Statistical Learning: Final Project

Mike Baietto, Dan Malee, Gerard Martinez, and Tony Galvan

West

10/29/2018

Final Project

Description

The goal of this project is to identify and explore interesting aspects of data in a real-world context, provide extensive explanations about each step, and report on our analysis of the results.

This write-up will consist of the following sections:

1. Exploratory Data Analysis (EDA)
2. Preprocessing
3. Initial Models
4. Model Optimization
5. Figures & Conclusion

Read In spambase.data file

```
#Start Fresh
rm(list=ls())
```

```
#Load libraries
library(tidyverse)
library(plm)
library(tidyr)
library(ggplot2)
library(scales)
library(readstata13)
library(corrplot)
library(reshape)
library(dplyr)
library(corrplot)
library(gridExtra)
library(rpart)
library(partykit)
library(caret)
library(mlbench)
library(tidyverse)
library(rminer)
library(randomForest)
library(reshape2)
library(MASS)
library(e1071)
library(wsrf)
library(kernlab)
```

EXPLORATORY DATA ANALYSIS (EDA) SECTION

LOAD THE DATA

```
# set working directory and import data
#setwd(dirname(rstudioapi::getSourceEditorContext()$path))
spamdata <- read.csv("spambase.data", header=FALSE)
dim(spamdata)
```

```
## [1] 4601   58
```

The “spambase” data set consists of 4601 observations with 58 variables. Let’s take a closer look at the data.

```
glimpse(spamdata)
```

```
## Observations: 4,601
## Variables: 58
## $ V1  <dbl> 0.00, 0.21, 0.06, 0.00, 0.00, 0.00, 0.00, 0.00, 0.15, 0.06...
## $ V2  <dbl> 0.64, 0.28, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.12...
## $ V3  <dbl> 0.64, 0.50, 0.71, 0.00, 0.00, 0.00, 0.00, 0.00, 0.46, 0.77...
## $ V4  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V5  <dbl> 0.32, 0.14, 1.23, 0.63, 0.63, 1.85, 1.92, 1.88, 0.61, 0.19...
## $ V6  <dbl> 0.00, 0.28, 0.19, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.32...
## $ V7  <dbl> 0.00, 0.21, 0.19, 0.31, 0.31, 0.00, 0.00, 0.00, 0.30, 0.38...
## $ V8  <dbl> 0.00, 0.07, 0.12, 0.63, 0.63, 1.85, 0.00, 1.88, 0.00, 0.00...
## $ V9  <dbl> 0.00, 0.00, 0.64, 0.31, 0.31, 0.00, 0.00, 0.00, 0.92, 0.06...
## $ V10 <dbl> 0.00, 0.94, 0.25, 0.63, 0.63, 0.00, 0.64, 0.00, 0.76, 0.00...
## $ V11 <dbl> 0.00, 0.21, 0.38, 0.31, 0.31, 0.00, 0.96, 0.00, 0.76, 0.00...
## $ V12 <dbl> 0.64, 0.79, 0.45, 0.31, 0.31, 0.00, 1.28, 0.00, 0.92, 0.64...
## $ V13 <dbl> 0.00, 0.65, 0.12, 0.31, 0.31, 0.00, 0.00, 0.00, 0.00, 0.25...
## $ V14 <dbl> 0.00, 0.21, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V15 <dbl> 0.00, 0.14, 1.75, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.12...
## $ V16 <dbl> 0.32, 0.14, 0.06, 0.31, 0.31, 0.00, 0.96, 0.00, 0.00, 0.00...
## $ V17 <dbl> 0.00, 0.07, 0.06, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V18 <dbl> 1.29, 0.28, 1.03, 0.00, 0.00, 0.00, 0.32, 0.00, 0.15, 0.12...
## $ V19 <dbl> 1.93, 3.47, 1.36, 3.18, 3.18, 0.00, 3.85, 0.00, 1.23, 1.67...
## $ V20 <dbl> 0.00, 0.00, 0.32, 0.00, 0.00, 0.00, 0.00, 0.00, 3.53, 0.06...
## $ V21 <dbl> 0.96, 1.59, 0.51, 0.31, 0.31, 0.00, 0.64, 0.00, 2.00, 0.71...
## $ V22 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V23 <dbl> 0.00, 0.43, 1.16, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.19...
## $ V24 <dbl> 0.00, 0.43, 0.06, 0.00, 0.00, 0.00, 0.00, 0.00, 0.15, 0.00...
## $ V25 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V26 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V27 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V28 <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V29 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V30 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V31 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V32 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V33 <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.15, 0.00...
## $ V34 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V35 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V36 <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V37 <dbl> 0.00, 0.07, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V38 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V39 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V40 <dbl> 0.00, 0.00, 0.06, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V41 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V42 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V43 <dbl> 0.00, 0.00, 0.12, 0.00, 0.00, 0.00, 0.00, 0.00, 0.30, 0.00...
## $ V44 <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.06...
## $ V45 <dbl> 0.00, 0.00, 0.06, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V46 <dbl> 0.00, 0.00, 0.06, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00...
## $ V47 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V48 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0...
## $ V49 <dbl> 0.000, 0.000, 0.010, 0.000, 0.000, 0.000, 0.000, 0.000, 0....
## $ V50 <dbl> 0.000, 0.132, 0.143, 0.137, 0.135, 0.223, 0.054, 0.206, 0....
## $ V51 <dbl> 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0....
```

```
## $ V52 <dbl> 0.778, 0.372, 0.276, 0.137, 0.135, 0.000, 0.164, 0.000, 0....
## $ V53 <dbl> 0.000, 0.180, 0.184, 0.000, 0.000, 0.000, 0.054, 0.000, 0....
## $ V54 <dbl> 0.000, 0.048, 0.010, 0.000, 0.000, 0.000, 0.000, 0.000, 0....
## $ V55 <dbl> 3.756, 5.114, 9.821, 3.537, 3.537, 3.000, 1.671, 2.450, 9....
## $ V56 <int> 61, 101, 485, 40, 40, 15, 4, 11, 445, 43, 6, 11, 61, 7, 24...
## $ V57 <int> 278, 1028, 2259, 191, 191, 54, 112, 49, 1257, 749, 21, 184...
## $ V58 <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
```

Let's see if we have any missing data.

```
# Find NA per Column
if (sum(is.na(spamdata)) == 0) {
  print(paste("The count of NA in dataset =", sum(is.na(spamdata))))
} else {
  cnt <- ncol(spamdata)
  for (i in 1:cnt) {
    print(colnames(spamdata)[i])
    print(sum(is.na(spamdata[,i])))
  }
}
```

```
## [1] "The count of NA in dataset = 0"
```

It is nice that we do not have to deal with missing data.

Next, we have to add names for our variables, correct a few variable data types, and move the response variable, "spam" to the first column.

```
#create a list of variable names
newnames <- c("freq_make", "freq_address", "freq_all", "freq_3d", "freq_our", "freq_over", "freq
_remove", "freq_internet",
              "freq_order", "freq_mail", "freq_receive", "freq_will", "freq_people", "freq_repor
t", "freq_addresses", "freq_free",
              "freq_business", "freq_email", "freq_you", "freq_credit", "freq_your", "freq_font"
, "freq_000", "freq_money",
              "freq_hp", "freq_hpl", "freq_george", "freq_650", "freq_lab", "freq_labs", "freq_t
elnet", "freq_857",
              "freq_data", "freq_415", "freq_85", "freq_technology", "freq_1999", "freq_parts",
"freq_pm", "freq_direct",
              "freq_vcs", "freq_meeting", "freq_original", "freq_project", "freq_re", "freq_edu"
, "freq_table", "freq_conference",
              "freq_semicolon", "freq_parentheses", "freq_bracket", "freq_exclamation_point", "f
req_dollar_sign", "freq_pound_sign",
              "caps_len_average", "caps_len_longest", "caps_len_total", "spam")

# set the variable names
names(spamdata) <- newnames

# tidy the data
spamdata$caps_len_longest <- as.double(spamdata$caps_len_longest)
spamdata$caps_len_total <- as.double(spamdata$caps_len_total)
spamdata <- spamdata %>% dplyr::select(spam,everything())
spamdata$spam <- as.factor(ifelse(spamdata$spam==1,"Y","N"))
```

Let's make sure our variables look better now.

```
glimpse(spamdata)
```

```
## Observations: 4,601
## Variables: 58
## $ spam                  <fct> Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, Y, ...
## $ freq_make             <dbl> 0.00, 0.21, 0.06, 0.00, 0.00, 0.00, 0.0...
## $ freq_address          <dbl> 0.64, 0.28, 0.00, 0.00, 0.00, 0.00, 0.0...
## $ freq_all              <dbl> 0.64, 0.50, 0.71, 0.00, 0.00, 0.00, 0.0...
## $ freq_3d               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_our              <dbl> 0.32, 0.14, 1.23, 0.63, 0.63, 1.85, 1.9...
## $ freq_over             <dbl> 0.00, 0.28, 0.19, 0.00, 0.00, 0.00, 0.0...
## $ freq_remove           <dbl> 0.00, 0.21, 0.19, 0.31, 0.31, 0.00, 0.0...
## $ freq_internet         <dbl> 0.00, 0.07, 0.12, 0.63, 0.63, 1.85, 0.0...
## $ freq_order            <dbl> 0.00, 0.00, 0.64, 0.31, 0.31, 0.00, 0.0...
## $ freq_mail             <dbl> 0.00, 0.94, 0.25, 0.63, 0.63, 0.00, 0.6...
## $ freq_receive          <dbl> 0.00, 0.21, 0.38, 0.31, 0.31, 0.00, 0.9...
## $ freq_will             <dbl> 0.64, 0.79, 0.45, 0.31, 0.31, 0.00, 1.2...
## $ freq_people           <dbl> 0.00, 0.65, 0.12, 0.31, 0.31, 0.00, 0.0...
## $ freq_report           <dbl> 0.00, 0.21, 0.00, 0.00, 0.00, 0.00, 0.0...
## $ freq_addresses        <dbl> 0.00, 0.14, 1.75, 0.00, 0.00, 0.00, 0.0...
## $ freq_free             <dbl> 0.32, 0.14, 0.06, 0.31, 0.31, 0.00, 0.9...
## $ freq_business         <dbl> 0.00, 0.07, 0.06, 0.00, 0.00, 0.00, 0.0...
## $ freq_email            <dbl> 1.29, 0.28, 1.03, 0.00, 0.00, 0.00, 0.3...
## $ freq_you              <dbl> 1.93, 3.47, 1.36, 3.18, 3.18, 0.00, 3.8...
## $ freq_credit           <dbl> 0.00, 0.00, 0.32, 0.00, 0.00, 0.00, 0.0...
## $ freq_your             <dbl> 0.96, 1.59, 0.51, 0.31, 0.31, 0.00, 0.6...
## $ freq_font             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_000              <dbl> 0.00, 0.43, 1.16, 0.00, 0.00, 0.00, 0.0...
## $ freq_money            <dbl> 0.00, 0.43, 0.06, 0.00, 0.00, 0.00, 0.0...
## $ freq_hp               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_hpl              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_george           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_650              <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.0...
## $ freq_lab              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_labs             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_telnet           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_857              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_data             <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.0...
## $ freq_415              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_85               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_technology       <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.0...
## $ freq_1999             <dbl> 0.00, 0.07, 0.00, 0.00, 0.00, 0.00, 0.0...
## $ freq_parts            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_pm               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_direct           <dbl> 0.00, 0.00, 0.06, 0.00, 0.00, 0.00, 0.0...
## $ freq_vcs              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_meeting          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_original         <dbl> 0.00, 0.00, 0.12, 0.00, 0.00, 0.00, 0.0...
## $ freq_project          <dbl> 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.0...
## $ freq_re               <dbl> 0.00, 0.00, 0.06, 0.00, 0.00, 0.00, 0.0...
## $ freq_edu              <dbl> 0.00, 0.00, 0.06, 0.00, 0.00, 0.00, 0.0...
## $ freq_table            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_conference       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ freq_semicolon        <dbl> 0.000, 0.000, 0.010, 0.000, 0.000, 0.00...
## $ freq_parentheses      <dbl> 0.000, 0.132, 0.143, 0.137, 0.135, 0.22...
```

```
## $ freq_bracket           <dbl> 0.000, 0.000, 0.000, 0.000, 0.000, 0.00...
## $ freq_exclamation_point <dbl> 0.778, 0.372, 0.276, 0.137, 0.135, 0.00...
## $ freq_dollar_sign       <dbl> 0.000, 0.180, 0.184, 0.000, 0.000, 0.00...
## $ freq_pound_sign        <dbl> 0.000, 0.048, 0.010, 0.000, 0.000, 0.00...
## $ caps_len_average       <dbl> 3.756, 5.114, 9.821, 3.537, 3.537, 3.00...
## $ caps_len_longest       <dbl> 61, 101, 485, 40, 40, 15, 4, 11, 445, 4...
## $ caps_len_total         <dbl> 278, 1028, 2259, 191, 191, 54, 112, 49,...
```

Let's get a clean, condensed look at the variable names.

```
names(spamdata)
```

```
##  [1] "spam"                  "freq_make"
##  [3] "freq_address"          "freq_all"
##  [5] "freq_3d"               "freq_our"
##  [7] "freq_over"             "freq_remove"
##  [9] "freq_internet"         "freq_order"
## [11] "freq_mail"             "freq_receive"
## [13] "freq_will"             "freq_people"
## [15] "freq_report"           "freq_addresses"
## [17] "freq_free"             "freq_business"
## [19] "freq_email"            "freq_you"
## [21] "freq_credit"           "freq_your"
## [23] "freq_font"             "freq_000"
## [25] "freq_money"            "freq_hp"
## [27] "freq_hpl"              "freq_george"
## [29] "freq_650"              "freq_lab"
## [31] "freq_labs"             "freq_telnet"
## [33] "freq_857"              "freq_data"
## [35] "freq_415"              "freq_85"
## [37] "freq_technology"       "freq_1999"
## [39] "freq_parts"            "freq_pm"
## [41] "freq_direct"           "freq_vcs"
## [43] "freq_meeting"          "freq_original"
## [45] "freq_project"          "freq_re"
## [47] "freq_edu"              "freq_table"
## [49] "freq_conference"       "freq_semicolon"
## [51] "freq_parentheses"      "freq_bracket"
## [53] "freq_exclamation_point" "freq_dollar_sign"
## [55] "freq_pound_sign"       "caps_len_average"
## [57] "caps_len_longest"      "caps_len_total"
```

Now, we can look at a table of values for the response variable.

```
table(spamdata$spam)
```

```
##
##    N    Y
## 2788 1813
```

We have 1813 observations that are classified as "spam" and 2788 observations that are classified as not "spam".

Next, let's check the data for outliers.

```
#manually look for outliers
# standard deviation methods
# 2 SD beyond the mean covers 95% of a normally distributed variable
# 3 SD +/- the mean covers 99.7% of a normally distributed variable

for (i in 2:ncol(spamdata)){
  temp <- spamdata[,i] > (mean(spamdata[,i] + 3*sd(spamdata[,i])))
  temp2 <- round(table(temp)[2]/nrow(spamdata) * 100,2)
  temp2
  print(paste(temp2, "pecent of the", names(spamdata)[i], "data exceeds 3 SD from the mean"))
}
```

```
## [1] "1.96 pecent of the freq_make data exceeds 3 SD from the mean"
## [1] "0.93 pecent of the freq_address data exceeds 3 SD from the mean"
## [1] "2.04 pecent of the freq_all data exceeds 3 SD from the mean"
## [1] "0.28 pecent of the freq_3d data exceeds 3 SD from the mean"
## [1] "1.76 pecent of the freq_our data exceeds 3 SD from the mean"
## [1] "2.26 pecent of the freq_over data exceeds 3 SD from the mean"
## [1] "2.15 pecent of the freq_remove data exceeds 3 SD from the mean"
## [1] "1.67 pecent of the freq_internet data exceeds 3 SD from the mean"
## [1] "2.46 pecent of the freq_order data exceeds 3 SD from the mean"
## [1] "1.61 pecent of the freq_mail data exceeds 3 SD from the mean"
## [1] "2.17 pecent of the freq_receive data exceeds 3 SD from the mean"
## [1] "2.22 pecent of the freq_will data exceeds 3 SD from the mean"
## [1] "1.93 pecent of the freq_people data exceeds 3 SD from the mean"
## [1] "2.3 pecent of the freq_report data exceeds 3 SD from the mean"
## [1] "2.15 pecent of the freq_addresses data exceeds 3 SD from the mean"
## [1] "1.5 pecent of the freq_free data exceeds 3 SD from the mean"
## [1] "2.11 pecent of the freq_business data exceeds 3 SD from the mean"
## [1] "2.3 pecent of the freq_email data exceeds 3 SD from the mean"
## [1] "1.3 pecent of the freq_you data exceeds 3 SD from the mean"
## [1] "1.65 pecent of the freq_credit data exceeds 3 SD from the mean"
## [1] "1.89 pecent of the freq_your data exceeds 3 SD from the mean"
## [1] "1.24 pecent of the freq_font data exceeds 3 SD from the mean"
## [1] "2.33 pecent of the freq_000 data exceeds 3 SD from the mean"
## [1] "0.7 pecent of the freq_money data exceeds 3 SD from the mean"
## [1] "1.87 pecent of the freq_hp data exceeds 3 SD from the mean"
## [1] "2.28 pecent of the freq_hpl data exceeds 3 SD from the mean"
## [1] "2.67 pecent of the freq_george data exceeds 3 SD from the mean"
## [1] "2.33 pecent of the freq_650 data exceeds 3 SD from the mean"
## [1] "1.39 pecent of the freq_lab data exceeds 3 SD from the mean"
## [1] "1.96 pecent of the freq_labs data exceeds 3 SD from the mean"
## [1] "1.33 pecent of the freq_telnet data exceeds 3 SD from the mean"
## [1] "1.11 pecent of the freq_857 data exceeds 3 SD from the mean"
## [1] "1.67 pecent of the freq_data data exceeds 3 SD from the mean"
## [1] "1.13 pecent of the freq_415 data exceeds 3 SD from the mean"
## [1] "1.78 pecent of the freq_85 data exceeds 3 SD from the mean"
## [1] "1.67 pecent of the freq_technology data exceeds 3 SD from the mean"
## [1] "2.28 pecent of the freq_1999 data exceeds 3 SD from the mean"
## [1] "0.46 pecent of the freq_parts data exceeds 3 SD from the mean"
## [1] "1.5 pecent of the freq_pm data exceeds 3 SD from the mean"
## [1] "1.33 pecent of the freq_direct data exceeds 3 SD from the mean"
## [1] "1.35 pecent of the freq_vcs data exceeds 3 SD from the mean"
## [1] "1.7 pecent of the freq_meeting data exceeds 3 SD from the mean"
## [1] "2.33 pecent of the freq_original data exceeds 3 SD from the mean"
## [1] "1.11 pecent of the freq_project data exceeds 3 SD from the mean"
## [1] "1.39 pecent of the freq_re data exceeds 3 SD from the mean"
## [1] "1.61 pecent of the freq_edu data exceeds 3 SD from the mean"
## [1] "0.61 pecent of the freq_table data exceeds 3 SD from the mean"
## [1] "1.06 pecent of the freq_conference data exceeds 3 SD from the mean"
## [1] "0.67 pecent of the freq_semicolon data exceeds 3 SD from the mean"
## [1] "1.06 pecent of the freq_parentheses data exceeds 3 SD from the mean"
## [1] "0.65 pecent of the freq_bracket data exceeds 3 SD from the mean"
## [1] "0.96 pecent of the freq_exclamation_point data exceeds 3 SD from the mean"
## [1] "1.35 pecent of the freq_dollar_sign data exceeds 3 SD from the mean"
```

```
## [1] "0.52 pecent of the freq_pound_sign data exceeds 3 SD from the mean"
## [1] "0.46 pecent of the caps_len_average data exceeds 3 SD from the mean"
## [1] "1.48 pecent of the caps_len_longest data exceeds 3 SD from the mean"
## [1] "1.87 pecent of the caps_len_total data exceeds 3 SD from the mean"
```

```
#sapply(dat_num, function(d) (d > (mean(d) + 2*sd(d))))
#sapply(dat_num, function(d) which(d > (mean(d) + 2*sd(d))))
```

