Math189 Final Project

Contribution: Qing Wen A15956558 (Method 1), Jiying Wang A15827560 (Method 2), Yihan Liao A16130859 (Method 3)

CREAT DATASET

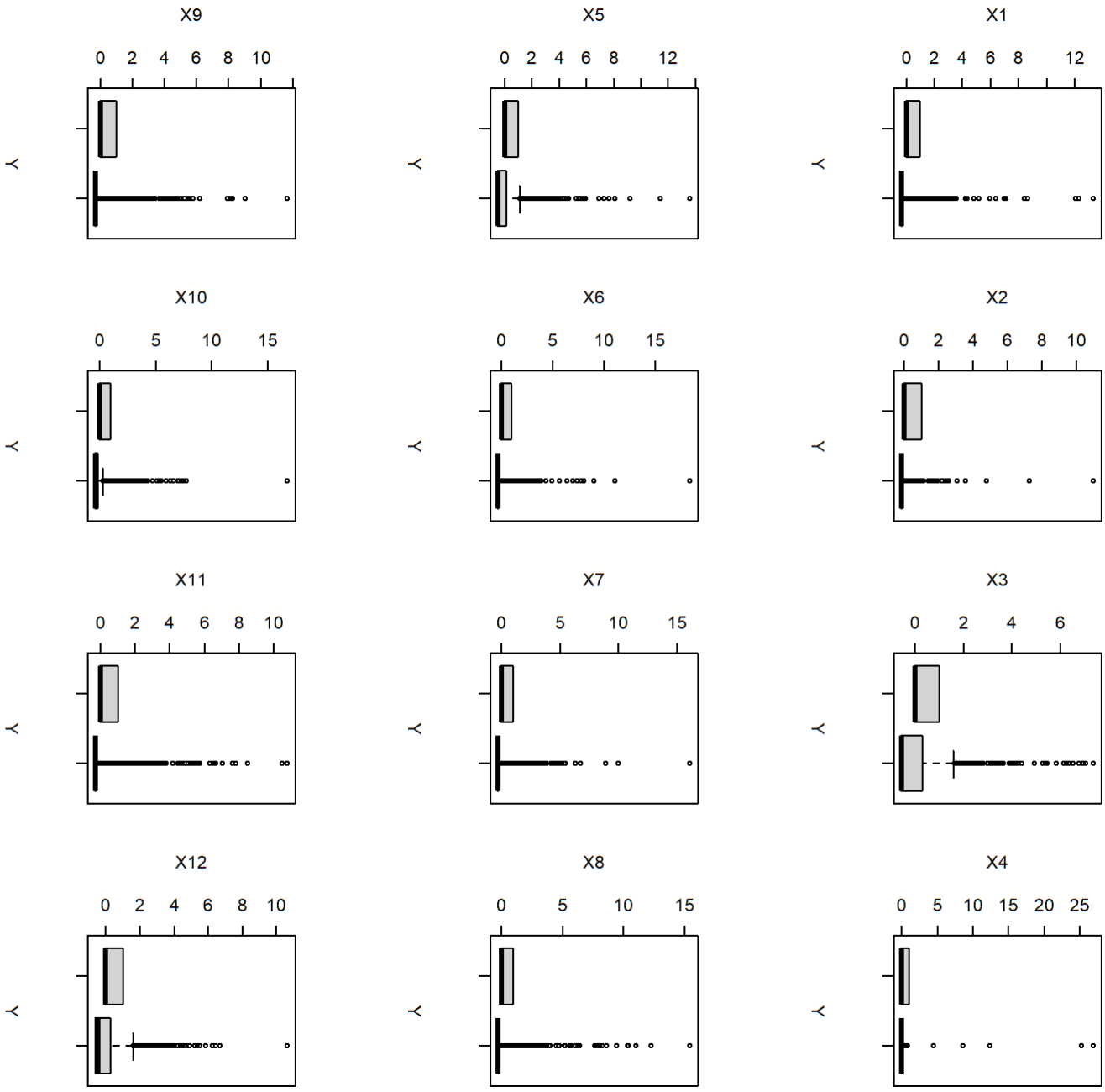
```
library(MASS)
library(e1071)

spam_train <- read.delim("spam-train.txt", sep = ",", header = F)
spam_test <- read.delim("spam-test.txt",sep = ",", header = F)

for (i in 1:57) {
  names(spam_train)[i] = paste0("X",i)
  names(spam_test)[i] = paste0("X",i)
}
names(spam_train)[58] = "Y"
names(spam_test)[58] = "Y"

#head(spam_train)
#head(spam_test)
```

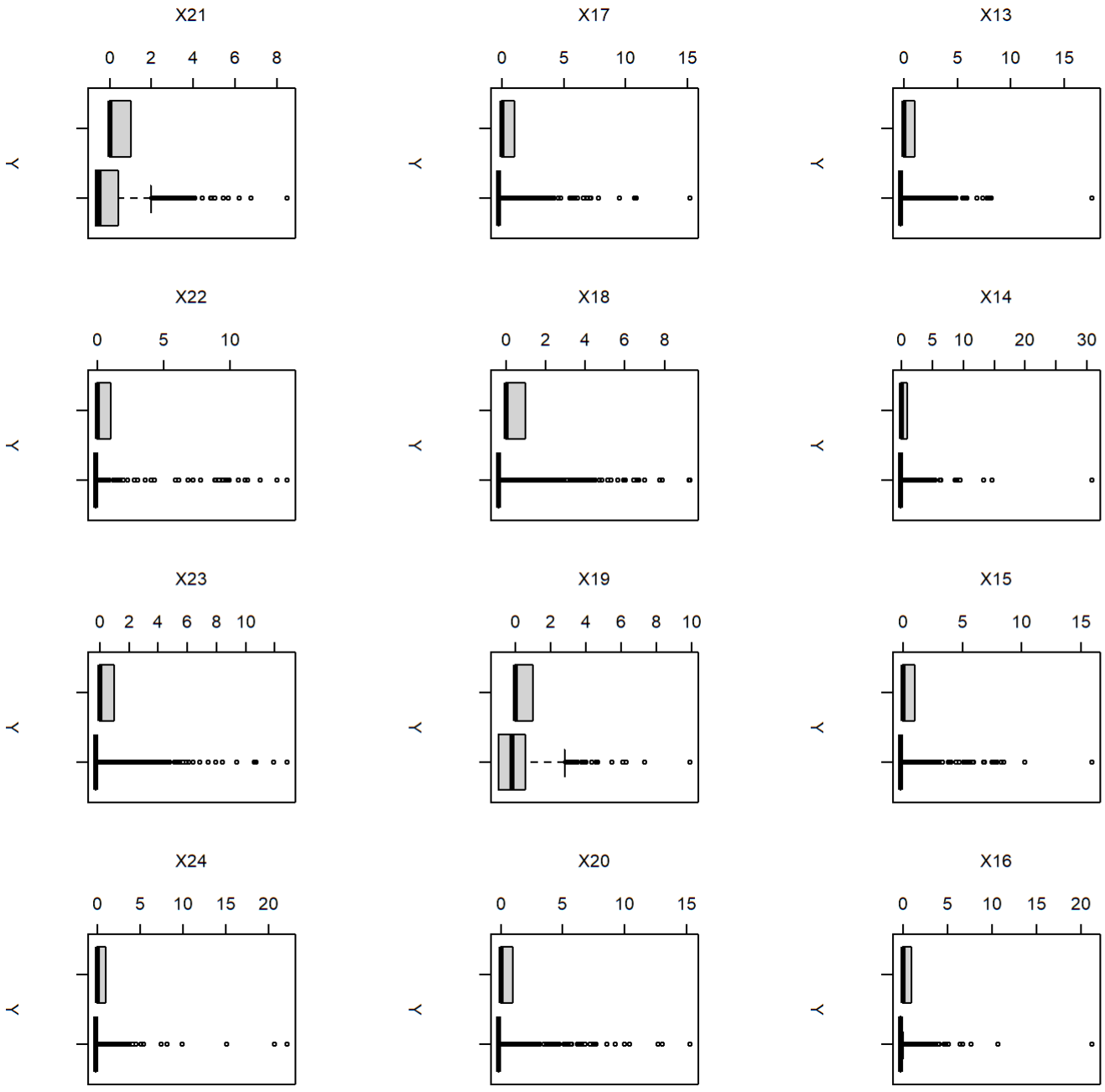
a. For each version of the data, visualize it using the tools introduced in the class.

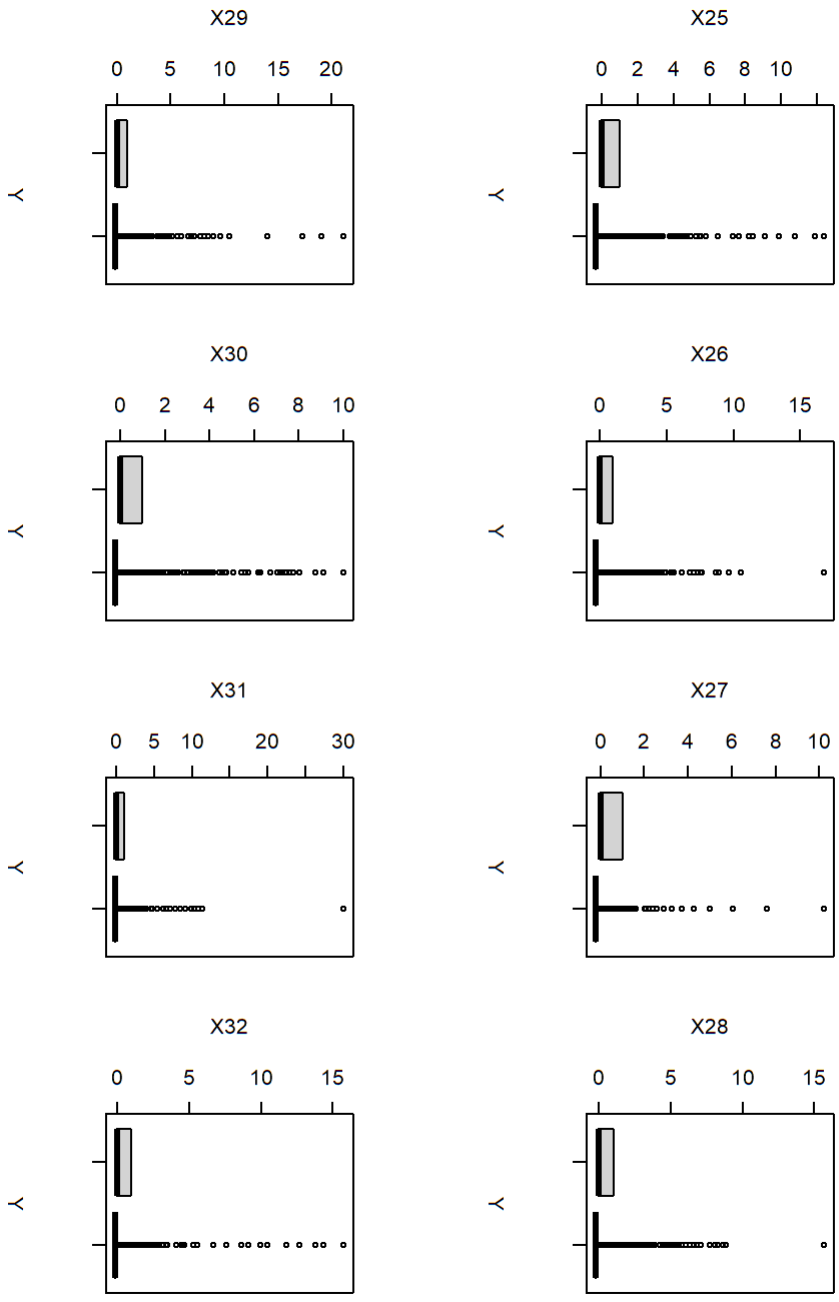
Method 1: standardize Dataset

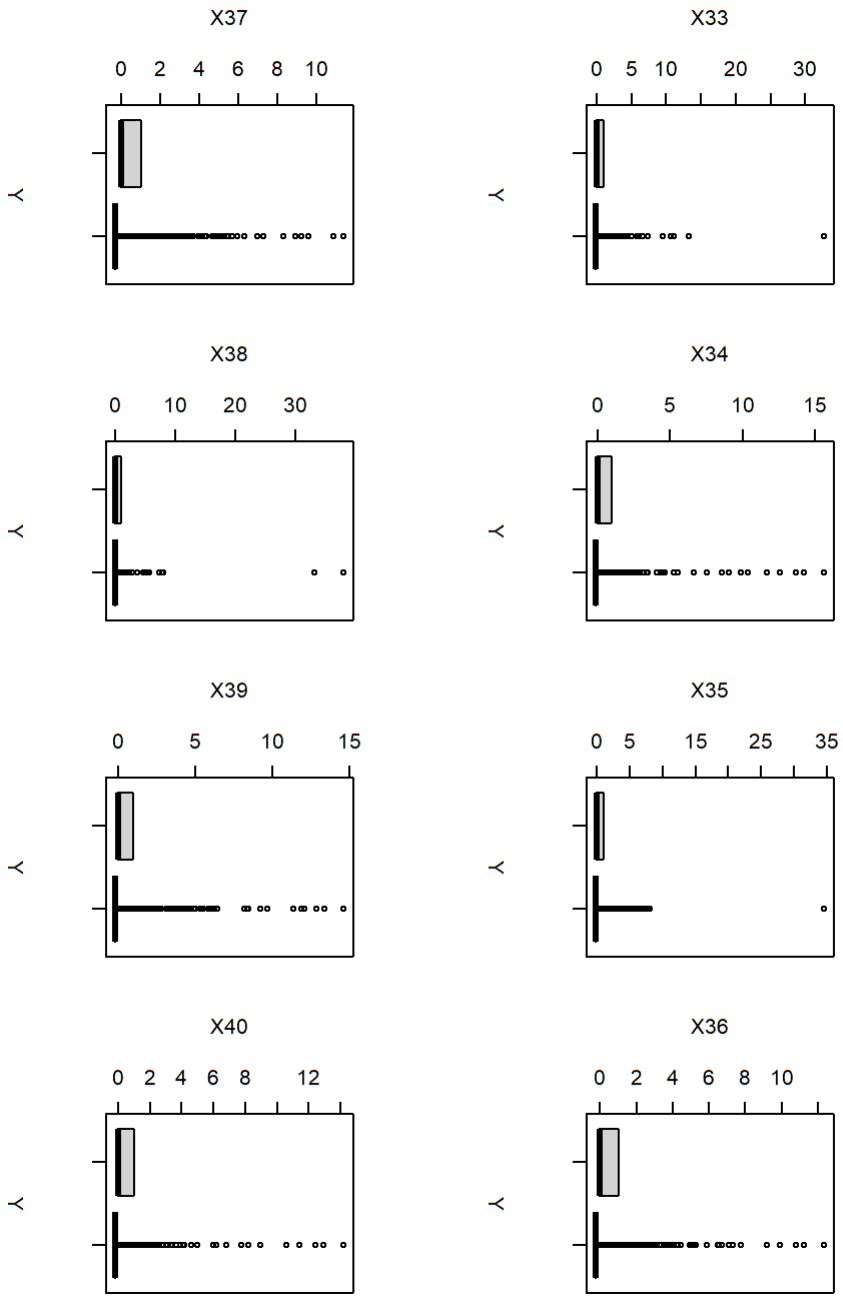
```
train.stand <- spam_train
train.stand[, -58] <- as.data.frame(scale(spam_train[, -58]))
test.stand <- spam_test
test.stand[, -58] <- as.data.frame(scale(spam_test[, -58]))
```

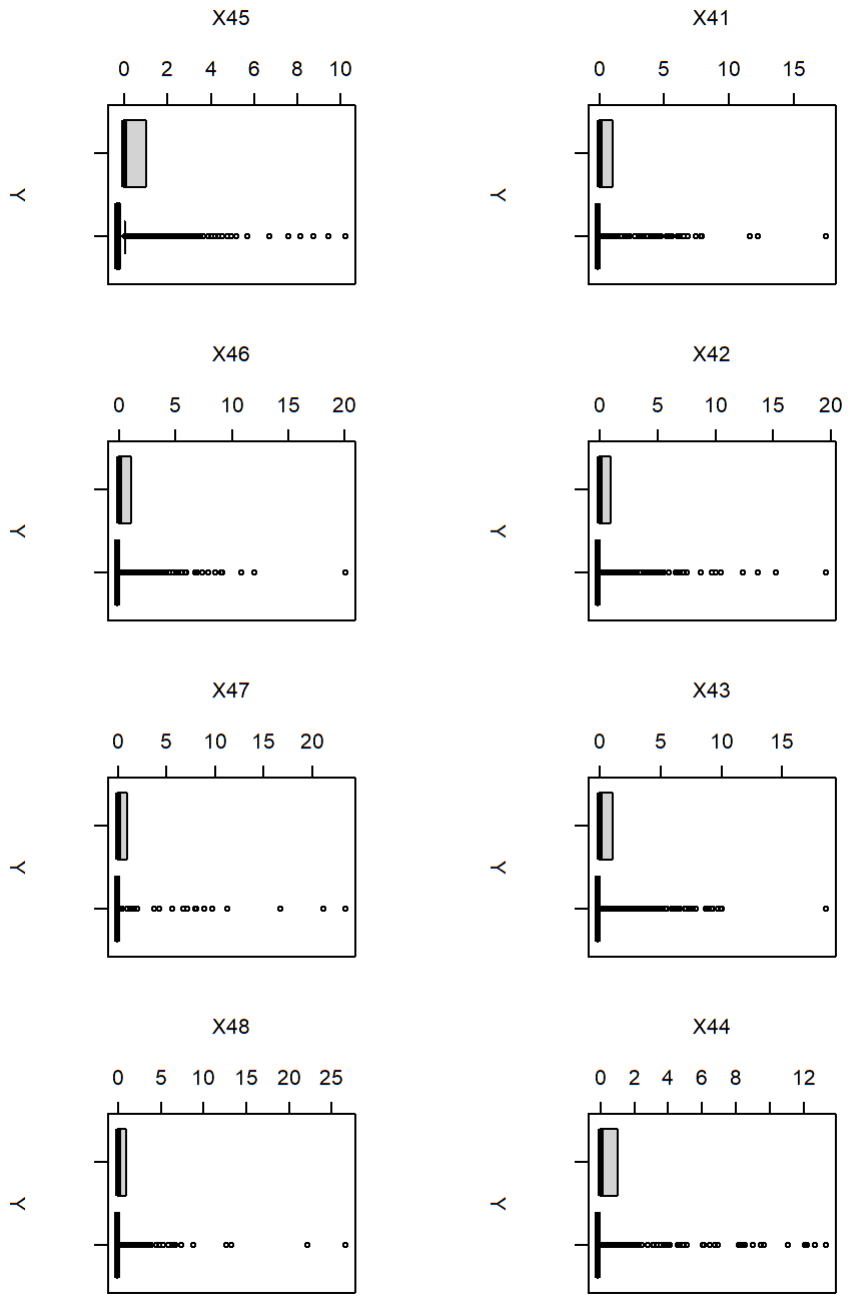
```
#(1)boxplot of Y ~ X_i

par(mfrow = c(2,4))
for (i in 1:57) {
  boxplot(train.stand$Y, train.stand[, i], xlab = "Y", ylab = paste0("X",i))
}
```

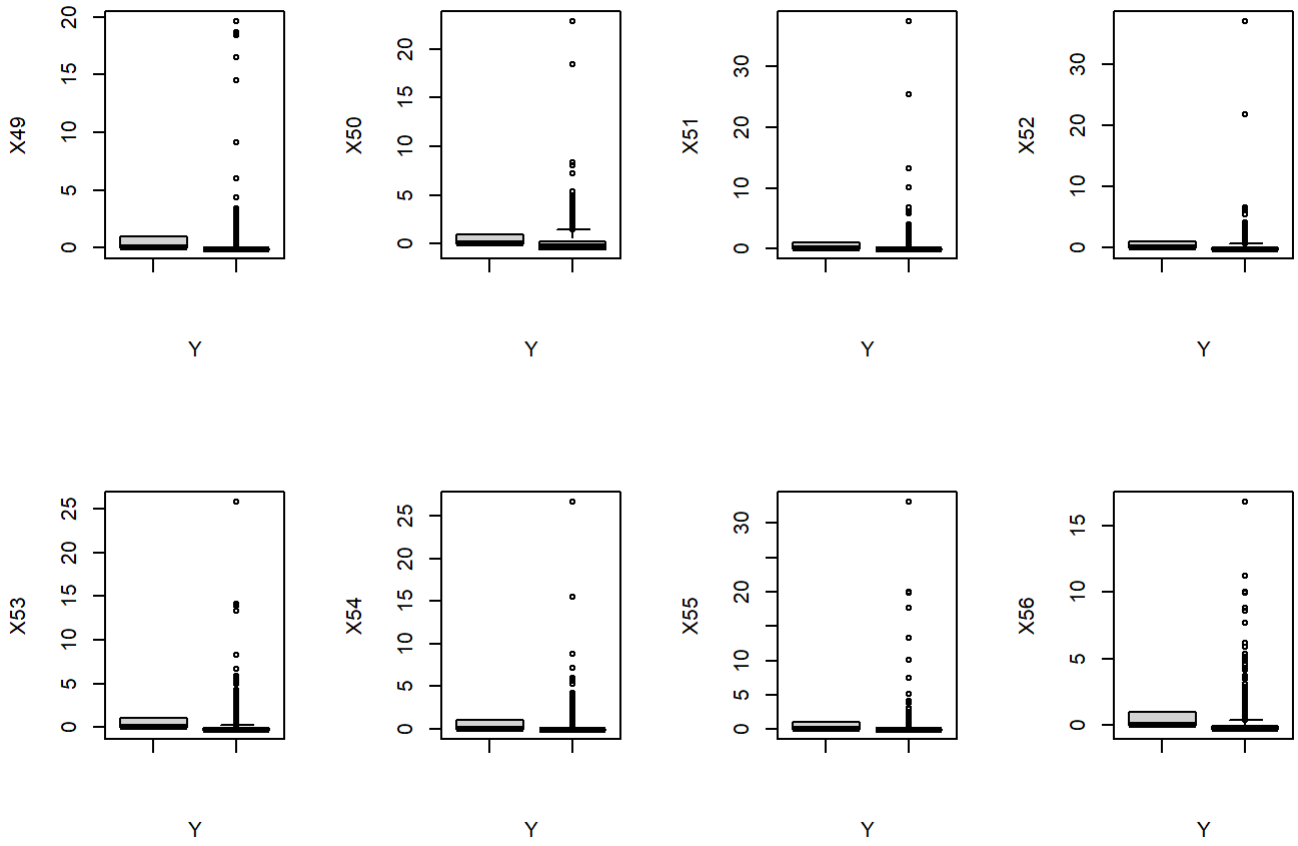

X9
0 2 4 6 8 10
Y
X5
0 2 4 6 8 12
Y
X1
0 2 4 6 8 12
Y
X10
0 5 10 15
Y
X6
0 5 10 15
Y
X2
0 2 4 6 8 10
Y
X11
0 2 4 6 8 10
Y
X7
0 5 10 15
Y
X3
0 2 4 6
Y
X12
0 2 4 6 8 10
Y
X8
0 5 10 15
Y
X4
0 5 10 15 20 25
Y


X21
0 2 4 6 8
Y
X17
0 5 10 15
Y
X13
0 5 10 15
Y
X22
0 5 10
Y
X18
0 2 4 6 8
Y
X14
0 5 10 20 30
Y
X23
0 2 4 6 8 10
Y
X19
0 2 4 6 8 10
Y
X15
0 5 10 15
Y
X24
0 5 10 15 20
Y
X20
0 5 10 15
Y
X16
0 5 10 15 20
Y

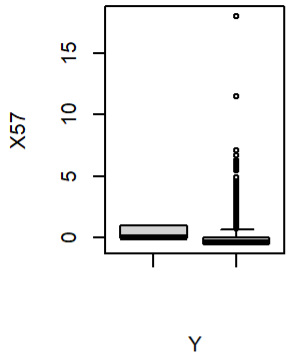

X29
0 5 10 15 20
Y
X25
0 2 4 6 8 10
Y
X30
0 2 4 6 8 10
Y
X26
0 5 10 15
Y
X31
0 5 10 20 30
Y
X27
0 2 4 6 8 10
Y
X32
0 5 10 15
Y
X28
0 5 10 15
Y


X37
0 2 4 6 8 10
Y
X33
0 5 10 20 30
Y
X38
0 10 20 30
Y
X34
0 5 10 15
Y
X39
0 5 10 15
Y
X35
0 5 15 25 35
Y
X40
0 2 4 6 8 12
Y
X36
0 2 4 6 8 10
Y

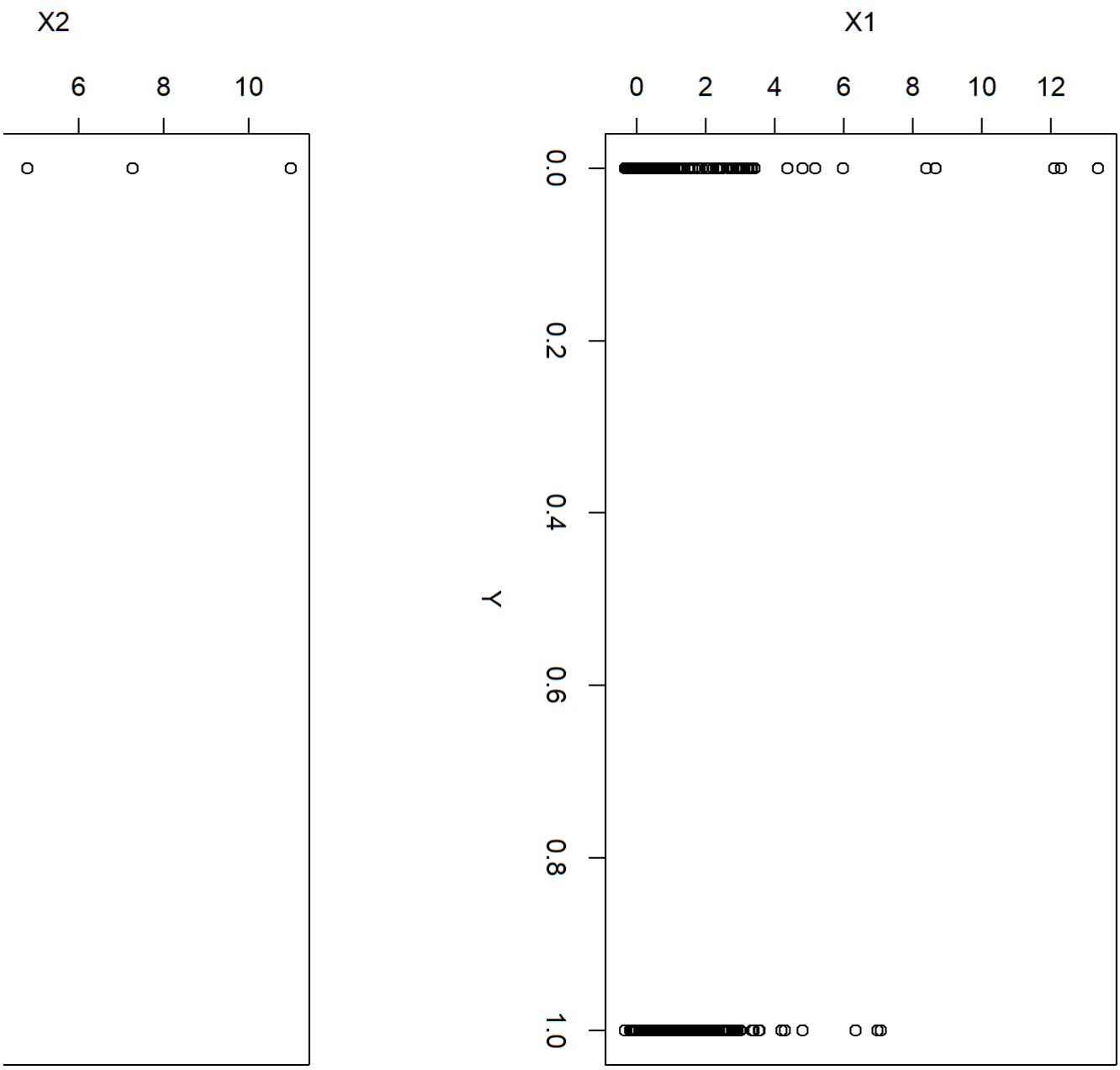

X45
0 2 4 6 8 10
Y
X41
0 5 10 15
Y
X46
0 5 10 15 20
Y
X42
0 5 10 15 20
Y
X47
0 5 10 15 20
Y
X43
0 5 10 15
Y
X48
0 5 10 15 20 25
Y
X44
0 2 4 6 8 12
Y

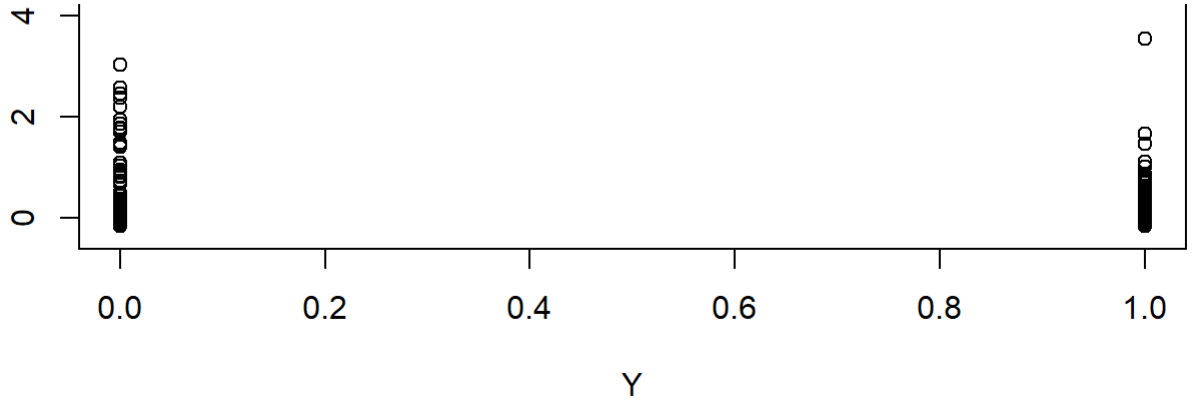


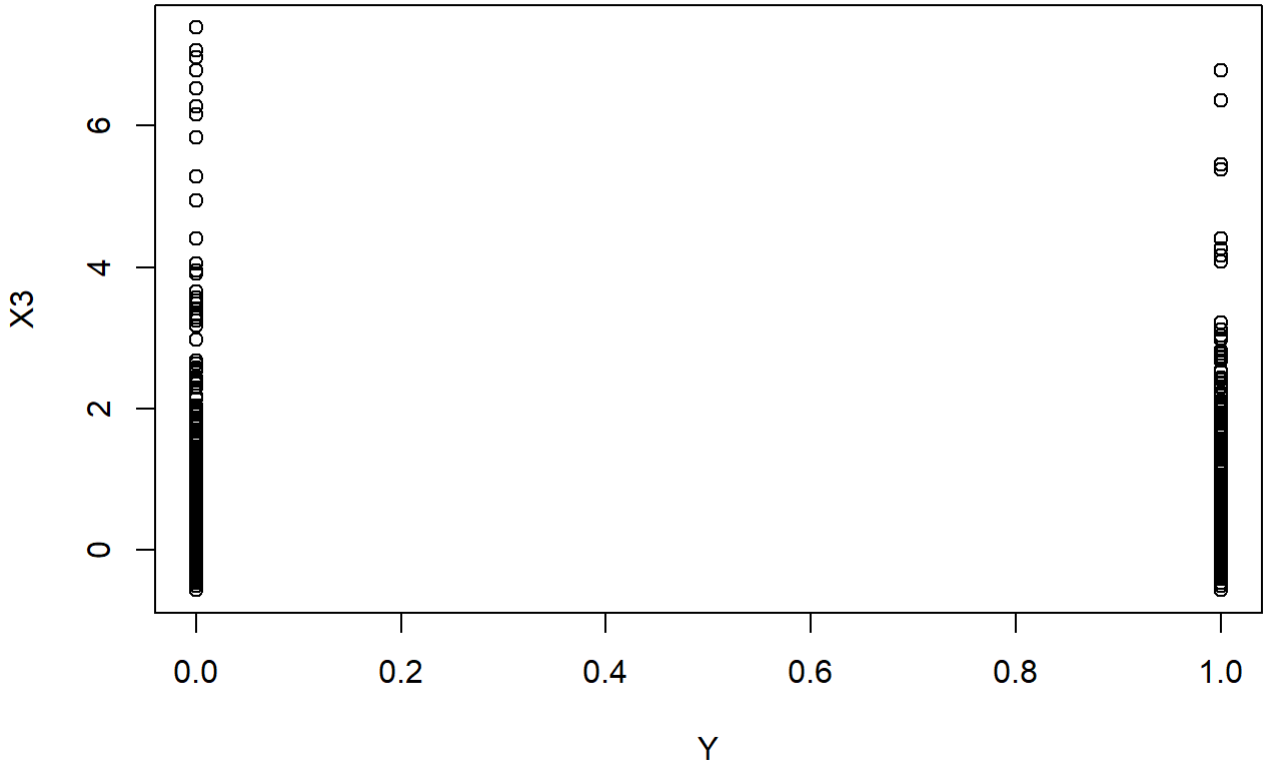

```
par(mfrow = c(1,1))
```

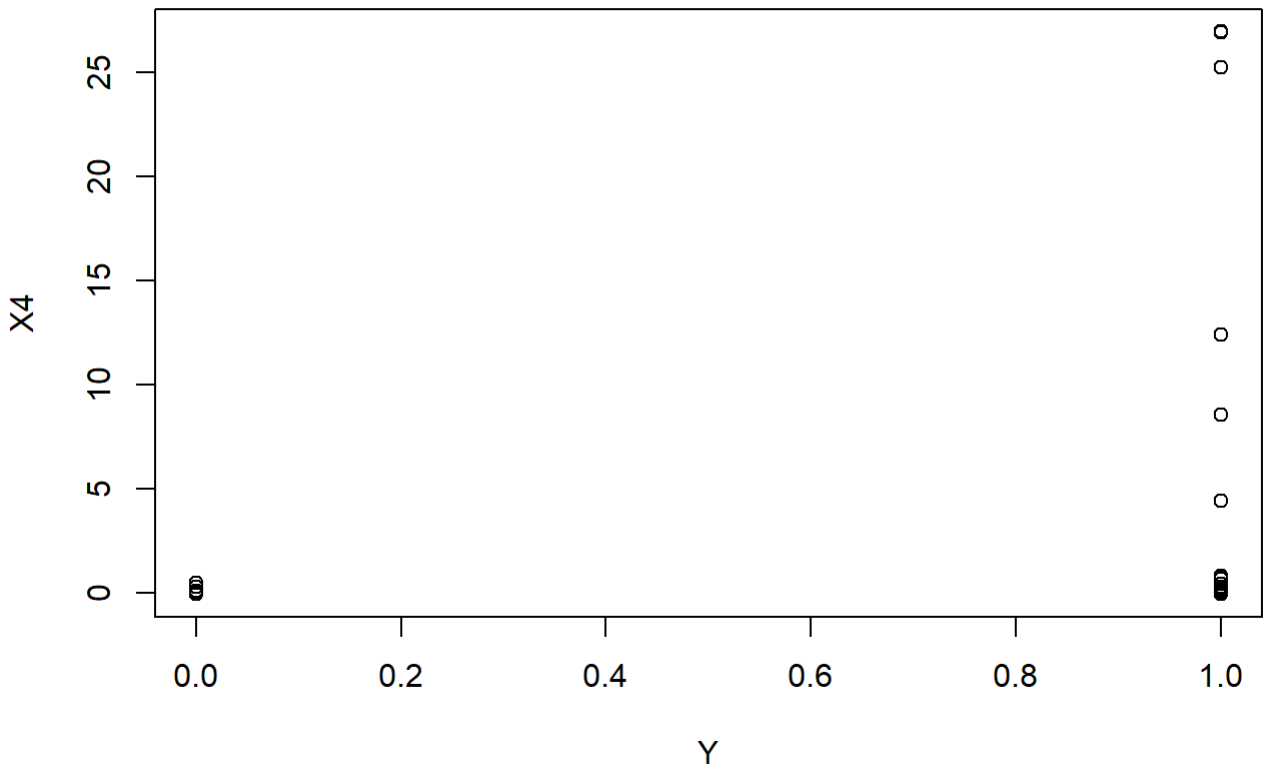


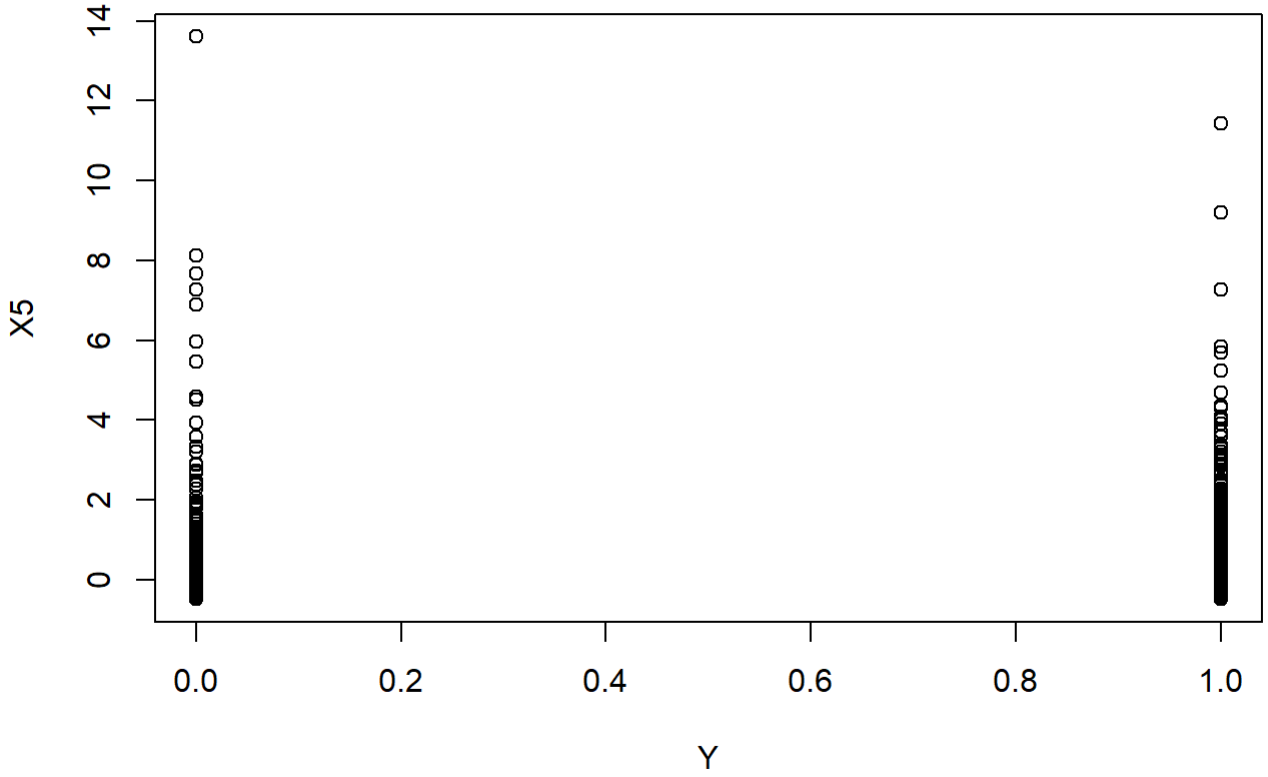

```
#(2)scatterplot of Y ~ X_i

for (i in 1:57){
  plot(train.stand$Y, train.stand[, i], xlab = "Y", ylab = paste0("X",i))
}
```

