STA 141B Assignment 2

Suvethika Kandasamy

2024-05-03

My approach to tackling this assignment began with the creation of a function to generate lists of headers, bodies, and attachments for individual files. This function could then be replicated and easily modified for various files. Initially, I focused on constructing a named vector for the headers. An early challenge arose from categories spanning multiple lines, requiring me to collapse these lines into a single one before separating the category's information and name. My first method encountered difficulties, so I opted for a simpler approach. I labeled the beginnings and ends of lines, particularly marking lines starting with tabs or spaces for collapsing into the previous line. I created unique markers for each part: the beginning of the line, the end of the line, and the tab, which didn't already exist in the files in that combination. After collapsing and separating them into proper lines, I extracted the names from the rest of the text, creating a name vector.

Next, I addressed the identification of attachments. For files without attachments, the code was straightforward, simply assigning the remaining lines as the body. I began by checking for the presence of a boundary string in the content type. I isolated the boundary string and used grep() to find which lines it was present on. I then created a function to separate attachments into a list between those line markers, and then read in the header and body in the same way as I would a regular file.

When applying the function to all files in "easy ham," I encountered another issue with a specific file where there wasn't a quotation before the boundary string. I updated the regular expression in the grep() function to recognize this boundary string variation. I identified the problematic file, investigated its structure, and adapted the function to accept both quotation variations.

After addressing this issue, everything ran smoothly when processing emails. However, I noticed that one email had an empty body. I investigated and was able to extract the body and attachments accurately. However, when running the function in the final draft version, it failed to process the email. I realized my mistake lay in the separation function for attachments. When copying it over to my final version, I omitted a crucial part of the function. Adding this fixed the issue.

Another problem I encountered was a missing value error when processing files. I identified the file causing this issue and investigated its structure using my original code version. The problem was with the boundary string, which had parentheses. I addressed this recurring problem by using gsub() to add '| before special characters.

My function worked well on all of "easy_ham" and "easy_ham_2," but I faced an issue when running it on "hard ham." A specific file claimed the last line was incomplete. I added an if statement after readlines to address such cases. I encountered another problem with a file containing a boundary string with spaces. I adjusted the gsub() to remove only trailing spaces.

In another instance, one attachment contained its own boundary strings because it had an attachment within an attachment, causing my function to break. I adapted the regular expression in grep() to handle this.

Finally, my function failed for two files in the spam folder because the boundary string differed between the body and header. I created an if statement to address this difference. The body version had a space after the equal sign while in the header version there was no space. I decided to create an if statement to add a space after the equal sign and see if that worked and if it didn't then continue trying other ways to

extract the boundary lines. I kept them as boundary lines because it seemed like an error and was clearly an attachment. If adding that space didn't solve the problem then it probably wasn't an attachment. A lot of the issues I faced were due to the boundary string detection related. The general method I used to debug is I have separate R script that prints each step out so I exactly identify where in code that there is an issue. I find which file is causing the problem and able to try different methods to fix the part that is causing the issue.After going in a debugging each file I was able to adapt my function to handle various cases.

When creating variables, I made a function to read emails, extract data, and return it as a vector, facilitating later addition to a data frame.

To streamline the process, I first read all the ham files and inputted that information into the data frame. Then, I added a data frame for spam before combining them for easier labeling.