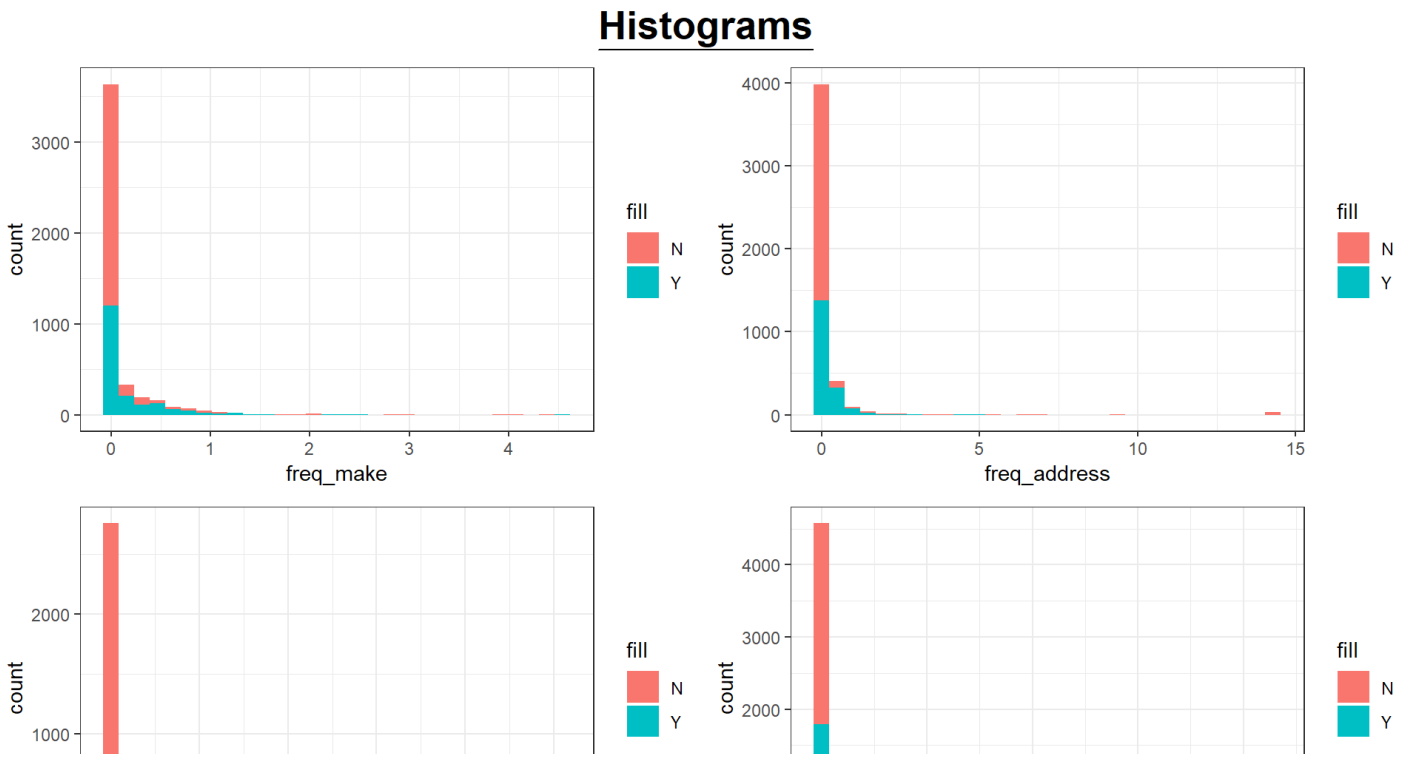
There are some outliers in all of the variable, but nothing seems too alarming.

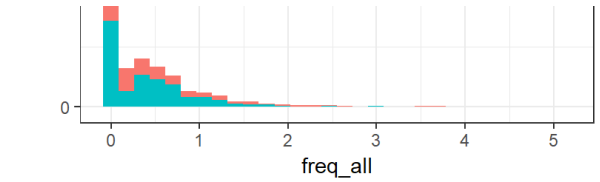
HISTOGRAMS

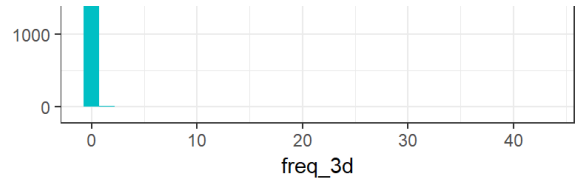
Let's create histograms for the unscaled predictor variables

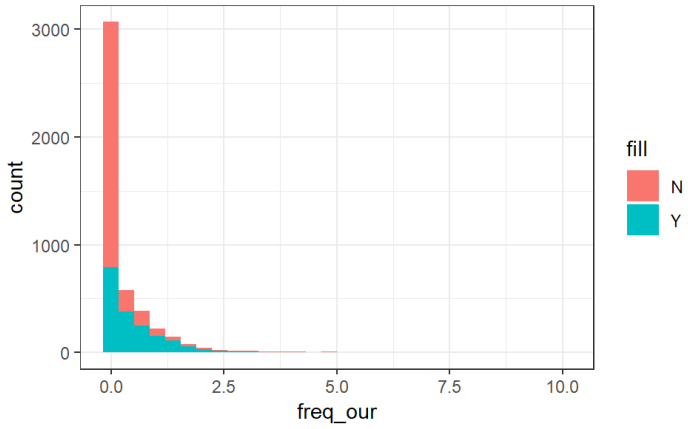
```
# Create Histograms
for (i in 2:ncol(spamdata)){
  assign(paste0("h", i), ggplot(data= spamdata, aes_string(colnames(spamdata)[i], fill=spamdata
$spam)) + geom_histogram() + #ggtitle(names(spamdata[i])) +
  theme_bw()+ theme(plot.title = element_text(hjust = 0.5)))
}
```

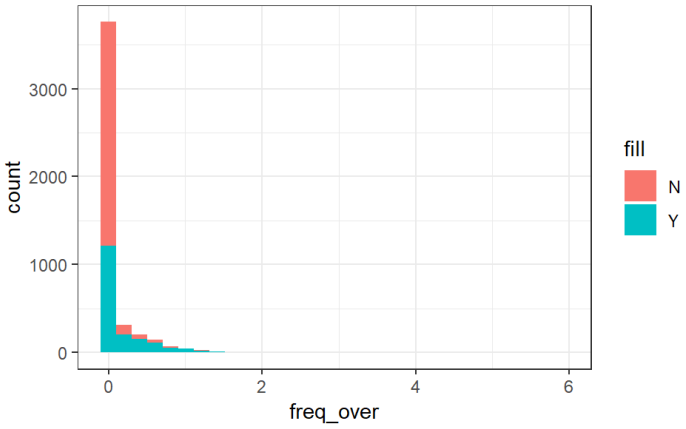
```
grid.arrange(h2, h3, h4, h5, h6, h7, h8, h9, h10, h11,
             h12, h13, h14, h15, h16, h17, h18, h19, h20, h21,
             h22, h23, h24, h25, h26, h27, h28, h29, h30, h31,
             h32, h33, h34, h35, h36, h37, h38, h39, h40, h41,
             h42, h43, h44, h45, h46, h47, h48, h49, h50, h51,
             h52, h53, h54, h55, h56, h57, h58,
             ncol=2,
             top=textGrob(expression(bold(underline("Histograms"))),
                          gp=gpar(fontsize=20,font=3)))
```

