Final Project

Yiwei Cao, Ruibo Zhang, Jianan Liu

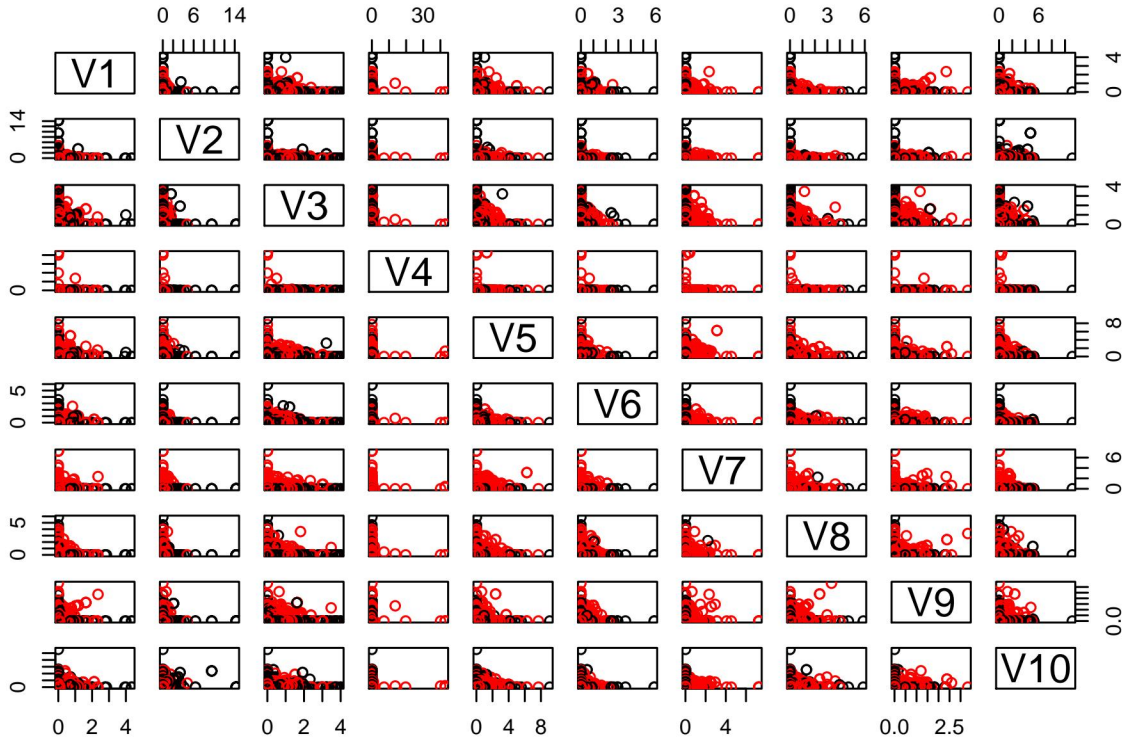
6/5/2020

Yiwei contributed to question (a) and (b), Ruibo contributed to question (c), and Jianan contributed to question (d). We worked together for the final question

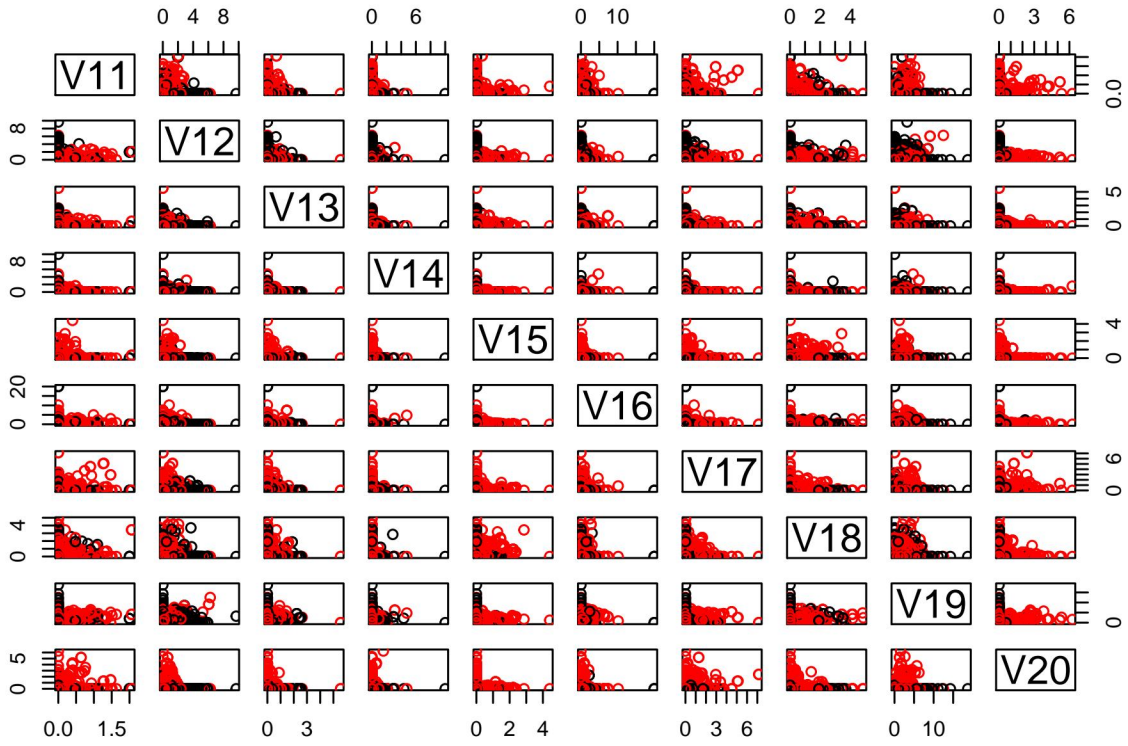
Question 1 The data set contains a training set of size 3065, and a test set of size 1536. One can imagine performing several kinds of preprocessing to this data. Try each of the following separately: 1) Standardize the columns so that they all have zero mean and unit variance; 2) Transform the features using log(xij+1); 3) Discretize each feature using I(xij>0). (a)For each version of the data, visualize it using the tools introduced in the class.

```
setwd("/Users/caoyiwei/Desktop/MATH\ 189")
test <- read.table(file = 'spam-test.txt', header = FALSE, fill = TRUE,  sep=",")
test1 <- test[,-58]
train <- read.table(file = 'spam-train.txt', header = FALSE, fill = TRUE, sep=",")
train1 <- train[,-58]
```

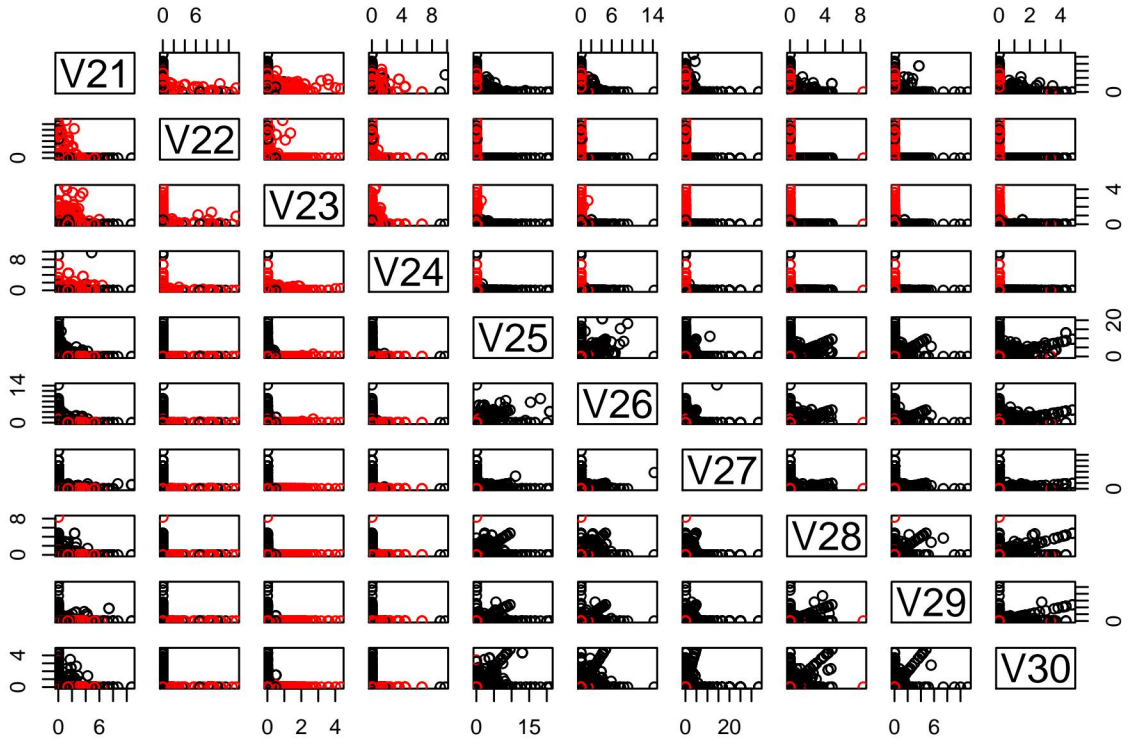
```
pairs(train1[,1:10], col = train[,58]+1)
```



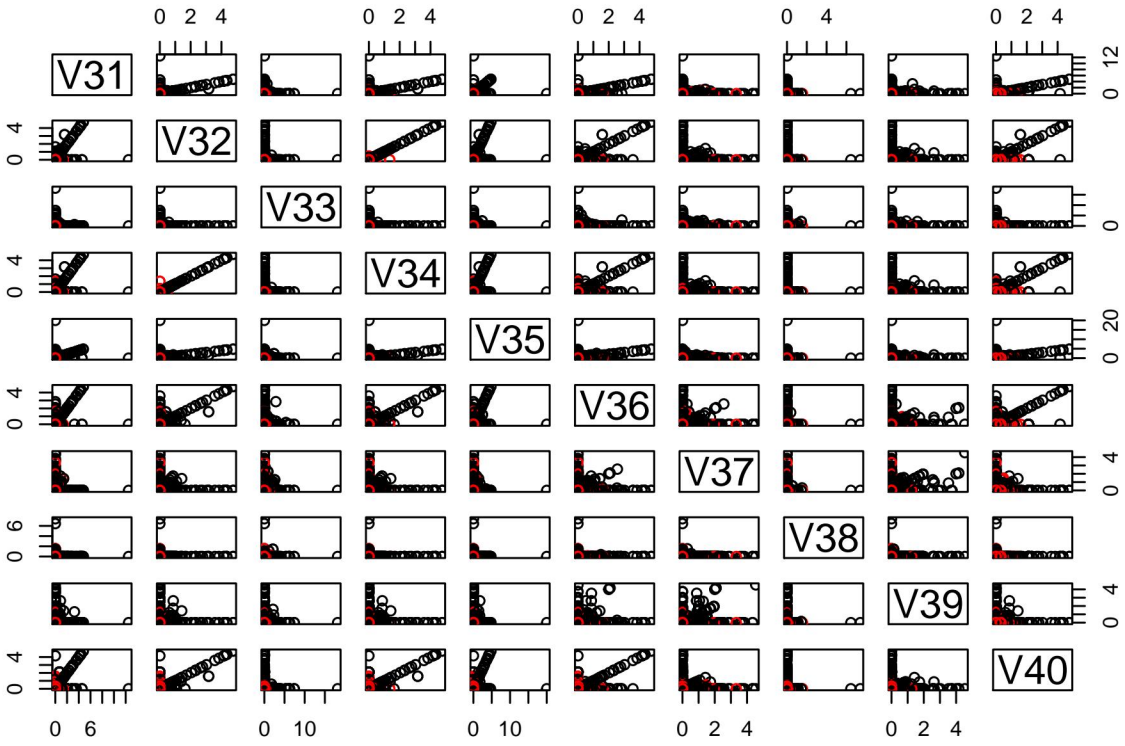

```
pairs(train1[,11:20], col = train[,58]+1)
```



