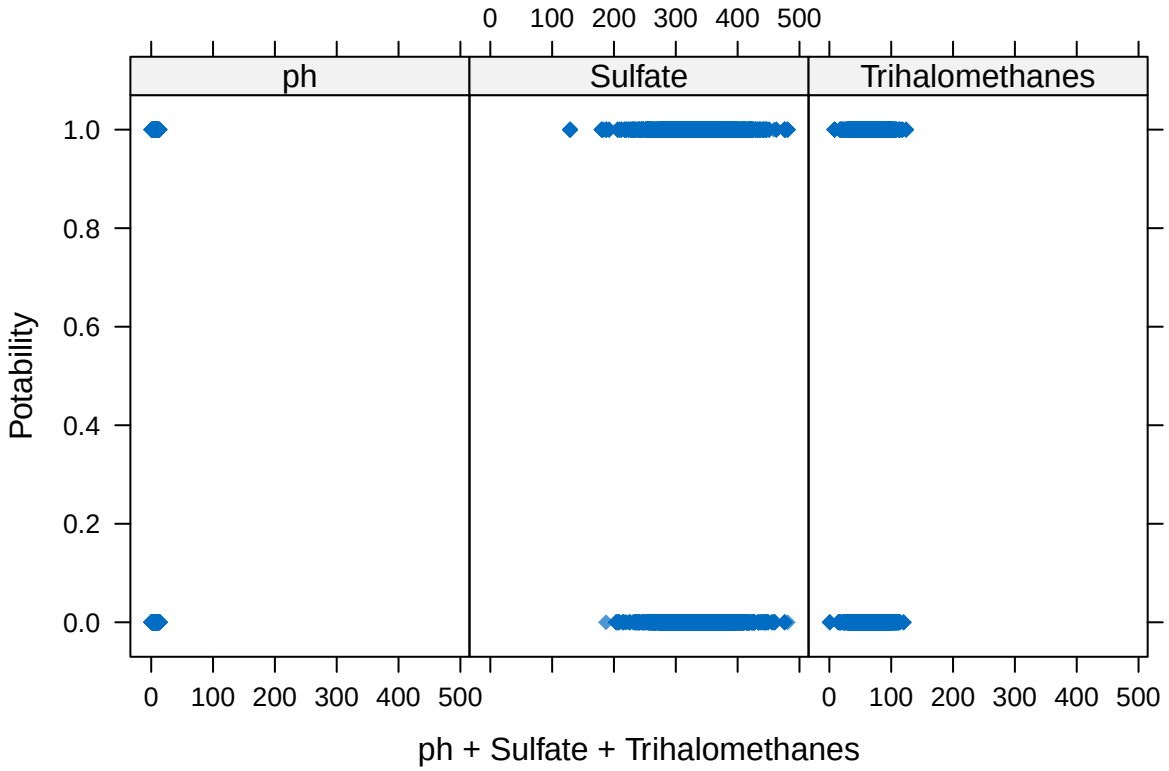
```
##
##  iter imp variable
##   1   1  ph  Sulfate  Trihalomethanes
##   1   2  ph  Sulfate  Trihalomethanes
##   1   3  ph  Sulfate  Trihalomethanes
##   1   4  ph  Sulfate  Trihalomethanes
##   1   5  ph  Sulfate  Trihalomethanes
##   2   1  ph  Sulfate  Trihalomethanes
##   2   2  ph  Sulfate  Trihalomethanes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