I cross-validated all columns to ensure they had the same number of observations. While looking through the variables there are some that stood out. For example when getting the hour sent I was getting some NAs and after further inspection there were issues with the emails header due to missing colons so my function wasn't able to accurately read those files. I left them in my data frame so the rows align up and at the end. In the future, I would removed rows that contained NAs in that column. It's often better to handle data cleaning and preprocessing as early as possible in your analysis pipeline. This ensures that your data is clean and consistent throughout your analysis. Correcting formatting issues, imputing missing values, or removing problematic rows early on can lead to more reliable results.

```
#method to compress mulitple header lines into one
add_line_markers <- function(lines) {
  for (i in 1:length(lines)) {
    if (!grepl("^)))))))))", lines[i])) {
      lines[i] <- paste0("&&&&&", lines[i], " &&&&&")
    }
  }
  return(lines)
}


separate_attachments <- function(boundarylinenum, attachments_and_body) {
  # Initialize a list to store attachments
  attachments <- list()
  # Loop through boundary lines to separate attachments
  for (i in 1:(length(boundarylinenum) - 1)) {
    start <- boundarylinenum[i] + 1
    end <- boundarylinenum[i + 1] - 1
    if (end >= start) {
         attachments[[i]] <- attachments_and_body[start:end]
       }
  }
  # Return only the attachments list
  return(attachments)
}


#processing attachments to create list for header and body
process_attachments <- function(attachments) {
  processed_attachments <- list()

  for (attachment in attachments) {
    # Find the index of the first blank line
    blank_line_index <- which(attachment == "")
```

```
    # Find the index of the header
    header_index <- blank_line_index[1] - 1
    header_index

    # Extract header lines
    headerlines <- attachment[1:header_index]

    # Handle tabs and spaces in the beginning
    tablines <- gsub("^%s+|\\t|^[ \t]+","))))))))", headerlines)

    # Add markers to the lines
    result <- add_line_markers(tablines)

    # Collapse lines into a single string
    big_string <- paste(result, collapse = "")

    # Remove line markers
    col <- gsub("&&&&&)))))))|))))))))", "", big_string)

    # Split the string based on markers
    separated <- strsplit(col, "&&&&&")

    # Remove empty elements
    entireheader <- separated[[1]][separated[[1]] != ""]

    # Extract the header names
    matched_sections <- regmatches(entireheader, regexpr("^\\w+: |^\\w+-\\w+: |^\\w+-\\w+-\\w+: |^\\w+-
    names <- gsub(": ", "", matched_sections)

    # Extract the content
    content <- gsub("^\\w+: |^\\w+-\\w+: |^\\w+-\\w+-\\w+: |^\\w+-\\w+-\\w+-\\w+: ", "", entireheader)

    # Create named vector for header
    headernamedvector <- setNames(content, names)

    # Extract the body of the attachment
    bodyofattachment <- attachment[blank_line_index[1]:length(attachment)]

    # Combine header and body
    attachment_info <- list(header = headernamedvector, body = as.vector(bodyofattachment))

    # Add to processed attachments list
    processed_attachments <- c(processed_attachments, list(attachment_info))
  }

  return(processed_attachments)
}

extracting_header <- function(headerlines) {
  # Handle tabs and spaces in the beginning
  tablines <- gsub("^%s+|\\t|^[ \t]+","))))))))", headerlines)

  # Add markers to the lines
```

```
  result <- add_line_markers(tablines)

  # Collapse lines into a single string
  big_string <- paste(result, collapse = "")

  # Remove line markers
  col <- gsub("&&&&&)))))))|)))))))", "", big_string)

  # Split the string based on markers
  separated <- strsplit(col, "&&&&&")

  # Remove empty elements
  entireheader <- separated[[1]][separated[[1]] != ""]

  # Extract the header names
  matched_sections <- regmatches(entireheader, regexpr("^\\w+: |^\\w+-\\w+: |^\\w+-\\w+-\\w+: |^\\w+-\\
  names <- gsub(": ", "", matched_sections)

  # Extract the content
  content <- gsub("^\\w+: |^\\w+-\\w+: |^\\w+-\\w+-\\w+: |^\\w+-\\w+-\\w+-\\w+: ", "", entireheader)

  # Create named vector for header
  headernamedvector <- setNames(content, names)

  return(headernamedvector)
}


process_email <- function(file_path) {
  # Read lines from the file
  lines <- readLines(file_path)

  #if the last line is incomplete
  if (nchar(lines[length(lines)]) == 0) {
    # Remove the incomplete last line
    lines <- lines[-length(lines)]
    # Add a newline character to the end
    lines <- c(lines, "")
  }

  # Find the index of the first blank line
  blank_line_index <- which(lines == "")