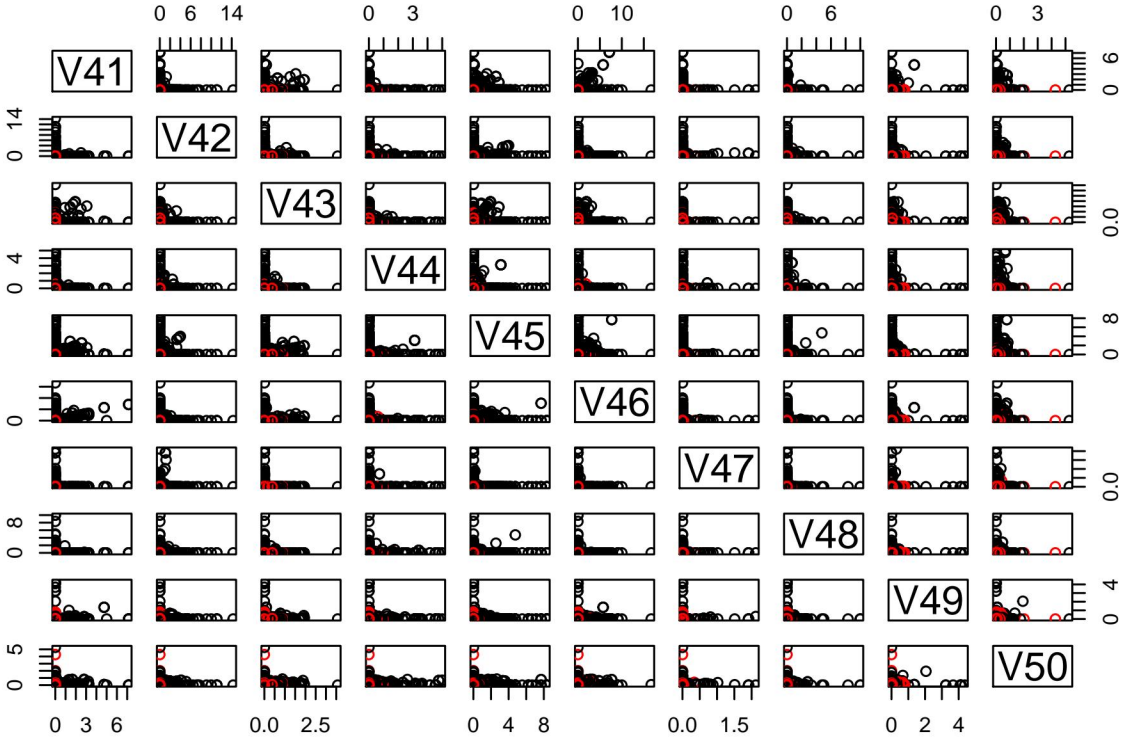

```
pairs(train1[,21:30], col = train[,58]+1)
```



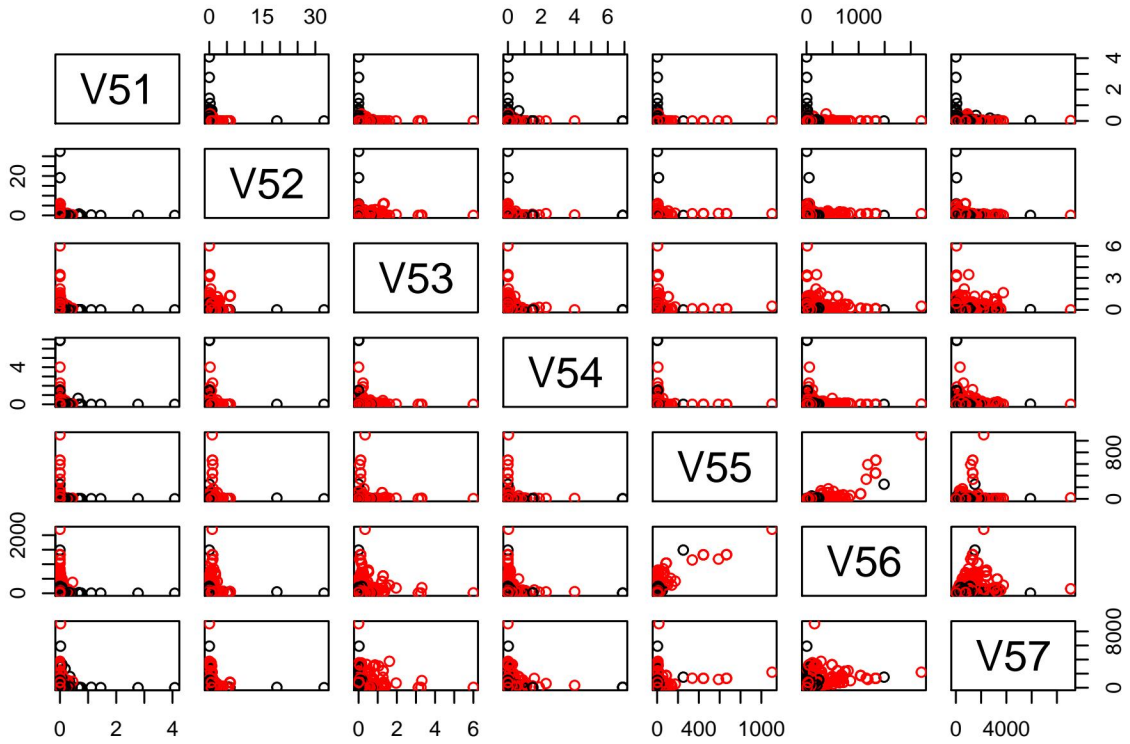

```
pairs(train1[,31:40], col = train[,58]+1)
```



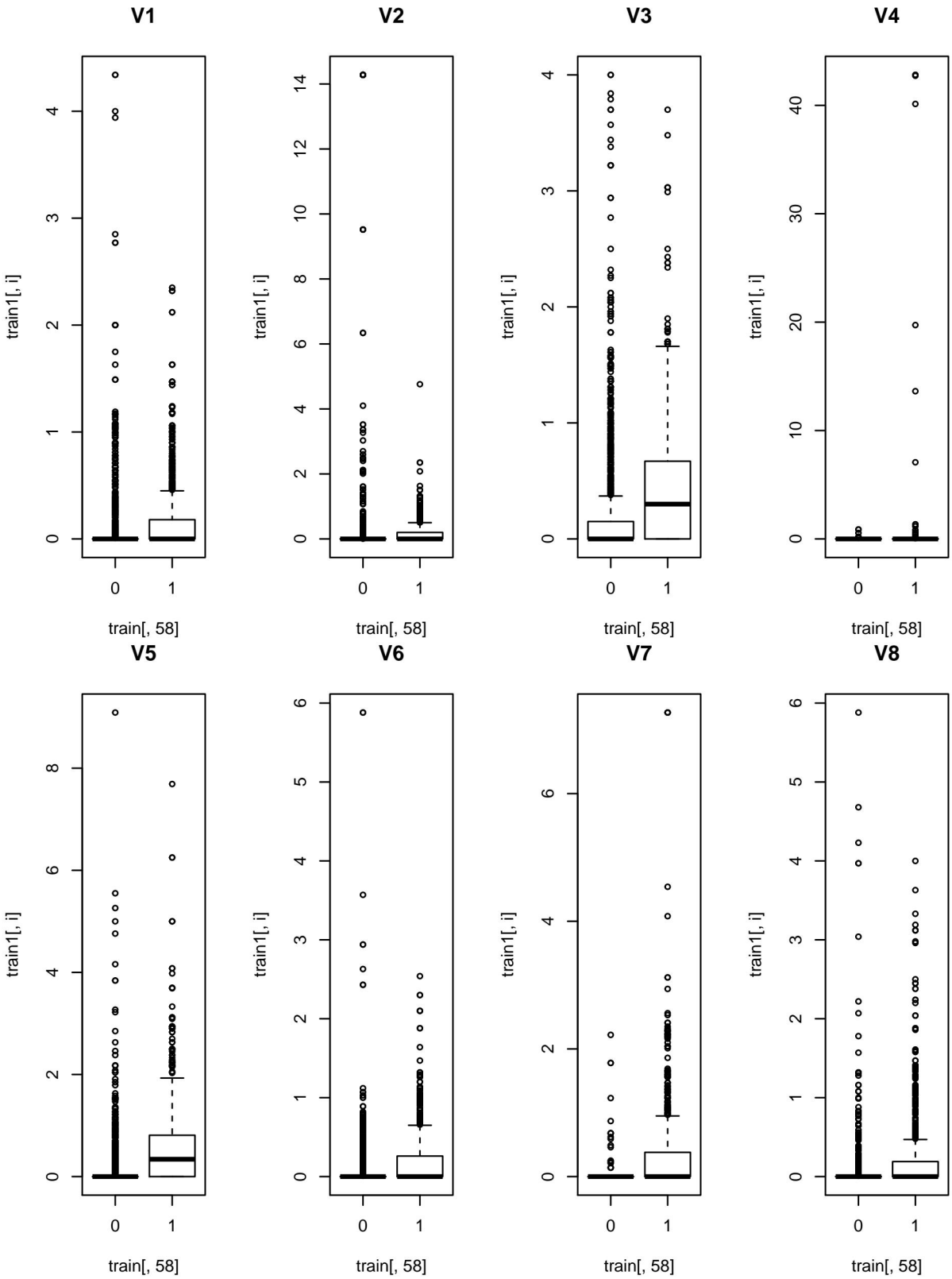

```
pairs(train1[,41:50], col = train[,58]+1)
```