```
summary(imp)
```

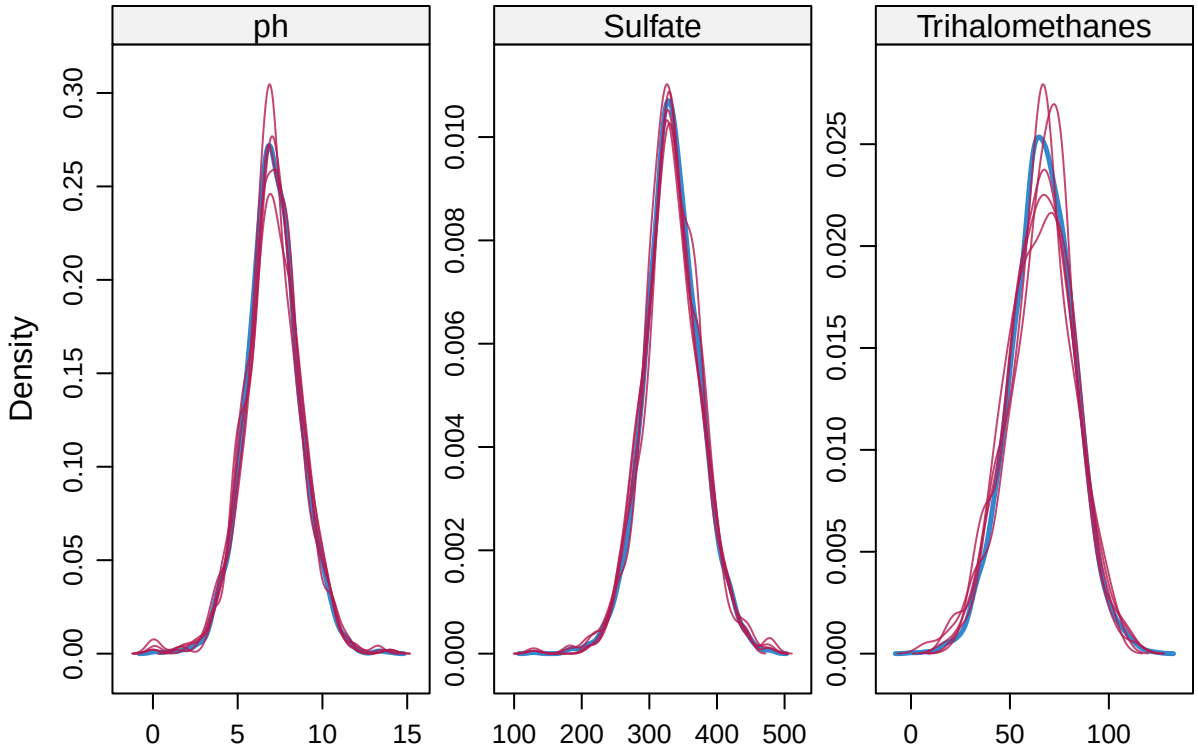
```
## Class: mids
## Number of multiple imputations:  5
## Imputation methods:
##              ph        Hardness          Solids     Chloramines         Sulfate
##           "pmm"              ""              ""              ""           "pmm"
##    Conductivity  Organic_carbon Trihalomethanes       Turbidity      Potability
##              ""              ""           "pmm"              ""              ""
## PredictorMatrix:
##              ph Hardness Solids Chloramines Sulfate Conductivity Organic_carbon
## ph            0        1      1           1       1            1              1
## Hardness      1        0      1           1       1            1              1
## Solids        1        1      0           1       1            1              1
## Chloramines   1        1      1           0       1            1              1
## Sulfate       1        1      1           1       0            1              1
## Conductivity  1        1      1           1       1            0              1
```

```
##              Trihalomethanes Turbidity Potability
## ph                         1         1          1
## Hardness                   1         1          1
## Solids                     1         1          1
## Chloramines                1         1          1
## Sulfate                    1         1          1
## Conductivity               1         1          1
```

```
data_noMissing <- complete(imp, 1)

xyplot(imp, Potability ~ ph + Sulfate + Trihalomethanes, pch = 18, cex = 1)
```




```
densityplot(imp)
```



The scatter plot of imputed data (red) and observed values (blue) shows that we did not produced any outliers. Density plots also shows no bad variation of imputed data.

2.4 Pre-process

2.4.1 Data split

```
indices <- createDataPartition(data_noMissing$Potability, p = 0.8, list = FALSE)
train <- data_noMissing[indices, ]
test <- data_noMissing[-indices, ]