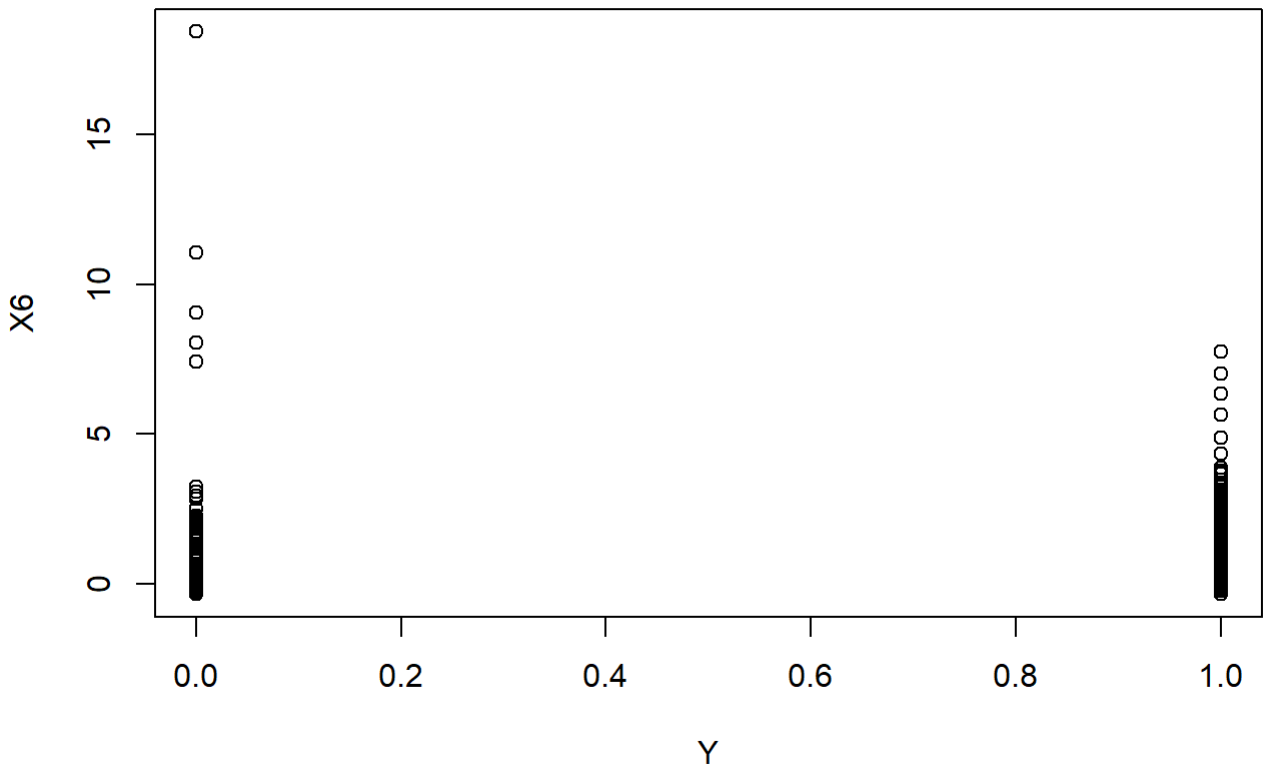

X2
6
8
10
X1
0
2
4
6
8
10
12
Y
0.0
0.2
0.4
0.6
0.8
1.0

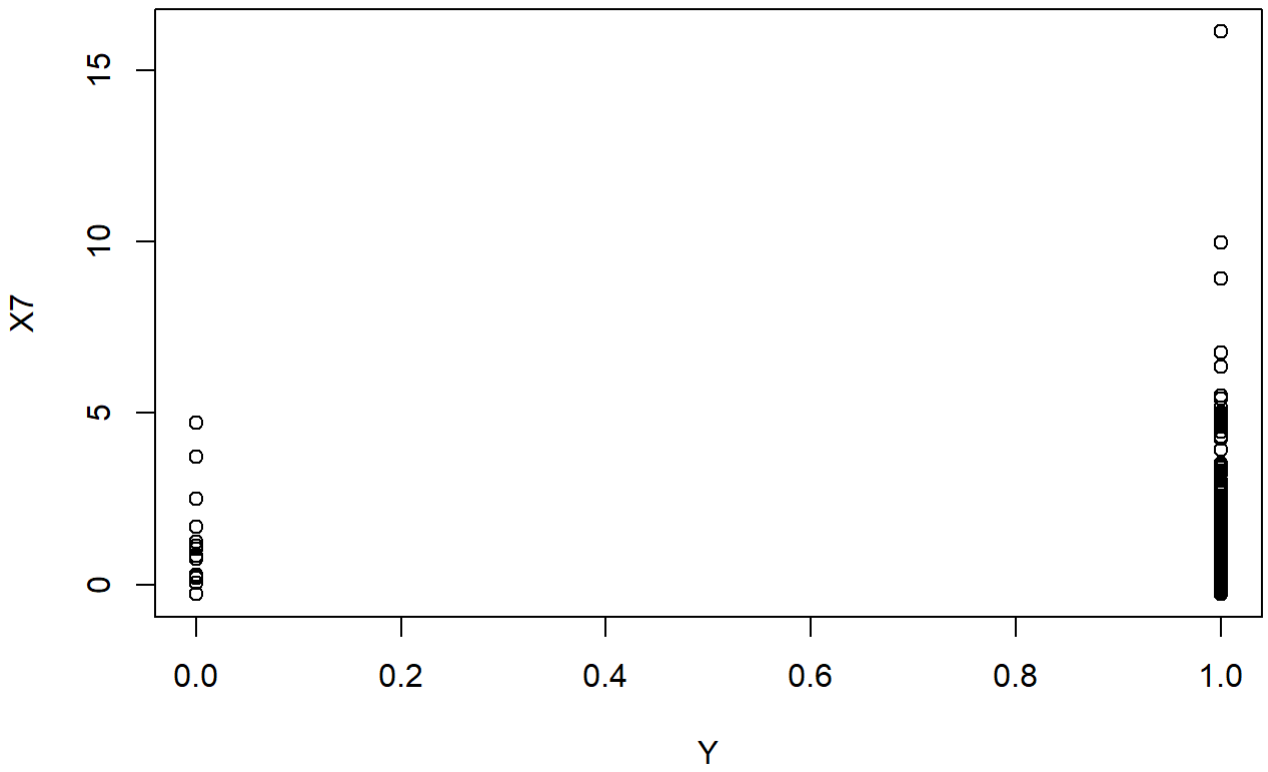

0
2
4
0.0
0.2
0.4
0.6
0.8
1.0
Y

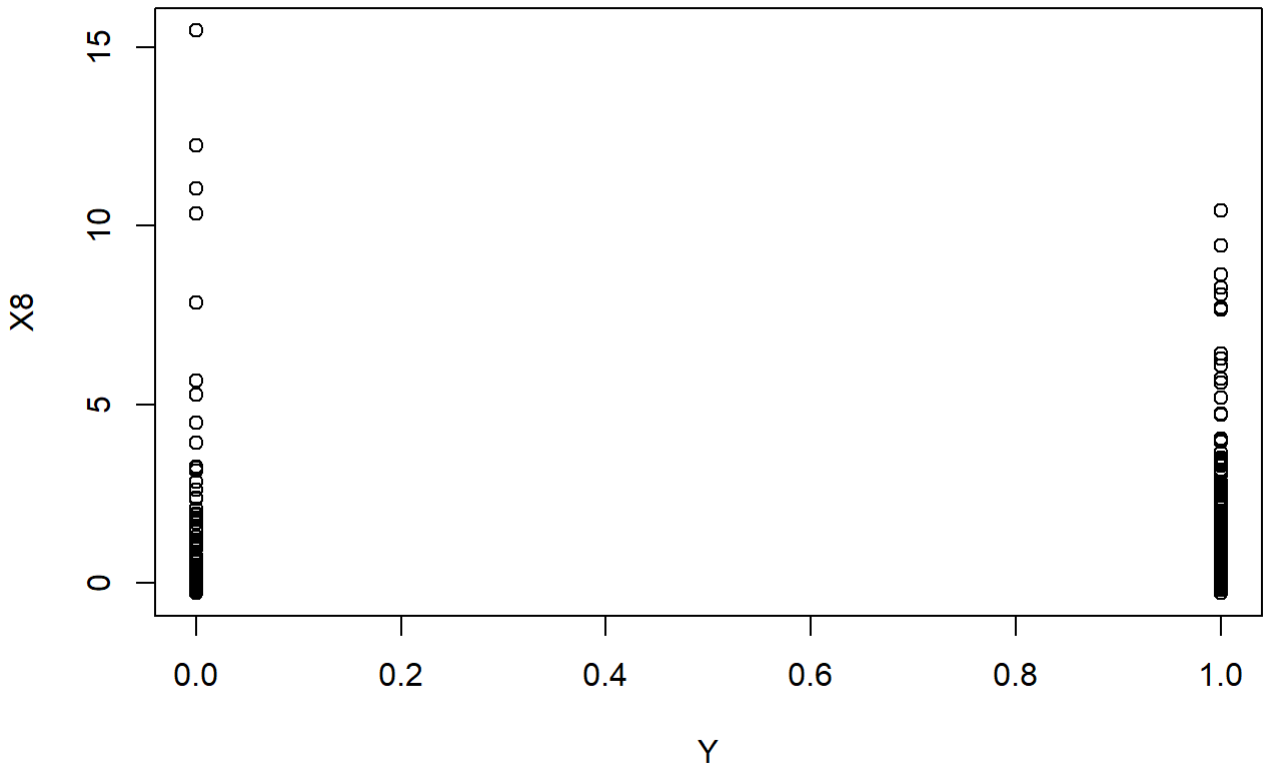

X3
0
2
4
6
0.0
0.2
0.4
0.6
0.8
1.0
Y

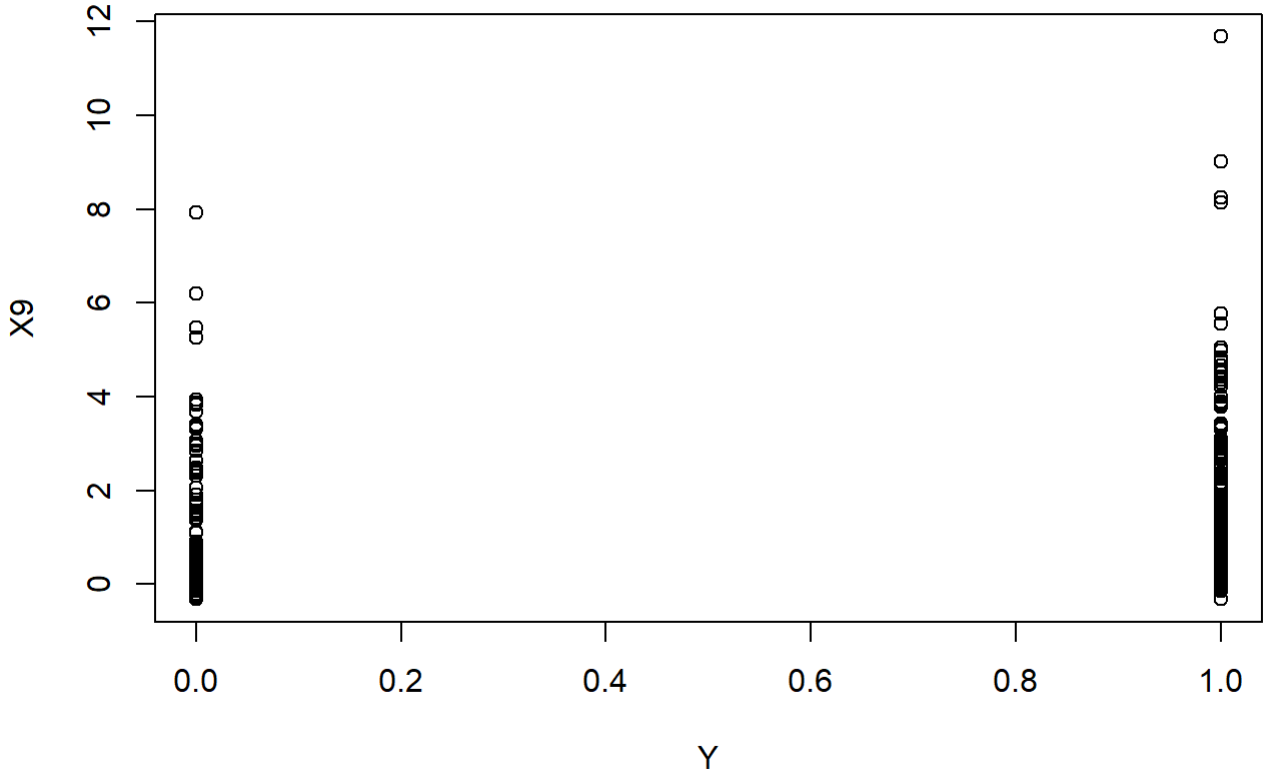

X4
0
5
10
15
20
25
0.0
0.2
0.4
0.6
0.8
1.0
Y

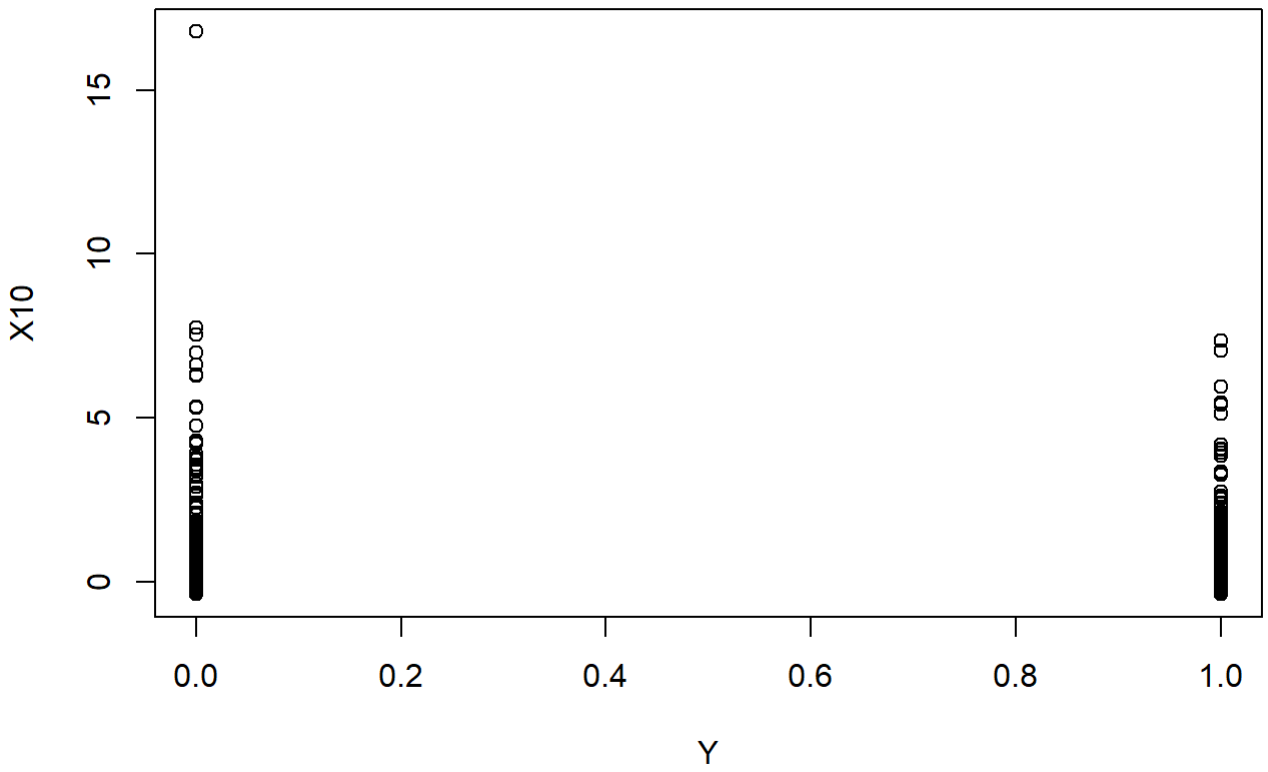

X5
0
2
4
6
8
10
12
14
0.0
0.2
0.4
0.6
0.8
1.0
Y

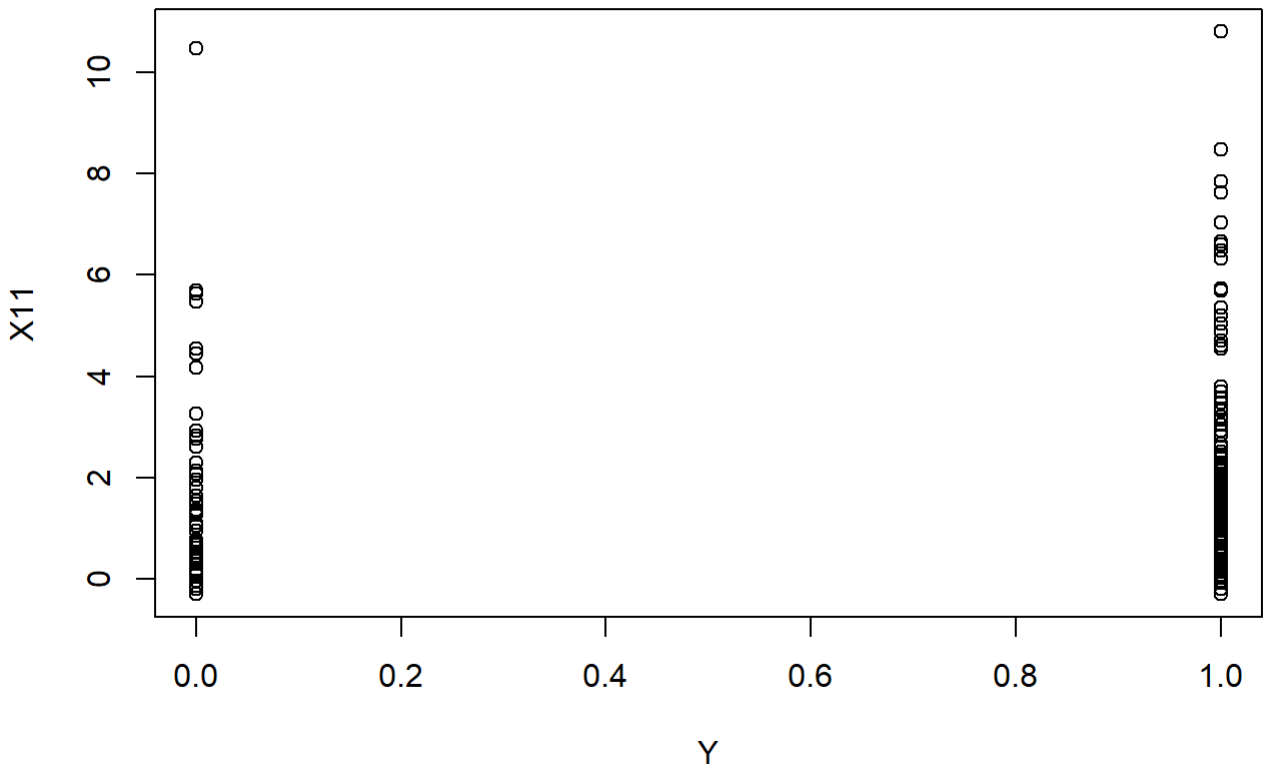

X6
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

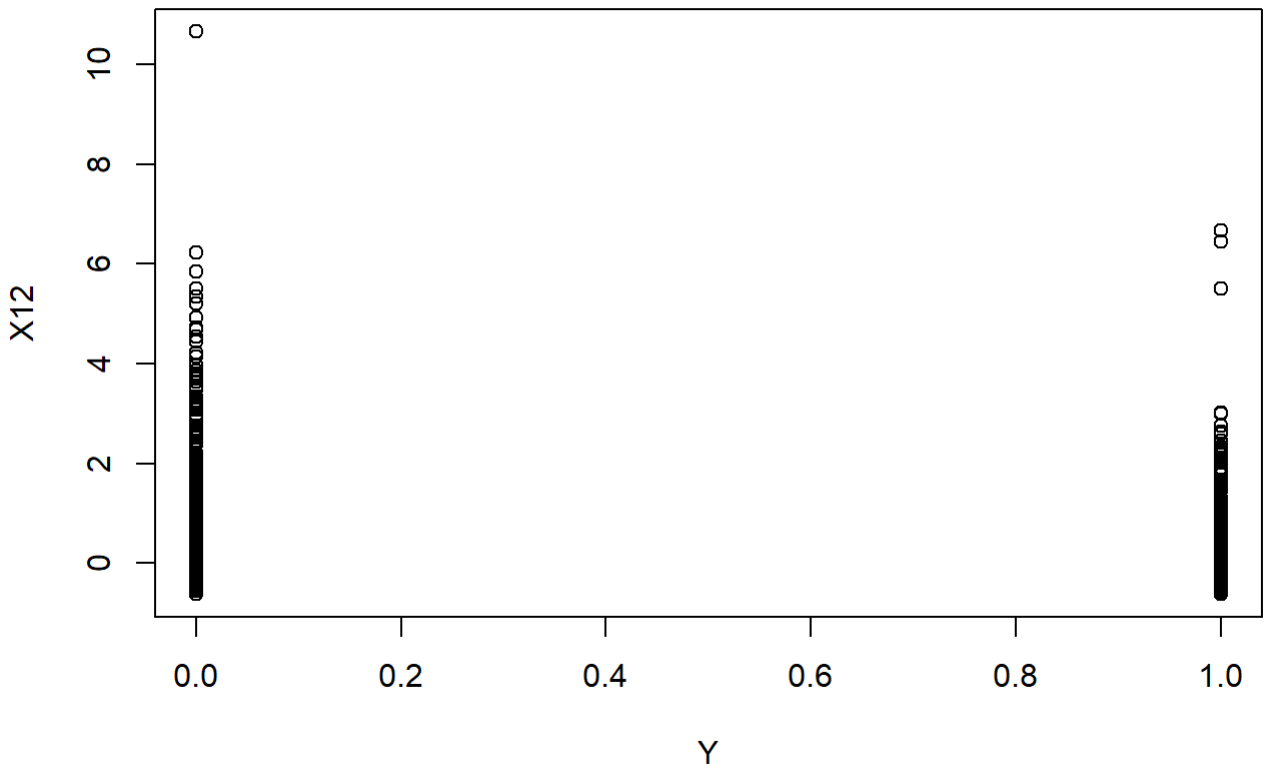

X7
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

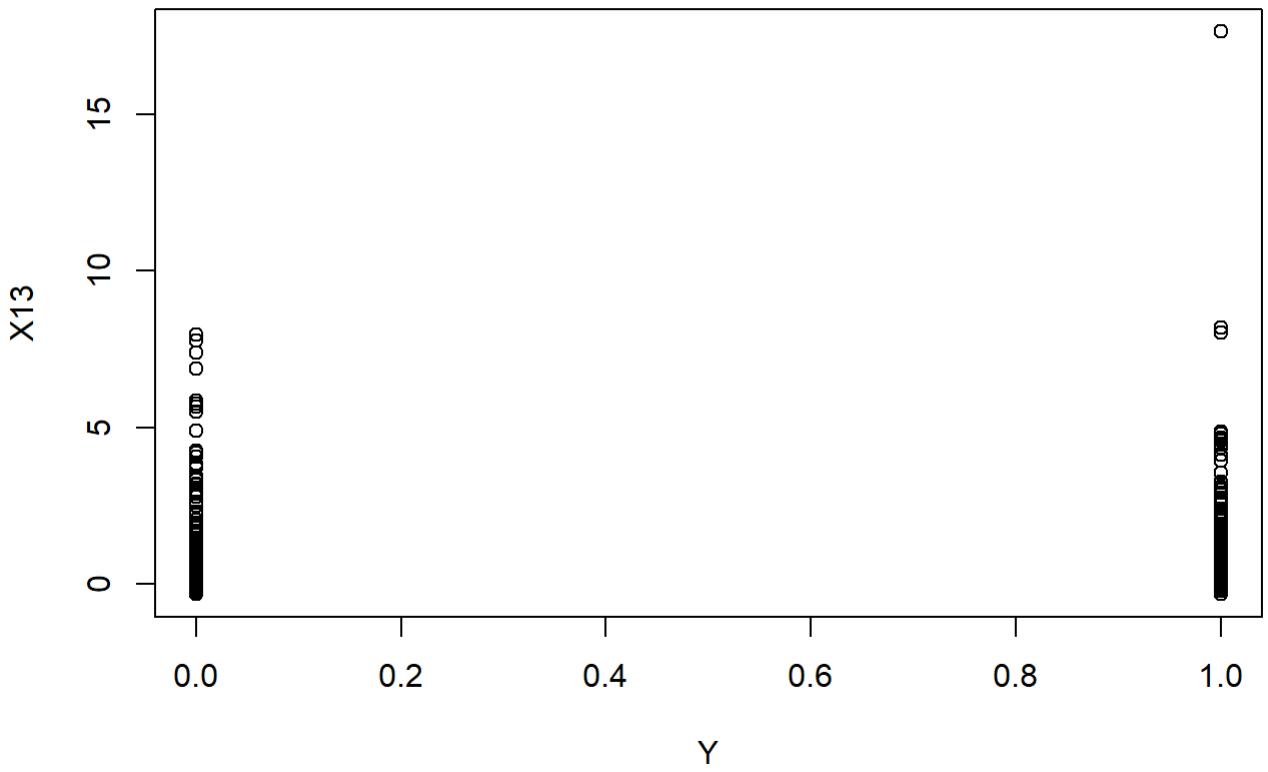

X8
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

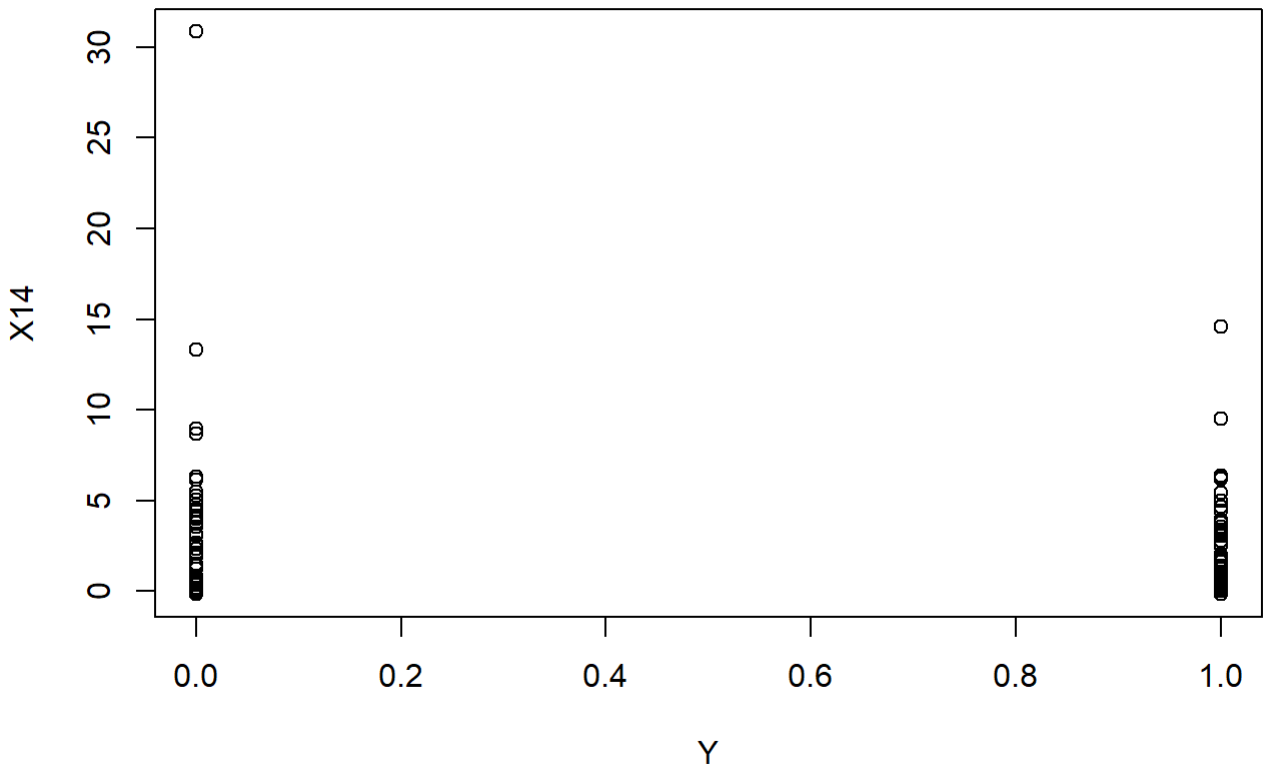

X9
0
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8
1.0
Y

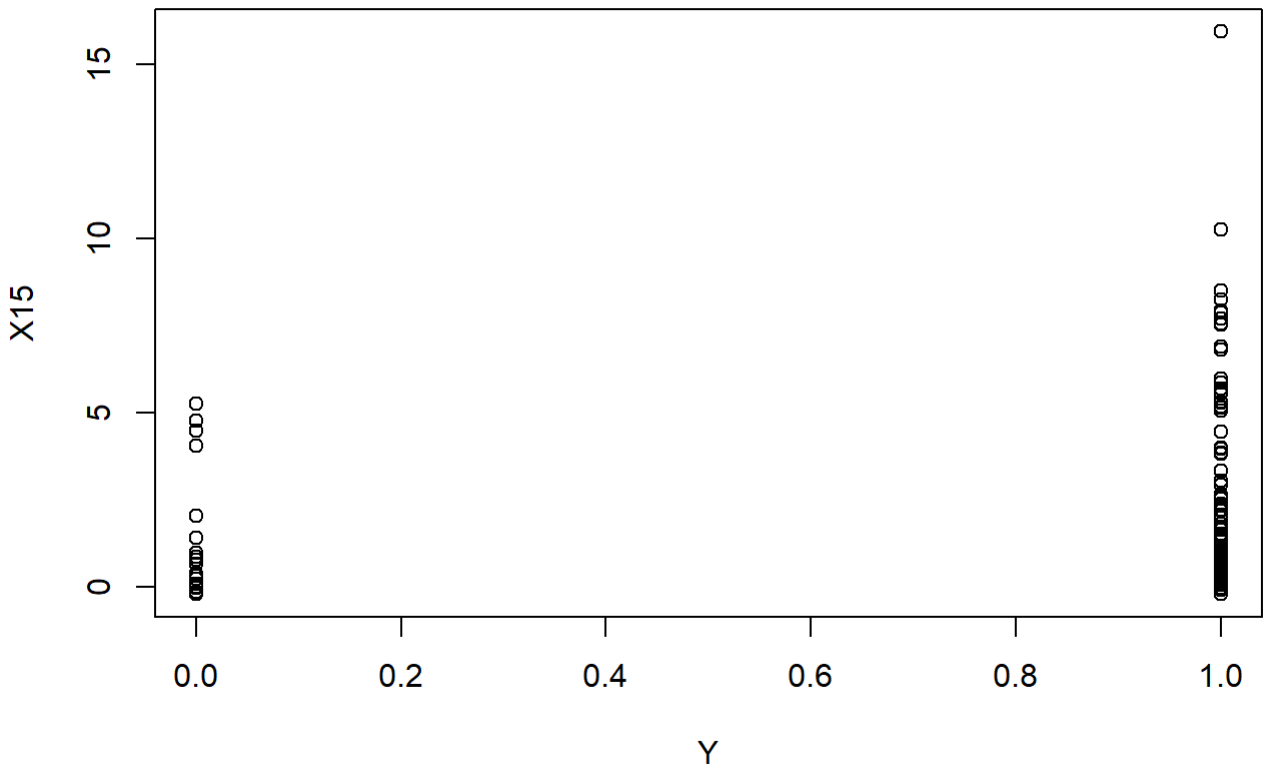

X10
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

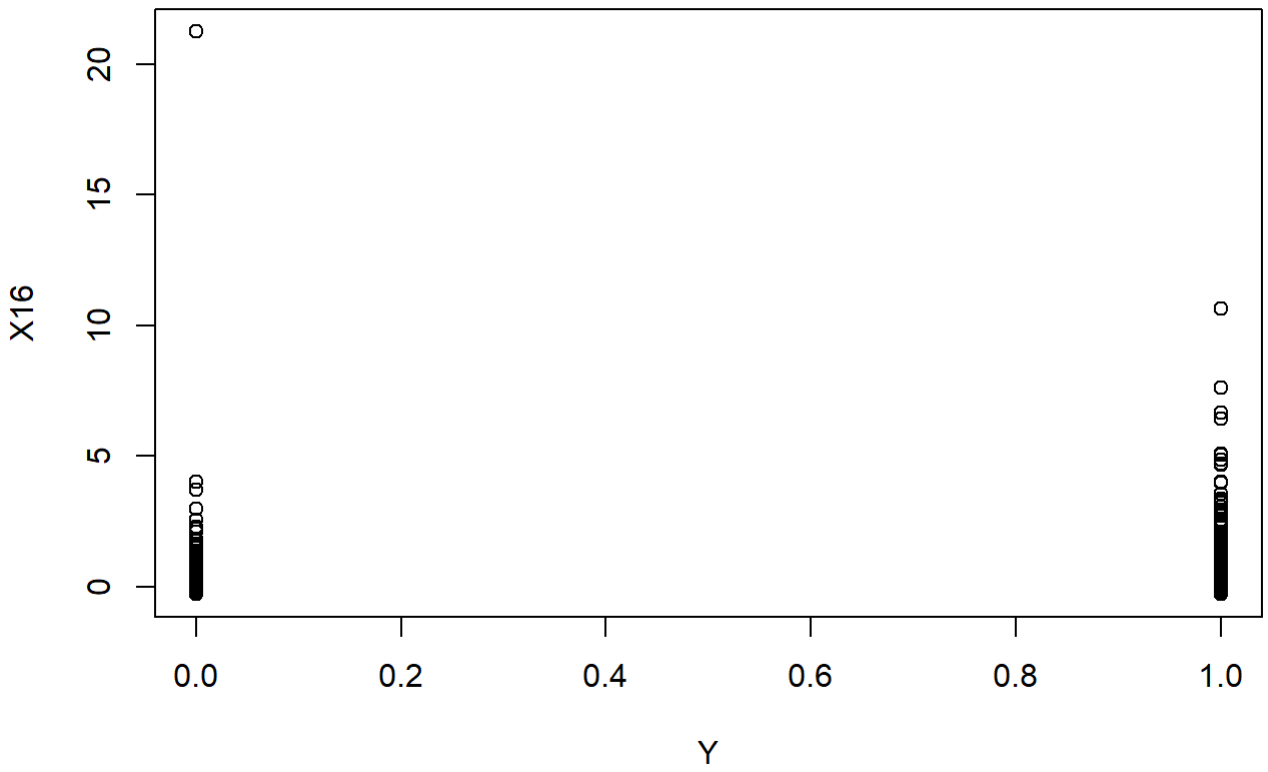

X11
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0
Y

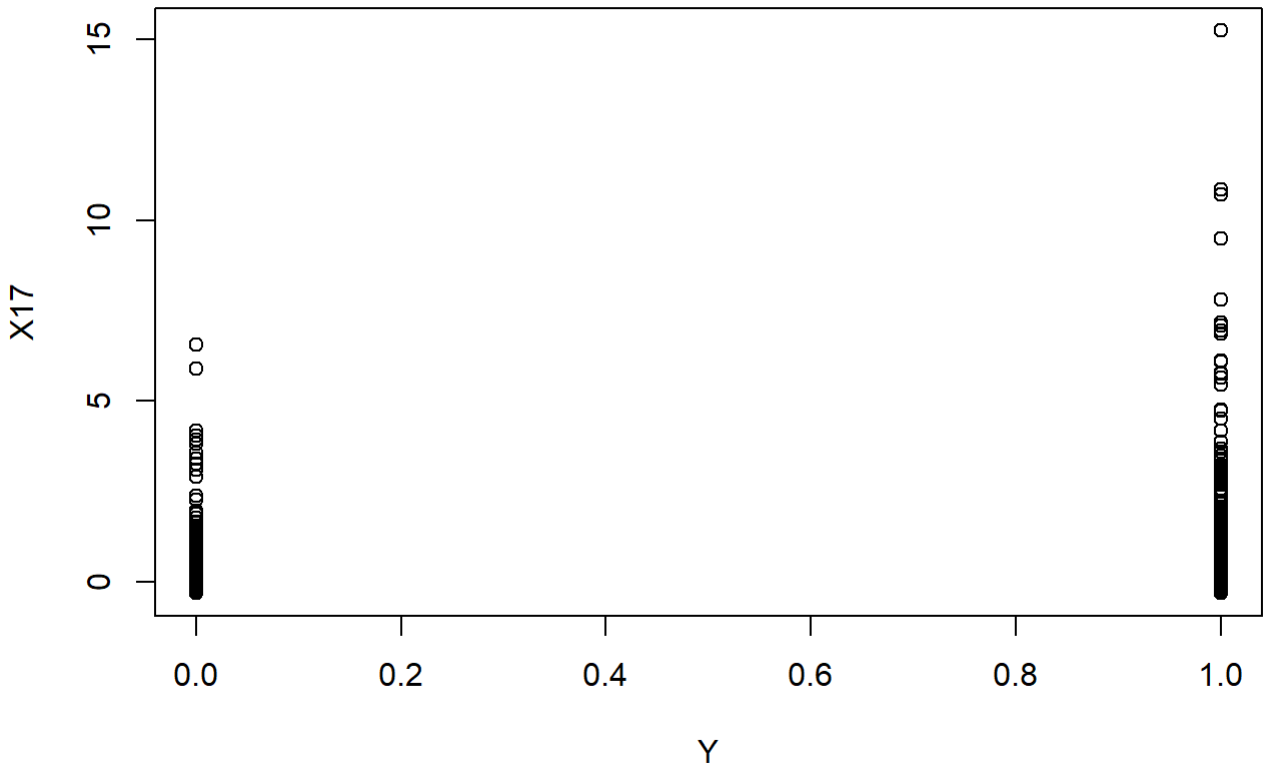

X12
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0
Y

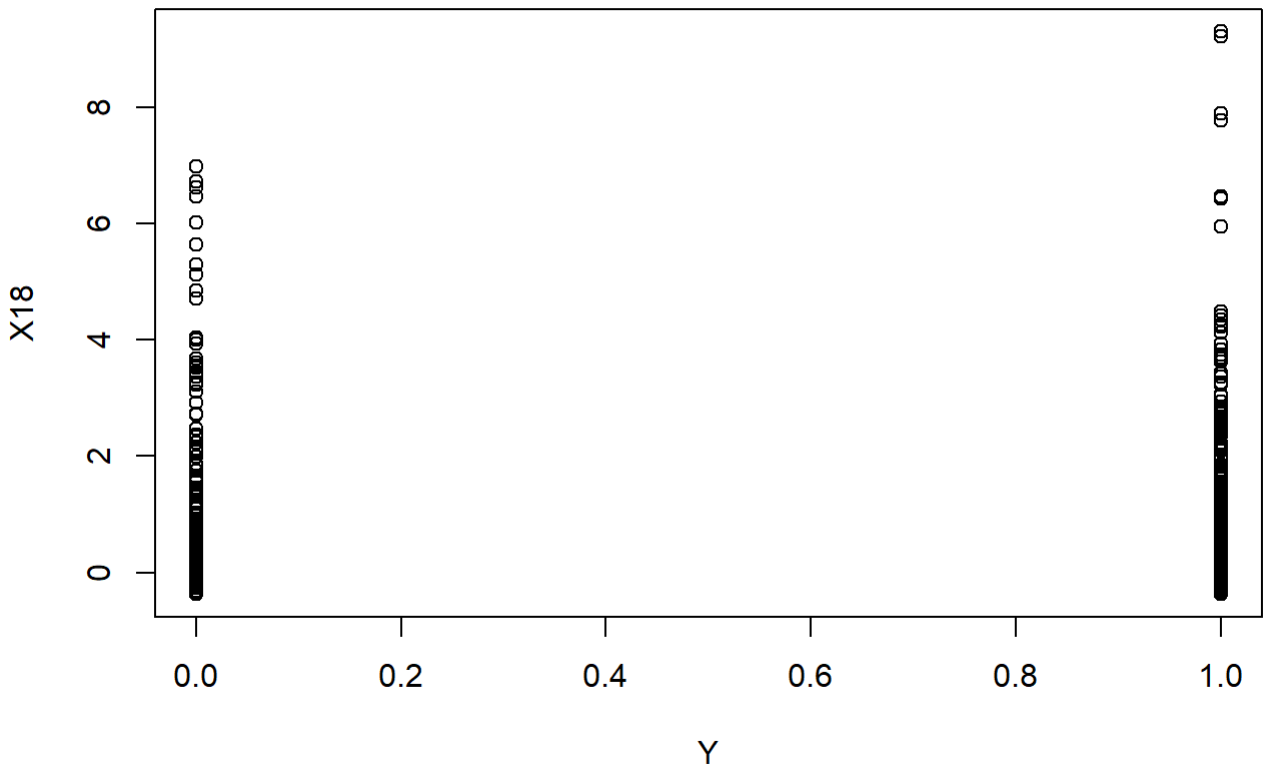

X13
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

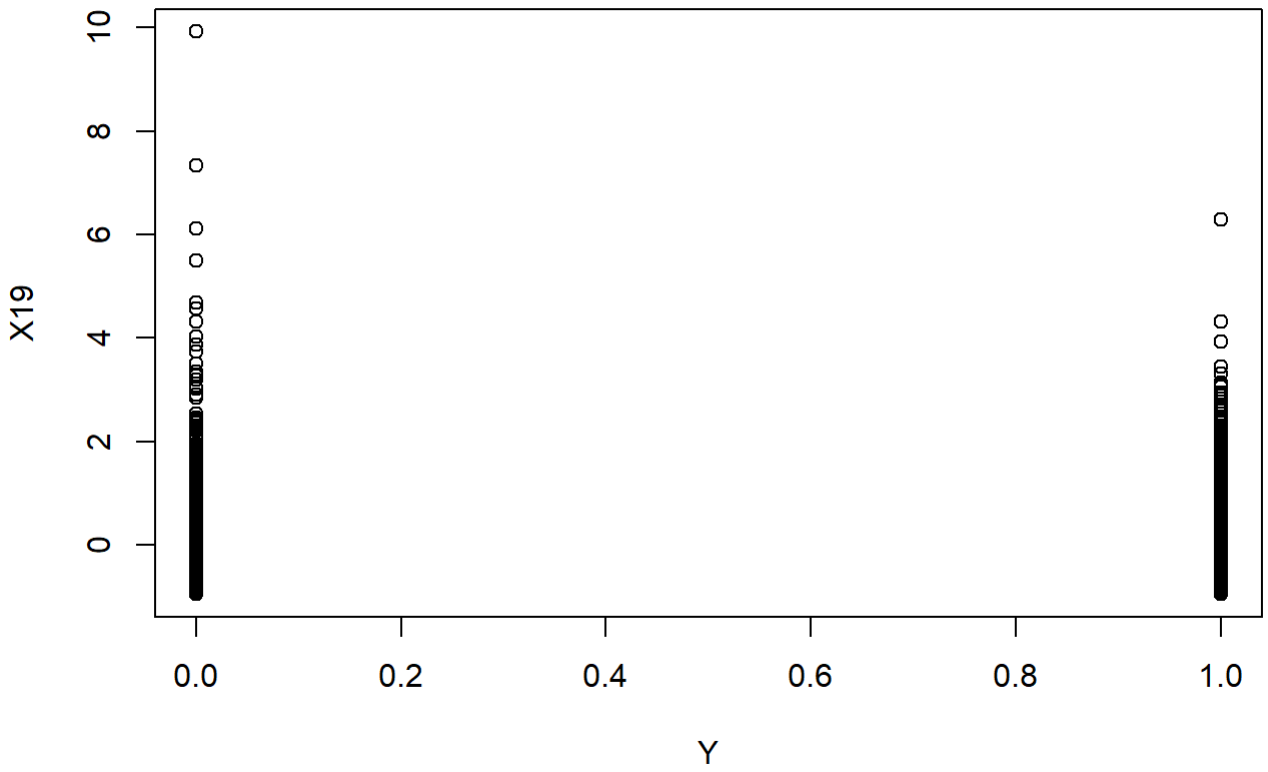

X14
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0
Y

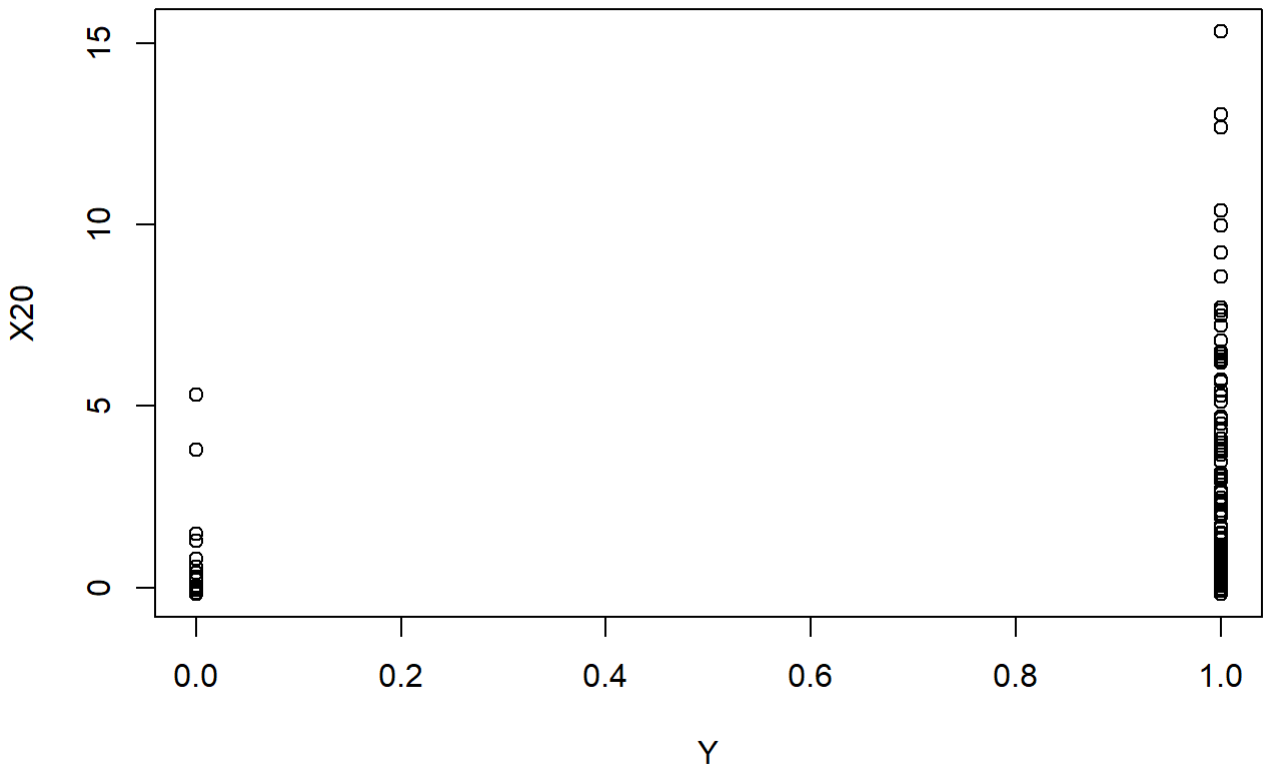

X15
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

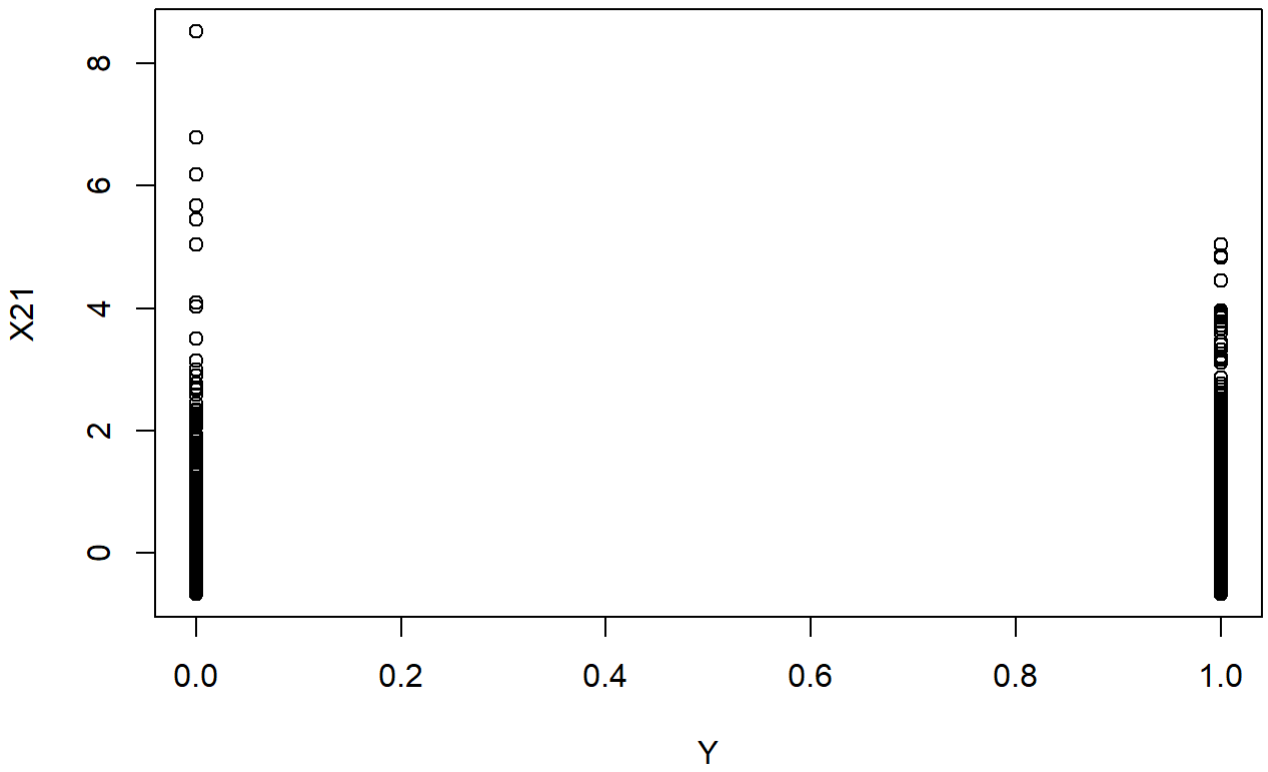

X16
0
5
10
15
20
0.0
0.2
0.4
0.6
0.8
1.0
Y

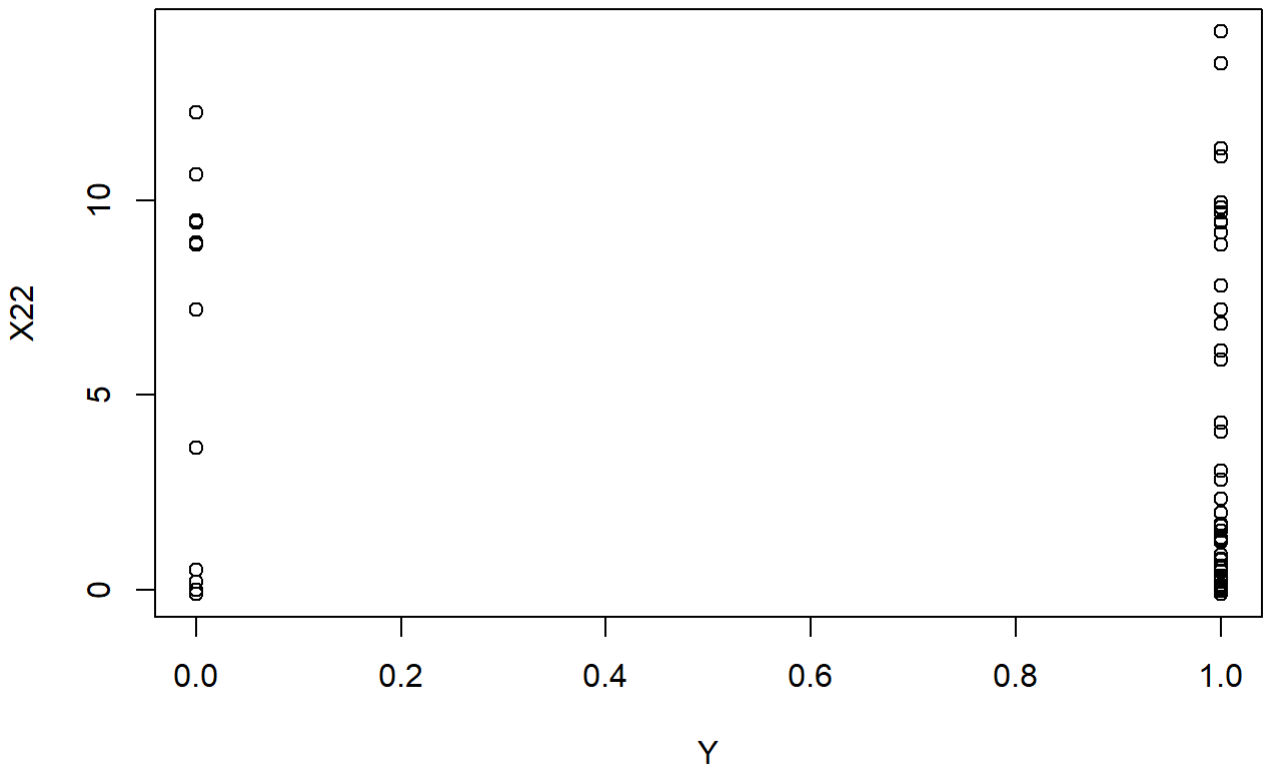

X17
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

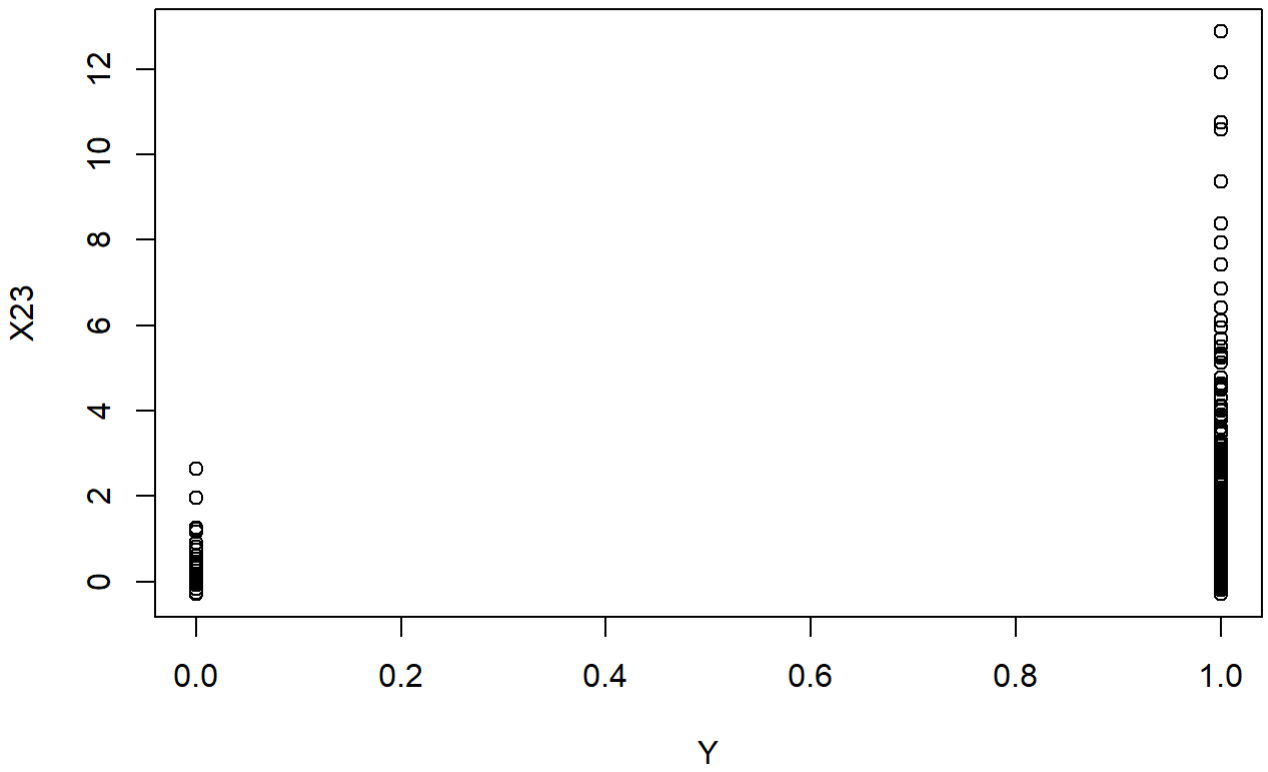

X18
0
2
4
6
8
0.0
0.2
0.4
0.6
0.8
1.0
Y

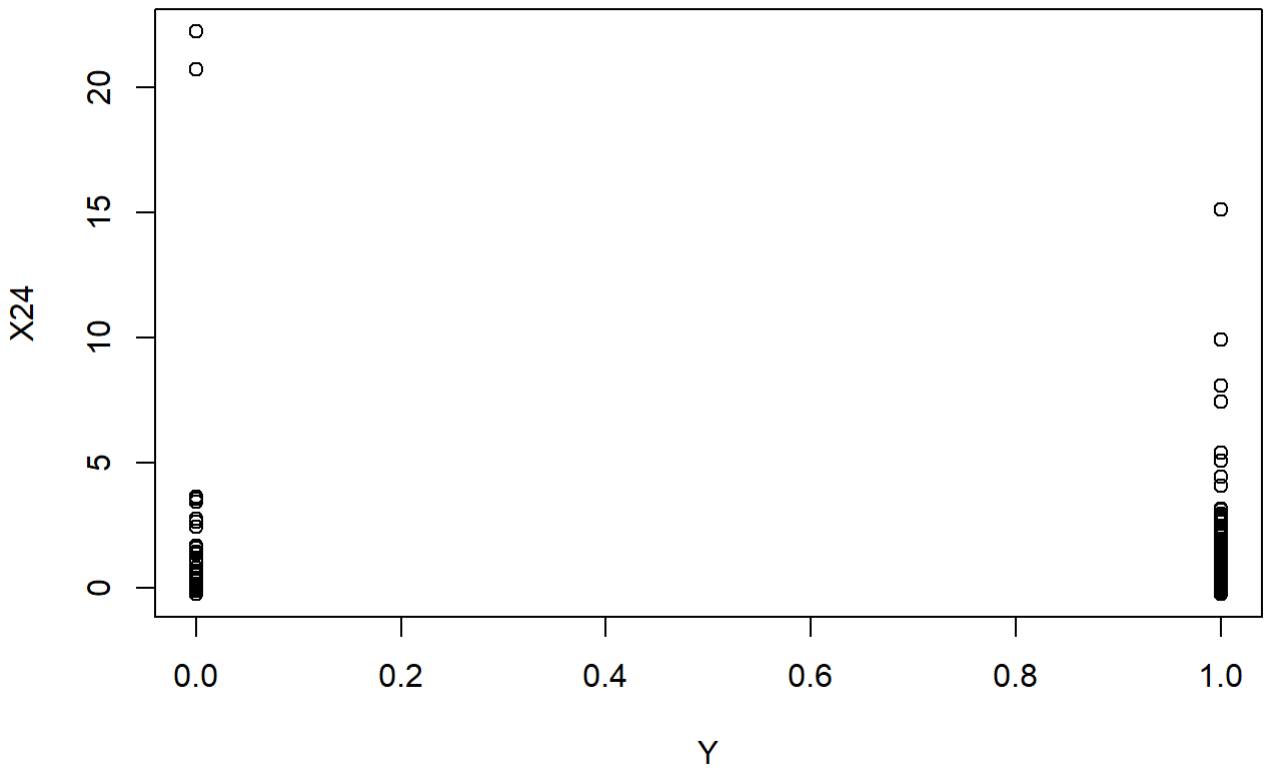

X19
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0
Y

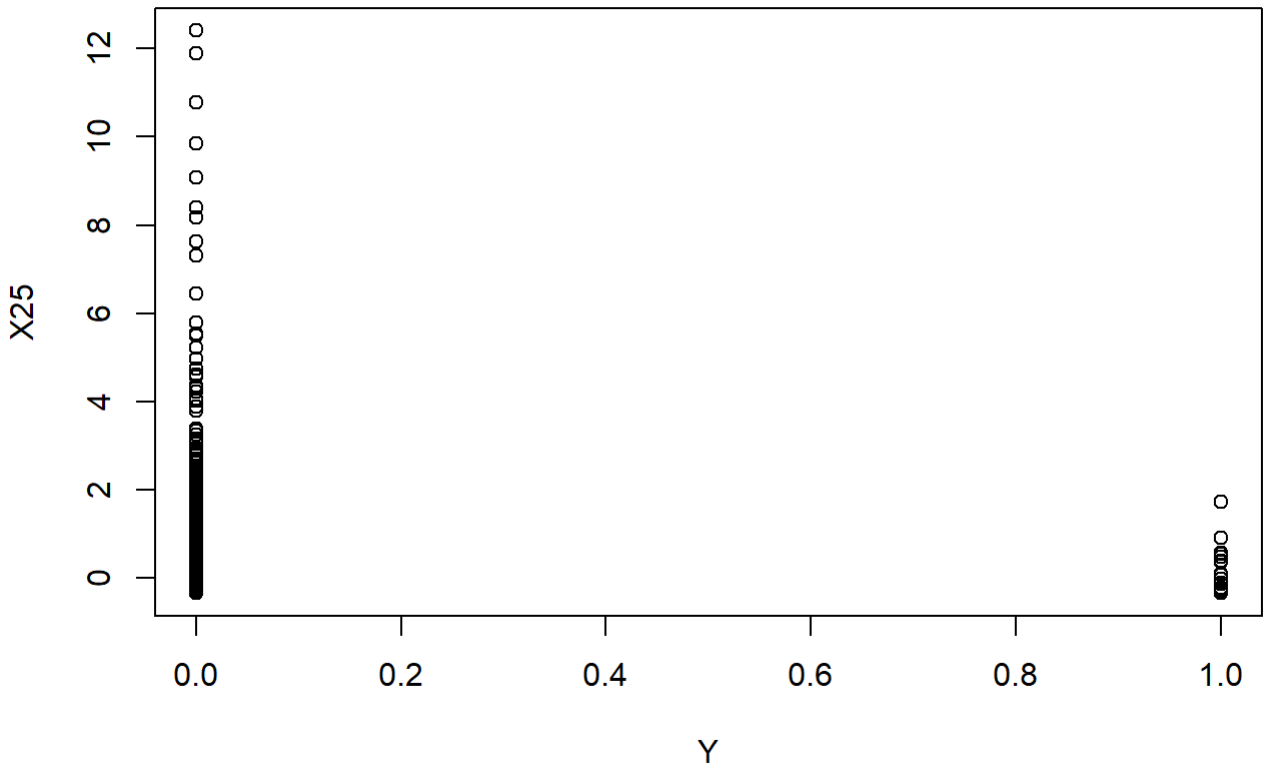

X20
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

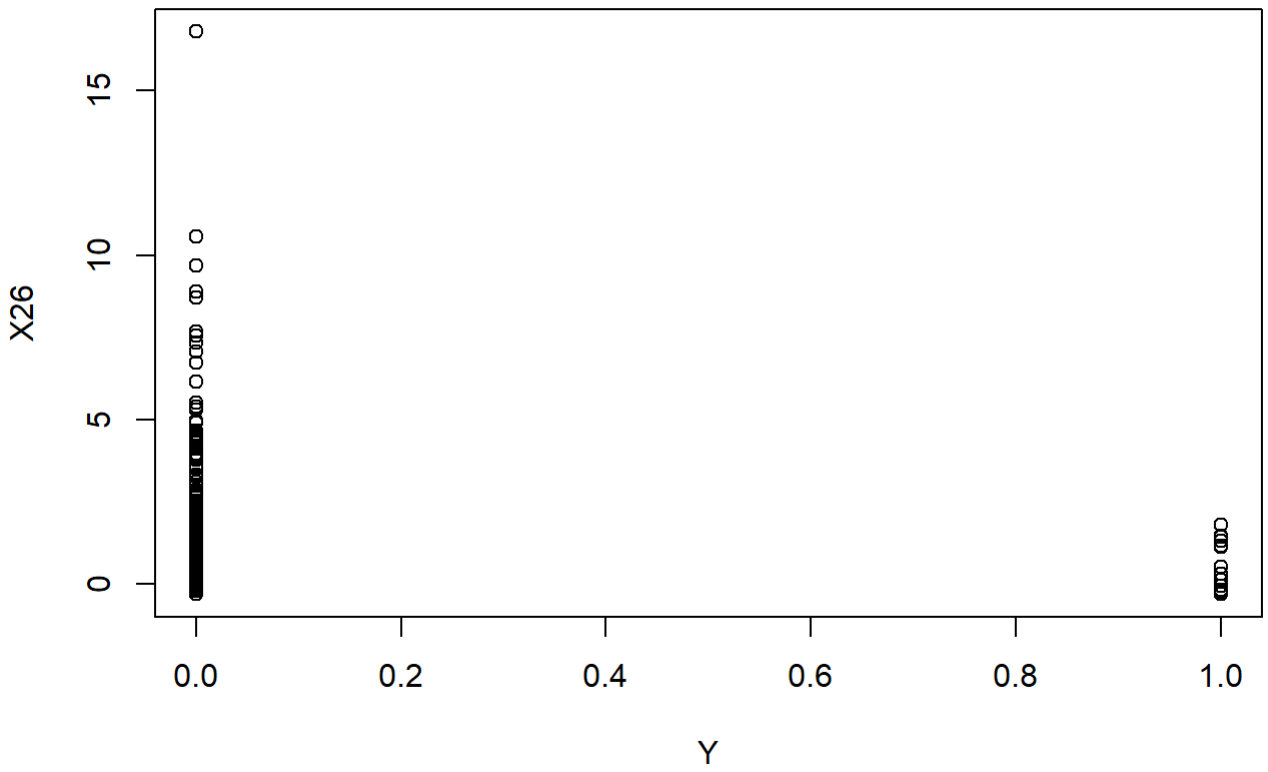

X21
0
2
4
6
8
0.0
0.2
0.4
0.6
0.8
1.0
Y

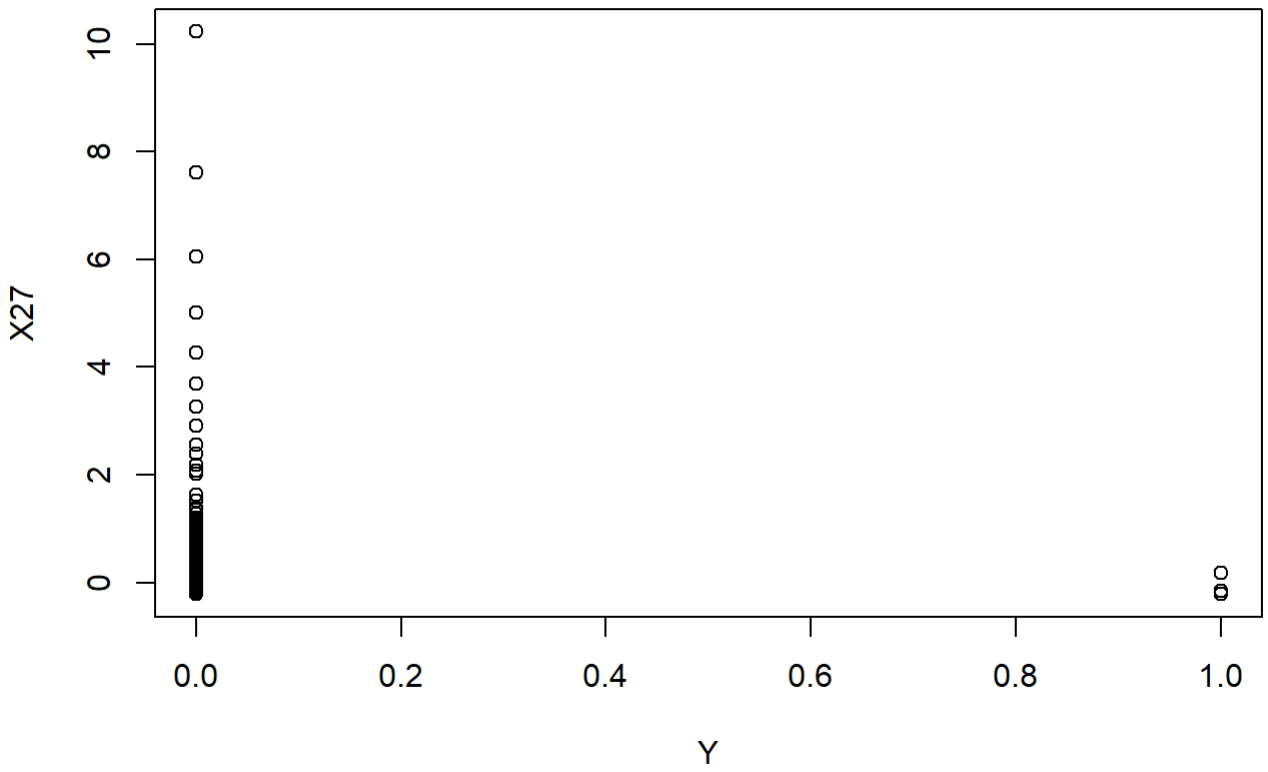

X22
0
5
10
0.0
0.2
0.4
0.6
0.8
1.0
Y

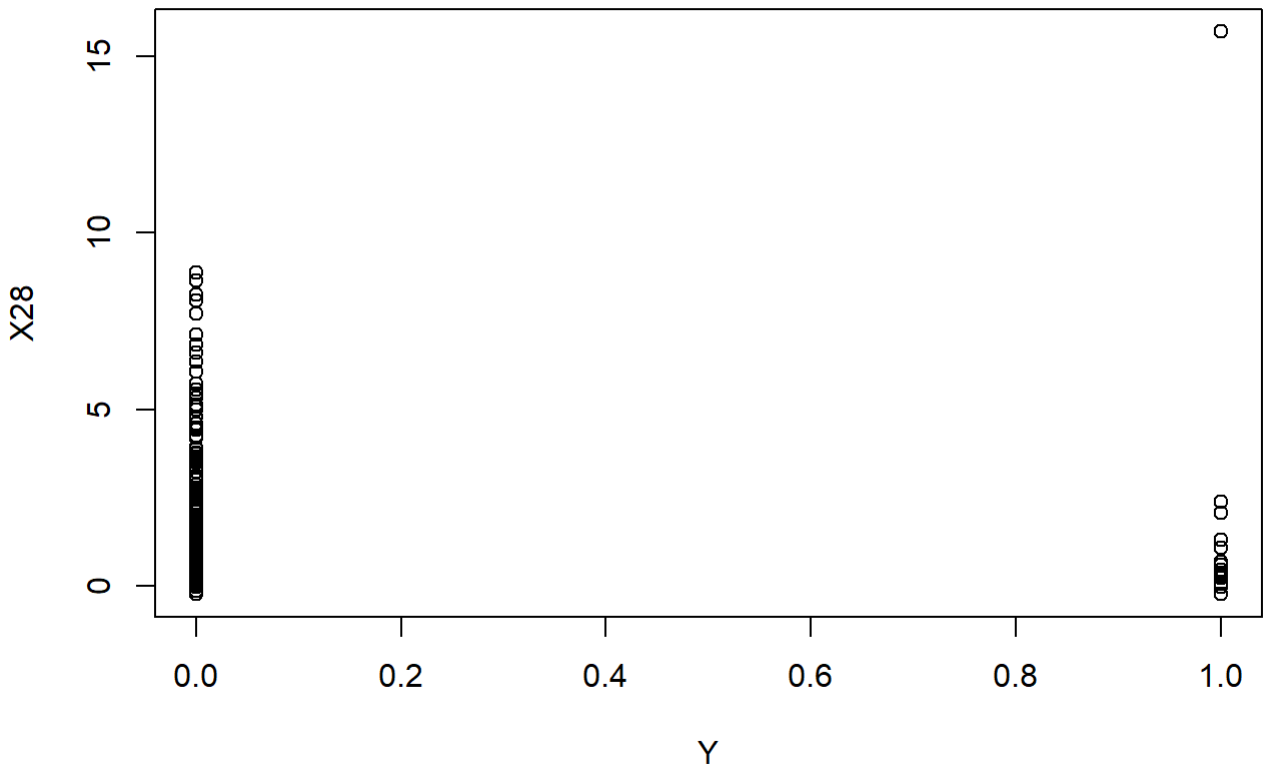

X23
0
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8
1.0
Y

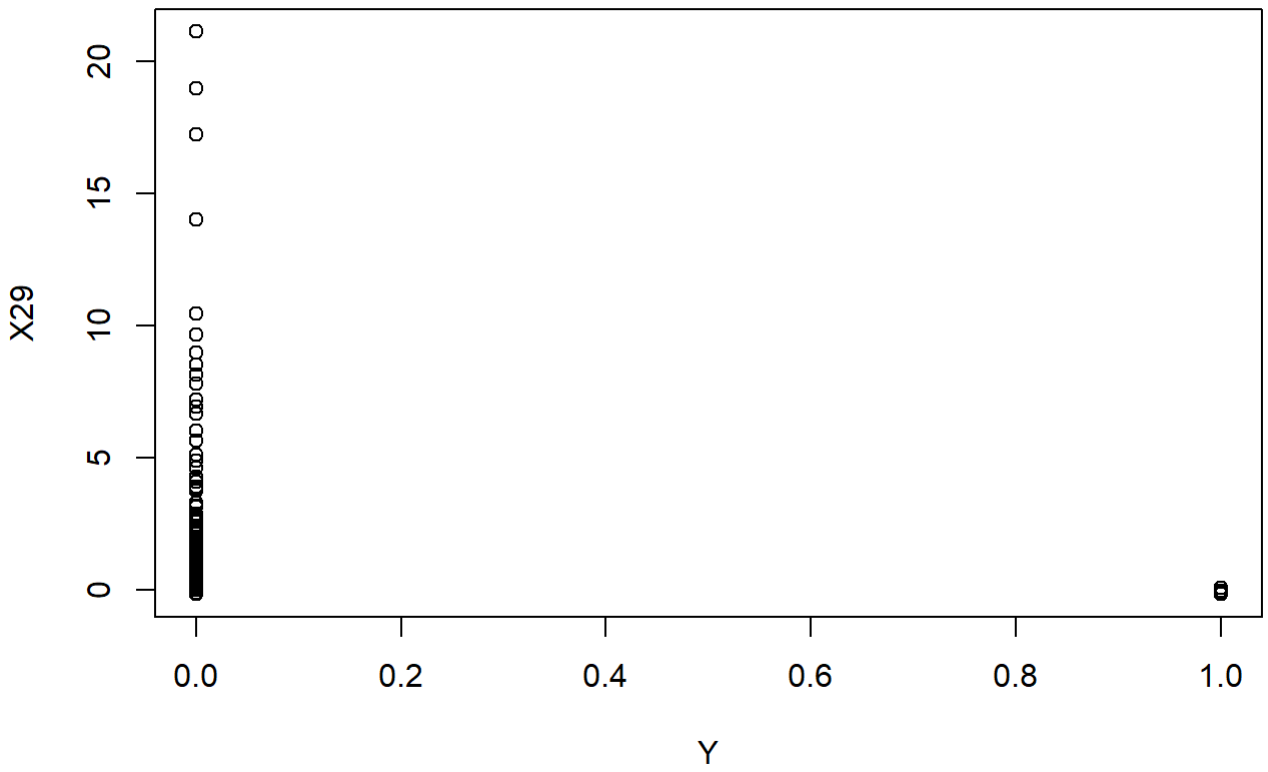

X24
0
5
10
15
20
0.0
0.2
0.4
0.6
0.8
1.0
Y