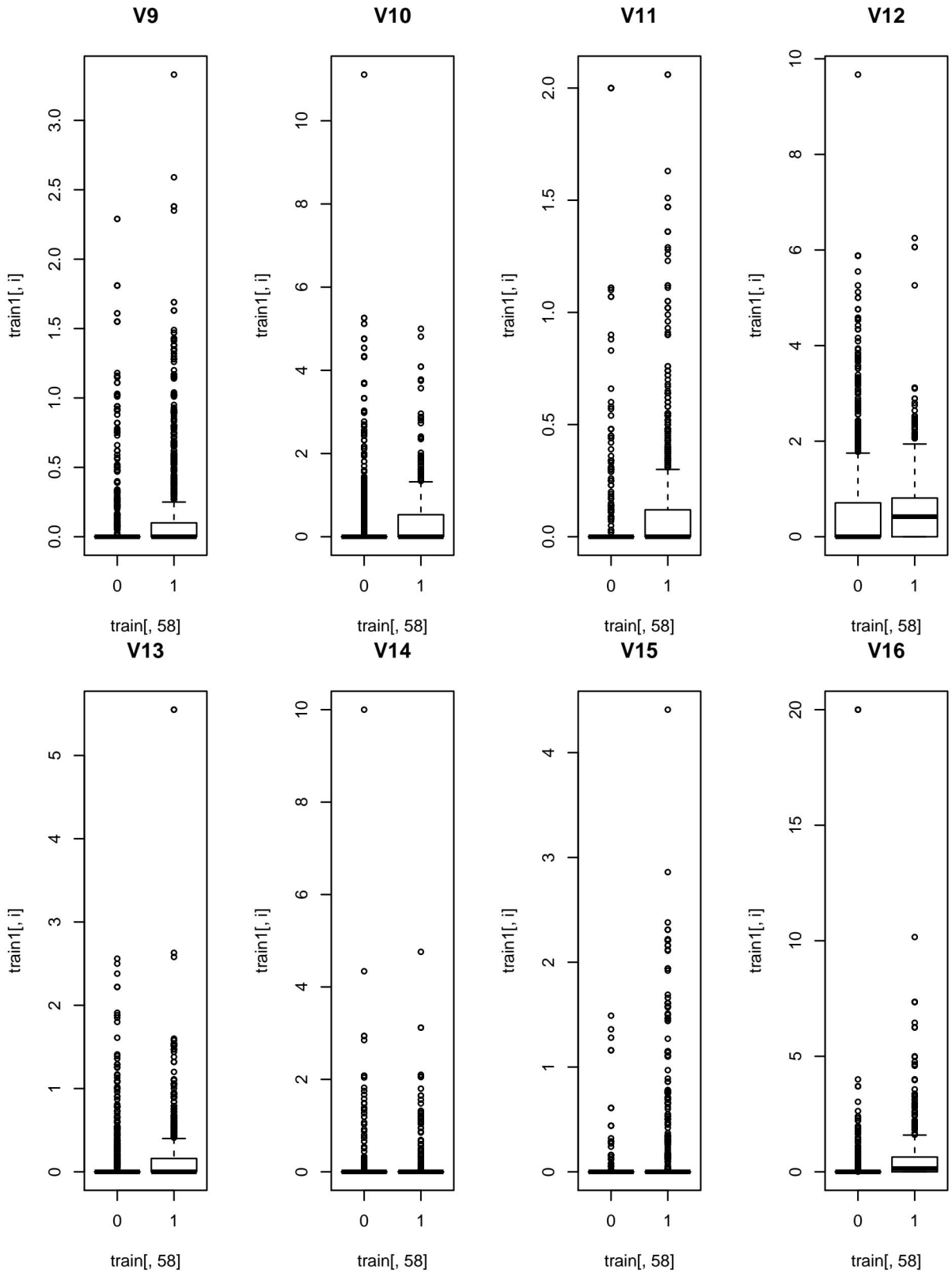


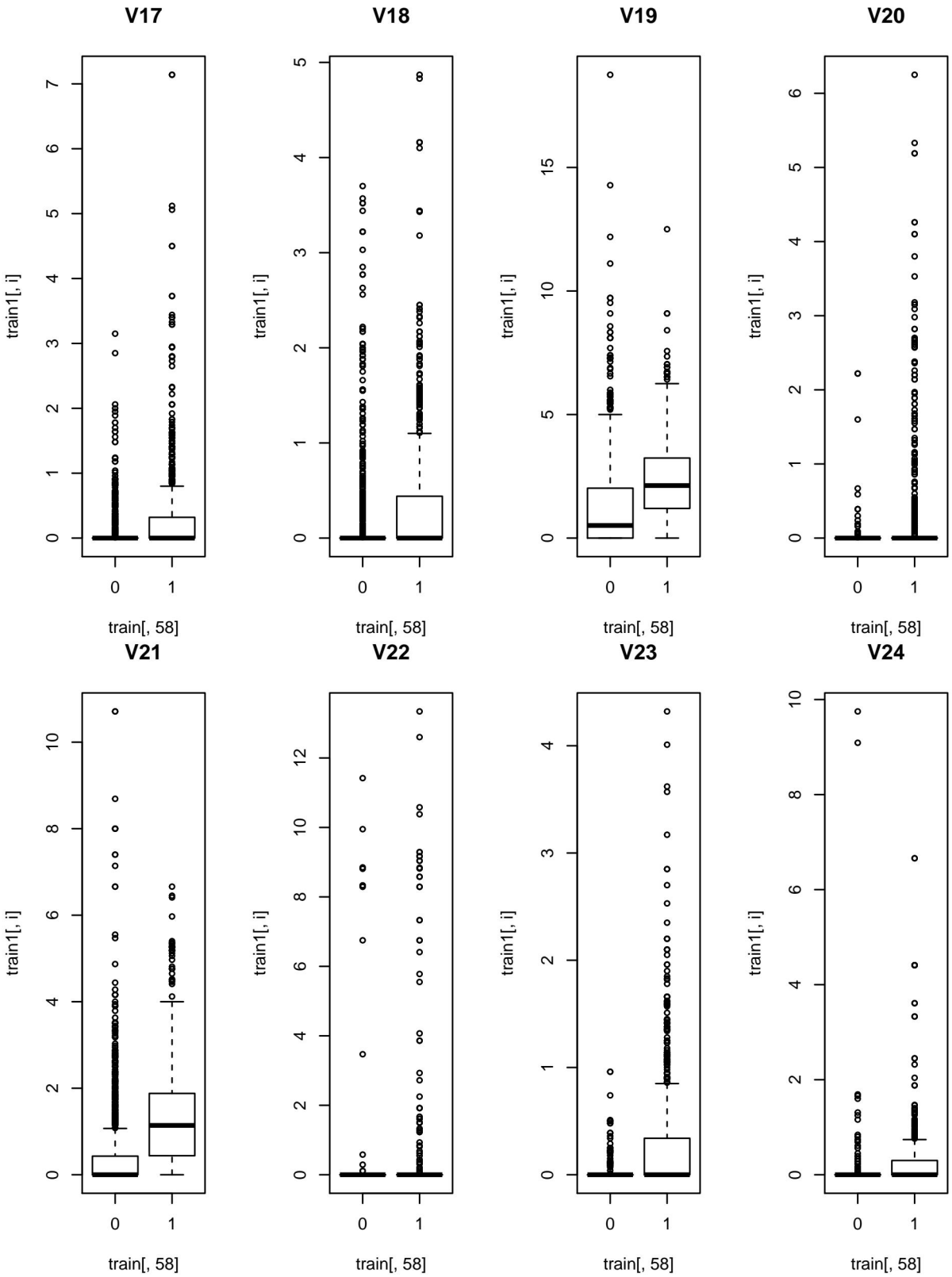

```
pairs(train1[,51:57], col = train[,58]+1)
```

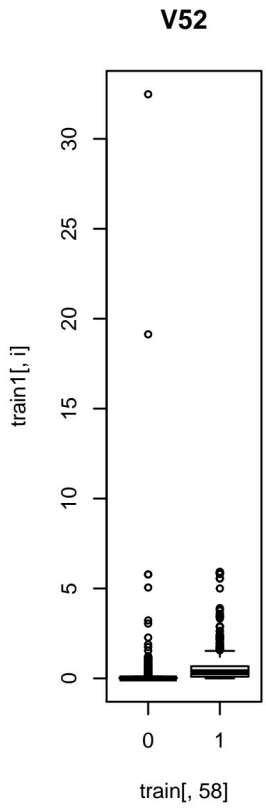


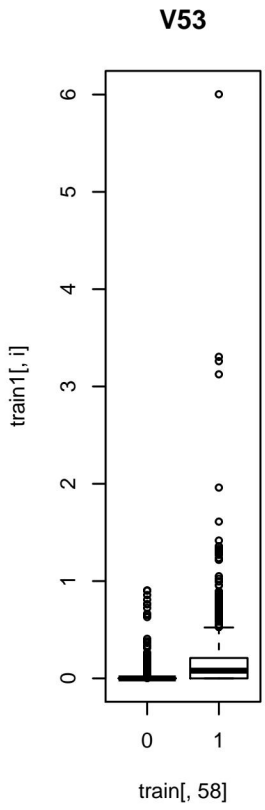

```
#From the pairwise data we selected certain variables that we think are most
#related to response variable to perform box plot
par(mfrow=c(1,4))
for(i in c(1:24,52:57)){
  boxplot(train1[,i]~train[,58],main=names(train1)[i])
}
```

