Final_Project

2023-03-11

Team: Hengrui Yuan Joanne Teng Jiaheng Dai

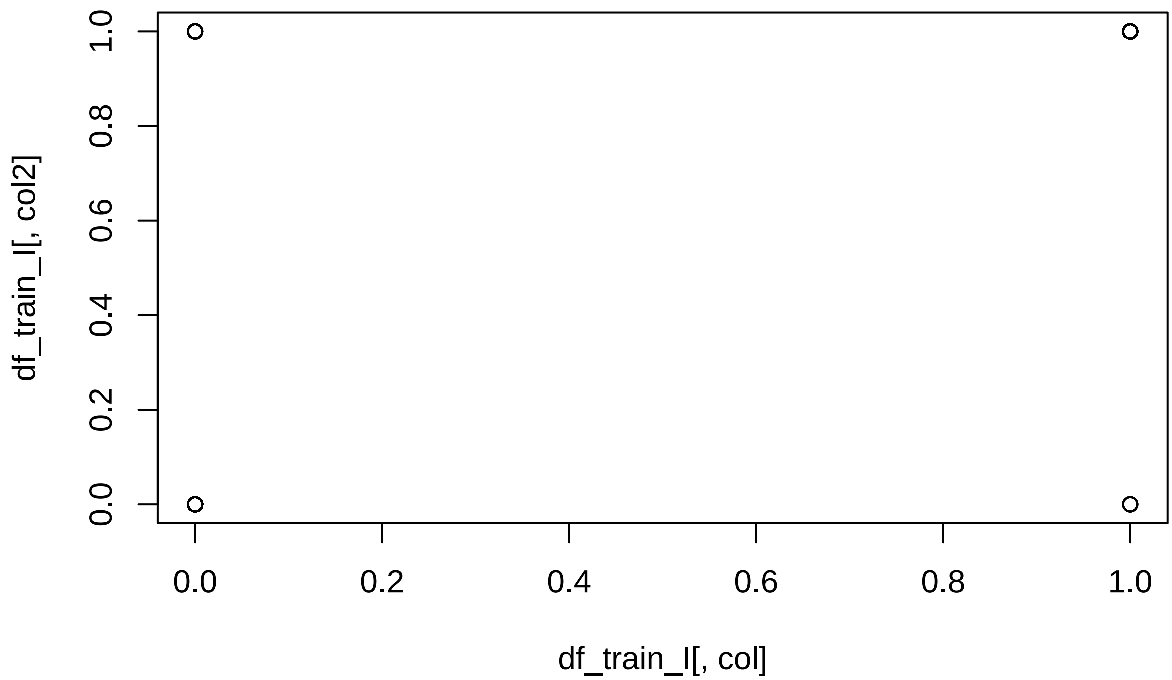
#1

```
# data processing
df_train <- read.csv("spam-train.txt")
df_test <- read.csv("spam-test.txt")
for (i in 1:57) {
names (df_train) [i]=paste("x",i,sep="")
names (df_test) [i]=paste("x",i,sep="")
}
names(df_train) [58]="Y"
names(df_test) [58]="Y"

# standardization
df_train_std <- df_train
df_train_std [,-58] <- as.data.frame (scale(df_train[,-58]))
df_test_std <- df_test
df_test_std [,-58] <- as.data.frame (scale(df_test [,-58]))

#log transform
df_train_log <- df_train
df_train_log [,-58] <- as.data.frame (log(df_train[,-58]+1))
df_test_log <- df_test
df_test_log [,-58] <- as.data.frame (log(df_test [,-58]+1))

#discretization transform
df_train_I <- as.data.frame (ifelse (df_train>0, 1,0))
df_test_I <- as.data.frame (ifelse (df_test>0, 1,0))
```

#a

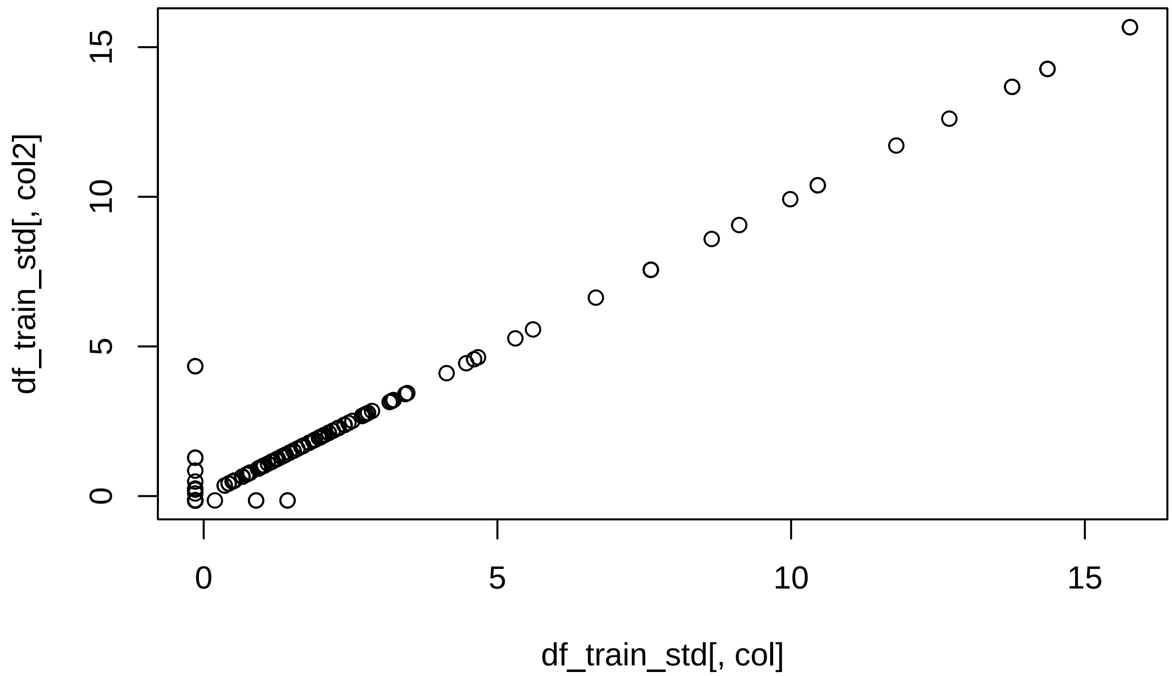
```
for (col in 1:ncol(df_train_I)) {
  for (col2 in col:ncol(df_train_I)) {
    if (col!=col2 &&
        !is.na(cor(df_train_I[,col], df_train_I[,col2]))
        && cor(df_train_I[,col], df_train_I[,col2]) > 0.8){
      cat("scatter plot between columns",col,"and",col2,'\n')
      plot(df_train_I[,col], df_train_I[,col2])
    }
  }
}
```

```
## scatter plot between columns 32 and 34
```

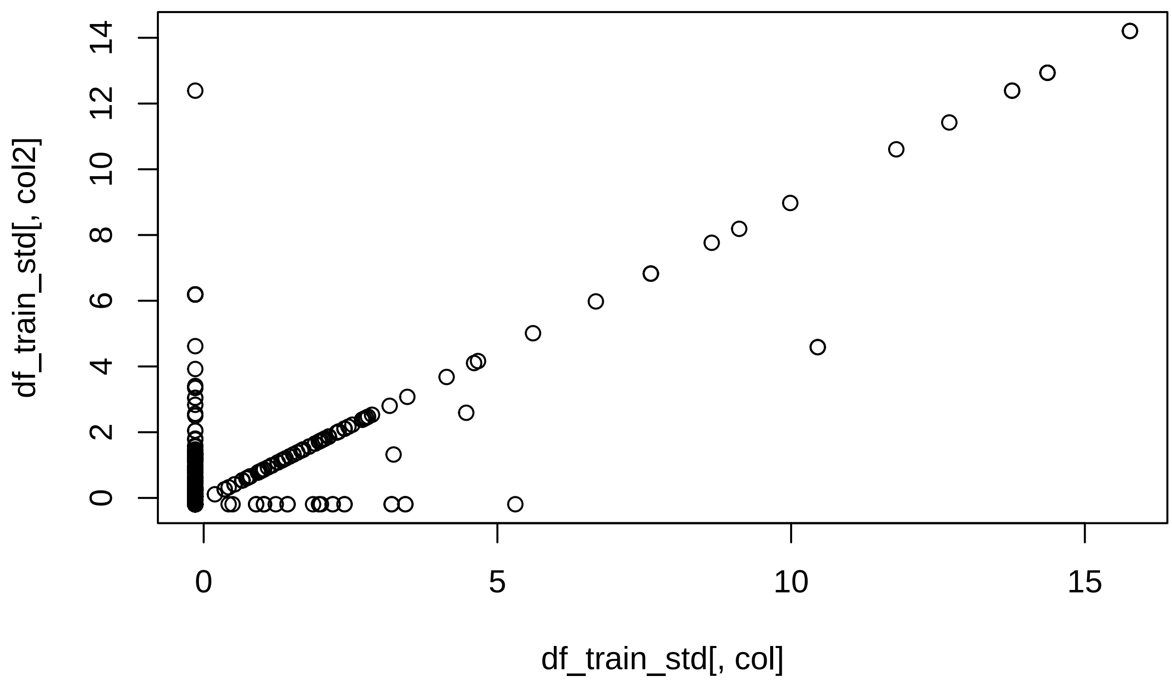



```r
for (col in 1:ncol(df_train_std)) {
  for (col2 in col:ncol(df_train_std)) {
    if (col!=col2 &&
        !is.na(cor(df_train_std[,col], df_train_std[,col2]))
        && cor(df_train_std[,col], df_train_std[,col2]) > 0.75){
      cat("scatter plot between columns",col,"and",col2,'\n')
      plot(df_train_std[,col], df_train_std[,col2])
    }

  }
}
```