label_index <- which(colnames(train) == "Potability")
```

Splitted a dataset by 80% / 20% rule. Created a training dataset with 2621 (80.006105%) observations and testing dataset with 655 (19.993895%) observations.

2.4.2 Data transformation

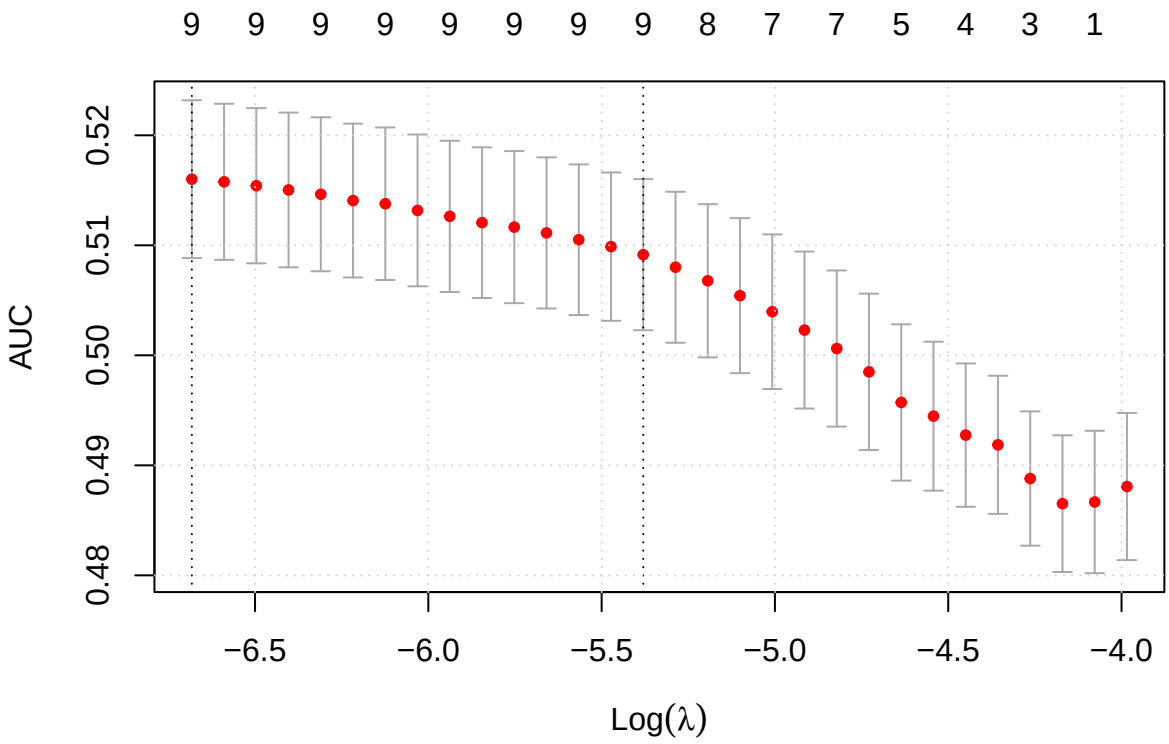
```
preProcValues <- preProcess(train[, -label_index], method = c("center", "scale"))
train[, -label_index] <- predict(preProcValues, train[, -label_index])
test[, -label_index] <- predict(preProcValues, test[, -label_index])
```

We also centered and scaled our data. We transformed the testing dataset based on the pre process of training dataset.

2.5 Logistic Regression

2.5.1 Simple Logistic Regression

```
fit_glm_cv <- cv.glmnet(as.matrix(train[, -label_index]), train[, label_index], family = "binomial", ty
plot(fit_glm_cv)
grid()
```



Best lambda:

```
fit_glm_cv$lambda.min
```

```
## [1] 0.001252873
```

At first glance, our model on the training set does not perform well. Let's take the best result and run a logistic regression again

Using fitted logistic regression model on the testing dataset:

```r
fit_glm <- glmnet(as.matrix(train[, -label_index]), train[, label_index], family = "binomial", lambda =
test_glm <- predict(fit_glm, as.matrix(test[, -label_index]), type = "class")
confusionMatrix <- confusionMatrix(as.factor(test_glm), as.factor(test$Potability), mode = "everything"
print(confusionMatrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 411 243
##          1   1   0
##
##                Accuracy : 0.6275
##                  95% CI : (0.5892, 0.6646)
##     No Information Rate : 0.629
##     P-Value [Acc > NIR] : 0.5496
##
##                   Kappa : -0.0031
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9976
##             Specificity : 0.0000
##          Pos Pred Value : 0.6284
##          Neg Pred Value : 0.0000
##               Precision : 0.6284
##                  Recall : 0.9976
##                      F1 : 0.7711
##              Prevalence : 0.6290
##          Detection Rate : 0.6275
##    Detection Prevalence : 0.9985
##       Balanced Accuracy : 0.4988
##
##        'Positive' Class : 0
##
```