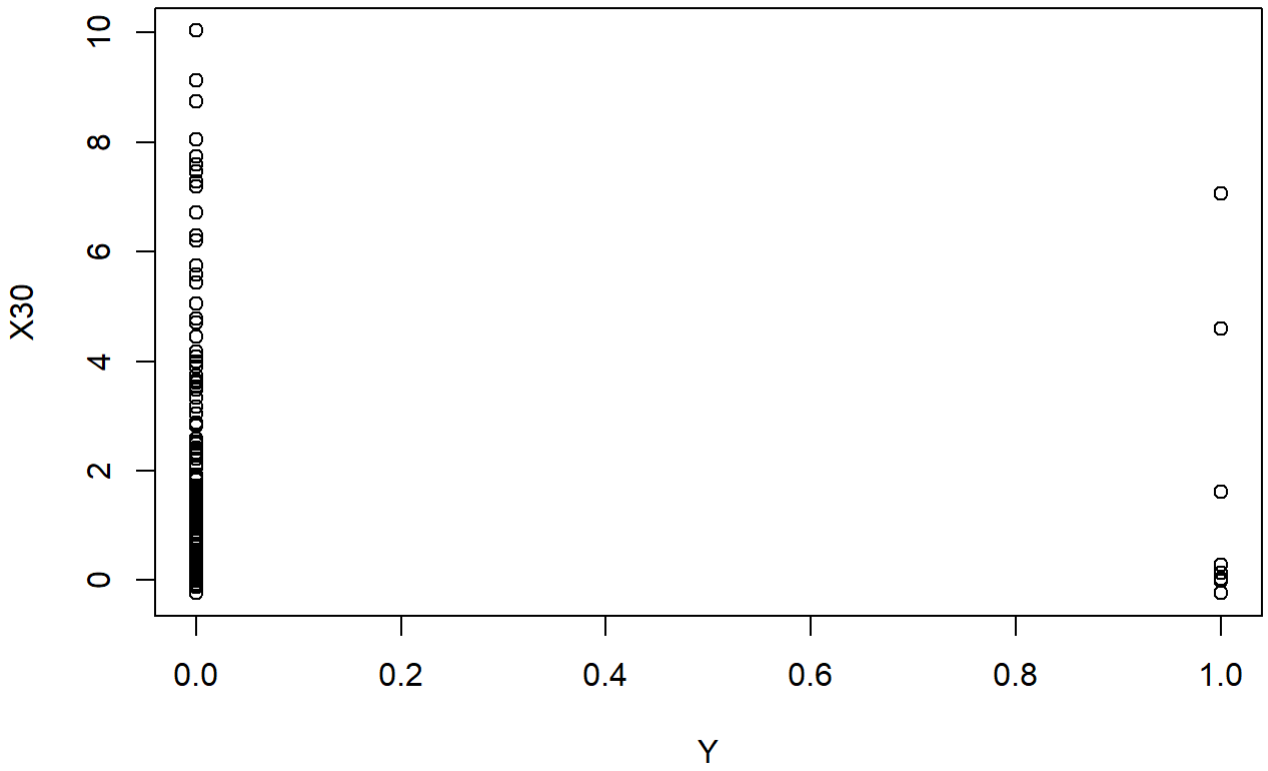

X25
0
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8
1.0
Y

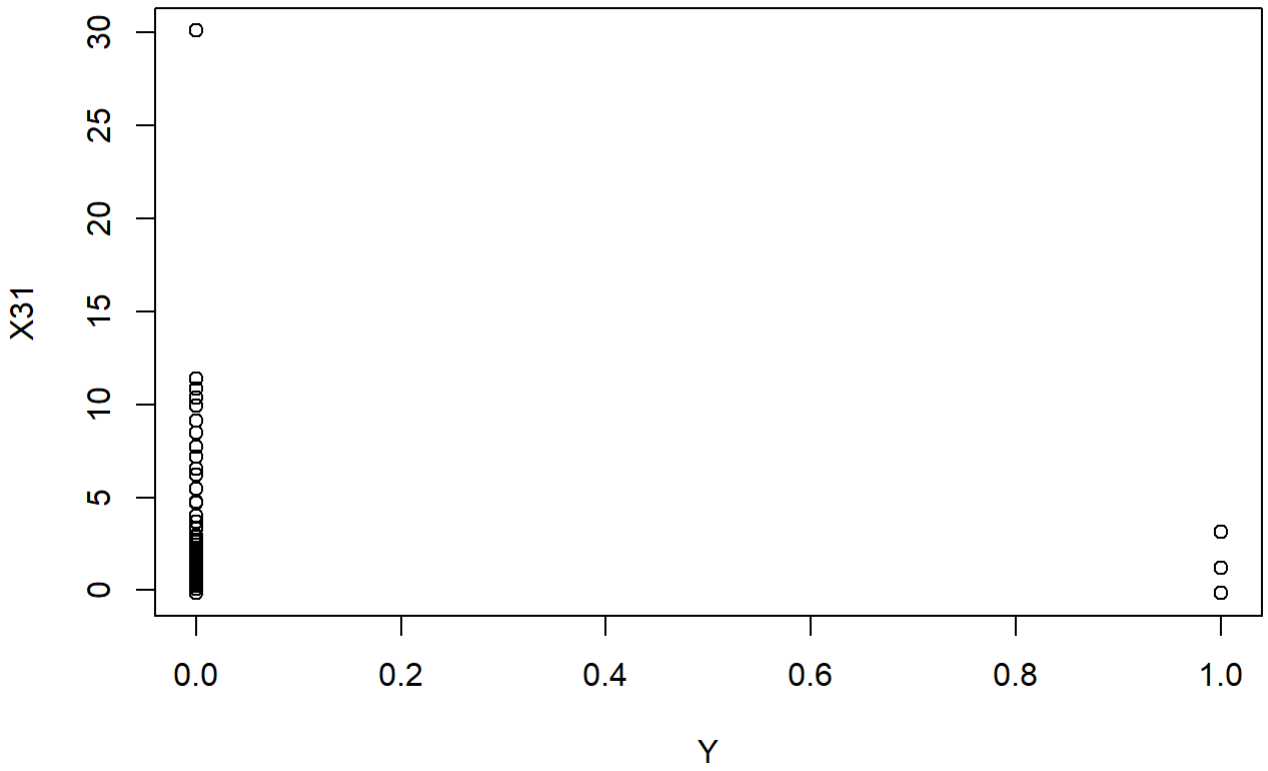

X26
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

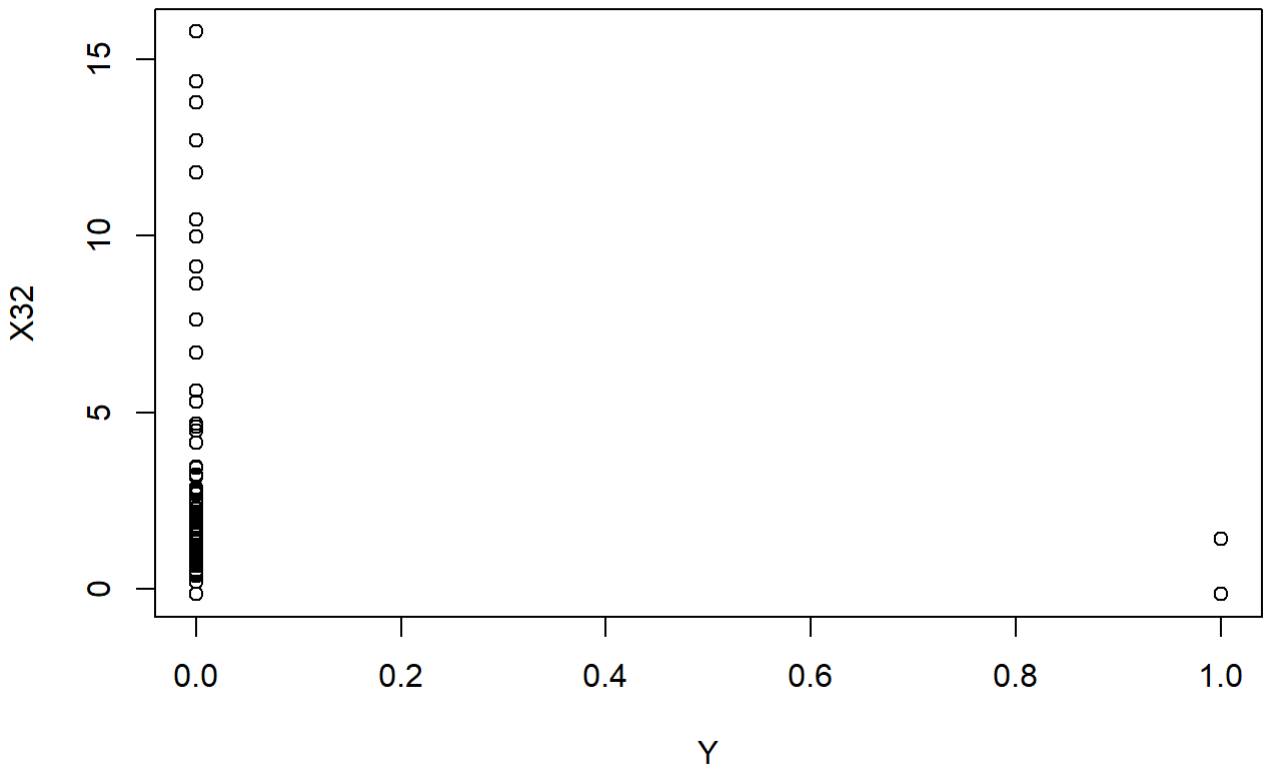

X27
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0
Y

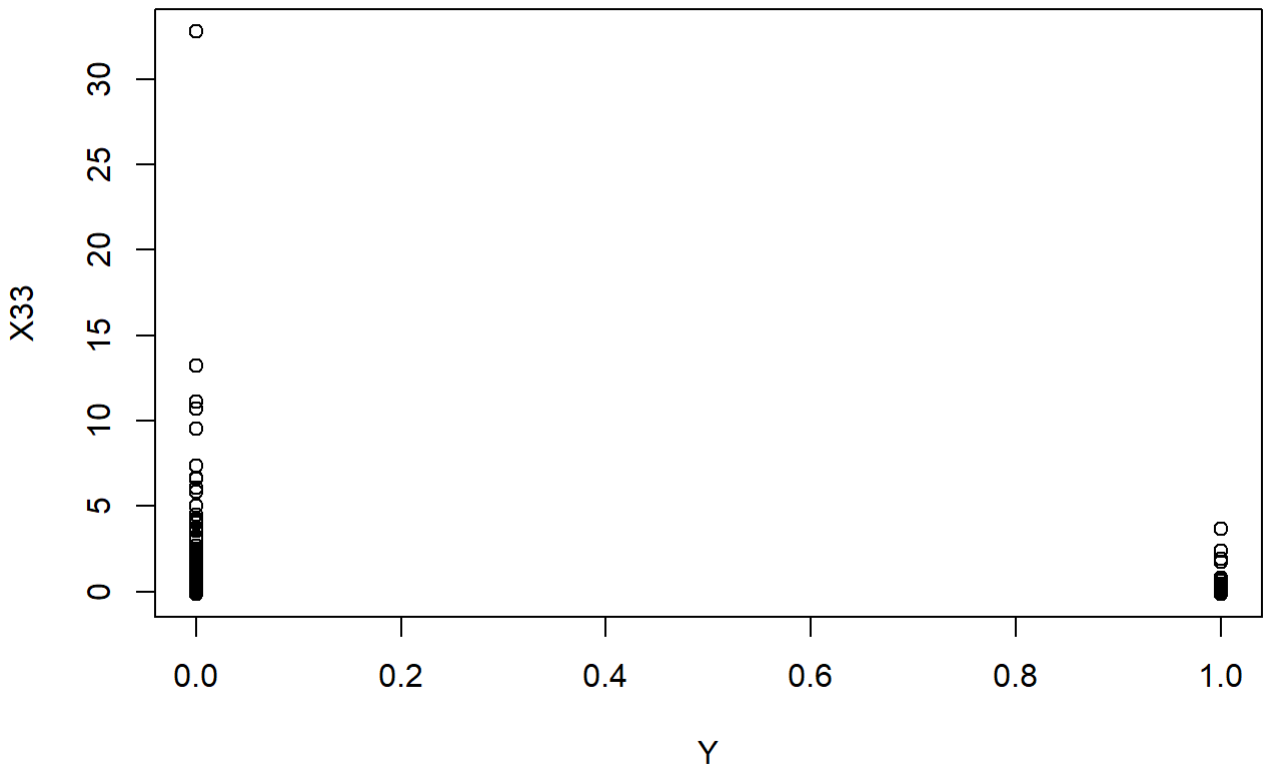

X28
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

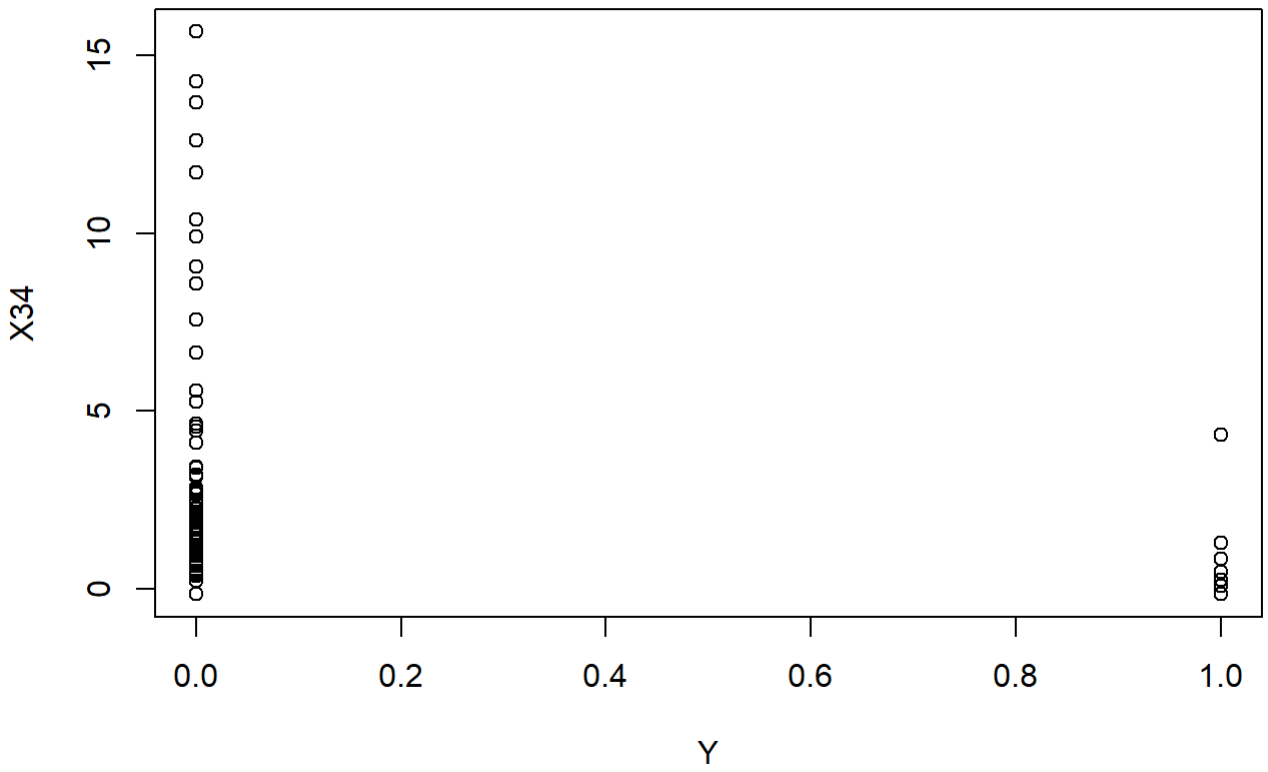

X29
0
5
10
15
20
0.0
0.2
0.4
0.6
0.8
1.0
Y

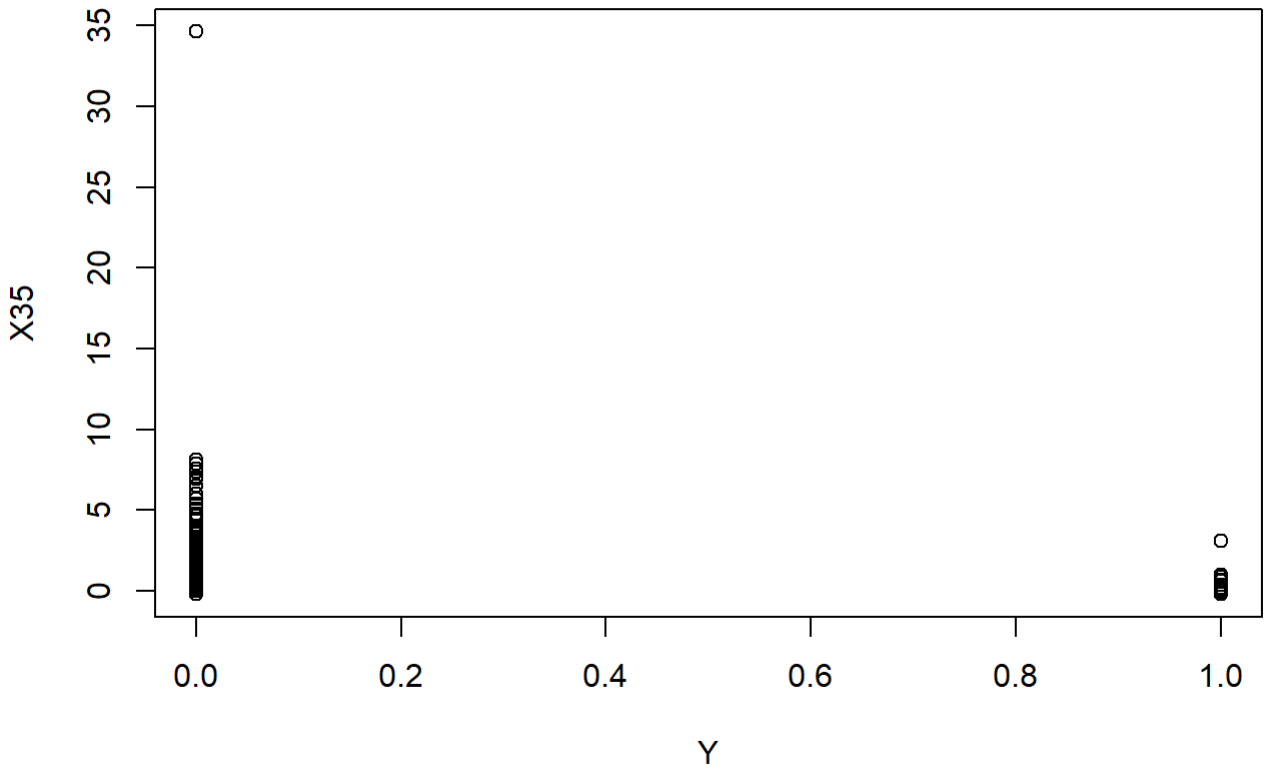

X30
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0
Y

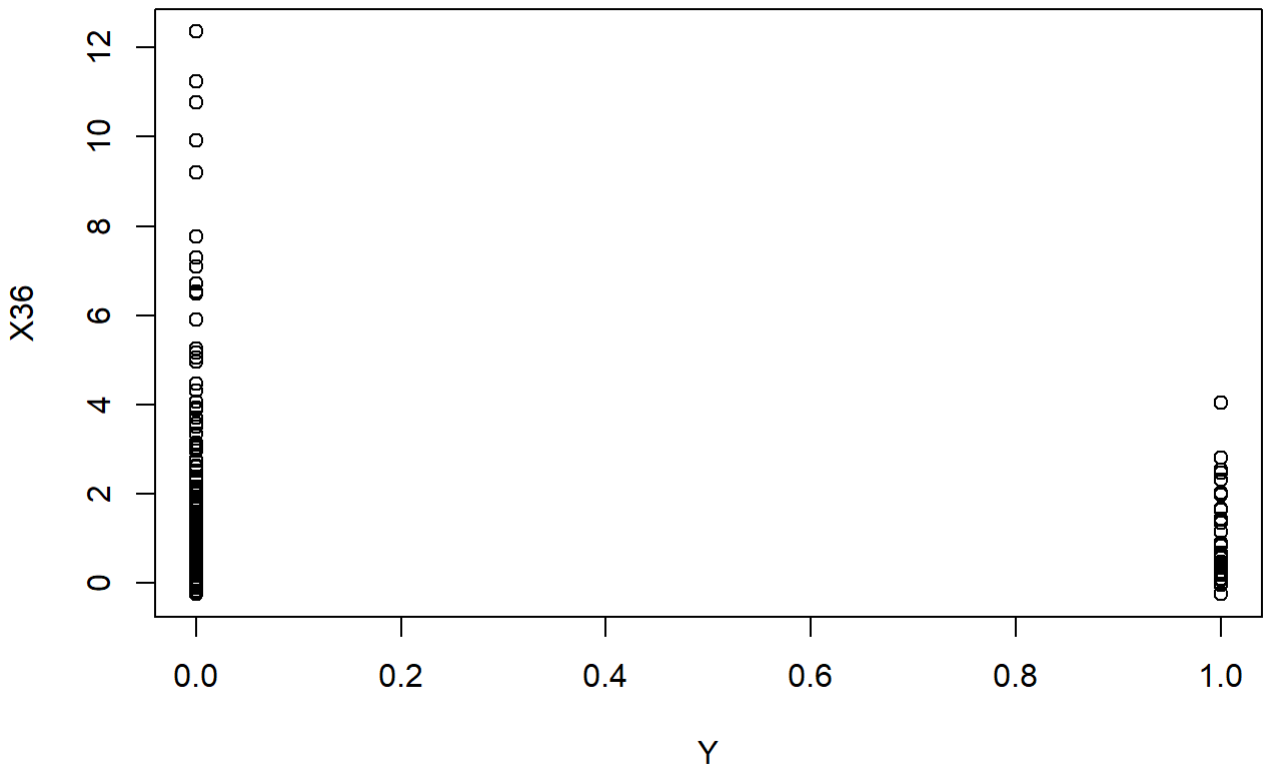

X31
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0
Y

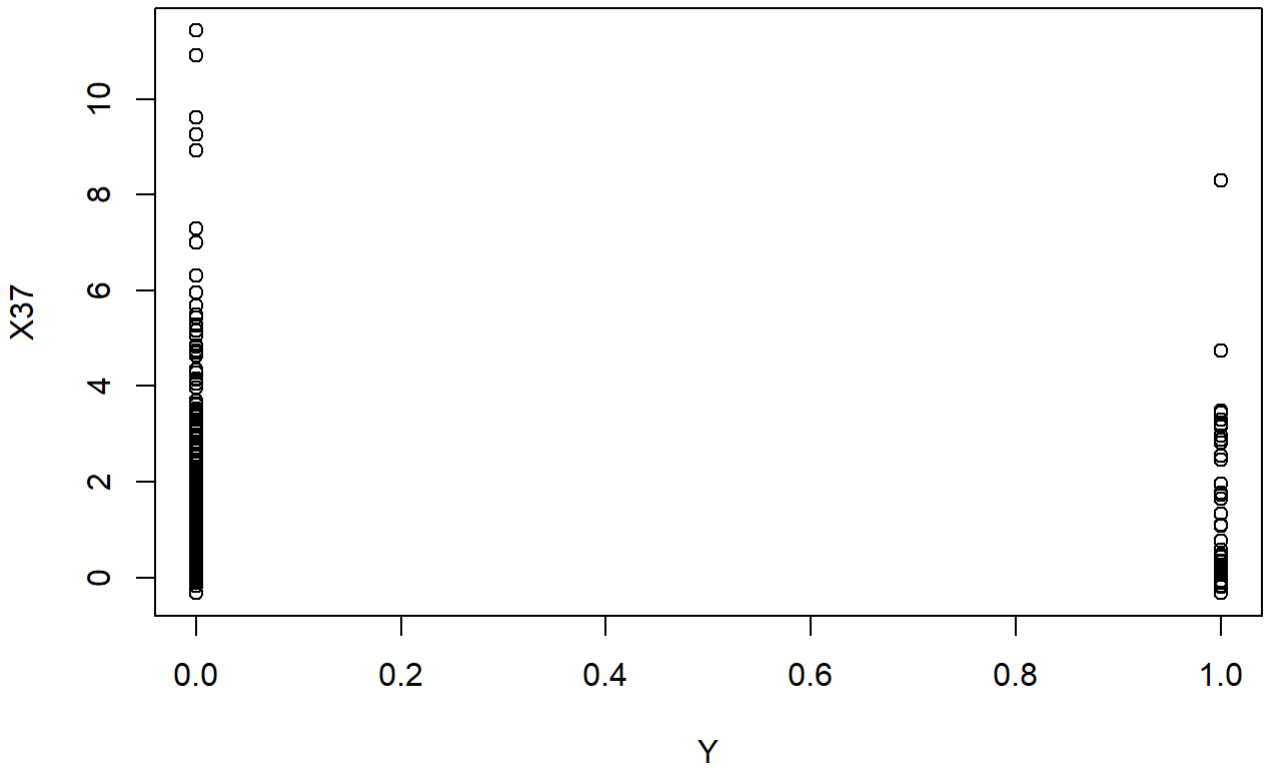

X32
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

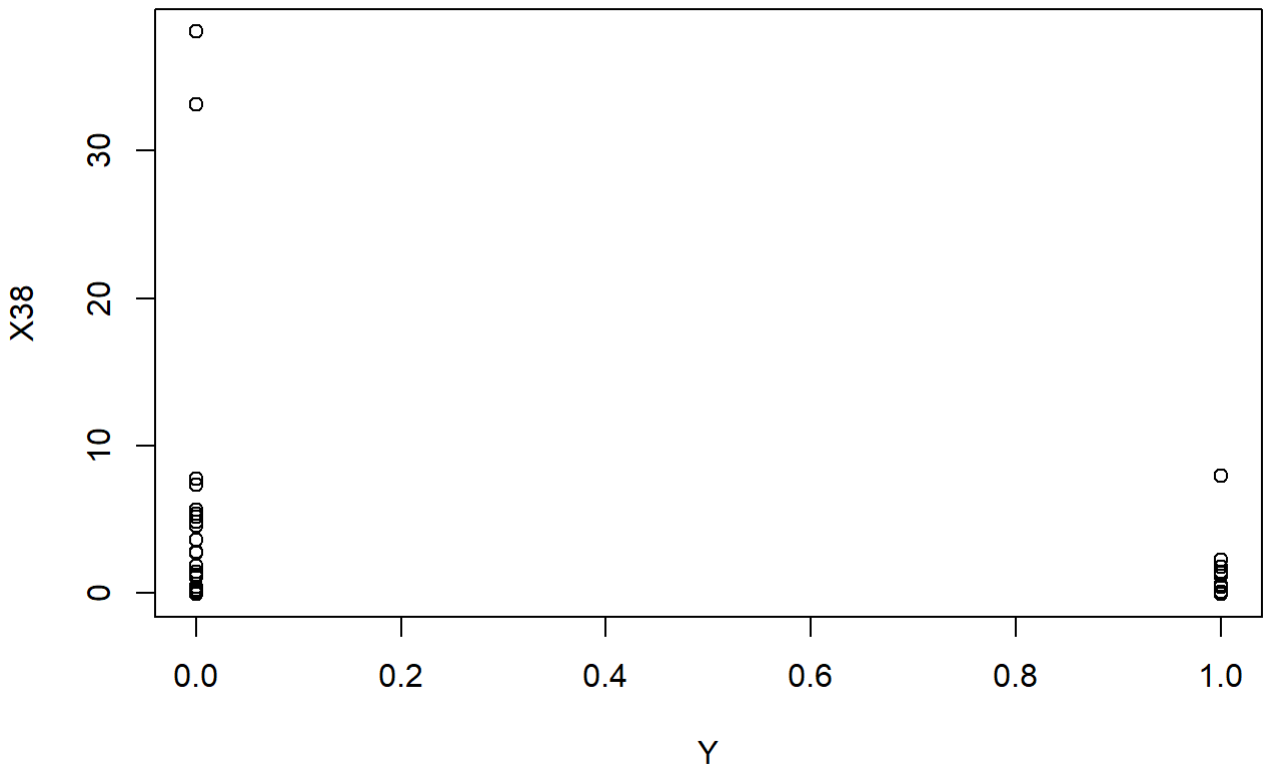

X33
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0
Y

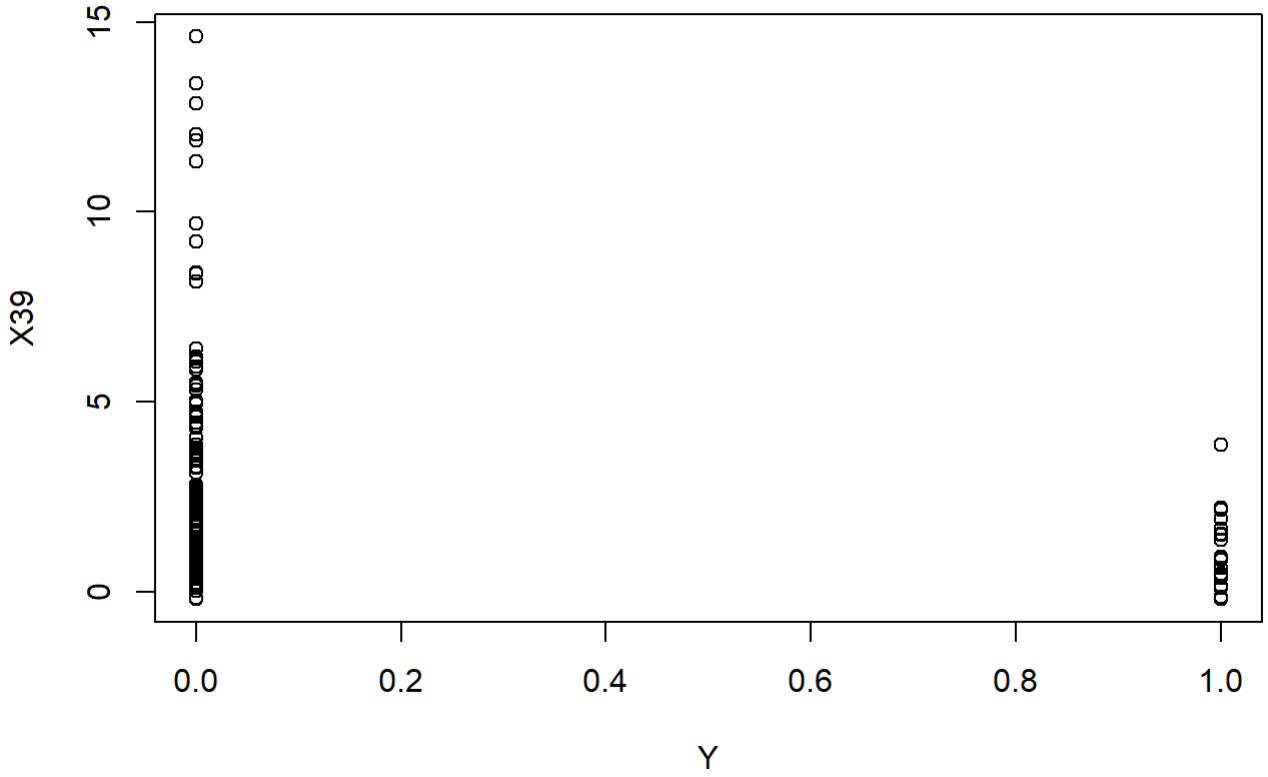

X34
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

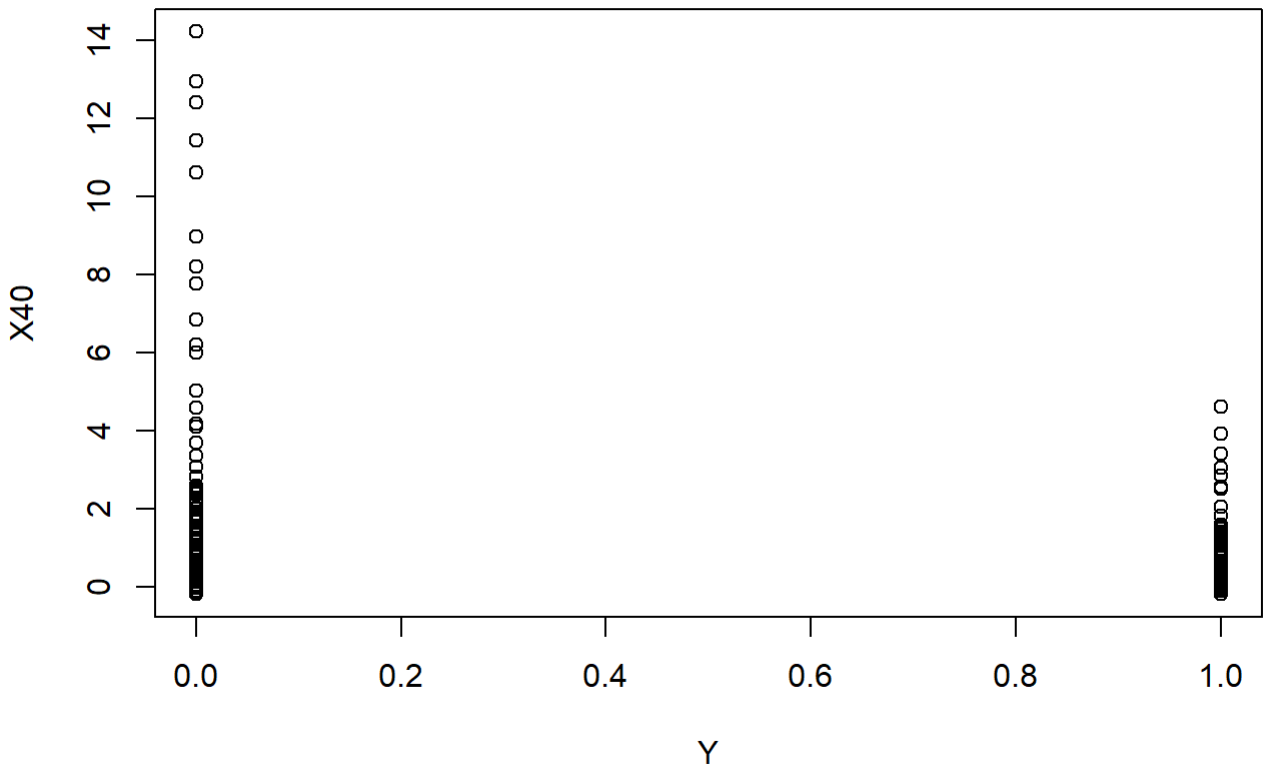

X35
0
5
10
15
20
25
30
35
0.0
0.2
0.4
0.6
0.8
1.0
Y

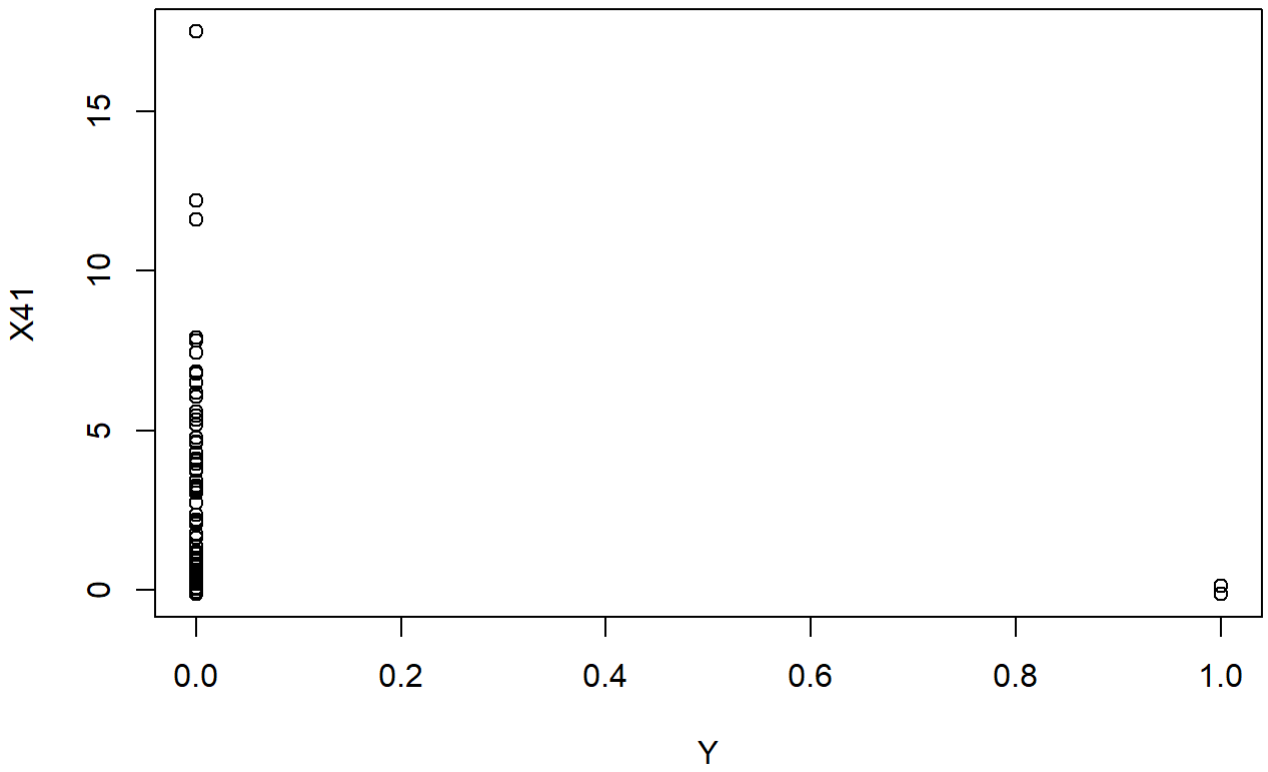

X36
0
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8
1.0
Y

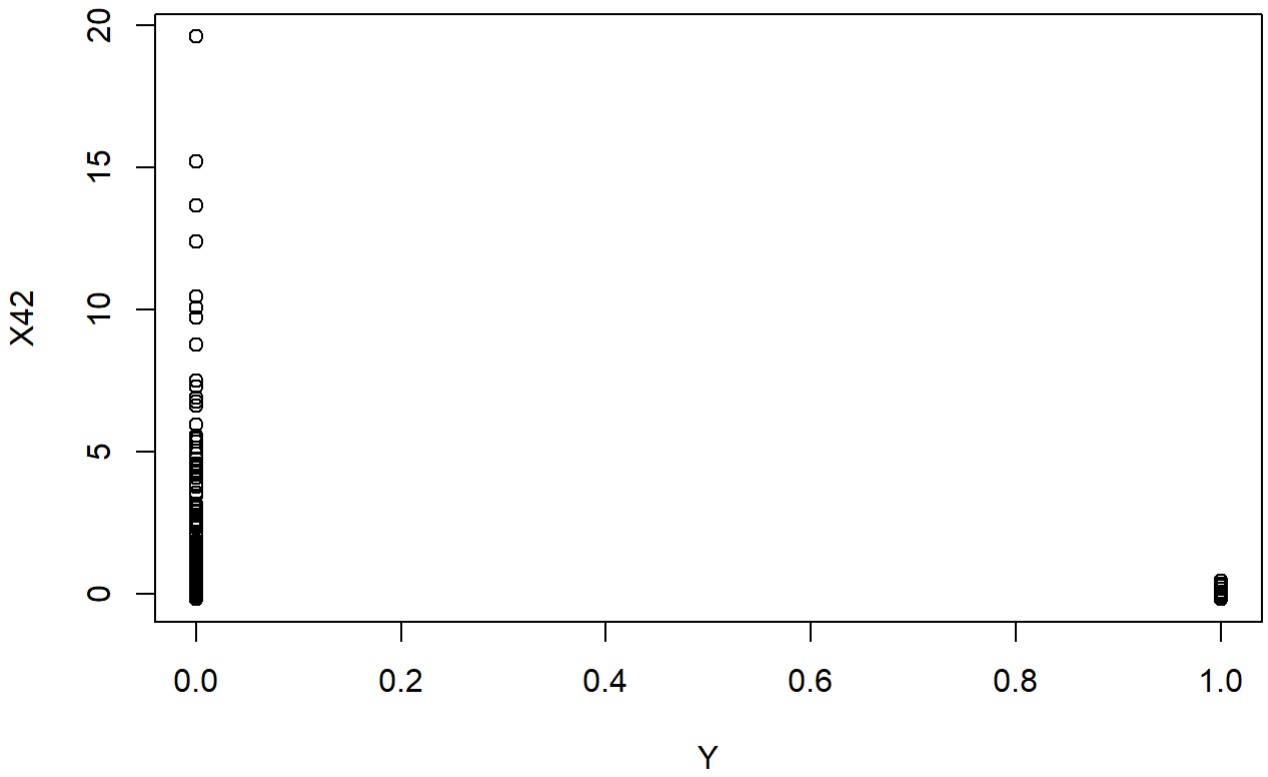

X37
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0
Y

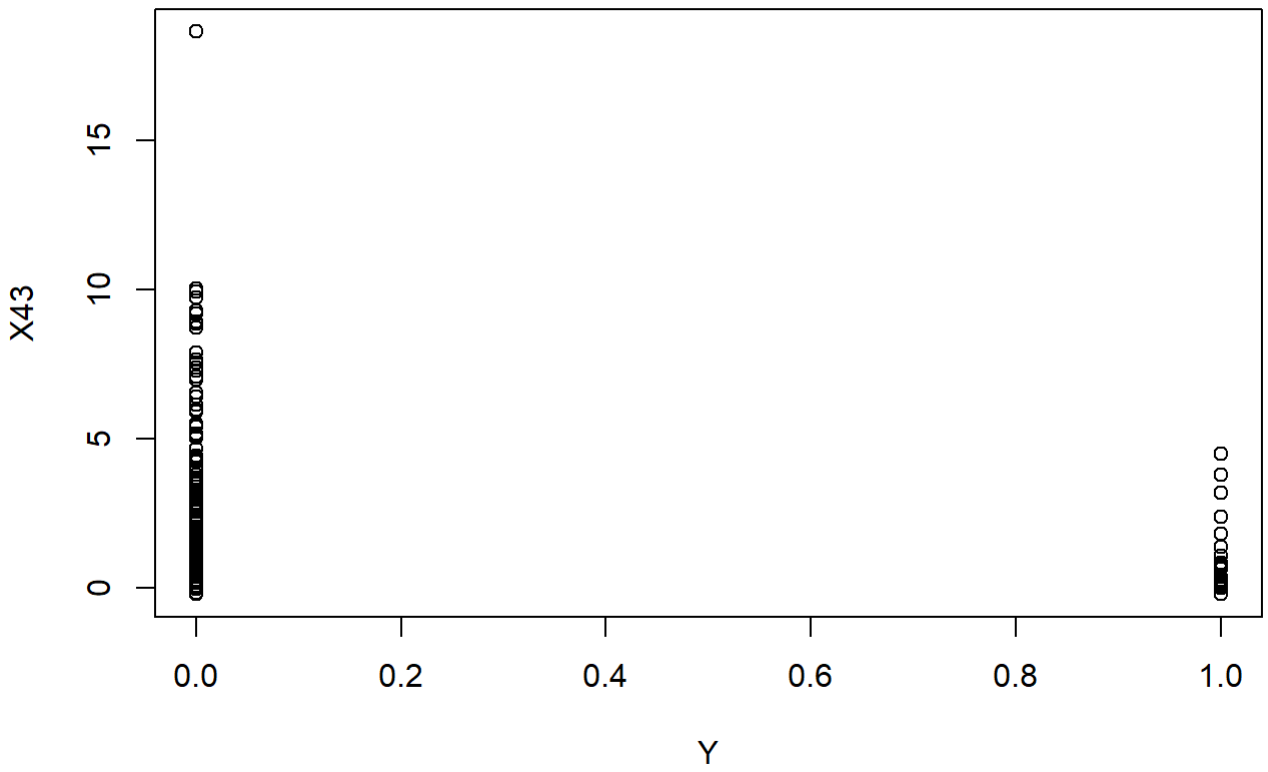

X38
0
10
20
30
0.0
0.2
0.4
0.6
0.8
1.0
Y

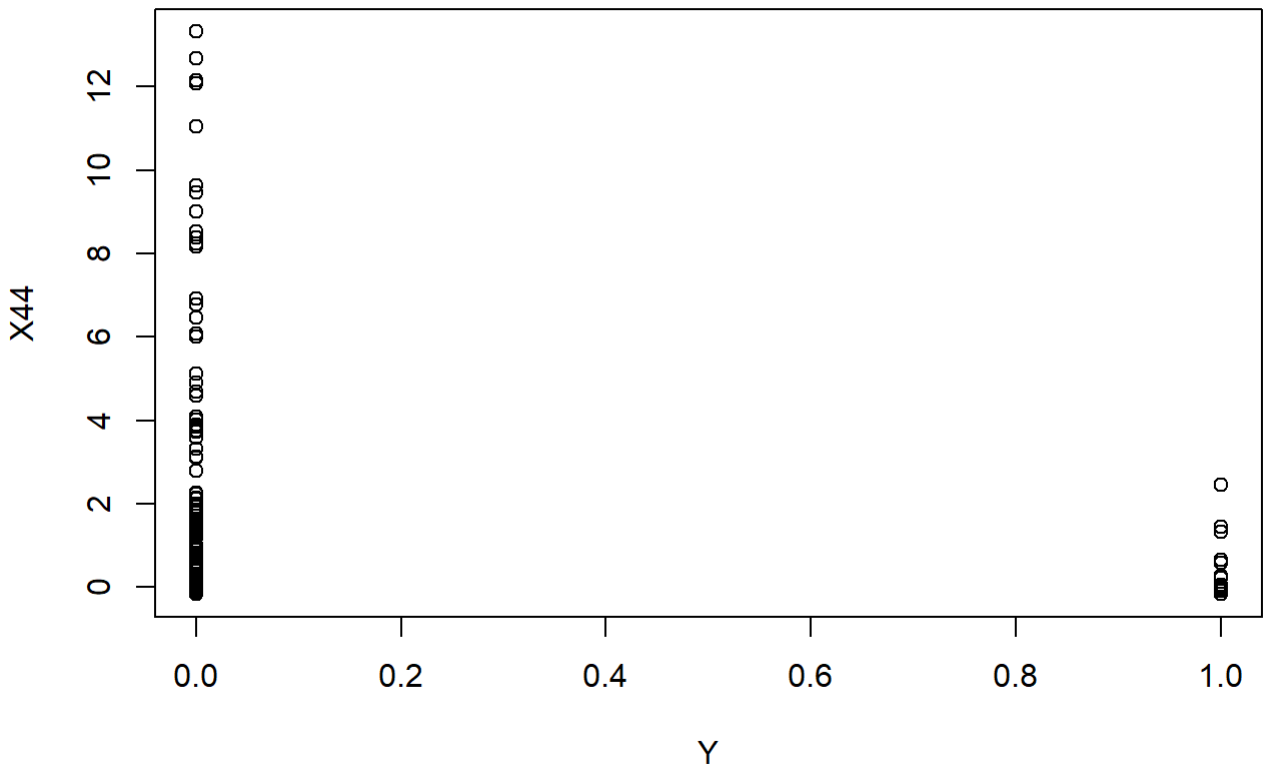

X39
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0
Y

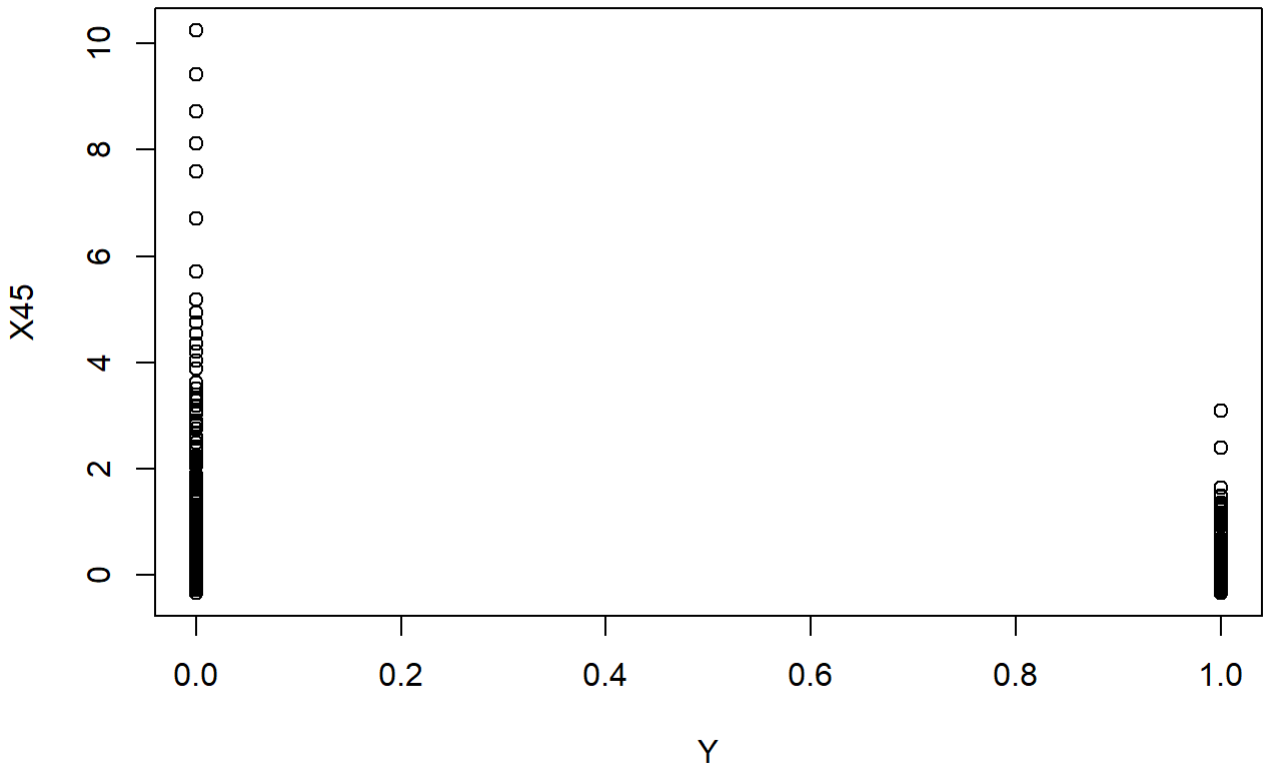

X40
0
2
4
6
8
10
12
14
0.0
0.2
0.4
0.6
0.8
1.0
Y

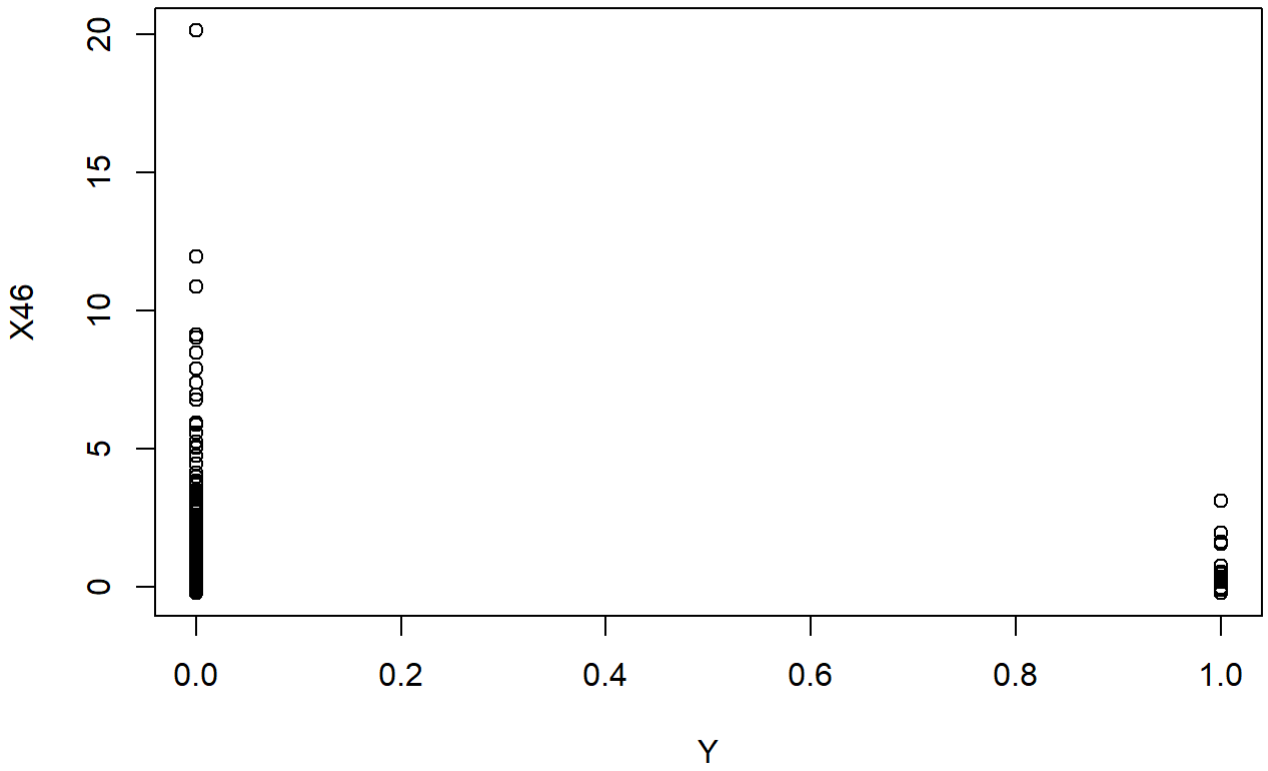

X41
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

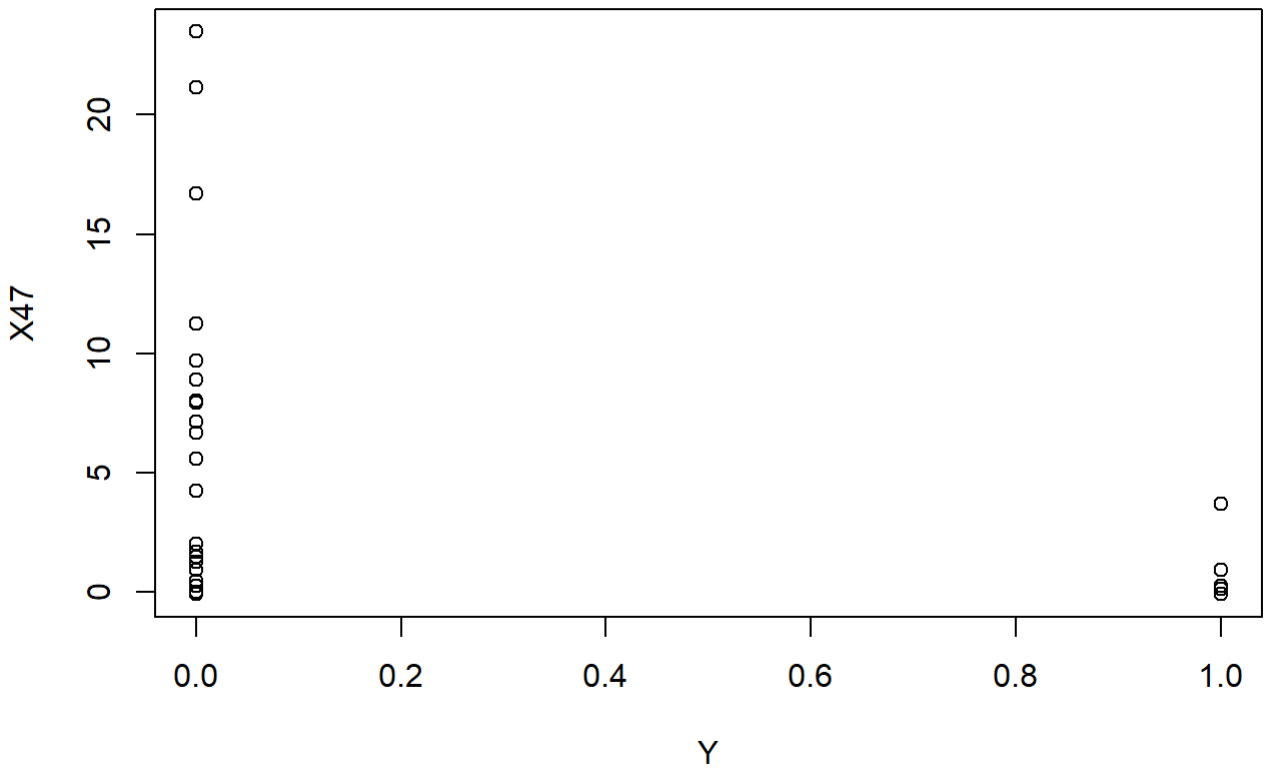

X42
0
5
10
15
20
0.0
0.2
0.4
0.6
0.8
1.0
Y

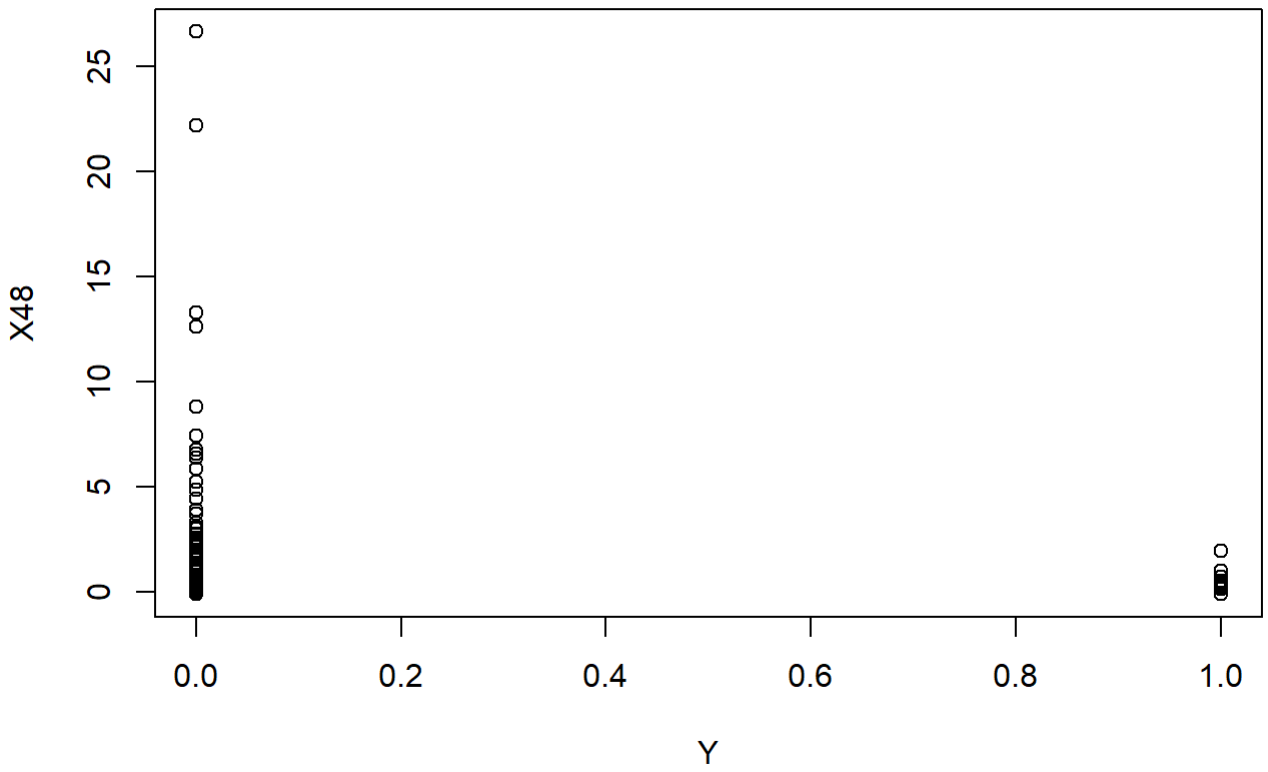

X43
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

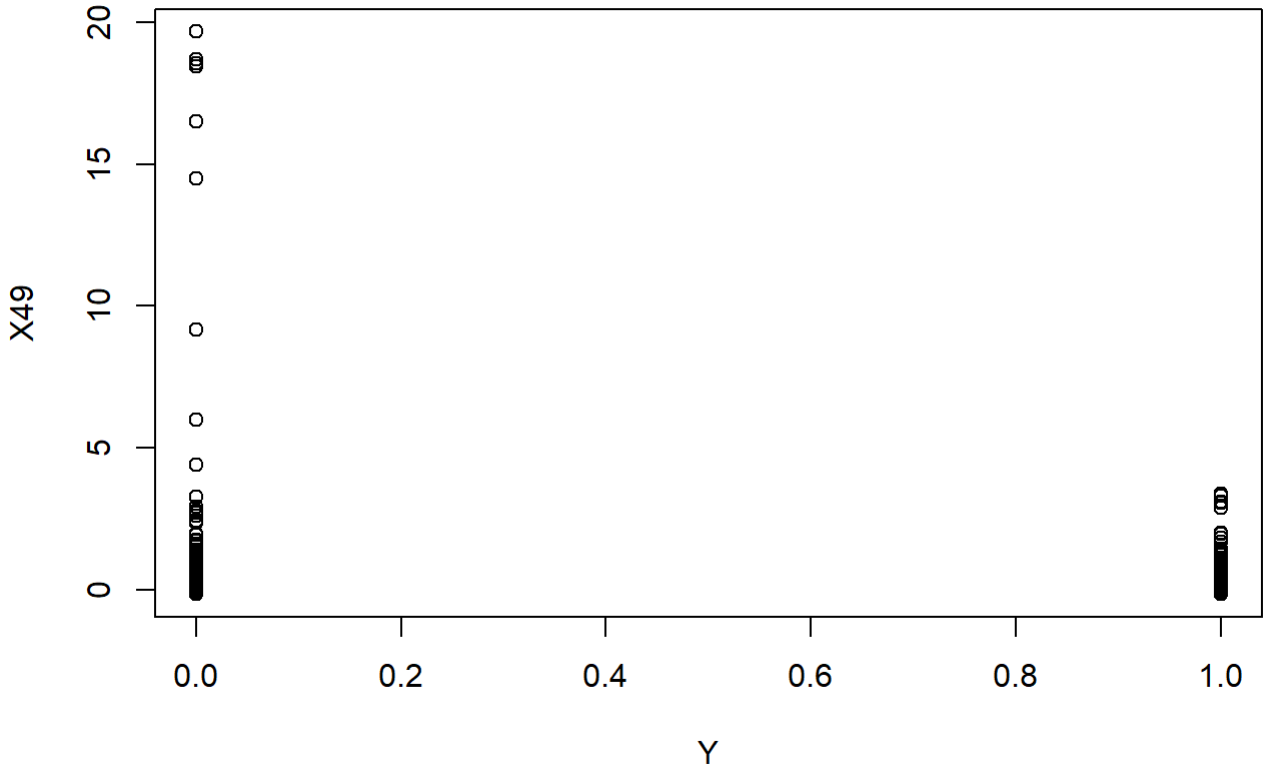

X44
0
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8
1.0
Y

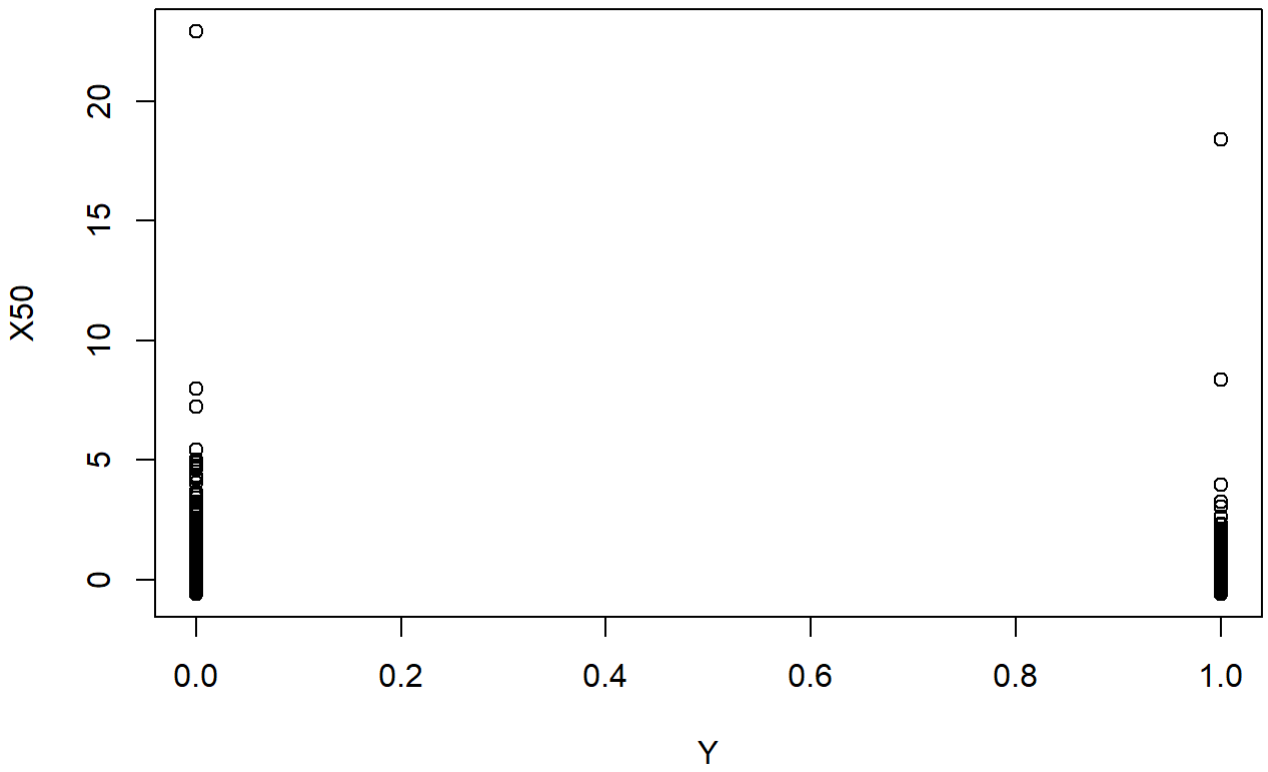

X45
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0
Y

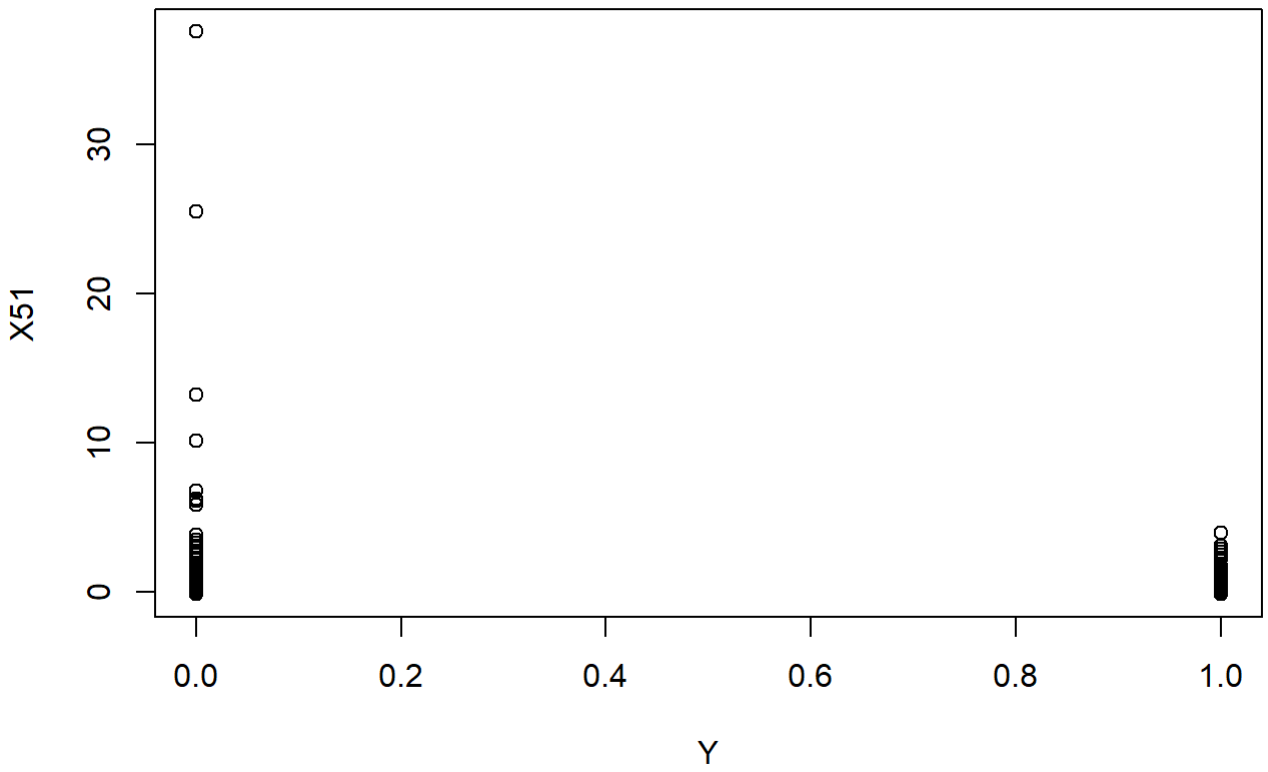

X46
0
5
10
15
20
0.0
0.2
0.4
0.6
0.8
1.0
Y

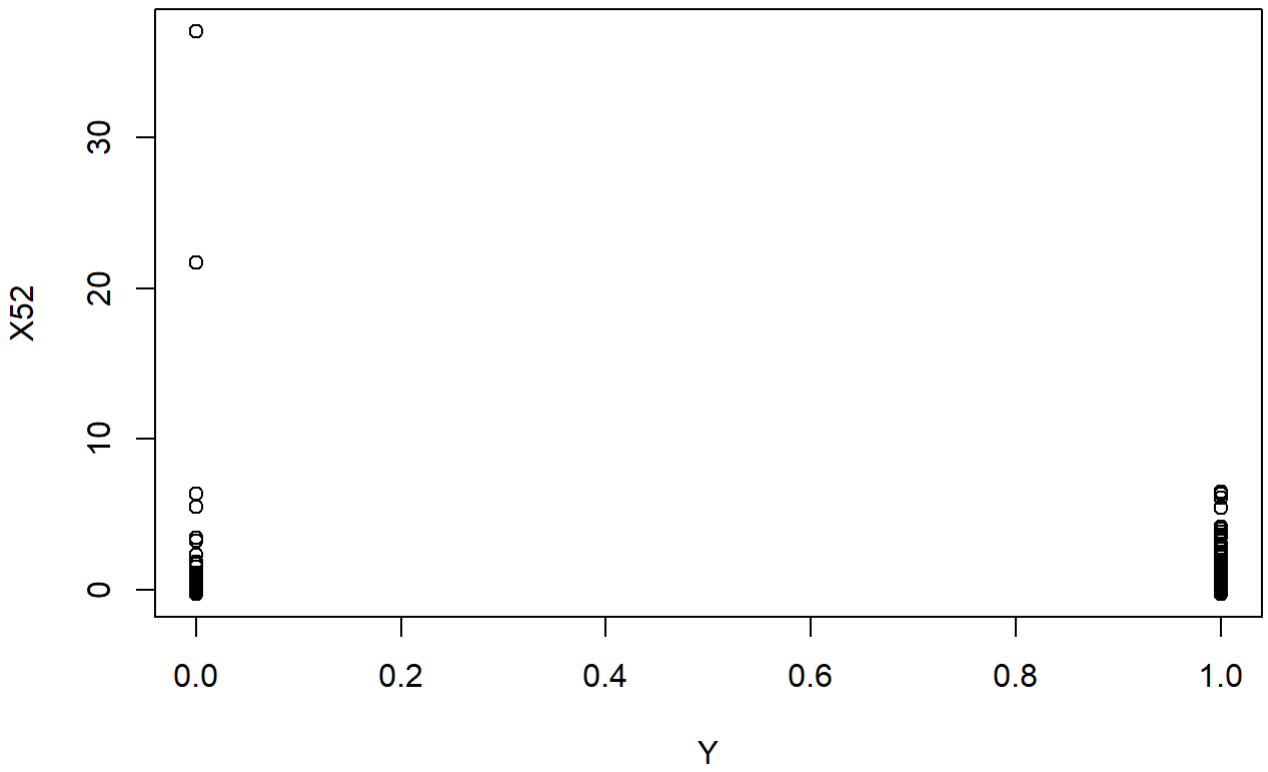

X47
0
5
10
15
20
0.0
0.2
0.4
0.6
0.8
1.0
Y

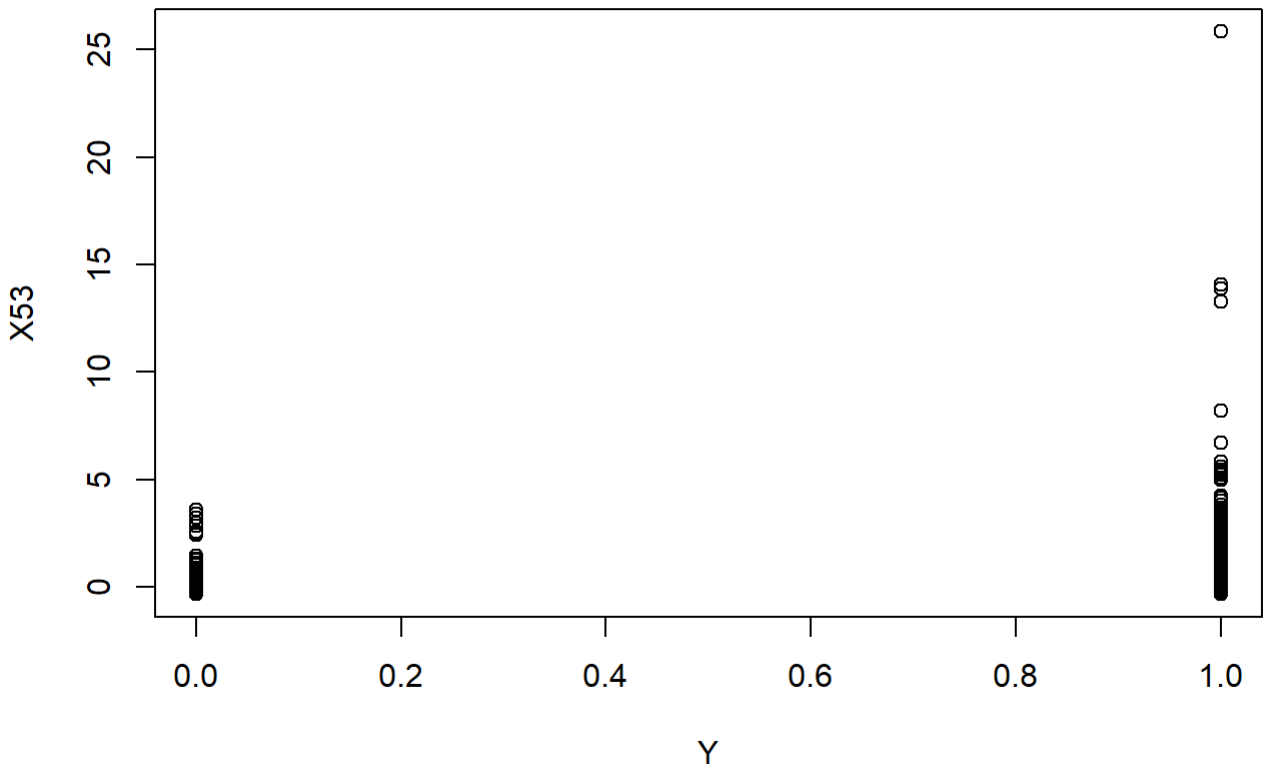

X48
0
5
10