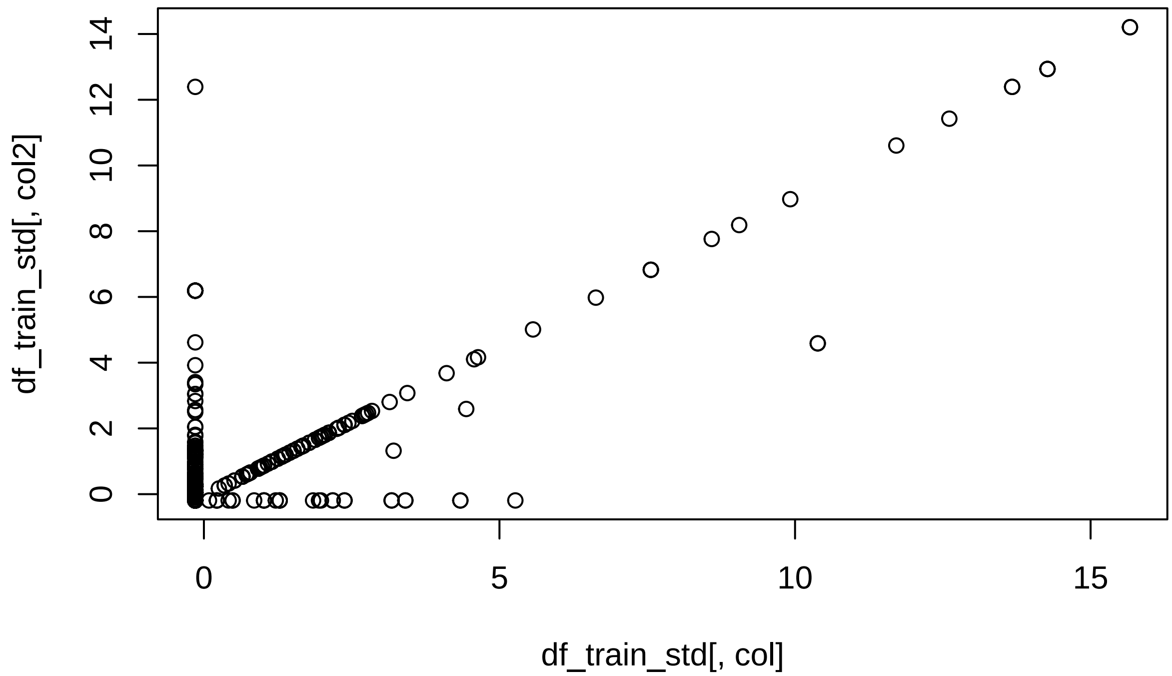
```
## scatter plot between columns 32 and 34
```




```
## scatter plot between columns 32 and 40
```

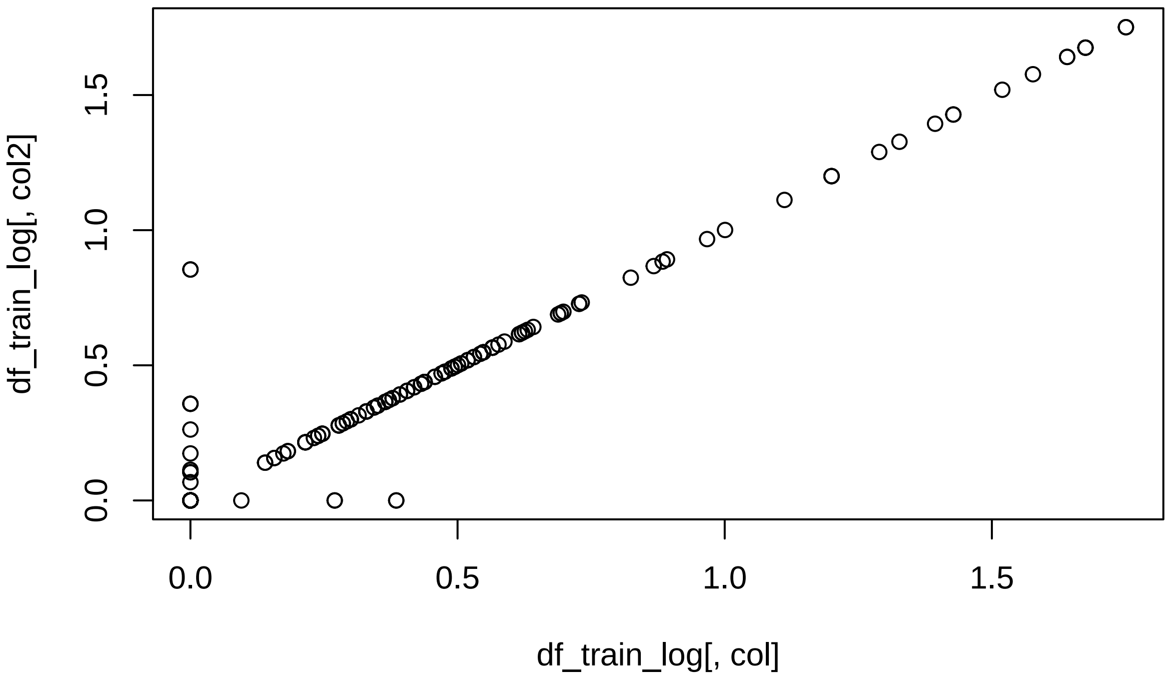



```
## scatter plot between columns 34 and 40
```

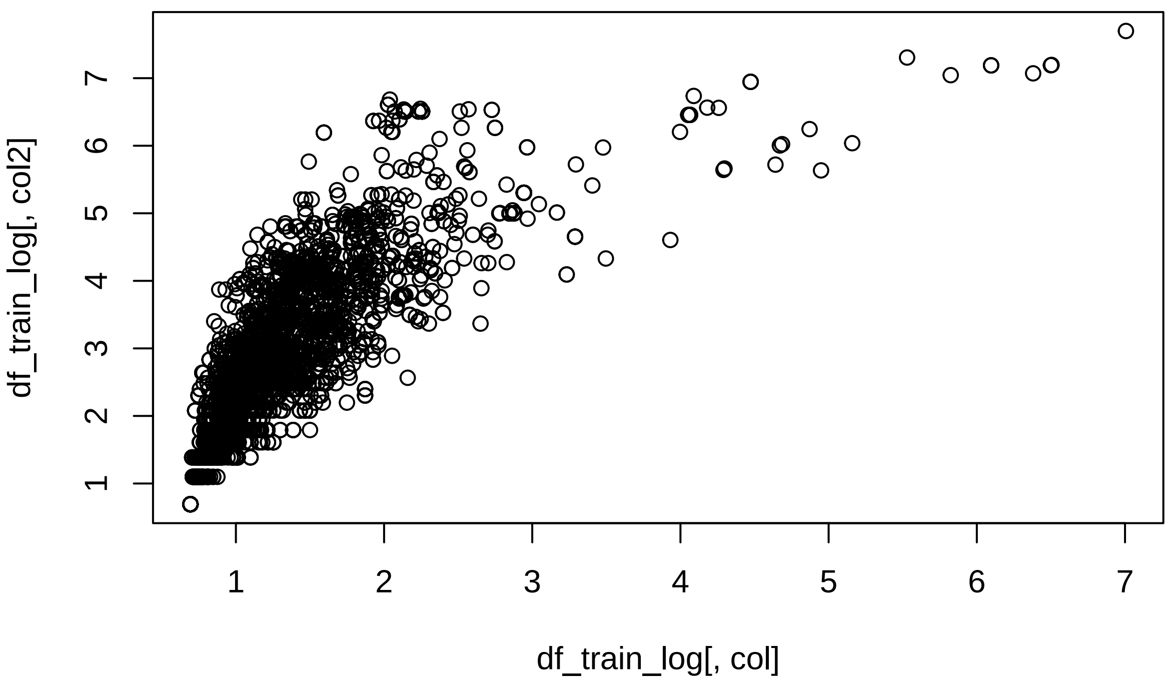



```r
for (col in 1:ncol(df_train_log)) {
  for (col2 in col:ncol(df_train_log)) {
    if (col!=col2 &&
        !is.na(cor(df_train_log[,col], df_train_log[,col2]))
        && cor(df_train_log[,col], df_train_log[,col2]) > 0.75){
      cat("scatter plot between columns",col,"and",col2,'\n')
      plot(df_train_log[,col], df_train_log[,col2])
    }

  }
}
```

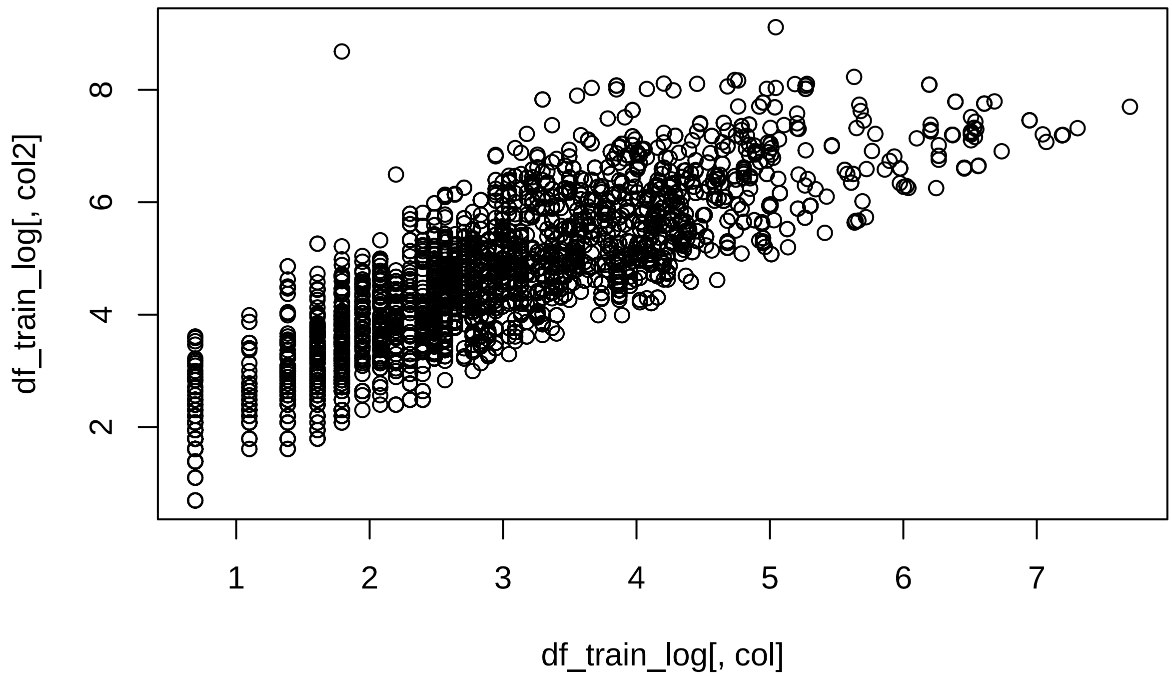
```
## scatter plot between columns 32 and 34
```