```r
print(classAcc(confusionMatrix))
```

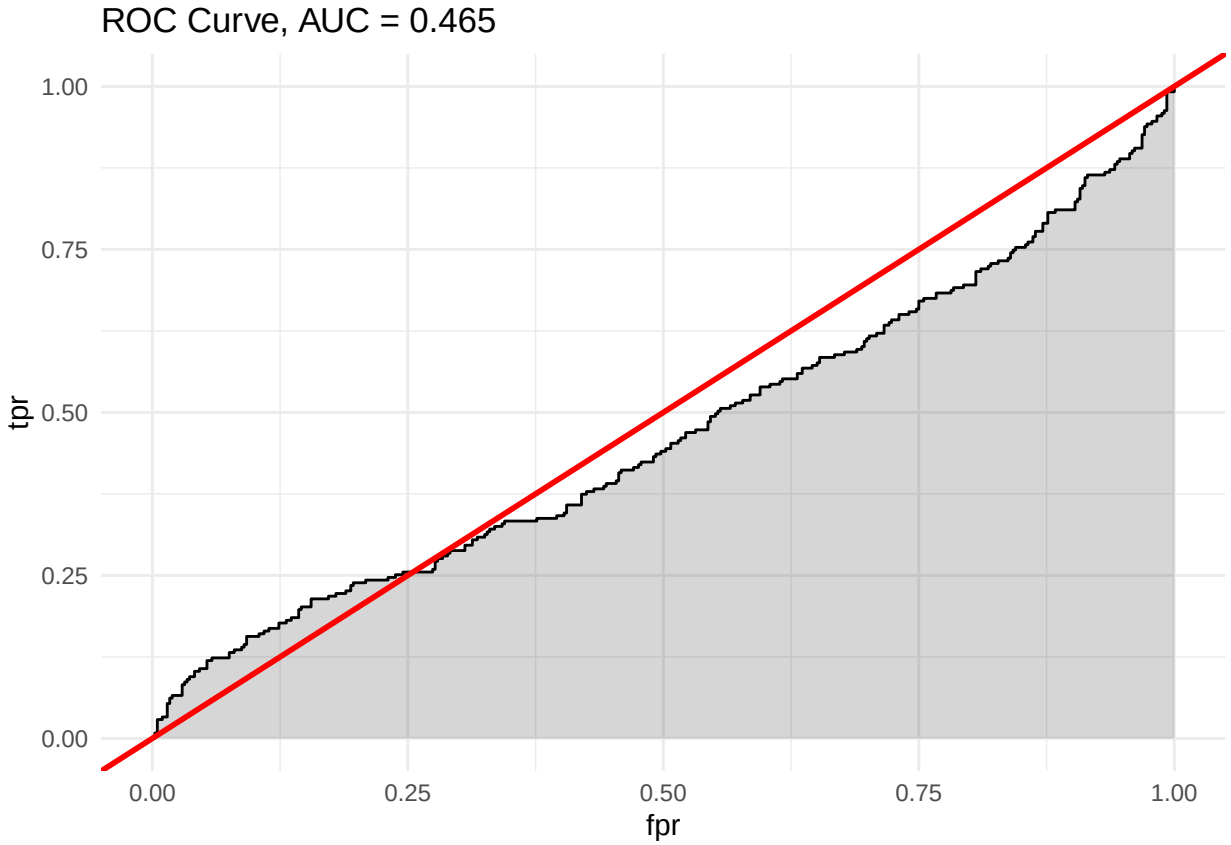
```
## Acc: 0 Acc: 1
##   99.8    0.0
```

The overall accuracy is 63%, but the No Information Rate is 0.63. We can see from the confusion table and from the NIR, that our model poorly predicts one class. Class "1" has 0% accuracy

Let's check the ROC curves:

```r
prob_glm <- predict(fit_glm, as.matrix(test[, -label_index]), type = "response", s = fit_glm$lambda)
pred_glm <- prediction(prob_glm, test[, label_index])
perf_glm <- performance(pred_glm, measure = "tpr", x.measure = "fpr")

# area under the curve
auc <- attr(performance(pred_glm, "auc"), "y.values")[[1]]
```

```r
auc <- round(auc, digits = 3)

roc_glm <- data.frame(fpr = attr(perf_glm, "x.values")[[1]], tpr = attr(perf_glm, "y.values")[[1]])

ggplot(roc_glm, aes(x = fpr, ymin = 0, ymax = tpr)) +
  geom_ribbon(alpha = 0.2) +
  geom_line(aes(y = tpr)) +
  ggtitle(paste0("ROC Curve, AUC = ", auc)) +
  geom_abline(slope = 1, color = "red", size = 1) +
  theme_minimal()
```



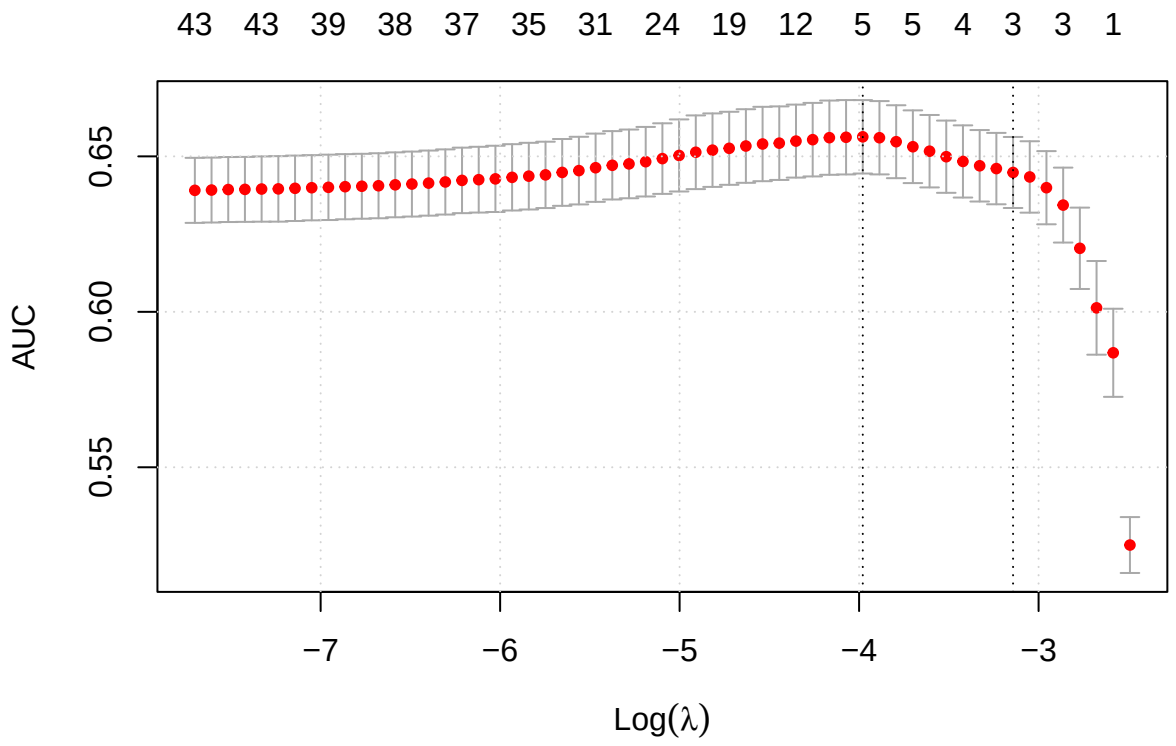
The AUC is 0.465, so our model is even worse than a random guess.

2.5.2 Logistic Regression with interactions

Let's try to add interactions between variables to see if that could lead to a better prediction.