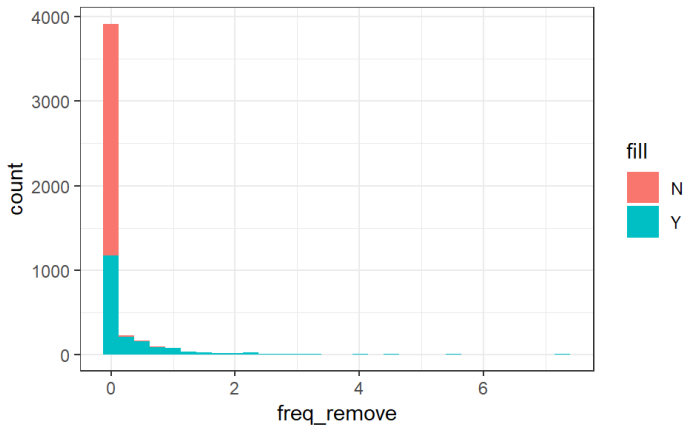


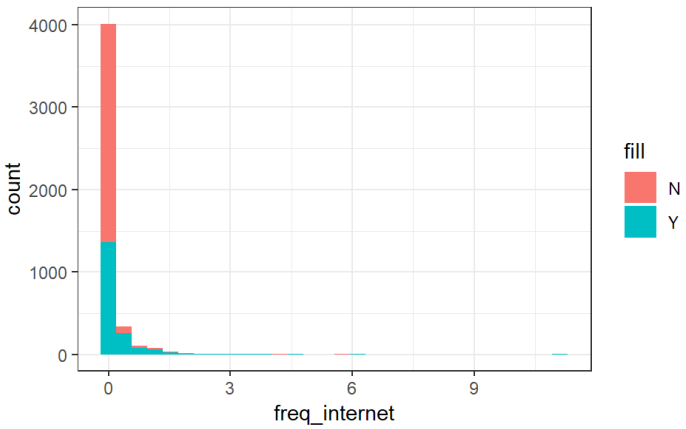

0
0
1
2
3
4
5
freq_all

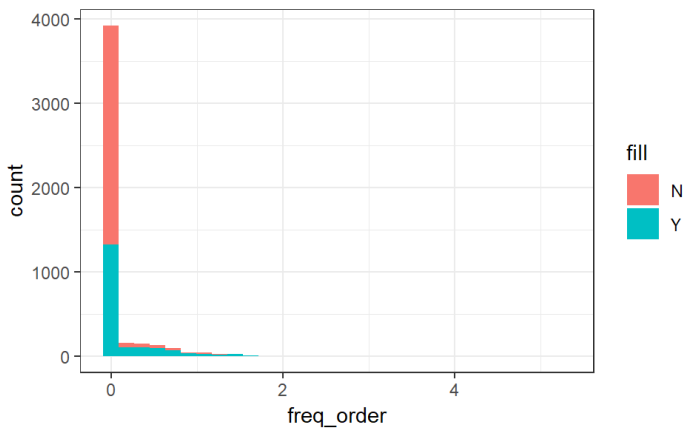

1000
0
0
10
20
30
40
freq_3d

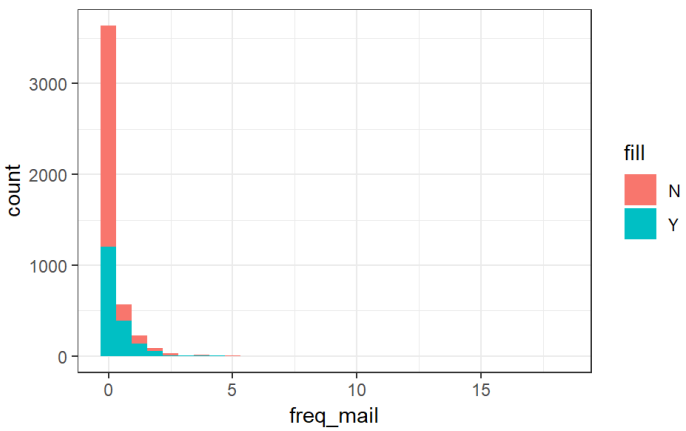

count
3000
2000
1000
0
0.0
2.5
5.0
7.5
10.0
freq_our
fill
N
Y

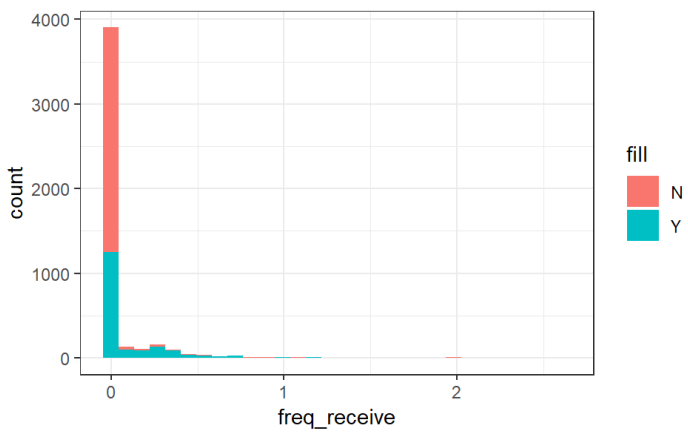

count
3000
2000
1000
0
0
2
4
6
freq_over
fill
N
Y

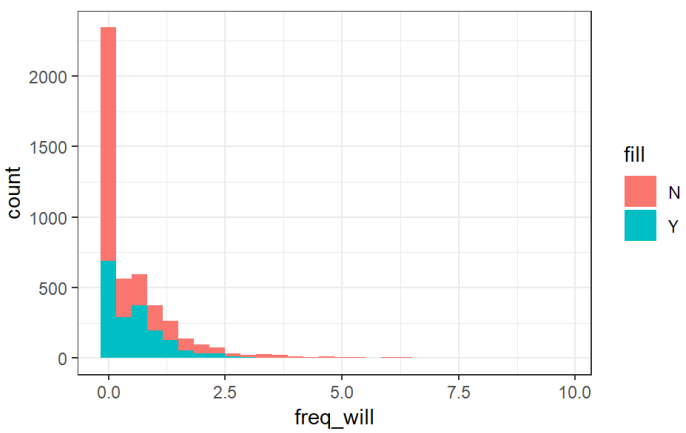

count
4000
3000
2000
1000
0
0
2
4
6
freq_remove
fill
N
Y


count
4000
3000
2000
1000
0
0
3
6
9
freq_internet
fill
N
Y

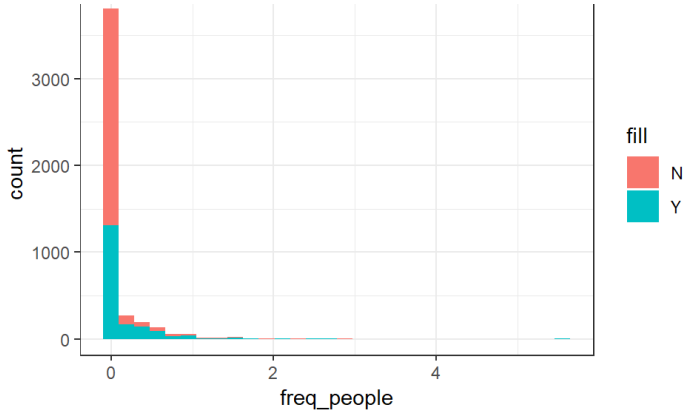

count
4000
3000
2000
1000
0
0
2
4
freq_order
fill
N
Y

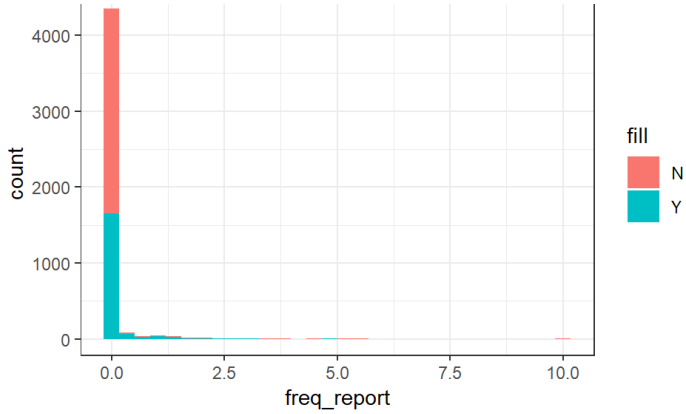

count
3000
2000
1000
0
0
5
10
15
freq_mail
fill
N
Y

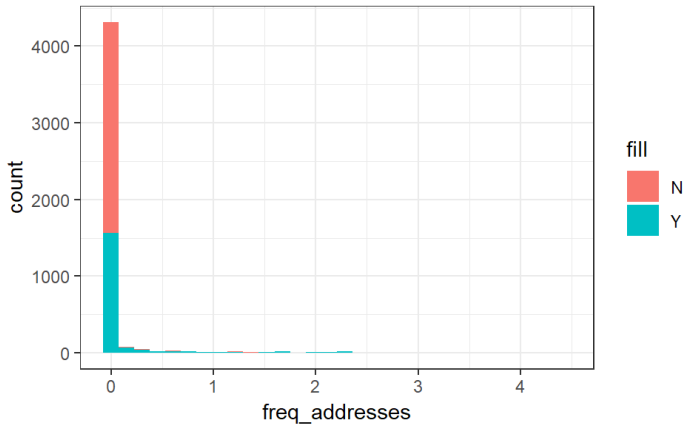

count
4000
3000
2000
1000
0
0
1
2
freq_receive
fill
N
Y

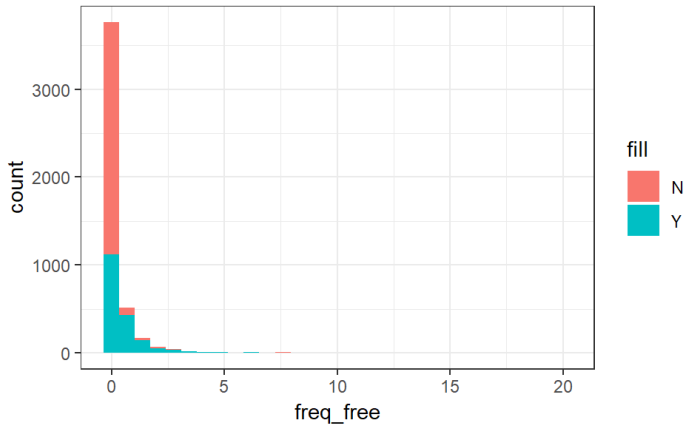

count
2000
1500
1000
500
0
0.0
2.5
5.0
7.5
10.0
freq_will
fill
N
Y

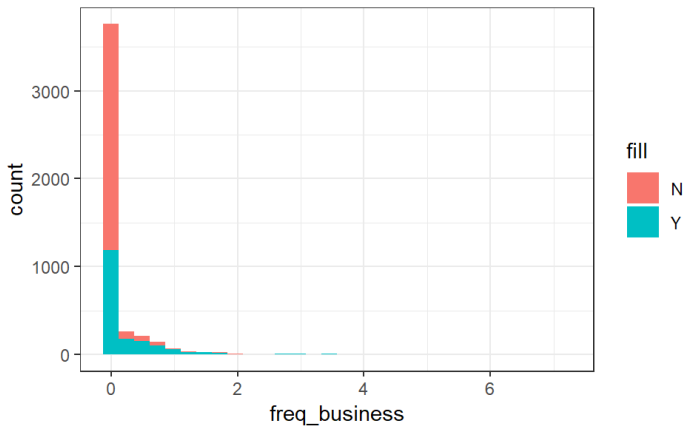

4000

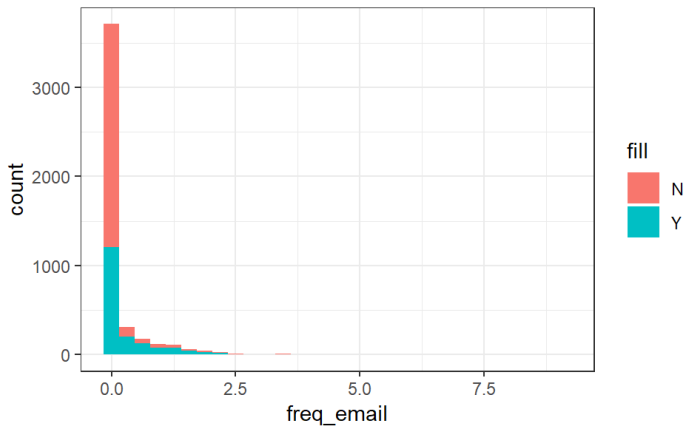

count
3000
2000
1000
0
0
2
4
freq_people
fill
N
Y

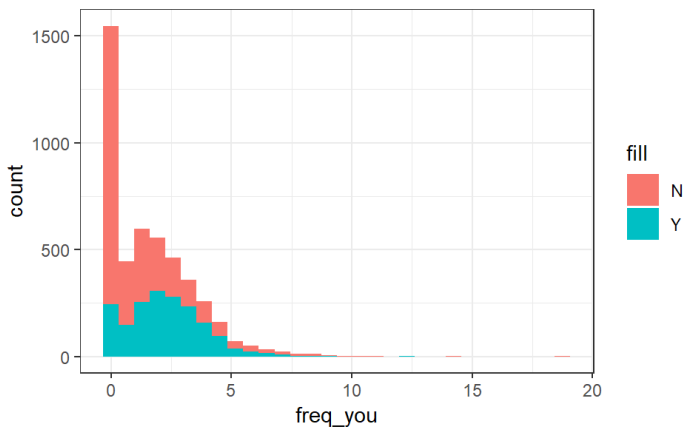

count
4000
3000
2000
1000
0
0.0
2.5
5.0
7.5
10.0
freq_report
fill
N
Y

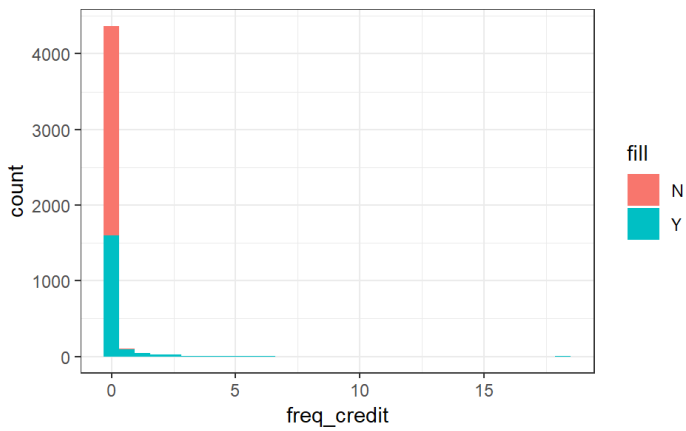

count
4000
3000
2000
1000
0
0
1
2
3
4
freq_addresses
fill
N
Y

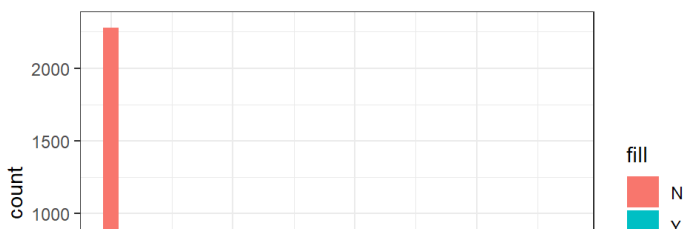

count
3000
2000
1000
0
0
5
10
15
20
freq_free
fill
N
Y

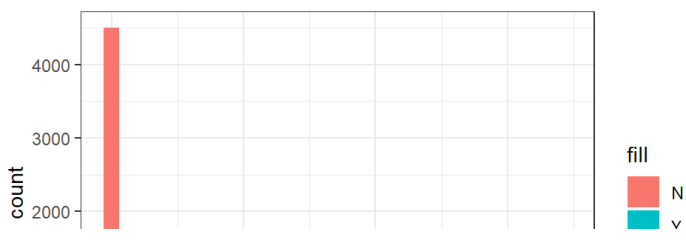

count
3000
2000
1000
0
0
2
4
6
freq_business
fill
N
Y

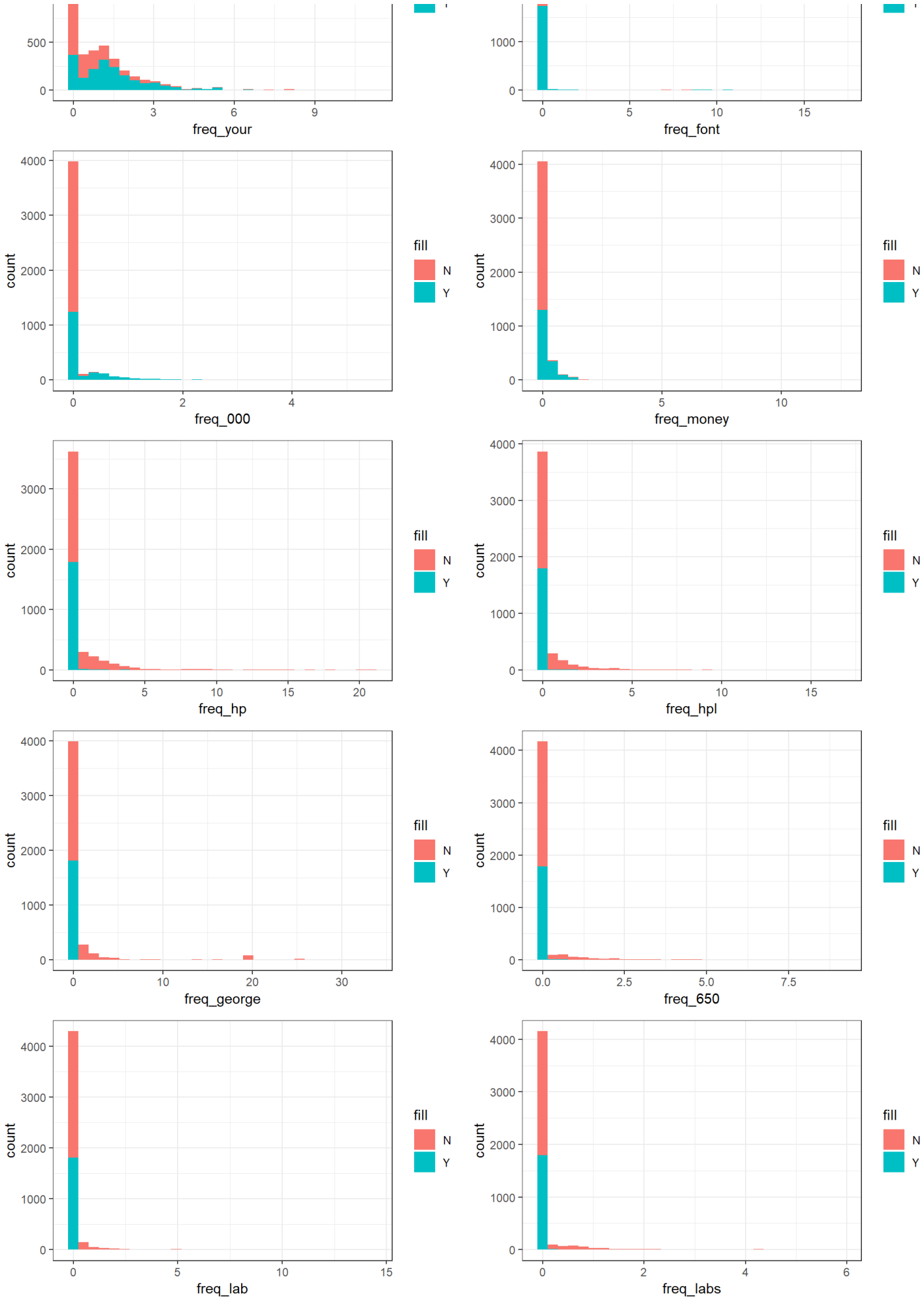

count
3000
2000
1000
0
0.0
2.5
5.0
7.5
freq_email
fill
N
Y

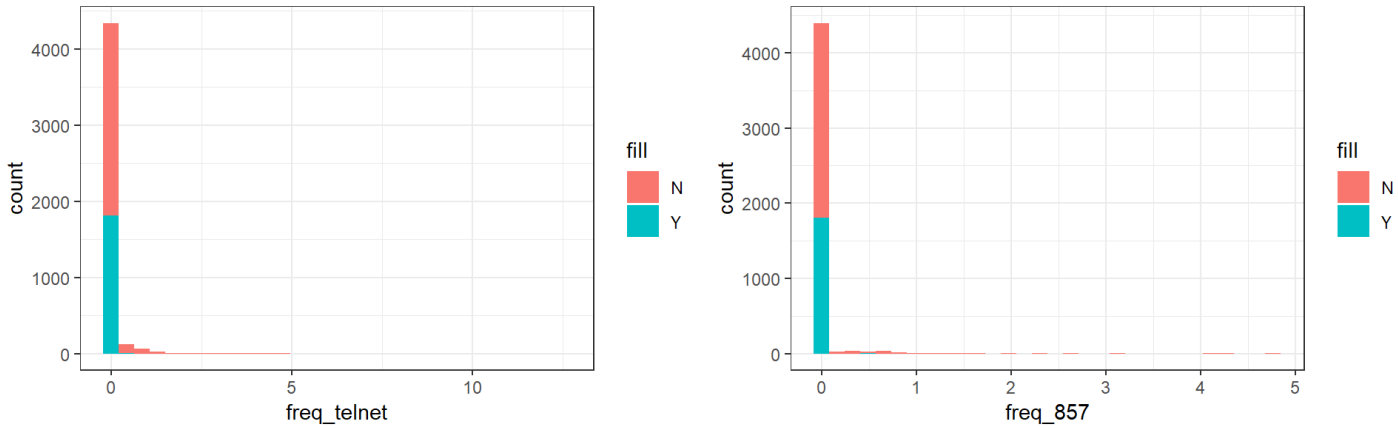

count
1500
1000
500
0
0
5
10
15
20
freq_you
fill
N
Y


count
4000
3000
2000
1000
0
0
5
10
15
freq_credit
fill
N
Y

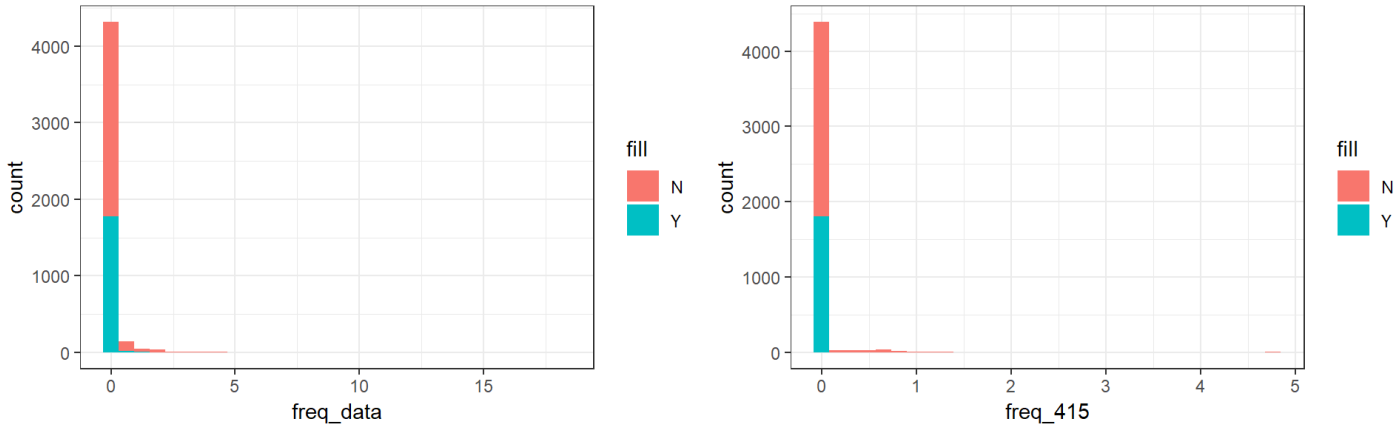

count
2000
1500
1000
fill
N


count
4000
3000
2000
fill
N

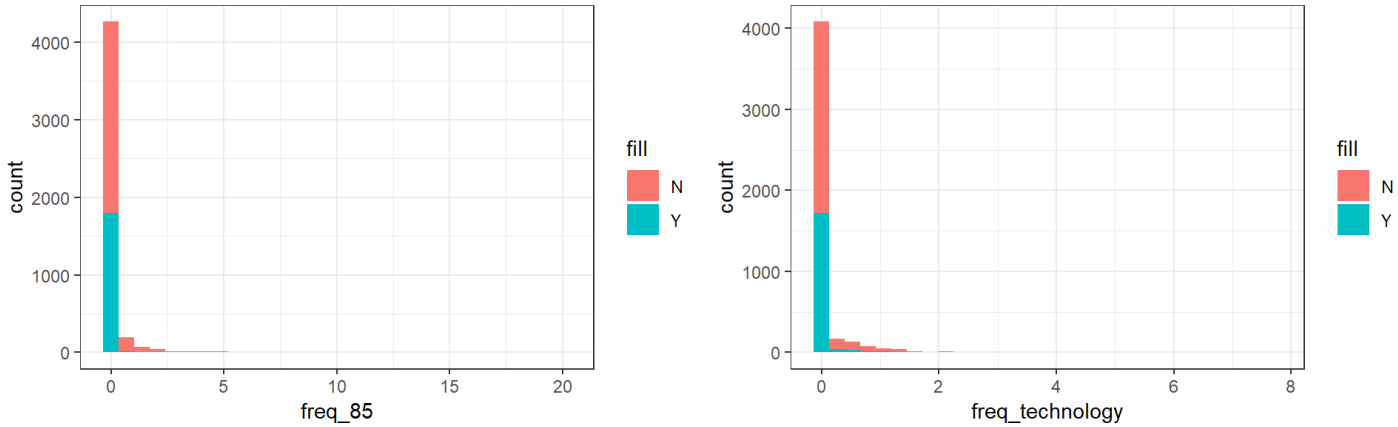

freq_your
freq_font
freq_000
freq_money
freq_hp
freq_hpl
freq_george
freq_650
freq_lab
freq_labs
count
fill
N
Y


count
4000
3000
2000
1000
0
0
5
10
freq_telnet
fill
N
Y

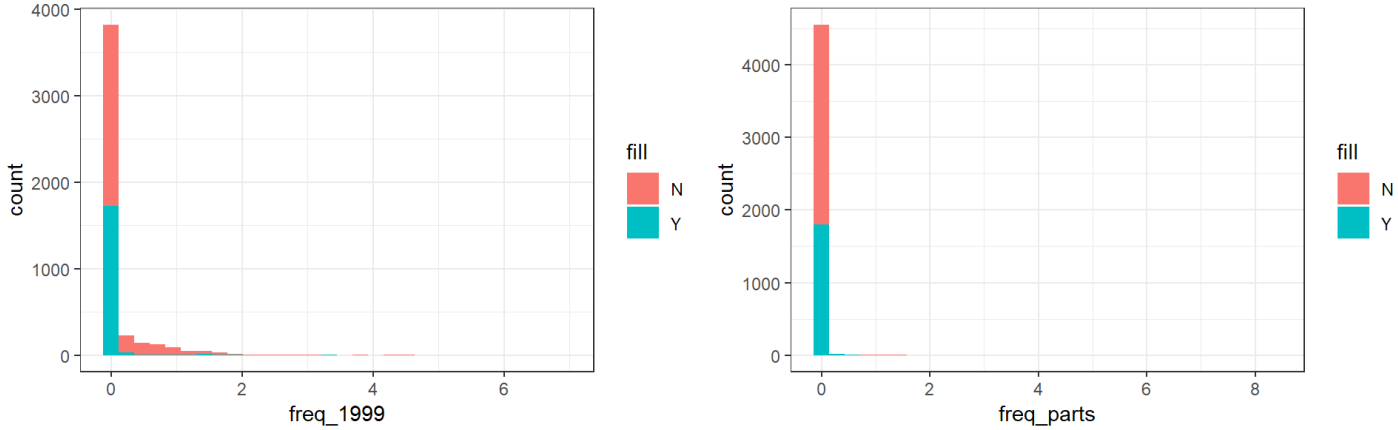
count
4000
3000
2000
1000
0
0
1
2
3
4
5
freq_857
fill
N
Y