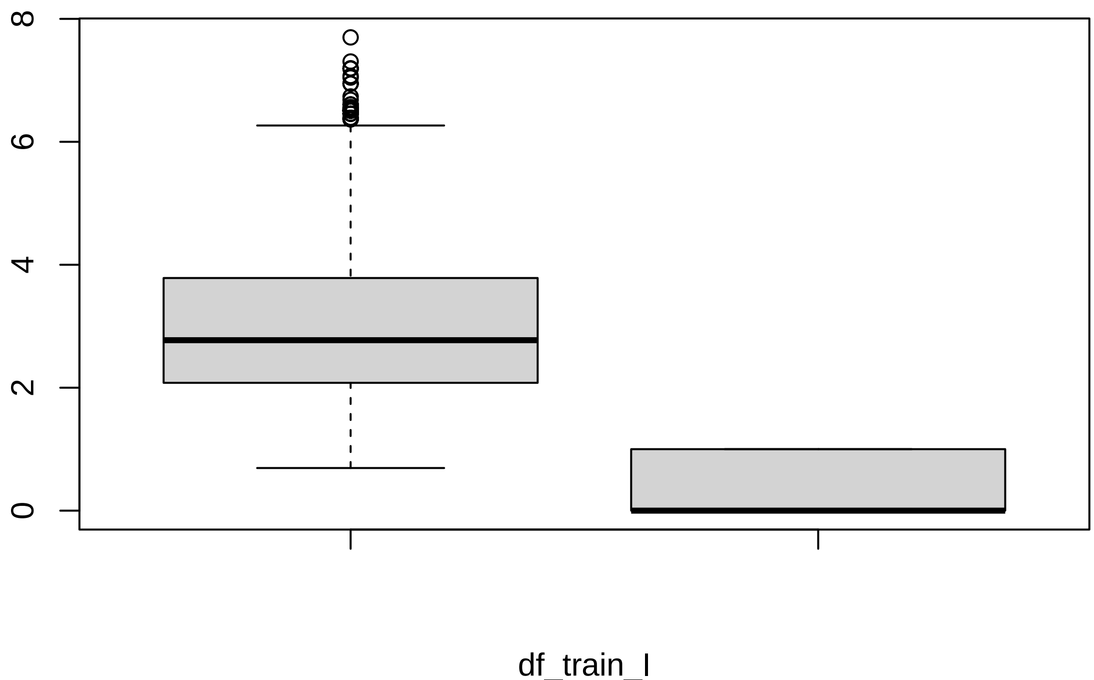

```
## scatter plot between columns 55 and 56
```




```
## scatter plot between columns 56 and 57
```

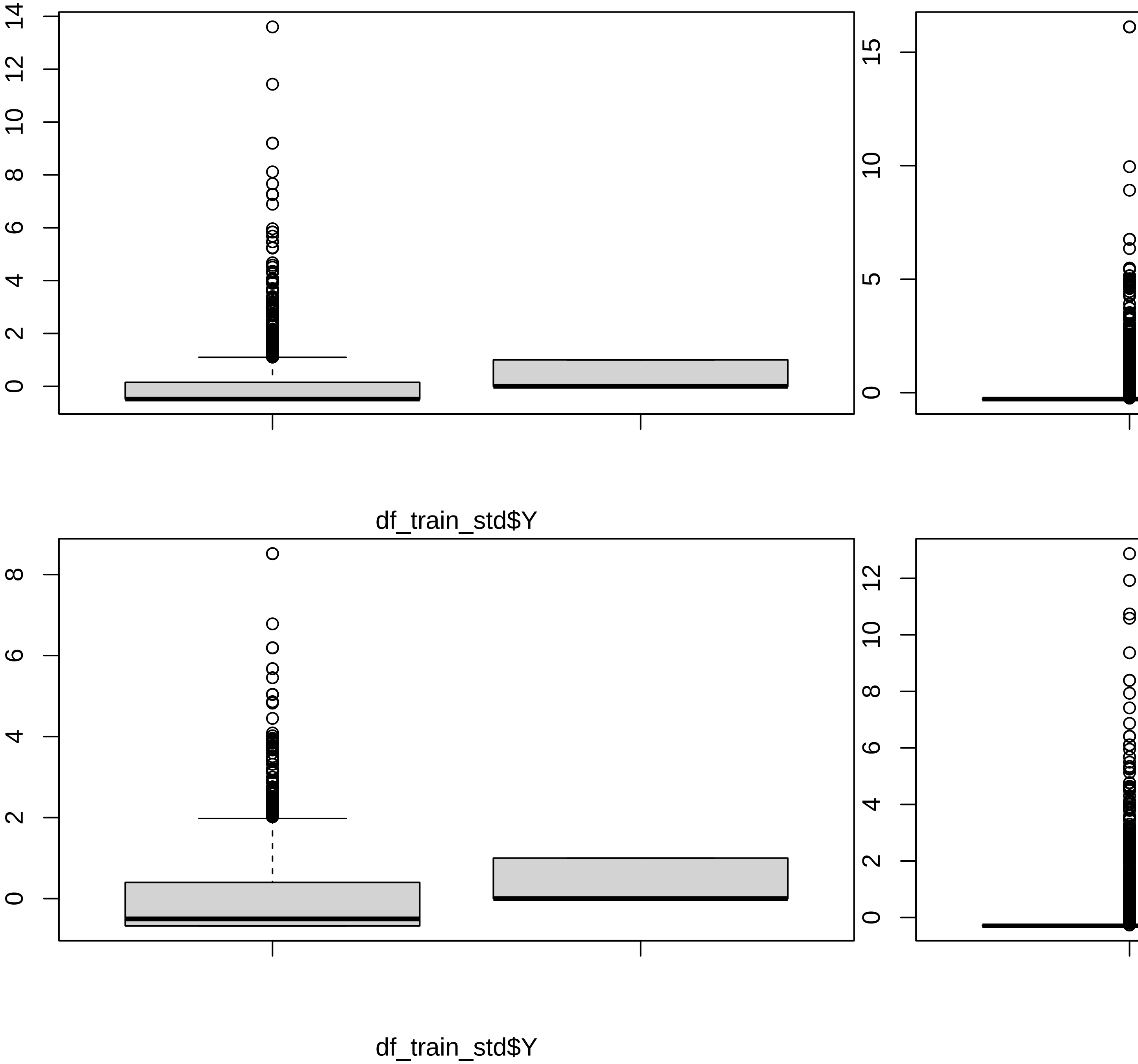


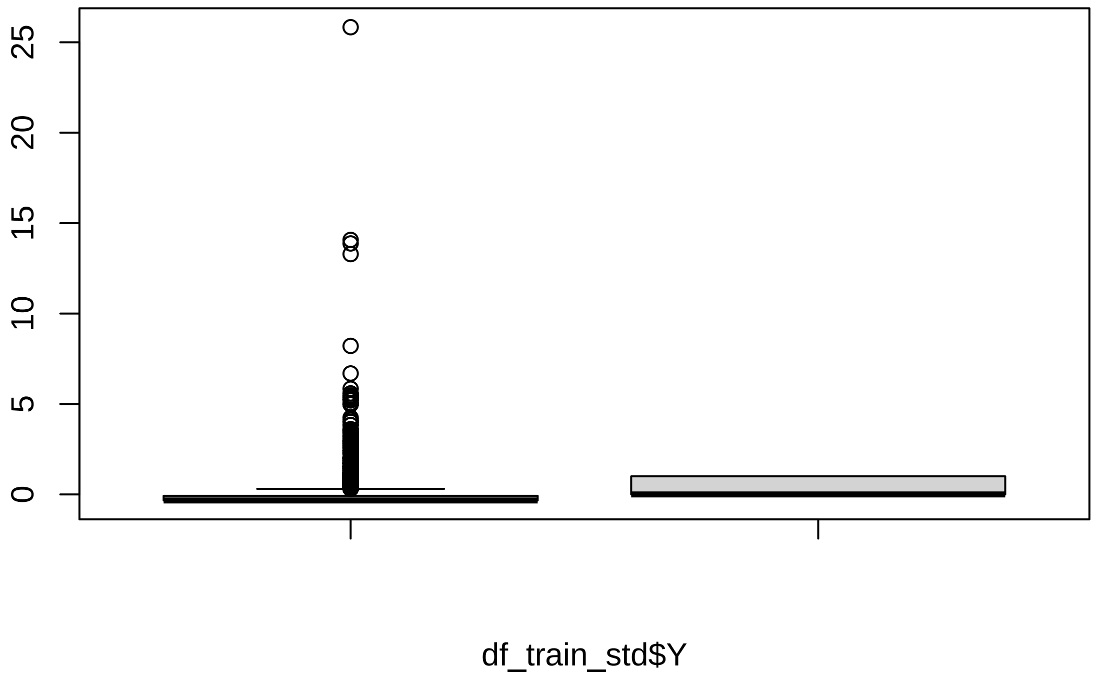

```
# box plot of variables and Y having high correlation
for (col in 1:57){
  if (cor(df_train_log[,col], df_train_log[,58]) > 0.5) {
    boxplot(df_train_log[,col],df_train_log$Y,xlab="df_train_I")
  }
}
```




```
for (col in 1:57){
  if (cor(df_train_std[,col], df_train_std[,58]) > 0.3) {
```

```
    boxplot(df_train_std[,col],df_train_std$Y,xlab="df_train_std$Y")
  }
}
```






```r
for (col in 1:57){
  if (!is.na(cor(df_train_I[,col], df_train_I[,58]))
        && cor(df_train_I[,col], df_train_I[,58]) > 0.8) {
    boxplot(df_train_I[,col],df_train_I$Y,xlab="df_train_I$Y")
  }
}
```