15
20
25
0.0
0.2
0.4
0.6
0.8
1.0
Y

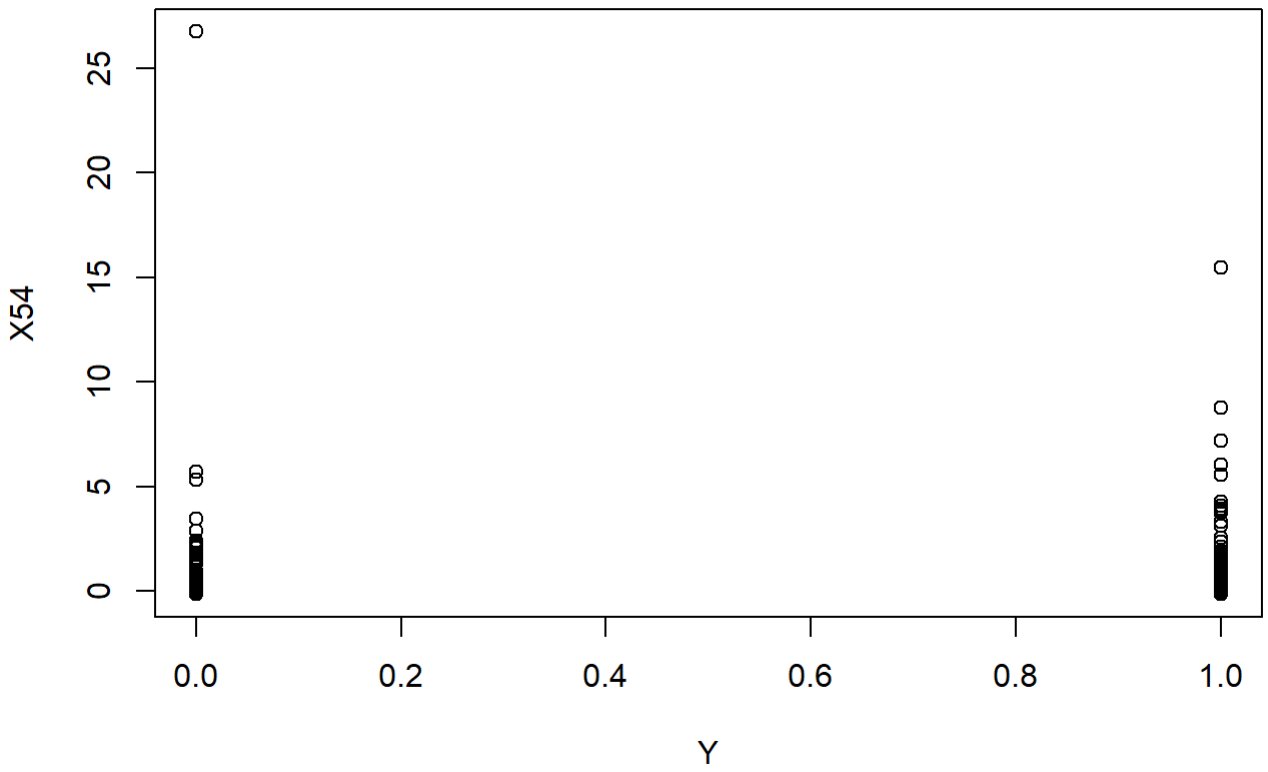

X49
0
5
10
15
20
0.0
0.2
0.4
0.6
0.8
1.0
Y

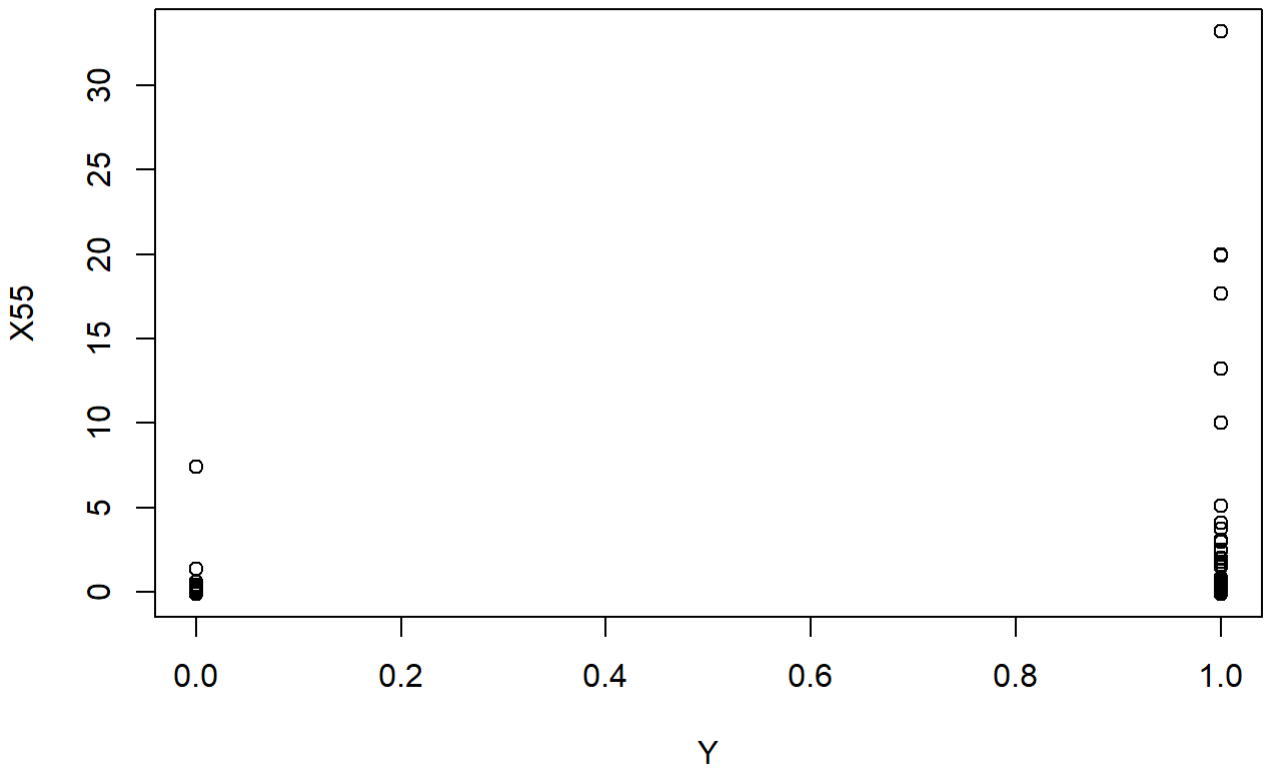

X50
0
5
10
15
20
0.0
0.2
0.4
0.6
0.8
1.0
Y

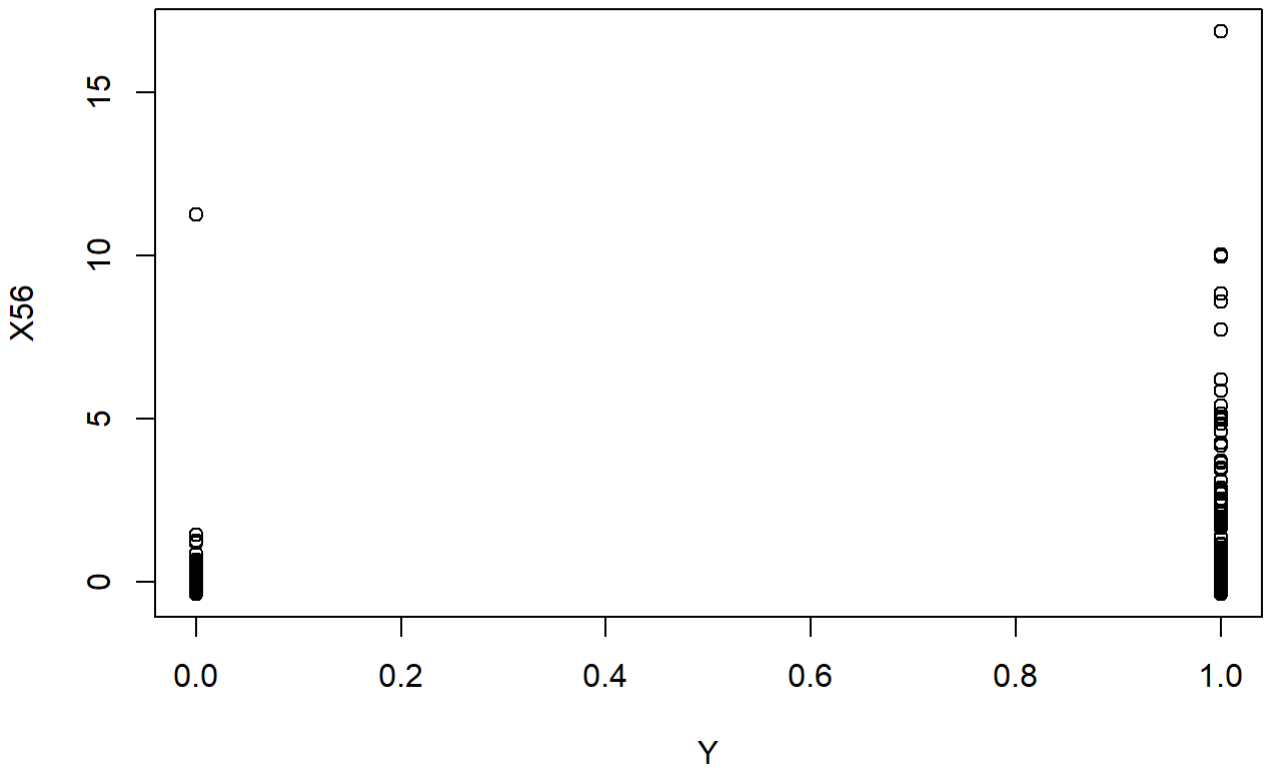

X51
0
10
20
30
0.0
0.2
0.4
0.6
0.8
1.0
Y

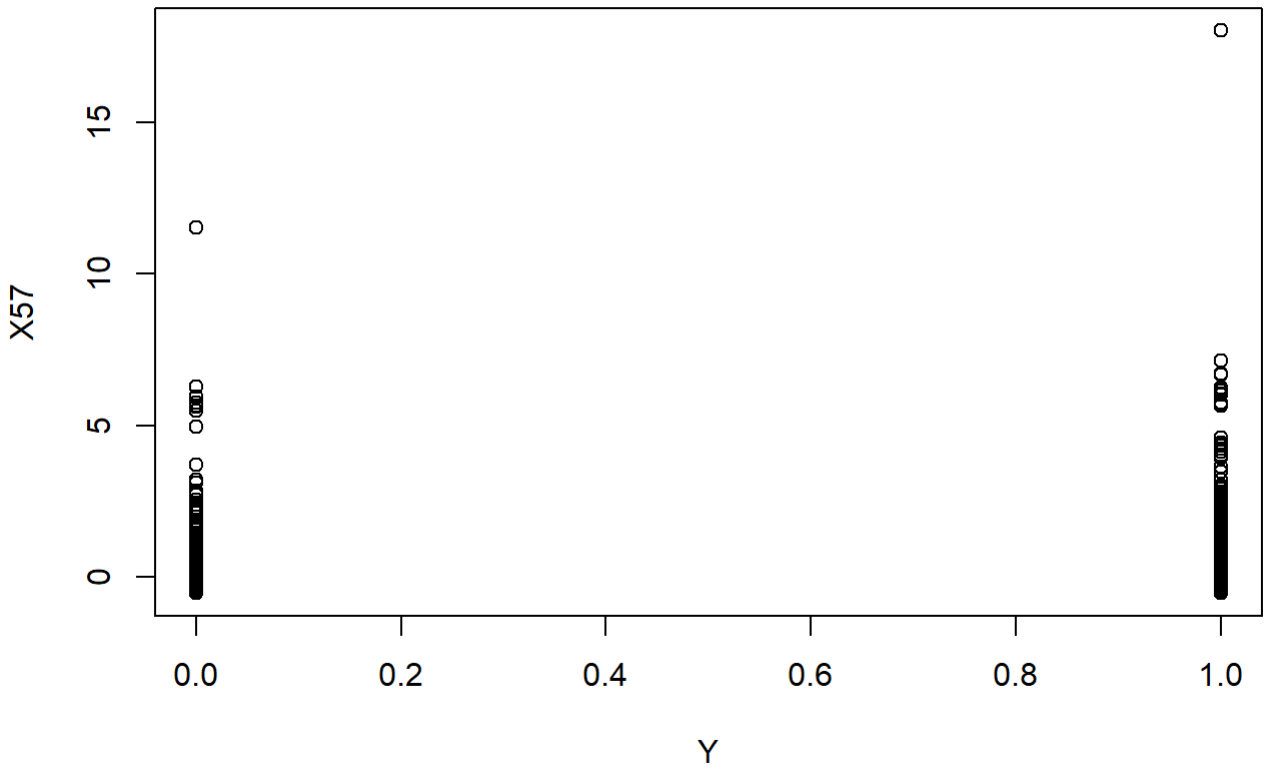

X52
0
10
20
30
0.0
0.2
0.4
0.6
0.8
1.0
Y


X53
0
5
10
15
20
25
0.0
0.2
0.4
0.6
0.8
1.0
Y

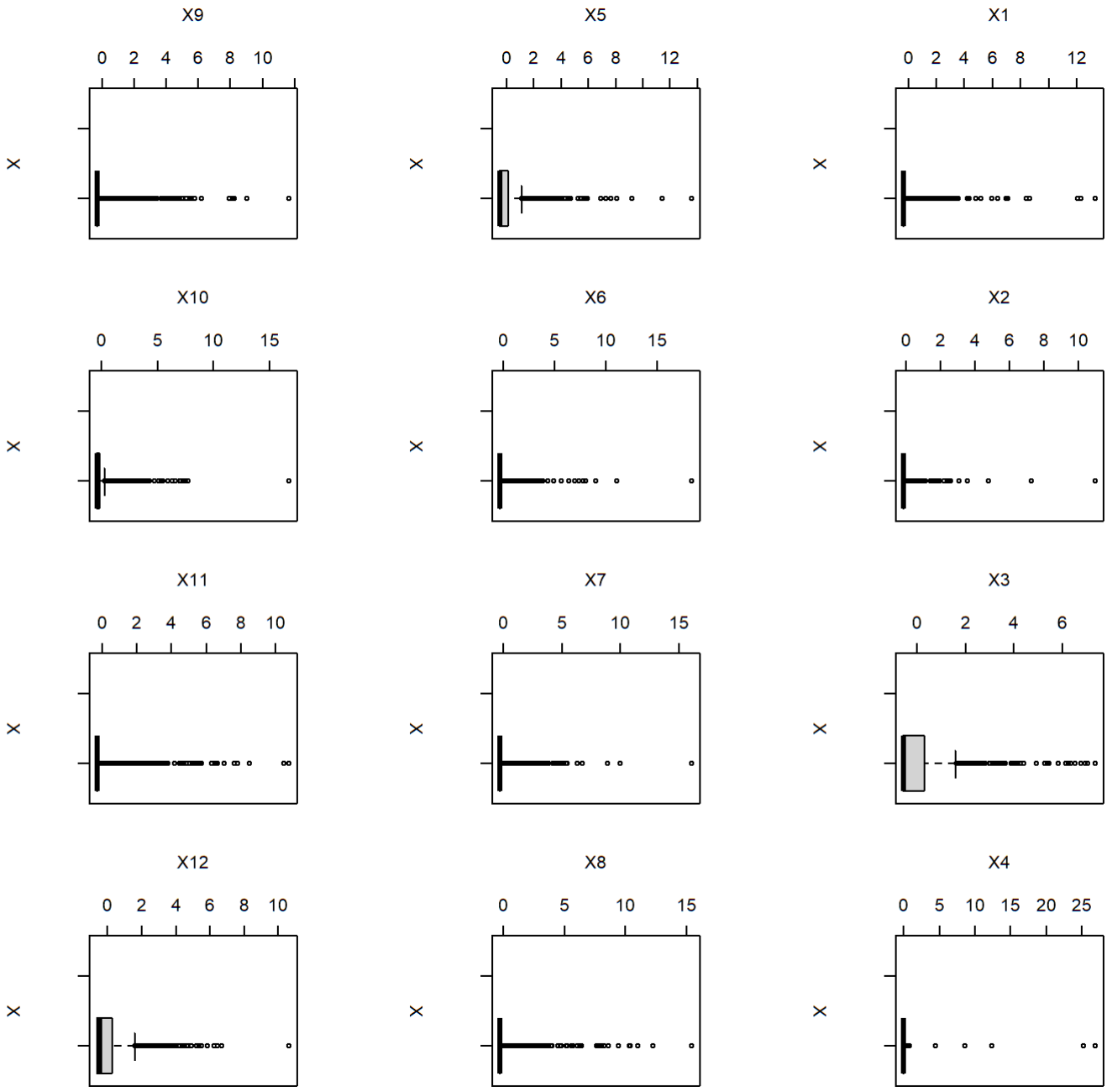

X54
0
5
10
15
20
25
0.0
0.2
0.4
0.6
0.8
1.0
Y

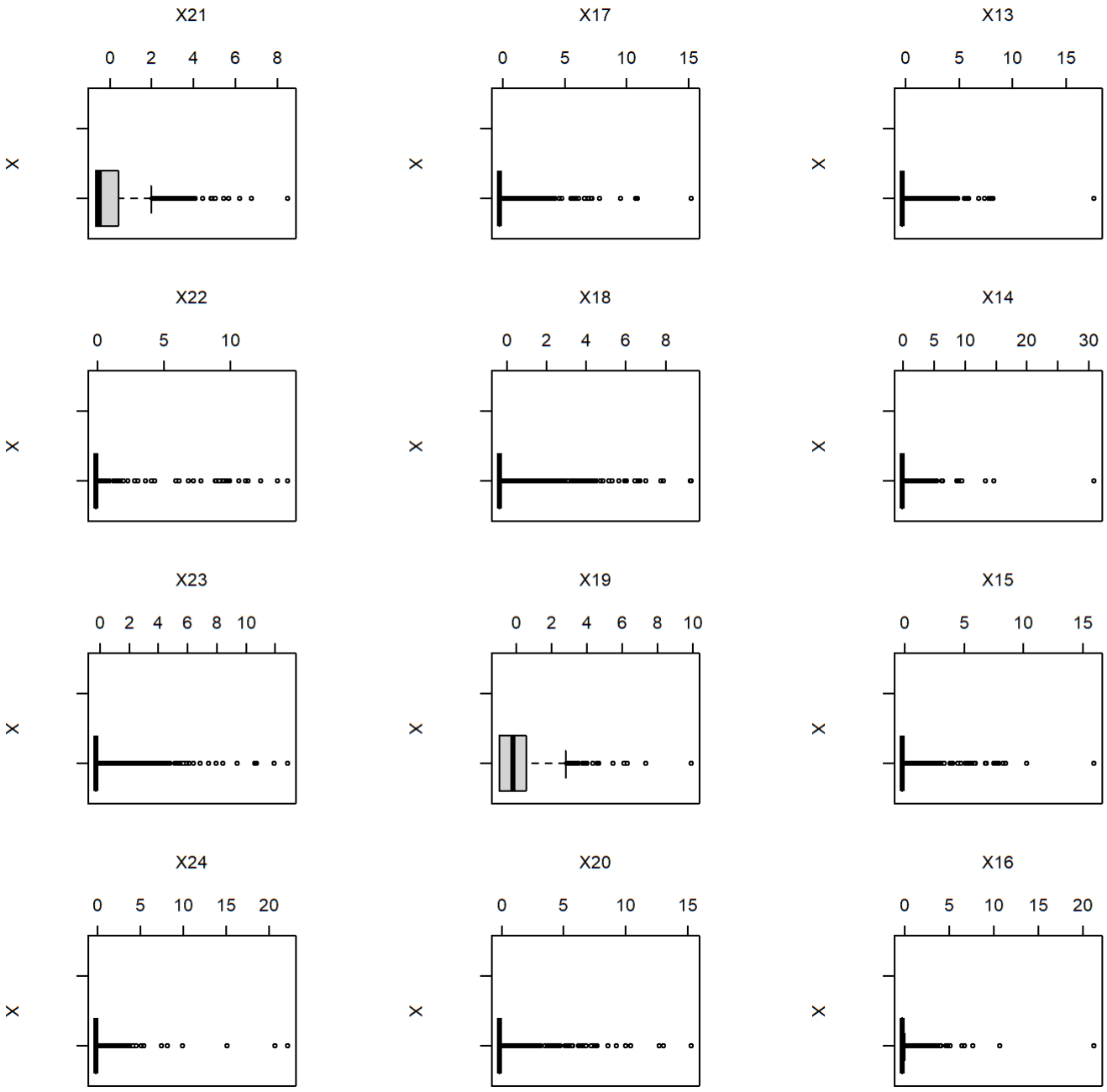

X55
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0
Y

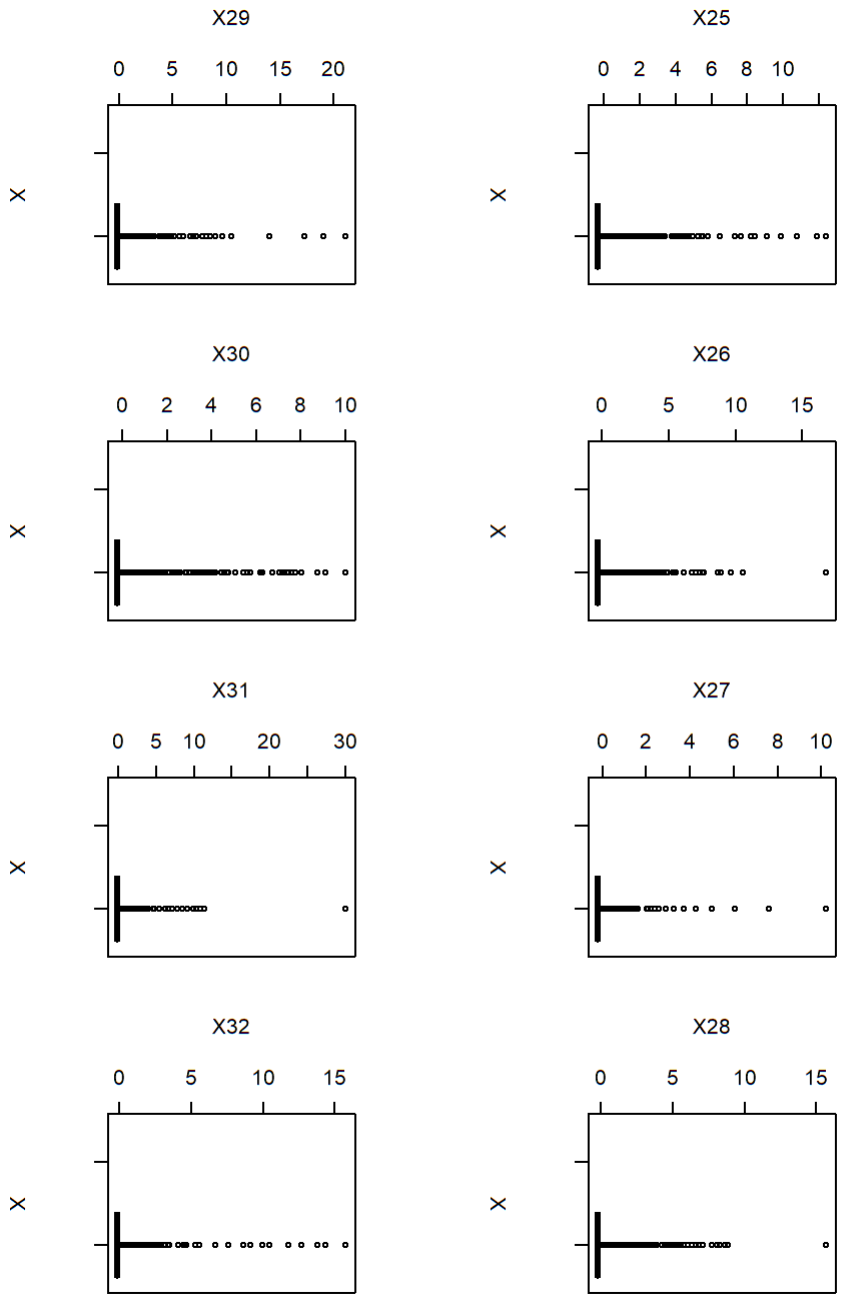

X56
0
5
10
15
0.0
0.2
0.4
0.6
0.8
1.0
Y

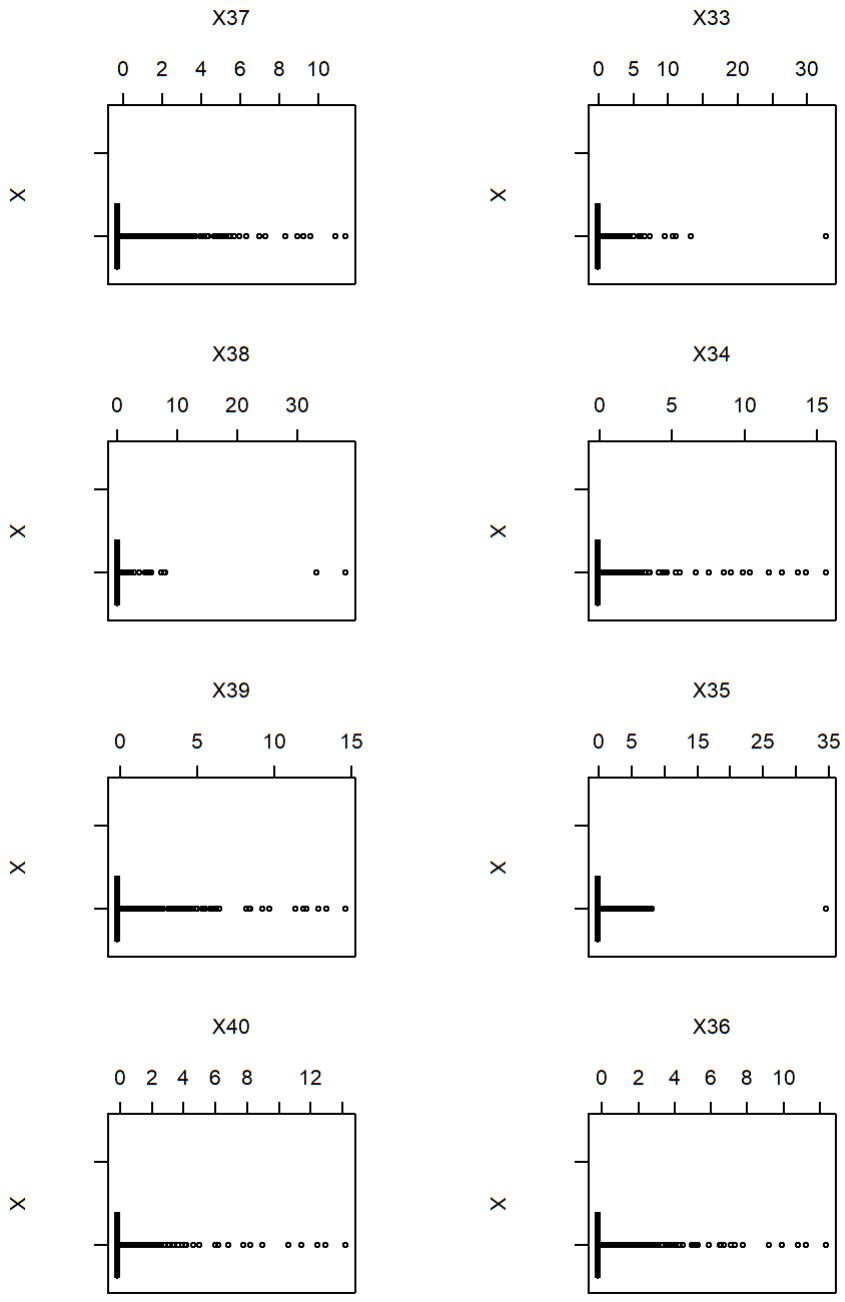


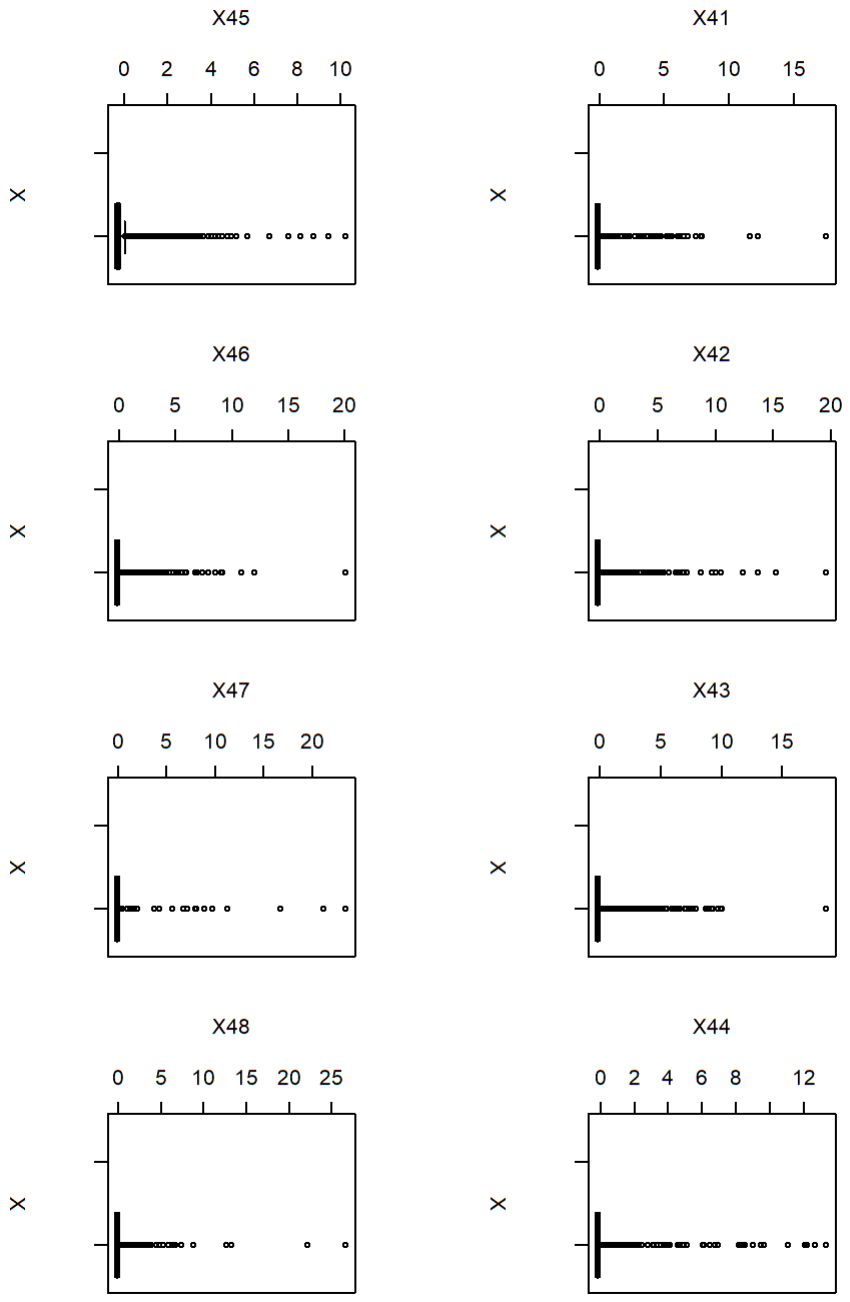

```
par(mfrow = c(1,1))

#(3)boxplot of X ~ X_i

par(mfrow = c(2,4))
for (i in 1:57) {
  boxplot(train.stand$X, train.stand[, i], xlab = "X", ylab = paste0("X",i))
}
```

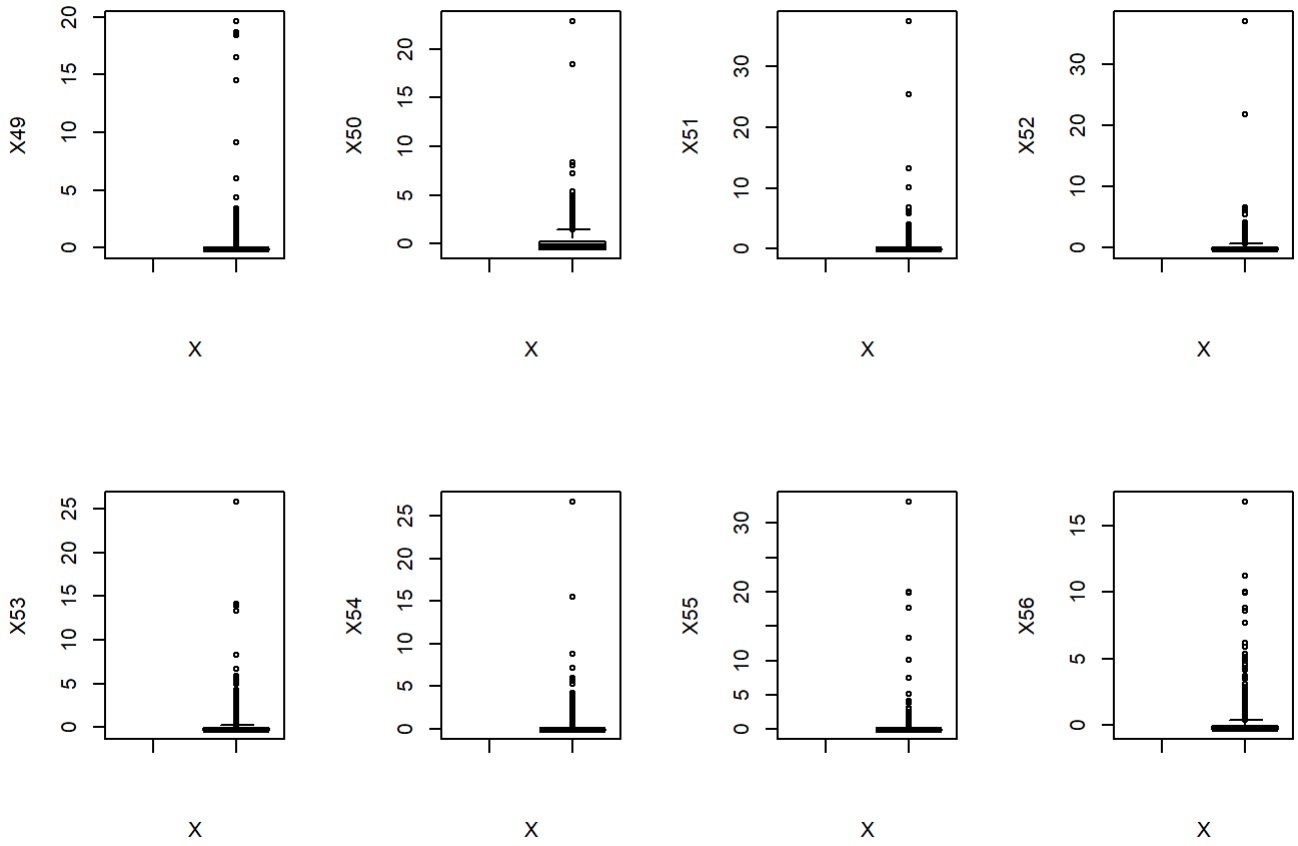

X9
0 2 4 6 8 10
X
X5
0 2 4 6 8 12
X
X1
0 2 4 6 8 12
X
X10
0 5 10 15
X
X6
0 5 10 15
X
X2
0 2 4 6 8 10
X
X11
0 2 4 6 8 10
X
X7
0 5 10 15
X
X3
0 2 4 6
X
X12
0 2 4 6 8 10
X
X8
0 5 10 15
X
X4
0 5 10 15 20 25
X


X21
0 2 4 6 8
X
X17
0 5 10 15
X
X13
0 5 10 15
X
X22
0 5 10
X
X18
0 2 4 6 8
X
X14
0 5 10 20 30
X
X23
0 2 4 6 8 10
X
X19
0 2 4 6 8 10
X
X15
0 5 10 15
X
X24
0 5 10 15 20
X
X20
0 5 10 15
X
X16
0 5 10 15 20
X

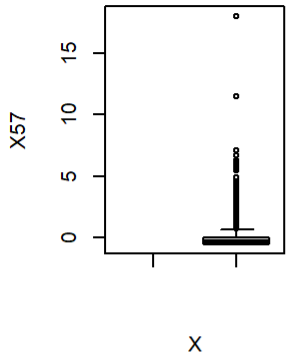

X29
0 5 10 15 20
X
X25
0 2 4 6 8 10
X
X30
0 2 4 6 8 10
X
X26
0 5 10 15
X
X31
0 5 10 20 30
X
X27
0 2 4 6 8 10
X
X32
0 5 10 15
X
X28
0 5 10 15
X


X37
0 2 4 6 8 10
X
X33
0 5 10 20 30
X
X38
0 10 20 30
X
X34
0 5 10 15
X
X39
0 5 10 15
X
X35
0 5 15 25 35
X
X40
0 2 4 6 8 12
X
X36
0 2 4 6 8 10
X


X45
0 2 4 6 8 10
X
X41
0 5 10 15
X
X46
0 5 10 15 20
X
X42
0 5 10 15 20
X
X47
0 5 10 15 20
X
X43
0 5 10 15
X
X48
0 5 10 15 20 25
X
X44
0 2 4 6 8 12
X



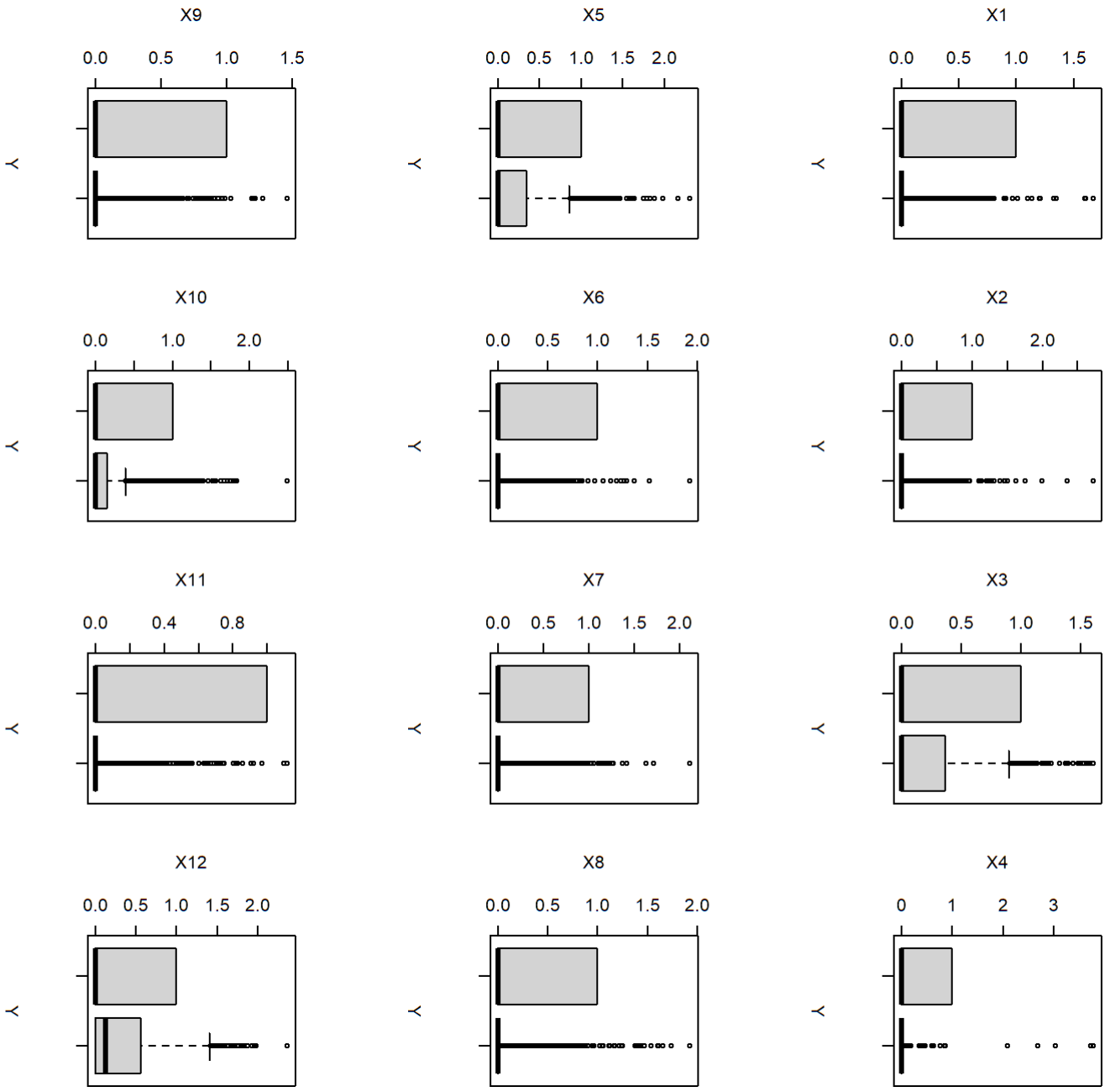

```
par(mfrow = c(1,1))
```

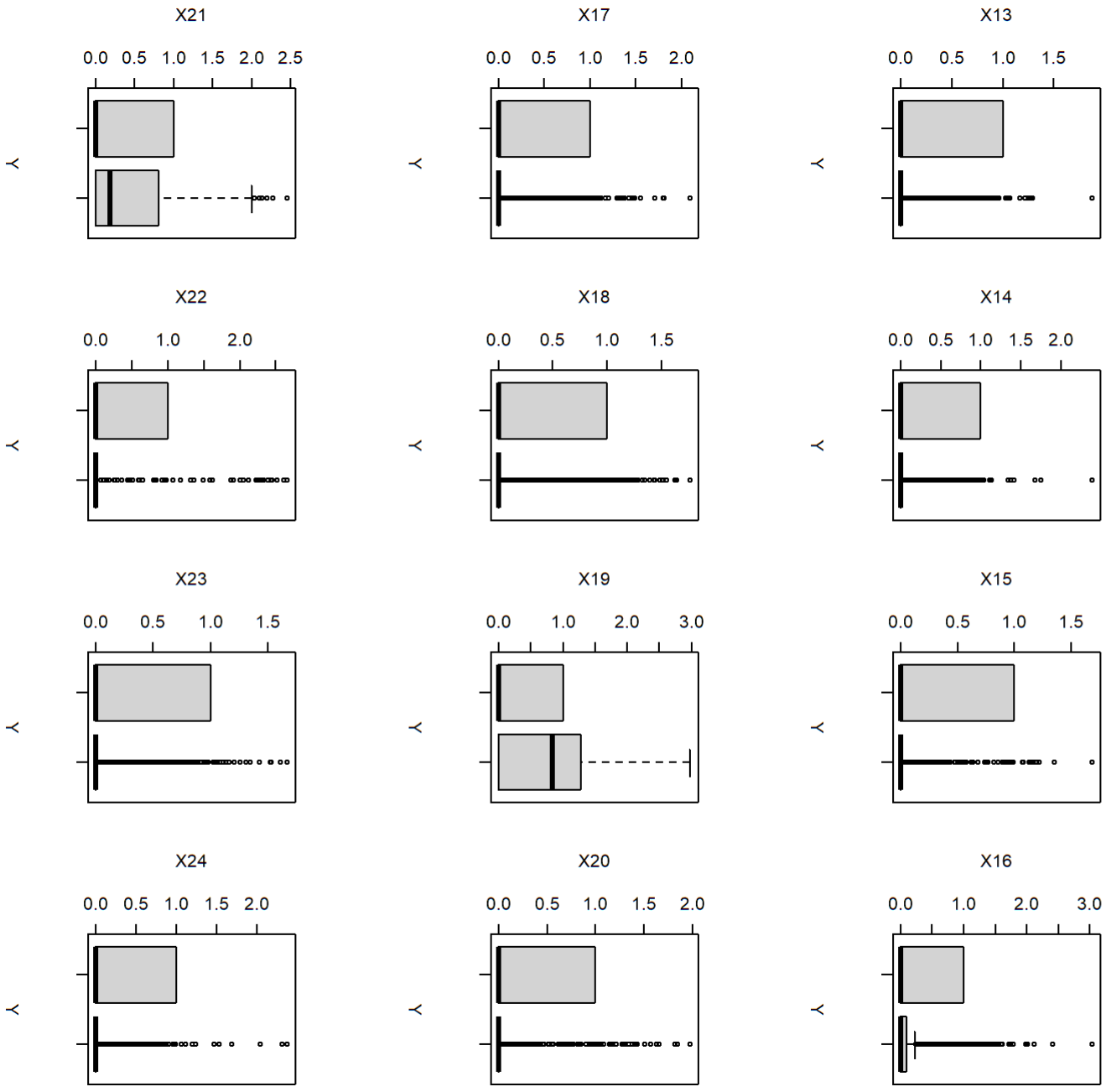


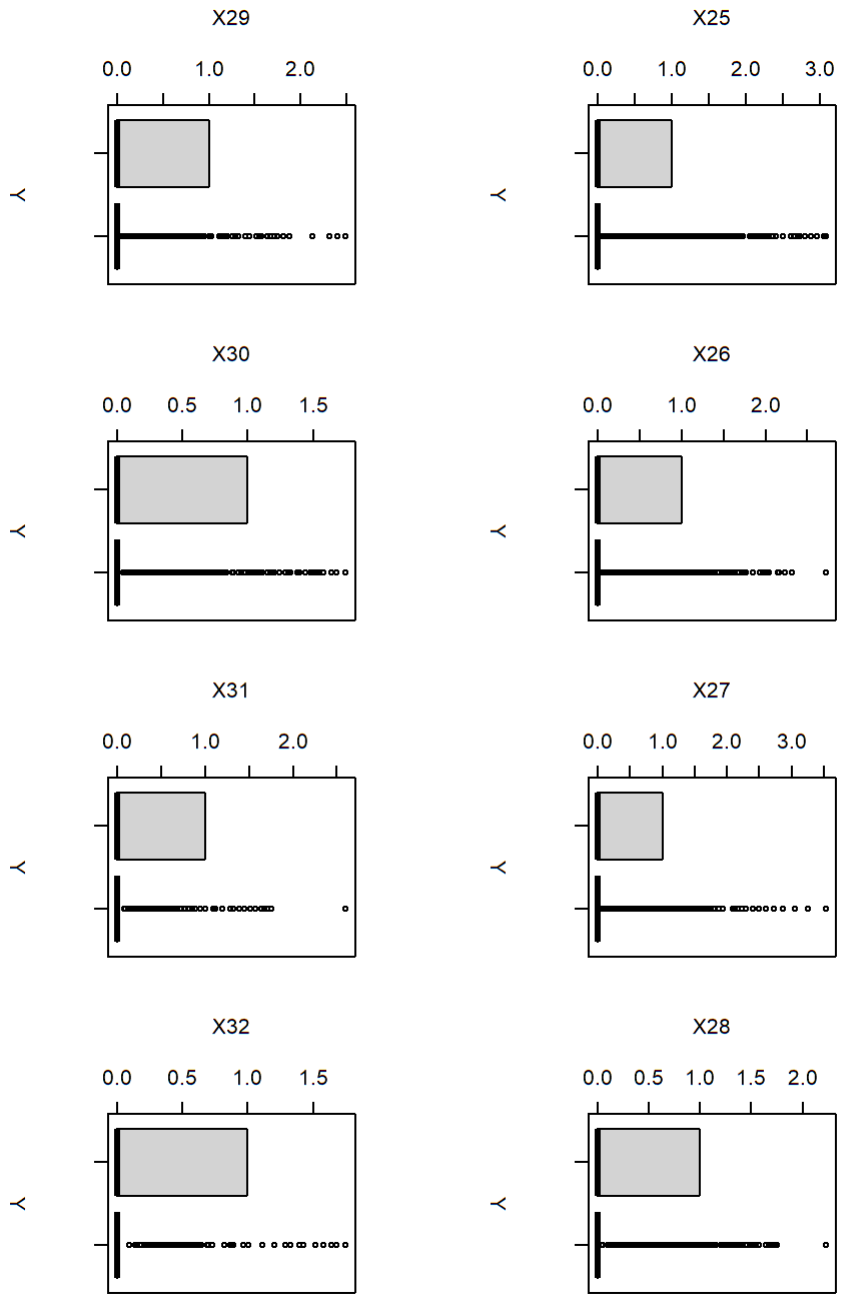
Method 2: Log-transformed

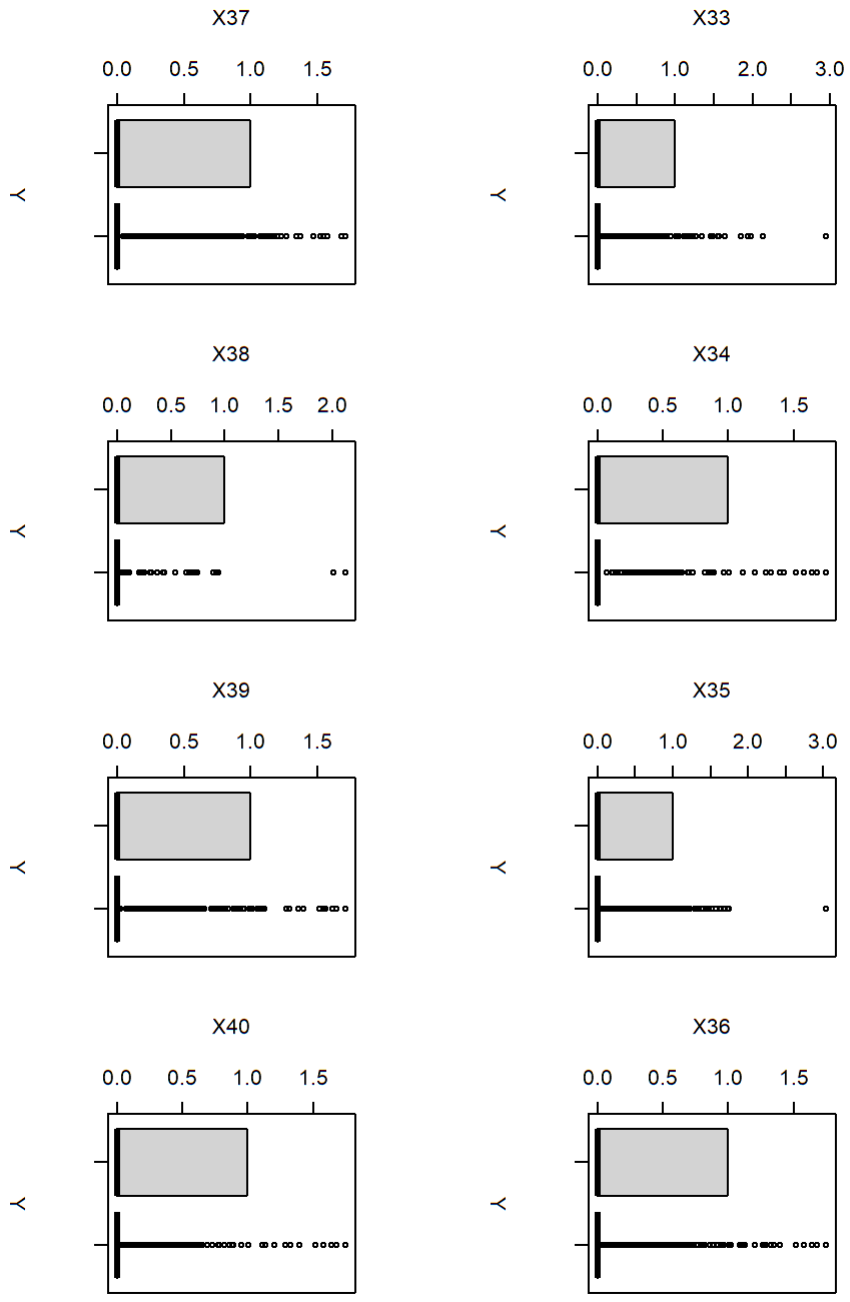
```
data.train.log <- spam_train
data.train.log[, -58] <- as.data.frame(log(spam_train[, -58]+1))
data.test.log <- spam_test
data.test.log[, -58] <- as.data.frame(log(spam_test[, -58]+1))
```