count
4000
3000
2000
1000
0
0
5
10
15
freq_data
fill
N
Y

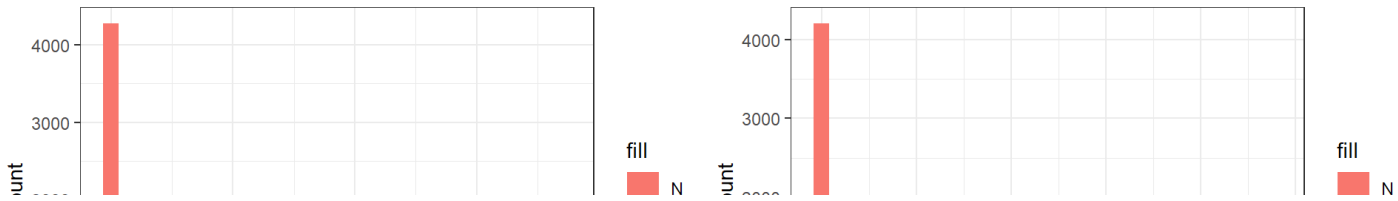
count
4000
3000
2000
1000
0
0
1
2
3
4
5
freq_415
fill
N
Y


count
4000
3000
2000
1000
0
0
5
10
15
20
freq_85
fill
N
Y

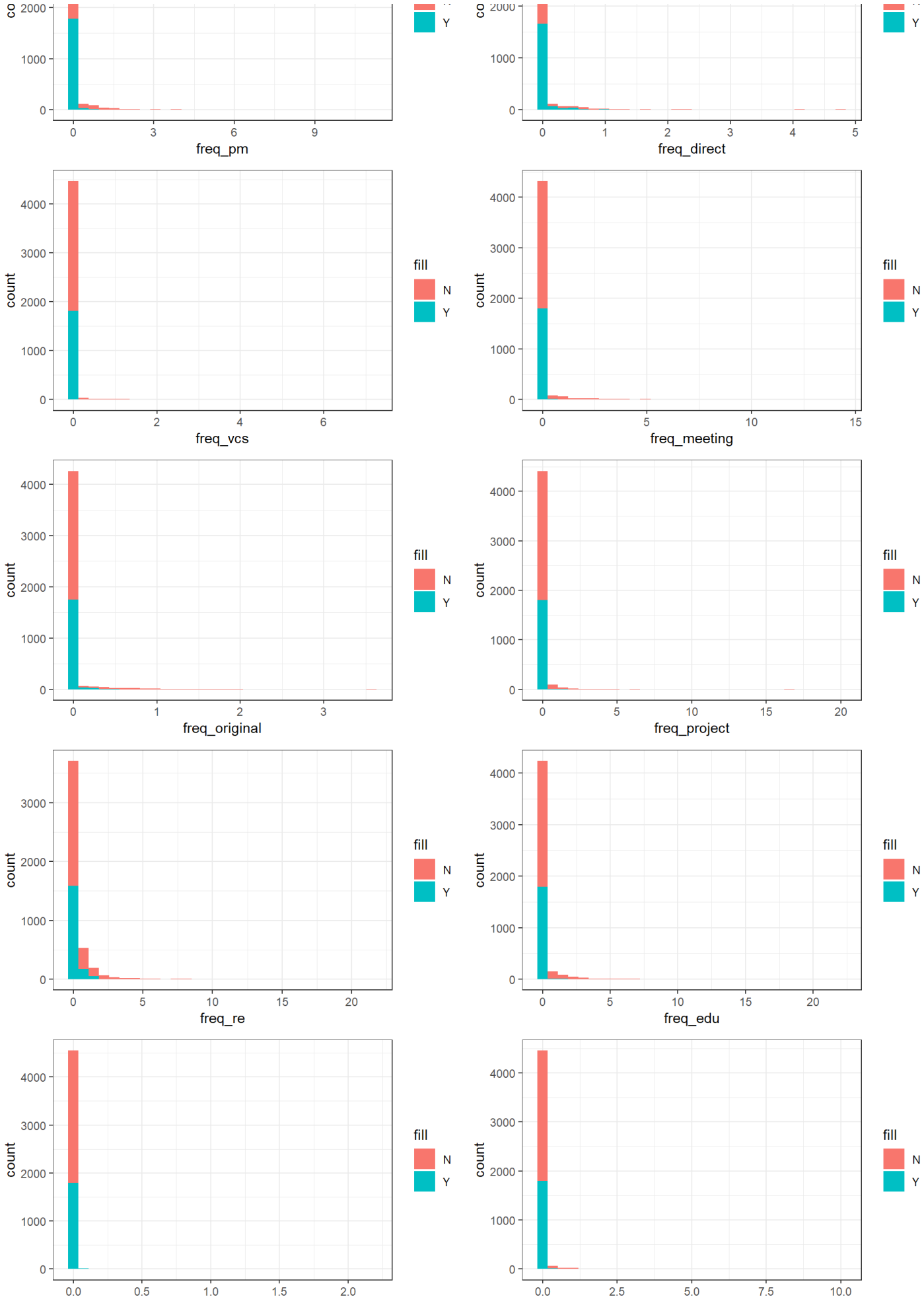
count
4000
3000
2000
1000
0
0
2
4
6
8
freq_technology
fill
N
Y

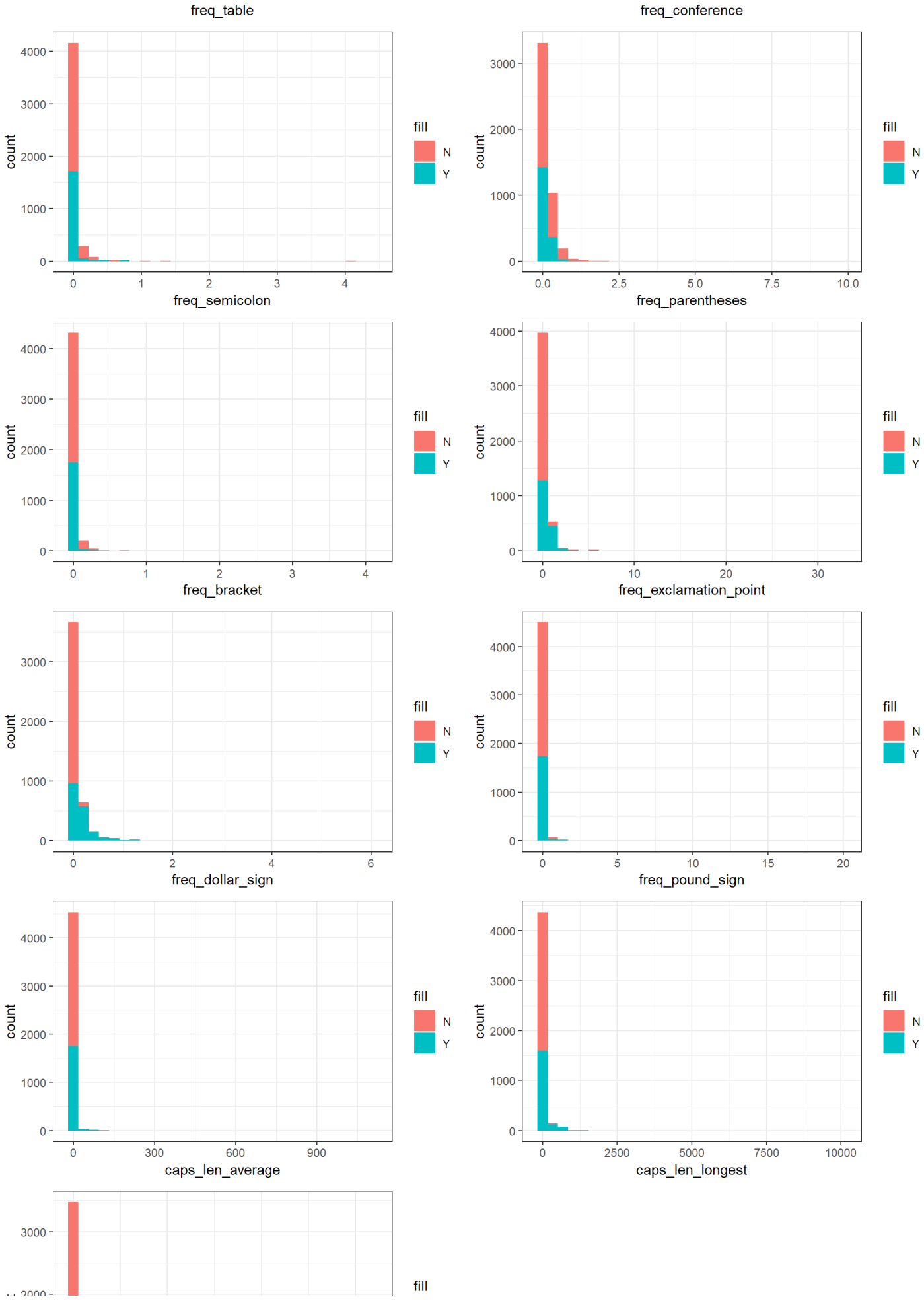

count
4000
3000
2000
1000
0
0
2
4
6
freq_1999
fill
N
Y

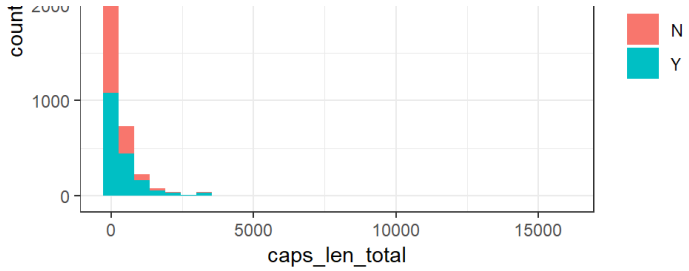
count
4000
3000
2000
1000
0
0
2
4
6
8
freq_parts
fill
N
Y


4000
3000
fill
N

4000
3000
fill
N


freq_pm
freq_direct
freq_vcs
freq_meeting
freq_original
freq_project
freq_re
freq_edu
count
fill
N
Y


freq_table
count
freq_semicolon
fill
N
Y
freq_conference
count
freq_parentheses
fill
N
Y
count
freq_bracket
fill
N
Y
freq_exclamation_point
count
fill
N
Y
count
freq_dollar_sign
fill
N
Y
freq_pound_sign
count
fill
N
Y
count
caps_len_average
fill
N
Y
caps_len_longest
count
fill



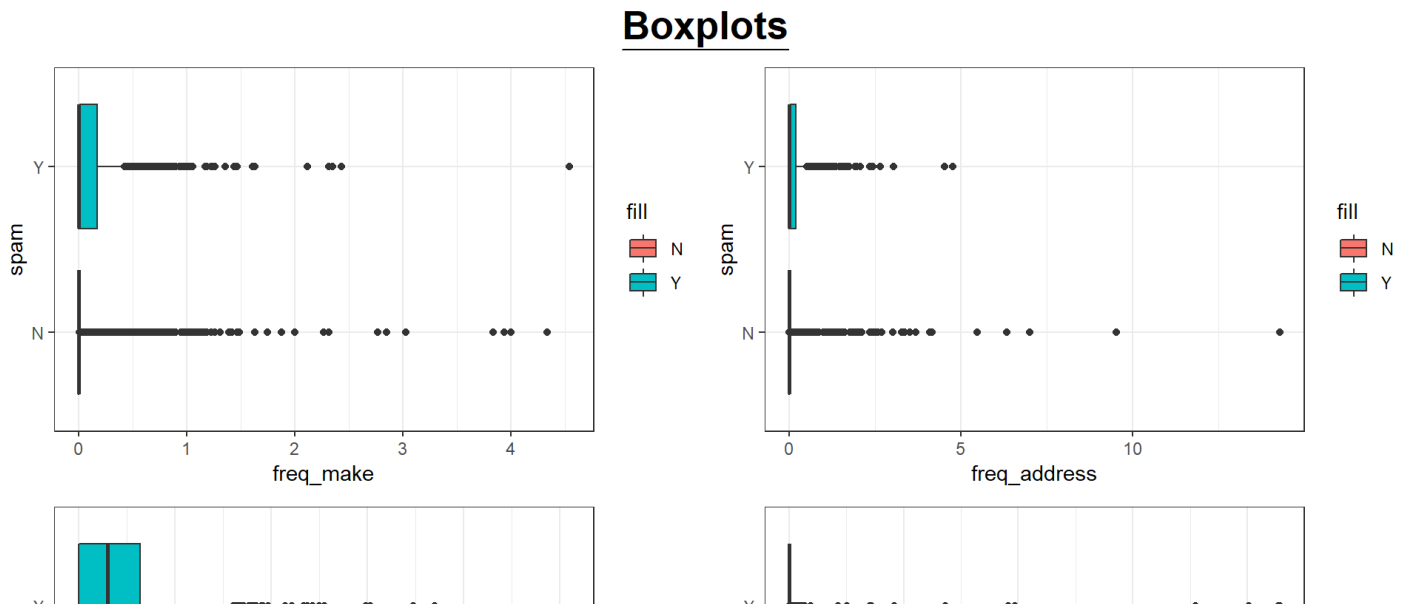
Based on these histograms, it seems clear that some variables are more important than others.

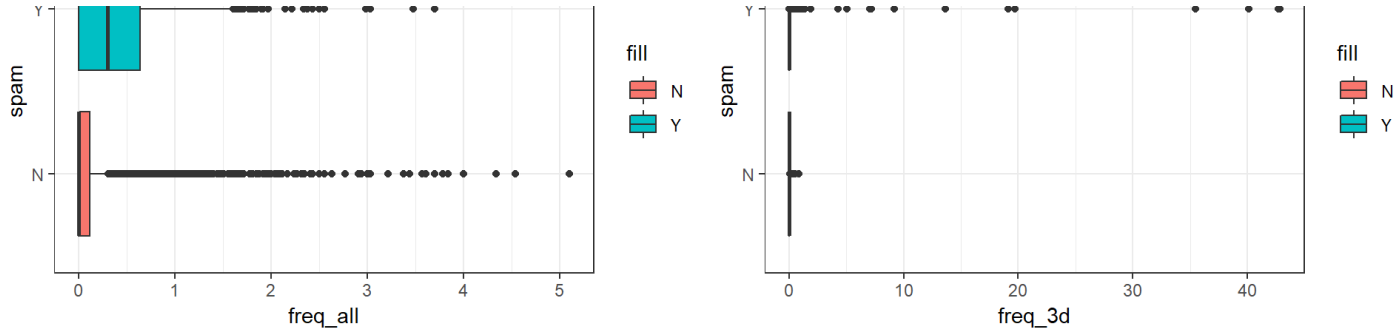
BOX PLOTS

Create box plots for the predictor variables.

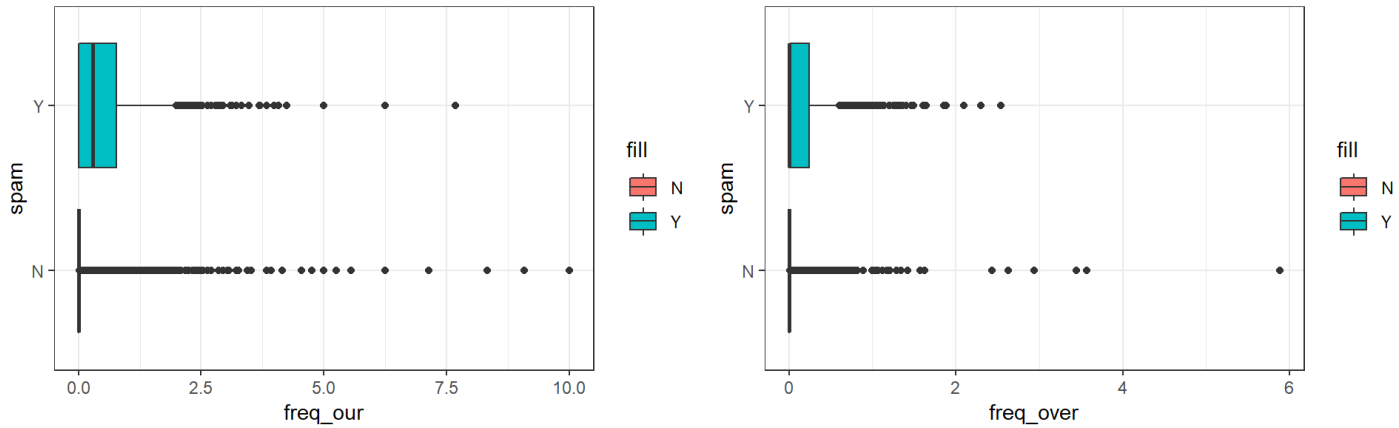
```
# Create Boxplots
for (i in 2:ncol(spamdata)){
  assign(paste0("p", i),
         ggplot( data = spamdata,
                 aes_string(x = names(spamdata[1]),
                     y = names(spamdata[i]),
                               fill = spamdata$spam)) +
                 geom_boxplot() + coord_flip() +
                   theme_bw()+
                   theme(plot.title = element_text(hjust = 1.5)))
}
```

```
grid.arrange(p2, p3, p4, p5, p6, p7, p8, p9, p10, p11,
             p12, p13, p14, p15, p16, p17, p18, p19, p20, p21,
             p22, p23, p24, p25, p26, p27, p28, p29, p30, p31,
             p32, p33, p34, p35, p36, p37, p38, p39, p40, p41,
             p42, p43, p44, p45, p46, p47, p48, p49, p50, p51,
             p52, p53, p54, p55, p56, p57, p58,
             ncol=2,
             top=textGrob(expression(bold(underline("Boxplots"))),
                          gp=gpar(fontsize=20,font=3)))
```