```r
formula <- as.formula(" ~ .^2")

# We add interactions of our primary variables
train_int <- model.matrix(formula, data = train[, -label_index])
train_int <- train_int[, 2:ncol(train_int)] %>%
  bind_cols(Potability = train$Potability) %>%
  as.matrix()
train_int_label_index <- which(colnames(train_int) == "Potability")
```

```
test_int <- model.matrix(formula, data = test[, -label_index])
test_int <- test_int[, 2:ncol(test_int)] %>%
  bind_cols(Potability = test$Potability) %>%
  as.matrix()

fit_glm_cv_int <- cv.glmnet(train_int[, -train_int_label_index], train_int[, train_int_label_index], fa
plot(fit_glm_cv_int)
grid()
```



Best lambda:

```
fit_glm_cv_int$lambda.min
```

```
## [1] 0.01869524
```

The addition of interactions seems to give a better result, let's try the fitted model on a test dataset:

```
fit_glm_int <- glmnet(train_int[, -train_int_label_index], train_int[, train_int_label_index], family =
test_glm_int <- predict(fit_glm_int, test_int[, -train_int_label_index], type = "class")
confusionMatrix <- confusionMatrix(as.factor(test_glm_int), as.factor(test_int[, train_int_label_index])
print(confusionMatrix)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction   0   1
##          0 400 199
##          1  12  44
##
##                Accuracy : 0.6779
##                  95% CI : (0.6406, 0.7135)
##     No Information Rate : 0.629
##     P-Value [Acc > NIR] : 0.005094
##
##                   Kappa : 0.1804
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9709
##             Specificity : 0.1811
##          Pos Pred Value : 0.6678
##          Neg Pred Value : 0.7857
##               Precision : 0.6678
##                  Recall : 0.9709
##                      F1 : 0.7913
##              Prevalence : 0.6290
##          Detection Rate : 0.6107
##    Detection Prevalence : 0.9145
##       Balanced Accuracy : 0.5760
##
##        'Positive' Class : 0
##
```