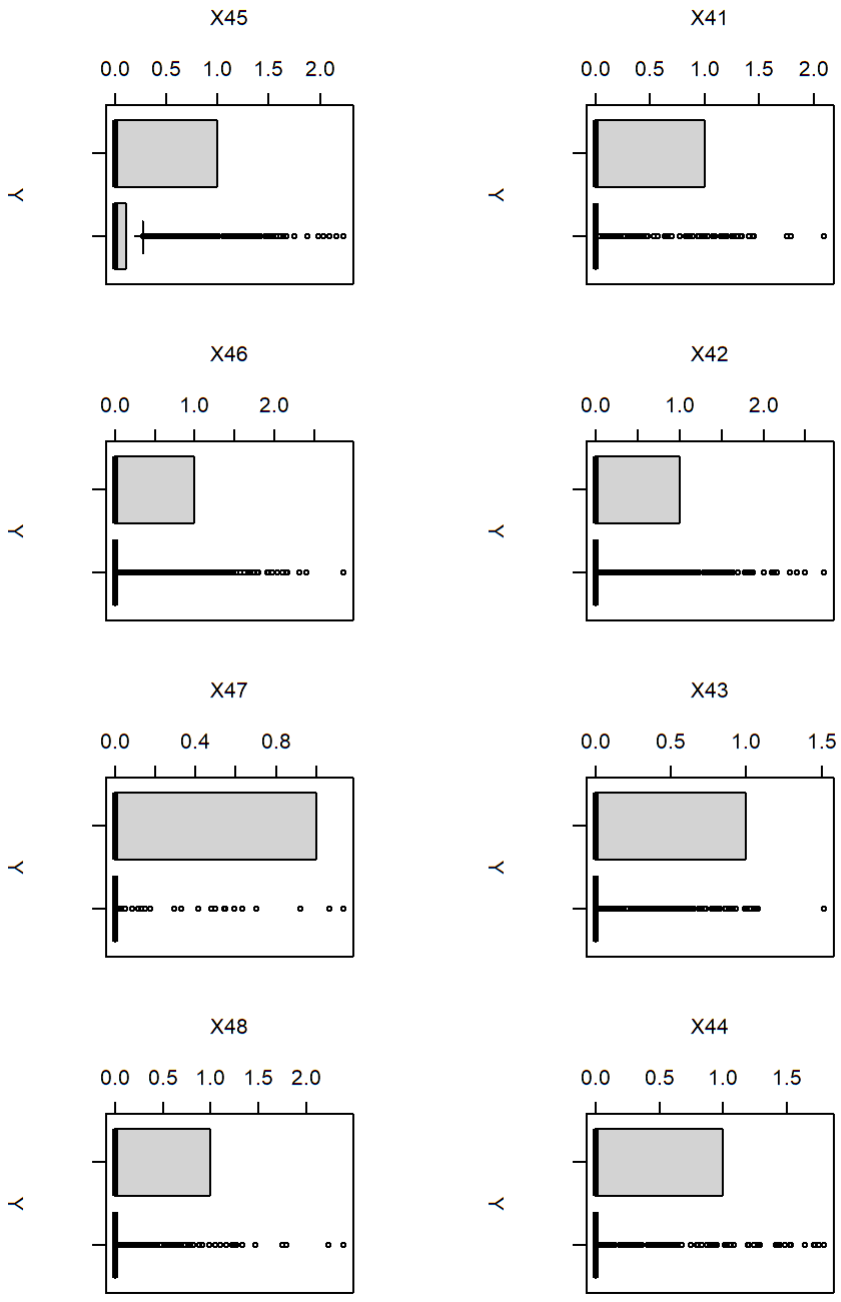
```
par(mfrow = c(2,4))
for (i in 1:57) {
  boxplot(data.train.log$Y, data.train.log[, i], xlab = "Y", ylab = paste0("X",i))
}
```

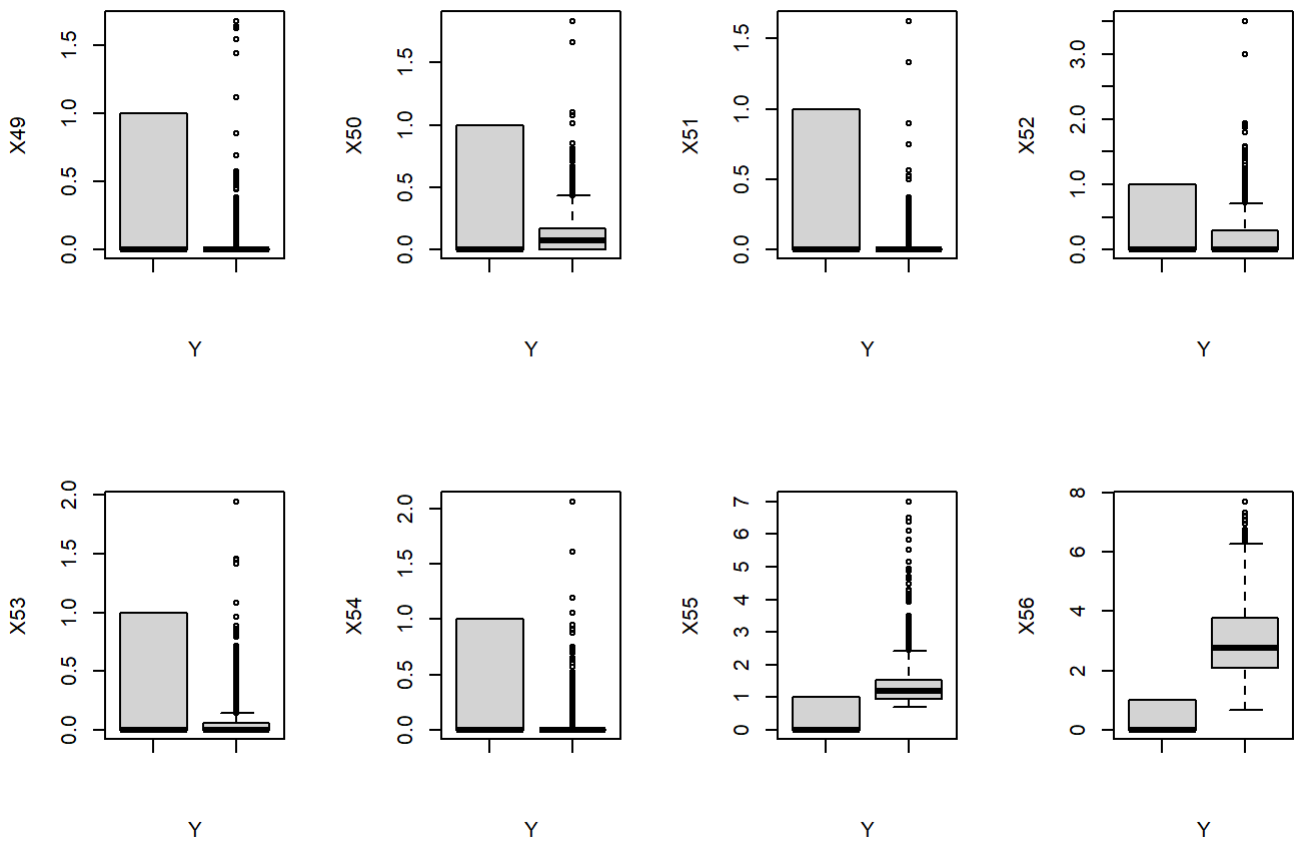

X9
0.0 0.5 1.0 1.5
Y
X5
0.0 0.5 1.0 1.5 2.0
Y
X1
0.0 0.5 1.0 1.5
Y
X10
0.0 1.0 2.0
Y
X6
0.0 0.5 1.0 1.5 2.0
Y
X2
0.0 1.0 2.0
Y
X11
0.0 0.4 0.8
Y
X7
0.0 0.5 1.0 1.5 2.0
Y
X3
0.0 0.5 1.0 1.5
Y
X12
0.0 0.5 1.0 1.5 2.0
Y
X8
0.0 0.5 1.0 1.5 2.0
Y
X4
0 1 2 3
Y


X21
0.0 0.5 1.0 1.5 2.0 2.5
Y
X17
0.0 0.5 1.0 1.5 2.0
Y
X13
0.0 0.5 1.0 1.5
Y
X22
0.0 1.0 2.0
Y
X18
0.0 0.5 1.0 1.5
Y
X14
0.0 0.5 1.0 1.5 2.0
Y
X23
0.0 0.5 1.0 1.5
Y
X19
0.0 1.0 2.0 3.0
Y
X15
0.0 0.5 1.0 1.5
Y
X24
0.0 0.5 1.0 1.5 2.0
Y
X20
0.0 0.5 1.0 1.5 2.0
Y
X16
0.0 1.0 2.0 3.0
Y

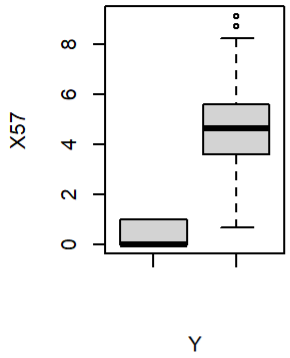

X29
0.0
1.0
2.0
Y
X25
0.0
1.0
2.0
3.0
Y
X30
0.0
0.5
1.0
1.5
Y
X26
0.0
1.0
2.0
Y
X31
0.0
1.0
2.0
Y
X27
0.0
1.0
2.0
3.0
Y
X32
0.0
0.5
1.0
1.5
Y
X28
0.0
0.5
1.0
1.5
2.0
Y


X37
0.0 0.5 1.0 1.5
Y
X33
0.0 1.0 2.0 3.0
Y
X38
0.0 0.5 1.0 1.5 2.0
Y
X34
0.0 0.5 1.0 1.5
Y
X39
0.0 0.5 1.0 1.5
Y
X35
0.0 1.0 2.0 3.0
Y
X40
0.0 0.5 1.0 1.5
Y
X36
0.0 0.5 1.0 1.5
Y


X45
0.0 0.5 1.0 1.5 2.0
Y
X41
0.0 0.5 1.0 1.5 2.0
Y
X46
0.0 1.0 2.0
Y
X42
0.0 1.0 2.0
Y
X47
0.0 0.4 0.8
Y
X43
0.0 0.5 1.0 1.5
Y
X48
0.0 0.5 1.0 1.5 2.0
Y
X44
0.0 0.5 1.0 1.5
Y



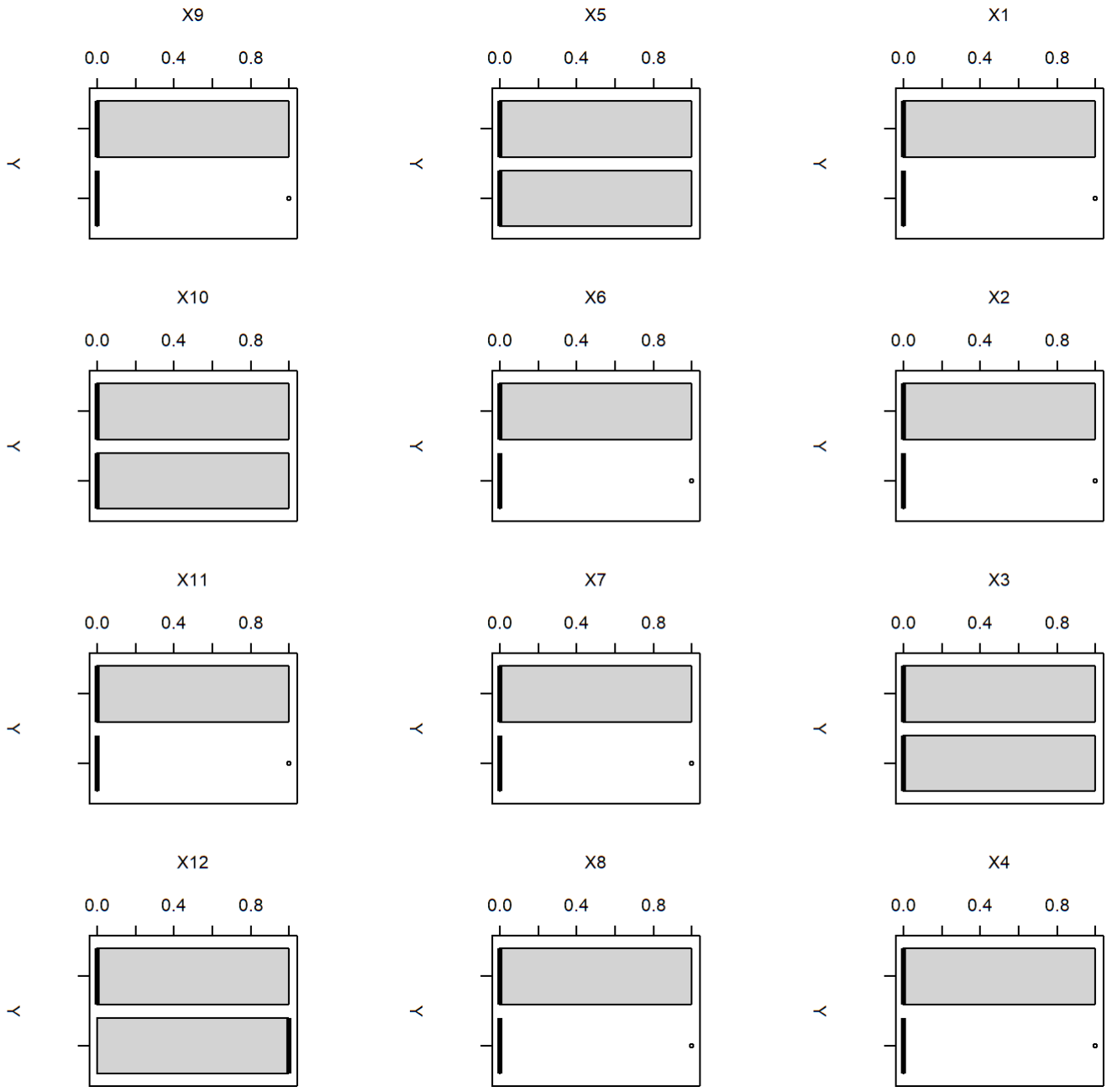

```
par(mfrow = c(1,1))
```

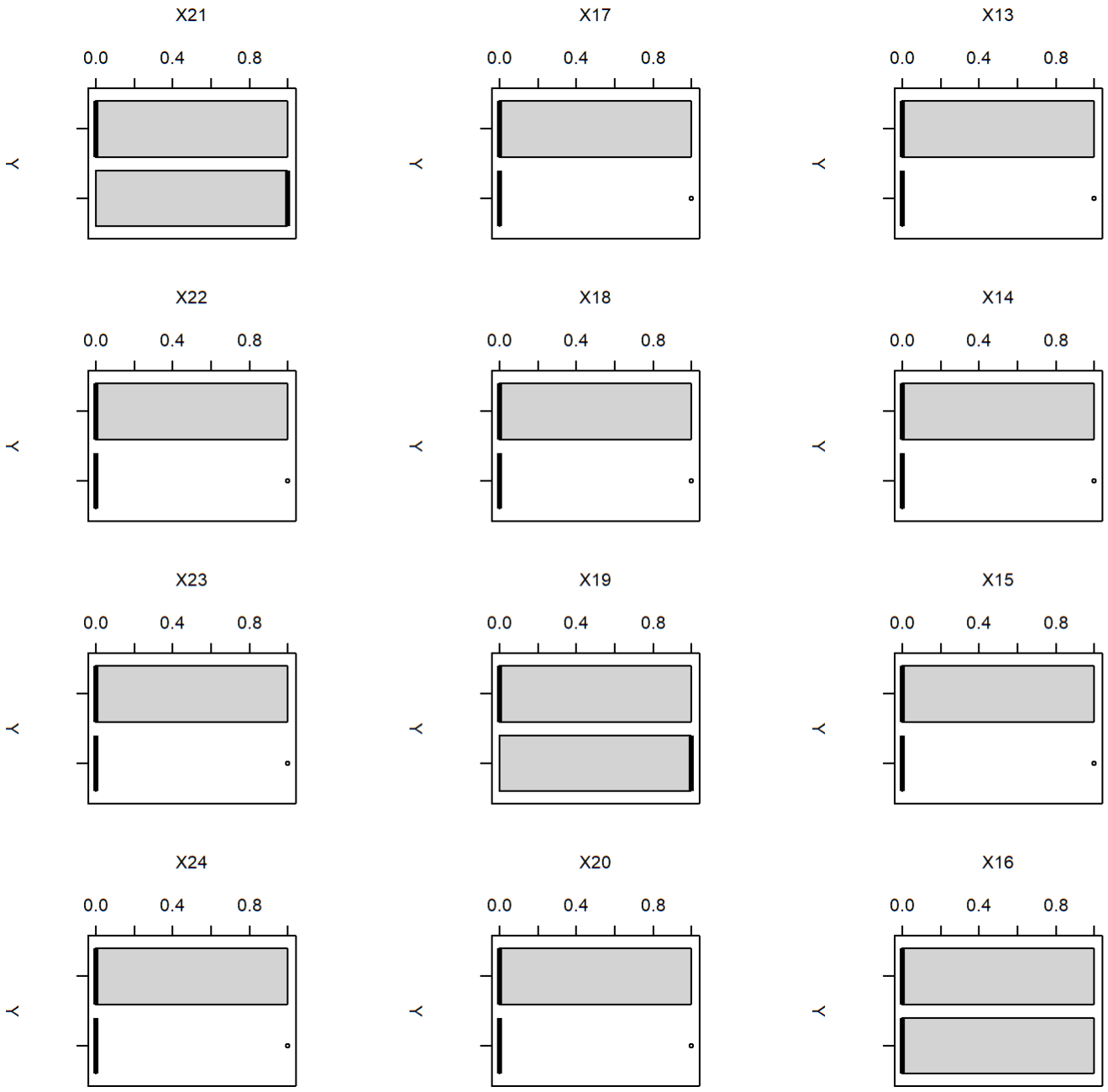


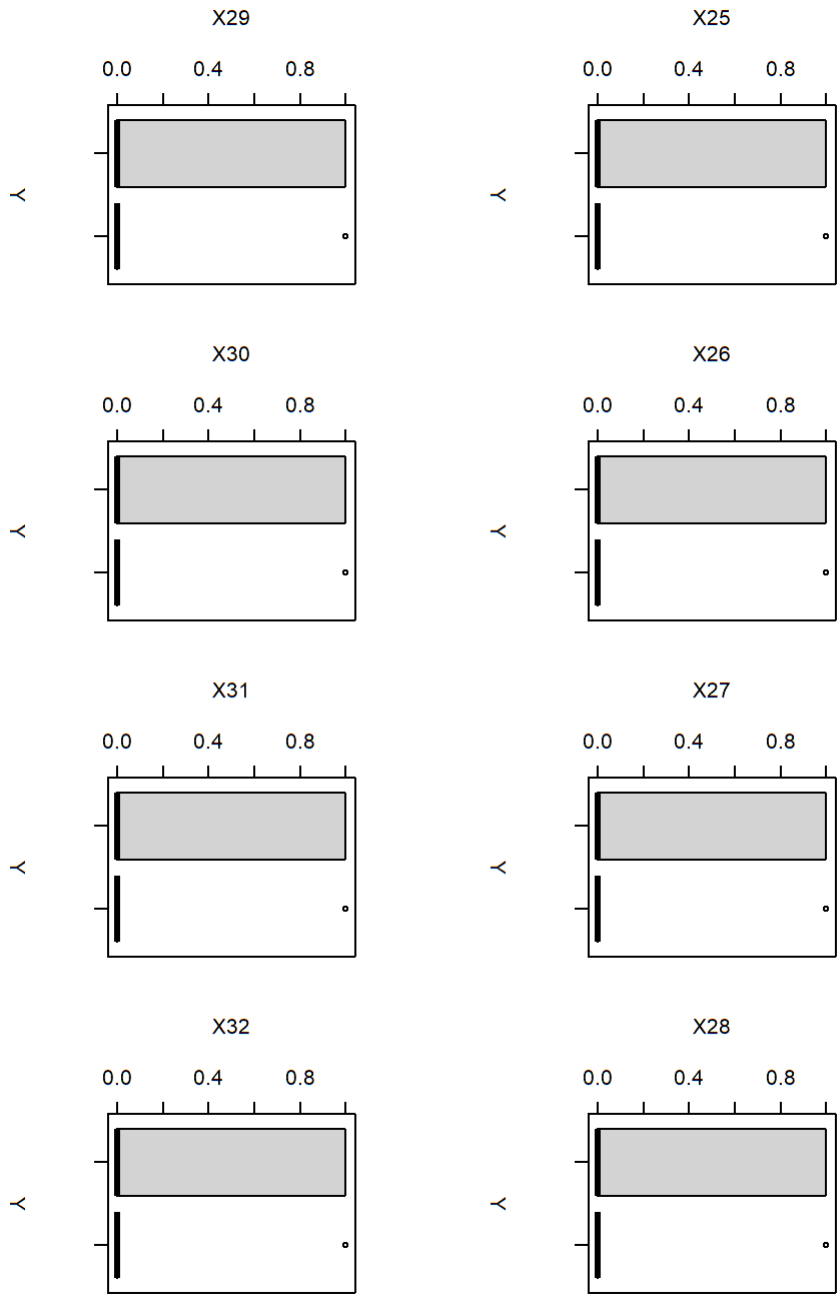
Method 3: Discretize each feature using I(Xij > 0)

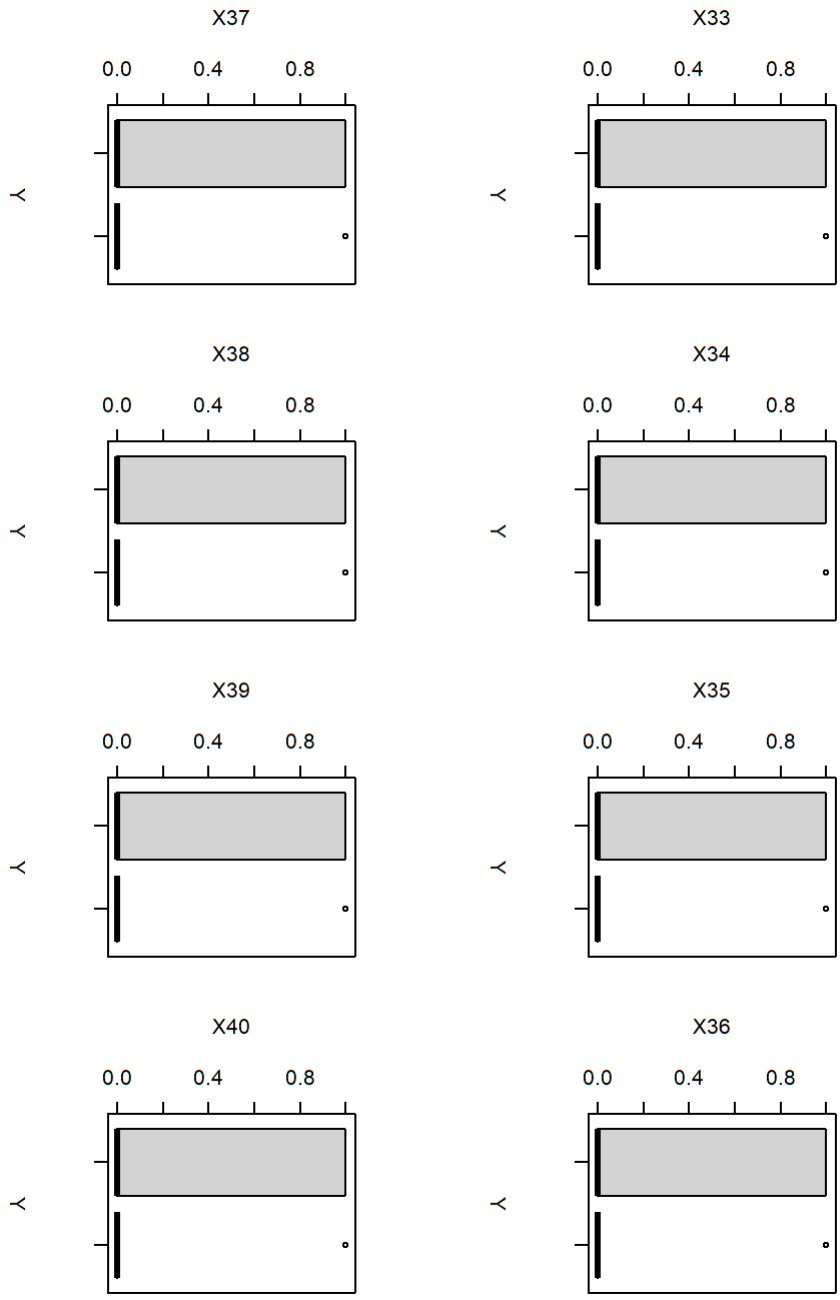
```
# discretized data
data.train.dis <- as.data.frame(ifelse(spam_train > 0, 1, 0))
data.test.dis <- as.data.frame(ifelse(spam_test > 0, 1, 0))
```

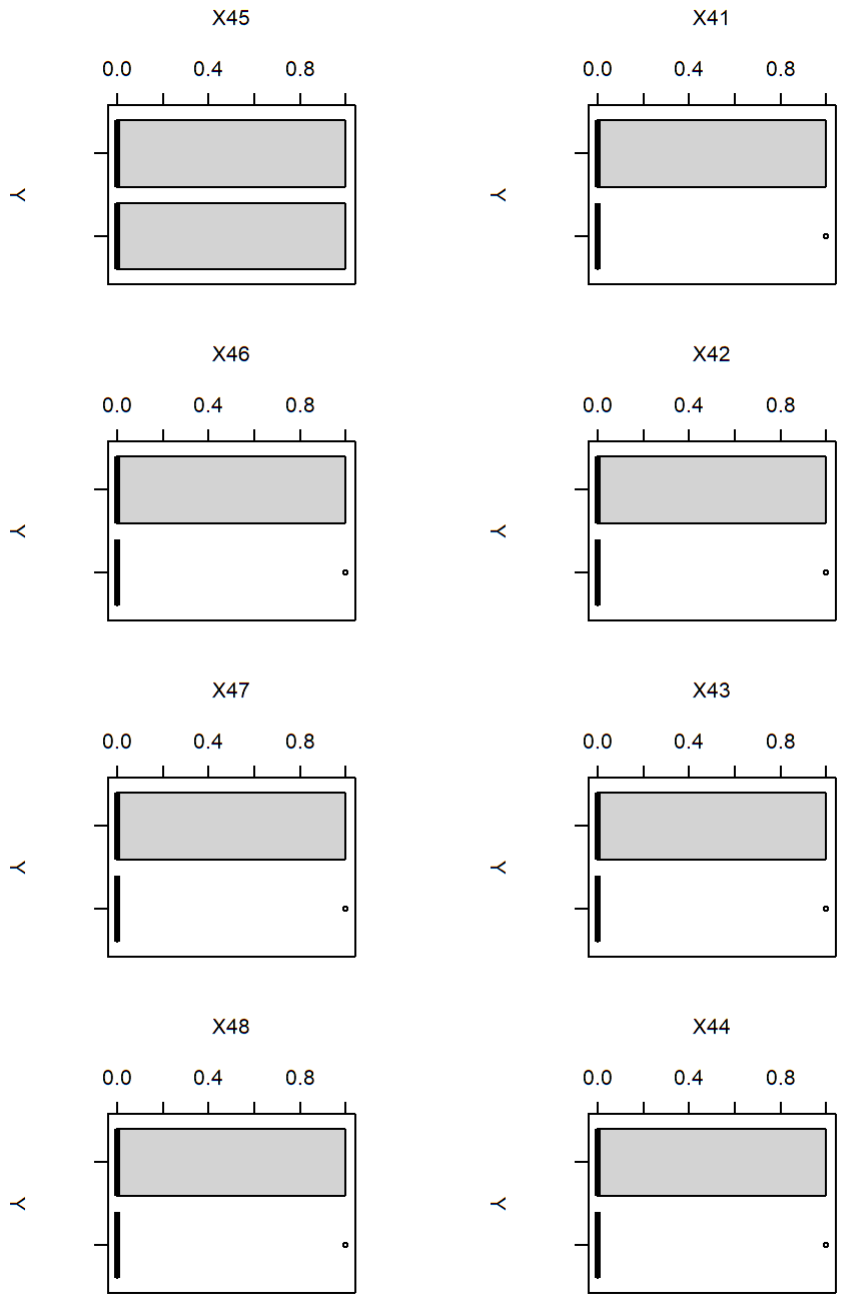
```r
#(1)boxplot of Y ~ X_i

par(mfrow = c(2,4))
for (i in 1:57) {
  boxplot(data.train.dis$Y, data.train.dis[, i], xlab = "Y", ylab = paste0("X",i))
}
```

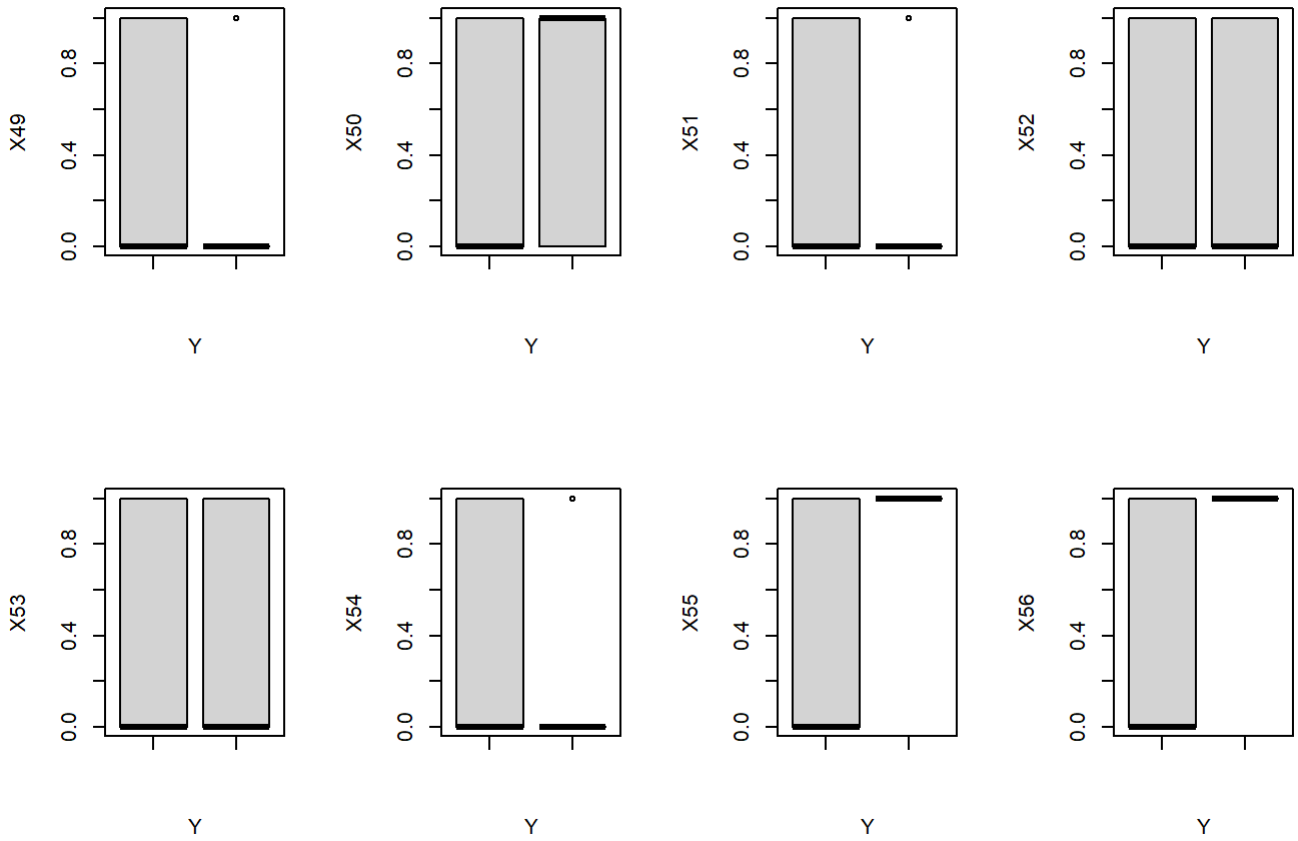

X9
X5
X1
X10
X6
X2
X11
X7
X3
X12
X8
X4
0.0
0.4
0.8
Y


X21
X17
X13
X22
X18
X14
X23
X19
X15
X24
X20
X16
0.0
0.4
0.8
Y

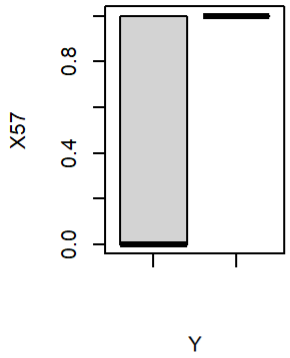

X29
X25
X30
X26
X31
X27
X32
X28
0.0
0.4
0.8
Y


X37
X33
X38
X34
X39
X35
X40
X36
0.0
0.4
0.8
Y


X45
X41
X46
X42
X47
X43
X48
X44
0.0
0.4
0.8
Y



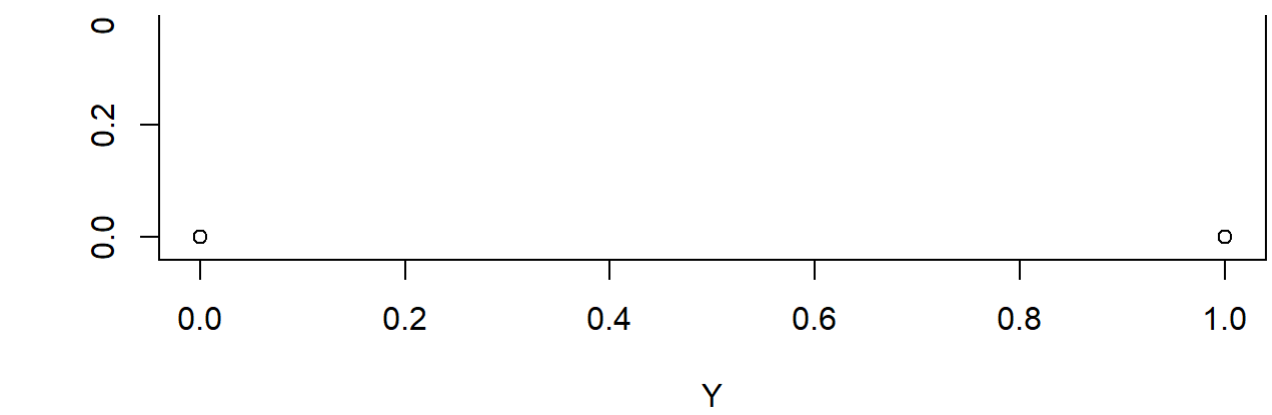

```
par(mfrow = c(1,1))
```

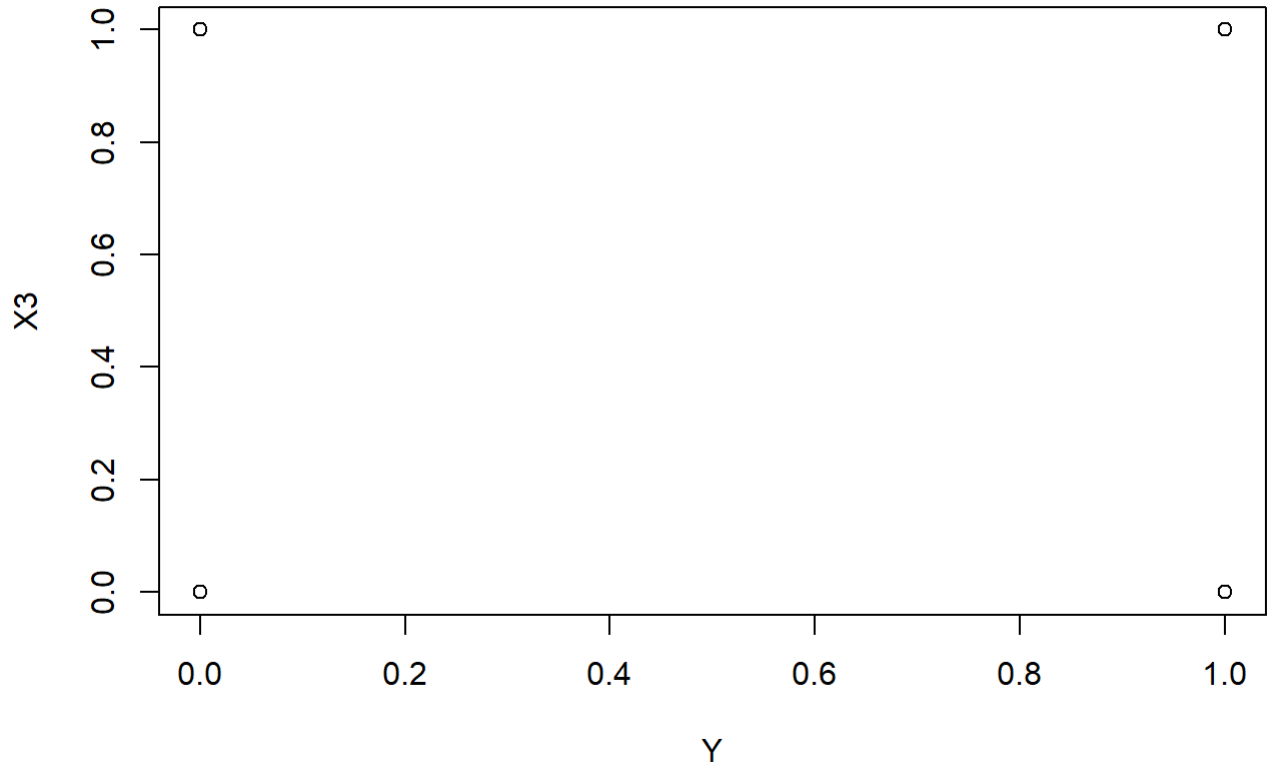


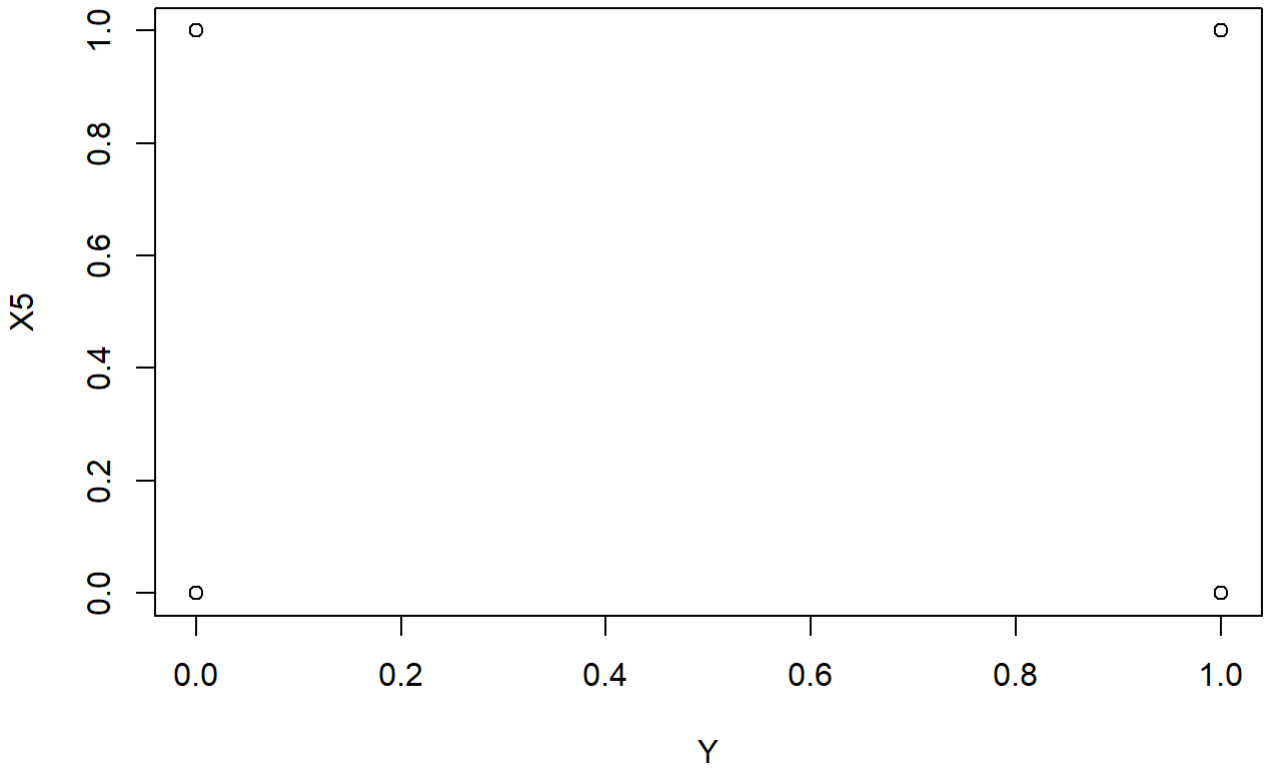

```
#(2)scatterplot of Y ~ X_i

for (i in 1:57){
  plot(data.train.dis$Y, data.train.dis[, i], xlab = "Y", ylab = paste0("X",i))
}
```

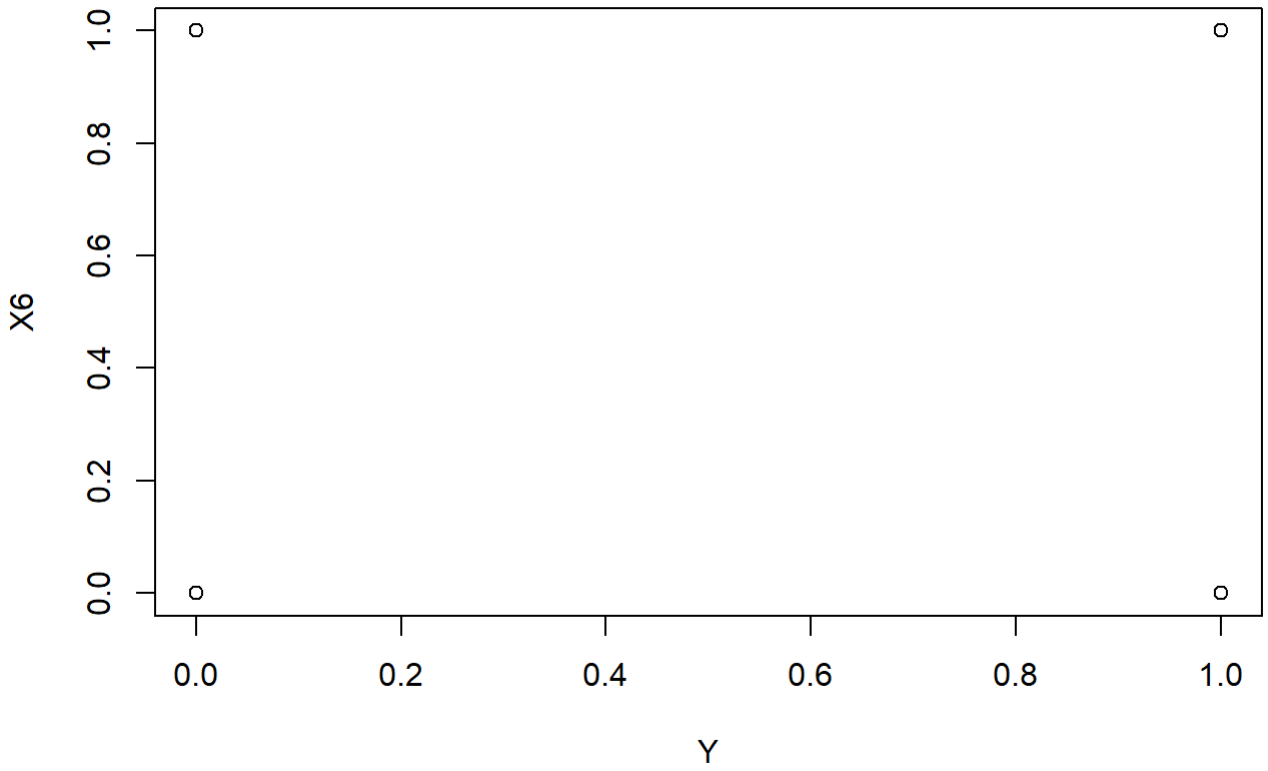
X1
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