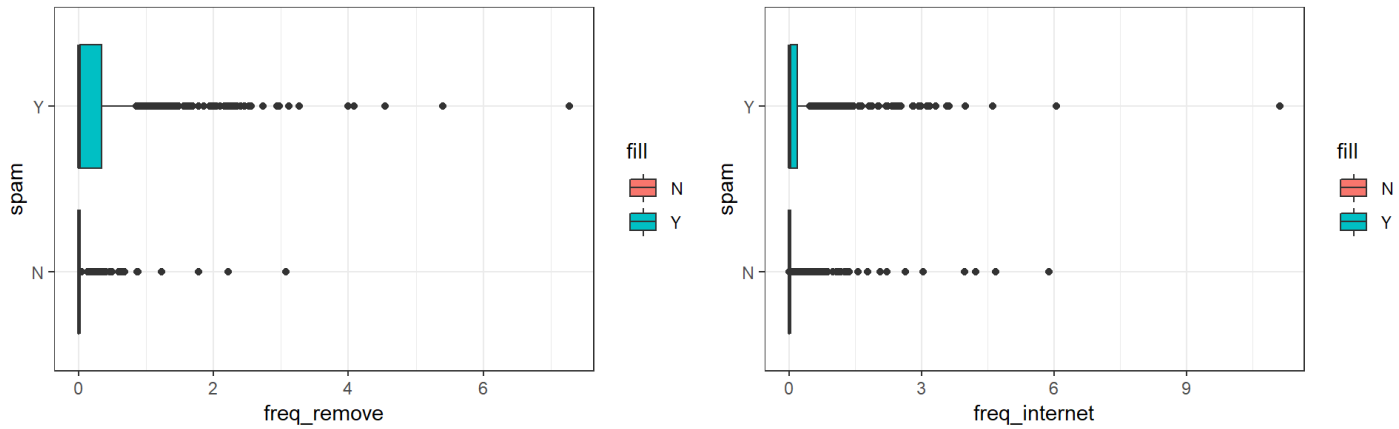



spam
Y
N
0
1
2
3
4
5
freq_all
fill
N
Y

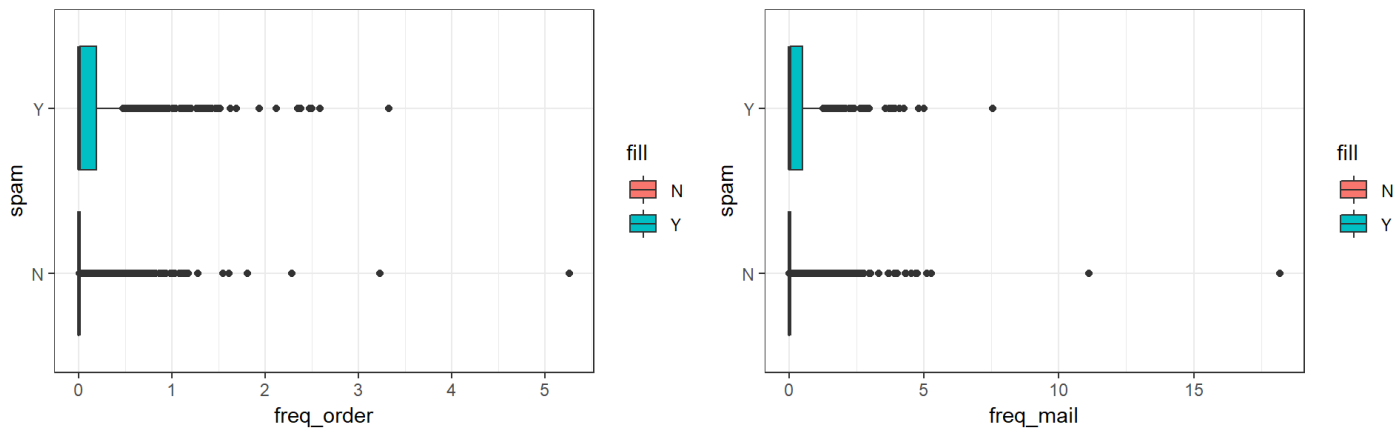
spam
Y
N
0
10
20
30
40
freq_3d
fill
N
Y


spam
Y
N
0.0
2.5
5.0
7.5
10.0
freq_our
fill
N
Y

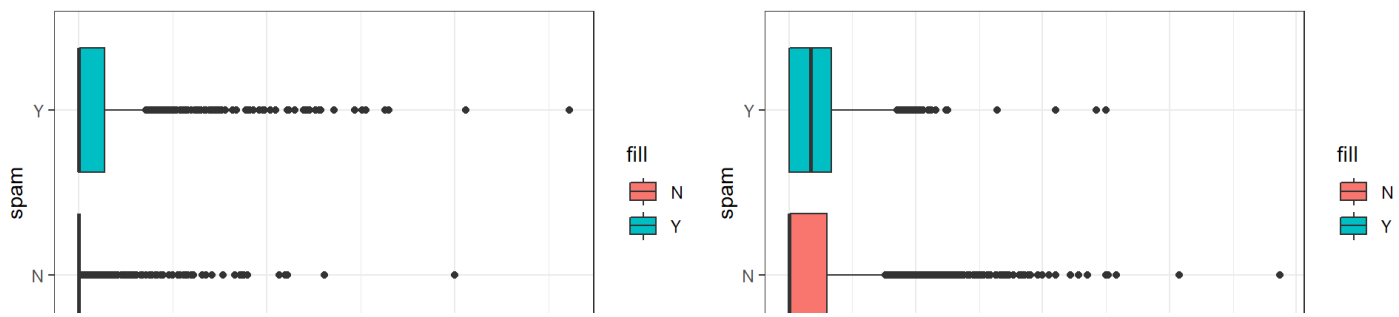
spam
Y
N
0
2
4
6
freq_over
fill
N
Y


spam
Y
N
0
2
4
6
freq_remove
fill
N
Y

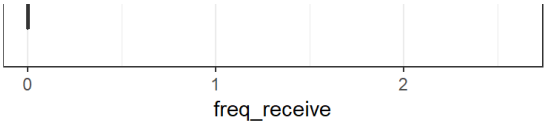
spam
Y
N
0
3
6
9
freq_internet
fill
N
Y

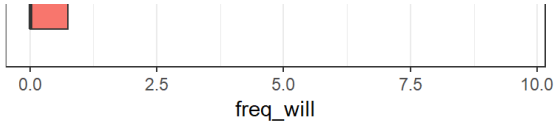

spam
Y
N
0
1
2
3
4
5
freq_order
fill
N
Y

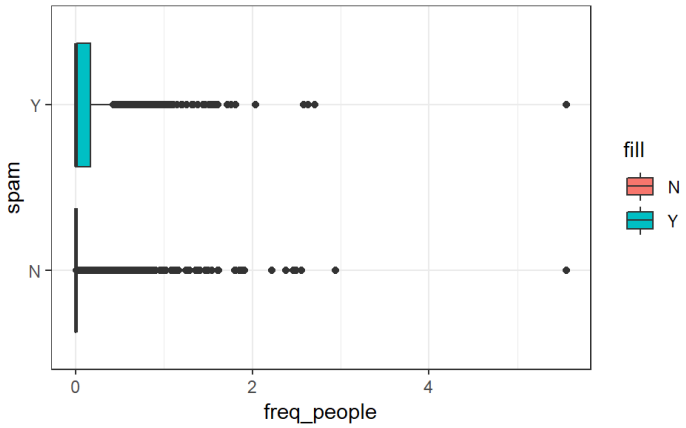
spam
Y
N
0
5
10
15
freq_mail
fill
N
Y

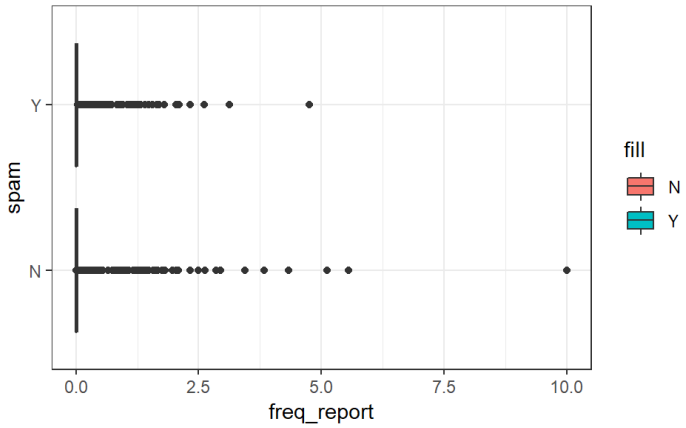

spam
Y
N
fill
N
Y

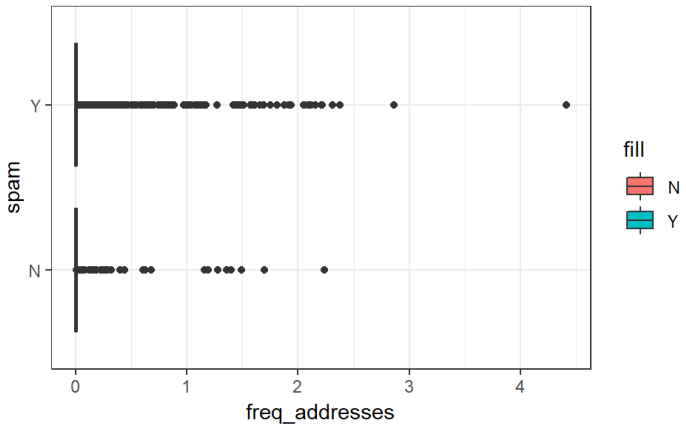
spam
Y
N
fill
N
Y

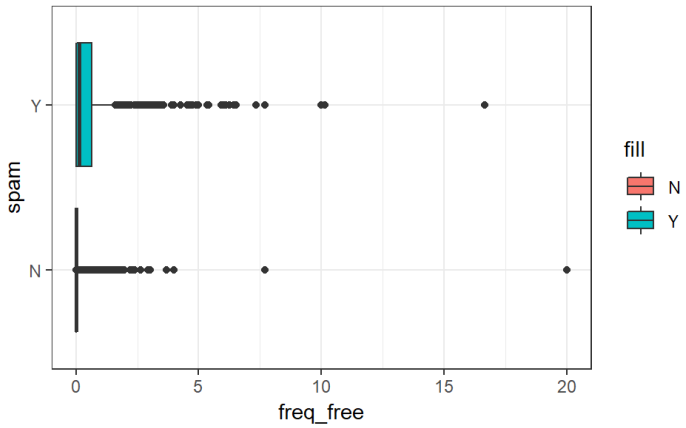

0
1
2
freq_receive

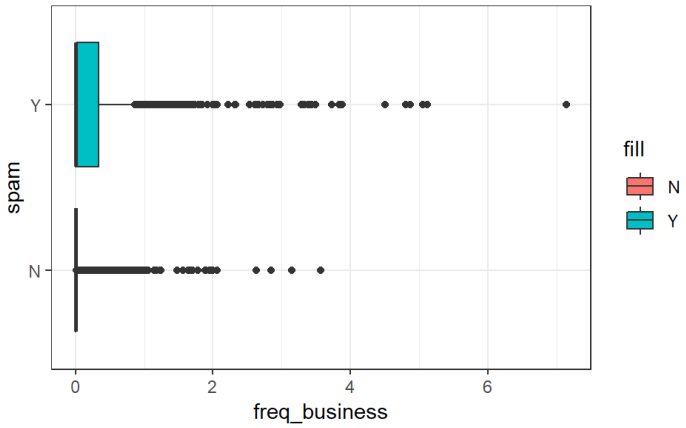

0.0
2.5
5.0
7.5
10.0
freq_will

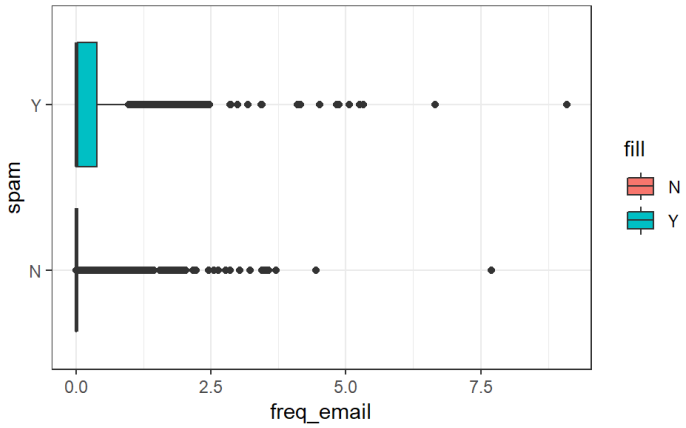

spam
Y
N
0
2
4
freq_people
fill
N
Y

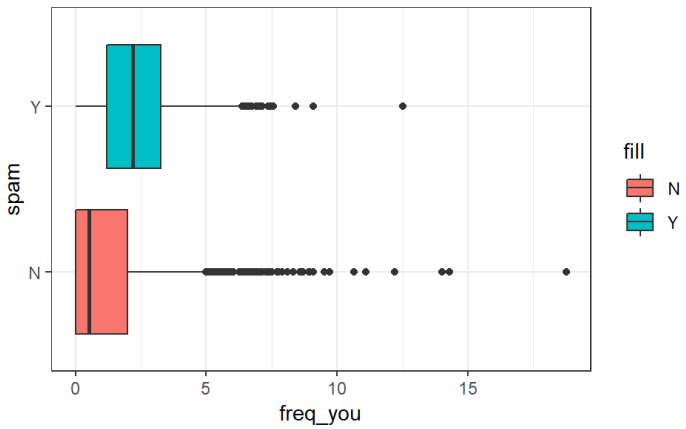

spam
Y
N
0.0
2.5
5.0
7.5
10.0
freq_report
fill
N
Y

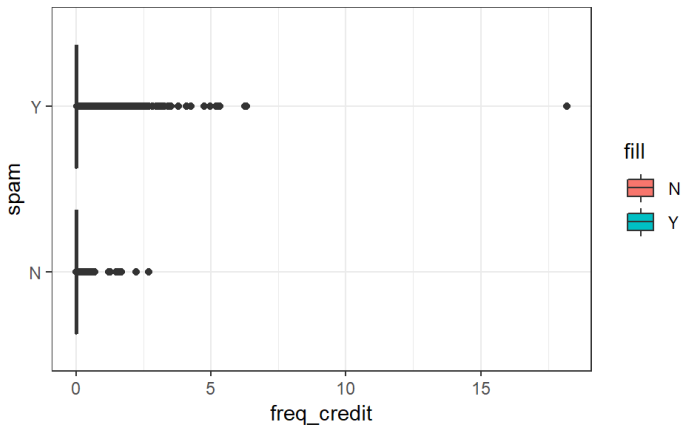

spam
Y
N
0
1
2
3
4
freq_addresses
fill
N
Y

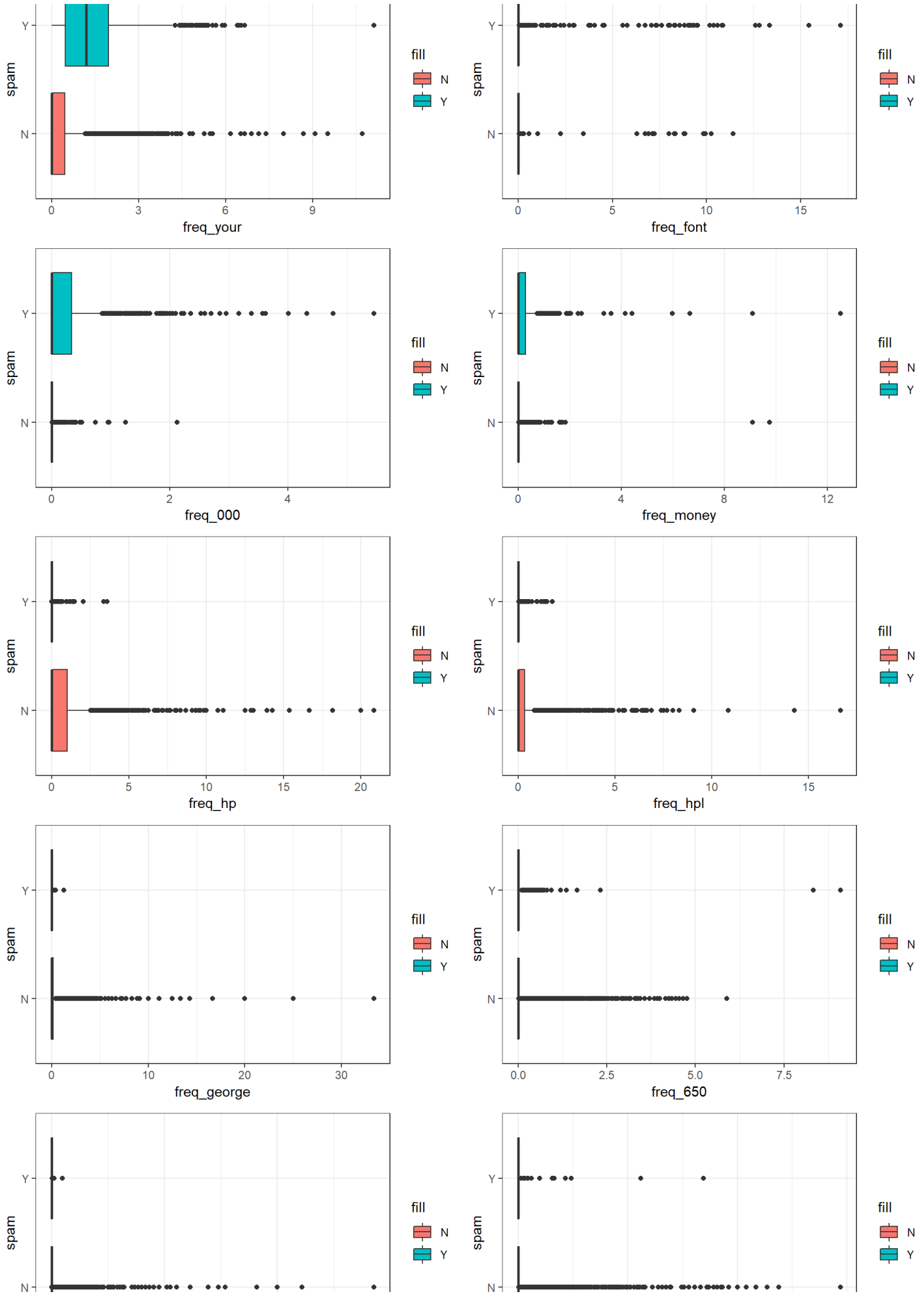

spam
Y
N
0
5
10
15
20
freq_free
fill
N
Y

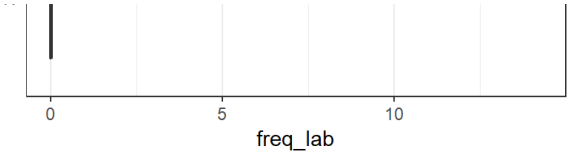

spam
Y
N
0
2
4
6
freq_business
fill
N
Y


spam
Y
N
0.0
2.5
5.0
7.5
freq_email
fill
N
Y

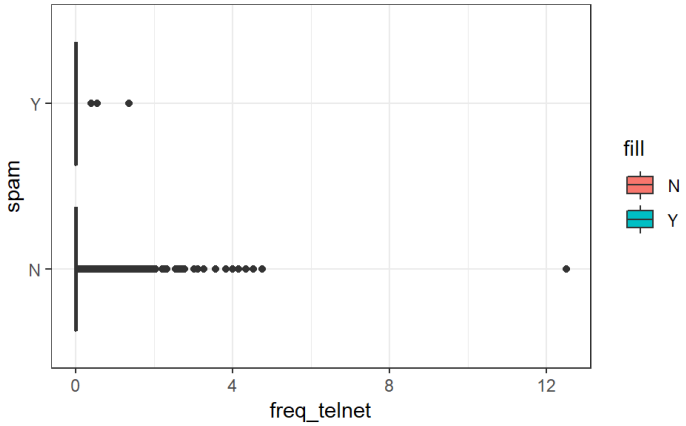

spam
Y
N
0
5
10
15
freq_you
fill
N
Y

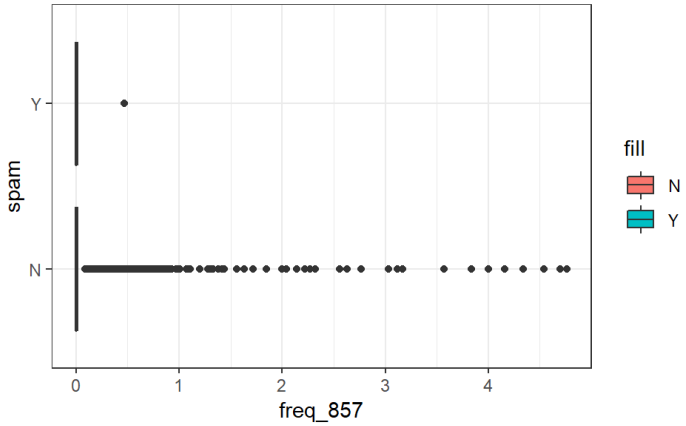

spam
Y
N
0
5
10
15
freq_credit
fill
N
Y

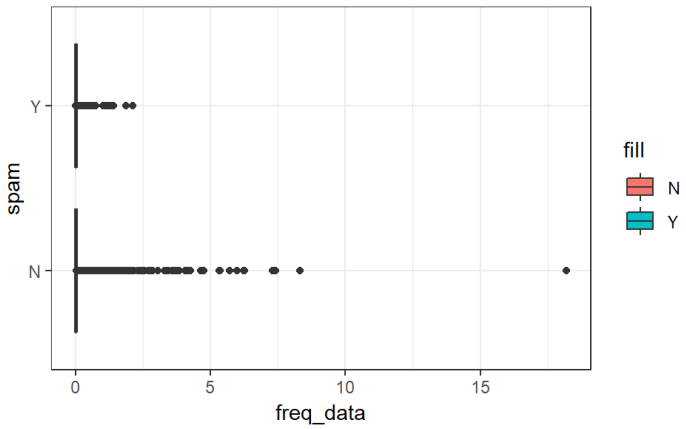

spam
Y
N
0
3
6
9
freq_your
fill
N
Y
0
5
10
15
freq_font
0
2
4
freq_000
0
4
8
12
freq_money
0
5
10
15
20
freq_hp
0
5
10
15
freq_hpl
0
10
20
30
freq_george
0.0
2.5
5.0
7.5
freq_650