#b

```r
#logistic regression
#on std train set& test set
Logis_Reg_std_train = glm(as.factor(Y)~., data = df_train_std,
                          family = "binomial")
summary(Logis_Reg_std_train)
```

```
##
## Call:
## glm(formula = as.factor(Y) ~ ., family = "binomial", data = df_train_std)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.3245  -0.1988  -0.0001   0.0940   3.6053
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -7.361613   1.762032  -4.178 2.94e-05 ***
## x1            -0.070481   0.085457  -0.825 0.409508
## x2            -0.212713   0.136583  -1.557 0.119379
## x3             0.025730   0.074728   0.344 0.730611
## x4             5.425751   2.634732   2.059 0.039464 *
## x5             0.410339   0.088985   4.611 4.00e-06 ***
```

```
## x6            0.084893   0.057809   1.469 0.141965
## x7            1.307830   0.198298   6.595 4.24e-11 ***
## x8            0.201156   0.073106   2.752 0.005931 **
## x9            0.216452   0.100402   2.156 0.031095 *
## x10           0.057377   0.060904   0.942 0.346146
## x11          -0.195643   0.075244  -2.600 0.009320 **
## x12          -0.035522   0.073025  -0.486 0.626654
## x13          -0.132190   0.110701  -1.194 0.232431
## x14          -0.003391   0.062971  -0.054 0.957058
## x15           0.310890   0.232421   1.338 0.181022
## x16           1.100543   0.164513   6.690 2.24e-11 ***
## x17           0.596501   0.140010   4.260 2.04e-05 ***
## x18          -0.029935   0.083925  -0.357 0.721328
## x19           0.153596   0.077824   1.974 0.048423 *
## x20           1.802268   0.509072   3.540 0.000400 ***
## x21           0.499770   0.085012   5.879 4.13e-09 ***
## x22           0.104744   0.158733   0.660 0.509332
## x23           1.172850   0.241043   4.866 1.14e-06 ***
## x24           0.099468   0.061699   1.612 0.106930
## x25          -3.272100   0.581579  -5.626 1.84e-08 ***
## x26          -0.448614   0.391061  -1.147 0.251310
## x27         -18.555579   3.802484  -4.880 1.06e-06 ***
## x28           0.244576   0.170329   1.436 0.151031
## x29          -2.429257   1.662404  -1.461 0.143936
## x30           0.011451   0.096669   0.118 0.905706
## x31          -0.082969   0.257128  -0.323 0.746941
## x32          -0.374474   0.953628  -0.393 0.694553
## x33          -0.462586   0.246538  -1.876 0.060610 .
## x34           0.853992   1.011821   0.844 0.398661
## x35          -0.611776   0.353254  -1.732 0.083304 .
## x36           0.076193   0.169608   0.449 0.653264
## x37          -0.260527   0.148920  -1.749 0.080214 .
## x38          -0.151496   0.121348  -1.248 0.211871
## x39          -0.026332   0.152992  -0.172 0.863348
## x40          -0.157475   0.176775  -0.891 0.373027
## x41         -18.567059  12.230659  -1.518 0.128995
## x42          -1.695622   0.583196  -2.907 0.003644 **
## x43          -0.454237   0.239231  -1.899 0.057599 .
## x44          -0.734060   0.357167  -2.055 0.039857 *
## x45          -0.885922   0.177301  -4.997 5.83e-07 ***
## x46          -1.085099   0.255167  -4.252 2.11e-05 ***
## x47          -0.642456   0.325245  -1.975 0.048234 *
## x48          -0.502701   0.383351  -1.311 0.189745
## x49          -0.207169   0.101127  -2.049 0.040502 *
## x50           0.047509   0.060039   0.791 0.428765
## x51          -0.065868   0.129006  -0.511 0.609646
## x52           0.248040   0.058142   4.266 1.99e-05 ***
## x53           1.016788   0.162218   6.268 3.66e-10 ***
## x54           0.590673   0.335770   1.759 0.078550 .
## x55          -0.562090   0.229800  -2.446 0.014445 *
## x56           1.082872   0.293775   3.686 0.000228 ***
## x57           0.616630   0.141199   4.367 1.26e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4120.0  on 3065  degrees of freedom
## Residual deviance: 1157.4  on 3008  degrees of freedom
## AIC: 1273.4
##
## Number of Fisher Scoring iterations: 13
```

```
cat("On standardized dataset, x4,x5,x7,x8,x9,x11,x16,x17,x19,x20,x21,x23,x25,x27,
    x42,x44,x45,x46,x47,x49,x52,x53,x55,x56,x57 are statistically significant on
    0.05 significance level.",'\n')
```

```
## On standardized dataset, x4,x5,x7,x8,x9,x11,x16,x17,x19,x20,x21,x23,x25,x27,
##     x42,x44,x45,x46,x47,x49,x52,x53,x55,x56,x57 are statistically significant on
##     0.05 significance level.
```

```
prob_std_train = predict(Logis_Reg_std_train,df_train_std, type='response')
pred_std_train = ifelse(prob_std_train>0.5, "1", "0")
table(pred_std_train, df_train_std$Y) # confusion matrix
```

```
##
## pred_std_train    0    1
##              0 1761  133
##              1   87 1085
```

```
cat("On standardized train set, we have 133 type one errors; 87 type two errors",'\n')
```

```
## On standardized train set, we have 133 type one errors; 87 type two errors
```

```
cat("classification error on standardized train set:"
    ,mean(pred_std_train !=  df_train_std$Y))
```

```
## classification error on standardized train set: 0.07175473
```

```
prob_std_test = predict(Logis_Reg_std_train,df_test_std, type='response')
pred_std_test = ifelse(prob_std_test>0.5, "1", "0")
table(pred_std_test, df_test_std$Y) # confusion matrix
```

```
##
## pred_std_test   0   1
##             0 877  69
##             1  39 548
```

```
cat("On standardized test set, we have 69 type one errors; 39 type two errors",'\n')
```

```
## On standardized test set, we have 69 type one errors; 39 type two errors
```

```r
cat("classification error on standardized test set:"
    ,mean(pred_std_test !=  df_test_std$Y))
```

```
## classification error on standardized test set: 0.0704501
```

```r
#on log train set& test set
Logis_Reg_log_train = glm(as.factor(Y)~., data = df_train_log,
                         family = "binomial")
summary(Logis_Reg_log_train)
```

```
##
## Call:
## glm(formula = as.factor(Y) ~ ., family = "binomial", data = df_train_log)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.0831  -0.1647  -0.0010   0.0739   3.7853
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -5.55361    0.47536 -11.683  < 2e-16 ***
## x1          -0.50525    0.52078  -0.970 0.331955
## x2          -0.48375    0.41287  -1.172 0.241326
## x3          -0.34268    0.32461  -1.056 0.291122
## x4           2.49036    2.49963   0.996 0.319109
## x5           1.68052    0.26735   6.286 3.26e-10 ***
## x6           0.49007    0.49976   0.981 0.326780
## x7           3.81919    0.63656   6.000 1.98e-09 ***
## x8           1.11891    0.39254   2.850 0.004366 **
## x9           0.22162    0.61448   0.361 0.718353
## x10          0.20794    0.26664   0.780 0.435468
## x11         -1.73051    0.64790  -2.671 0.007563 **
## x12         -0.13019    0.21705  -0.600 0.548625
## x13         -1.47818    0.59699  -2.476 0.013284 *
## x14          0.49815    0.49244   1.012 0.311725
## x15          2.35454    1.31509   1.790 0.073389 .
## x16          2.00188    0.30550   6.553 5.64e-11 ***
## x17          2.00033    0.49917   4.007 6.14e-05 ***
## x18         -0.62599    0.34041  -1.839 0.065927 .
## x19          0.04967    0.17069   0.291 0.771070
## x20          4.74705    1.75988   2.697 0.006989 **
## x21          0.92793    0.20837   4.453 8.46e-06 ***
## x22          0.19784    0.59582   0.332 0.739858
## x23          3.39784    0.89163   3.811 0.000139 ***
## x24          1.27695    0.41124   3.105 0.001902 **
## x25         -3.97125    0.60153  -6.602 4.06e-11 ***
## x26         -0.43396    0.74531  -0.582 0.560393
## x27         -5.92236    1.42774  -4.148 3.35e-05 ***
## x28          1.27690    0.58913   2.167 0.030201 *
## x29         -5.52545    3.47037  -1.592 0.111344
## x30         -0.08833    0.47636  -0.185 0.852891
## x31         -1.17928    2.44794  -0.482 0.629989
```

```
## x32          -4.26131    4.43663  -0.960 0.336813
## x33          -1.44589    0.73243  -1.974 0.048370 *
## x34           0.86732    4.05415   0.214 0.830598
## x35          -2.60248    1.20496  -2.160 0.030788 *
## x36           0.44061    0.70994   0.621 0.534841
## x37          -1.55260    0.59961  -2.589 0.009615 **
## x38          -1.10219    1.36375  -0.808 0.418971
## x39           0.09939    0.80741   0.123 0.902026
## x40          -1.66153    1.14748  -1.448 0.147621
## x41         -45.30209   35.39196  -1.280 0.200541
## x42          -4.12654    1.24565  -3.313 0.000924 ***
## x43          -5.08561    1.94169  -2.619 0.008815 **
## x44          -2.90440    1.49695  -1.940 0.052354 .
## x45          -2.02986    0.41499  -4.891 1.00e-06 ***
## x46          -2.21581    0.52201  -4.245 2.19e-05 ***
## x47          -7.41902    4.88355  -1.519 0.128716
## x48          -2.02098    1.39842  -1.445 0.148405
## x49          -1.58851    0.79263  -2.004 0.045059 *
## x50          -0.01172    0.62116  -0.019 0.984949
## x51          -3.40427    2.64864  -1.285 0.198691
## x52           2.24783    0.29972   7.500 6.39e-14 ***
## x53           4.93003    0.88667   5.560 2.70e-08 ***
## x54          -0.01276    2.13277  -0.006 0.995225
## x55           0.57047    0.33492   1.703 0.088515 .
## x56           0.09317    0.19497   0.478 0.632742
## x57           0.75138    0.13167   5.707 1.15e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4120.00  on 3065  degrees of freedom
## Residual deviance:  930.67  on 3008  degrees of freedom
## AIC: 1046.7
##
## Number of Fisher Scoring iterations: 12
```

```r
cat("On log dataset, x5,x7,x8,x11,x13,,x16,x17,x20,x21,x23,x24,x25,x27,x28,x33,
    x35,x37,x42,x43,x45,x46,x49,x52,x53,x57 are statistically significant on
    0.05 significance level.",'\n')
```

```
## On log dataset, x5,x7,x8,x11,x13,,x16,x17,x20,x21,x23,x24,x25,x27,x28,x33,
##     x35,x37,x42,x43,x45,x46,x49,x52,x53,x57 are statistically significant on
##     0.05 significance level.
```

```r
prob_log_train = predict(Logis_Reg_log_train,df_train_log, type='response')
pred_log_train = ifelse(prob_log_train>0.5, "1", "0")
table(pred_log_train, df_train_log$Y) # confusion matrix
```

```
##
## pred_log_train    0    1
##              0 1765   94
##              1   83 1124
```

```r
cat("On log train set, we have 94 type one errors; 83 type two errors",'\n')
```

```
## On log train set, we have 94 type one errors; 83 type two errors
```

```r
cat("classification error on log train set:"
    ,mean(pred_log_train !=  df_train_log$Y))
```

```
## classification error on log train set: 0.05772994
```

```r
prob_log_test = predict(Logis_Reg_log_train,df_test_log, type='response')
pred_log_test = ifelse(prob_log_test>0.5, "1", "0")
table(pred_log_test, df_test_log$Y) # confusion matrix
```

```
##
## pred_log_test   0   1
##             0 879  50
##             1  37 567
```

```r
cat("On log test set, we have 50 type one errors; 37 type two errors",'\n')
```

```
## On log test set, we have 50 type one errors; 37 type two errors
```

```r
cat("classification error on log test set:"
    ,mean(pred_log_test !=  df_test_log$Y))
```

```
## classification error on log test set: 0.05675147
```

```r
#on discretized train set& test set
Logis_Reg_I_train = glm(as.factor(Y)~., data = df_train_I,
                        family = "binomial")
summary(Logis_Reg_I_train)
```

```
##
## Call:
## glm(formula = as.factor(Y) ~ ., family = "binomial", data = df_train_I)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.6393  -0.1906  -0.0130   0.0600   3.9295
##
## Coefficients: (3 not defined because of singularities)
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.102416   0.189853 -11.074  < 2e-16 ***
## x1          -0.303288   0.289817  -1.046 0.295340
## x2          -0.378468   0.275804  -1.372 0.169991
## x3          -0.199096   0.212662  -0.936 0.349166
## x4           1.096263   0.824254   1.330 0.183516
## x5           1.268089   0.216147   5.867 4.44e-09 ***
## x6           0.251837   0.272999   0.922 0.356278
```

```
## x7           2.986596   0.386284   7.732 1.06e-14 ***
## x8           0.875956   0.316310   2.769 0.005618 **
## x9           0.228803   0.325213   0.704 0.481713
## x10          0.742334   0.238269   3.116 0.001836 **
## x11         -1.162232   0.334524  -3.474 0.000512 ***
## x12         -0.078386   0.194281  -0.403 0.686606
## x13         -1.161882   0.311431  -3.731 0.000191 ***
## x14          0.941406   0.452030   2.083 0.037286 *
## x15          2.006006   0.693341   2.893 0.003813 **
## x16          1.984574   0.226462   8.763  < 2e-16 ***
## x17          1.096489   0.319793   3.429 0.000606 ***
## x18         -0.857060   0.264974  -3.235 0.001219 **
## x19          0.006173   0.224878   0.027 0.978102
## x20          1.670868   0.554535   3.013 0.002586 **
## x21          0.834541   0.210275   3.969 7.22e-05 ***
## x22          0.811704   0.555362   1.462 0.143857
## x23          1.787932   0.392434   4.556 5.21e-06 ***
## x24          1.385800   0.343260   4.037 5.41e-05 ***
## x25         -3.611805   0.473169  -7.633 2.29e-14 ***
## x26         -0.640909   0.497469  -1.288 0.197628
## x27         -4.432614   0.740639  -5.985 2.17e-09 ***
## x28          1.981100   0.457456   4.331 1.49e-05 ***
## x29         -1.174979   0.668921  -1.757 0.078998 .
## x30         -0.183169   0.519468  -0.353 0.724382
## x31         -1.558354   1.033708  -1.508 0.131673
## x32         -2.211033   1.150862  -1.921 0.054707 .
## x33         -0.926303   0.562100  -1.648 0.099366 .
## x34          0.536590   1.068201   0.502 0.615435
## x35         -0.973396   0.565678  -1.721 0.085295 .
## x36          0.636609   0.417224   1.526 0.127055
## x37         -1.440825   0.348517  -4.134 3.56e-05 ***
## x38          1.173481   0.741366   1.583 0.113453
## x39          0.037746   0.413234   0.091 0.927221
## x40         -0.611575   0.557754  -1.096 0.272861
## x41         -5.823142   3.179724  -1.831 0.067050 .
## x42         -2.410828   0.508740  -4.739 2.15e-06 ***
## x43         -1.500588   0.638112  -2.352 0.018693 *
## x44         -1.301654   0.521225  -2.497 0.012514 *
## x45         -1.391111   0.235936  -5.896 3.72e-09 ***
## x46         -1.789878   0.363562  -4.923 8.52e-07 ***
## x47         -0.695889   1.130614  -0.615 0.538227
## x48         -1.512213   0.617514  -2.449 0.014330 *
## x49         -0.070817   0.275485  -0.257 0.797130
## x50          0.185426   0.196175   0.945 0.344553
## x51         -0.056807   0.409947  -0.139 0.889789
## x52          1.476323   0.186253   7.926 2.26e-15 ***
## x53          1.858610   0.250030   7.434 1.06e-13 ***
## x54         -0.794192   0.338407  -2.347 0.018933 *
## x55                NA         NA      NA       NA
## x56                NA         NA      NA       NA
## x57                NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4120.0  on 3065  degrees of freedom
## Residual deviance: 1014.6  on 3011  degrees of freedom
## AIC: 1124.6
##
## Number of Fisher Scoring iterations: 9
```

```
cat("On discretized dataset, x5,x7,x8,x10,x11,x13,x14,x15,x16,x17,x18,x20,x21,
    x23,x24,x25,x27,x28,x37,x42,x43,x44,x45,x46,x48,x52,x53,x54 are statistically
    significant on 0.05 significance level.",'\n')
```

```
## On discretized dataset, x5,x7,x8,x10,x11,x13,x14,x15,x16,x17,x18,x20,x21,
##     x23,x24,x25,x27,x28,x37,x42,x43,x44,x45,x46,x48,x52,x53,x54 are statistically
##     significant on 0.05 significance level.
```

```
prob_I_train = predict(Logis_Reg_I_train,df_train_I, type='response')
pred_I_train = ifelse(prob_I_train>0.5, "1", "0")
table(pred_I_train, df_train_I$Y) # confusion matrix
```

```
##
## pred_I_train    0    1
##            0 1778  105
##            1   70 1113
```

```
cat("On discretized train set, we have 105 type one errors; 70 type two errors",'\n')
```

```
## On discretized train set, we have 105 type one errors; 70 type two errors
```

```
cat("classification error on discretized train set:"
    ,mean(pred_I_train !=  df_train_I$Y))
```

```
## classification error on discretized train set: 0.05707763
```

```
prob_I_test = predict(Logis_Reg_I_train,df_test_I, type='response')
pred_I_test = ifelse(prob_I_test>0.5, "1", "0")
table(pred_I_test, df_test_I$Y) # confusion matrix
```

```
##
## pred_I_test   0   1
##           0 859  67
##           1  57 550
```

```
cat("On discretized test set, we have 67 type one errors; 57 type two errors",'\n') ###
```

```
## On discretized test set, we have 67 type one errors; 57 type two errors
```

```r
cat("classification error on discretized test set:"
    ,mean(pred_I_test !=  df_test_I$Y))
```

```
## classification error on discretized test set: 0.08088715
```