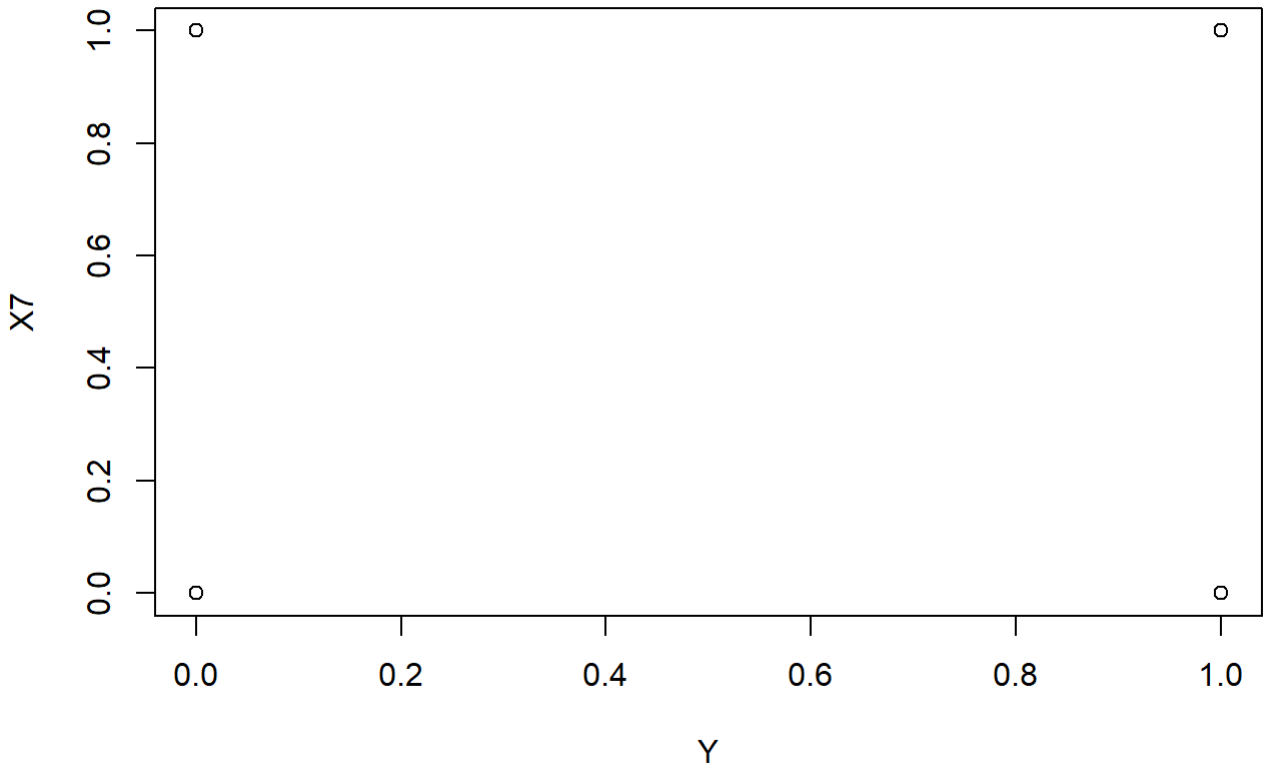
X2
.4
0.6
0.8
1.0

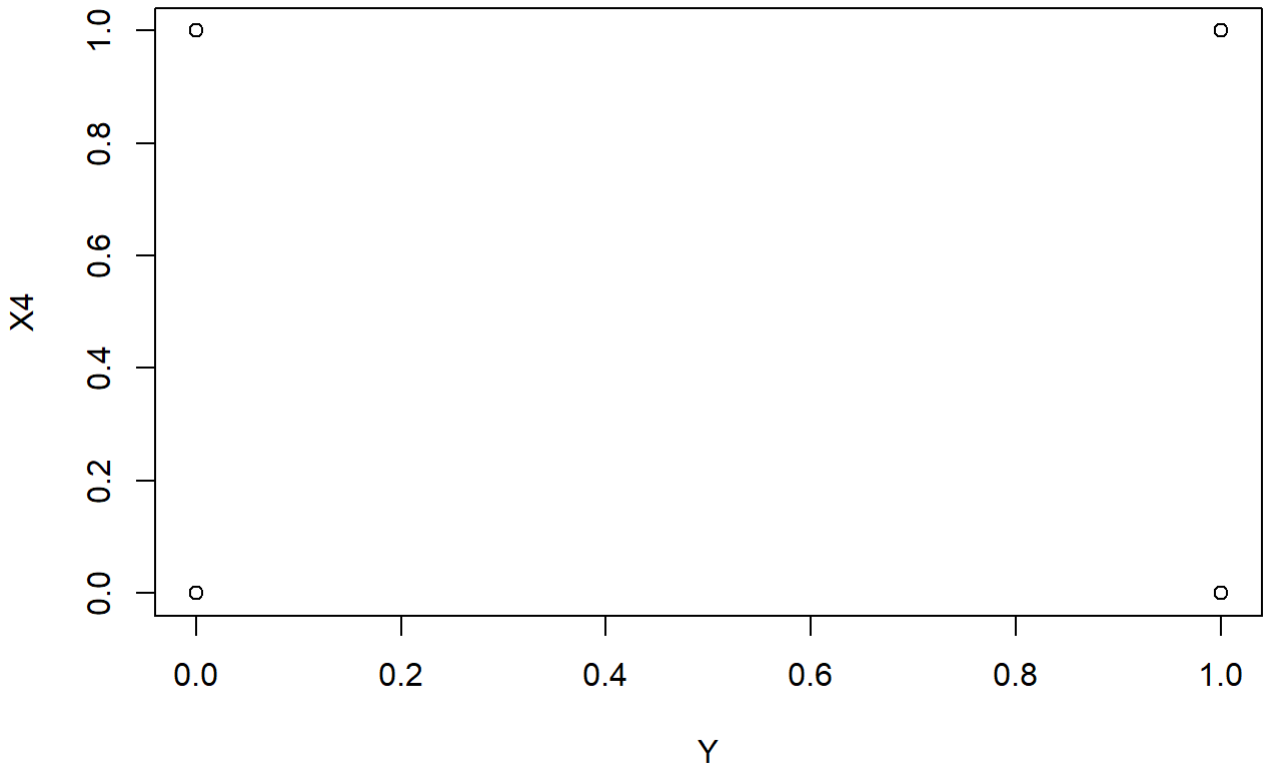

0.0
0.2
0.4
0.6
0.8
1.0
Y

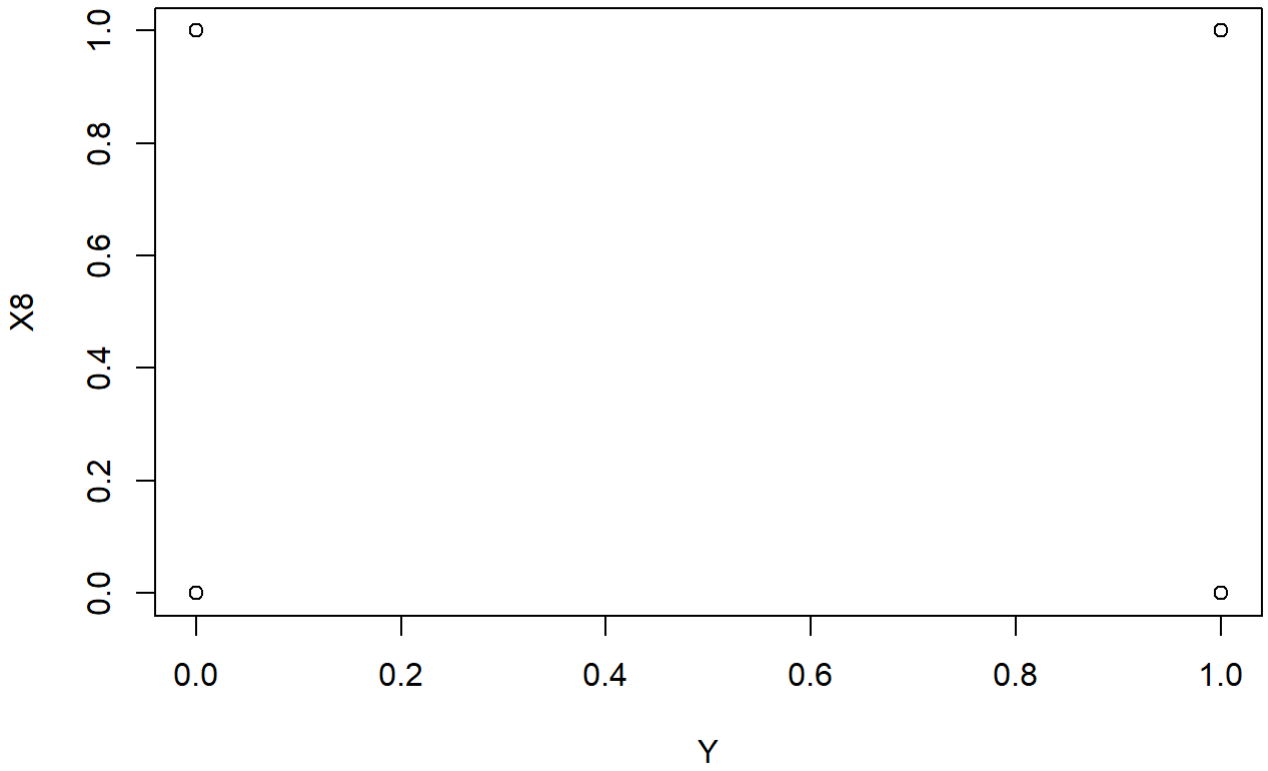

X3
0.0
0.2
0.4
0.6
0.8
1.0
Y

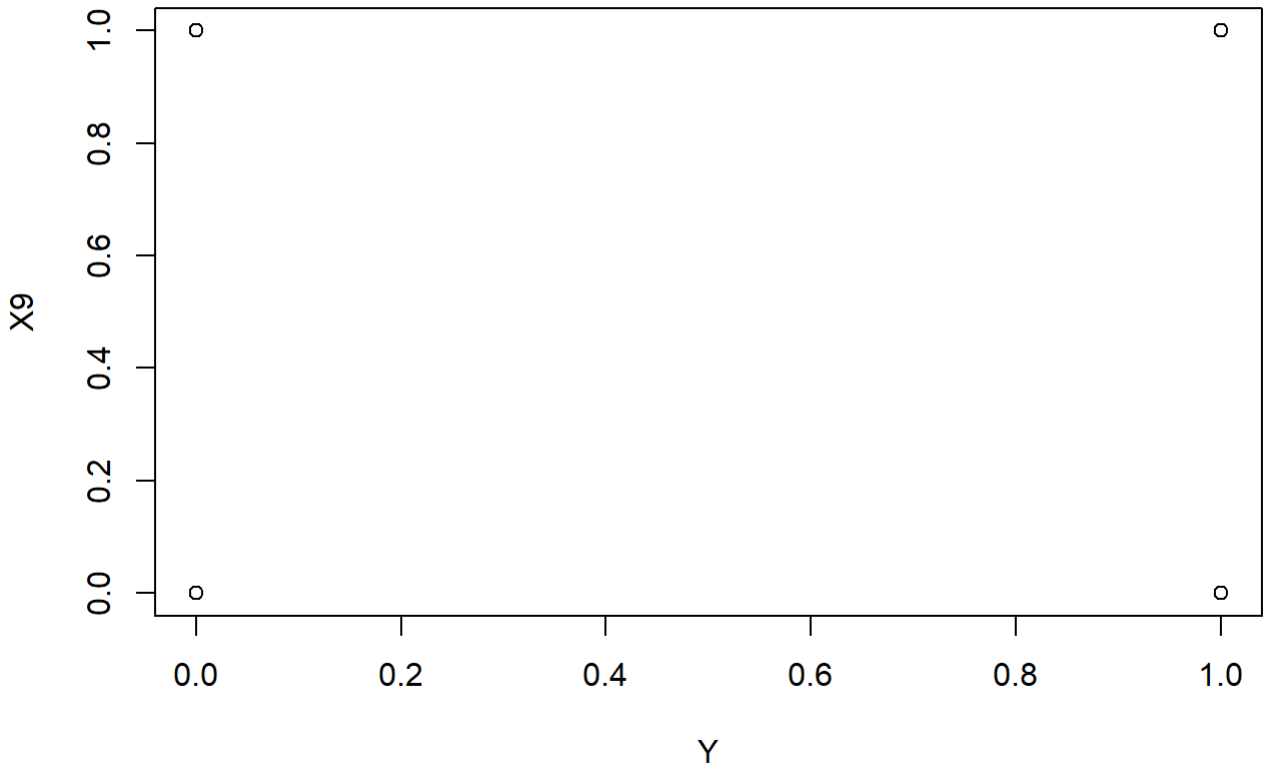

X5
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

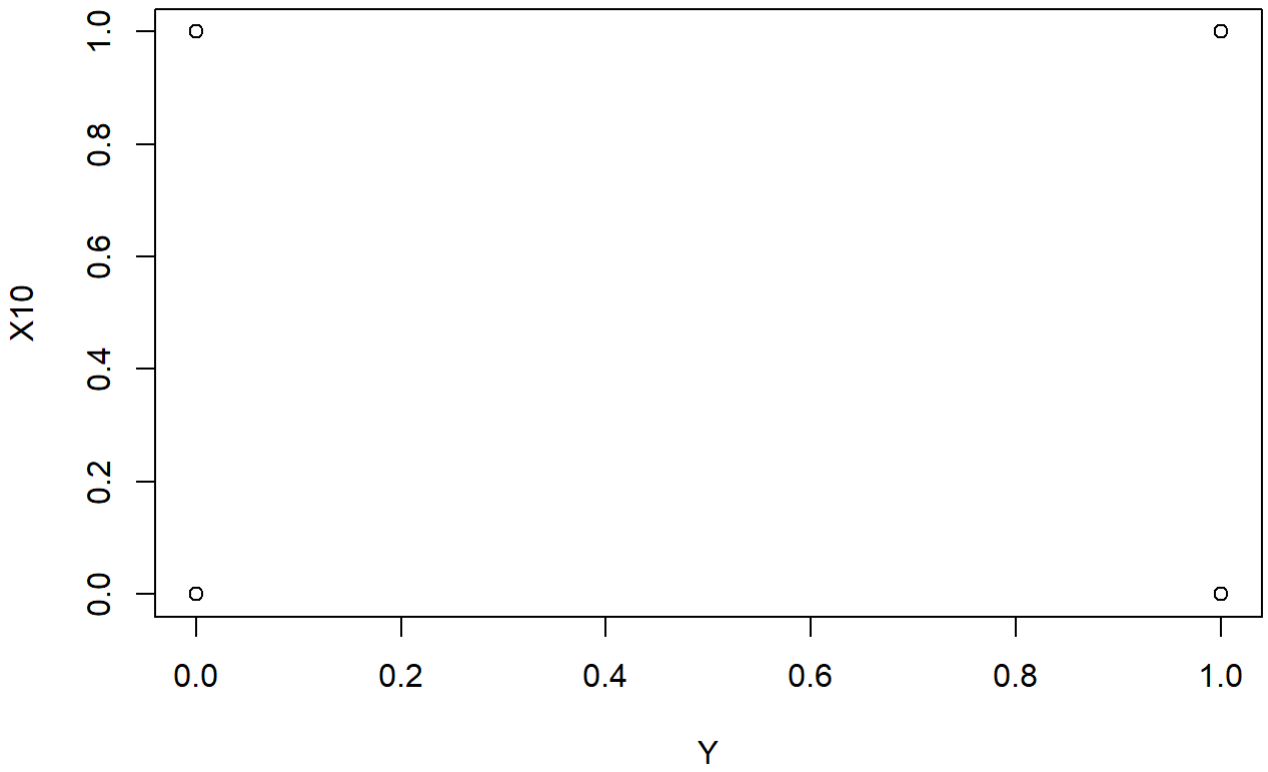

X6
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

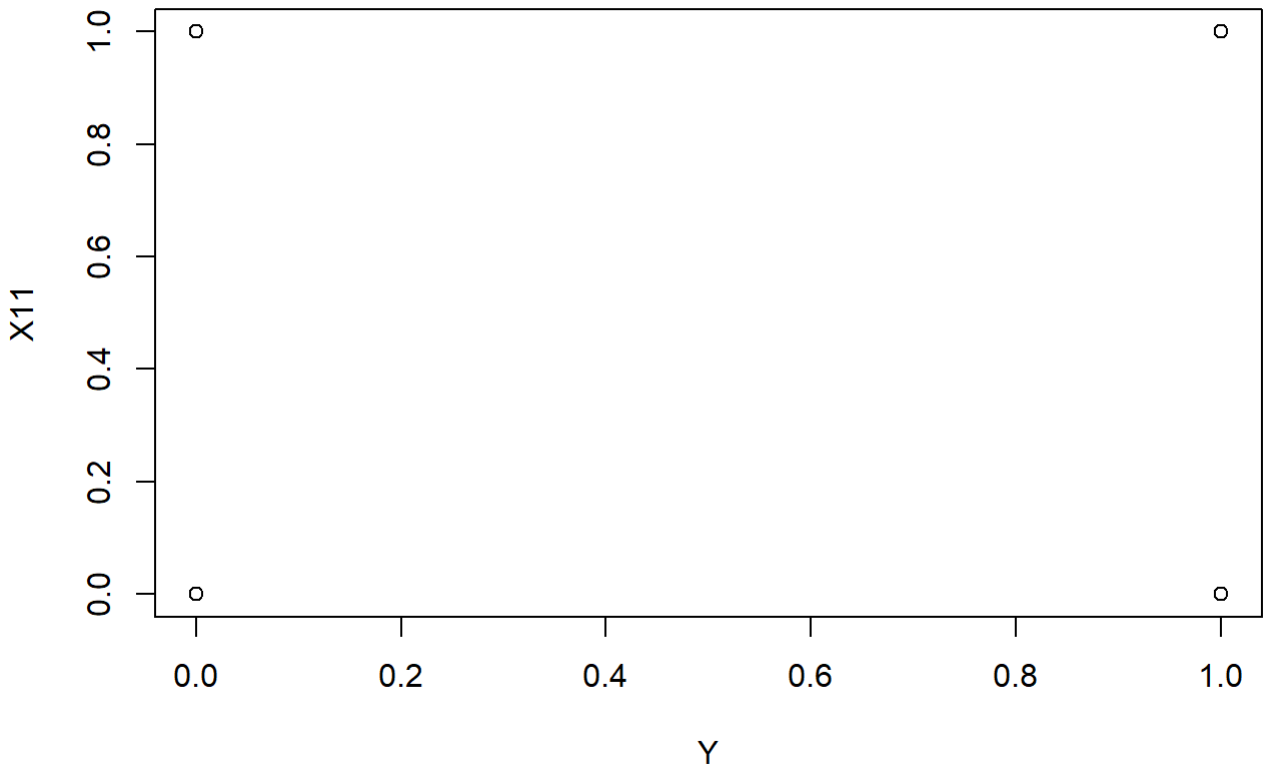

X7
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

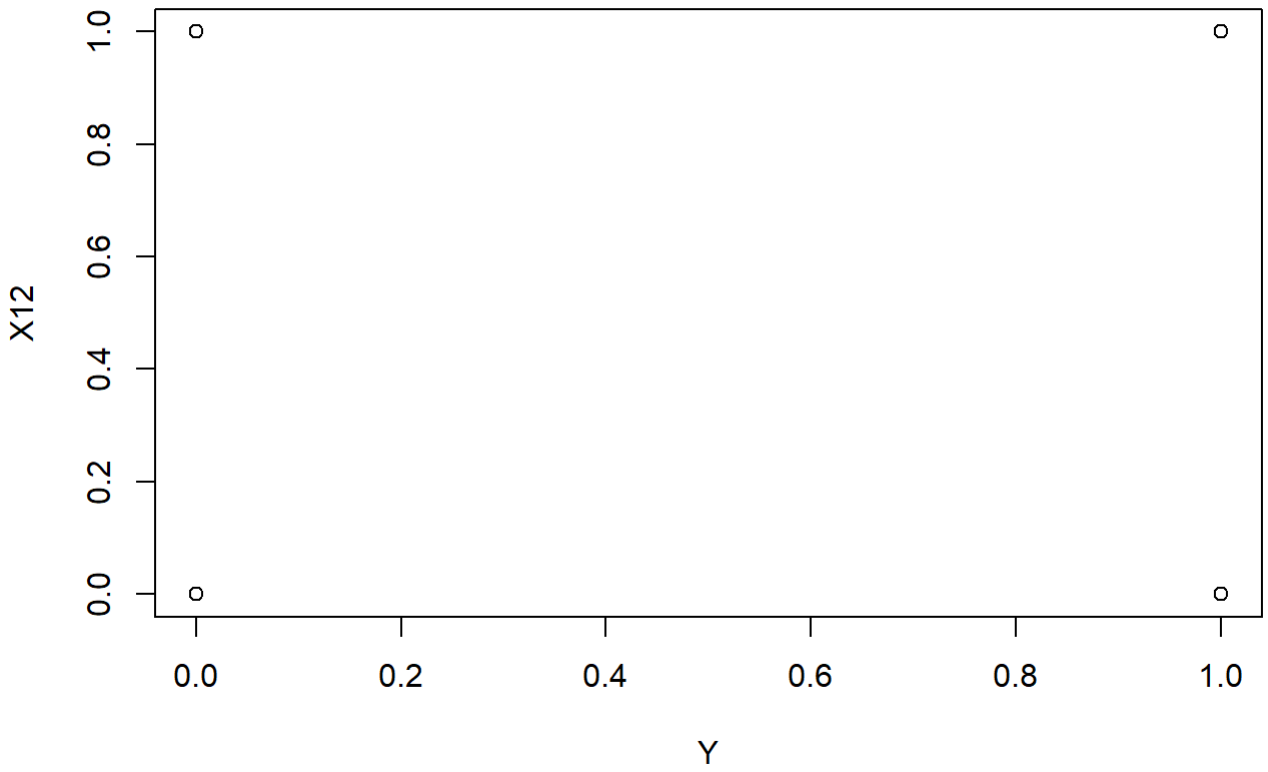

X4
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

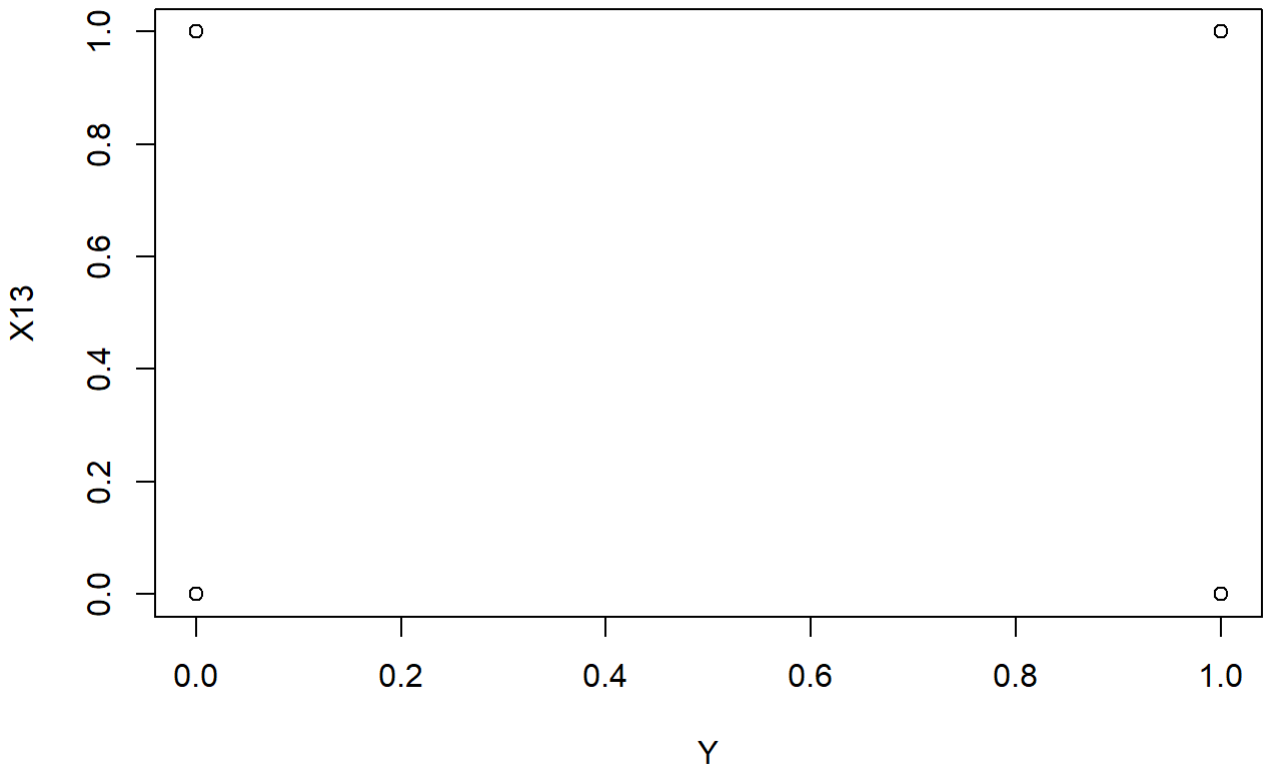

X8
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

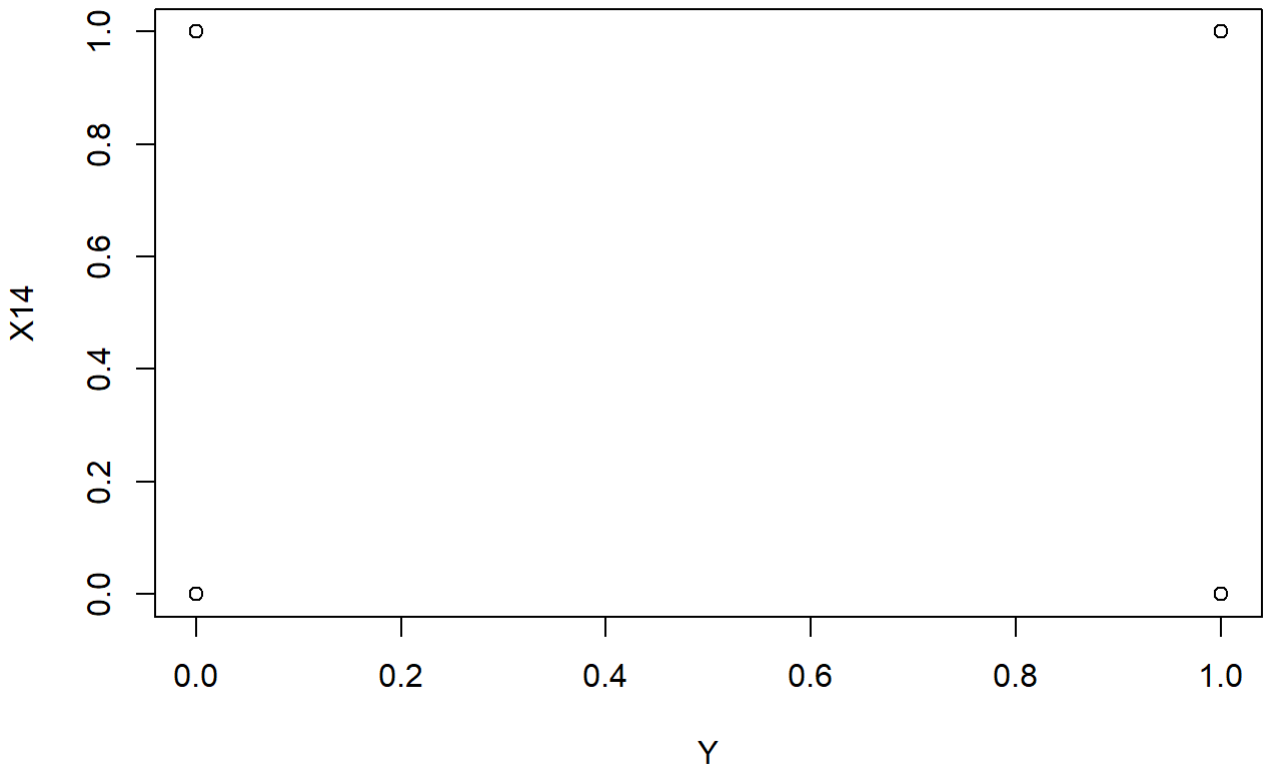

X9
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

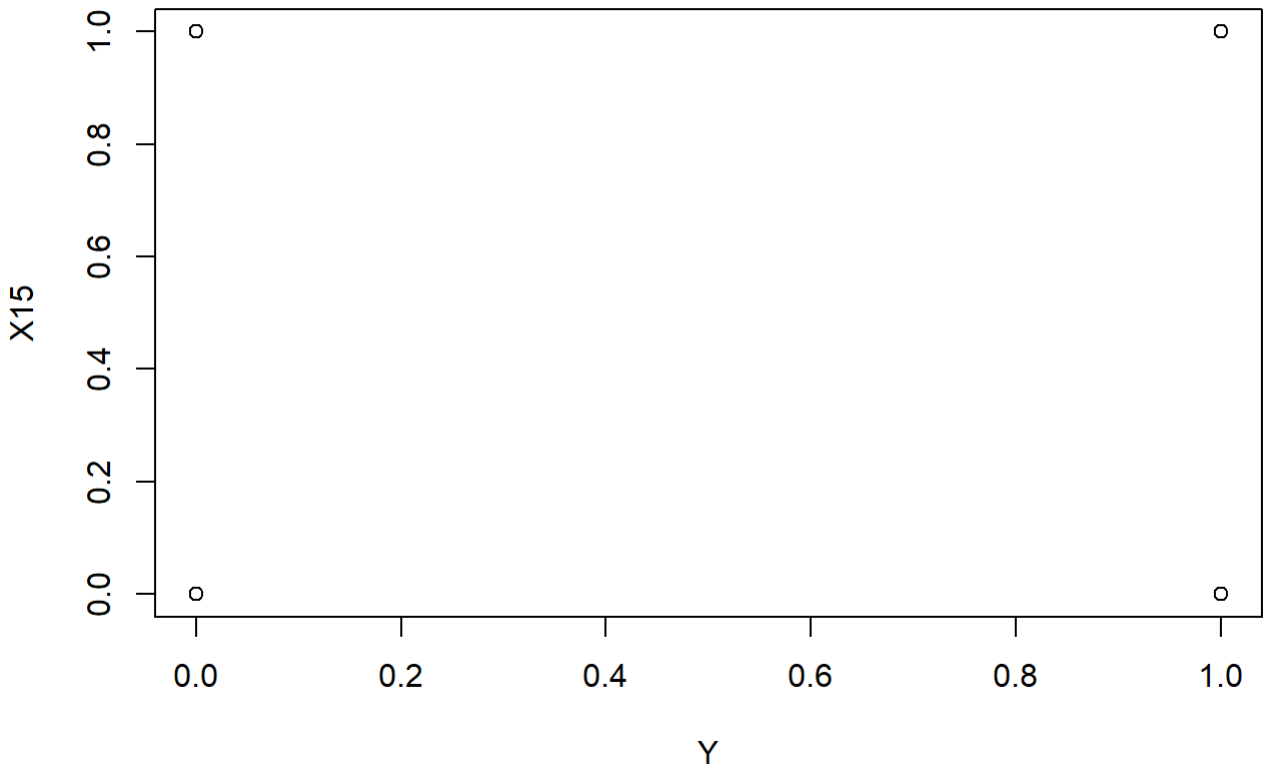

X10
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

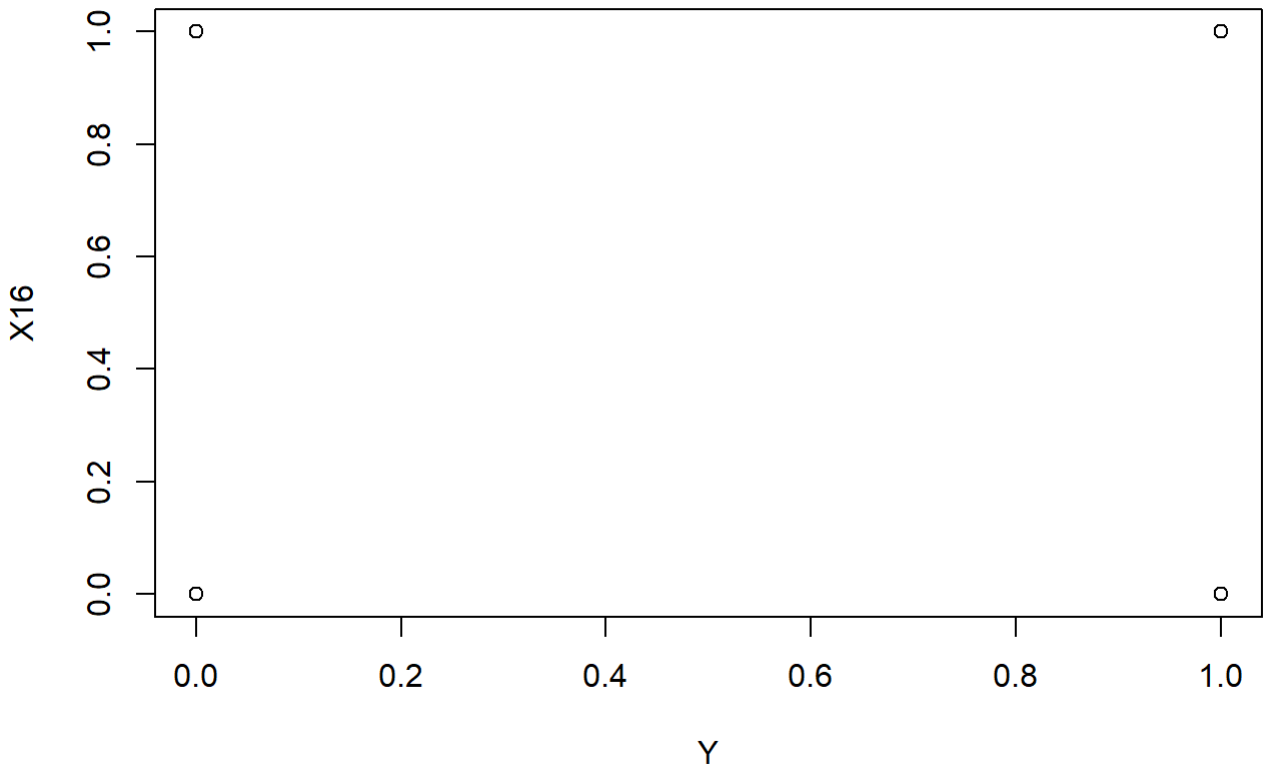

X11
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

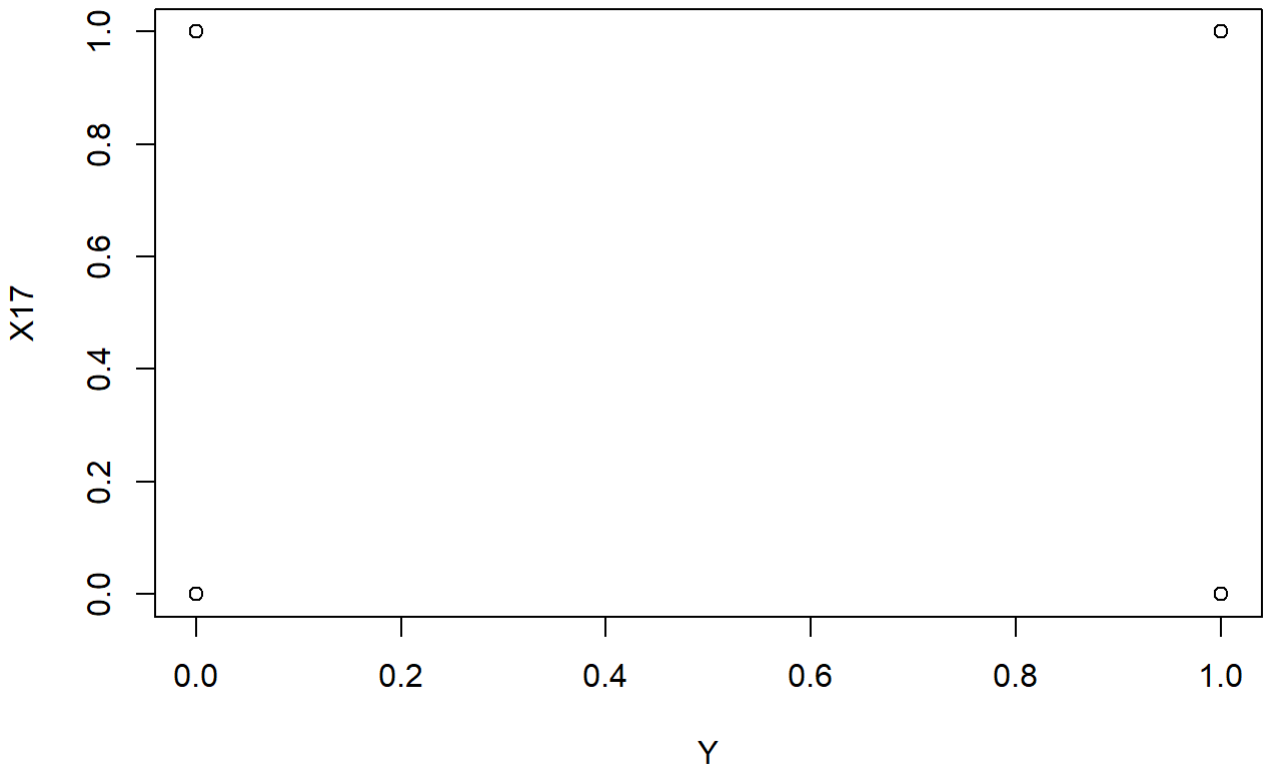

X12
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

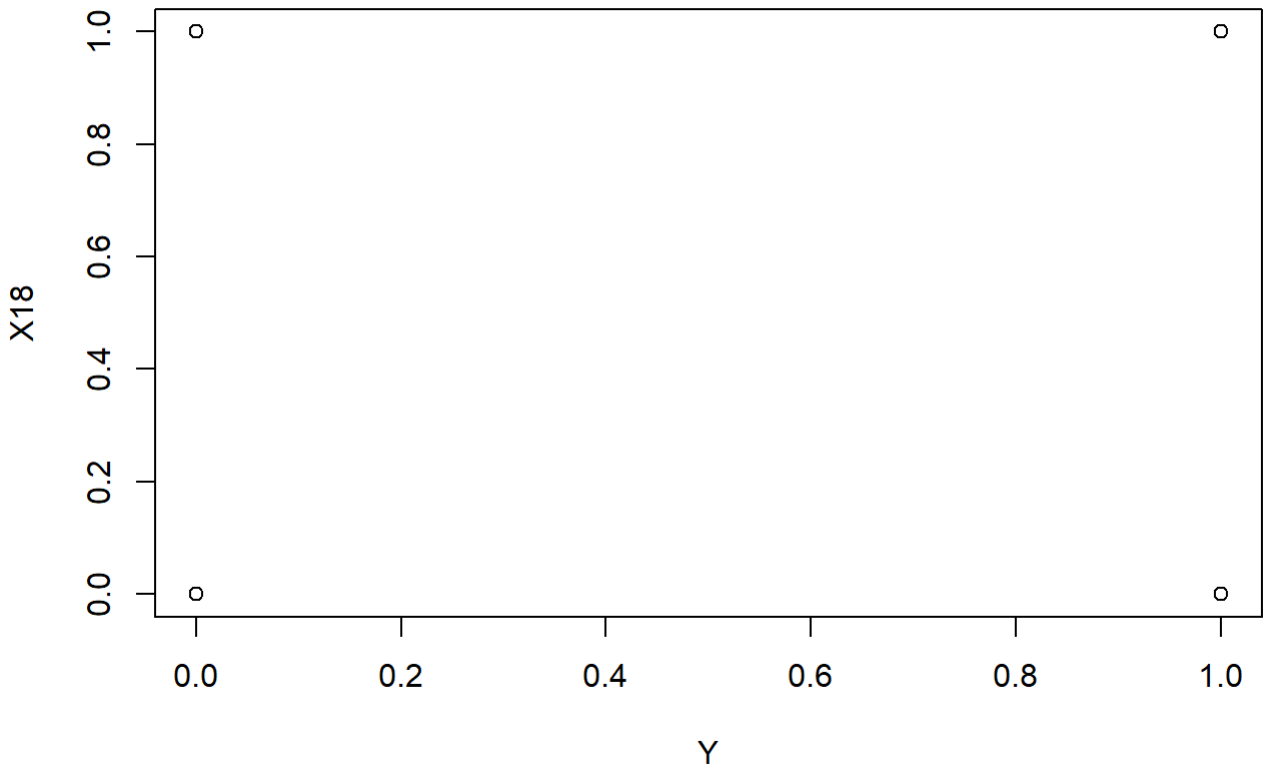

X13
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

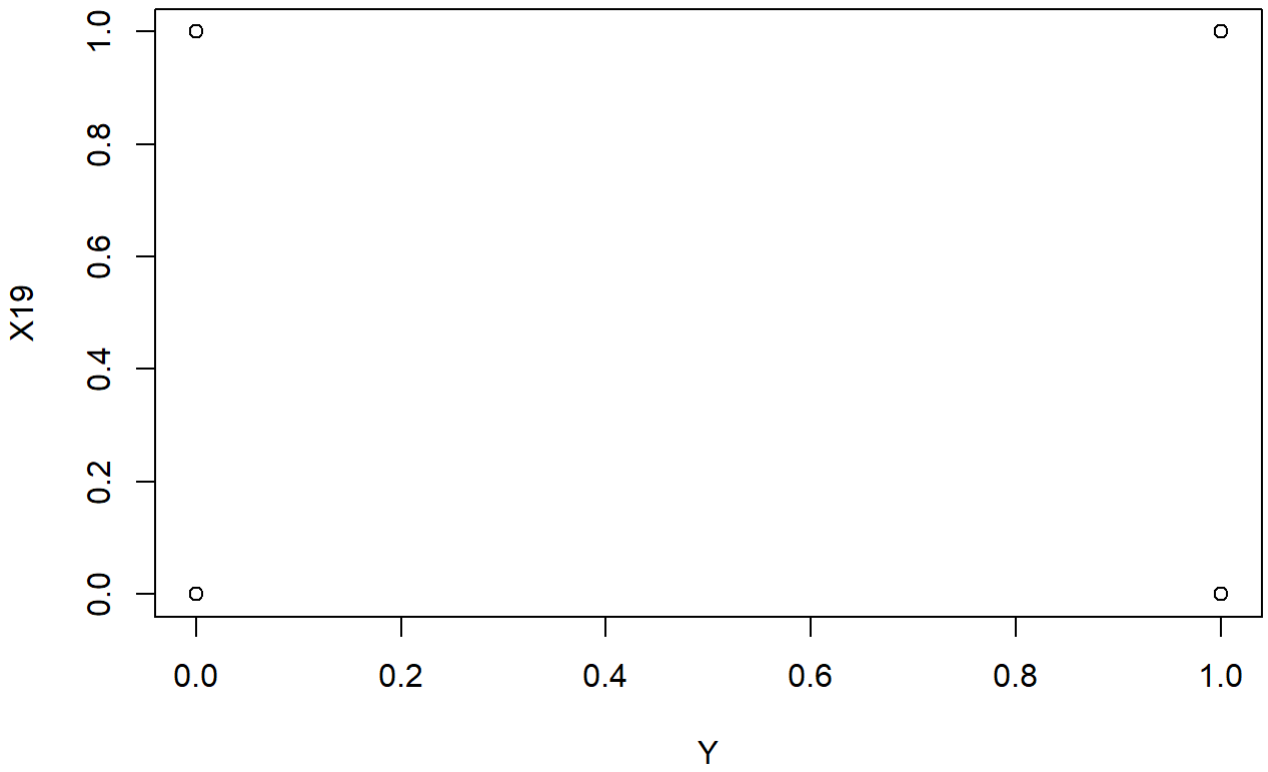

X14
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

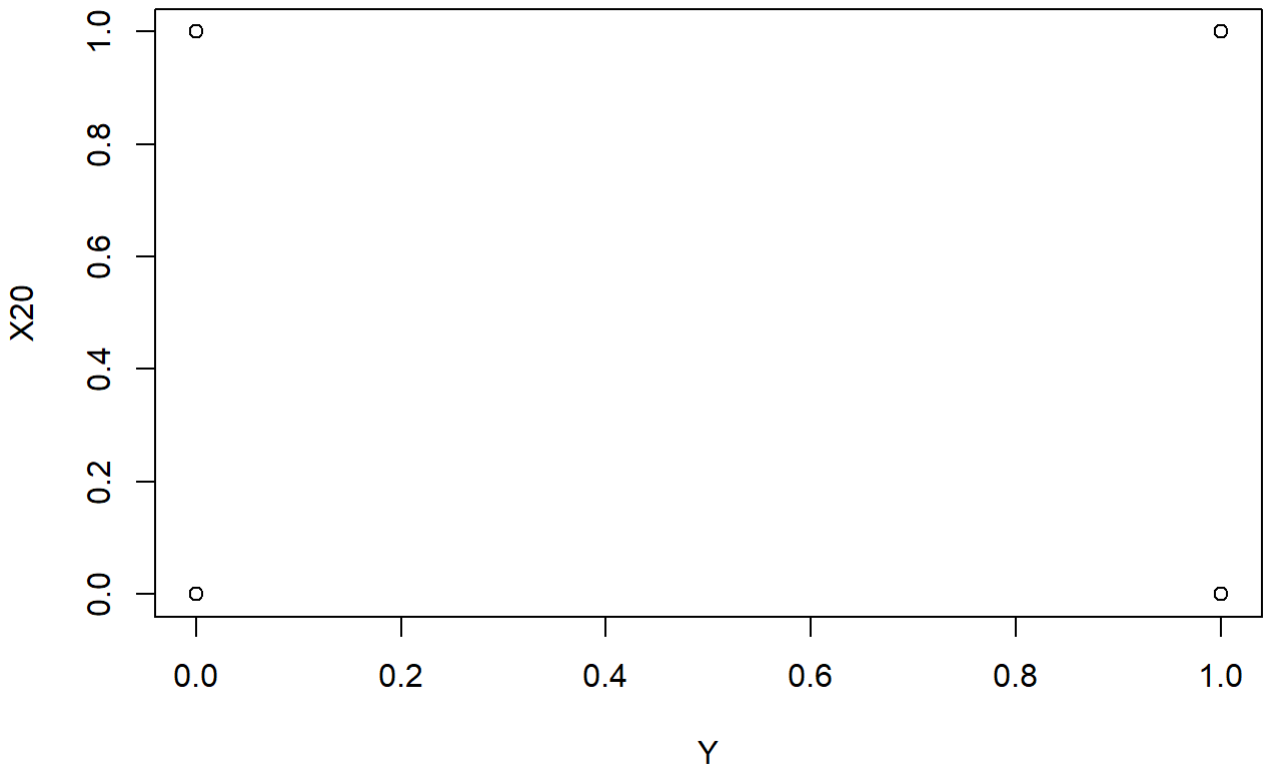

X15
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

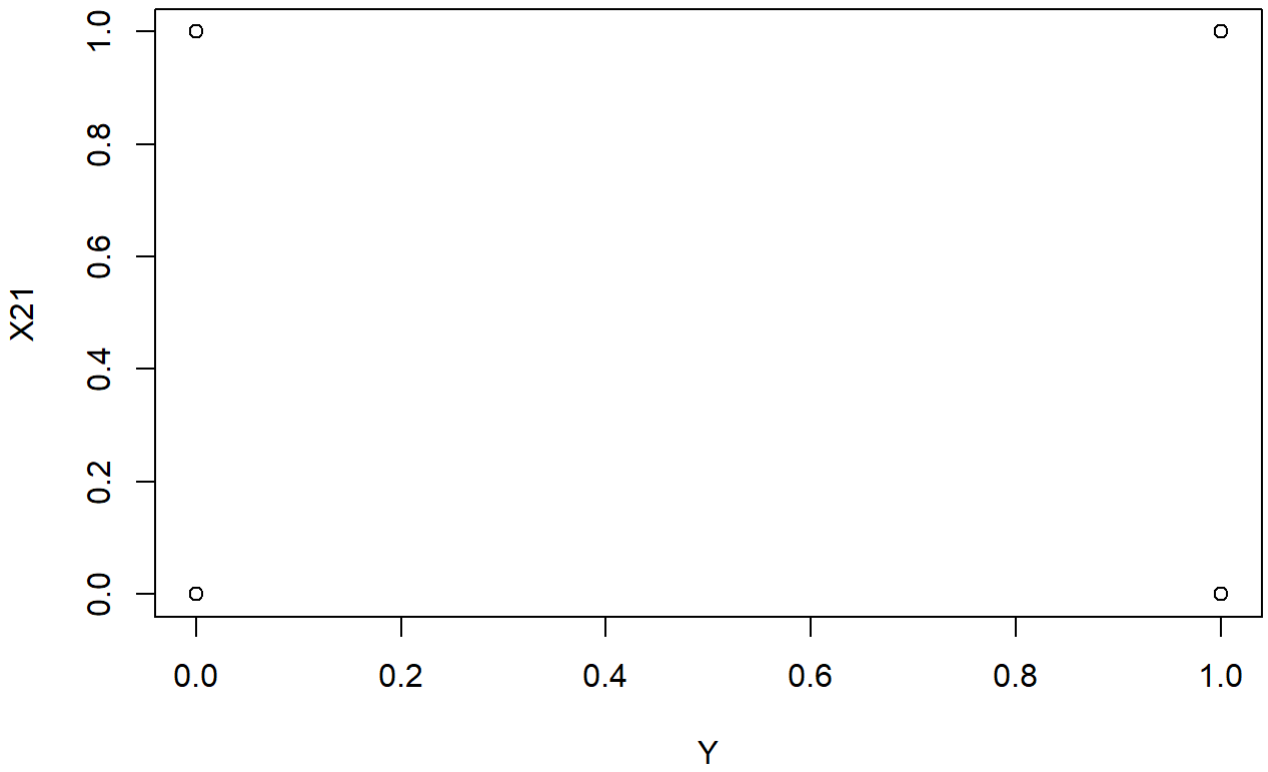

X16
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

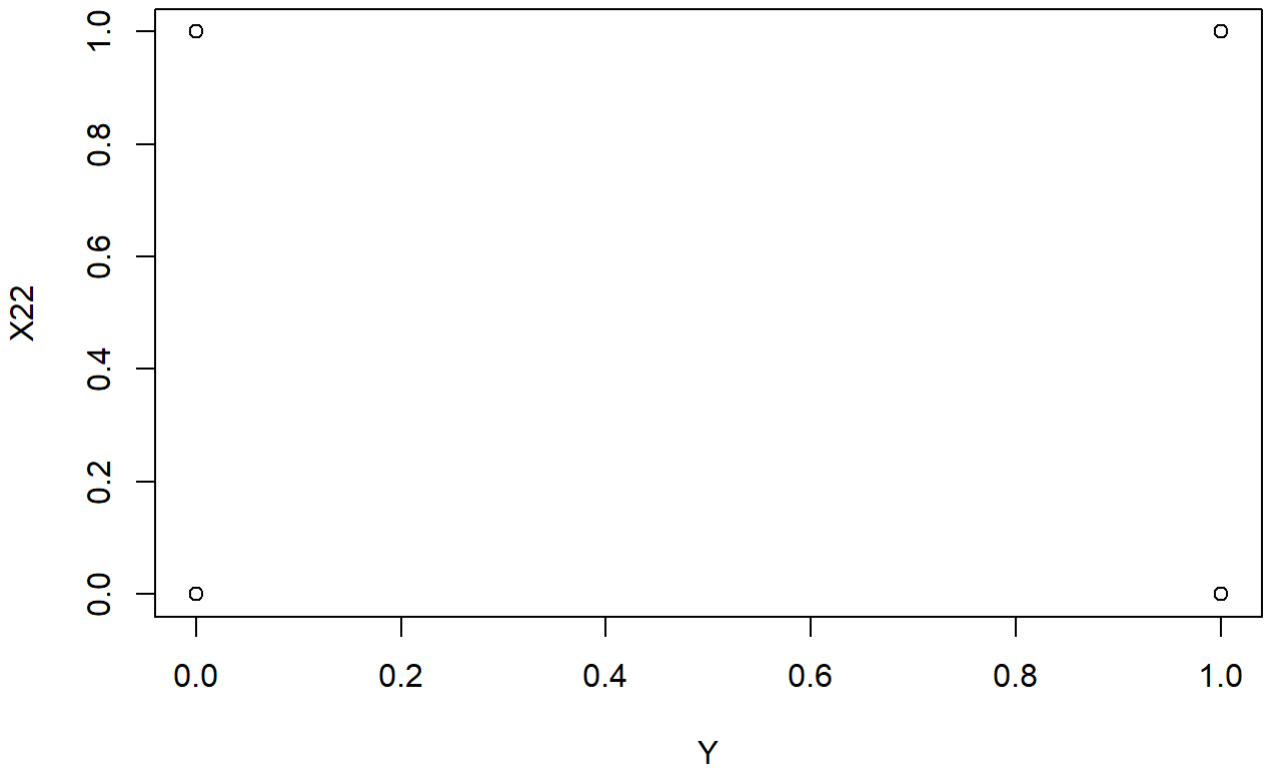

X17
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

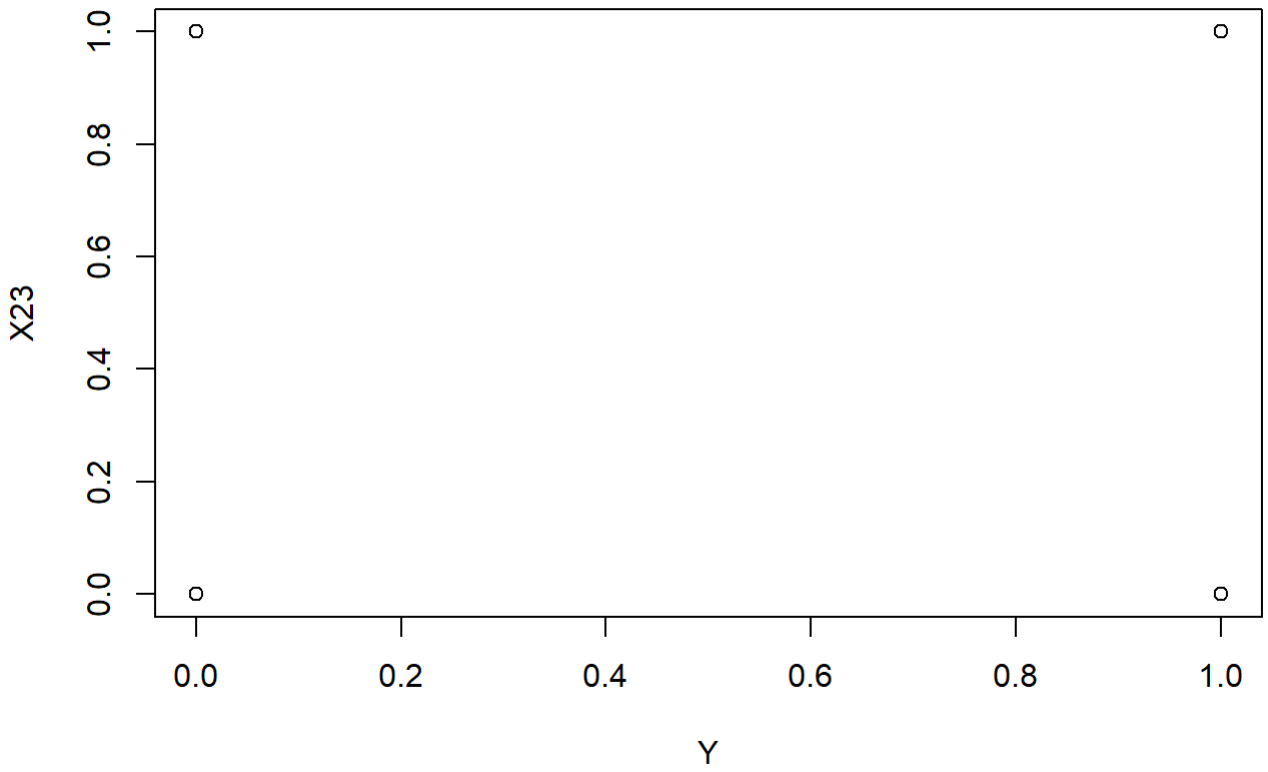

X18
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

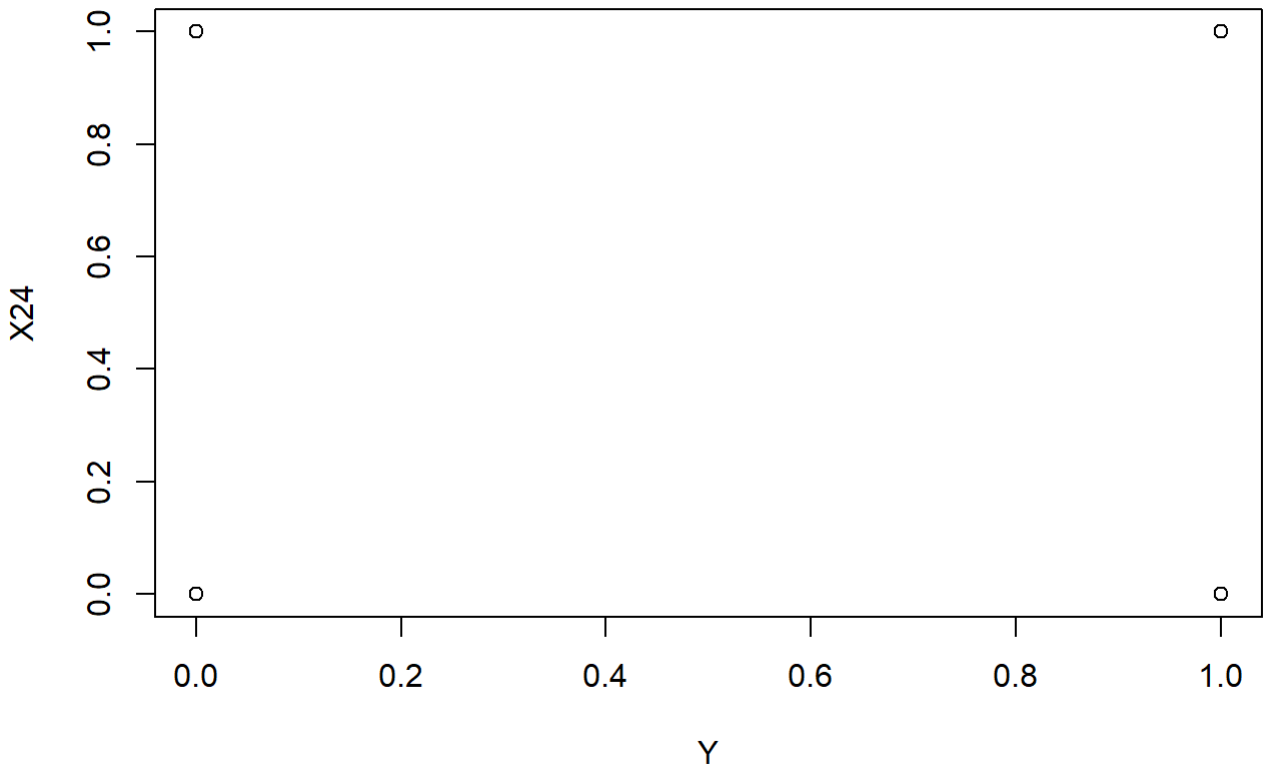

X19
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

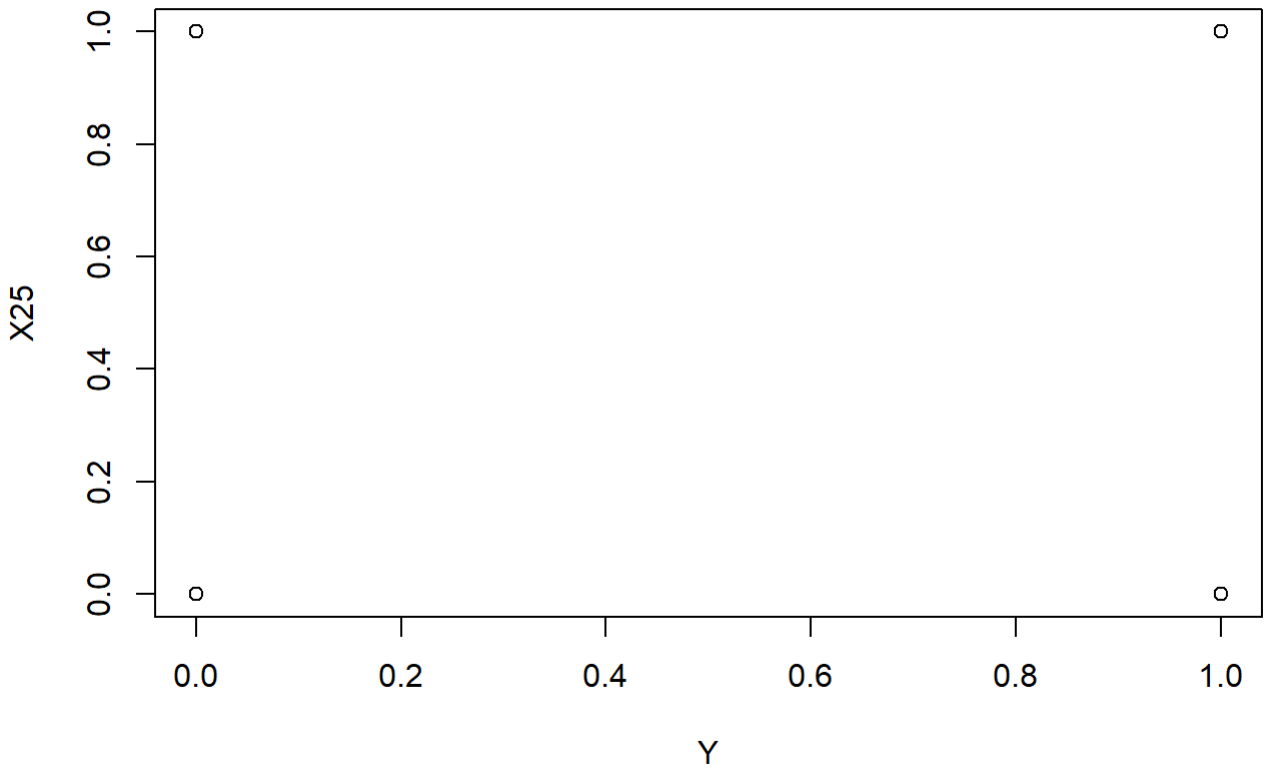

X20
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

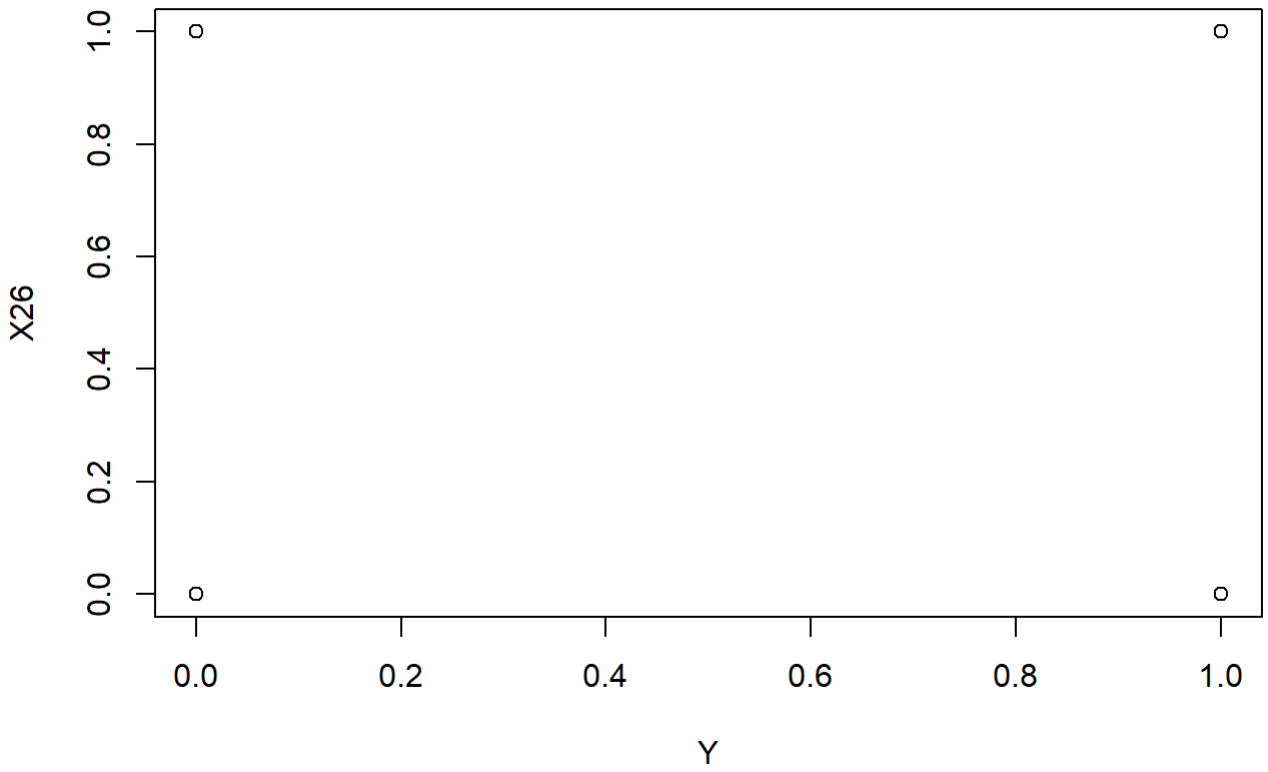

X21
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

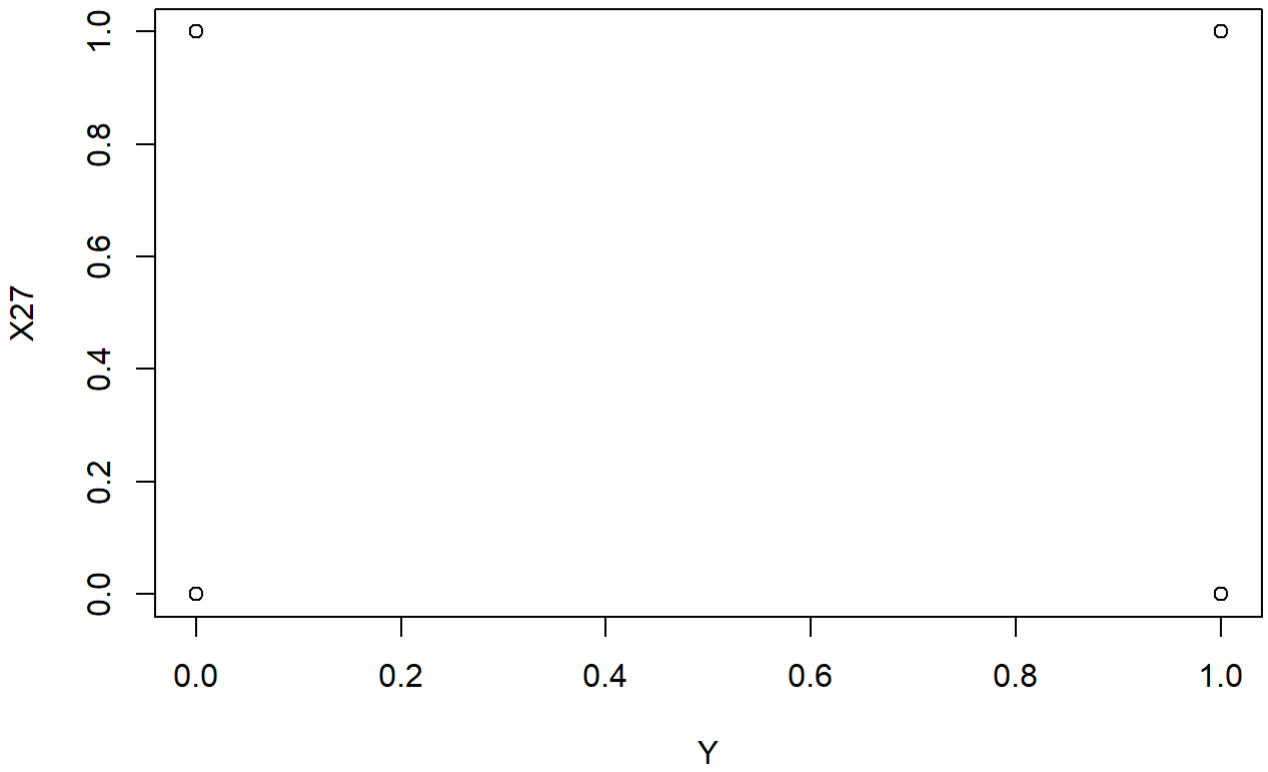

X22
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

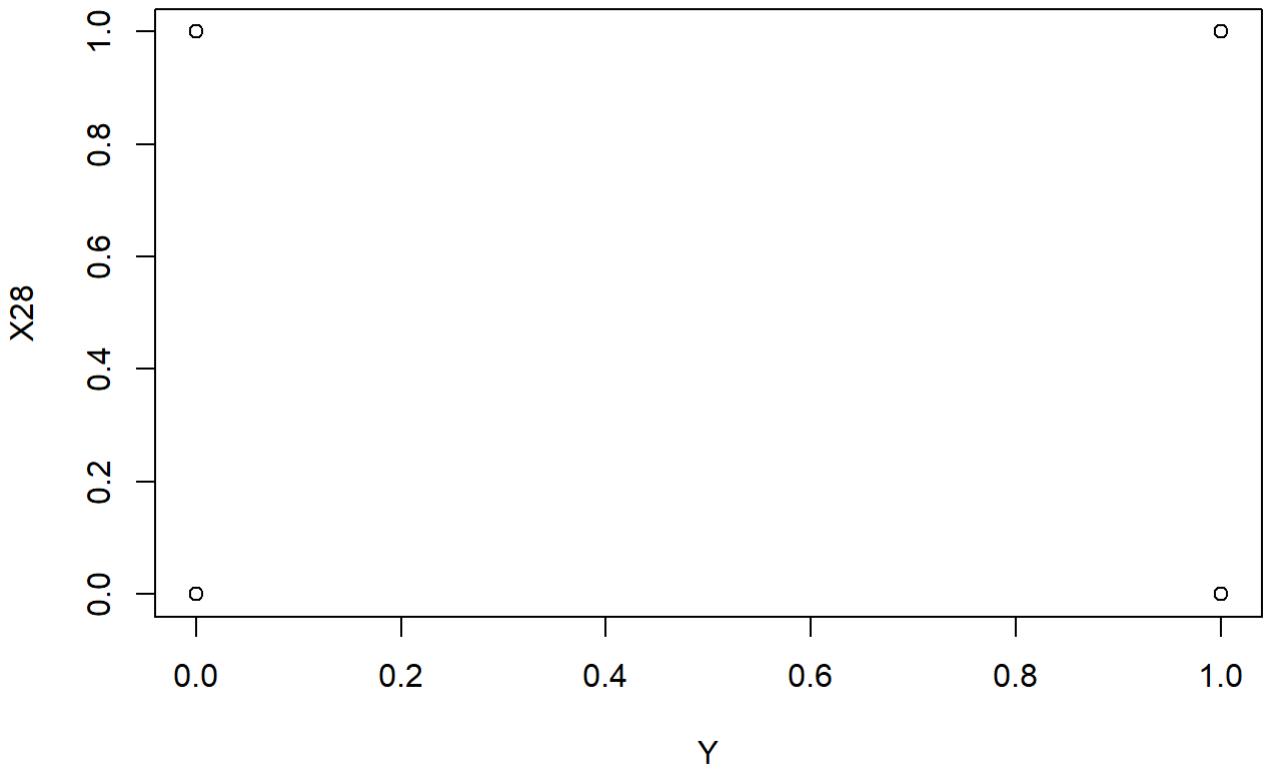

X23
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

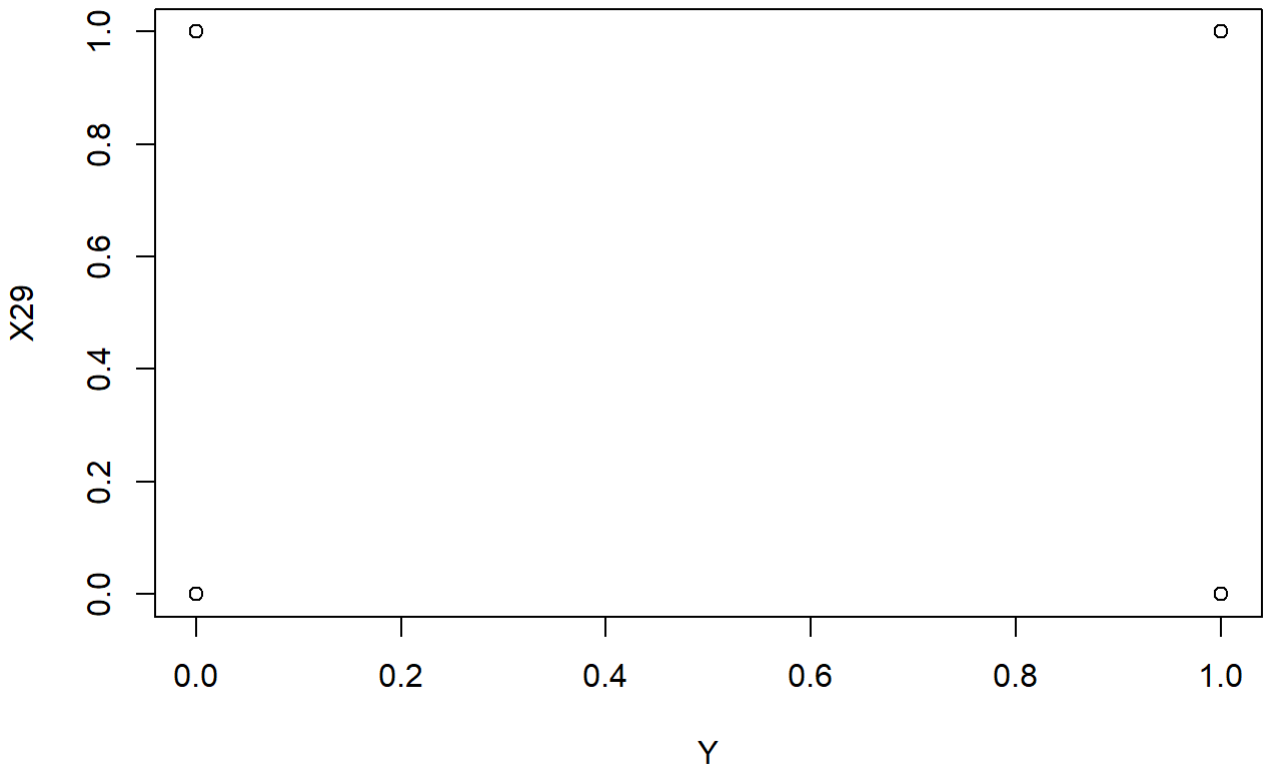

X24
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

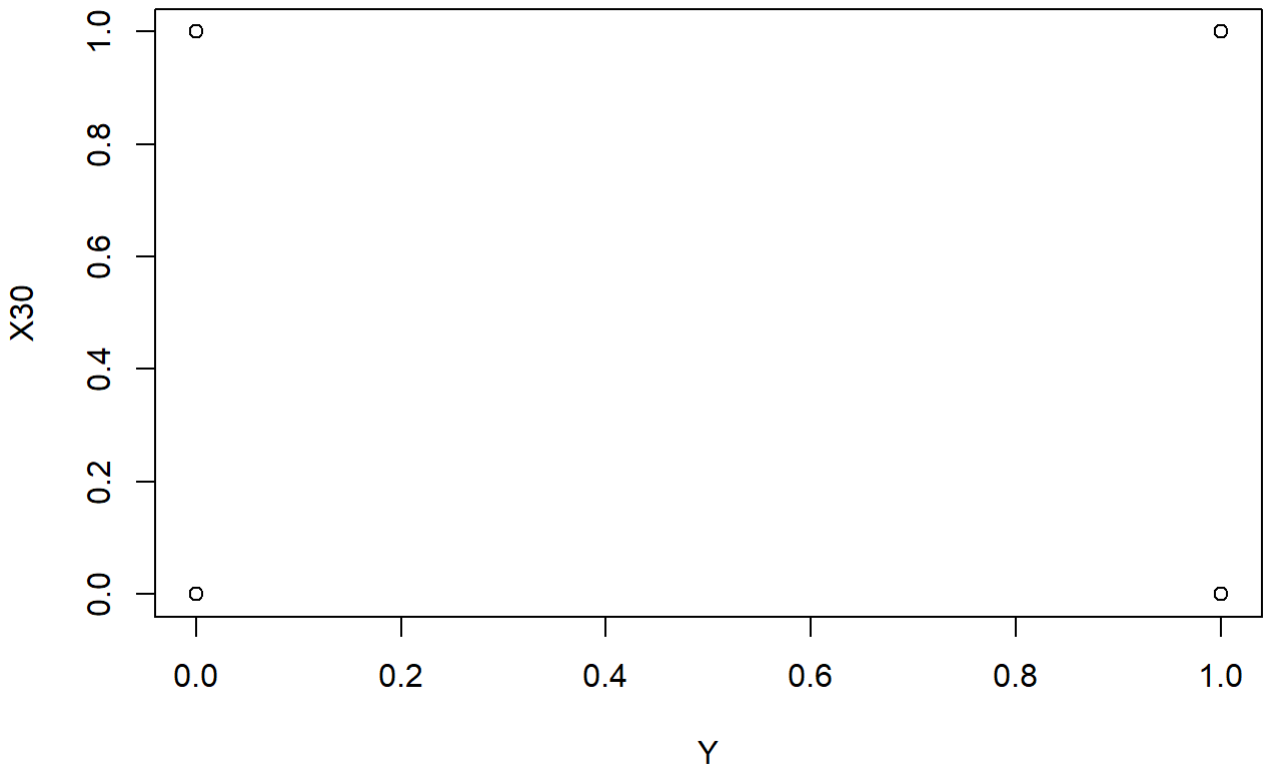

X25
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

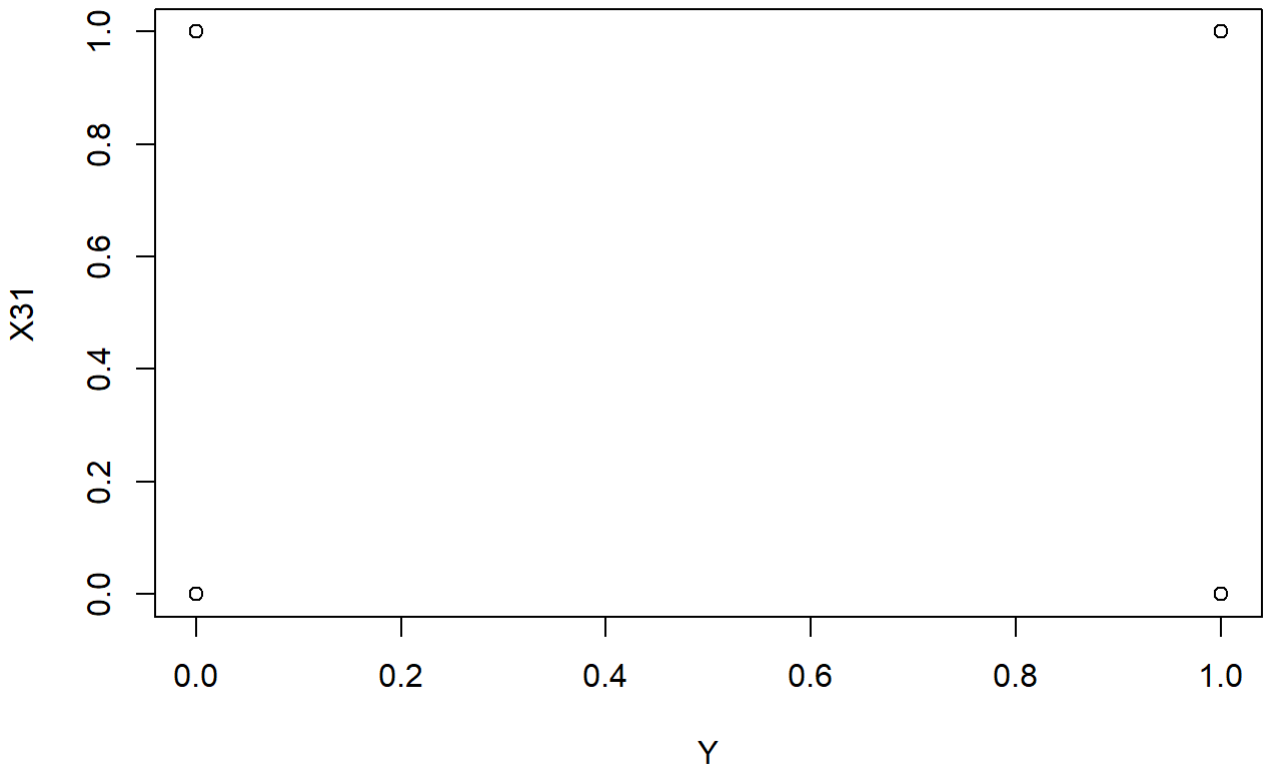

X26
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

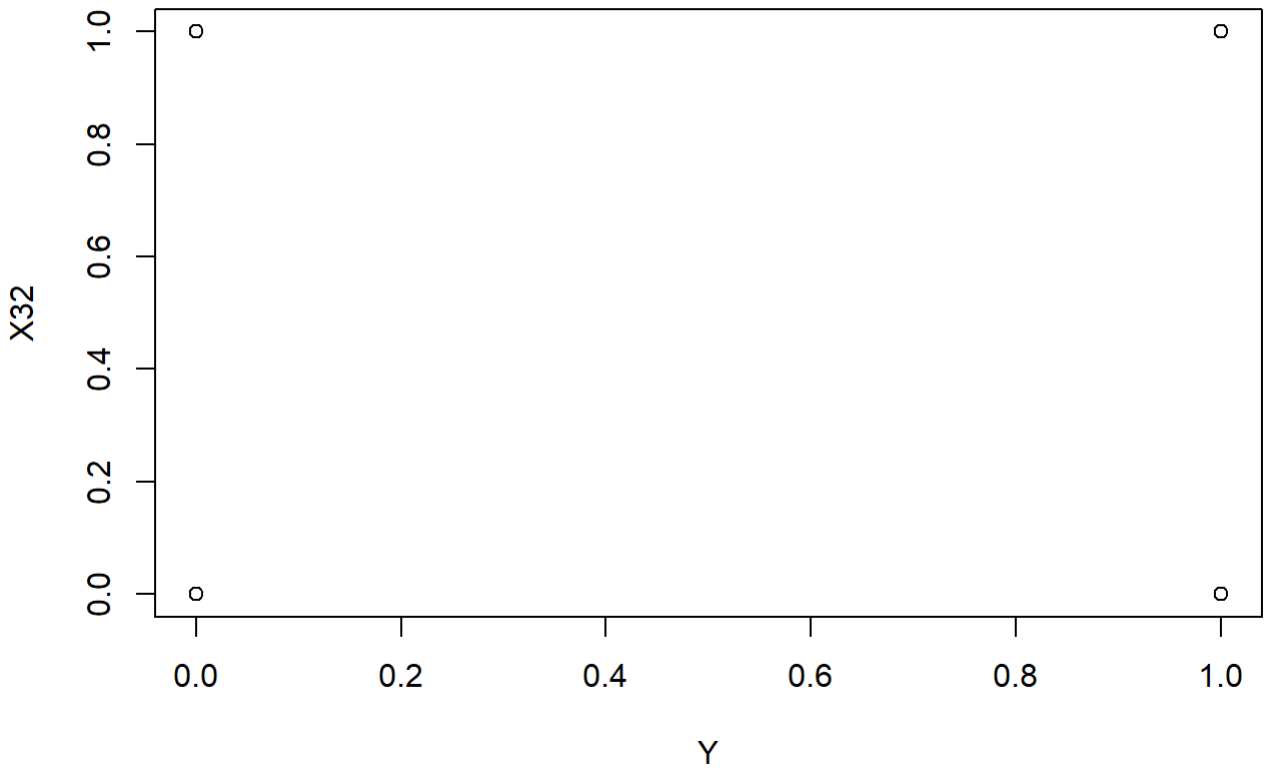

X27
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

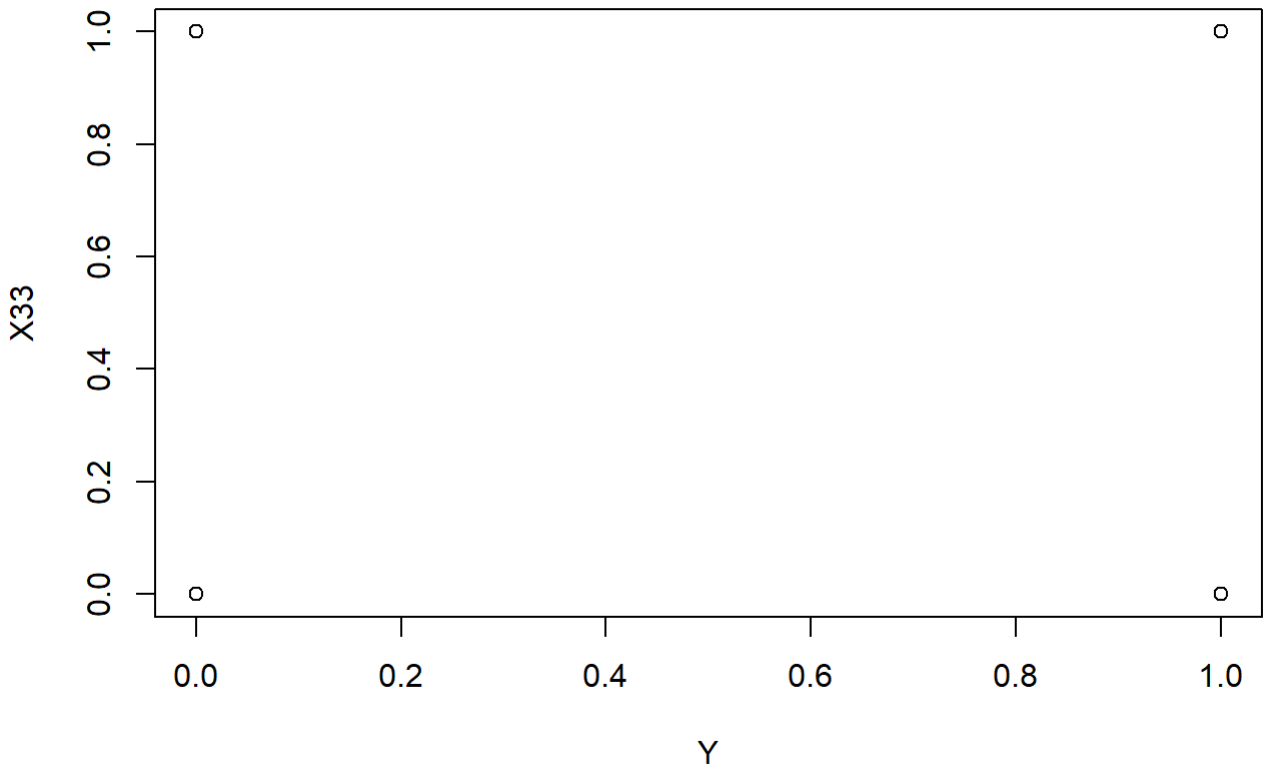

X28
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

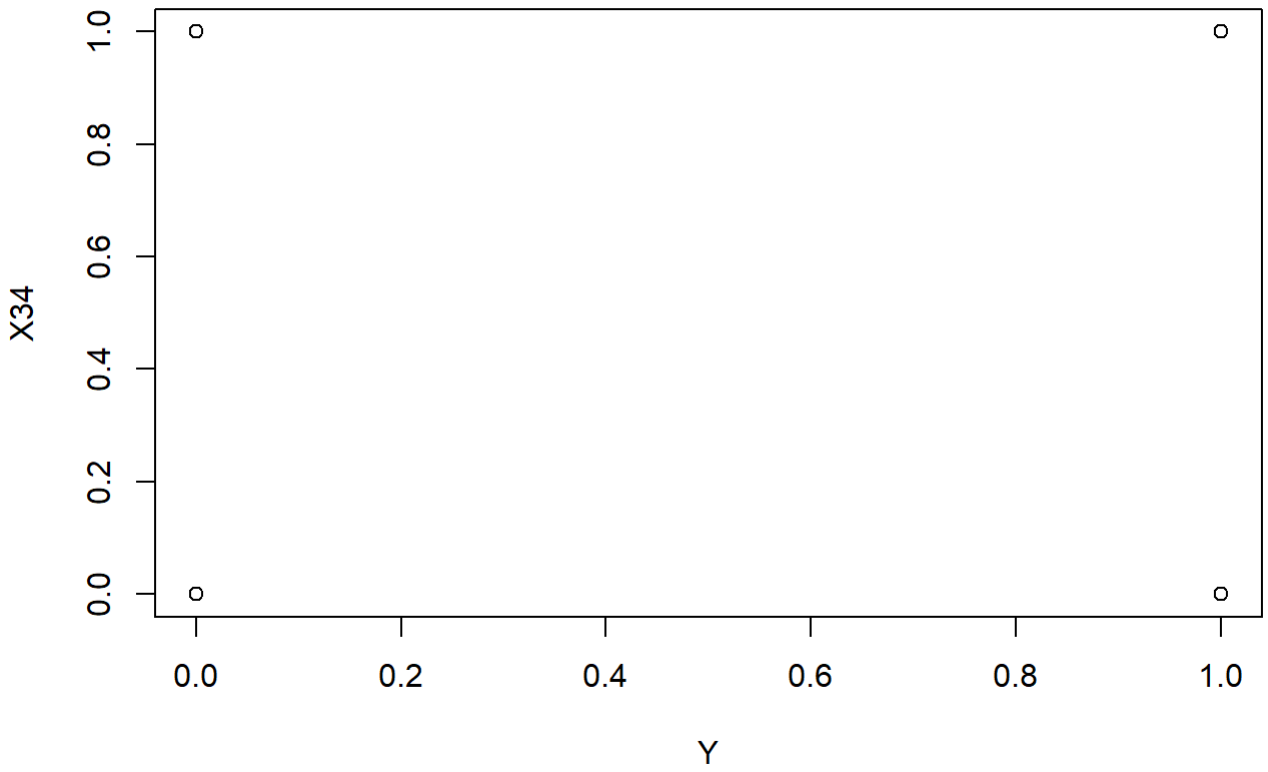

X29
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

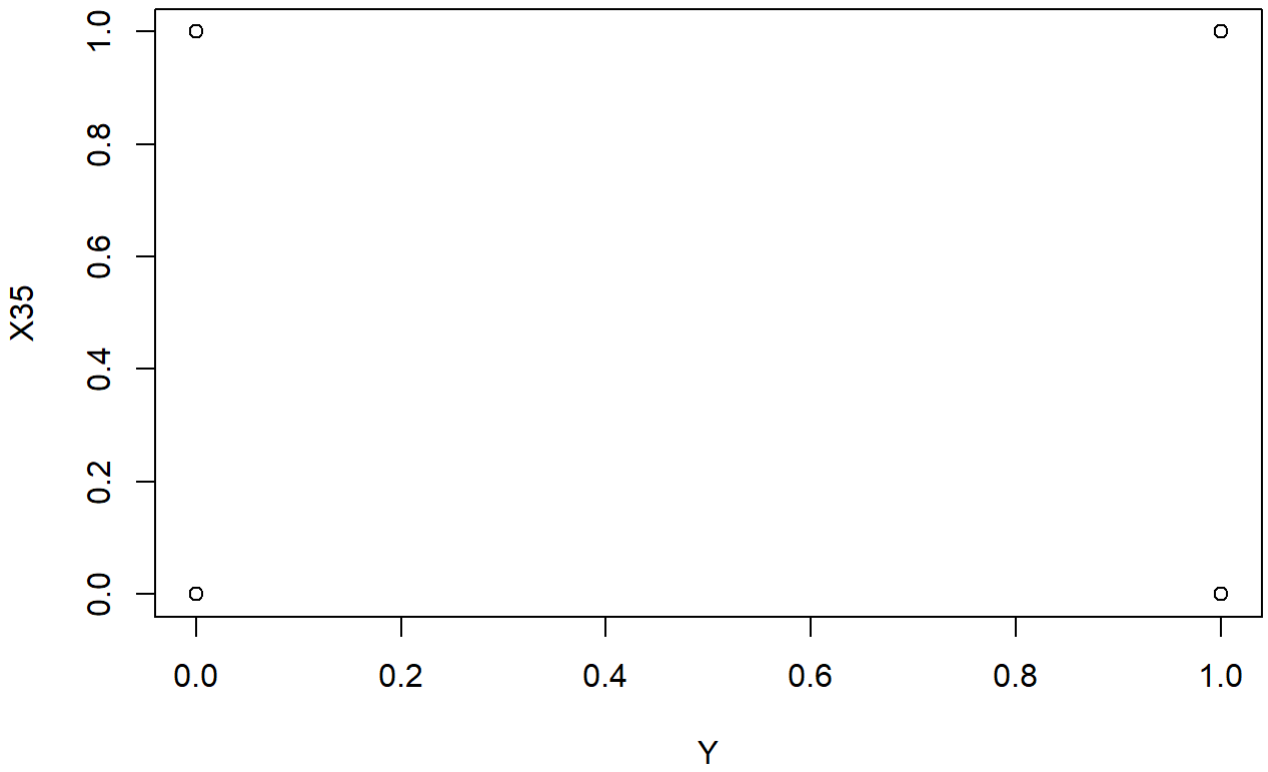

X30
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

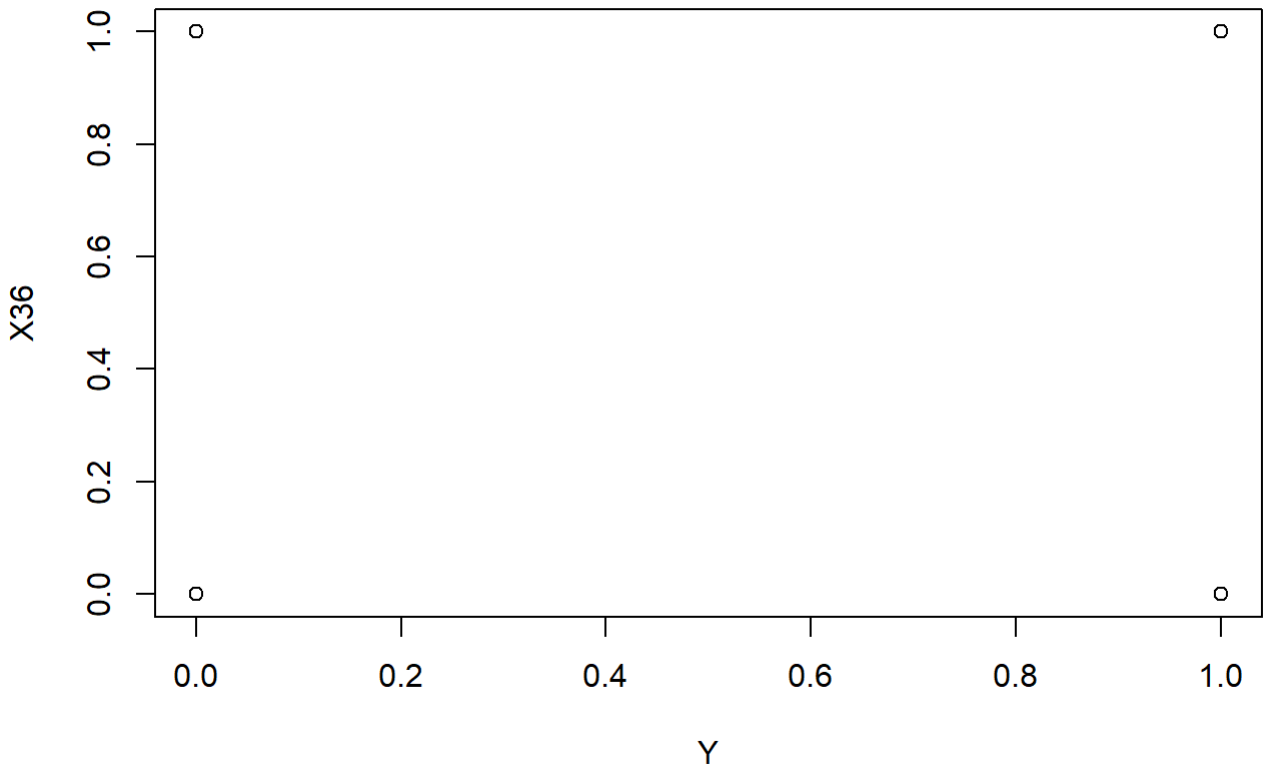

X31
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

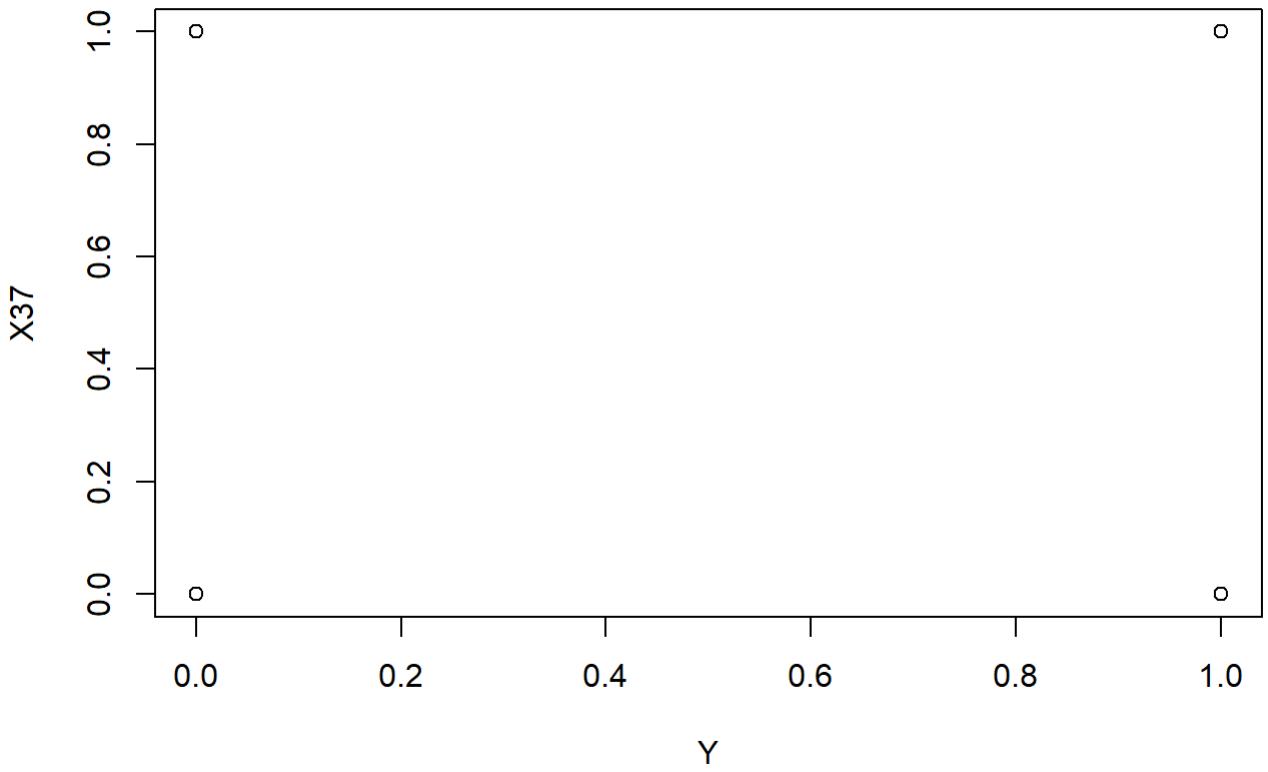

X32
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

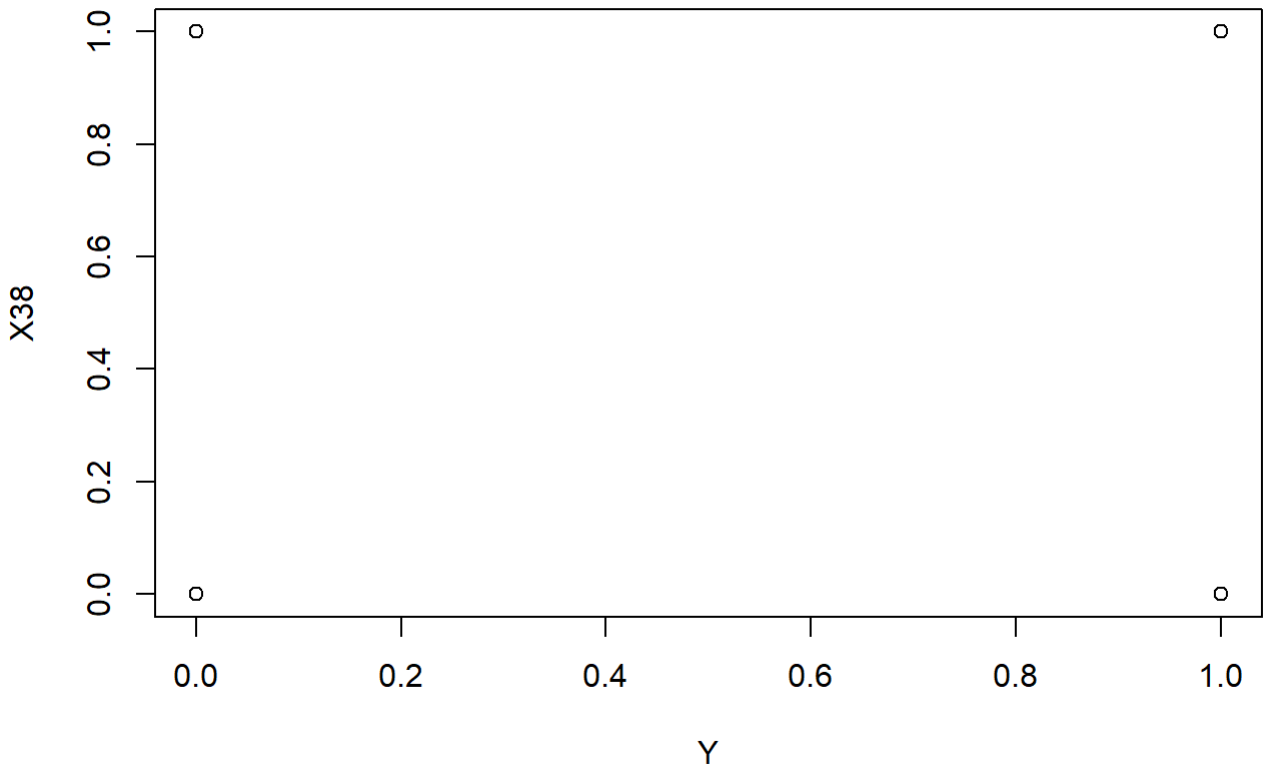

X33
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

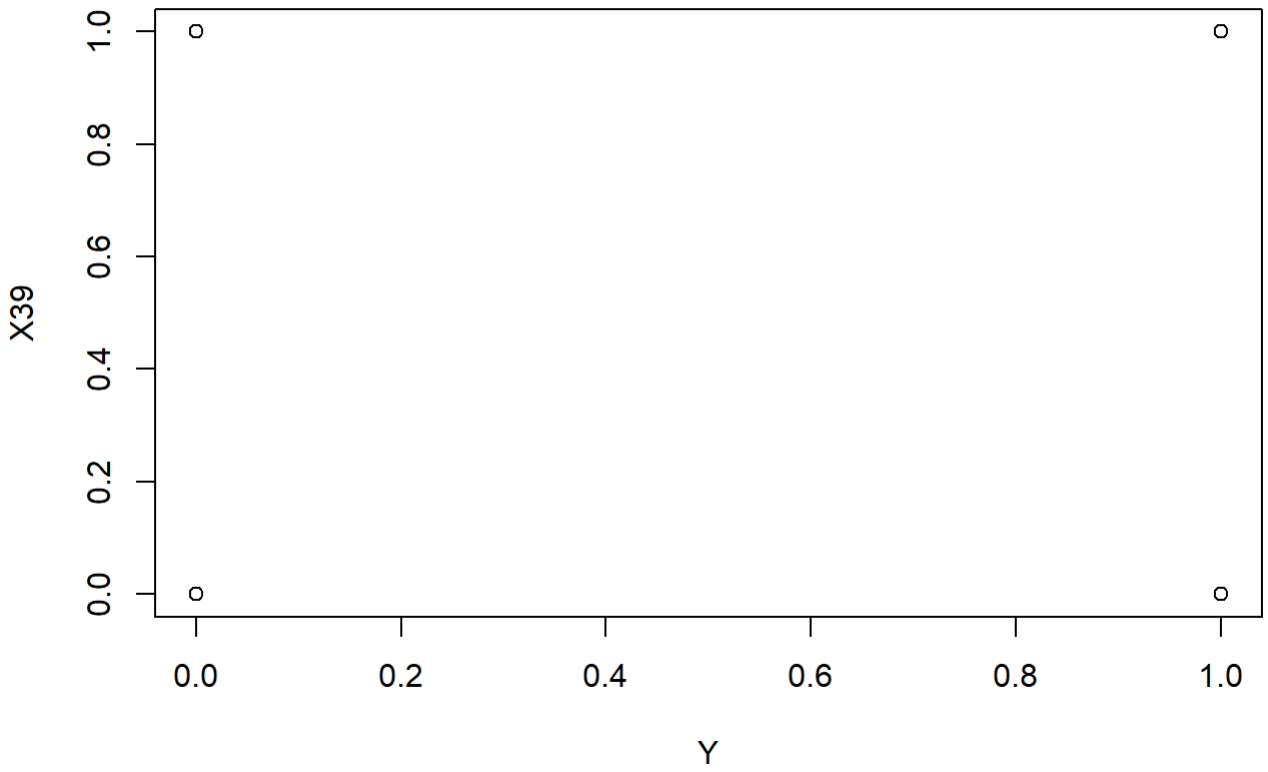

X34
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

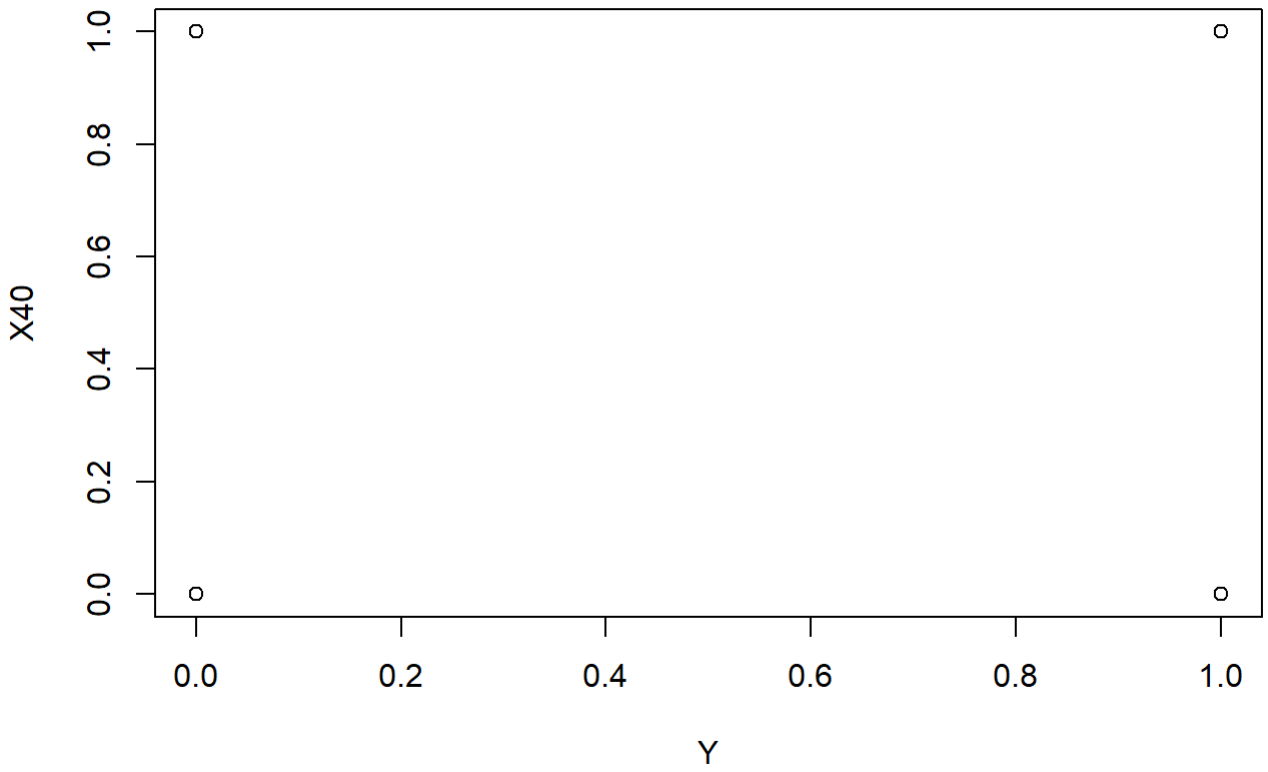

X35
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

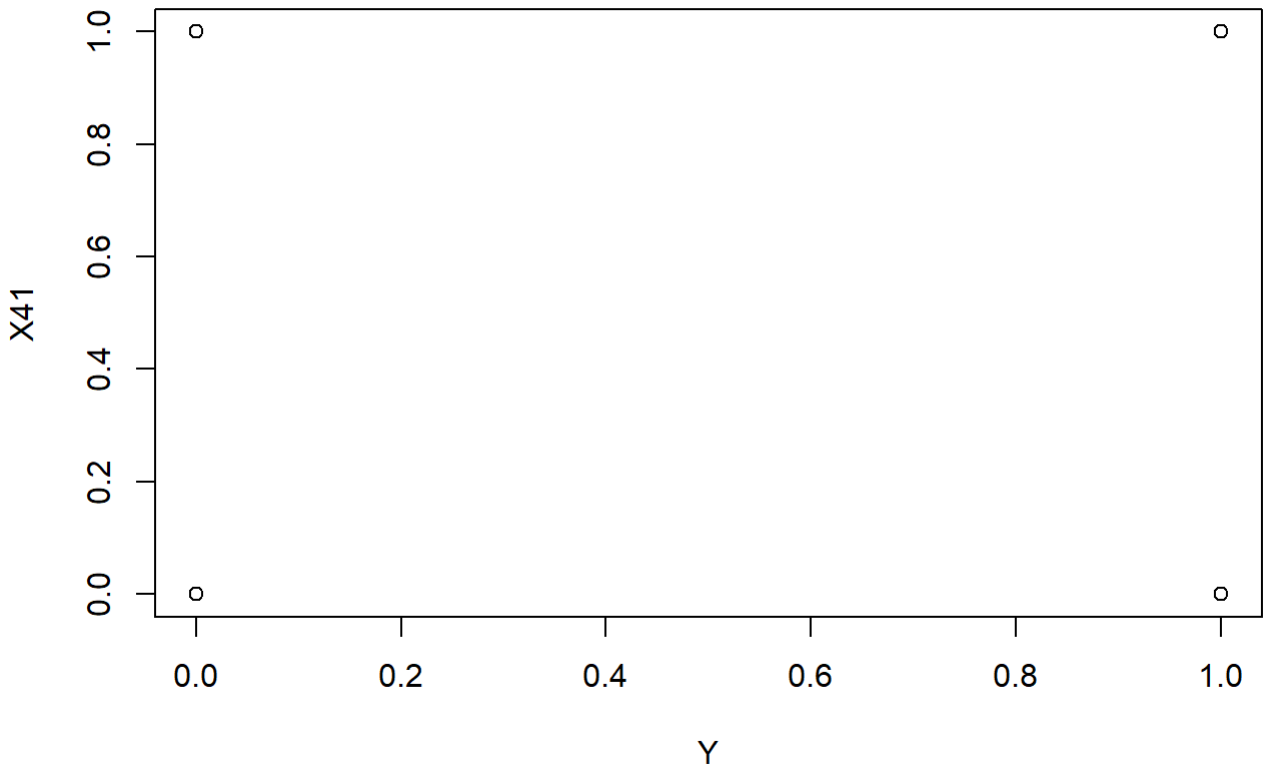

X36
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

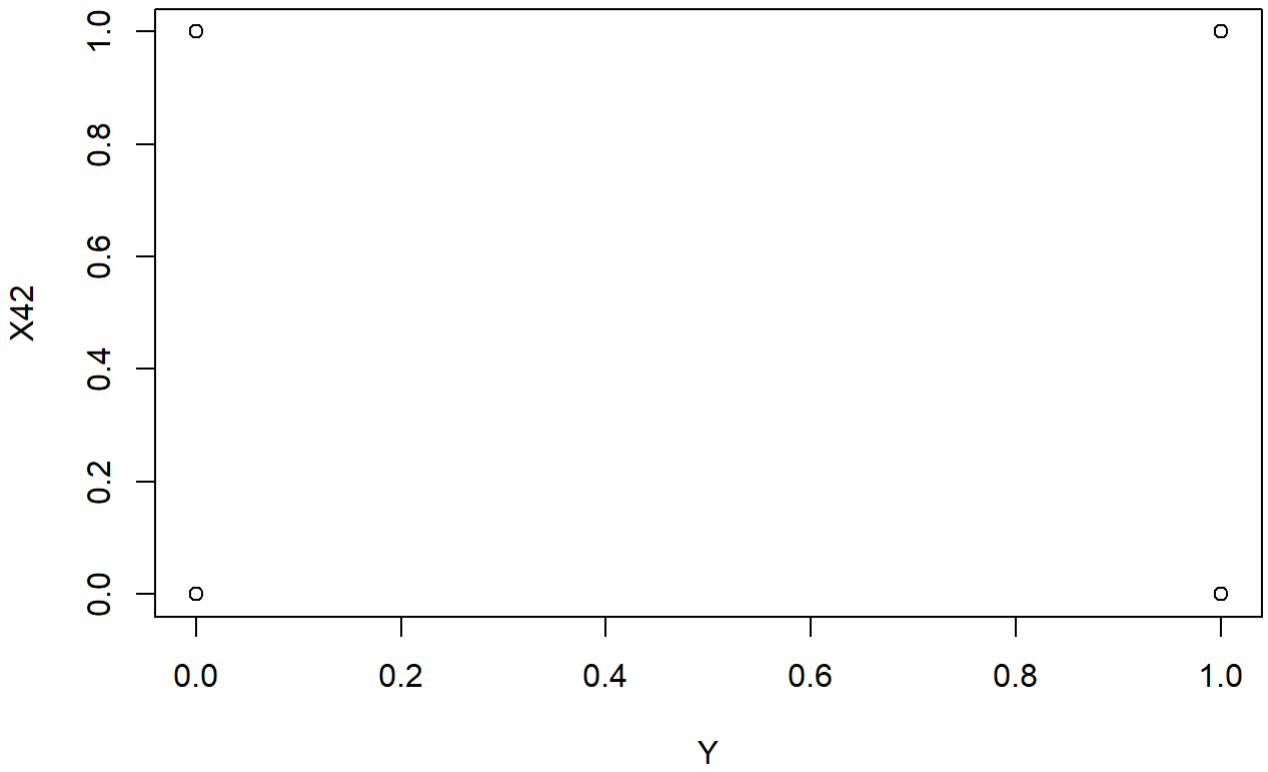

X37
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

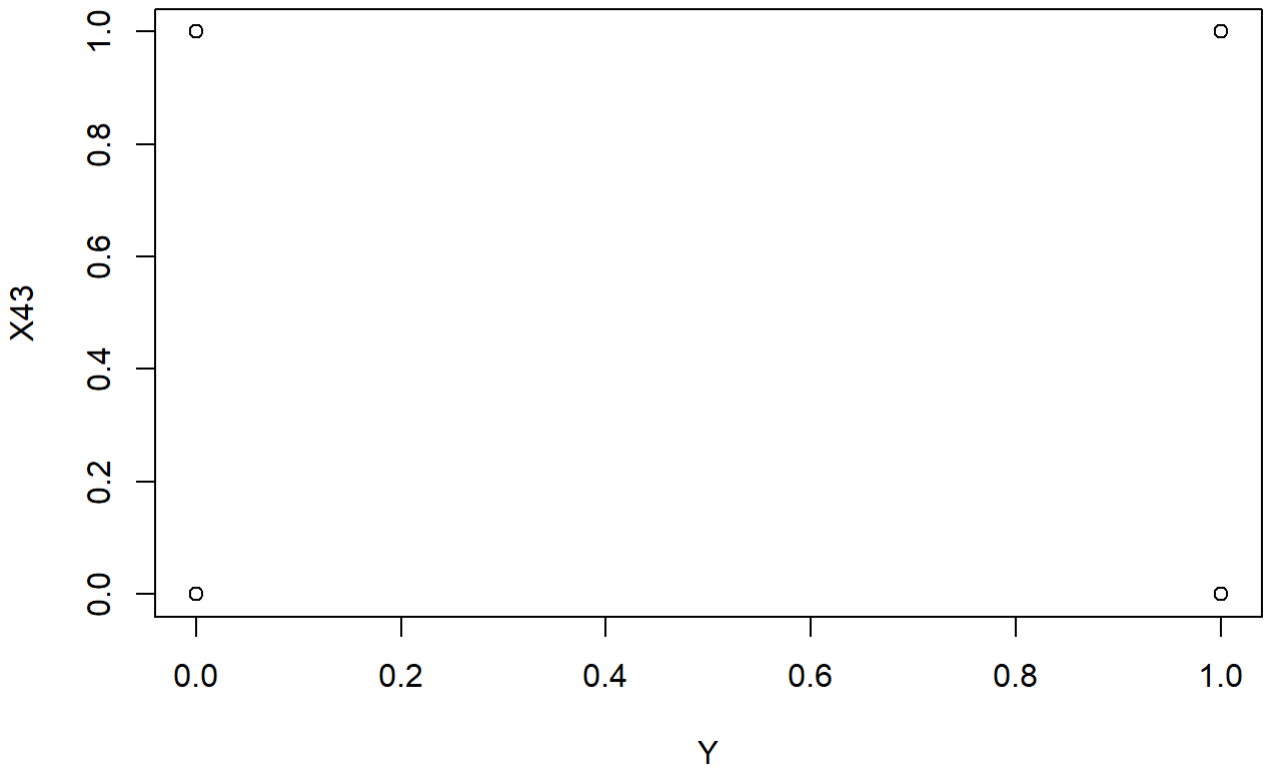

X38
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

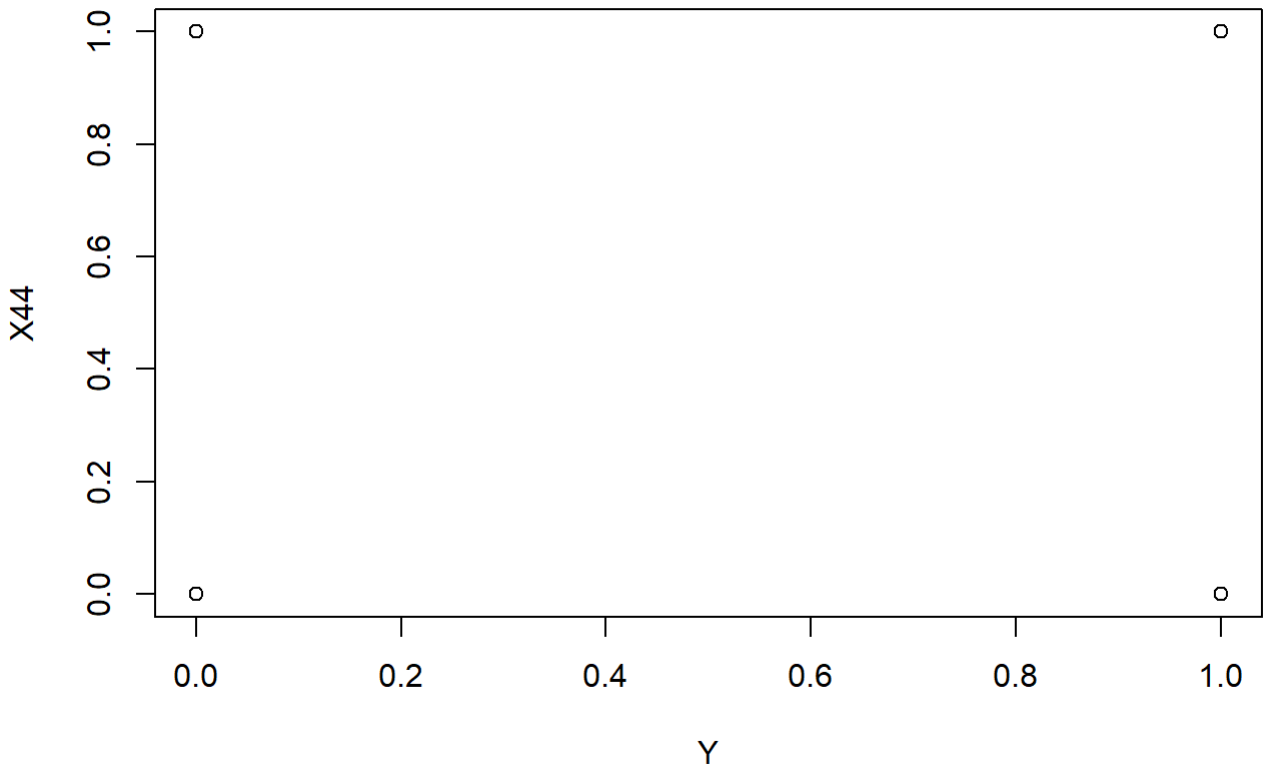

X39
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

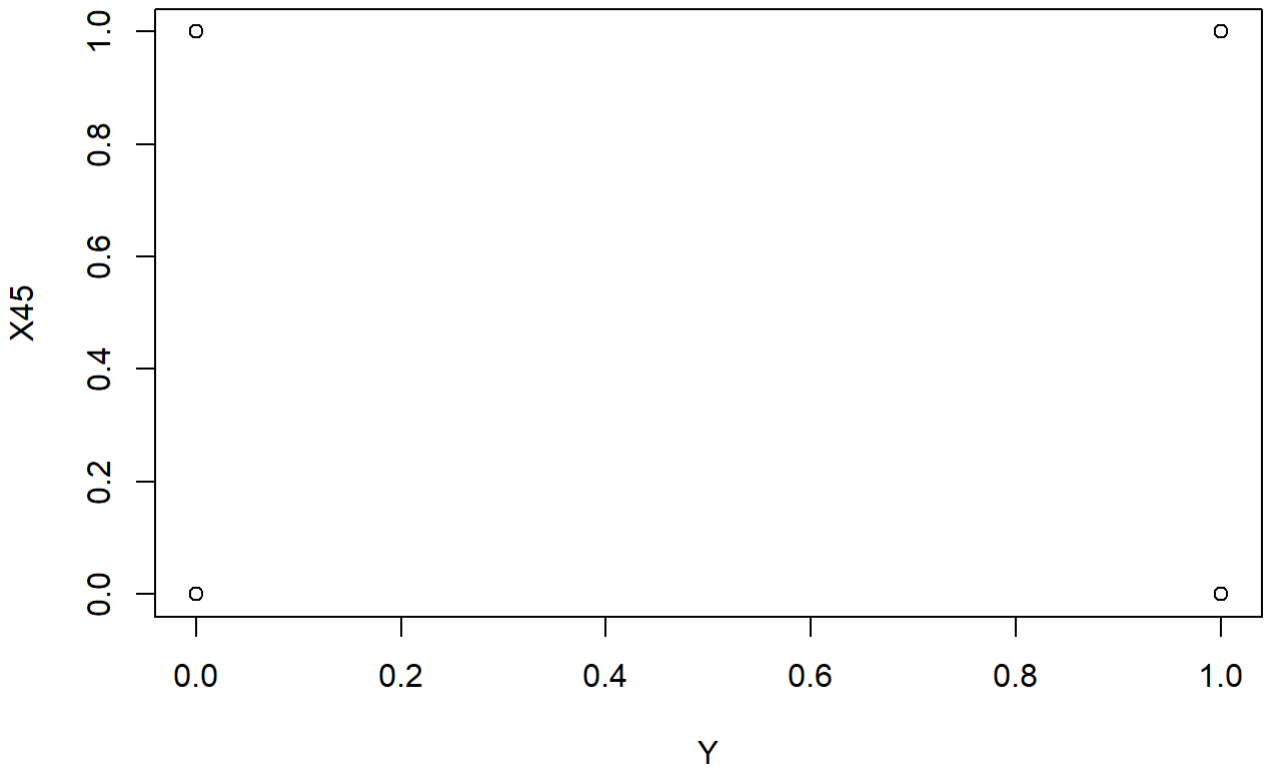

X40
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

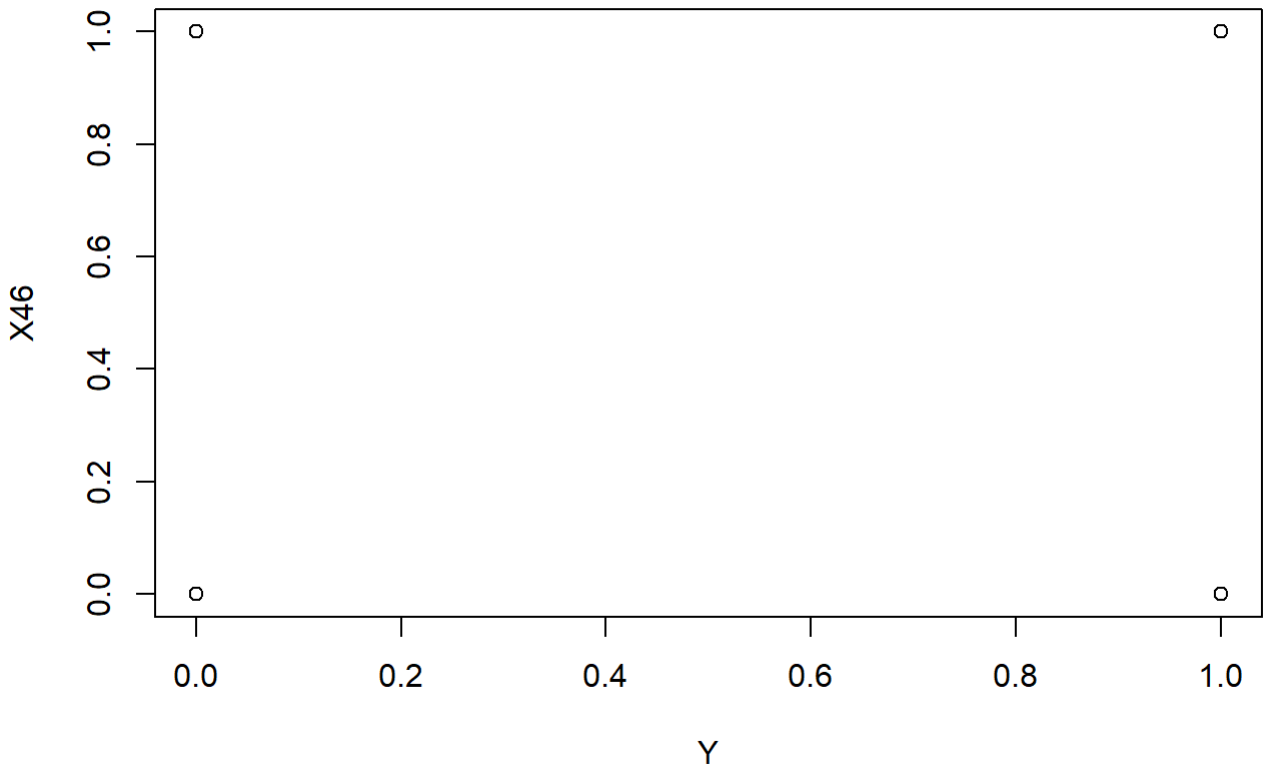

X41
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

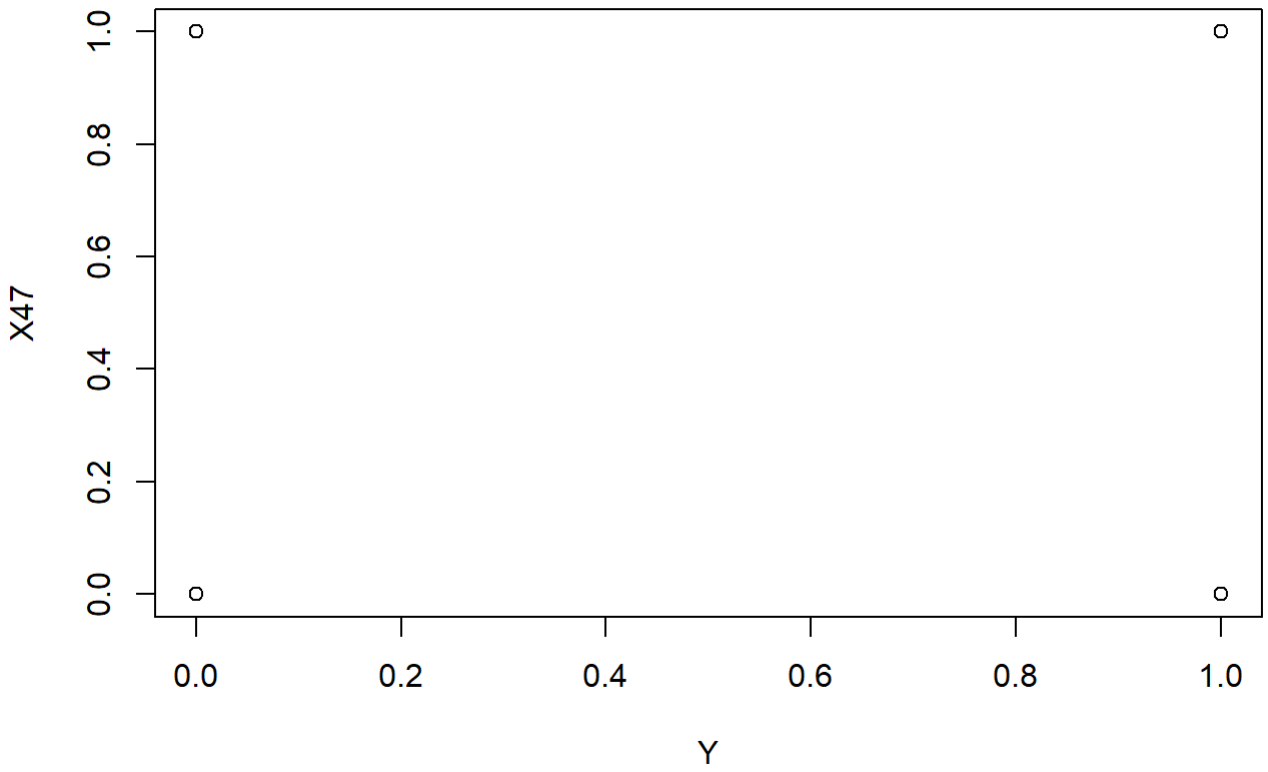

X42
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

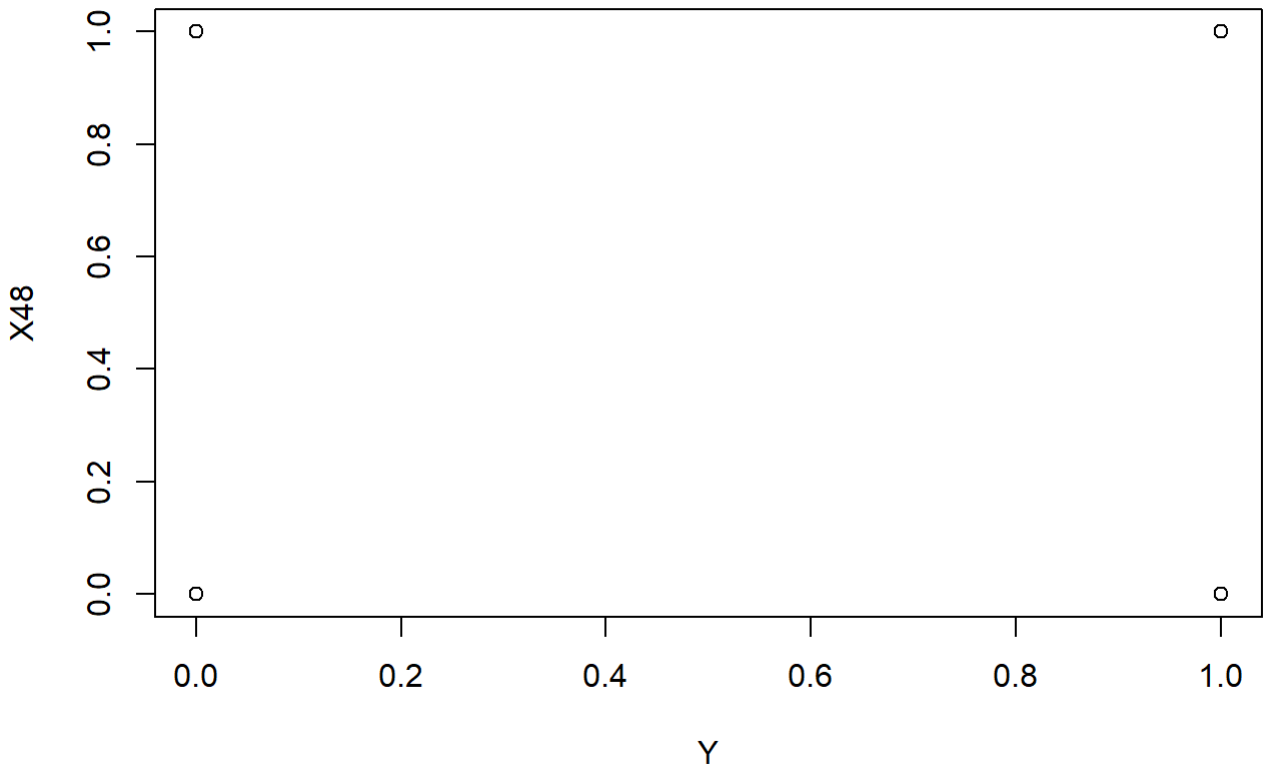

X43
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

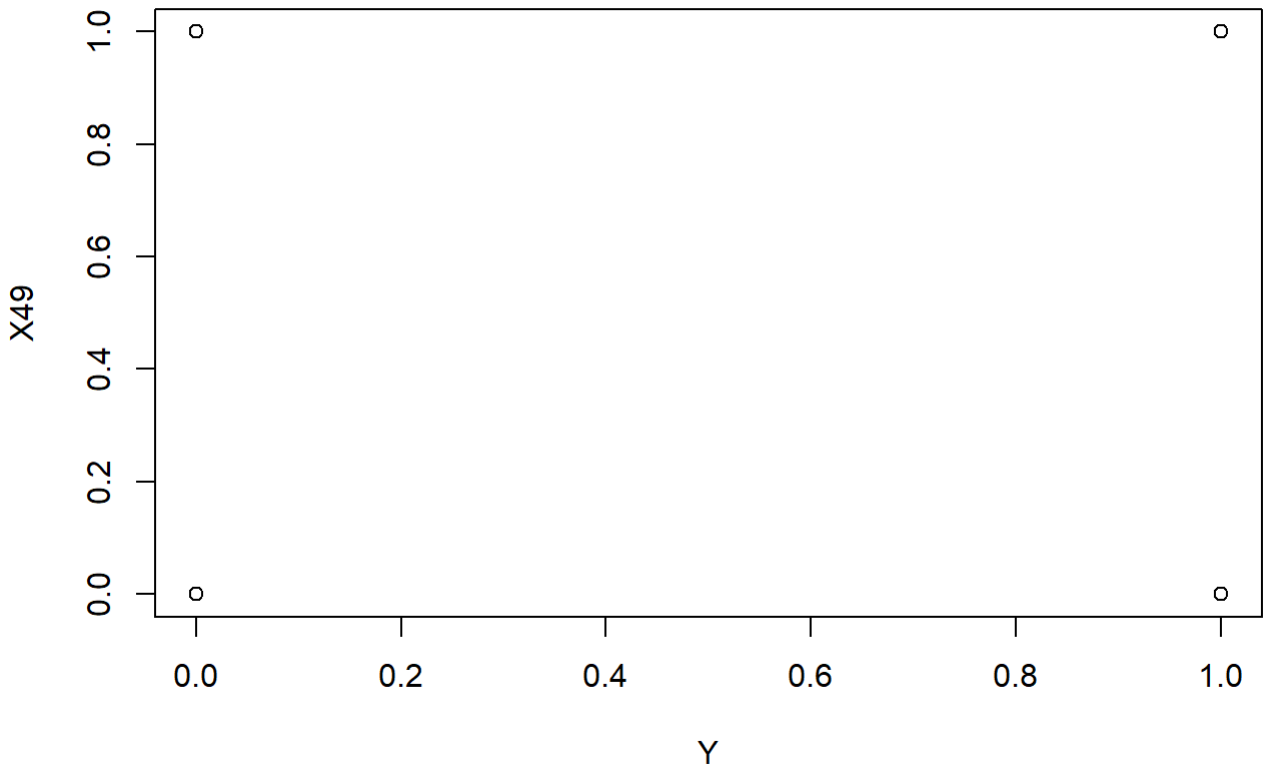

X44
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

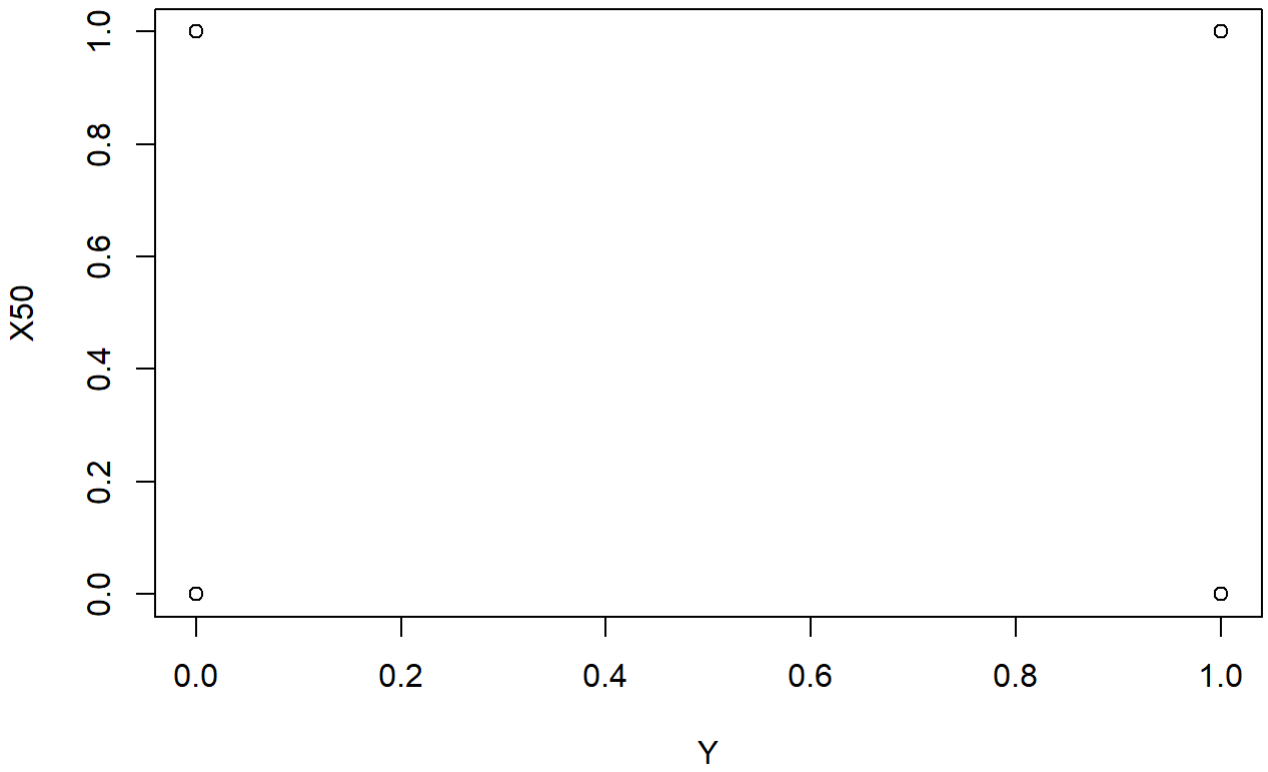

X45
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

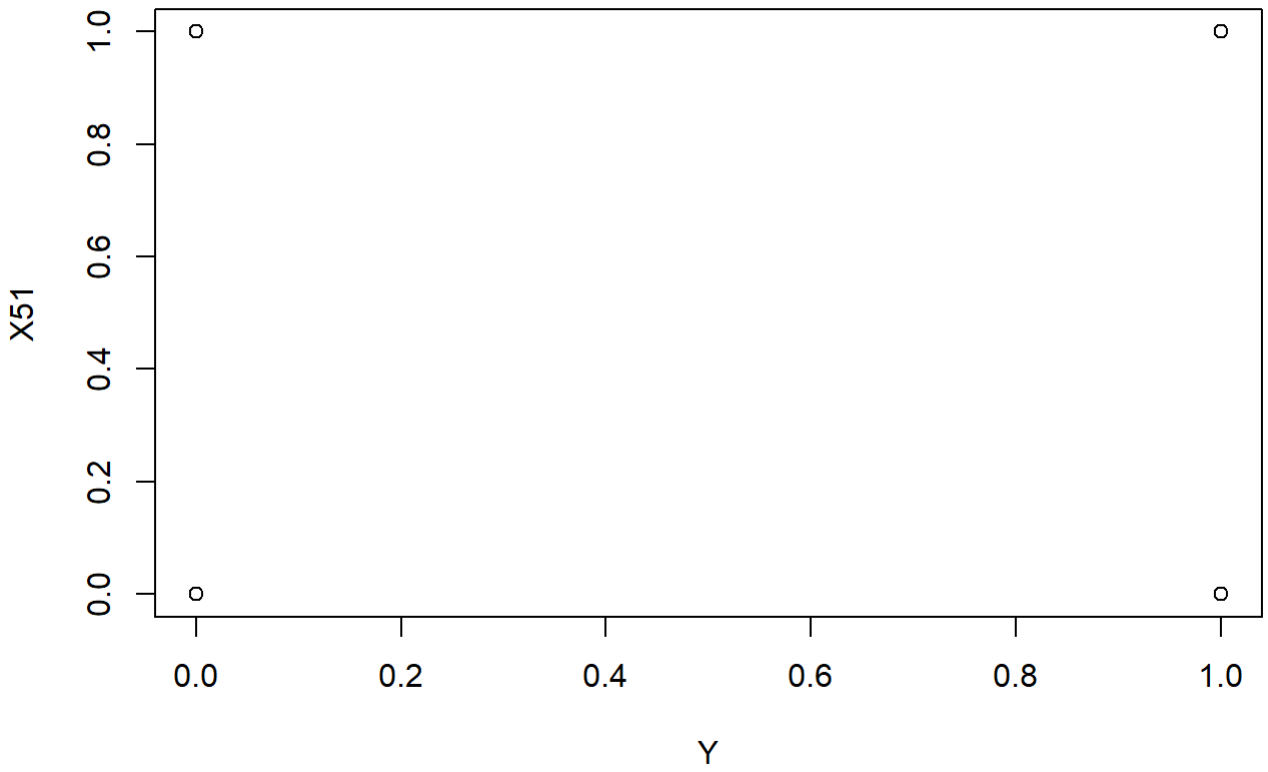

X46
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

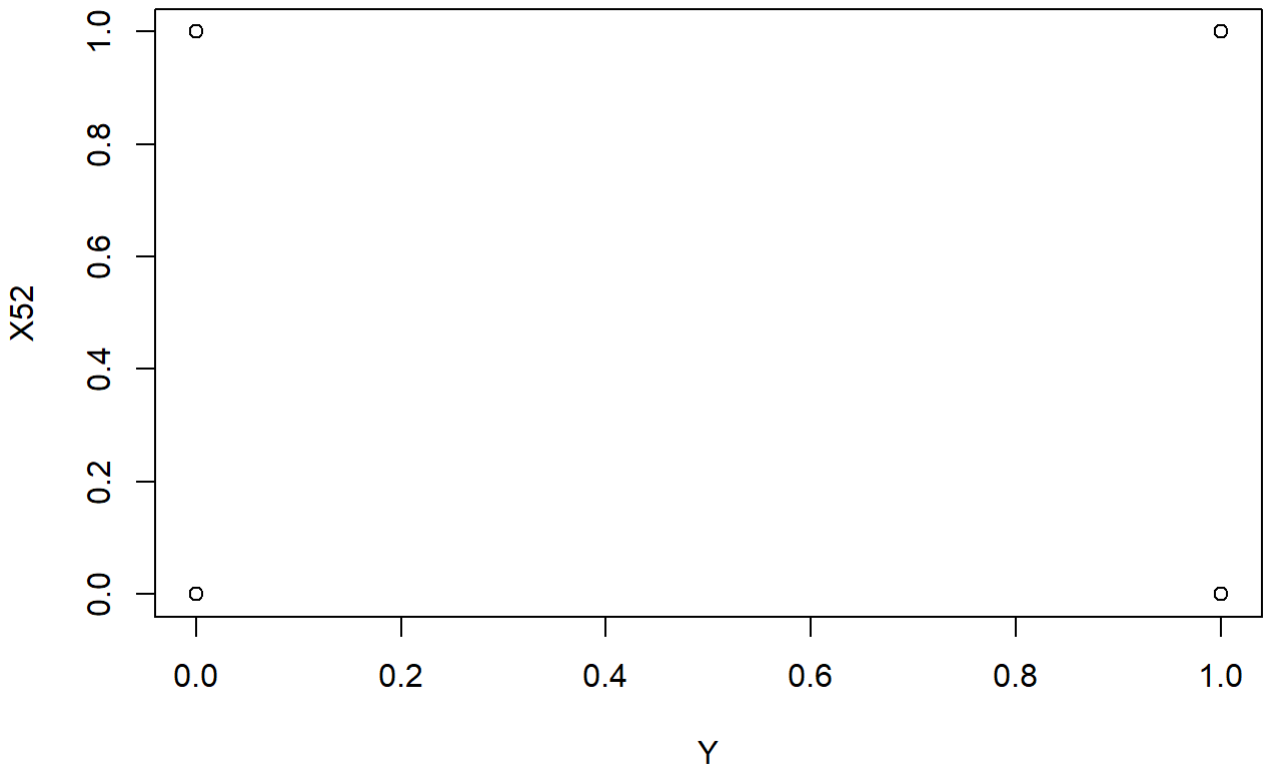

X47
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

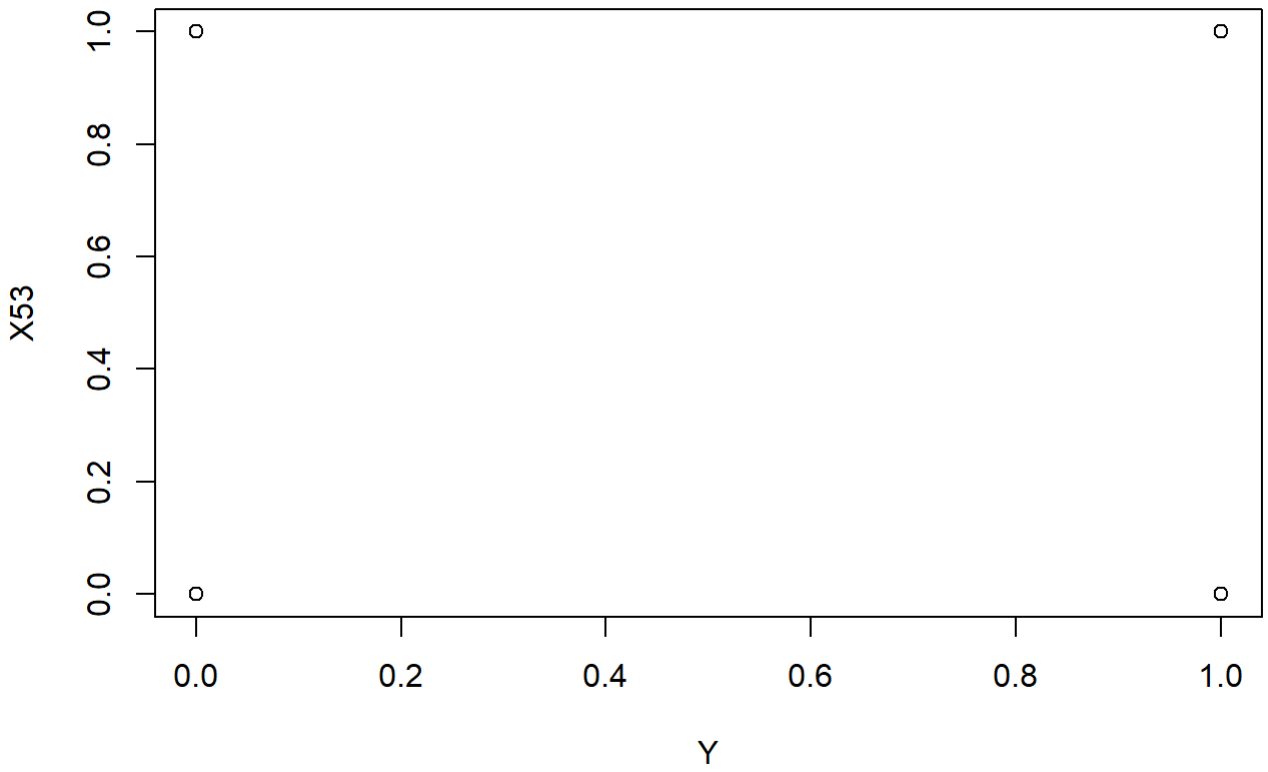

X48
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

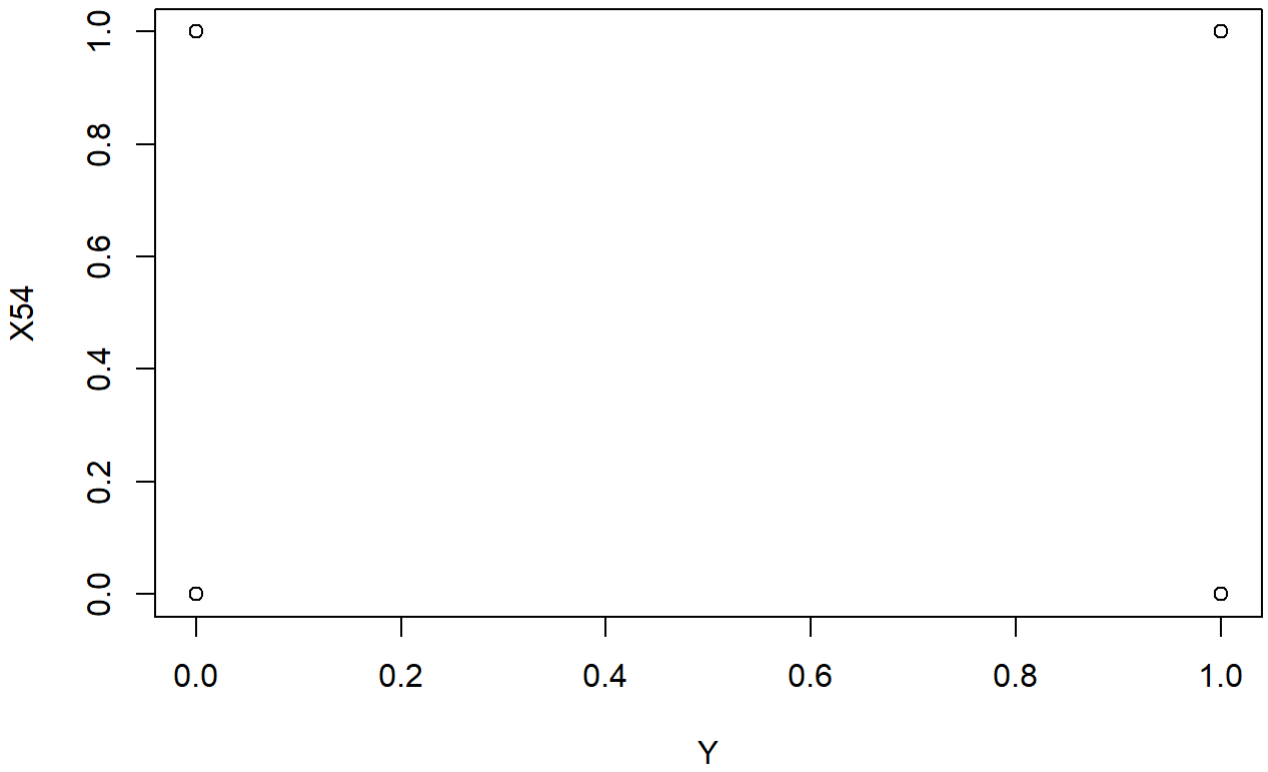

X49
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

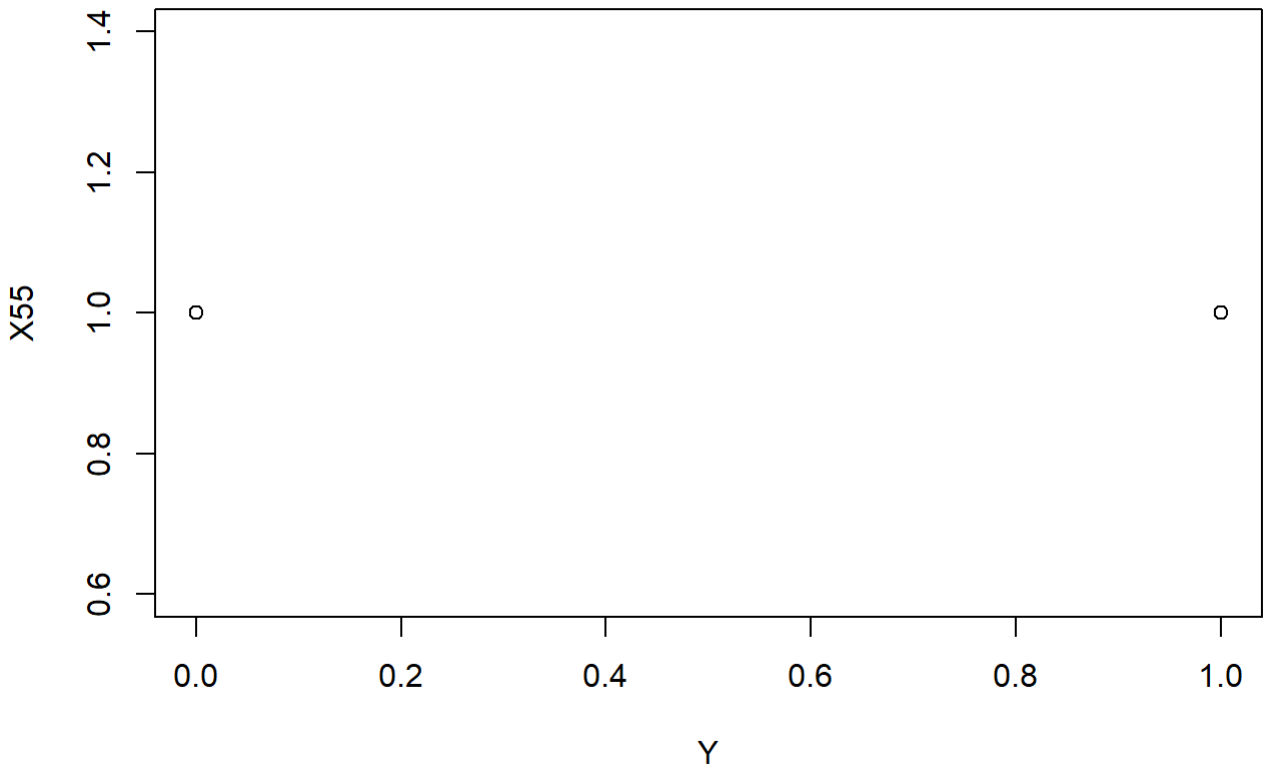

X50
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

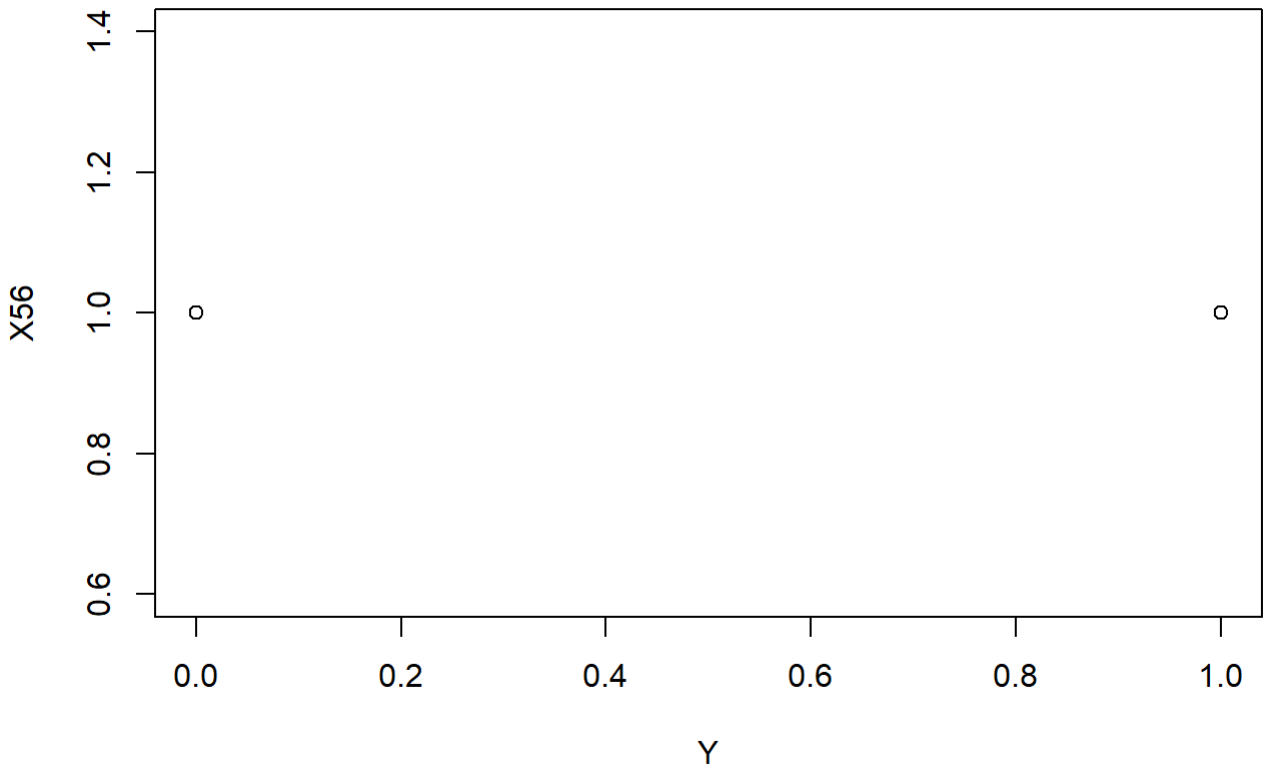

X51
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0

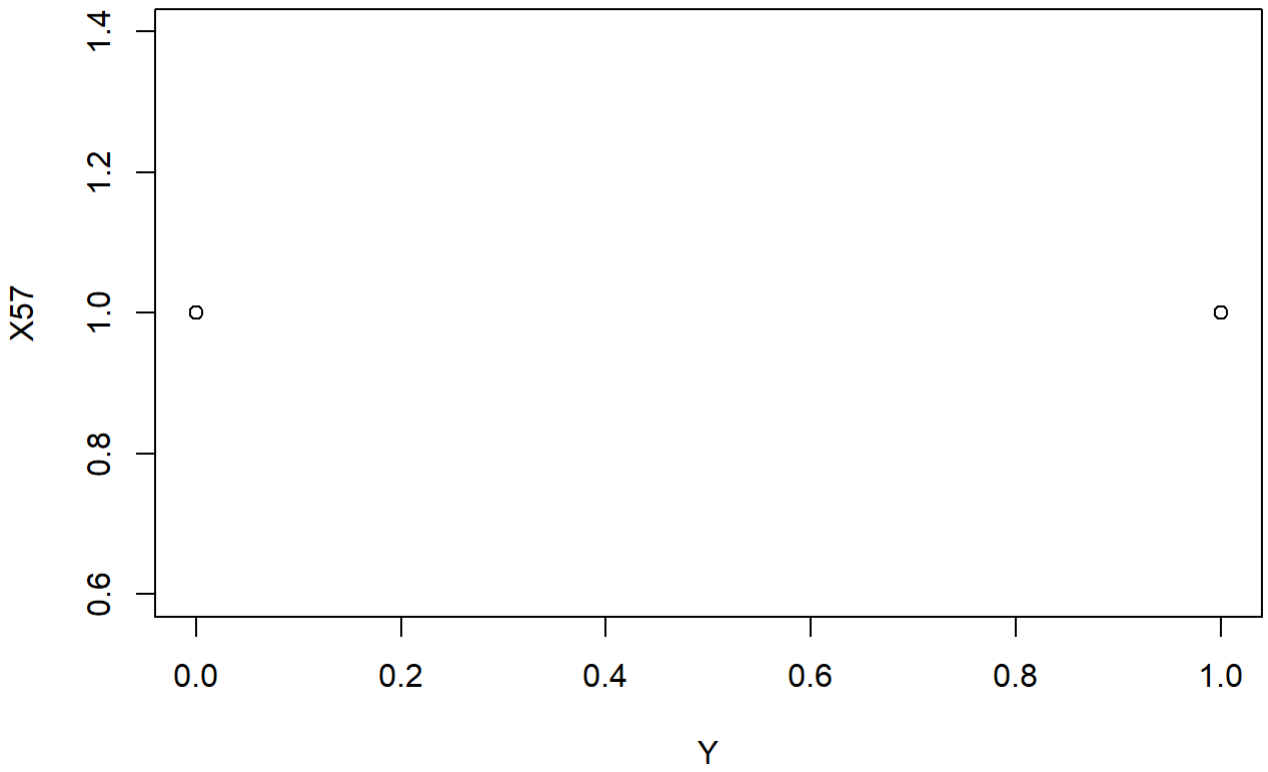

X52
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0


X53
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0


X54
0.0
0.2
0.4
0.6
0.8
1.0
Y
0.0
0.2
0.4
0.6
0.8
1.0


X55
1.4
1.2
1.0
0.8
0.6
0.0
0.2
0.4
0.6
0.8
1.0
Y


X56
1.4
1.2
1.0
0.8
0.6
0.0
0.2
0.4
0.6
0.8
1.0
Y




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

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).

Method 1: standardize Dataset

```
#(b)Fit a logistic regression model.

fit.glm.stand <- glm(Y ~., data = train.stand, family = "binomial")
```

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

```
summary(fit.glm.stand)
```

```
##
## Call:
## glm(formula = Y ~ ., family = "binomial", data = train.stand)
##
## 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)  -7.36294    1.76165  -4.180 2.92e-05 ***
## X1           -0.07047    0.08544  -0.825 0.409508
## X2           -0.21268    0.13656  -1.557 0.119379
## X3            0.02573    0.07472   0.344 0.730612
## X4            5.42487    2.63430   2.059 0.039464 *
## X5            0.41029    0.08897   4.611 4.00e-06 ***
## X6            0.08488    0.05780   1.469 0.141965
## X7            1.30763    0.19827   6.595 4.24e-11 ***
## X8            0.20112    0.07309   2.752 0.005931 **
## X9            0.21642    0.10039   2.156 0.031095 *
## X10           0.05737    0.06090   0.942 0.346145
## X11          -0.19561    0.07523  -2.600 0.009319 **
## X12          -0.03552    0.07302  -0.486 0.626655
## X13          -0.13217    0.11069  -1.194 0.232431
## X14          -0.00339    0.06296  -0.054 0.957058
## X15           0.31084    0.23239   1.338 0.181023
## X16           1.10038    0.16449   6.690 2.24e-11 ***
## X17           0.59641    0.13999   4.260 2.04e-05 ***
## X18          -0.02993    0.08391  -0.357 0.721327
## X19           0.15357    0.07781   1.974 0.048423 *
## X20           1.80199    0.50899   3.540 0.000400 ***
## X21           0.49973    0.08500   5.879 4.13e-09 ***
## X22           0.10473    0.15871   0.660 0.509332
## X23           1.17267    0.24101   4.866 1.14e-06 ***
## X24           0.09945    0.06169   1.612 0.106930
## X25          -3.27164    0.58150  -5.626 1.84e-08 ***
## X26          -0.44855    0.39100  -1.147 0.251312
## X27         -18.55268    3.80185  -4.880 1.06e-06 ***
## X28           0.24526    0.17081   1.436 0.151031
```

```
## X29          -2.42887    1.66214  -1.461 0.143936
## X30           0.01145    0.09666   0.118 0.905705
## X31          -0.08296    0.25709  -0.323 0.746941
## X32          -0.37441    0.95348  -0.393 0.694553
## X33          -0.46280    0.24665  -1.876 0.060610 .
## X34           0.85386    1.01167   0.844 0.398662
## X35          -0.61202    0.35339  -1.732 0.083302 .
## X36           0.07618    0.16958   0.449 0.653264
## X37          -0.26049    0.14890  -1.749 0.080214 .
## X38          -0.15147    0.12133  -1.248 0.211871
## X39          -0.02633    0.15297  -0.172 0.863349
## X40          -0.15745    0.17675  -0.891 0.373028
## X41         -18.56408   12.22870  -1.518 0.128996
## X42          -1.69535    0.58310  -2.907 0.003644 **
## X43          -0.45417    0.23919  -1.899 0.057599 .
## X44          -0.73394    0.35711  -2.055 0.039857 *
## X45          -0.88579    0.17727  -4.997 5.83e-07 ***
## X46          -1.08493    0.25513  -4.252 2.11e-05 ***
## X47          -0.64235    0.32519  -1.975 0.048234 *
## X48          -0.50262    0.38329  -1.311 0.189745
## X49          -0.20714    0.10111  -2.049 0.040502 *
## X50           0.04754    0.06007   0.791 0.428765
## X51          -0.06586    0.12898  -0.511 0.609646
## X52           0.24800    0.05813   4.266 1.99e-05 ***
## X53           1.01664    0.16220   6.268 3.66e-10 ***
## X54           0.59058    0.33572   1.759 0.078551 .
## X55          -0.56200    0.22976  -2.446 0.014445 *
## X56           1.08271    0.29373   3.686 0.000228 ***
## X57           0.61655    0.14118   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: 4121.0  on 3066  degrees of freedom
## Residual deviance: 1157.4  on 3009  degrees of freedom
## AIC: 1273.4
##
## Number of Fisher Scoring iterations: 13
```

```
lr.pred.train.stand <- predict(fit.glm.stand, train.stand, type = "response")
lr.pred.train.stand <- ifelse(lr.pred.train.stand > 0.5, 1, 0)
table(predict = lr.pred.train.stand, truth = train.stand$Y)
```

```
##        truth
## predict    0    1
##       0 1762  133
##       1   87 1085
```

```
train.error.stand = mean(lr.pred.train.stand != train.stand$Y)
train.error.stand
```

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

```
lr.pred.test.stand <- predict(fit.glm.stand, test.stand, type = "response")
lr.pred.test.stand <- ifelse(lr.pred.test.stand > 0.5, 1, 0)
table(predict = lr.pred.test.stand, truth = test.stand$Y)
```

```
##        truth
## predict   0   1
##       0 877  70
##       1  39 548
```

```
lr.test.error.stand = mean(lr.pred.test.stand != test.stand$Y)
lr.test.error.stand
```

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

```
#Interpretation

#The deviance residuals are a measure of model fit. This part of output shows the distribution of the deviance residuals for individual cases used in the model.
#Usually, we set the significant level at 0.05. For thoes whose p value is less or equal to 0.05, they are statistical signi ficant. From the result, we can tell there are statistical significant 26 features.
#The classification error on training set is 0.07173133 and on test set is 0.07105606.
```

Method 2: Log-transformed

```
fit.glm.log <- glm(Y ~., data = data.train.log, family = "binomial")
```

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

```
summary(fit.glm.log)
```

```
##
## Call:
## glm(formula = Y ~ ., family = "binomial", data = data.train.log)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.0831  -0.1646  -0.0010   0.0738   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 0.241325
## 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 0.326779
## 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 0.718349
## X10           0.20794    0.26664   0.780 0.435466
## X11          -1.73051    0.64790  -2.671 0.007563 **
## X12          -0.13019    0.21705  -0.600 0.548628
## X13          -1.47819    0.59699  -2.476 0.013284 *
## X14           0.49815    0.49244   1.012 0.311724
## 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           0.04966    0.17069   0.291 0.771075
## X20           4.74708    1.75988   2.697 0.006989 **
## X21           0.92793    0.20837   4.453 8.46e-06 ***
## X22           0.19783    0.59582   0.332 0.739860
## X23           3.39784    0.89163   3.811 0.000139 ***
## X24           1.27695    0.41124   3.105 0.001902 **
## X25          -3.97126    0.60152  -6.602 4.06e-11 ***
## X26          -0.43395    0.74531  -0.582 0.560401
## X27          -5.92242    1.42772  -4.148 3.35e-05 ***
## X28           1.27690    0.58913   2.167 0.030202 *
```

```
## X29          -5.52545    3.47037  -1.592 0.111344
## X30          -0.08833    0.47636  -0.185 0.852892
## X31          -1.17924    2.44793  -0.482 0.629997
## X32          -4.26131    4.43665  -0.960 0.336814
## X33          -1.44590    0.73243  -1.974 0.048368 *
## X34           0.86735    4.05419   0.214 0.830595
## X35          -2.60252    1.20495  -2.160 0.030784 *
## X36           0.44061    0.70994   0.621 0.534840
## X37          -1.55260    0.59961  -2.589 0.009615 **
## X38          -1.10219    1.36375  -0.808 0.418971
## X39           0.09940    0.80741   0.123 0.902025
## X40          -1.66152    1.14748  -1.448 0.147622
## X41         -45.30209   35.39198  -1.280 0.200542
## X42          -4.12654    1.24565  -3.313 0.000924 ***
## X43          -5.08561    1.94170  -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          -7.41904    4.88356  -1.519 0.128715
## X48          -2.02099    1.39842  -1.445 0.148405
## X49          -1.58851    0.79263  -2.004 0.045059 *
## X50          -0.01172    0.62116  -0.019 0.984945
## X51          -3.40426    2.64864  -1.285 0.198693
## 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 0.088513 .
## X56           0.09317    0.19497   0.478 0.632744
## 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: 4121.01  on 3066  degrees of freedom
## Residual deviance:  930.67  on 3009  degrees of freedom
## AIC: 1046.7
##
## Number of Fisher Scoring iterations: 12
```