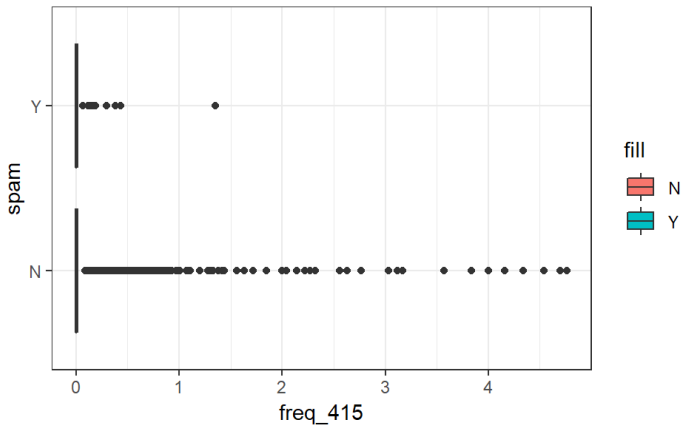

0
5
10
freq_lab

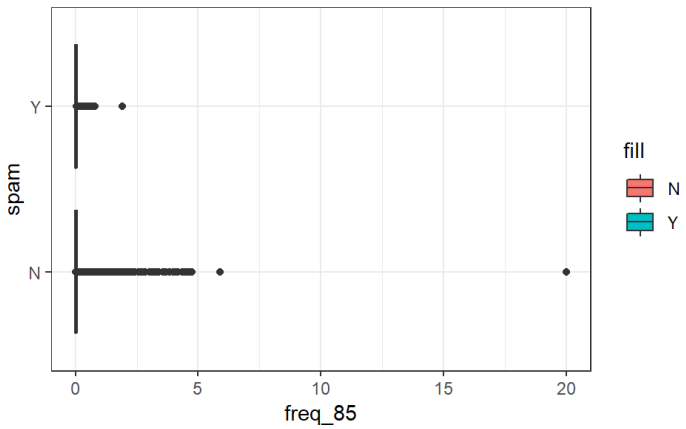

0
2
4
6
freq_labs

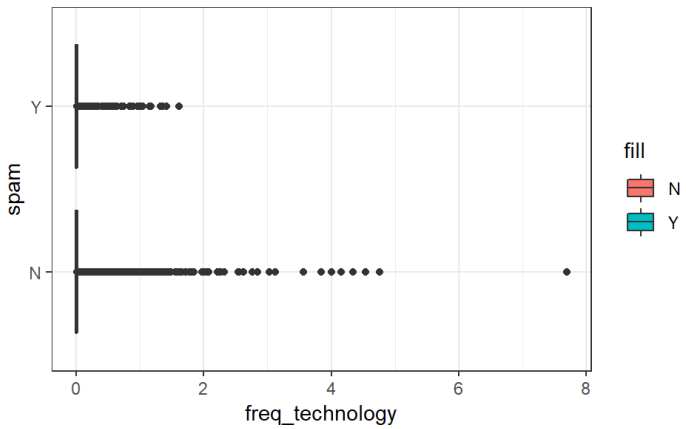

spam
Y
N
0
4
8
12
freq_telnet
fill
N
Y

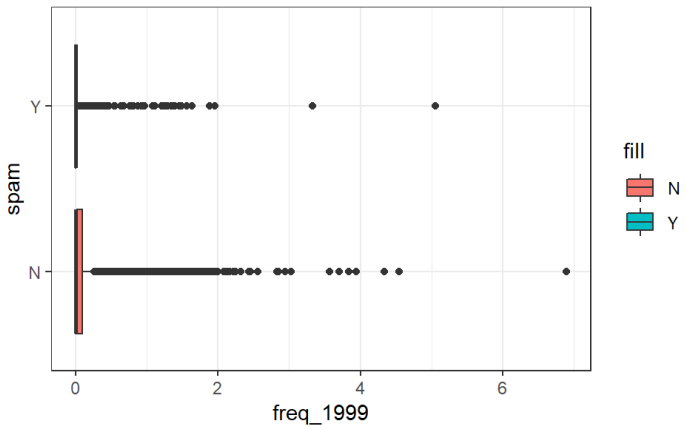

spam
Y
N
0
1
2
3
4
freq_857
fill
N
Y

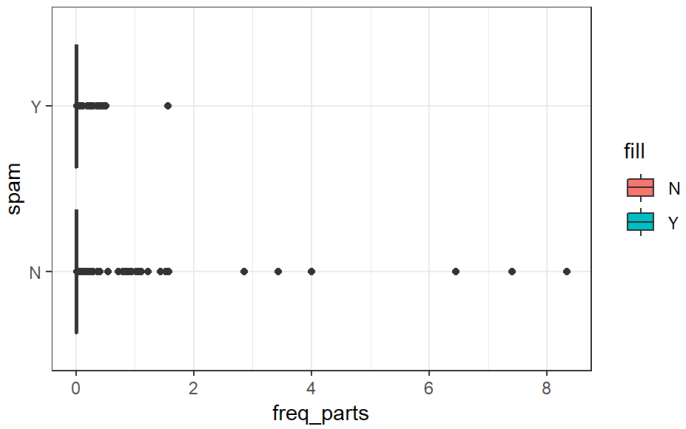

spam
Y
N
0
5
10
15
freq_data
fill
N
Y

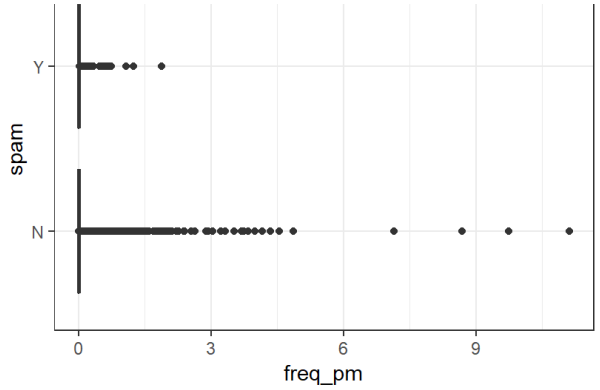

spam
Y
N
0
1
2
3
4
freq_415
fill
N
Y

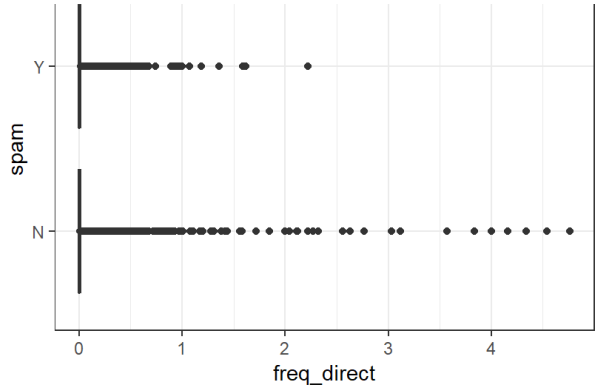

spam
Y
N
0
5
10
15
20
freq_85
fill
N
Y

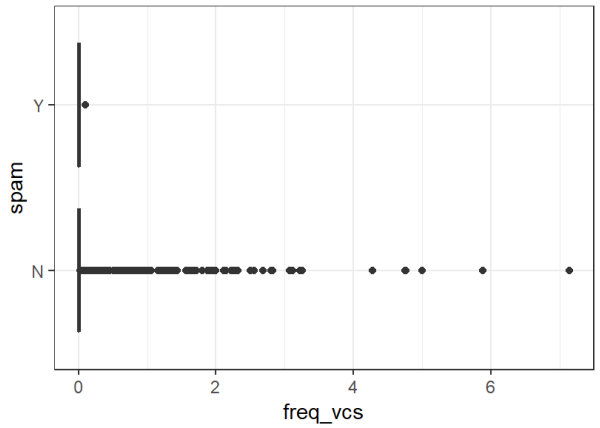

spam
Y
N
0
2
4
6
8
freq_technology
fill
N
Y

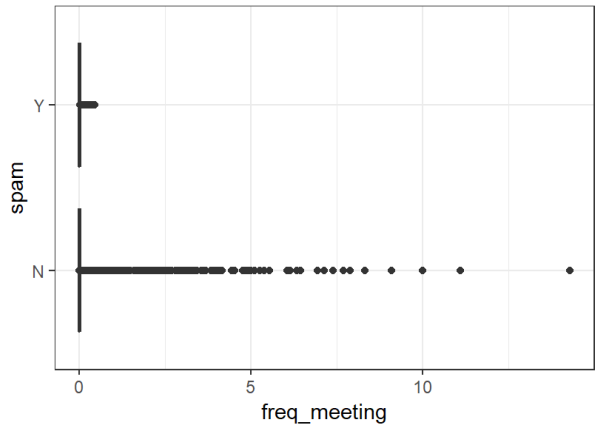

spam
Y
N
0
2
4
6
freq_1999
fill
N
Y

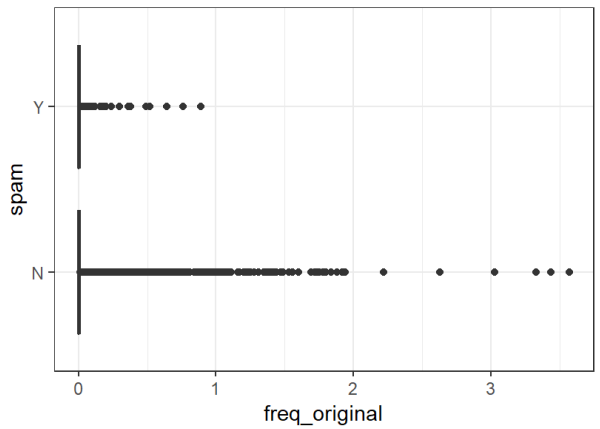

spam
Y
N
0
2
4
6
8
freq_parts
fill
N
Y

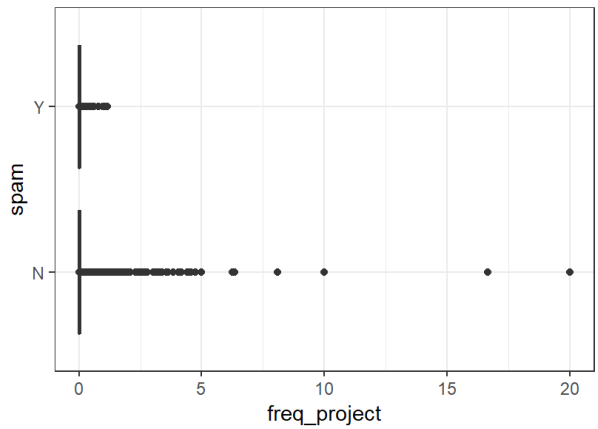

spam
Y
N
0
3
6
9
freq_pm
fill
N
Y

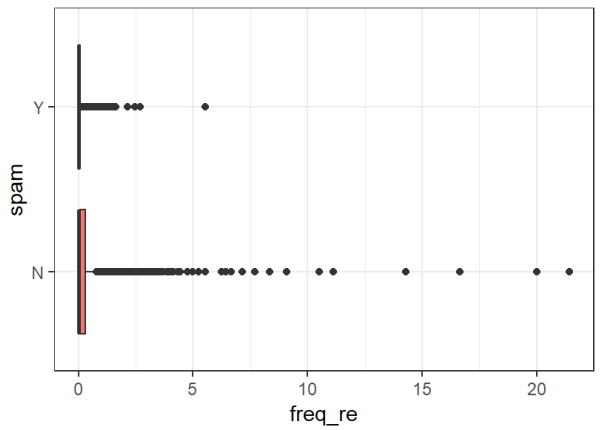

spam
Y
N
0
1
2
3
4
freq_direct
fill
N
Y

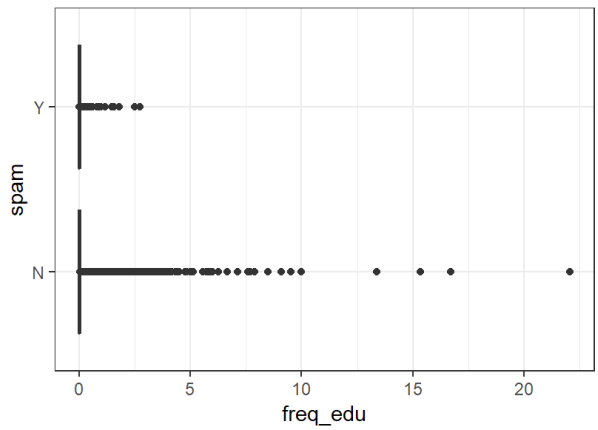

spam
Y
N
0
2
4
6
freq_vcs
fill
N
Y

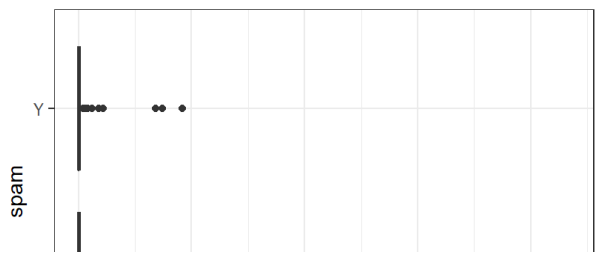

spam
Y
N
0
5
10
freq_meeting
fill
N
Y

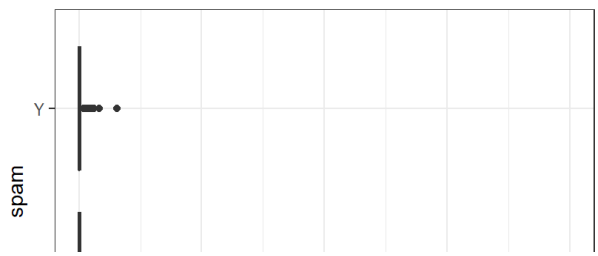

spam
Y
N
0
1
2
3
freq_original
fill
N
Y

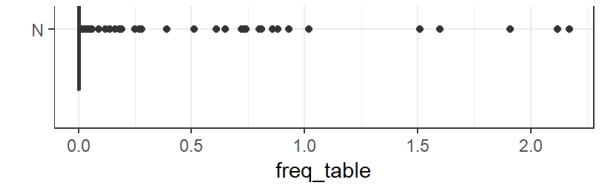

spam
Y
N
0
5
10
15
20
freq_project
fill
N
Y

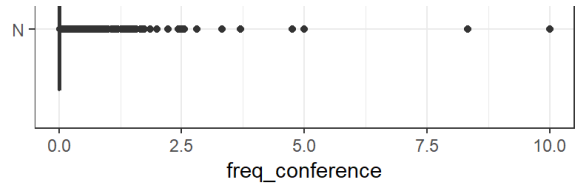

spam
Y
N
0
5
10
15
20
freq_re
fill
N
Y

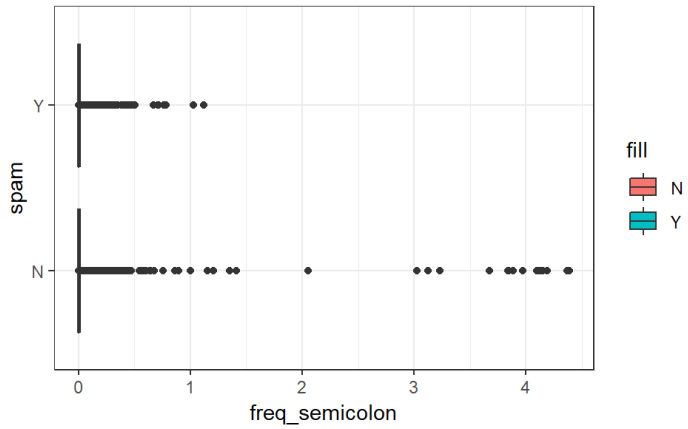

spam
Y
N
0
5
10
15
20
freq_edu
fill
N
Y

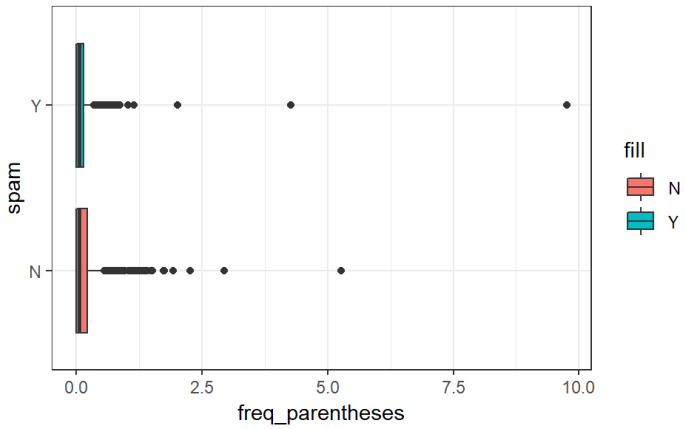

spam
Y
fill
N
Y

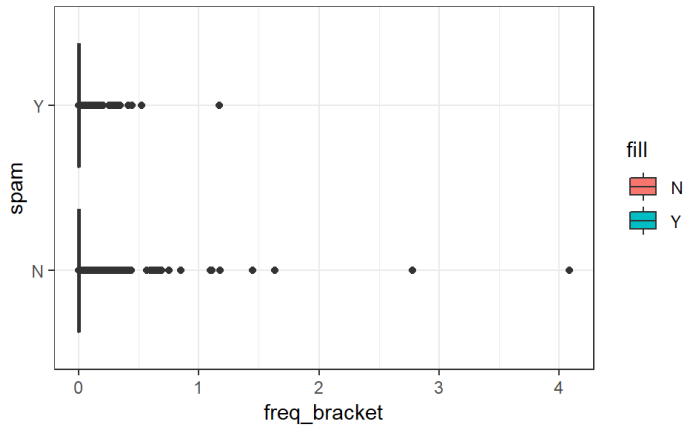

spam
Y
fill
N
Y

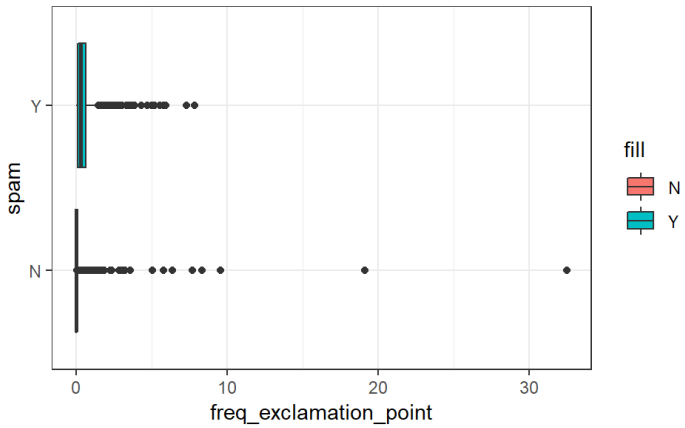

N
0.0
0.5
1.0
1.5
2.0
freq_table

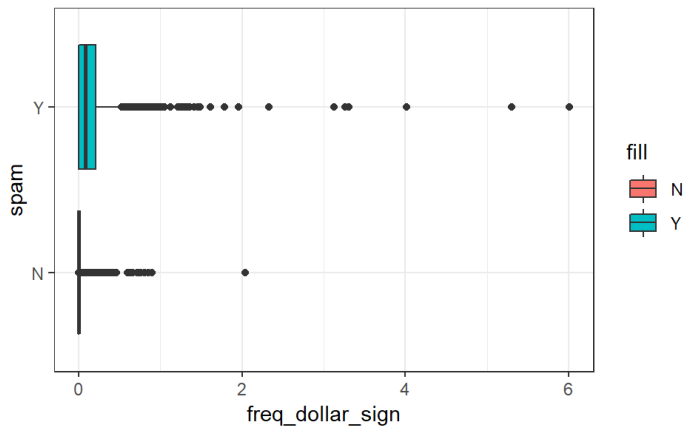

N
0.0
2.5
5.0
7.5
10.0
freq_conference

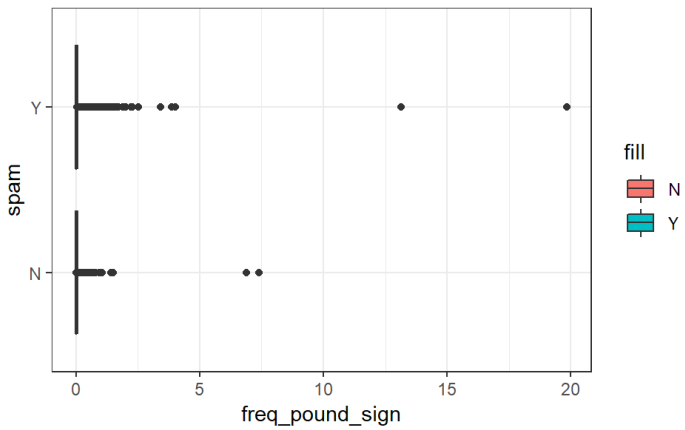

spam
Y
N
0
1
2
3
4
freq_semicolon
fill
N
Y

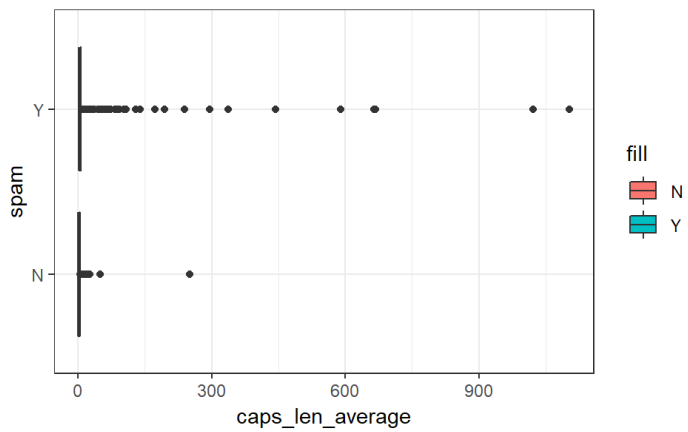

spam
Y
N
0.0
2.5
5.0
7.5
10.0
freq_parentheses
fill
N
Y

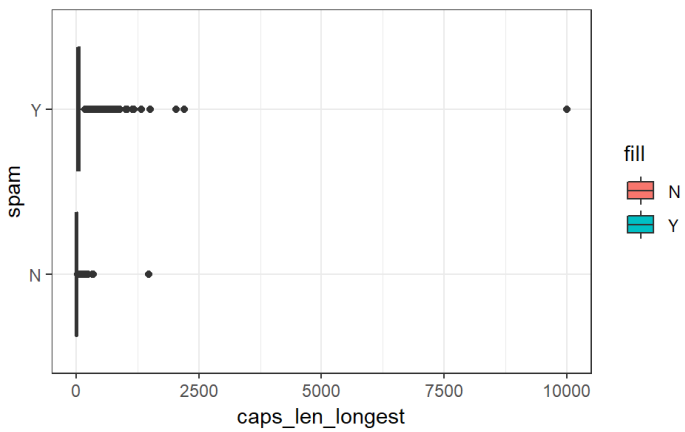

spam
Y
N
0
1
2
3
4
freq_bracket
fill
N
Y

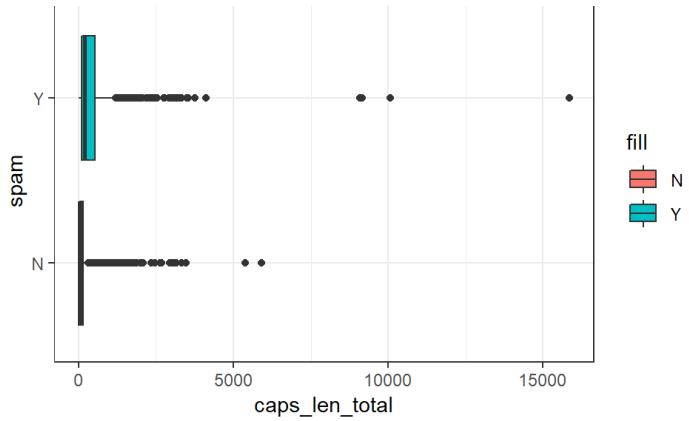

spam
Y
N
0
10
20
30
freq_exclamation_point
fill
N
Y


spam
Y
N
0
2
4
6
freq_dollar_sign
fill
N
Y


spam
Y
N
0
5
10
15
20
freq_pound_sign
fill
N
Y


spam
Y
N
0
300
600
900
caps_len_average
fill
N
Y


spam
Y
N
0
2500
5000
7500
10000
caps_len_longest
fill
N
Y



PREPROCESSING SECTION

Let's rescale the data so that all of the values are between 0 and 1.

```
#Rescale the numeric data
spamdata2 <- spamdata
rescale_x <- function(x){(x-min(x))/(max(x)-min(x))}

for (i in 2:ncol(spamdata)){
spamdata2[[i]] <- rescale_x(spamdata[[i]])
}
```

We will do a summary of the scaled data to make sure the scaling function worked as expected.

```
summary(spamdata2)
```

```
##  spam       freq_make        freq_address        freq_all
##  N:2788   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  Y:1813   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000
##           Median :0.00000   Median :0.00000   Median :0.00000
##           Mean   :0.02303   Mean   :0.01492   Mean   :0.05503
##           3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.08235
##           Max.   :1.00000   Max.   :1.00000   Max.   :1.00000
##     freq_3d            freq_our         freq_over        freq_remove
##  Min.   :0.000000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.000000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000
##  Median :0.000000   Median :0.00000   Median :0.00000   Median :0.00000
##  Mean   :0.001528   Mean   :0.03122   Mean   :0.01631   Mean   :0.01571
##  3rd Qu.:0.000000   3rd Qu.:0.03800   3rd Qu.:0.00000   3rd Qu.:0.00000
##  Max.   :1.000000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000
##  freq_internet        freq_order        freq_mail         freq_receive
##  Min.   :0.000000   Min.   :0.00000   Min.   :0.000000   Min.   :0.00000
##  1st Qu.:0.000000   1st Qu.:0.00000   1st Qu.:0.000000   1st Qu.:0.00000
##  Median :0.000000   Median :0.00000   Median :0.000000   Median :0.00000
##  Mean   :0.009477   Mean   :0.01712   Mean   :0.013169   Mean   :0.02292
##  3rd Qu.:0.000000   3rd Qu.:0.00000   3rd Qu.:0.008801   3rd Qu.:0.00000
##  Max.   :1.000000   Max.   :1.00000   Max.   :1.000000   Max.   :1.00000
##    freq_will        freq_people       freq_report       freq_addresses
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.000000   Min.   :0.00000
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.000000   1st Qu.:0.00000
##  Median :0.01034   Median :0.00000   Median :0.000000   Median :0.00000
##  Mean   :0.05602   Mean   :0.01692   Mean   :0.005863   Mean   :0.01116
##  3rd Qu.:0.08273   3rd Qu.:0.00000   3rd Qu.:0.000000   3rd Qu.:0.00000
##  Max.   :1.00000   Max.   :1.00000   Max.   :1.000000   Max.   :1.00000
##    freq_free       freq_business       freq_email         freq_you
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000
##  Median :0.00000   Median :0.00000   Median :0.00000   Median :0.06987
##  Mean   :0.01244   Mean   :0.01997   Mean   :0.02032   Mean   :0.08865
##  3rd Qu.:0.00500   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.14080
##  Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000
##   freq_credit         freq_your         freq_font           freq_000
##  Min.   :0.000000   Min.   :0.00000   Min.   :0.000000   Min.   :0.00000
##  1st Qu.:0.000000   1st Qu.:0.00000   1st Qu.:0.000000   1st Qu.:0.00000
##  Median :0.000000   Median :0.01980   Median :0.000000   Median :0.00000
##  Mean   :0.004707   Mean   :0.07289   Mean   :0.007088   Mean   :0.01865
##  3rd Qu.:0.000000   3rd Qu.:0.11431   3rd Qu.:0.000000   3rd Qu.:0.00000
##  Max.   :1.000000   Max.   :1.00000   Max.   :1.000000   Max.   :1.00000
##    freq_money          freq_hp           freq_hpl         freq_george
##  Min.   :0.000000   Min.   :0.00000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.000000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000
##  Median :0.000000   Median :0.00000   Median :0.00000   Median :0.00000
##  Mean   :0.007541   Mean   :0.02638   Mean   :0.01593   Mean   :0.02302
##  3rd Qu.:0.000000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000
##  Max.   :1.000000   Max.   :1.00000   Max.   :1.00000   Max.   :1.00000
##     freq_650          freq_lab          freq_labs         freq_telnet
##  Min.   :0.00000   Min.   :0.000000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.:0.00000   1st Qu.:0.000000   1st Qu.:0.00000   1st Qu.:0.00000
##  Median :0.00000   Median :0.000000   Median :0.00000   Median :0.00000
```