#c

```r
#lda & qda
library(MASS)
#lda on standardized dataset
lda_std_train <- lda(as.factor(Y)~.,data=df_train_std)
print(lda_std_train)
```

```
## Call:
## lda(as.factor(Y) ~ ., data = df_train_std)
##
## Prior probabilities of groups:
##         0         1
## 0.6027397 0.3972603
##
## Group means:
##           x1          x2         x3          x4         x5         x6
## 0 -0.1069174  0.03514763 -0.1602290 -0.04479982 -0.2503329 -0.1658362
## 1  0.1622195 -0.05332744  0.2431061  0.06797214  0.3798154  0.2516135
##           x7         x8         x9        x10        x11          x12
## 0 -0.2653823 -0.1561049 -0.1705740 -0.1080780 -0.1928989 -0.009950874
## 1  0.4026491  0.2368489  0.2588019  0.1639804  0.2926742  0.015097878
##          x13         x14        x15        x16        x17        x18        x19
## 0 -0.1170612 -0.03595389 -0.1706871 -0.1798817 -0.1910515 -0.1680778 -0.2368653
## 1  0.1776101  0.05455073  0.2589735  0.2729239  0.2898712  0.2550147  0.3593818
##          x20        x21         x22        x23        x24        x25        x26
## 0 -0.1704232 -0.3148982 -0.07377009 -0.2760402 -0.1857145  0.2142886  0.1932191
## 1  0.2585732  0.4777765  0.11192704  0.4188197  0.2817737 -0.3251276 -0.2931600
##          x27        x28        x29        x30         x31         x32
## 0  0.1424733  0.1392816  0.1081494  0.1270863  0.09765674  0.09291623
## 1 -0.2161665 -0.2113238 -0.1640888 -0.1928206 -0.14816884 -0.14097635
##          x33         x34        x35        x36        x37         x38
## 0  0.0850902  0.08801796  0.1201857  0.1225343  0.1513690  0.02318666
## 1 -0.1291024 -0.13354449 -0.1823507 -0.1859141 -0.2296633 -0.03517976
##          x39         x40         x41        x42        x43         x44
## 0  0.1058538  0.05531977  0.08445125  0.1064889  0.1058379  0.09722614
## 1 -0.1606057 -0.08393344 -0.12813292 -0.1615693 -0.1605817 -0.14751553
##          x45        x46         x47         x48         x49         x50
## 0  0.1320741  0.1258356  0.04790730  0.06393950  0.04390115  0.08091938
## 1 -0.2003884 -0.1909230 -0.07268694 -0.09701165 -0.06660864 -0.12277423
##           x51        x52        x53         x54         x55        x56
## 0  0.05399394 -0.1860095 -0.2738910 -0.02848037 -0.08535606 -0.2423106
## 1 -0.08192184  0.2822214  0.4155588  0.04321159  0.12950575  0.3676437
##          x57
## 0 -0.2335307
## 1  0.3543224
##
```

```
## Coefficients of linear discriminants:
##               LD1
## x1  -0.0526740031
## x2  -0.0617536733
## x3   0.0733126469
## x4   0.0766915865
## x5   0.2967065872
## x6   0.0794902809
## x7   0.3558985084
## x8   0.1479128218
## x9   0.0295281412
## x10  0.0038643527
## x11  0.0452143600
## x12 -0.0852813073
## x13 -0.0048451332
## x14 -0.0483942118
## x15 -0.0007720204
## x16  0.2233197957
## x17  0.0833658517
## x18  0.1300997822
## x19  0.1213140013
## x20  0.1137772863
## x21  0.3001263541
## x22  0.1866456962
## x23  0.2671603077
## x24  0.1743541760
## x25 -0.1713407032
## x26 -0.0704307788
## x27 -0.1655407020
## x28 -0.0276966221
## x29 -0.0695662839
## x30 -0.0555452458
## x31 -0.0164452174
## x32 -0.2719340352
## x33 -0.0994725729
## x34  0.4116799767
## x35 -0.0693811328
## x36  0.0164011009
## x37 -0.0970879515
## x38 -0.0602028507
## x39 -0.0196065988
## x40 -0.0218984182
## x41 -0.0085585910
## x42 -0.1036432569
## x43 -0.0364043829
## x44 -0.0983251073
## x45 -0.1726127294
## x46 -0.1307041959
## x47 -0.0520853099
## x48 -0.0554336366
## x49 -0.1288334500
## x50  0.0075783935
## x51 -0.0107957126
## x52  0.2011100673
```

```
## x53  0.2654845875
## x54  0.0489521280
## x55  0.0042117602
## x56  0.0721214059
## x57  0.2452356327
```

```r
lda_pred_std_train=predict(lda_std_train, df_train_std)
lda_std_train_error=mean(lda_pred_std_train$class!=df_train_std$Y)
cat("Classification error of LDA on standardized train set is",lda_std_train_error,'\n')
```

```
## Classification error of LDA on standardized train set is 0.1017613
```

```r
lda_pred_std_test=predict(lda_std_train, df_test_std)
lda_std_test_error=mean(lda_pred_std_test$class!=df_test_std$Y)
cat("Classification error of LDA on standardized test set is",lda_std_test_error,'\n')
```

```
## Classification error of LDA on standardized test set is 0.1030659
```

```r
#lda on log dataset
lda_log_train <- lda(as.factor(Y)~.,data=df_train_log)
print(lda_log_train)
```

```
## Call:
## lda(as.factor(Y) ~ ., data = df_train_log)
##
## Prior probabilities of groups:
##         0         1
## 0.6027397 0.3972603
##
## Group means:
##           x1         x2        x3          x4         x5         x6          x7
## 0 0.05315155 0.07959758 0.1375519 0.001253274 0.09927244 0.03851495 0.005018299
## 1 0.12320915 0.11265753 0.2939761 0.025570246 0.35891627 0.14007448 0.198678037
##           x8         x9       x10        x11       x12        x13        x14
## 0 0.02911729 0.03003888 0.1015924 0.01420926 0.2975586 0.04352078 0.02267674
## 1 0.13367902 0.11714435 0.2365405 0.08677934 0.3644886 0.10872460 0.04616668
##          x15        x16        x17        x18       x19         x20       x21
## 0 0.00561555 0.04252057 0.03960108 0.06499006 0.5870133 0.003600182 0.2333573
## 1 0.08059068 0.30381482 0.17613046 0.20519828 1.0562134 0.102872725 0.7281596
##          x22         x23         x24        x25         x26         x27
## 0 0.01117143 0.005257954 0.009195921 0.39115828 0.222766623 0.299920360
## 1 0.06764085 0.160941819 0.152543234 0.01468332 0.009955504 0.001910686
##           x28          x29         x30         x31          x32         x33
## 0 0.112979197 0.0770728349 0.098920500 0.059246231 0.0425018738 0.069848897
## 1 0.007535808 0.0003541024 0.009257066 0.001064792 0.0006326148 0.009409053
##           x34         x35        x36        x37         x38         x39
## 0 0.042327477 0.106264961 0.08920778 0.12410631 0.008446234 0.059000174
## 1 0.002790771 0.008187984 0.01804994 0.02458314 0.003963517 0.009167473
##          x40          x41         x42         x43        x44        x45
## 0 0.04852852 0.0437052621 0.091363412 0.043201950 0.05400213 0.20361459
## 1 0.02872142 0.0001565028 0.002957748 0.008381538 0.00445594 0.08500154
##          x46          x47        x48        x49        x50         x51
```

```
## 0 0.13172703 0.0078437046 0.03439685 0.03016528 0.12674641 0.017786021
## 1 0.01214328 0.0004951255 0.00340204 0.01683878 0.09315826 0.006727094
##          x52        x53        x54      x55      x56      x57
## 0 0.06452462 0.01174497 0.01688401 1.126298 2.388785 4.066896
## 1 0.34984733 0.13587984 0.03553386 1.638094 3.722761 5.393894
##
## Coefficients of linear discriminants:
##              LD1
## x1  -0.575235671
## x2  -0.162912246
## x3  -0.047392824
## x4   0.435354896
## x5   0.715502854
## x6   0.194987045
## x7   1.552542592
## x8   0.657935099
## x9  -0.141445606
## x10 -0.040037396
## x11  0.129640354
## x12 -0.236072919
## x13 -0.394014627
## x14 -0.166042142
## x15  0.069497045
## x16  0.849856603
## x17  0.170920743
## x18  0.253772283
## x19  0.072952107
## x20  0.265146805
## x21  0.481891318
## x22  0.603328790
## x23  1.131377296
## x24  1.181131786
## x25 -0.631451448
## x26 -0.045044927
## x27 -0.236185078
## x28  0.070308337
## x29 -0.303299321
## x30  0.012955773
## x31  0.015237992
## x32 -0.043543551
## x33 -0.587684805
## x34  1.039301922
## x35 -0.434103893
## x36  0.219527939
## x37 -0.446717853
## x38 -0.568521114
## x39 -0.055309147
## x40 -0.311751355
## x41  0.038068320
## x42 -0.389185065
## x43 -0.114871179
## x44 -0.524748092
## x45 -0.388516847
## x46 -0.587599870
```

```
## x47 -1.026554566
## x48 -0.439257727
## x49 -0.996247164
## x50 -0.147432468
## x51 -0.195222568
## x52  1.206070483
## x53  1.729503387
## x54  0.045268764
## x55 -0.001562929
## x56  0.100691654
## x57  0.208945883
```

```
lda_pred_log_train=predict(lda_log_train, df_train_log)
lda_log_train_error=mean(lda_pred_log_train$class!=df_train_log$Y)
cat("Classification error of LDA on log train set is",lda_log_train_error,'\n')
```

```
## Classification error of LDA on log train set is 0.0603392
```

```
lda_pred_log_test=predict(lda_log_train, df_test_log)
lda_log_test_error=mean(lda_pred_log_test$class!=df_test_log$Y)
cat("Classification error of LDA on log test set is",lda_std_test_error,'\n')
```

```
## Classification error of LDA on log test set is 0.1030659
```

```
#qda on standardized dataset
qda_std_train <- qda(as.factor(Y)~.,data=df_train_std)
print(qda_std_train)
```

```
## Call:
## qda(as.factor(Y) ~ ., data = df_train_std)
##
## Prior probabilities of groups:
##         0         1
## 0.6027397 0.3972603
##
## Group means:
##           x1          x2         x3          x4         x5         x6
## 0 -0.1069174  0.03514763 -0.1602290 -0.04479982 -0.2503329 -0.1658362
## 1  0.1622195 -0.05332744  0.2431061  0.06797214  0.3798154  0.2516135
##           x7          x8         x9         x10        x11          x12
## 0 -0.2653823 -0.1561049 -0.1705740 -0.1080780 -0.1928989 -0.009950874
## 1  0.4026491  0.2368489  0.2588019  0.1639804  0.2926742  0.015097878
##          x13         x14        x15        x16        x17        x18        x19
## 0 -0.1170612 -0.03595389 -0.1706871 -0.1798817 -0.1910515 -0.1680778 -0.2368653
## 1  0.1776101  0.05455073  0.2589735  0.2729239  0.2898712  0.2550147  0.3593818
##          x20        x21         x22        x23        x24        x25        x26
## 0 -0.1704232 -0.3148982 -0.07377009 -0.2760402 -0.1857145  0.2142886  0.1932191
## 1  0.2585732  0.4777765  0.11192704  0.4188197  0.2817737 -0.3251276 -0.2931600
##          x27        x28        x29        x30         x31         x32
## 0  0.1424733  0.1392816  0.1081494  0.1270863  0.09765674  0.09291623
## 1 -0.2161665 -0.2113238 -0.1640888 -0.1928206 -0.14816884 -0.14097635
##          x33         x34        x35        x36        x37          x38
```

```
## 0  0.0850902  0.08801796  0.1201857  0.1225343  0.1513690  0.02318666
## 1 -0.1291024 -0.13354449 -0.1823507 -0.1859141 -0.2296633 -0.03517976
##          x39         x40         x41        x42        x43         x44
## 0  0.1058538  0.05531977  0.08445125  0.1064889  0.1058379  0.09722614
## 1 -0.1606057 -0.08393344 -0.12813292 -0.1615693 -0.1605817 -0.14751553
##          x45        x46         x47         x48         x49         x50
## 0  0.1320741  0.1258356  0.04790730  0.06393950  0.04390115  0.08091938
## 1 -0.2003884 -0.1909230 -0.07268694 -0.09701165 -0.06660864 -0.12277423
##           x51        x52        x53         x54         x55        x56
## 0  0.05399394 -0.1860095 -0.2738910 -0.02848037 -0.08535606 -0.2423106
## 1 -0.08192184  0.2822214  0.4155588  0.04321159  0.12950575  0.3676437
##          x57
## 0 -0.2335307
## 1  0.3543224
```

```
qda_pred_std_train=predict(qda_std_train, df_train_std)
qda_std_train_error=mean(qda_pred_std_train$class!=df_train_std$Y)
cat("Classification error of QDA on standardized train set is",qda_std_train_error,'\n')
```

```
## Classification error of QDA on standardized train set is 0.1787345
```

```
qda_pred_std_test=predict(qda_std_train, df_test_std)
qda_std_test_error=mean(qda_pred_std_test$class!=df_test_std$Y)
cat("Classification error of QDA on standardized test set is",qda_std_test_error,'\n')
```

```
## Classification error of QDA on standardized test set is 0.1748206
```

```
#qda on log daataset
qda_log_train <- qda(as.factor(Y)~.,data=df_train_log)
print(qda_log_train)
```

```
## Call:
## qda(as.factor(Y) ~ ., data = df_train_log)
##
## Prior probabilities of groups:
##         0         1
## 0.6027397 0.3972603
##
## Group means:
##           x1         x2        x3          x4         x5         x6          x7
## 0 0.05315155 0.07959758 0.1375519 0.001253274 0.09927244 0.03851495 0.005018299
## 1 0.12320915 0.11265753 0.2939761 0.025570246 0.35891627 0.14007448 0.198678037
##           x8         x9       x10        x11       x12        x13        x14
## 0 0.02911729 0.03003888 0.1015924 0.01420926 0.2975586 0.04352078 0.02267674
## 1 0.13367902 0.11714435 0.2365405 0.08677934 0.3644886 0.10872460 0.04616668
##          x15        x16        x17        x18       x19         x20       x21
## 0 0.00561555 0.04252057 0.03960108 0.06499006 0.5870133 0.003600182 0.2333573
## 1 0.08059068 0.30381482 0.17613046 0.20519828 1.0562134 0.102872725 0.7281596
##          x22         x23         x24        x25         x26         x27
## 0 0.01117143 0.005257954 0.009195921 0.39115828 0.222766623 0.299920360
## 1 0.06764085 0.160941819 0.152543234 0.01468332 0.009955504 0.001910686
##           x28          x29         x30         x31          x32         x33
```

```
## 0 0.112979197 0.0770728349 0.098920500 0.059246231 0.0425018738 0.069848897
## 1 0.007535808 0.0003541024 0.009257066 0.001064792 0.0006326148 0.009409053
##           x34          x35        x36        x37         x38         x39
## 0 0.042327477 0.106264961 0.08920778 0.12410631 0.008446234 0.059000174
## 1 0.002790771 0.008187984 0.01804994 0.02458314 0.003963517 0.009167473
##          x40          x41         x42         x43        x44        x45
## 0 0.04852852 0.0437052621 0.091363412 0.043201950 0.05400213 0.20361459
## 1 0.02872142 0.0001565028 0.002957748 0.008381538 0.00445594 0.08500154
##          x46          x47        x48        x49        x50         x51
## 0 0.13172703 0.0078437046 0.03439685 0.03016528 0.12674641 0.017786021
## 1 0.01214328 0.0004951255 0.00340204 0.01683878 0.09315826 0.006727094
##          x52        x53        x54      x55      x56      x57
## 0 0.06452462 0.01174497 0.01688401 1.126298 2.388785 4.066896
## 1 0.34984733 0.13587984 0.03553386 1.638094 3.722761 5.393894
```

```
qda_pred_log_train=predict(qda_log_train, df_train_log)
qda_log_train_error=mean(qda_pred_log_train$class!=df_train_log$Y)
cat("Classification error of QDA on log train set is",qda_log_train_error,'\n')
```

```
## Classification error of QDA on log train set is 0.1588389
```

```
qda_pred_log_test=predict(qda_log_train, df_test_log)
qda_log_test_error=mean(qda_pred_log_test$class!=df_test_log$Y)
cat("Classification error of QDA on log test set is",qda_log_test_error,'\n')
```

```
## Classification error of QDA on log test set is 0.1572081
```