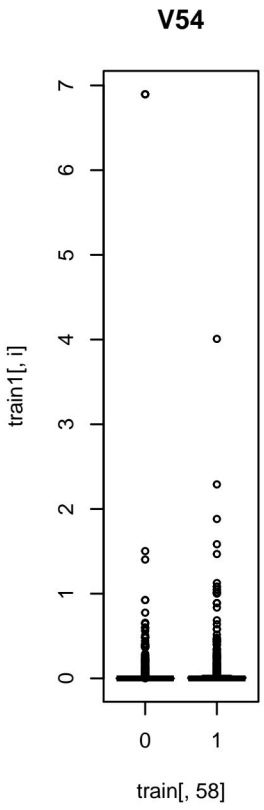

V1
V2
V3
V4
V5
V6
V7
V8
train1[, i]
train[, 58]
0
1

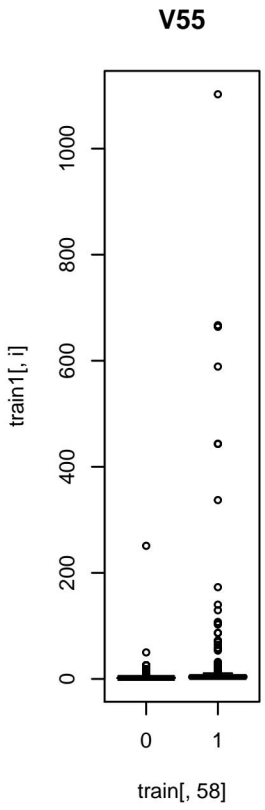

V9
V10
V11
V12
V13
V14
V15
V16
train1[, i]
train[, 58]
0
1


V17
V18
V19
V20
V21
V22
V23
V24
train1[, i]
train[, 58]
0
1

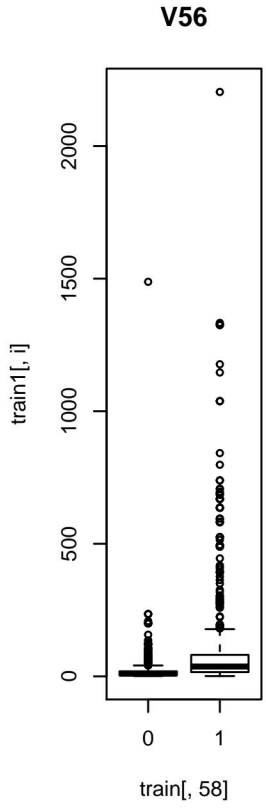


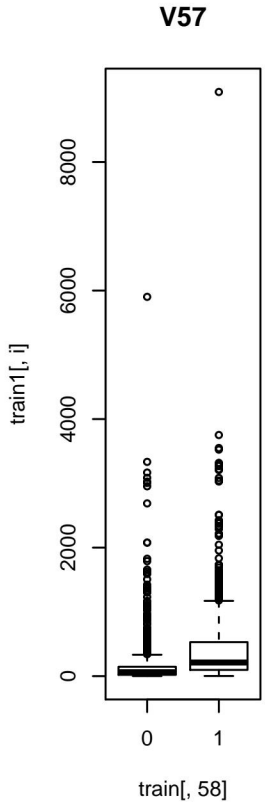




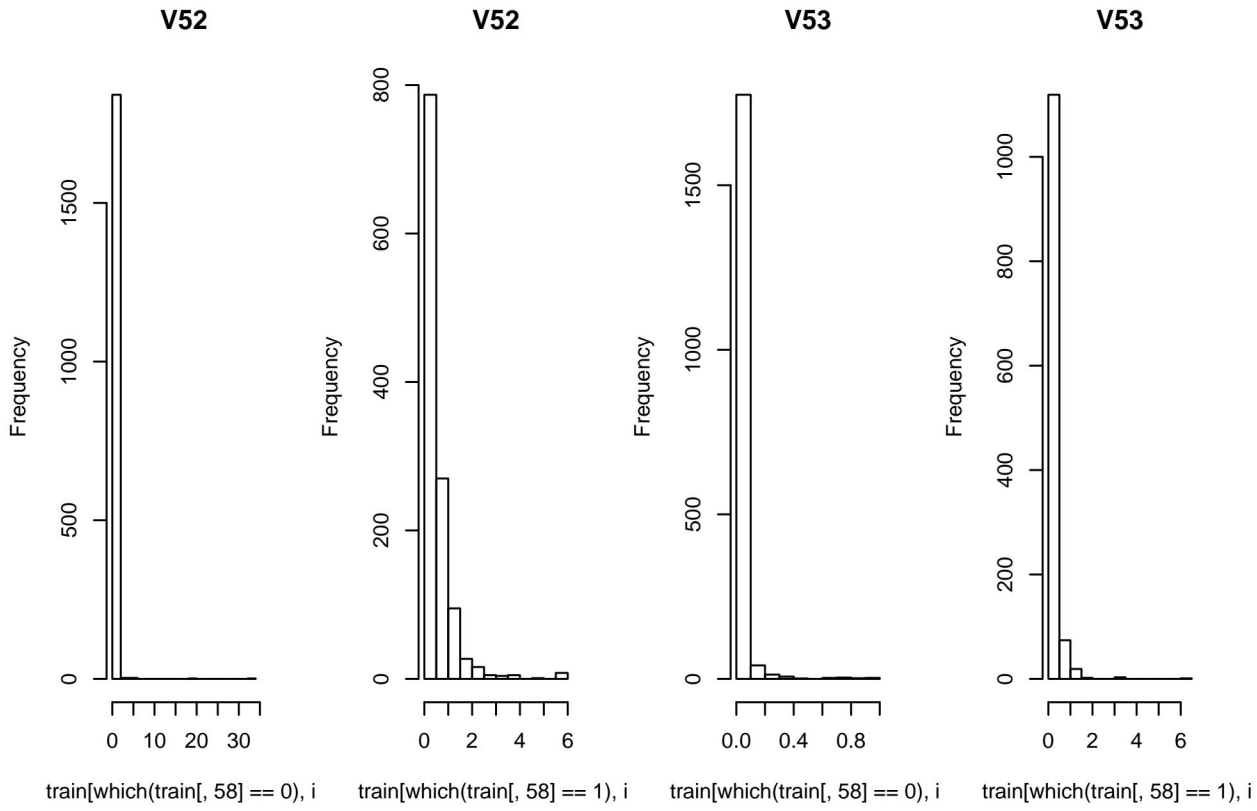


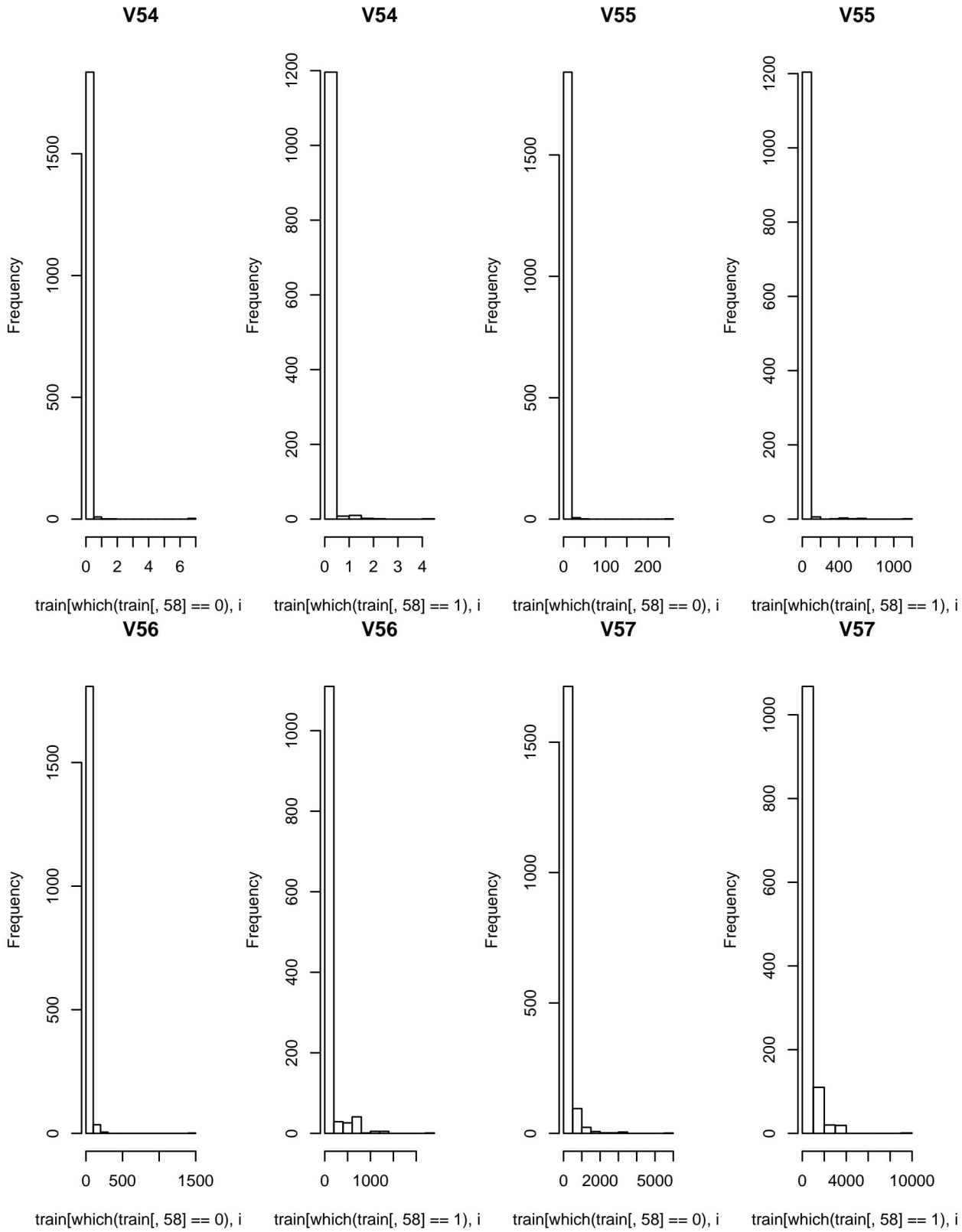

```
par(mfrow=c(1,4))
```





```r
for(i in c(52:57)) {
  hist(train[which(train[,58]==0),i], main=names(train1)[i])
  hist(train[which(train[,58]==1),i], main=names(train1)[i])
}
```