```
##  Mean   :0.01373   Mean   :0.006927   Mean   :0.01749   Mean   :0.00518
##  3rd Qu.:0.00000   3rd Qu.:0.000000   3rd Qu.:0.00000   3rd Qu.:0.00000
##  Max.   :1.00000   Max.   :1.000000   Max.   :1.00000   Max.   :1.00000
##     freq_857          freq_data           freq_415
##  Min.   :0.000000   Min.   :0.000000   Min.   :0.00000
##  1st Qu.:0.000000   1st Qu.:0.000000   1st Qu.:0.00000
##  Median :0.000000   Median :0.000000   Median :0.00000
##  Mean   :0.009884   Mean   :0.005348   Mean   :0.01005
##  3rd Qu.:0.000000   3rd Qu.:0.000000   3rd Qu.:0.00000
##  Max.   :1.000000   Max.   :1.000000   Max.   :1.00000
##     freq_85         freq_technology     freq_1999         freq_parts
##  Min.   :0.000000   Min.   :0.00000   Min.   :0.00000   Min.   :0.000000
##  1st Qu.:0.000000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.000000
##  Median :0.000000   Median :0.00000   Median :0.00000   Median :0.000000
##  Mean   :0.005271   Mean   :0.01268   Mean   :0.01988   Mean   :0.001585
##  3rd Qu.:0.000000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.000000
##  Max.   :1.000000   Max.   :1.00000   Max.   :1.00000   Max.   :1.000000
##     freq_pm          freq_direct         freq_vcs
##  Min.   :0.000000   Min.   :0.00000   Min.   :0.000000
##  1st Qu.:0.000000   1st Qu.:0.00000   1st Qu.:0.000000
##  Median :0.000000   Median :0.00000   Median :0.000000
##  Mean   :0.007077   Mean   :0.01362   Mean   :0.006116
##  3rd Qu.:0.000000   3rd Qu.:0.00000   3rd Qu.:0.000000
##  Max.   :1.000000   Max.   :1.00000   Max.   :1.000000
##   freq_meeting      freq_original      freq_project        freq_re
##  Min.   :0.000000   Min.   :0.00000   Min.   :0.00000   Min.   :0.000000
##  1st Qu.:0.000000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.000000
##  Median :0.000000   Median :0.00000   Median :0.00000   Median :0.000000
##  Mean   :0.009267   Mean   :0.01291   Mean   :0.00396   Mean   :0.014063
##  3rd Qu.:0.000000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.005135
##  Max.   :1.000000   Max.   :1.00000   Max.   :1.00000   Max.   :1.000000
##     freq_edu          freq_table       freq_conference
##  Min.   :0.000000   Min.   :0.000000   Min.   :0.000000
##  1st Qu.:0.000000   1st Qu.:0.000000   1st Qu.:0.000000
##  Median :0.000000   Median :0.000000   Median :0.000000
##  Mean   :0.008155   Mean   :0.002509   Mean   :0.003187
##  3rd Qu.:0.000000   3rd Qu.:0.000000   3rd Qu.:0.000000
##  Max.   :1.000000   Max.   :1.000000   Max.   :1.000000
##  freq_semicolon     freq_parentheses    freq_bracket
##  Min.   :0.000000   Min.   :0.000000   Min.   :0.00000
##  1st Qu.:0.000000   1st Qu.:0.000000   1st Qu.:0.00000
##  Median :0.000000   Median :0.006665   Median :0.00000
##  Mean   :0.008797   Mean   :0.014257   Mean   :0.00416
##  3rd Qu.:0.000000   3rd Qu.:0.019278   3rd Qu.:0.00000
##  Max.   :1.000000   Max.   :1.000000   Max.   :1.00000
##  freq_exclamation_point freq_dollar_sign   freq_pound_sign
##  Min.   :0.000000       Min.   :0.000000   Min.   :0.000000
##  1st Qu.:0.000000       1st Qu.:0.000000   1st Qu.:0.000000
##  Median :0.000000       Median :0.000000   Median :0.000000
##  Mean   :0.008285       Mean   :0.012629   Mean   :0.002231
##  3rd Qu.:0.009699       3rd Qu.:0.008662   3rd Qu.:0.000000
##  Max.   :1.000000       Max.   :1.000000   Max.   :1.000000
##  caps_len_average    caps_len_longest    caps_len_total
##  Min.   :0.0000000   Min.   :0.0000000   Min.   :0.000000
```

```
##  1st Qu.:0.0005338   1st Qu.:0.0005006   1st Qu.:0.002146
##  Median :0.0011584   Median :0.0014017   Median :0.005934
##  Mean   :0.0038053   Mean   :0.0051234   Mean   :0.017821
##  3rd Qu.:0.0024567   3rd Qu.:0.0042050   3rd Qu.:0.016730
##  Max.   :1.0000000   Max.   :1.0000000   Max.   :1.000000
```

The scaling function worked. Let's replace spamdata with the scaled spamdata2

```
spamdata_unscaled <- spamdata
spamdata <- spamdata2
```

INITIAL MODELS SECTION

Define our constants

```
SEED <- 0 # random seed
REPS <- 50 # training replications
RATIO <- 0.6 # train-test split ratio
NTREE <- 100 # number of trees
```

Define function for the common training tasks.

```
##############################################################
# Generic wrapper function for training and prediction
#
# @param name
#     name of the model e.g. "Decision Tree"
# @param build_model
#     function to perform the model-specific training
#     e.g. build_model <- function(data) {
#               return(rpart(Y ~ ., data = data))
#          }
#
# @param split_ratio
#     train to test split ratio e.g. 0.75
#
# @param predict_call
#     variable to control how predict is called
#
# @returns
#     list containing the error % for each rep and the
#     model for each rep
#
##############################################################

train_wrapper <- function(name,
                          build_model,
                          split_ratio = 0.6,
                          predict_call = 1) {
  # set random seed to use same data splits
  set.seed(SEED)

  # vector to store error for each repetition
  errors <- rep(0, REPS)

  # vector to store models for each repetition
  models <- vector(mode = "list", length = REPS)

  # vector to store yhat predictions for each repetition
  yhats <- vector(mode = "list", length = REPS)

  # vector to store test sets
  tests <- vector(mode = "list", length = REPS)

  for (r in 1:length(errors)) {
    # split data for train and test
    id = holdout(spamdata$spam, ratio=split_ratio, mode='stratified')
    train <- spamdata[id$tr,]
    test <- spamdata[id$ts,]
    tests[[r]] <- test

    # build model on training set
    model <- build_model(train)

    # store model
```

```
    models[[r]] <- model

    # predict on test set
    yhat <- switch (predict_call,
                    predict(model, test %>% dplyr::select(-spam), type = 'class'),
                    predict(model, test %>% dplyr::select(-spam)),
                    predict(model, test %>% dplyr::select(-spam))$class)

    # store predictions
    yhats[[r]] <- yhat

    # store error
    errors[r] <- mean(yhat != test$spam)

    # print progress
    cat(name, "[rep]:", r, "[error]:", errors[r], "\n")
  }

  # print confusion matrix for most accurate model
  index <- which.min(errors)
  print(confusionMatrix(table(yhats[[index]], tests[[index]]$spam)))

  return(list("errors" = errors, "models" = models))
}
```