```
print(classAcc(confusionMatrix))
```

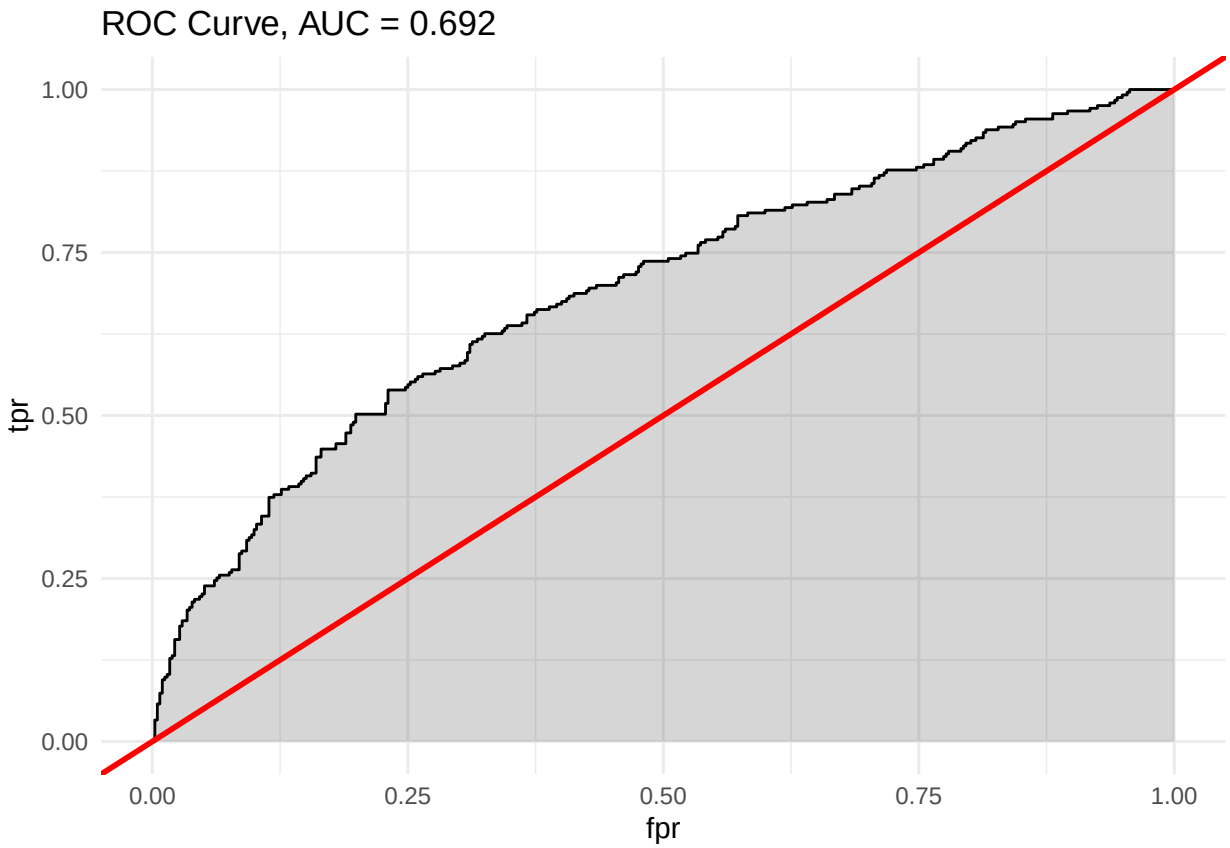
```
## Acc: 0 Acc: 1
##   97.1   18.1
```

The overall accuracy is 68%, but now, the addition of interactions between variables led to 18% accuracy of the "1" class, while the accuracy for class "0" only dropped to 97%.

Let's check the ROC curves:

```
prob_glm_int <- predict(fit_glm_int, test_int[, -train_int_label_index], type = "response", s = fit_glm
pred_glm_int <- prediction(prob_glm_int, test_int[, train_int_label_index])
perf_glm_int <- performance(pred_glm_int, measure = "tpr", x.measure = "fpr")

# area under the curve
auc <- attr(performance(pred_glm_int, "auc"), "y.values")[[1]]
auc <- round(auc, digits = 3)

roc_glm_int <- data.frame(fpr = attr(perf_glm_int, "x.values")[[1]], tpr = attr(perf_glm_int, "y.values

ggplot(roc_glm_int, aes(x = fpr, ymin = 0, ymax = tpr)) +
  geom_ribbon(alpha = 0.2) +
  geom_line(aes(y = tpr)) +
  ggtitle(paste0("ROC Curve, AUC = ", auc)) +
```