#d

```
#linear and nonlinear SVM
library(e1071)
#non linear on std set
tune.gaussian.std=tune(svm, as.factor(Y)~., data=df_train_std,
                       kernel ="radial",
                   ranges=list(cost=c(0.005, 0.05, 0.5),
                               gamma=c(0.1,0.5,1)))
# Check the selection results
summary(tune.gaussian.std)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   0.5   0.1
##
## - best performance: 0.08415086
##
## - Detailed performance results:
##    cost gamma      error dispersion
```

```
## 1 0.005   0.1 0.39727279 0.03572213
## 2 0.050   0.1 0.27365183 0.03283814
## 3 0.500   0.1 0.08415086 0.01719672
## 4 0.005   0.5 0.39727279 0.03572213
## 5 0.050   0.5 0.39727279 0.03572213
## 6 0.500   0.5 0.13634477 0.01898058
## 7 0.005   1.0 0.39727279 0.03572213
## 8 0.050   1.0 0.39727279 0.03572213
## 9 0.500   1.0 0.21202870 0.03777716
```

```
# Choose the best model (classifier with optimal C)
bestmod.gaussian.std =tune.gaussian.std$best.model
# Check the classifier
summary(bestmod.gaussian.std)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = as.factor(Y) ~ ., data = df_train_std,
##     ranges = list(cost = c(0.005, 0.05, 0.5), gamma = c(0.1, 0.5,
##         1)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.5
##
## Number of Support Vectors:  1652
##
##  ( 879 773 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
cat("Best cost for gaussian SVM on std dataset is:", 0.5, "gamma:",0.1,'\n')
```

```
## Best cost for gaussian SVM on std dataset is: 0.5 gamma: 0.1
```

```
gaussian_svm_std_train_error = mean(predict(bestmod.gaussian.std, df_train_std)!=
                                    df_train_std$Y)
gaussian_svm_std_test_error = mean(predict(bestmod.gaussian.std, df_test_std)!=
                                   df_test_std$Y)
cat("gaussian_svm_std_train_error:", gaussian_svm_std_train_error, '\n')
```

```
## gaussian_svm_std_train_error: 0.03065884
```

```
cat("gaussian_svm_std_test_error:", gaussian_svm_std_test_error, '\n')
```

```
## gaussian_svm_std_test_error: 0.07762557
```

```r
#linear svm on std set
tune.linear.std=tune(svm, as.factor(Y)~., data=df_train_std,
                        kernel ="linear",
                    ranges=list(cost=c(0.005, 0.05, 0.5)))
# Check the selection results
summary(tune.linear.std)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.5
##
## - best performance: 0.07306742
##
## - Detailed performance results:
##    cost      error dispersion
## 1 0.005 0.08611271 0.01496371
## 2 0.050 0.07764046 0.01812194
## 3 0.500 0.07306742 0.01456664
```

```r
# Choose the best model (classifier with optimal C)
bestmod.linear.std =tune.linear.std$best.model
# Check the classifier
summary(bestmod.linear.std)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = as.factor(Y) ~ ., data = df_train_std,
##     ranges = list(cost = c(0.005, 0.05, 0.5)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.5
##
## Number of Support Vectors:  638
##
##  ( 329 309 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```r
cat("Best cost for linear SVM on std dataset is:", 0.5,'\n')
```

```
## Best cost for linear SVM on std dataset is: 0.5
```

```r
linear_svm_std_train_error = mean(predict(bestmod.linear.std, df_train_std)!=
                                    df_train_std$Y)
linear_svm_std_test_error = mean(predict(bestmod.linear.std, df_test_std)!=
                                    df_test_std$Y)
cat("linear_svm_std_train_error:", linear_svm_std_train_error, '\n')
```

```
## linear_svm_std_train_error: 0.06621005
```

```r
cat("linear_svm_std_test_error:", linear_svm_std_test_error, '\n')
```

```
## linear_svm_std_test_error: 0.06849315
```

```r
#non linear on log set
tune.gaussian.log=tune(svm, as.factor(Y)~., data=df_train_log,
                       kernel ="radial",
                       ranges=list(cost=c(0.005, 0.05, 0.5),
                                   gamma=c(0.1,0.5,1)))
# Check the selection results
summary(tune.gaussian.log)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   0.5   0.1
##
## - best performance: 0.08805433
##
## - Detailed performance results:
##    cost gamma      error dispersion
## 1 0.005   0.1 0.39727385 0.01962127
## 2 0.050   0.1 0.34508314 0.01554972
## 3 0.500   0.1 0.08805433 0.01993560
## 4 0.005   0.5 0.39727385 0.01962127
## 5 0.050   0.5 0.39727385 0.01962127
## 6 0.500   0.5 0.21265249 0.01740336
## 7 0.005   1.0 0.39727385 0.01962127
## 8 0.050   1.0 0.39727385 0.01962127
## 9 0.500   1.0 0.22993549 0.01875394
```

```r
# Choose the best model (classifier with optimal C)
bestmod.gaussian.log =tune.gaussian.log$best.model
# Check the classifier
summary(bestmod.gaussian.log)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = as.factor(Y) ~ ., data = df_train_log,
```

```
##     ranges = list(cost = c(0.005, 0.05, 0.5), gamma = c(0.1, 0.5,
##         1)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.5
##
## Number of Support Vectors:  1881
##
##  ( 1011 870 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
cat("Best cost for gaussian SVM on log dataset is:", 0.5, "gamma:",0.1,'\n')
```

```
## Best cost for gaussian SVM on log dataset is: 0.5 gamma: 0.1
```

```
gaussian_svm_log_train_error = mean(predict(bestmod.gaussian.log, df_train_log)!=
                                      df_train_log$Y)
gaussian_svm_log_test_error = mean(predict(bestmod.gaussian.log, df_test_log)!=
                                     df_test_log$Y)
cat("gaussian_svm_log_train_error:", gaussian_svm_log_train_error, '\n')
```

```
## gaussian_svm_log_train_error: 0.01696021
```

```
cat("gaussian_svm_log_test_error:", gaussian_svm_log_test_error, '\n')
```

```
## gaussian_svm_log_test_error: 0.06066536
```

```
#linear svm on log set
tune.linear.log=tune(svm, as.factor(Y)~., data=df_train_log,
                     kernel ="linear",
                     ranges=list(cost=c(0.005, 0.05, 0.5)))
# Check the selection results
summary(tune.linear.log)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  0.005
##
```

```
## - best performance: 0.06067254
##
## - Detailed performance results:
##    cost      error dispersion
## 1 0.005 0.06067254 0.01147780
## 2 0.050 0.06230334 0.01087308
## 3 0.500 0.06426518 0.01294339
```

```
# Choose the best model (classifier with optimal C)
bestmod.linear.log =tune.linear.log$best.model
# Check the classifier
summary(bestmod.linear.log)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = as.factor(Y) ~ ., data = df_train_log,
##     ranges = list(cost = c(0.005, 0.05, 0.5)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.005
##
## Number of Support Vectors:  764
##
##  ( 383 381 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
cat("Best cost for linear SVM on log dataset is:", 0.005,'\n')
```

```
## Best cost for linear SVM on log dataset is: 0.005
```

```
linear_svm_log_train_error = mean(predict(bestmod.linear.log, df_train_log)!=
                                    df_train_log$Y)
linear_svm_log_test_error = mean(predict(bestmod.linear.log, df_test_log)!=
                                   df_test_log$Y)
cat("linear_svm_log_train_error:", linear_svm_log_train_error, '\n')
```

```
## linear_svm_log_train_error: 0.05936073
```

```
cat("linear_svm_log_test_error:", linear_svm_log_test_error, '\n')
```

```
## linear_svm_log_test_error: 0.06066536
```

```r
#non linear on I set
tune.gaussian.I=tune(svm, as.factor(Y)~., data=df_train_I,
                     kernel ="radial",
                     ranges=list(cost=c(0.005, 0.05, 0.5),
                                 gamma=c(0.1,0.5,1)))
# Check the selection results
summary(tune.gaussian.I)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   0.5   0.5
##
## - best performance: 0.04859701
##
## - Detailed performance results:
##    cost gamma      error dispersion
## 1 0.005   0.1 0.30561623 0.02138926
## 2 0.050   0.1 0.07926167 0.01707658
## 3 0.500   0.1 0.05675523 0.01548598
## 4 0.005   0.5 0.39726320 0.02132422
## 5 0.050   0.5 0.33430521 0.02429868
## 6 0.500   0.5 0.04859701 0.01361423
## 7 0.005   1.0 0.39726320 0.02132422
## 8 0.050   1.0 0.39726320 0.02132422
## 9 0.500   1.0 0.12131741 0.01754943
```

```r
# Choose the best model (classifier with optimal C)
bestmod.gaussian.I =tune.gaussian.I$best.model
# Check the classifier
summary(bestmod.gaussian.I)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = as.factor(Y) ~ ., data = df_train_I,
##     ranges = list(cost = c(0.005, 0.05, 0.5), gamma = c(0.1, 0.5,
##         1)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.5
##
## Number of Support Vectors:  1706
##
##  ( 885 821 )
##
```

```
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
cat("Best cost for gaussian SVM on discretized dataset is:", 0.5, "gamma:",0.5,'\n')
```

```
## Best cost for gaussian SVM on discretized dataset is: 0.5 gamma: 0.5
```

```
gaussian_svm_I_train_error = mean(predict(bestmod.gaussian.I, df_train_I)!=
                                     df_train_I$Y)
gaussian_svm_I_test_error = mean(predict(bestmod.gaussian.I, df_test_I)!=
                                    df_test_I$Y)
cat("gaussian_svm_I_train_error:", gaussian_svm_I_train_error, '\n')
```

```
## gaussian_svm_I_train_error: 0.02217873
```

```
cat("gaussian_svm_I_test_error:", gaussian_svm_I_test_error, '\n')
```

```
## gaussian_svm_I_test_error: 0.0541422
```

```
#linear svm on I set
tune.linear.I=tune(svm, as.factor(Y)~., data=df_train_I,
                     kernel ="linear",
                     ranges=list(cost=c(0.005, 0.05, 0.5)))
# Check the selection results
summary(tune.linear.I)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.5
##
## - best performance: 0.06816121
##
## - Detailed performance results:
##    cost      error dispersion
## 1 0.005 0.08578272 0.01317338
## 2 0.050 0.07077452 0.01278327
## 3 0.500 0.06816121 0.01815947
```

```
# Choose the best model (classifier with optimal C)
bestmod.linear.I =tune.linear.I$best.model
# Check the classifier
summary(bestmod.linear.I)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = as.factor(Y) ~ ., data = df_train_I,
##     ranges = list(cost = c(0.005, 0.05, 0.5)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.5
##
## Number of Support Vectors:  582
##
##  ( 294 288 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
cat("Best cost for linear SVM on discretized dataset is:", 0.5,'\n')
```

```
## Best cost for linear SVM on discretized dataset is: 0.5
```

```
linear_svm_I_train_error = mean(predict(bestmod.linear.I, df_train_I)!=
                                    df_train_I$Y)
linear_svm_I_test_error = mean(predict(bestmod.linear.I, df_test_I)!=
                                   df_test_I$Y)
cat("linear_svm_I_train_error:", linear_svm_I_train_error, '\n')
```

```
## linear_svm_I_train_error: 0.06001305
```

```
cat("linear_svm_I_test_error:", linear_svm_I_test_error, '\n')
```

```
## linear_svm_I_test_error: 0.07371168
```