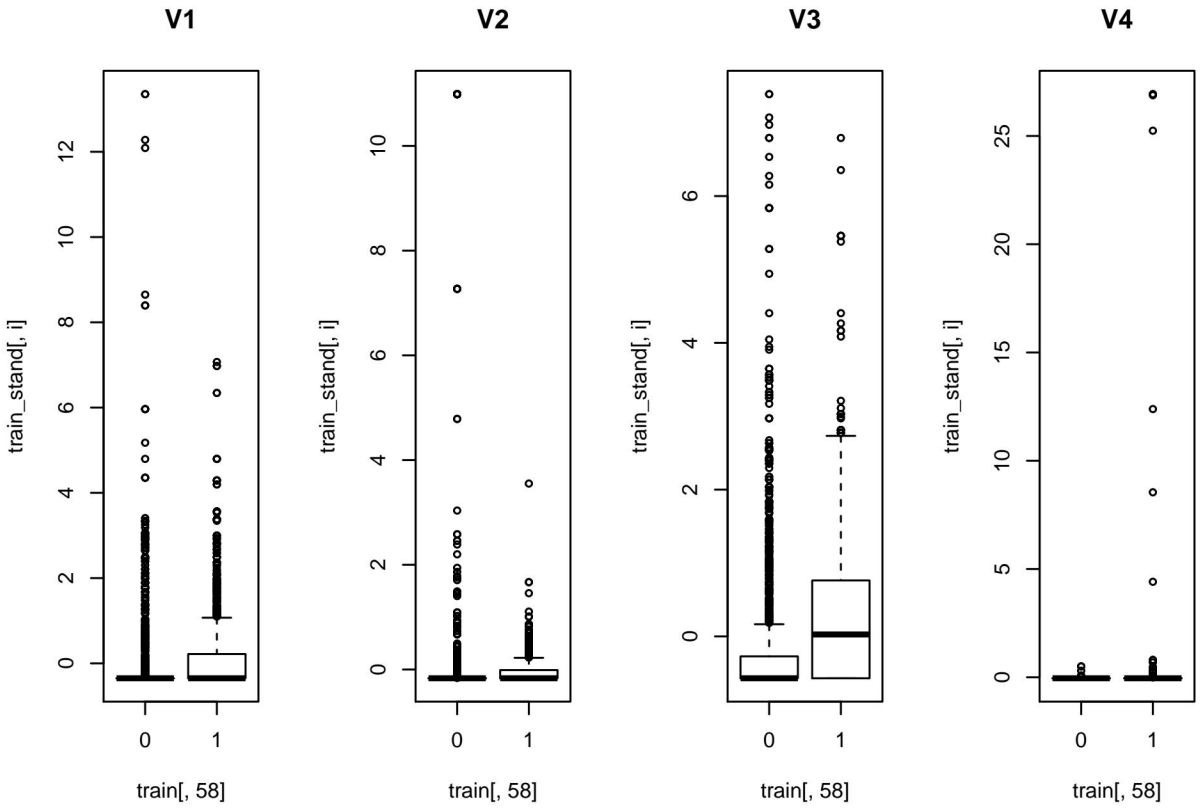


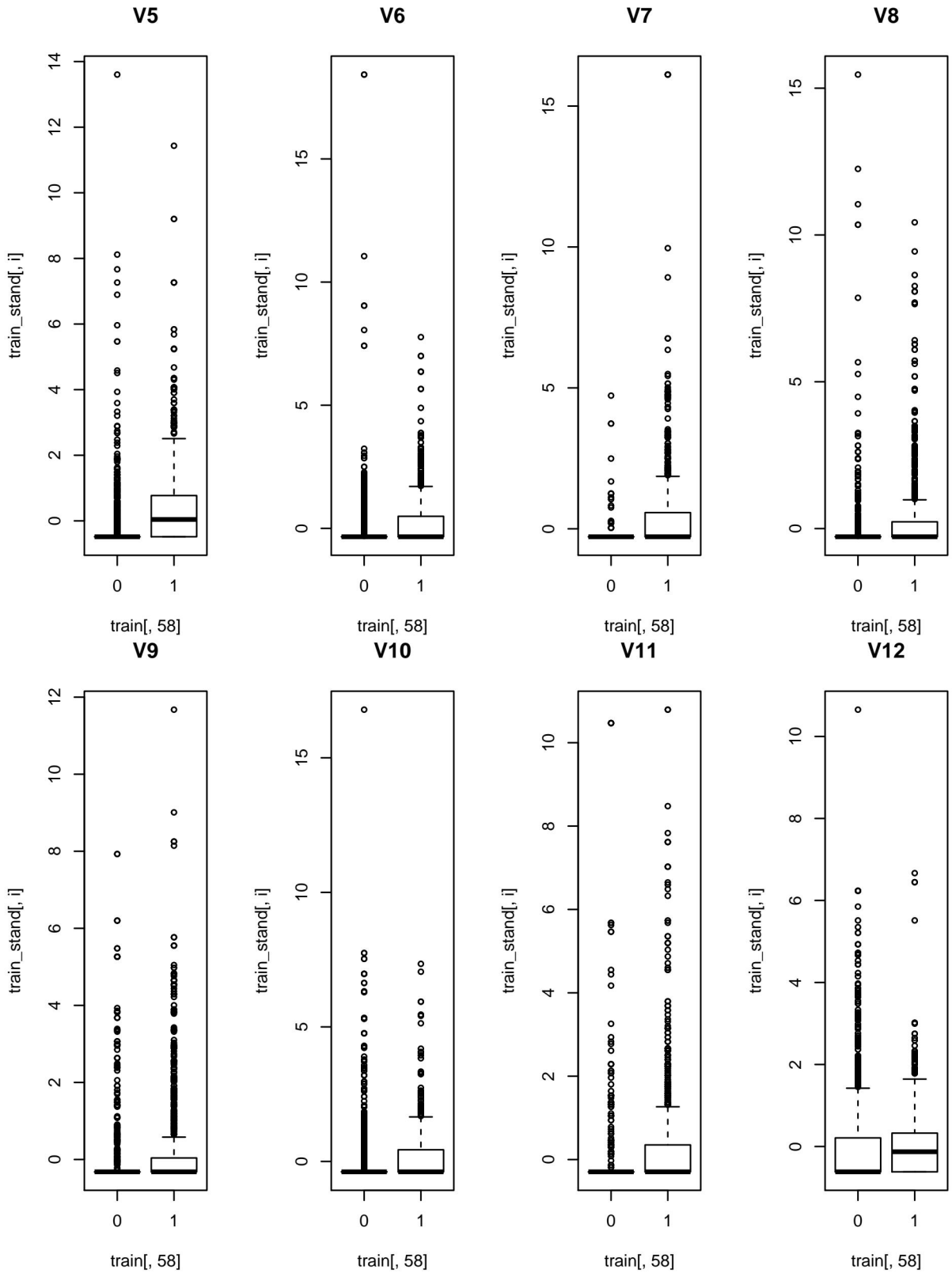


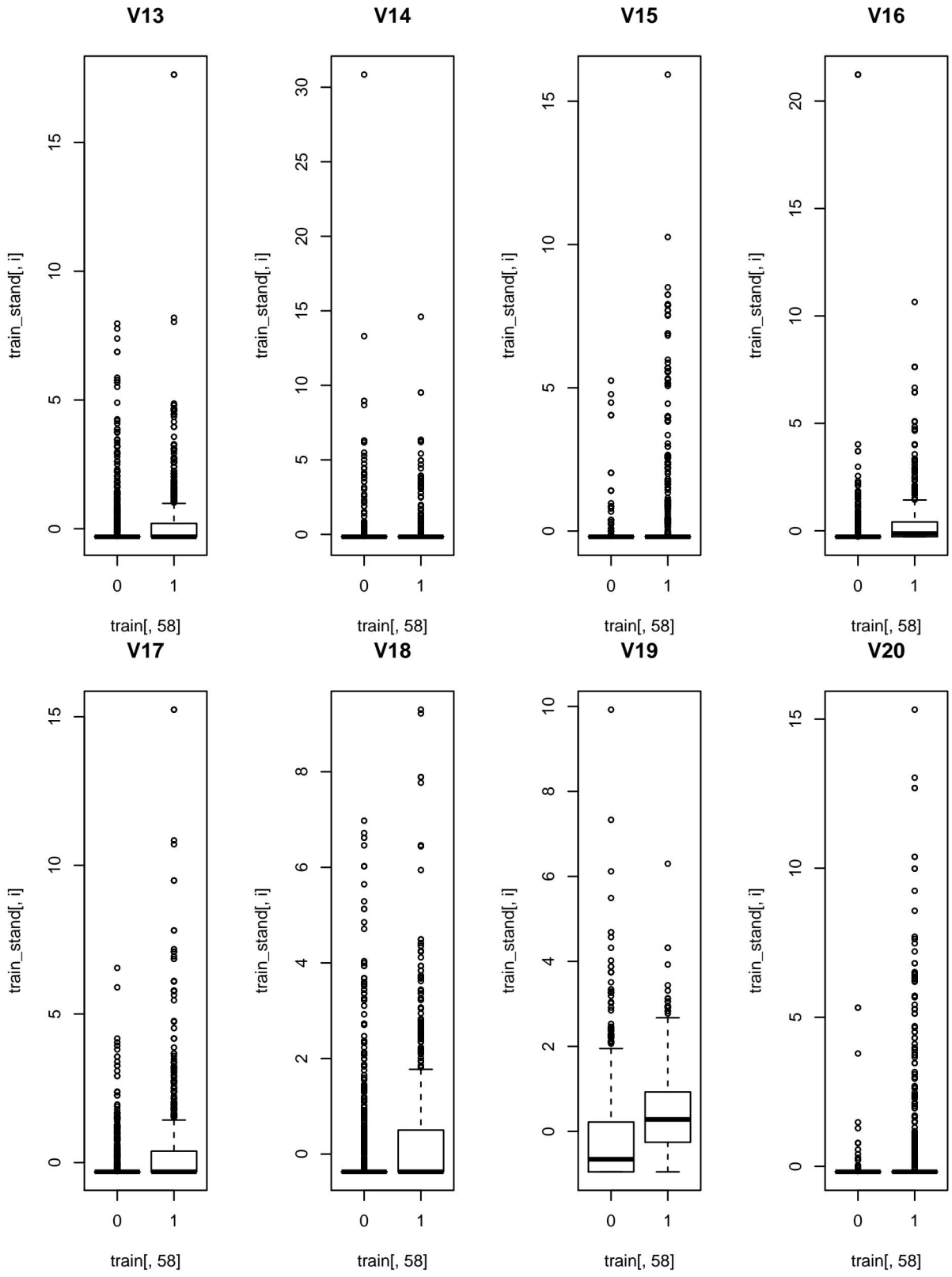
From data, we notice the spam (y=1), not spam(y=0). From the box plot, we found that if the data is not spam, in most cases, the value of data in each variable tend to be smaller compare to email that is spam. Noticed in paired diagram that V1-V24 and V52-V57 have

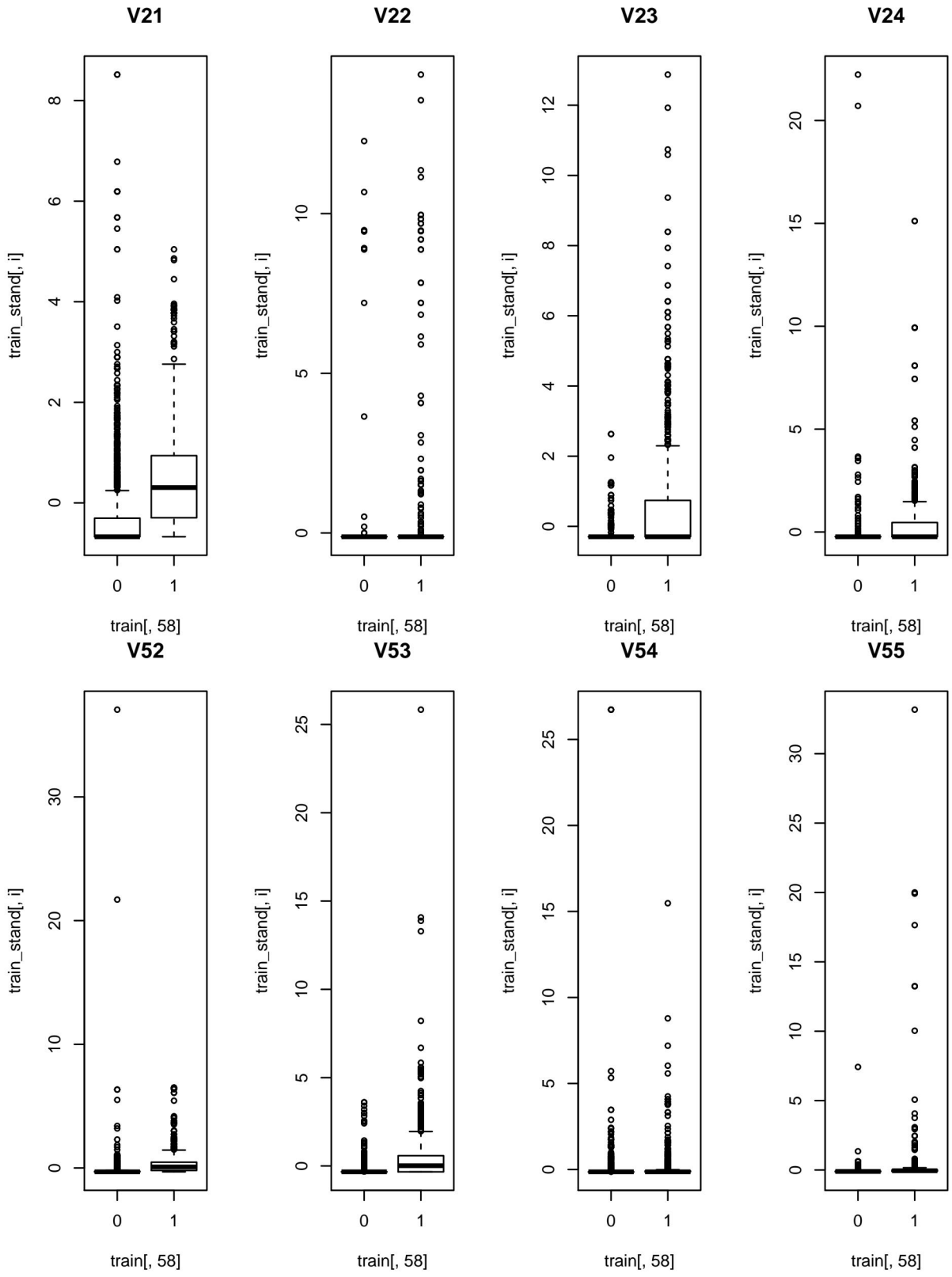
more separation in spam or not data, so the guess is these variables are more related to check if the emails are spam or not.

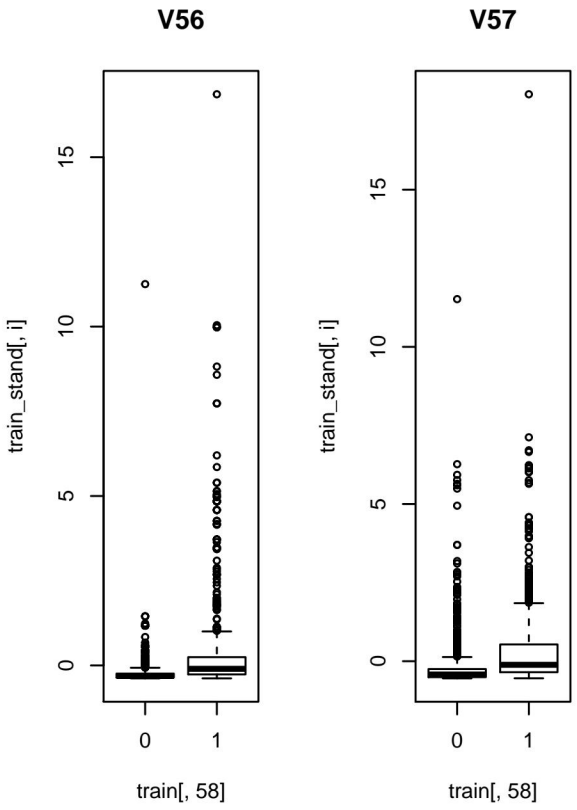
```
library('standardize')
train_stand <- scale(train1,center = TRUE, scale = TRUE)
test_stand <- scale(test1,center = TRUE, scale = TRUE)
par(mfrow=c(1,4))
for(i in c(1:24,52:57)){
  boxplot(train_stand[,i]~train[,58],main=names(train1)[i])
}
```