```
  geom_abline(slope = 1, color = "red", size = 1) +
  theme_minimal()
```



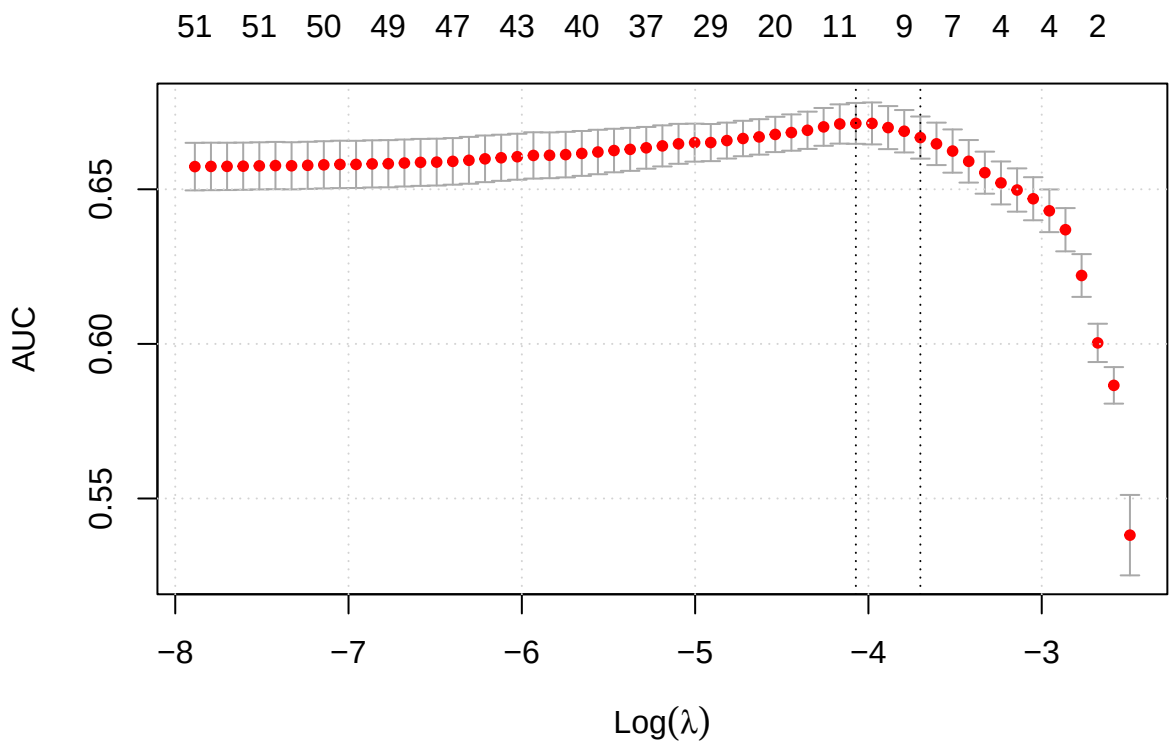
This time, the AUC is 0.692 and our model looks better than a random guess.

2.5.3 Logistic Regression with interactions and 2nd order polynomials

Interactions seemed to help in prediction, now let's try to add 2nd order polynomials of variables.

```
formula <- as.formula(paste(" ~ .^2 + ", paste("poly(", colnames(train[, -label_index]), ",2, raw=TRUE)

# We add interactions of our primary variables
train_int_pol2 <- model.matrix(formula, data = train[, -label_index])
train_int_pol2 <- train_int_pol2[, 2:ncol(train_int_pol2)] %>%
  bind_cols(Potability = train$Potability) %>%
  as.matrix()
train_int_pol2_label_index <- which(colnames(train_int_pol2) == "Potability")

test_int_pol2 <- model.matrix(formula, data = test[, -label_index])
test_int_pol2 <- test_int_pol2[, 2:ncol(test_int_pol2)] %>%
  bind_cols(Potability = test$Potability) %>%
  as.matrix()

fit_glm_cv_int_pol2 <- cv.glmnet(train_int_pol2[, -train_int_pol2_label_index], train_int_pol2[, train_
```

```
plot(fit_glm_cv_int_pol2)
grid()
```



Best lambda:

```
fit_glm_cv_int_pol2$lambda.min
```

```
## [1] 0.01703441
```

The addition of interactions and 2nd order polynomials seems to give a slightly better result than just with interactions, let's try the fitted model on a test dataset:

```
fit_glm_int_pol2 <- glmnet(train_int_pol2[, -train_int_pol2_label_index], train_int_pol2[, train_int_pol
test_glm_int_pol2 <- predict(fit_glm_int_pol2, test_int_pol2[, -train_int_pol2_label_index], type = "cla
confusionMatrix <- confusionMatrix(as.factor(test_glm_int_pol2), as.factor(test_int_pol2[, train_int_pol
print(confusionMatrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 397 189
##          1  15  54
##
```