#e

```
library(tree)
library(randomForest)
tree_std <- tree(as.factor(Y)~., data = df_train_std)
#cv pruned tree
cross_val =cv.tree(tree_std, K=10)
cv_size = cross_val$size[which.min(cross_val$dev)]
tree_prune_std = prune.tree(tree_std, best=cv_size)
summary(tree_prune_std)
```

```
##
## Classification tree:
## tree(formula = as.factor(Y) ~ ., data = df_train_std)
```

```
## Variables actually used in tree construction:
## [1] "x52" "x7"  "x24" "x16" "x23" "x27" "x53" "x55" "x25"
## Number of terminal nodes:  13
## Residual mean deviance:  0.4929 = 1505 / 3053
## Misclassification error rate: 0.08774 = 269 / 3066
```

```
ptree_std_train_error=mean(predict(tree_std, df_train_std, type="class")!=df_train_std$Y)
ptree_std_test_error=mean(predict(tree_std, df_test_std, type="class")!=df_test_std$Y)
#bagging
bag_std = randomForest(as.factor(Y)~., data = df_train_std,
                           mtry=(ncol(df_train_std)-1), importance=TRUE, ntree=100)
bag_std_train_error=mean(predict(bag_std, df_train_std, type="class")!=df_train_std$Y)
bag_std_test_error=mean(predict(bag_std, df_test_std, type="class")!=df_test_std$Y)
cat("error of pruned tree on std train set: ", ptree_std_train_error, '\n')
```

```
## error of pruned tree on std train set:  0.08773646
```

```
cat("error of pruned tree on std test set: ", ptree_std_test_error, '\n')
```

```
## error of pruned tree on std test set:  0.1272016
```

```
cat("error of tree with bagging on std train set: ", bag_std_train_error, '\n')
```

```
## error of tree with bagging on std train set:  0.0003261579
```

```
cat("error of tree with bagging  on std test set: ", bag_std_test_error, '\n')
```

```
## error of tree with bagging  on std test set:  0.04631442
```

```
tree_log <- tree(as.factor(Y)~., data = df_train_log)
#cv pruned tree
cross_val =cv.tree(tree_log, K=10)
cv_size = cross_val$size[which.min(cross_val$dev)]
tree_prune_log = prune.tree(tree_log, best=cv_size)
summary(tree_prune_log)
```

```
##
## Classification tree:
## tree(formula = as.factor(Y) ~ ., data = df_train_log)
## Variables actually used in tree construction:
## [1] "x52" "x7"  "x24" "x16" "x23" "x27" "x53" "x55" "x25"
## Number of terminal nodes:  13
## Residual mean deviance:  0.4929 = 1505 / 3053
## Misclassification error rate: 0.08774 = 269 / 3066
```

```
ptree_log_train_error=mean(predict(tree_log, df_train_log, type="class")!=df_train_log$Y)
ptree_log_test_error=mean(predict(tree_log, df_test_log, type="class")!=df_test_log$Y)
#bagging
bag_log = randomForest(as.factor(Y)~., data = df_train_log,
                           mtry=(ncol(df_train_log)-1), importance=TRUE, ntree=100)
bag_log_train_error=mean(predict(bag_log, df_train_log, type="class")!=df_train_log$Y)
bag_log_test_error=mean(predict(bag_log, df_test_log, type="class")!=df_test_log$Y)
cat("error of pruned tree on log train set: ", ptree_log_train_error, '\n')
```

```
## error of pruned tree on log train set:  0.08773646
```

```r
cat("error of pruned tree on log test set: ", ptree_log_test_error, '\n')
```

```
## error of pruned tree on log test set:  0.09067189
```

```r
cat("error of tree with bagging on log train set: ", bag_log_train_error, '\n')
```

```
## error of tree with bagging on log train set:  0
```

```r
cat("error of tree with bagging  on log test set: ", bag_log_test_error, '\n')
```

```
## error of tree with bagging  on log test set:  0.03652968
```

```r
tree_I <- tree(as.factor(Y)~., data = df_train_I)
#cv pruned tree
cross_val =cv.tree(tree_I, K=10)
cv_size = cross_val$size[which.min(cross_val$dev)]
tree_prune_I = prune.tree(tree_I, best=cv_size)
summary(tree_prune_I)
```

```
##
## Classification tree:
## tree(formula = as.factor(Y) ~ ., data = df_train_I)
## Variables actually used in tree construction:
## [1] "x7"  "x53" "x16" "x25" "x52" "x45"
## Number of terminal nodes:  8
## Residual mean deviance:  0.5656 = 1730 / 3058
## Misclassification error rate: 0.1171 = 359 / 3066
```

```r
ptree_I_train_error=mean(predict(tree_I, df_train_I, type="class")!=df_train_I$Y)
ptree_I_test_error=mean(predict(tree_I, df_test_I, type="class")!=df_test_I$Y)
#bagging
bag_I = randomForest(as.factor(Y)~., data = df_train_I,
                           mtry=(ncol(df_train_I)-1), importance=TRUE, ntree=100)
bag_I_train_error=mean(predict(bag_I, df_train_I, type="class")!=df_train_I$Y)
bag_I_test_error=mean(predict(bag_I, df_test_I, type="class")!=df_test_I$Y)
cat("error of pruned tree on discretized train set: ", ptree_I_train_error, '\n')
```

```
## error of pruned tree on discretized train set:  0.1170907
```

```r
cat("error of pruned tree on discretized test set: ", ptree_I_test_error, '\n')
```

```
## error of pruned tree on discretized test set:  0.1226354
```

```r
cat("error of tree with bagging on discretized train set: ", bag_I_train_error, '\n')
```

```
## error of tree with bagging on discretized train set:  0.006849315
```

```
cat("error of tree with bagging on discretized test set: ", bag_I_test_error, '\n')
```

```
## error of tree with bagging on discretized test set:  0.04631442
```