V5
V6
V7
V8
V9
V10
V11
V12
train_stand[, i]
train[, 58]
0
1

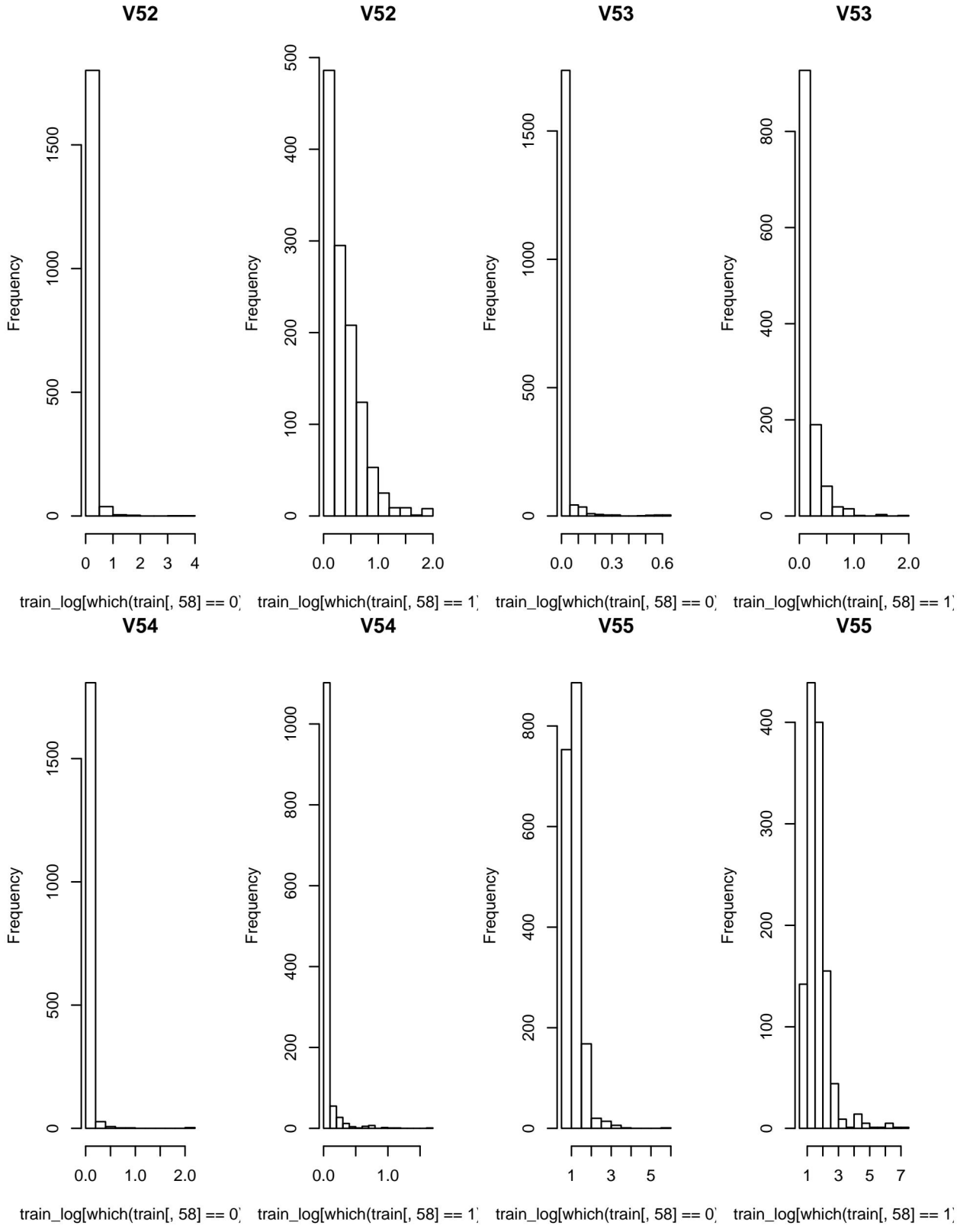

V13
V14
V15
V16
V17
V18
V19
V20
train_stand[, i]
train[, 58]
0
1

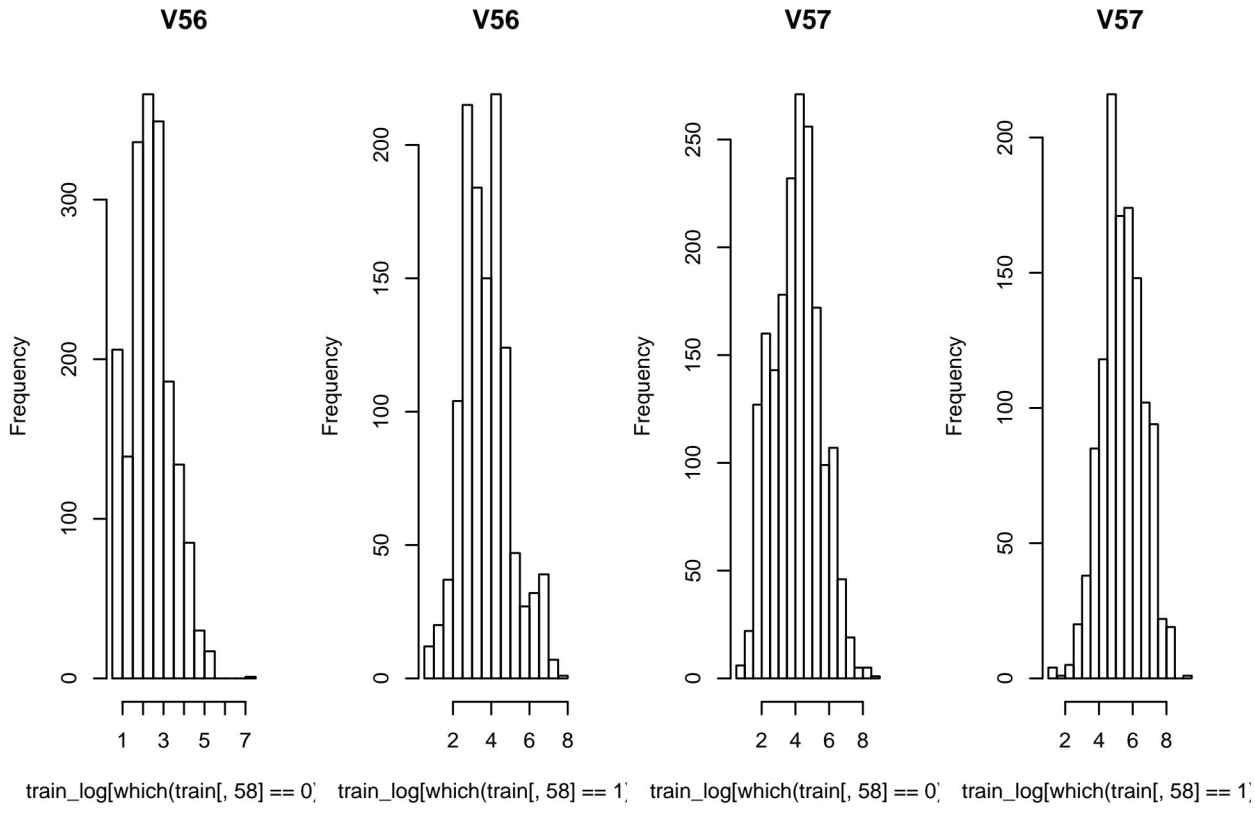

V21
V22
V23
V24
V52
V53
V54
V55
train_stand[, i]
train[, 58]



We derived same conclusion from the boxplot above that the email that is not spam tends to have smaller value as compared with email that is spam

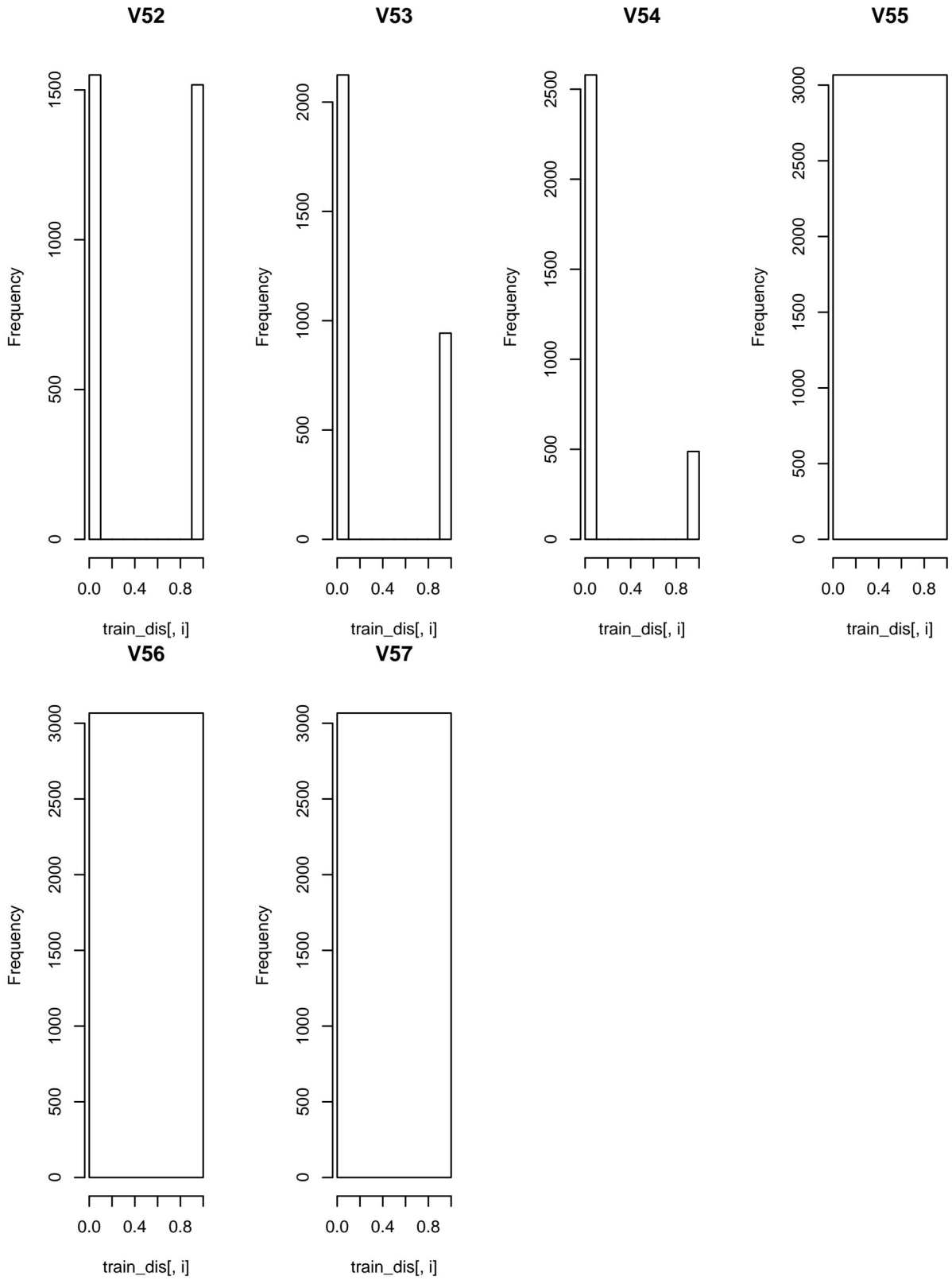
```
train_log <- as.matrix(log(train1+1))
#log1p(train1)
test_log <- as.matrix(log(test1+1))
par(mfrow=c(1,4))
for(i in c(52:57)) {
  hist(train_log[which(train[,58]==0),i], main=names(train1)[i])
  hist(train_log[which(train[,58]==1),i], main=names(train1)[i])
}
```


V52
Frequency
train_log[which(train[, 58] == 0)
V52
Frequency
train_log[which(train[, 58] == 1)
V53
Frequency
train_log[which(train[, 58] == 0)
V53
Frequency
train_log[which(train[, 58] == 1)
V54
Frequency
train_log[which(train[, 58] == 0)
V54
Frequency
train_log[which(train[, 58] == 1)
V55
Frequency
train_log[which(train[, 58] == 0)
V55
Frequency
train_log[which(train[, 58] == 1)



Similar conclusion apply to logged data. However, we found that the histogram for logged data perfomed differently for the V55,V56,V57. The distribution is more like a normal distribution rather than clustered at one value. It seems to us that the log transformed data is the most effective data for us to view and analyze.

```
n <- nrow(train1)
train_dis <- list()
test_dis <- list()
for(i in 1:57){
  train_dis[[i]] <- as.numeric(train1[,i]>0)
  test_dis[[i]] <- as.numeric(test[,i]>0)
}
train_dis = do.call(cbind, train_dis)
test_dis = do.call(cbind,test_dis)
par(mfrow=c(1,4))
for(i in c(52:57)) {
  hist(train_dis[,i], main=names(train1)[i])
}
```



From the historgram, we noticed that some data has especially high spam frequency (V3,V5,V12,V50,V52). V55, V56, V57 all have value of 1, thus they are clustered at 0-1 segment. We still think that the log transformation is the most effective.

(b) For each version of the data, fit a logistic regression model. Interpret the results, and report the classification errors on both the training and test sets. Do any of the 57 features/ predictors appear to be statistically significant? If so, which ones? (Hint: consider this as a multiple testing problem).

```
library(MASS)
train_set <- c(rep(TRUE,3067),rep(FALSE,1534))
A <- as.data.frame(rbind(train_stand,test_stand))
A <- as.data.frame(cbind(A,c(train[,58],test[,58])))
train.set <- A[train_set,]
test.set <- A[!train_set,]
#Logistic regression fit on standardized train data
glm1 <- glm(A$`c(train[, 58], test[, 58])`~.,data = A, family = "binomial",subset = train_set)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
#Apply Bonferroni Correction
which(coef(summary(glm1))[-1,4]<0.05/57)
```

```
##  V5  V7 V16 V17 V20 V21 V23 V25 V27 V45 V46 V52 V53 V56 V57
##   5   7  16  17  20  21  23  25  27  45  46  52  53  56  57
```

```
predicted1 <- predict(glm1,train.set, type = "response")
glm.pred1 <- ifelse(predicted1 > 0.5, 1, 0)
table(glm.pred1,train[,58])
```

```
##
## glm.pred1    0    1
##         0 1762  133
##         1   87 1085
```

```
mean(glm.pred1 != train[,58])
```

```
## [1] 0.07173133
```

```
report_Logistic <-matrix(c('Standardized Train Data','Logistic Regression',mean(glm.pred1 != train[,58
colnames(report_Logistic)<-c("Data Version","Method","Test Error")
#Logistic regression predict on standardized test data
predicted1t <- predict(glm1,test.set, type = "response")
glm.pred1t <- ifelse(predicted1t > 0.5, 1, 0)
table(glm.pred1t,test[,58])
```

```
##
## glm.pred1t   0   1
##          0 877  70
##          1  39 548
```

```
mean(glm.pred1t != test[,58])
```

```
## [1] 0.07105606
```

```
report_Logistic <- rbind(report_Logistic,c('Standardized Test Data','Logistic Regression', mean(glm.pr

#Logistic regression on log train data
#The last three columns are linearly correlated, we have to delete them otherwise the logistic regress
A <- as.data.frame(rbind(train_log,test_log))
A <- as.data.frame(cbind(A,c(train[,58],test[,58])))
train.set <- A[train_set,]
test.set <- A[!train_set,]
glm2 <- glm(A$`c(train[, 58], test[, 58])`~.,data = A,family = "binomial", subset = train_set)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
which(coef(summary(glm2))[-1,4]<0.05/57)
```

```
##  V5  V7 V16 V17 V21 V23 V25 V27 V45 V46 V52 V53 V57
##   5   7  16  17  21  23  25  27  45  46  52  53  57
```

```
predicted2 <- predict(glm2,train.set, type = "response")
glm.pred2 <- ifelse(predicted2 > 0.5, 1, 0)
table(glm.pred2,train[,58])
```

```
##
## glm.pred2    0    1
##         0 1766   94
##         1   83 1124
```

```
mean(glm.pred2 != train[,58])
```

```
## [1] 0.05771112
```

```
report_Logistic <- rbind(report_Logistic,c('Logged Train Data','Logistic Regression', mean(glm.pred2 !
#Logistic regression fit on log test data
predicted2t <- predict(glm2,test.set, type = "response")
glm.pred2t <- ifelse(predicted2t > 0.5, 1, 0)
table(glm.pred2t,test[,58])
```

```
##
## glm.pred2t   0   1
##          0 879  50
##          1  37 568
```

```
mean(glm.pred2t != test[,58])
```

```
## [1] 0.05671447
```

```
report_Logistic <- rbind(report_Logistic,c('Logged Test Data','Logistic Regression', mean(glm.pred2t !

#Logistic regression on discretize train data
#The last three columns of discreted data give all "1"s which means that
#they would be perfectly collinear, and logistic regression would not perform well
#so we cut out the last three columns
A <- as.data.frame(rbind(train_dis[,-c(55:57)],test_dis[,-c(55:57)]))
A <- as.data.frame(cbind(A,c(train[,58],test[,58])))
train.set <- A[train_set,]
test.set <- A[!train_set,]
glm3 <- glm(A$`c(train[, 58], test[, 58])`~.,data = A,family = "binomial",subset = train_set)
which(coef(summary(glm3))[-1,4]<0.05/57)
```

```
##  V5  V7 V11 V13 V16 V17 V21 V23 V24 V25 V27 V28 V37 V42 V45 V46 V52 V53
##   5   7  11  13  16  17  21  23  24  25  27  28  37  42  45  46  52  53
```

```
predicted3 <- predict(glm3,train.set, type = "response")
glm.pred3 <- ifelse(predicted3 > 0.5, 1, 0)
table(glm.pred3,train[,58])
```

```
##
## glm.pred3    0    1
##         0 1779  105
##         1   70 1113
```

```r
mean(glm.pred3 != train[,58])
```

```
## [1] 0.05705902
```

```r
report_Logistic <- rbind(report_Logistic,c('Discretized Train Data','Logistic Regression', mean(glm.pr
#Logistic regression on discretize test data
predicted3t <- predict(glm3,test.set, type = "response")
glm.pred3t <- ifelse(predicted3t > 0.5, 1, 0)
table(glm.pred3t,test[,58])
```

```
##
## glm.pred3t   0   1
##          0 859  67
##          1  57 551
```

```r
mean(glm.pred3t != test[,58])
```

```
## [1] 0.08083442
```

```r
report_Logistic <- rbind(report_Logistic,c('Discretized Test Data','Logistic Regression', mean(glm.pre

report_Logistic
```

```
##      Data Version              Method                Test Error
## [1,] "Standardized Train Data" "Logistic Regression" "0.071731333550701"
## [2,] "Standardized Test Data"  "Logistic Regression" "0.0710560625814863"
## [3,] "Logged Train Data"       "Logistic Regression" "0.0577111183567004"
## [4,] "Logged Test Data"        "Logistic Regression" "0.0567144719687093"
## [5,] "Discretized Train Data"  "Logistic Regression" "0.0570590153244213"
## [6,] "Discretized Test Data"   "Logistic Regression" "0.0808344198174707"
```

```r
report_Logistic[which.min(report_Logistic[,3]),]
```

```
##          Data Version                Method            Test Error
##    "Logged Test Data" "Logistic Regression"  "0.0567144719687093"
```