```
##                Accuracy : 0.6885
##                  95% CI : (0.6515, 0.7239)
##     No Information Rate : 0.629
##     P-Value [Acc > NIR] : 0.0008193
##
##                   Kappa : 0.2178
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9636
##             Specificity : 0.2222
##          Pos Pred Value : 0.6775
##          Neg Pred Value : 0.7826
##               Precision : 0.6775
##                  Recall : 0.9636
##                      F1 : 0.7956
##              Prevalence : 0.6290
##          Detection Rate : 0.6061
##    Detection Prevalence : 0.8947
##       Balanced Accuracy : 0.5929
##
##        'Positive' Class : 0
##
```

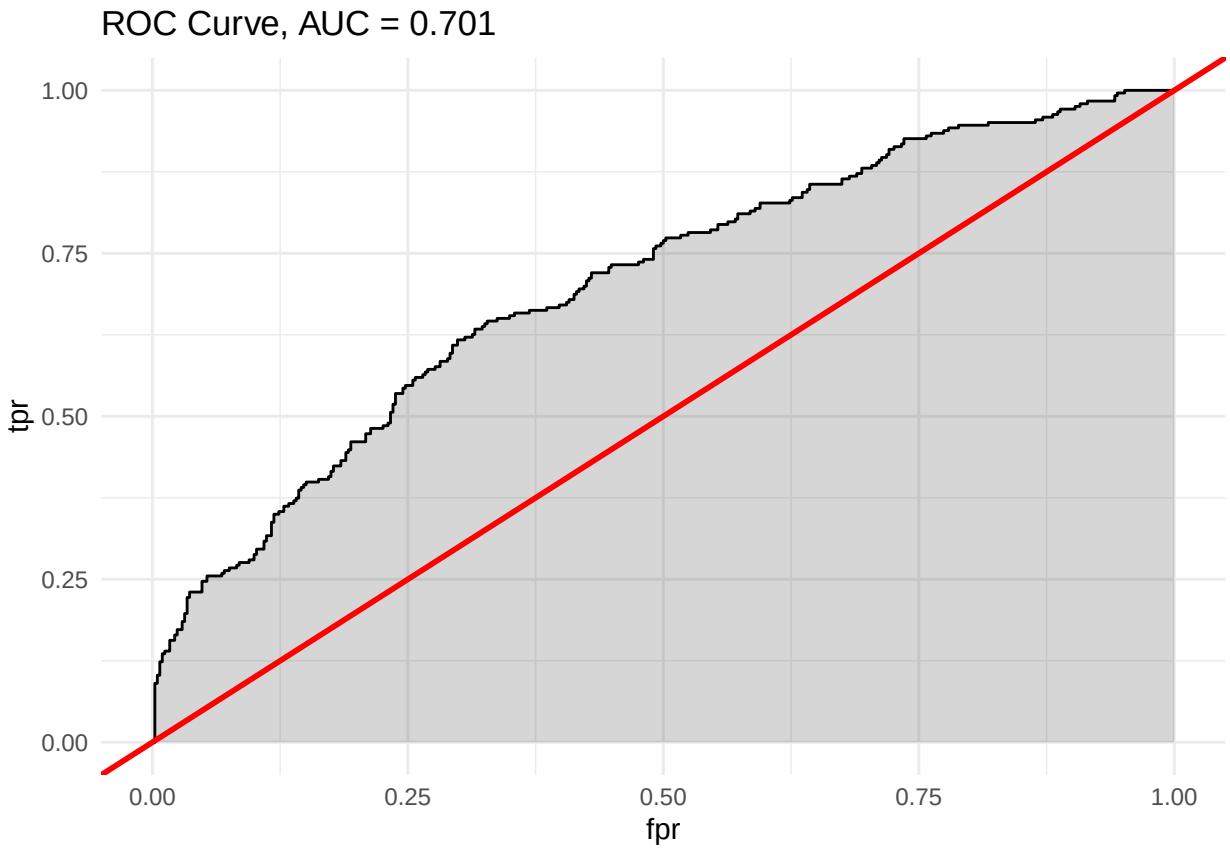
```
print(classAcc(confusionMatrix))
```

```
## Acc: 0 Acc: 1
##   96.4   22.2
```

The overall accuracy is 69%, but now, the addition of interactions between variables and 2nd order polynomials led to 21% accuracy of the "1" class, while the accuracy for class "0" only dropped to 96%.

Let's check the ROC curves:

```
prob_glm_int_pol2 <- predict(fit_glm_int_pol2, test_int_pol2[, -train_int_pol2_label_index], type = "res
pred_glm_int_pol2 <- prediction(prob_glm_int_pol2, test_int_pol2[, train_int_pol2_label_index])
perf_glm_int_pol2 <- performance(pred_glm_int_pol2, measure = "tpr", x.measure = "fpr")

# area under the curve
auc <- attr(performance(pred_glm_int_pol2, "auc"), "y.values")[[1]]
auc <- round(auc, digits = 3)

roc_glm_int_pol2 <- data.frame(fpr = attr(perf_glm_int_pol2, "x.values")[[1]], tpr = attr(perf_glm_int_p

ggplot(roc_glm_int_pol2, aes(x = fpr, ymin = 0, ymax = tpr)) +
  geom_ribbon(alpha = 0.2) +
  geom_line(aes(y = tpr)) +
  ggtitle(paste0("ROC Curve, AUC = ", auc)) +
  geom_abline(slope = 1, color = "red", size = 1) +
  theme_minimal()
```



This time, the AUC increased to 0.7. Overall, just a small increase after the interactions.

2.6 Lab 2 Conclusion

A logistic regression with interactions between variables and 2nd order polynomials seemed to give the best prediction. Addition of 2nd order polynomial just slightly increased the AUC and accuracy, so there's no need to try the next order.

Best lambda, obtained from the cross-validation, is 0.017. The overall accuracy on a test dataset is 69%, while the accurcy for "0" class is 96.4% and for class "1" is 22.2%.

The AUC is 0.701