Regular Trees

```
build_regular_tree <- function(training_data) {
  rpart(spam ~ ., data = training_data)
}

regular_tree_errors <- train_wrapper("Regular Trees", build_regular_tree)
```

```
#store regular tree error data as RData files
save(regular_tree_errors, file='regular_tree_errors.rdata')
```

Bagged Trees

Use all variables at each split (mtry = p where p = number of predictors).

```
build_bagged_tree <- function(training_data) {
  model <- randomForest(spam ~ .,
                        data = training_data,
                        mtry = ncol(training_data) - 1,
                        ntree = NTREE)
  return(model)
}

bagged_tree_errors <- train_wrapper("Bagged Trees", build_bagged_tree)
```

```
#store bagged tree error data as RData files
save(bagged_tree_errors, file='bagged_tree_errors.rdata')
```

Random Forest

Use random subset of variables at each split (mtry = square root of p).

```
build_rf <- function(training_data) {
  model <- randomForest(spam ~ .,
                        data = training_data,
                        mtry = sqrt(ncol(training_data) - 1),
                        ntree = NTREE)
  return(model)
}

rf_errors <- train_wrapper("Random Forest", build_rf)
```

```
#store random forest error data as RData files
save(rf_errors, file='rf_errors.rdata')
```

Weighted Random Forest

Note: the selected weights were chosen through trial and error, thus may be overfitted to this particular random seed.

```
build_weighted_rf <- function(training_data) {
  model <- randomForest(spam ~ .,
                        data = training_data,
                        ntree = NTREE,
                        classwt = c(0.3, 0.8))
  return(model)
}

weighted_rf_errors <- train_wrapper("Weighted Random Forest", build_weighted_rf)
```

```
#store weighted random forest error data as RData files
save(weighted_rf_errors, file='weighted_rf_errors.rdata')
```

Balanced Random Forest

Use under/down sampling based on size of minority class.

```
# minority class is 'Y'
build_balanced_rf <- function(training_data) {
  minority_class_size <- sum(spamdata$spam=='Y') # min(table(training_data$spam))

  model <- randomForest(spam ~ .,
                        data = training_data,
                        ntree = NTREE,
                        sampsize = minority_class_size)
  return(model)
}

balanced_rf_errors <- train_wrapper("Balanced Random Forest", build_balanced_rf)
```

```
#store balanced random forest error data as RData files
save(balanced_rf_errors, file='balanced_rf_errors.rdata')
```

LDA

```
build_lda <- function(training_data) {
  model <- lda(spam ~ ., data = training_data)
  return(model)
}

lda_errors <- train_wrapper("LDA",
                            build_lda,
                            split_ratio = RATIO,
                            predict_call = 3)$errors
```

```
#store LDA error data as RData files
save(lda_errors, file='lda_errors.rdata')
```

C-SVM

```
build_csvm <- function(training_data) {
  model <- svm(spam ~ ., data = training_data, type = "C-classification")
  return(model)
}

csvm_errors <- train_wrapper("C-SVM",
                             build_csvm,
                             split_ratio = RATIO)$errors
```

```
#store C-SVM error data as RData files
save(csvm_errors, file='csvm_errors.rdata')
```

nu-SVM

```
build_nusvm <- function(training_data) {
  model <- svm(spam ~ ., data = training_data, type = "nu-classification")
  return(model)
}

nusvm_errors <- train_wrapper("nu-SVM",
                              build_nusvm,
                              split_ratio = RATIO)$errors
```

```
#store C-SVM error data as RData files
save(nusvm_errors, file='nusvm_errors.rdata')
```

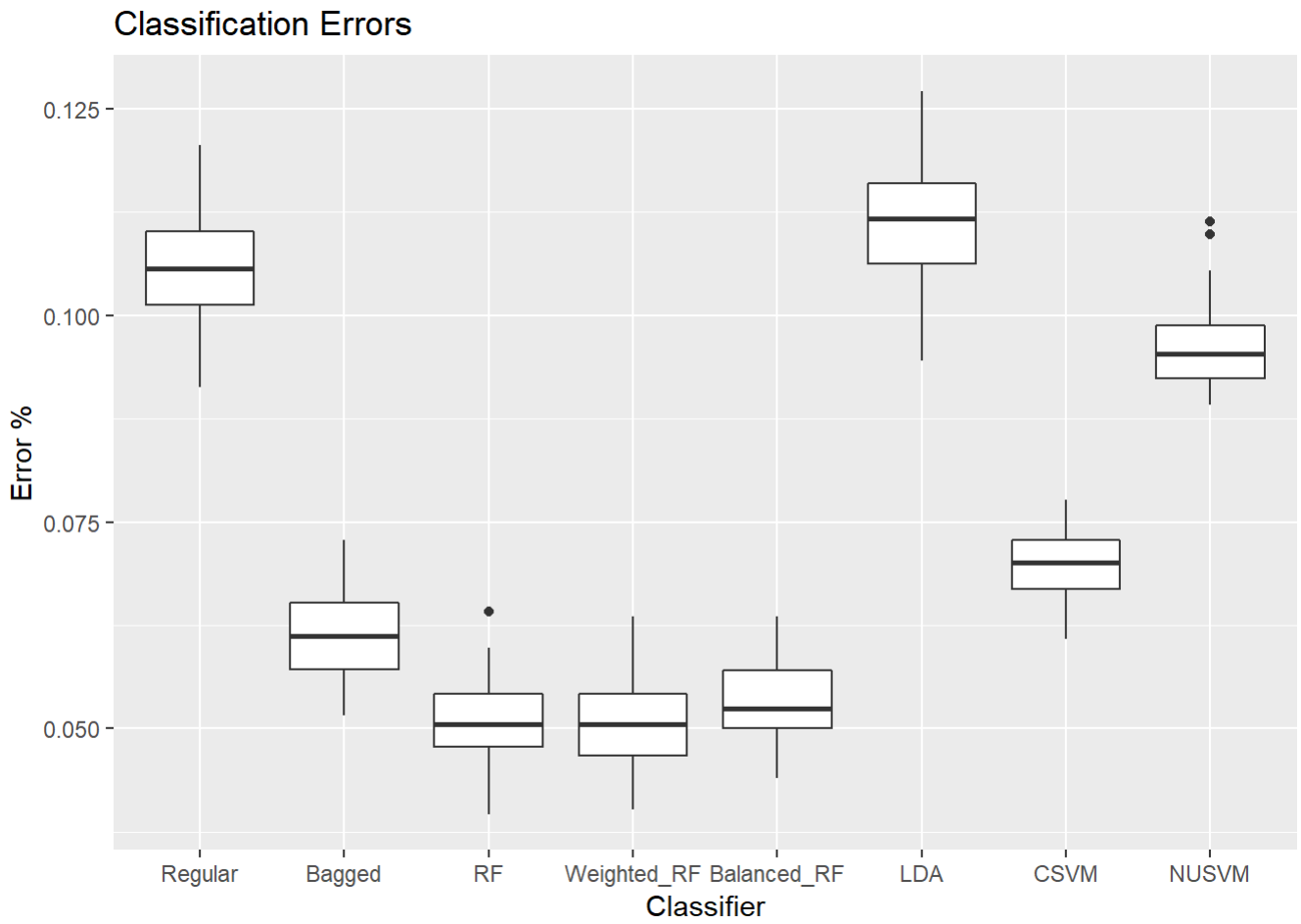
```
#Load all of the error data
load(file='regular_tree_errors.rdata')
load(file='bagged_tree_errors.rdata')
load(file='rf_errors.rdata')
load(file='weighted_rf_errors.rdata')
load(file='balanced_rf_errors.rdata')
load(file='lda_errors.rdata')
load(file='csvm_errors.rdata')
load(file='nusvm_errors.rdata')
```

Compare Results

Combine error results into comparative boxplot.

```
# aggregate to data frame
combined_errors <- as.tibble(data.frame(
  Regular = regular_tree_errors$errors,
  Bagged = bagged_tree_errors$errors,
  RF = rf_errors$errors,
  Weighted_RF = weighted_rf_errors$errors,
  Balanced_RF = balanced_rf_errors$errors,
  LDA = lda_errors,
  CSVM = csvm_errors,
  NUSVM = nusvm_errors))
```

```
# plot errors
combined_errors %>%
  melt(measure.vars = c("Regular", "Bagged", "RF", "Weighted_RF", "Balanced_RF", "LDA", "CSVM",
                        "NUSVM")) %>%
  ggplot(aes(x = variable, y = value)) + geom_boxplot() + ggtitle("Classification Errors") +
  xlab("Classifier") + ylab("Error %")
```



As we can see from the above boxplots, we have Weighted Random Forest as our best performing model, with a very small advantage over a regular Random Forest. I believe there may be some room for optimization in our models, so we will attempt optimization for both the Weighted and regular Random Forest. We will also attempt optimization for the C-SVM model just so we can see if optimization impacts support vector machine model accuracy significantly.

MODEL OPTIMIZATION SECTION

RANDOMFOREST OPMTIMIZATION

Try to optimize the RandomForest model using tuneLength from the caret package. As FYI, this tuning model will take about 5 - 10 minutes to run.

```
# set control
control <- trainControl(method = 'cv', number = 5)
# retrain the model
RF_tune_model <- train(spam ~ ., data = spamdata, method = 'rf', trControl = control,
                       tuneLength = 5)
```

```
#store the tuned random forest model data
save(RF_tune_model, file='RF_tune_model.rdata')
```

```
#load the tuned random forest model data
load(file='RF_tune_model.rdata')
```

```
# summarize the model
print(RF_tune_model)
```

```
## Random Forest
##
## 4601 samples
##   57 predictor
##    2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 3680, 3681, 3682, 3681, 3680
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9426179  0.8786202
##   15    0.9502259  0.8952354
##   29    0.9484877  0.8917584
##   43    0.9447925  0.8839576
##   57    0.9450104  0.8844086
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 15.
```

```
# calculate accuracy of the 'regular' random forest run earlier
mean(rf_errors$errors)
```

```
## [1] 0.051
```

```
1 - mean(rf_errors$errors)
```

```
## [1] 0.949
```

Our regular Random Forest ran with mtry = sqrt(ncol(training_data -1)) = sqrt(57) = 7.55, and that resulted in an accuracy of approximately 0.9494. With our parameter tuning, we see that mtry = 15 is the best choice with a resulting accuracy of 0.9528.

WEIGHTED RANDOMFOREST OPMTIMIZATION

Try to optimize the Weighted RandomForest model using tuneLength from the caret package, using the method ‘wsrf’ for Weighted Subspace Random Forest. As FYI, the weighted RF tuning model will take about 20 - 40 minutes to run.

```
# set control
control <- trainControl(method = 'cv', number = 5)
# retrain the model
Weight_RF_tune_model <- train(spam ~ ., data = spamdata, method = 'wsrf', trControl = control,
                       tuneLength = 5)
```

```
#store the tuned random forest model data
save(Weight_RF_tune_model, file='Weight_RF_tune_model.rdata')
```

```
#load the tuned random forest model data
load(file='Weight_RF_tune_model.rdata')
```

```
# summarize the model
print(Weight_RF_tune_model)
```

```
## Weighted Subspace Random Forest
##
## 4601 samples
##   57 predictor
##    2 classes: 'N', 'Y'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 3681, 3681, 3681, 3681, 3680
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9515345  0.8977854
##   15    0.9411028  0.8756662
##   29    0.9419726  0.8774942
##   43    0.9404504  0.8743092
##   57    0.9413202  0.8761141
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

Our results do worse than the orignal weighted random forest, which tells me that maybe the better way to tune a weighted random forest is to experiment with the weights. Let’s give that a try:

```
#  Create a sequence to try some values
v.w1 = c(0.01, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1)
v.w2 = c(0.01, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1)
cv.for.w = matrix(0, ncol=length(v.w1), nrow=length(v.w2))
colnames(cv.for.w) = v.w1
rownames(cv.for.w) = v.w2

# set random seed to use same data splits
set.seed(SEED)
id = holdout(spamdata$spam, ratio=.6, mode='stratified')
sd.tr = spamdata[id$tr,]
sd.te = spamdata[id$ts,]

for(j in 1:length(v.w1))
  {
  for(i in 1:length(v.w2))
    {
    # loop through each value of w to try
    w.wrf.xy = randomForest(spam~., data=sd.tr, ntree = 50, classwt=c(v.w1[j],v.w2[i]))

    # get the cross validation error for each w value
    yhat_twrf = predict(w.wrf.xy, sd.te[,-1])
    cv.for.w[j,i] = mean(yhat_twrf!=sd.te[,1])
    }
  }

# find the optimal weight values
twrf.opt.err = min(cv.for.w)
min.err = which(cv.for.w == twrf.opt.err, arr.ind = TRUE)
w1.opt = v.w1[min.err[1]]
w2.opt = v.w2[min.err[2]]

1-twrf.opt.err
```

```
## [1] 0.9505435
```

Compare to original

```
# calculate accuracy of the 'regular' weighted random forest run earlier
mean(weighted_rf_errors$errors)
```

```
## [1] 0.05096739
```

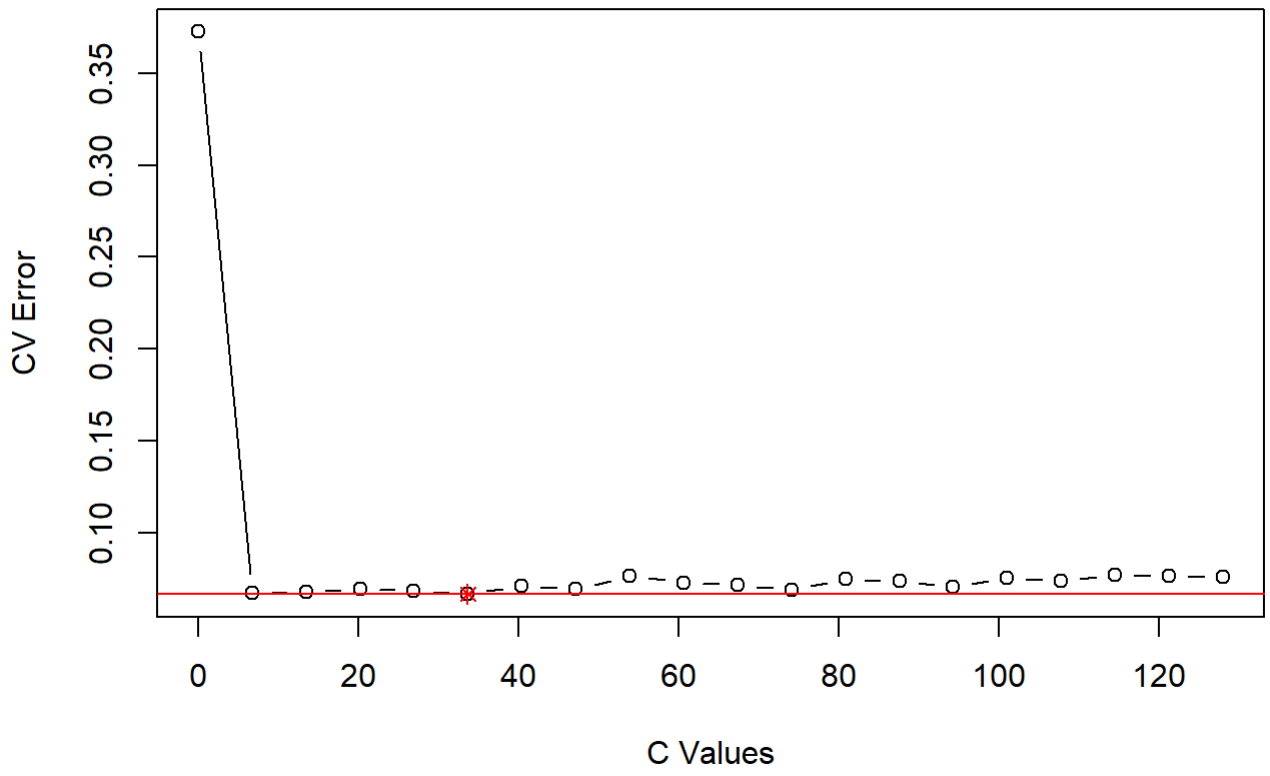
```
1 - mean(weighted_rf_errors$errors)
```

```
## [1] 0.9490326
```

C-SVM OPMTIMIZATION

Try to optimize the C-SVM model “by hand”

```
#Cross Validating your C - value
# Number of C to observe
n.c = 20

#  Create a sequence to try out 20 values between 2^-7 and 2^7
v.c = seq(2^(-7),2^7, length=n.c)
cv.for.c = numeric(n.c)

for(j in 1:n.c)
{
  # loop through each value of C to try
  c.svm.xy = ksvm(spam~., data=spamdata, cross=5, C=v.c[j],
                  type='C-svc')

  # get the cross validation error for each C value
  cv.for.c[j] = cross(c.svm.xy)
}

# find the optimal C value
c.opt = v.c[min(which(cv.for.c==min(cv.for.c)))]

# plot the CV error for the C values
# the optimal value is in red with a line to show it's the lowest value
plot(x=v.c, y=cv.for.c, xlab='C Values', ylab='CV Error',
     main='SVM Optimization',type='b')
points(x=c.opt,y=min(cv.for.c),col='red',pch=8)
abline(h=min(cv.for.c),col='red')
```



Let's say 27 is our optimum value of C based on the plot above

```
# Use optimal value of C below to re-run SVM
csvm_spam =   ksvm(spam~., data=spamdata, cross=5, C= 27,
                   type='C-svc')

y_hat_csvm = predict(csvm_spam, spamdata[,-1])

1-mean(y_hat_csvm!=spamdata[,1])
```

```
## [1] 0.9843512
```

```
# calculate accuracy of the 'regular' C-SVM run earlier
mean(csvm_errors)
```

```
## [1] 0.06969565
```

```
1 - mean(csvm_errors)
```

```
## [1] 0.9303043
```

Using an optimized c-svm we get an error rate of 1.69% for an accuracy of 98.31%, which well surpasses our non-optimized c-svm accuracy of 93.91%. This makes an optimized c-svm with C = 27 as our best performing model.

FIGURES & CONCLUSION SECTION

COMPARISON

Plot a comparison of the base and optimized models

```
mod_names = c('RF', 'RF_Opt',
              'WRF', 'WRF_Opt',
              'CSVM', 'CSVM_Opt')
mod_acrcy = c(1 - mean(rf_errors$errors), 0.9502259,
              1 - mean(weighted_rf_errors$errors), 1-twrf.opt.err,
              1 - mean(csvm_errors), 1 - mean(y_hat_csvm!=spamdata[,1]))

mod_type = c(1,1,2,2,3,3)

err_comp = data.frame(cbind(mod_names, mod_acrcy, mod_type))

ggplot(err_comp, aes(x = mod_names, y = mod_acrcy, fill = mod_type)) + geom_col()
```