1) In logistic regression on standardized data, the features with index 5 7 16 17 20 21 23 25 27 45 46 52 53 56 57 are significant after Bonferroni Correction.
2) In logistic regression on logged data, the features with index 5 7 16 17 21 23 25 27 45 46 52 53 57 are significant after Bonferroni Correction.
3) In logistic regression on discretized data, the features with index 5 7 11 13 16 17 21 23 24 25 27 28 37 42 45 46 52 53 are significant after Bonferroni Correction.

Findings: From the report produced, it seems that the log transformed test data has the least test error rate which is 0.0567. This means that the logistic regression model performs best on the log transformed data.

(c)Apply both linear and quadratic discriminant analysis methods to the standardized data, and the log transformed data. What are the classification errors (training and test)?

```r
#standardized data setting
library(MASS)
trainStand <- as.data.frame(cbind(train_stand,train[,58]))
testStand <- as.data.frame(cbind(test_stand,test[,58]))
totalStand01 <-as.data.frame( rbind(trainStand,testStand))
totalStand <-as.data.frame(rbind(train_stand,test_stand))

traink <- c(rep(TRUE,3067),rep(FALSE,1534))
train_set_stand <- as.matrix(totalStand[traink,])
test_set_stand <- as.matrix(totalStand[!traink,])
```

```
truev58train <- totalStand01$V58[traink]
truev58test <- totalStand01$V58[!traink]

#lda method for standardized train data
lda.fit_trainstan <- lda(totalStand01$V58~.,data = totalStand01,subset = traink)
lda.pred_trainstan <- predict(lda.fit_trainstan,as.data.frame(train_set_stand ))
lda.class_trainstan <- lda.pred_trainstan$class
table(predict=lda.class_trainstan,truth =truev58train)
```

```
##        truth
## predict    0    1
##       0 1770  233
##       1   79  985
```

```
mean(lda.class_trainstan!= truev58train)
```

```
## [1] 0.1017281
```

```
#lda method for standarduzed test data

lda.fit_teststan <- lda(totalStand01$V58~.,data = totalStand01,subset = traink)
lda.pred_teststan <- predict(lda.fit_trainstan,as.data.frame(test_set_stand ))
lda.class_teststan <- lda.pred_teststan$class
table(predict=lda.class_teststan,truth =truev58test)
```

```
##        truth
## predict   0   1
##       0 873 115
##       1  43 503
```

```
mean(lda.class_teststan!= truev58test)
```

```
## [1] 0.1029987
```

```
#qda method for standardized train data
qda.fit_trainstan <- qda(totalStand01$V58~.,data = totalStand01,subset = traink)
qda.pred_trainstan <- predict(qda.fit_trainstan,as.data.frame(train_set_stand ))
qda.class_trainstan <- qda.pred_trainstan$class
table(predict=qda.class_trainstan,truth =truev58train)
```

```
##        truth
## predict    0    1
##       0 1369   68
##       1  480 1150
```

```
mean(qda.class_trainstan!= truev58train)
```

```
## [1] 0.1786762
```

```
#qda method for standardized test data,
qda.fit_teststan <- qda(totalStand01$V58~.,data = totalStand01,subset = traink)
qda.pred_teststan <- predict(qda.fit_trainstan,as.data.frame(test_set_stand ))
qda.class_teststan <- qda.pred_teststan$class
table(predict=qda.class_teststan,truth =truev58test)
```

```
##        truth
## predict   0   1
##       0 673  25
```

```
##       1 243 593
```

```
mean(qda.class_teststan!= truev58test)
```

```
## [1] 0.1747066
```

```
#************************************************************
#log data setting

trainLog <- as.data.frame(cbind(train_log,train[,58]))
testLog <- as.data.frame(cbind(test_log,test[,58]))
totalLog01 <-as.data.frame( rbind(trainLog,testLog))
totalLog <-as.data.frame(rbind(train_log,test_log))

traink <- c(rep(TRUE,3067),rep(FALSE,1534))
train_set_log <- as.matrix(totalLog[traink,])
test_set_log <- as.matrix(totalLog[!traink,])
truev58train <- totalLog01$V58[traink]
truev58test <- totalLog01$V58[!traink]

#lda method for logistic train data

lda.fit_trainLog<- lda(totalLog01$V58~.,data = totalLog01,subset = traink)
lda.pred_trainLog <- predict(lda.fit_trainLog,as.data.frame(train_set_log ))
lda.class_trainLog <- lda.pred_trainLog$class
table(predict=lda.class_trainLog,truth =truev58train)
```

```
##        truth
## predict    0    1
##       0 1795  131
##       1   54 1087
```

```
mean(lda.class_trainLog!= truev58train)
```

```
## [1] 0.06031953
```

```
#lda method for logarduzed test data
lda.fit_testLog<- lda(totalLog01$V58~.,data = totalLog01,subset = traink)
lda.pred_testLog <- predict(lda.fit_testLog,as.data.frame(test_set_log ))
lda.class_testLog <- lda.pred_testLog$class
table(predict=lda.class_testLog,truth =truev58test)
```

```
##        truth
## predict   0   1
##       0 885  69
##       1  31 549
```

```
mean(lda.class_testLog!= truev58test)
```

```
## [1] 0.06518905
```

```
#qda method for logardized train data
qda.fit_trainLog<- qda(totalLog01$V58~.,data = totalLog01,subset = traink)
qda.pred_trainLog <- predict(qda.fit_trainLog,as.data.frame(train_set_log ))
qda.class_trainLog <- qda.pred_trainLog$class
table(predict=qda.class_trainLog,truth =truev58train)
```

```
##        truth
## predict    0    1
```

```
##       0 1433   71
##       1  416 1147
```

```
mean(qda.class_trainLog!= truev58train)
```

```
## [1] 0.1587871
```

```
#qda method for logardized test data,
qda.fit_testLog<- qda(totalLog01$V58~.,data = totalLog01,subset = traink)
qda.pred_testLog <- predict(qda.fit_testLog,as.data.frame(test_set_log ))
qda.class_testLog <- qda.pred_testLog$class
table(predict=qda.class_testLog,truth =truev58test)
```

```
##        truth
## predict   0   1
##       0 702  27
##       1 214 591
```

```
mean(qda.class_testLog!= truev58test)
```

```
## [1] 0.1571056
```

```
report_lda<-matrix(c('Standardized Train Data','LDA', mean(lda.class_trainstan!= truev58train)),ncol=3
colnames(report_lda)<-c("Data Version","Method","Test Error")
report_lda <- rbind(report_lda,c('Standardized Test Data','LDA', mean(lda.class_teststan!= truev58test
report_lda <- rbind(report_lda,c('Standardized Train Data','QDA', mean(qda.class_trainstan!= truev58tr
report_lda <- rbind(report_lda,c('Standardized Test Data','QDA', mean(qda.class_teststan!= truev58test
report_lda <- rbind(report_lda,c('Logistic Train Data','LDA', mean(lda.class_trainLog!= truev58train))
report_lda <- rbind(report_lda,c('Logistic Test Data','LDA',mean(lda.class_testLog!= truev58test)))
report_lda <- rbind(report_lda,c('Logistic Train Data','QDA',mean(qda.class_trainLog!= truev58train)))
report_lda <- rbind(report_lda,c('Logistic Test Data','QDA', mean(qda.class_testLog!= truev58test)))
report_lda
```

```
##      Data Version              Method Test Error
## [1,] "Standardized Train Data" "LDA"  "0.10172807303554"
## [2,] "Standardized Test Data"  "LDA"  "0.102998696219035"
## [3,] "Standardized Train Data" "QDA"  "0.178676230844473"
## [4,] "Standardized Test Data"  "QDA"  "0.17470664928292"
## [5,] "Logistic Train Data"     "LDA"  "0.0603195304858168"
## [6,] "Logistic Test Data"      "LDA"  "0.0651890482398957"
## [7,] "Logistic Train Data"     "QDA"  "0.158787088359961"
## [8,] "Logistic Test Data"      "QDA"  "0.157105606258149"
```