```
lr.pred.train.log <- predict(fit.glm.log, data.train.log, type = "response")
lr.pred.train.log <- ifelse(lr.pred.train.log > 0.5, 1, 0)
table(predict = lr.pred.train.log, truth = data.train.log$Y)
```

```
##        truth
## predict    0    1
##       0 1766   94
##       1   83 1124
```

```
train.error.log = mean(lr.pred.train.log != data.train.log$Y)
train.error.log
```

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

```
lr.pred.test.log <- predict(fit.glm.log, data.test.log, type = "response")
lr.pred.test.log <- ifelse(lr.pred.test.log > 0.5, 1, 0)
table(predict = lr.pred.test.log, truth = data.test.log$Y)
```

```
##        truth
## predict   0   1
##       0 879  50
##       1  37 568
```

```
lr.test.error.log = mean(lr.pred.test.log != data.test.log$Y)
lr.test.error.log
```

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

```
#Interpretation:

#In this step, we use the transformed dataset to fit a logistic regression model. The output shows the distribution of the d
eviance residuals for individual cases used in the model. We set the significant level at 0.05. For thoes whoes p value is l
ess or equal to 0.05, they are statistical significant. From the result, we can tell there are statistical significant 30 fe
atures. What's more, the classification error on training set is 0.05771112 and on test set is 0.05671447.
```

Method 3: Discretize each feature using I(Xij > 0)

```
fit.glm.dis <- glm(Y ~., data = data.train.dis, family = "binomial")
summary(fit.glm.dis)
```

```
##
## Call:
## glm(formula = Y ~ ., family = "binomial", data = data.train.dis)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.6393  -0.1904  -0.0130   0.0600   3.9295
##
## Coefficients: (3 not defined because of singularities)
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.102414   0.189853 -11.074  < 2e-16 ***
## X1          -0.303292   0.289818  -1.046 0.295335
## X2          -0.378470   0.275804  -1.372 0.169989
## X3          -0.199095   0.212662  -0.936 0.349167
## X4           1.096282   0.824259   1.330 0.183511
## X5           1.268090   0.216147   5.867 4.44e-09 ***
## X6           0.251840   0.273000   0.922 0.356271
## X7           2.986605   0.386285   7.732 1.06e-14 ***
## X8           0.875957   0.316310   2.769 0.005618 **
## X9           0.228813   0.325213   0.704 0.481695
## X10          0.742343   0.238269   3.116 0.001836 **
## X11         -1.162239   0.334525  -3.474 0.000512 ***
## X12         -0.078381   0.194282  -0.403 0.686624
## X13         -1.161887   0.311432  -3.731 0.000191 ***
## X14          0.941421   0.452030   2.083 0.037283 *
## X15          2.006003   0.693342   2.893 0.003813 **
## X16          1.984579   0.226463   8.763  < 2e-16 ***
## X17          1.096497   0.319793   3.429 0.000606 ***
## X18         -0.857063   0.264975  -3.235 0.001219 **
## X19          0.006163   0.224878   0.027 0.978137
## X20          1.670892   0.554536   3.013 0.002586 **
## X21          0.834548   0.210275   3.969 7.22e-05 ***
## X22          0.811703   0.555363   1.462 0.143859
## X23          1.787937   0.392435   4.556 5.21e-06 ***
## X24          1.385796   0.343260   4.037 5.41e-05 ***
## X25         -3.611845   0.473164  -7.633 2.29e-14 ***
## X26         -0.640878   0.497465  -1.288 0.197646
## X27         -4.432733   0.740612  -5.985 2.16e-09 ***
## X28          1.981086   0.457457   4.331 1.49e-05 ***
```

```
## X29         -1.174992   0.668922  -1.757 0.078996 .
## X30         -0.183166   0.519469  -0.353 0.724387
## X31         -1.558298   1.033703  -1.507 0.131685
## X32         -2.211046   1.150863  -1.921 0.054706 .
## X33         -0.926369   0.562091  -1.648 0.099337 .
## X34          0.536636   1.068210   0.502 0.615408
## X35         -0.973451   0.565672  -1.721 0.085273 .
## X36          0.636619   0.417226   1.526 0.127050
## X37         -1.440826   0.348518  -4.134 3.56e-05 ***
## X38          1.173486   0.741369   1.583 0.113453
## X39          0.037749   0.413235   0.091 0.927214
## X40         -0.611572   0.557756  -1.096 0.272866
## X41         -5.823151   3.179731  -1.831 0.067051 .
## X42         -2.410825   0.508741  -4.739 2.15e-06 ***
## X43         -1.500599   0.638114  -2.352 0.018692 *
## X44         -1.301660   0.521227  -2.497 0.012514 *
## X45         -1.391117   0.235936  -5.896 3.72e-09 ***
## X46         -1.789877   0.363562  -4.923 8.52e-07 ***
## X47         -0.695873   1.130612  -0.615 0.538235
## X48         -1.512213   0.617515  -2.449 0.014331 *
## X49         -0.070815   0.275485  -0.257 0.797135
## X50          0.185424   0.196176   0.945 0.344561
## X51         -0.056805   0.409948  -0.139 0.889793
## X52          1.476322   0.186253   7.926 2.26e-15 ***
## X53          1.858618   0.250030   7.434 1.06e-13 ***
## X54         -0.794196   0.338409  -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: 4121.0  on 3066  degrees of freedom
## Residual deviance: 1014.6  on 3012  degrees of freedom
## AIC: 1124.6
##
## Number of Fisher Scoring iterations: 9
```

```
lr.pred.train.dis <- predict(fit.glm.dis, data.train.dis, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
lr.pred.train.dis <- ifelse(lr.pred.train.dis > 0.5, 1, 0)
table(predict = lr.pred.train.dis, truth = data.train.dis$Y)
```

```
##        truth
## predict    0    1
##       0 1779  105
##       1   70 1113
```

```
train.error.dis = mean(lr.pred.train.dis != data.train.dis$Y)
train.error.dis
```

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

```
lr.pred.test.dis <- predict(fit.glm.dis, data.test.dis, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
lr.pred.test.dis <- ifelse(lr.pred.test.dis > 0.5, 1, 0)
table(predict = lr.pred.test.dis, truth = data.test.dis$Y)
```

```
##        truth
## predict   0   1
##       0 859  67
##       1  57 551
```

```
lr.test.error.dis = mean(lr.pred.test.dis != data.test.dis$Y)
lr.test.error.dis
```

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

```
#Interpretation

#The deviance residuals are a measure of model fit. This part of output shows the distribution of the deviance residuals for individual cases used in the model.
#Usually, we set the significant level at 0.05. For thoes whoes p value is less or equal to 0.05, they are statistical significant.The classification error on training set is 0.05705902 and on test set is 0.08083442.
```

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)?

Method 1: standardize Dataset

```
#(linear discriminant analysis-lda )
fit.lda.stand <- lda(Y ~., data = train.stand)
summary(fit.lda.stand)
```

```
##         Length Class  Mode
## prior     2    -none- numeric
## counts    2    -none- numeric
## means   114    -none- numeric
## scaling  57    -none- numeric
## lev       2    -none- character
## svd       1    -none- numeric
## N         1    -none- numeric
## call      3    -none- call
## terms     3    terms  call
## xlevels   0    -none- list
```

```
lda.pred.train.stand <- predict(fit.lda.stand, train.stand)$class
table(predict = lda.pred.train.stand, truth = train.stand$Y)
```

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

```
lda.train.error.stand = mean(lda.pred.train.stand != train.stand$Y)
lda.train.error.stand
```

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

```
lda.pred.test.stand <- predict(fit.lda.stand, test.stand)$class
table(predict = lda.pred.test.stand, truth = test.stand$Y)
```

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

```
lda.test.error.stand = mean(lda.pred.test.stand != test.stand$Y)
lda.test.error.stand
```

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

```
#The classification error on training set is 0.1017281.
```

```
#(quadratic driscriminant analysis-qda )
fit.qda.stand <- qda(Y ~., data = train.stand)
summary(fit.qda.stand)
```

```
##         Length Class  Mode
## prior      2   -none- numeric
## counts     2   -none- numeric
## means    114   -none- numeric
## scaling 6498   -none- numeric
## ldet       2   -none- numeric
## lev        2   -none- character
## N          1   -none- numeric
## call       3   -none- call
## terms      3   terms  call
## xlevels    0   -none- list
```

```
qda.pred.train.stand <- predict(fit.qda.stand, train.stand)$class
table(predict = qda.pred.train.stand, truth = train.stand$Y)
```

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

```
qda.train.error.stand = mean(qda.pred.train.stand != train.stand$Y)
qda.train.error.stand
```

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

```
qda.pred.test.stand <- predict(fit.qda.stand, test.stand)$class
table(predict = qda.pred.test.stand, truth = test.stand$Y)
```

```
##        truth
## predict   0   1
##       0 673  25
##       1 243 593
```

```
qda.test.error.stand = mean(qda.pred.test.stand != test.stand$Y)
qda.test.error.stand
```

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

```
#The classification error on trainning set is 0.1747066.
```

Method 2: Log-transformed

```
library(MASS)

#(linear discriminant analysis-lda )
fit.lda.log <- lda(Y ~., data = data.train.log)
summary(fit.lda.log)
```

```
##         Length Class  Mode
## prior     2    -none- numeric
## counts    2    -none- numeric
## means   114    -none- numeric
## scaling  57    -none- numeric
## lev       2    -none- character
## svd       1    -none- numeric
## N         1    -none- numeric
## call      3    -none- call
## terms     3    terms  call
## xlevels   0    -none- list
```

```
lda.pred.train.log <- predict(fit.lda.log, data.train.log)$class
table(predict = lda.pred.train.log, truth = data.train.log$Y)
```

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

```
lda.train.error.log = mean(lda.pred.train.log != data.train.log$Y)
lda.train.error.log
```

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

```
#The classification error on training set is 0.06031953.

lda.pred.test.log <- predict(fit.lda.log, data.test.log)$class
table(predict = lda.pred.test.log, truth = data.test.log$Y)
```

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

```
lda.test.error.log = mean(lda.pred.test.log != data.test.log$Y)
lda.test.error.log
```

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

```
#The classification error on testing set is 0.06518905.
```

```
#(quadratic discriminant analysis-qda )
fit.qda.log <- qda(Y ~., data = data.train.log)
summary(fit.qda.log)
```

```
##         Length Class  Mode
## prior      2   -none- numeric
## counts     2   -none- numeric
## means    114   -none- numeric
## scaling 6498   -none- numeric
## ldet       2   -none- numeric
## lev        2   -none- character
## N          1   -none- numeric
## call       3   -none- call
## terms      3   terms  call
## xlevels    0   -none- list
```

```
qda.pred.train.log <- predict(fit.qda.log, data.train.log)$class
table(predict = qda.pred.train.log, truth = data.train.log$Y)
```

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

```
qda.train.error.log = mean(qda.pred.train.log != data.train.log$Y)
qda.train.error.log
```

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

```
#The classification error on training set is 0.1587871.

qda.pred.test.log <- predict(fit.qda.log, data.test.log)$class
table(predict = qda.pred.test.log, truth = data.test.log$Y)
```

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

```
qda.test.error.log = mean(qda.pred.test.log != data.test.log$Y)
qda.test.error.log
```

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

```
#The classification error on testing set is 0.1571056.
```

d. Apply linear and nonlinear support vector machine classifiers to each version of the data.

Method 1: standardize Dataset

```
# linear SVM
fit.linsvm.stand <- svm(as.factor(Y) ~., data = train.stand, kernel = "linear",
                        scale = F)
summary(fit.linsvm.stand)
```

```
##
## Call:
## svm(formula = as.factor(Y) ~ ., data = train.stand, kernel = "linear",
##     scale = F)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  621
##
##  ( 315 306 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
linsvm.pred.train.stand <- predict(fit.linsvm.stand, train.stand)
table(predict = linsvm.pred.train.stand, truth = train.stand$Y)
```

```
##        truth
## predict    0    1
##       0 1768  118
##       1   81 1100
```

```
linsvm.train.error.stand = mean(linsvm.pred.train.stand != train.stand$Y)
linsvm.train.error.stand
```

```
## [1] 0.06488425
```

```
#The classification error on training set is 0.06488425.

linsvm.pred.test.stand <- predict(fit.linsvm.stand, test.stand)
table(predict = linsvm.pred.test.stand, truth = test.stand$Y)
```

```
##        truth
## predict   0   1
##       0 873  67
##       1  43 551
```

```
linsvm.test.error.stand = mean(linsvm.pred.test.stand != test.stand$Y)
linsvm.test.error.stand
```

```
## [1] 0.07170795
```

```
#The classification error on test set is 0.07170795.
```

```
# nonlinear SVM
fit.nlsvm.stand <- svm(as.factor(Y) ~., data = train.stand, kernel = "radial", scale = F)
summary(fit.nlsvm.stand)
```

```
##
## Call:
## svm(formula = as.factor(Y) ~ ., data = train.stand, kernel = "radial",
##     scale = F)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  926
##
##  ( 492 434 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
nlsvm.pred.train.stand <- predict(fit.nlsvm.stand, train.stand)