#tabulate classification errors for different models and dataset

```
library("knitr")
err_T <- matrix(nrow = 7, ncol = 6)
colnames(err_T) <- c("Std_train","Std_test","Log_train","Log_test",
                         "Discretized_train", "Discretized_test")
row.names(err_T) <- c("Logistic_Regression","LDA","QDA","Linear_SVM",
                     "Gaussian_SVM","Pruned_Tree","Bagging")

err_T[1,1] = mean(pred_std_train !=  df_train_std$Y)
err_T[1,2] = mean(pred_std_test !=  df_test_std$Y)
err_T[1,3] = mean(pred_log_train !=  df_train_log$Y)
err_T[1,4] = mean(pred_log_test !=  df_test_log$Y)
err_T[1,5] = mean(pred_I_train !=  df_train_I$Y)
err_T[1,6] = mean(pred_I_test !=  df_test_I$Y)

err_T[2,1] = lda_std_train_error
err_T[2,2] = lda_std_test_error
err_T[2,3] = lda_log_train_error
err_T[2,4] = lda_log_test_error
err_T[2,5] = NA
err_T[2,6] = NA

err_T[3,1] = qda_std_train_error
err_T[3,2] = qda_std_test_error
err_T[3,3] = qda_log_train_error
err_T[3,4] = qda_log_test_error
err_T[3,5] = NA
err_T[3,6] = NA

err_T[4,1] = linear_svm_std_train_error
err_T[4,2] = linear_svm_std_test_error
err_T[4,3] = linear_svm_log_train_error
err_T[4,4] = linear_svm_log_test_error
err_T[4,5] = linear_svm_I_train_error
err_T[4,6] = linear_svm_I_test_error

err_T[5,1] = gaussian_svm_std_train_error
err_T[5,2] = gaussian_svm_std_test_error
err_T[5,3] = gaussian_svm_log_train_error
err_T[5,4] = gaussian_svm_log_test_error
err_T[5,5] = gaussian_svm_I_train_error
err_T[5,6] = gaussian_svm_I_test_error

err_T[6,1] = ptree_std_train_error
err_T[6,2] = ptree_std_test_error
err_T[6,3] = ptree_log_train_error
err_T[6,4] = ptree_log_test_error
err_T[6,5] = ptree_I_train_error
```

```r
err_T[6,6] = ptree_I_test_error

err_T[7,1] = bag_std_train_error
err_T[7,2] = bag_std_test_error
err_T[7,3] = bag_log_train_error
err_T[7,4] = bag_log_test_error
err_T[7,5] = bag_I_train_error
err_T[7,6] = bag_I_test_error
kable(err_T)
```

	Std_train	Std_test	Log_train	Log_test	Discretized_train	Discretized_test
Logistic_Regression	0.0717547	0.0704501	0.0577299	0.0567515	0.0570776	0.0808871
LDA	0.1017613	0.1030659	0.0603392	0.0652316	NA	NA
QDA	0.1787345	0.1748206	0.1588389	0.1572081	NA	NA
Linear_SVM	0.0662100	0.0684932	0.0593607	0.0606654	0.0600130	0.0737117
Gaussian_SVM	0.0306588	0.0776256	0.0169602	0.0606654	0.0221787	0.0541422
Pruned_Tree	0.0877365	0.1272016	0.0877365	0.0906719	0.1170907	0.1226354
Bagging	0.0003262	0.0463144	0.0000000	0.0365297	0.0068493	0.0463144

Based on the Table of Classification errors, comparatively, tree based classifiers with bagging have best overall performance on three versions of transformed data sets.

In conclusion, we recommend performing the classification task with tree based classifiers with bagging because the classification errors on three versions of transformed data set are smallest among all models applied.