(d) Apply linear and nonlinear support vector machine classifiers to each version of the data. What are the classification errors (training and test)?

```
library(e1071)
#On the standardized data,find the best linear model
data.train.stand <- data.frame(x=train_stand,y=as.factor(train[,58]))
svmfit.linear.stand <- svm(y~.,data=data.train.stand,kernel="linear",cost=10,scale=FALSE)
tune.linear.stand <- tune(svm, y~.,data=data.train.stand,kernel="linear", ranges=list(cost=c(8,9)))
bestmod.linear.stand <- tune.linear.stand$best.model

#Calculate test error, standardized
ypred.linear.stand <- predict(bestmod.linear.stand,data.train.stand)
table(predict = ypred.linear.stand,truth = data.train.stand$y)
```

```
##        truth
```

```
## predict    0    1
##       0 1774  116
##       1   75 1102
```

```
test.error.stand <- mean(ypred.linear.stand!=data.train.stand$y)
test.error.stand
```

```
## [1] 0.06227584
```

```
#On the log data,find the best linear model
data.train.log <- data.frame(x=train_log,y=as.factor(train[,58]))
svmfit.linear.log <- svm(y~.,data=data.train.log,kernel="linear",cost=10,scale=FALSE)
tune.linear.log <- tune(svm, y~.,data=data.train.log,kernel="linear", ranges=list(cost=c(8,9)))
bestmod.linear.log <- tune.linear.log$best.model

#Calculate test error, logged
ypred.linear.log <- predict(bestmod.linear.log,data.train.log)
table(predict = ypred.linear.log,truth = data.train.log$y)
```

```
##        truth
## predict    0    1
##       0 1768   87
##       1   81 1131
```

```
test.error.log <- mean(ypred.linear.log!=data.train.log$y)
test.error.log
```

```
## [1] 0.05477665
```

```
#On the Discretized data,find the best linear model
#For discretized data, the last three rows are all '1's, this will cause perfect
#colinear, thus we delete the last three rows
data.train.dis <- data.frame(x=train_dis[,-c(55,56,57)],y=as.factor(train[,58]))
svmfit.linear.dis <- svm(y~.,data=data.train.dis,kernel="linear",cost=8,scale=FALSE)
tune.linear.dis <- tune(svm, y~.,data=data.train.dis,kernel="linear", ranges=list(cost=c(9,10)))
bestmod.linear.dis <- tune.linear.dis$best.model
#Calculate test error, standardized
ypred.linear.dis <- predict(bestmod.linear.dis,data.train.dis)
table(predict = ypred.linear.dis,truth = data.train.dis$y)
```

```
##        truth
## predict    0    1
##       0 1776  105
##       1   73 1113
```

```
test.error.dis <- mean(ypred.linear.dis!=data.train.dis$y)
test.error.dis
```

```
## [1] 0.05803717
```

```
#On the standardized testing data,fit the best linear model to make prediction
data.test.stand <- data.frame(x=test_stand,y=as.factor(test[,58]))
ypred.linear.stand.test <- predict(bestmod.linear.stand,data.test.stand)
#Calculate test error, standardized
table(predict = ypred.linear.stand.test,truth = data.test.stand$y)
```

```
##        truth
## predict   0   1
##       0 875  66
```

```
##       1  41 552
```

```
test.error.stand.test <- mean(ypred.linear.stand.test!=data.test.stand$y)
test.error.stand.test
```

```
## [1] 0.06975228
```

```
#On the logged testing data,fit the best linear model to make prediction
data.test.log <- data.frame(x=test_log,y=as.factor(test[,58]))
ypred.linear.log.test <- predict(bestmod.linear.log,data.test.log)
#Calculate test error, logged
table(predict = ypred.linear.log.test,truth = data.test.log$y)
```

```
##        truth
## predict   0   1
##       0 882  47
##       1  34 571
```

```
test.error.log.test <- mean(ypred.linear.log.test!=data.test.log$y)
test.error.log.test
```

```
## [1] 0.05280313
```

```
#On the discretized testing data,find the best linear model
data.test.dis <- data.frame(x=test_dis[,-c(55,56,57)],y=as.factor(test[,58]))
ypred.linear.dis.test <- predict(bestmod.linear.dis,data.test.dis)
#Calculate test error, discretized
table(predict = ypred.linear.dis.test,truth = data.test.dis$y)
```

```
##        truth
## predict   0   1
##       0 864  64
##       1  52 554
```

```
test.error.dis.test <- mean(ypred.linear.dis.test!=data.test.dis$y)
test.error.dis.test
```

```
## [1] 0.0756193
```

```
#On the standardized data,find the best gaussian model
svmfit.gaussian.stand <- svm(y~.,data=data.train.stand,kernel="radial",gamma=1,cost=10)
tune.gaussian.stand = tune(svm,y~.,data=data.train.stand,kernel="radial",            ranges=list(cost=
bestmod.gaussian.stand <- tune.gaussian.stand$best.model

#Calculate test error, standardized
ypred.gaussian.stand <- predict(bestmod.gaussian.stand,data.train.stand)
table(predict=ypred.gaussian.stand, truth=data.train.stand$y)
```

```
##        truth
## predict    0    1
##       0 1835   40
##       1   14 1178
```

```
test.error.stand.g <- mean(ypred.gaussian.stand!=data.train.stand$y)
test.error.stand.g
```

```
## [1] 0.01760678
```

```
#On the logged data,find the best gaussian model
svmfit.gaussian.log <- svm(y~.,data=data.train.log,kernel="radial",gamma=1,cost=10)
```

```
tune.gaussian.log = tune(svm,y~.,data=data.train.log,kernel="radial",            ranges=list(cost=c(8,
bestmod.gaussian.log <- tune.gaussian.log$best.model

#Calculate test error, logged
ypred.gaussian.log <- predict(bestmod.gaussian.log,data.train.log)
table(predict=ypred.gaussian.log, truth=data.train.log$y)
```

```
##        truth
## predict    0    1
##       0 1848   18
##       1    1 1200
```

```
test.error.log.g <- mean(ypred.gaussian.log!=data.train.log$y)
test.error.log.g
```

```
## [1] 0.006194979
```

```
#On the discretized data,find the best gaussian model
svmfit.gaussian.dis <- svm(y~.,data=data.train.dis,kernel="radial",gamma=1,cost=10)
tune.gaussian.dis = tune(svm,y~.,data=data.train.dis,kernel="radial",            ranges=list(cost=c(8,
bestmod.gaussian.dis <- tune.gaussian.dis$best.model

#Calculate test error, discretized
ypred.gaussian.dis <- predict(bestmod.gaussian.dis,data.train.dis)
table(predict=ypred.gaussian.dis, truth=data.train.dis$y)
```

```
##        truth
## predict    0    1
##       0 1828   56
##       1   21 1162
```

```
test.error.dis.g <- mean(ypred.gaussian.dis!=data.train.dis$y)
test.error.dis.g
```

```
## [1] 0.02510597
```

```
#On the testing standardized data,fit the best gaussian model to make prediction
ypred.gaussian.stand.test <- predict(bestmod.gaussian.stand,data.test.stand)
table(predict=ypred.gaussian.stand.test, truth=data.test.stand$y)
```

```
##        truth
## predict   0   1
##       0 897  54
##       1  19 564
```

```
test.error.stand.g.test <- mean(ypred.gaussian.stand.test!=data.test.stand$y)
test.error.stand.g.test
```

```
## [1] 0.04758801
```

```
#On the testing logged data,fit the best gaussian model to make prediction
ypred.gaussian.log.test <- predict(bestmod.gaussian.log,data.test.log)
table(predict=ypred.gaussian.log.test, truth=data.test.log$y)
```

```
##        truth
## predict   0   1
##       0 896  37
##       1  20 581
```

```r
test.error.log.g.test <- mean(ypred.gaussian.log.test!=data.test.log$y)
test.error.log.g.test
```

```
## [1] 0.03715776
```

```r
#On the testing discretized data,fit the best gaussian model to make prediction
ypred.gaussian.dis.test <- predict(bestmod.gaussian.dis,data.test.dis)
table(predict=ypred.gaussian.dis.test, truth=data.test.dis$y)
```

```
##        truth
## predict   0   1
##       0 886  46
##       1  30 572
```

```r
test.error.dis.g.test <- mean(ypred.gaussian.dis.test!=data.test.dis$y)
test.error.dis.g.test
```

```
## [1] 0.04954368
```

```r
report.training <-matrix(c('Standardized Train Data', 'SVM Linear',test.error.stand),ncol=3,byrow=TRUE
colnames(report.training)<-c("Data Version","Method","Test Error")
report.training <- rbind(report.training,c('Logged Train Data','SVM Linear', test.error.log),c('Discre

report.testing <-matrix(c('Standardized Test Data', 'SVM Linear',test.error.stand.test),ncol=3,byrow=T
colnames(report.testing)<-c("Data Version","Method","Test Error")
report.testing <- rbind(report.testing,c('Logged Test Data','SVM Linear', test.error.log.test),c('Disc

report_SVM <- rbind(report.training,report.testing)
```

```r
report <- rbind(report_Logistic,report_lda,report_SVM)
report
```

```
##       Data Version              Method                Test Error
##  [1,] "Standardized Train Data" "Logistic Regression" "0.071731333550701"
##  [2,] "Standardized Test Data"  "Logistic Regression" "0.0710560625814863"
##  [3,] "Logged Train Data"       "Logistic Regression" "0.0577111183567004"
##  [4,] "Logged Test Data"        "Logistic Regression" "0.0567144719687093"
##  [5,] "Discretized Train Data"  "Logistic Regression" "0.0570590153244213"
##  [6,] "Discretized Test Data"   "Logistic Regression" "0.0808344198174707"
##  [7,] "Standardized Train Data" "LDA"                 "0.10172807303554"
##  [8,] "Standardized Test Data"  "LDA"                 "0.102998696219035"
##  [9,] "Standardized Train Data" "QDA"                 "0.178676230844473"
## [10,] "Standardized Test Data"  "QDA"                 "0.17470664928292"
## [11,] "Logistic Train Data"     "LDA"                 "0.0603195304858168"
## [12,] "Logistic Test Data"      "LDA"                 "0.0651890482398957"
## [13,] "Logistic Train Data"     "QDA"                 "0.158787088359961"
## [14,] "Logistic Test Data"      "QDA"                 "0.157105606258149"
## [15,] "Standardized Train Data" "SVM Linear"          "0.0622758395826541"
## [16,] "Logged Train Data"       "SVM Linear"          "0.0547766547114444"
## [17,] "Discretized Train Data"  "SVM Linear"          "0.0580371698728399"
## [18,] "Standardized Train Data" "SVM Gaussian"        "0.0176067818715357"
## [19,] "Logged Train Data"       "SVM Gaussian"        "0.00619497880665145"
## [20,] "Discretized Train Data"  "SVM Gaussian"        "0.0251059667427454"
## [21,] "Standardized Test Data"  "SVM Linear"          "0.0697522816166884"
## [22,] "Logged Test Data"        "SVM Linear"          "0.0528031290743155"
## [23,] "Discretized Test Data"   "SVM Linear"          "0.075619295958279"
```

```
## [24,] "Standardized Test Data"  "SVM Gaussian"        "0.0475880052151239"
## [25,] "Logged Test Data"        "SVM Gaussian"        "0.0371577574967405"
## [26,] "Discretized Test Data"   "SVM Gaussian"        "0.0495436766623207"
```

```
report[which.min(report[,3]),]
```

```
##          Data Version                Method            Test Error
##   "Logged Train Data"        "SVM Gaussian" "0.00619497880665145"
```

Logged train data using SVM Gaussian Model gives us the smallest error rate which is 0.006847. Among the test set, the logged test data using SVM Gaussian Model gives the smallest error rate which is 0.03716. This makes sense because from the data we visualized, we think that the log transformed data is the most effective in viewing. The LDA and QDA methods give the worst performance. The test error rate for test set exceeds 10% for three versions of the data. It seems that LDA na QDA do not work well for our data set. Logistic Regression ranked at the second. The average test error rate for test set is around 6% which is much lower than LDA/QDA. In conclusion, the SVM Gaussian model performs best. Among all three methods, it seems that the log transformed data perfromed best among three data versions.

```
significant_var <- which(coef(summary(glm2))[-1,4]<0.05/57)
significant <- c(5,7,16,17,21,23,25,27,45,46,52,53,57)
train.final <- as.matrix(train_log[,significant])
train.final <- data.frame(x=train.final,y=as.factor(train[,58]))
svmfit <- svm(train.final$y~.,data = train.final, kernel="radial",gamma=1,cost=10)
tune <- tune(svm,y~.,data=train.final,kernel="radial", ranges=list(cost=c(8,10),gamma=c(0.03,0.05)))
bestmod <- tune$best.model
tune$best.parameters
```

```
##   cost gamma
## 4   10  0.05
```

```
ypred.final.train <- predict(bestmod,data.train.log)
table(predict=ypred.final.train, truth=data.train.log$y)
```

```
##        truth
## predict    0    1
##       0 1803  100
##       1   46 1118
```

```
error_train <- mean(ypred.final.train!=data.train.log$y)
error_train
```

```
## [1] 0.04760352
```

```
ypred.final.test <- predict(bestmod,data.test.log)
table(predict=ypred.final.test, truth=data.test.log$y)
```

```
##        truth
## predict   0   1
##       0 890  51
##       1  26 567
```

```
error_test <- mean(ypred.final.test !=data.test.log$y)
error_test
```

```
## [1] 0.05019557
```

We selected 13 variables with index as 5,7,16,17,21,23,25,27,45,46,52,53,57 from the logistic regression in part (b) after Bonferroni Correction. We then run SVM Gaussian Model on log transformed variables with above index. However, we obtianed the test error rate of 0.05 as compared with the full data SVM Gaussian test

error rate which is 0.03716. The new method we designed actually generates a higher test error rate. We tried to figure this out and one possible reason could be that we deleted too many variables and among the variables we deleted, there are variables that are actually significant. After all, we still concluded that the SVM Gaussian model on full log transformed data set gives best result.