table(predict = nlsvm.pred.train.stand, truth = train.stand$Y)
```

```
##        truth
## predict    0    1
##       0 1786   95
##       1   63 1123
```

```
nlsvm.train.error.stand = mean(nlsvm.pred.train.stand != train.stand$Y)
nlsvm.train.error.stand
```

```
## [1] 0.05151614
```

```
#The classification error on training set is 0.05151614.

nlsvm.pred.test.stand <- predict(fit.nlsvm.stand, test.stand)
table(predict = nlsvm.pred.test.stand, truth = test.stand$Y)
```

```
##        truth
## predict   0   1
##       0 875  58
##       1  41 560
```

```
nlsvm.test.error.stand = mean(nlsvm.pred.test.stand != test.stand$Y)
nlsvm.test.error.stand
```

```
## [1] 0.06453716
```

```
#The classification error on test set is 0.06453716.
```

Method 2: Log-transformed

```
library(e1071)

# linear SVM
fit.linsvm.log <- svm(as.factor(Y) ~., data = data.train.log, kernel = "linear",
                      scale = F)
summary(fit.linsvm.log)
```

```
##
## Call:
## svm(formula = as.factor(Y) ~ ., data = data.train.log, kernel = "linear",
##     scale = F)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  529
##
##  ( 273 256 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
linsvm.pred.train.log <- predict(fit.linsvm.log, data.train.log)
table(predict = linsvm.pred.train.log, truth = data.train.log$Y)
```

```
##        truth
## predict    0    1
##       0 1772  102
##       1   77 1116
```

```
linsvm.train.error.log = mean(linsvm.pred.train.log != data.train.log$Y)
linsvm.train.error.log
```

```
## [1] 0.05836322
```

```
#The classification error on training set is 0.05836322.

linsvm.pred.test.log <- predict(fit.linsvm.log, data.test.log)
table(predict = linsvm.pred.test.log, truth = data.test.log$Y)
```

```
##        truth
## predict   0   1
##       0 880  50
##       1  36 568
```

```
linsvm.test.error.log = mean(linsvm.pred.test.log != data.test.log$Y)
linsvm.test.error.log
```

```
## [1] 0.05606258
```

```
#The classification error on testing set is 0.05606258.
```

```
# nonlinear SVM
fit.nlsvm.log <- svm(as.factor(Y) ~., data = data.train.log, kernel = "radial", scale = F)
summary(fit.nlsvm.log)
```

```
##
## Call:
## svm(formula = as.factor(Y) ~ ., data = data.train.log, kernel = "radial",
##     scale = F)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  877
##
##  ( 442 435 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
nlsvm.pred.train.log <- predict(fit.nlsvm.log, data.train.log)
table(predict = nlsvm.pred.train.log, truth = data.train.log$Y)
```

```
##        truth
## predict    0    1
##       0 1776  111
##       1   73 1107
```

```
nlsvm.train.error.log = mean(nlsvm.pred.train.log != data.train.log$Y)
nlsvm.train.error.log
```

```
## [1] 0.05999348
```

```
#The classification error on training set is 0.05999348.

nlsvm.pred.test.log <- predict(fit.nlsvm.log, data.test.log)
table(predict = nlsvm.pred.test.log, truth = data.test.log$Y)
```

```
##        truth
## predict   0   1
##       0 877  48
##       1  39 570
```

```
nlsvm.test.error.log = mean(nlsvm.pred.test.log != data.test.log$Y)
nlsvm.test.error.log
```

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

```
#The classification error on testing set is 0.05671447.
```

Method 3: Discretize each feature using I(Xij > 0)

```
library(e1071)



fit.linsvm.dis <- svm(as.factor(Y) ~., data = data.train.dis, kernel = "linear",
                      scale = F)
summary(fit.linsvm.dis)
```

```
##
## Call:
## svm(formula = as.factor(Y) ~ ., data = data.train.dis, kernel = "linear",
##     scale = F)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  560
##
##  ( 280 280 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
# training set
linsvm.pred.train.dis <- predict(fit.linsvm.dis, data.train.dis)
table(predict = linsvm.pred.train.dis, truth = data.train.dis$Y)
```

```
##        truth
## predict    0    1
##       0 1780  116
##       1   69 1102
```

```
linsvm.train.error.dis = mean(linsvm.pred.train.dis != data.train.dis$Y)
linsvm.train.error.dis
```

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

```
# The classification error on training set is 0.06031953.

# testing set
linsvm.pred.test.dis <- predict(fit.linsvm.dis, data.test.dis)
table(predict = linsvm.pred.test.dis, truth = data.test.dis$Y)
```

```
##        truth
## predict   0   1
##       0 865  63
##       1  51 555
```

```
linsvm.test.error.dis = mean(linsvm.pred.test.dis!= data.test.dis$Y)
linsvm.test.error.dis
```

```
## [1] 0.07431551
```

```
# The classification error on testing set is 0.07431551.
```

nonlinear SVM

```
fit.nlsvm.dis <- svm(as.factor(Y) ~., data = data.train.dis, kernel = "radial", scale = F)
summary(fit.nlsvm.dis)
```

```
##
## Call:
## svm(formula = as.factor(Y) ~ ., data = data.train.dis, kernel = "radial",
##     scale = F)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  814
##
##  ( 411 403 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
# training set
nlsvm.pred.train.dis <- predict(fit.nlsvm.dis, data.train.dis)
table(predict = nlsvm.pred.train.dis, truth = data.train.dis$Y)
```

```
##        truth
## predict    0    1
##       0 1784  124
##       1   65 1094
```

```
nlsvm.train.error.dis = mean(nlsvm.pred.train.dis != data.train.dis$Y)
nlsvm.train.error.dis
```

```
## [1] 0.06162374
```

```
# The classification error on training set is 0.06162374.

# testing set
nlsvm.pred.test.dis <- predict(fit.nlsvm.dis, data.test.dis)
table(predict = nlsvm.pred.test.dis, truth = data.test.dis$Y)
```

```
##        truth
## predict   0   1
##       0 874  74
##       1  42 544
```

```
nlsvm.test.error.dis = mean(nlsvm.pred.test.dis != data.test.dis$Y)
nlsvm.test.error.dis
```

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

```
# The classification error on testing set is 0.0756193.
```

Find the method with the best performance

```
# Finally, use either a single method with properly chosen tuning parameter or a combination of several methods to design a
 classifier with test error rate as small as possible.
# We choose considering tuning parameter C for SVM with gaussian kernel

# Using standardized data

tune.gaussian.stand <- tune(svm, as.factor(Y) ~.,
                            data = train.stand, kernel = "radial", scale = F,
                            ranges = list(cost=c(10,50,100)))
summary(tune.gaussian.stand)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 0.05054395
##
## - Detailed performance results:
##   cost      error dispersion
## 1   10 0.05543420 0.01410697
## 2   50 0.05184901 0.01423665
## 3  100 0.05054395 0.01254041
```

```
bestmod.gaussian.stand <- tune.gaussian.stand$best.model

bestmod.pred.train.stand <- predict(bestmod.gaussian.stand, train.stand)
table(predict = bestmod.pred.train.stand, truth = train.stand$Y)
```

```
##        truth
## predict    0    1
##       0 1845   28
##       1    4 1190
```

```
bestmod.train.error.stand = mean(bestmod.pred.train.stand != train.stand$Y)
bestmod.train.error.stand
```

```
## [1] 0.01043365
```

```
bestmod.pred.test.stand <- predict(bestmod.gaussian.stand, test.stand)
table(predict = bestmod.pred.test.stand, truth = test.stand$Y)
```

```
##        truth
## predict   0   1
##       0 884  50
##       1  32 568
```

```
bestmod.test.error.stand = mean(bestmod.pred.test.stand != test.stand$Y)
bestmod.test.error.stand
```

```
## [1] 0.05345502
```

```
# Using log transformed data
tune.gaussian.log <- tune(svm, as.factor(Y) ~.,
                          data = data.train.log, kernel = "radial", scale = F,
                          ranges = list(cost=c(10,50,100)))
summary(tune.gaussian.log)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   100
##
## - best performance: 0.04629665
##
## - Detailed performance results:
##   cost      error  dispersion
## 1   10 0.05476890 0.011742865
## 2   50 0.05052905 0.010057610
## 3  100 0.04629665 0.006645477
```

```
bestmod.gaussian.log <- tune.gaussian.log$best.model

bestmod.pred.train.log <- predict(bestmod.gaussian.log, data.train.log)
table(predict = bestmod.pred.train.log, truth = data.train.log$Y)
```

```
##        truth
## predict    0    1
##       0 1817   65
##       1   32 1153
```

```
bestmod.train.error.log = mean(bestmod.pred.train.log != data.train.log$Y)
bestmod.train.error.log
```

```
## [1] 0.031627
```

```
bestmod.pred.test.log <- predict(bestmod.gaussian.log, data.test.log)
table(predict = bestmod.pred.test.log, truth = data.test.log$Y)
```

```
##        truth
## predict   0   1
##       0 892  36
##       1  24 582
```

```
bestmod.test.error.log = mean(bestmod.pred.test.log != data.test.log$Y)
bestmod.test.error.log
```

```
## [1] 0.03911343
```

```
# using discritized data

tune.gaussian.dis <- tune(svm, as.factor(Y) ~.,
                          data = data.train.dis, kernel = "radial", scale = F,
                          ranges = list(cost=c(10,50,100)))
summary(tune.gaussian.dis)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    50
##
## - best performance: 0.05087288
##
## - Detailed performance results:
##   cost      error dispersion
## 1   10 0.06229163 0.01782742
## 2   50 0.05087288 0.01255080
## 3  100 0.05250367 0.01312132
```

```
bestmod.gaussian.dis <- tune.gaussian.dis$best.model

bestmod.pred.train.dis <- predict(bestmod.gaussian.dis, data.train.dis)
table(predict = bestmod.pred.train.dis, truth = data.train.dis$Y)
```

```
##        truth
## predict    0    1
##       0 1814   65
##       1   35 1153
```

```
bestmod.train.error.dis = mean(bestmod.pred.train.dis != data.train.dis$Y)
bestmod.train.error.dis
```

```
## [1] 0.03260515
```

```
bestmod.pred.test.dis <- predict(bestmod.gaussian.dis, data.test.dis)
table(predict = bestmod.pred.test.dis, truth = data.test.dis$Y)
```

```
##        truth
## predict   0   1
##       0 881  50
##       1  35 568
```

```
bestmod.test.error.dis = mean(bestmod.pred.test.dis != data.test.dis$Y)
bestmod.test.error.dis
```

```
## [1] 0.05541069
```

Summary error table

```
Error_table= matrix(data = NA,ncol = 3,nrow = 16)
colnames(Error_table) = c("combination","training error", "testing error")

Error_table[,1]=c("log regression-standardized",
                  "lda-standardized",
                  "qda-standardized",
                  "linear svm-standardized",
                  "nonlinear svm-standardize",
                  "tuned-standardized",
                  "log regression-log",
                  "lda-log",
                  "qda-log",
                  "linear svm-log",
                  "nonlinear svm-log",
                   "tuned-log",
                  "log regression-discretized",
                  "linear svm-discretized",
                  "nonlinear svm-discretized",
                  "tuned-discretized")

Error_table[,2] = c(train.error.stand,
                lda.train.error.stand,
                qda.train.error.stand,
                linsvm.train.error.stand,
                nlsvm.train.error.stand,
                bestmod.train.error.stand,
                train.error.log,
                lda.train.error.log,
                qda.train.error.log,
                linsvm.train.error.log,
                nlsvm.train.error.log,
                 bestmod.train.error.log,
                train.error.dis,
                linsvm.train.error.dis,
                nlsvm.train.error.dis,
                bestmod.train.error.dis
               )

Error_table[,3] = c(lr.test.error.stand,
```

```
            lda.test.error.stand,
            qda.test.error.stand,
            linsvm.test.error.stand,
            nlsvm.test.error.stand,
            bestmod.test.error.stand,
            lr.test.error.log,
            lda.test.error.log,
            qda.test.error.log,
            linsvm.test.error.log,
            nlsvm.test.error.log,
            bestmod.test.error.log,
            lr.test.error.dis,
            linsvm.test.error.dis,
            nlsvm.test.error.dis,
            bestmod.test.error.dis)
Error_table
```

```
##       combination                   training error       testing error
##  [1,] "log regression-standardized" "0.071731333550701"  "0.0710560625814863"
##  [2,] "lda-standardized"            "0.10172807303554"   "0.102998696219035"
##  [3,] "qda-standardized"            "0.178676230844473"  "0.17470664928292"
##  [4,] "linear svm-standardized"     "0.0648842517117705" "0.0717079530638853"
##  [5,] "nonlinear svm-standardize"   "0.0515161395500489" "0.0645371577574967"
##  [6,] "tuned-standardized"          "0.0104336485164656" "0.0534550195567145"
##  [7,] "log regression-log"          "0.0577111183567004" "0.0567144719687093"
##  [8,] "lda-log"                     "0.0603195304858168" "0.0651890482398957"
##  [9,] "qda-log"                     "0.158787088359961"  "0.157105606258149"
## [10,] "linear svm-log"              "0.0583632213889795" "0.0560625814863103"
## [11,] "nonlinear svm-log"           "0.0599934789696772" "0.0567144719687093"
## [12,] "tuned-log"                   "0.0316269970655364" "0.0391134289439374"
## [13,] "log regression-discretized"  "0.0570590153244213" "0.0808344198174707"
## [14,] "linear svm-discretized"      "0.0603195304858168" "0.0743155149934811"
## [15,] "nonlinear svm-discretized"   "0.061623736550375"  "0.075619295958279"
## [16,] "tuned-discretized"           "0.032605151613955"  "0.0554106910039113"
```