  # Find the index of the header
  header_index <- blank_line_index[1] - 1

  # Extract header lines
  headerlines <- lines[2:header_index]
  finalheader <- extracting_header(headerlines)

  # Extract the body of the email
  body <- lines[blank_line_index[1]:length(lines)]
  #headerindex[1]:length(lines)
```

```
# Check for attachments
if (grepl("boundary=", finalheader["Content-Type"])) {
  isolatedboundary <- regmatches(finalheader["Content-Type"], regexpr("boundary=\"(.*?)\"|boundary=([
  boundarystring <- gsub("\"|^boundary=", "", isolatedboundary)
  boundarystring <- gsub("([\\{\\}\\(\\)\\+\\*\\?\\^\\.$])", "\\\\\\1", boundarystring)
  boundarystring <- gsub("\\s+$", "", boundarystring)
  boundarystring <- gsub("^\\s+", "", boundarystring)
  boundarylinenum <- grep(paste0(boundarystring, "\\b"), body)
  if (length(boundarylinenum) ==0){
    boundarylinenum <- grep(boundarystring, body)
  }

  # case to handle those two files with a error in the boundary string
  if (length(boundarylinenum) == 0){
    boundarystring <- gsub("=", "= ", boundarystring)
    boundarylinenum <- grep(paste0(boundarystring, "\\b"), body)
    if (length(boundarylinenum) ==0){
      boundarylinenum <- grep(boundarystring, body)
    }

  #another method to extract boundary string
  if (length(boundarylinenum) == 0){
    # Get the Content-Type header value
    content_type <- headernamedvector["Content-Type"]
    # Split the header to extract the boundary
    boundary_split <- strsplit(content_type, ";\\s+")[[1]]
    # Find the part of the split that contains 'boundary'
    boundary_part <- grep("boundary=", boundary_split, value = TRUE)
    # Extract the boundary string
    boundarystring <- gsub("^boundary=", "", boundary_part)
    boundarystring <- gsub("([\\{\\}\\(\\)\\+\\*\\?\\^\\.$])", "\\\\\\1", boundarystring)
    boundarystring <- gsub("\\s+$", "", boundarystring)
    boundarystring <- gsub("^\\s+", "", boundarystring)
    #boundarystring <- gsub("^\\s+|\\s+$", "", boundarystring)
    boundarylinenum <- grep(paste0(boundarystring, "\\b"), body)
    if (length(boundarylinenum) ==0){
      boundarylinenum <- grep(boundarystring, body)
    }
  }

  }

  if (length(boundarylinenum) == 1){
    boundarylinenum <- c(boundarylinenum,length(body))
  }
  #boundarylinenum <- c(boundarylinenum,length(body)-1)

  # Extracting lines before the first line of boundarylinenum
  body_before <- body[1:(boundarylinenum[1] - 1)]

  # Extracting lines after the last line of boundarylinenum
  body_after <- body[(boundarylinenum[length(boundarylinenum)] + 1):length(body)]
```

```
    # Combining both parts
    finalbody <- c(body_before, body_after)

    attachments <- separate_attachments(boundarylinenum, body)
    #finalbody <- attachments[length(attachments)]

    #attachments <- attachments[-length(attachments)]
    processed_attachments <- process_attachments(attachments)
  }
  else {
    finalbody <- lines[header_index:length(lines)]
    processed_attachments <- list()
  }

  list(header = finalheader, body = finalbody, attachments = processed_attachments)
}

makelistofemails <- function(files){
  listofemails<-list()
  for (file in files){
    emailinfo <- process_email(file)
    listofemails[[length(listofemails) + 1]] <- list(emailinfo)
  }
  return (listofemails)
}
```

```
files <- list.files(
  path = c(
    "~/Downloads/SpamAssassin/easy_ham",
    "~/Downloads/SpamAssassin/easy_ham_2",
    "~/Downloads/SpamAssassin/hard_ham"
  ),
  full.names = TRUE,
  recursive = TRUE  # Include files from subdirectories
)
hamemails <- lapply(files, makelistofemails)
```

```
spamfiles <- c(
  list.files(path = "~/Downloads/SpamAssassin/spam", full.names = TRUE)[2:700],
  list.files(path = "~/Downloads/SpamAssassin/spam_2", full.names = TRUE)
)
spamemails <- lapply(spamfiles, makelistofemails)
#head(spamemails)
```

```
count_body_lines <- function(emails) {
  numLinesInBody <- vector("numeric", length(emails))
  for (i in seq_along(emails)) {
    numLinesInBody[i] <- length(emails[[i]][[1]][[1]]$body)
  }
  return(numLinesInBody)
}

#length(count_body_lines(emails))
```

```r
find_Re_subject <- function(emails) {
  isRe <- logical(length(emails))
  for (i in seq_along(emails)) {
    subjectline <- emails[[i]][[1]][[1]]$header["Subject"]
    isRe[i] <- grepl("^Re:", subjectline)
  }
  return(isRe)
}
#length(find_Re_subject(hamemails))
```

```r
#replyUnderline logical whether the Reply-To field in the header has an underline and numbers/letters
count_reply_underline <- function(emails) {
  replyUnderline <- logical(length(emails))
  for (i in seq_along(emails)) {
    reply_to <- emails[[i]][[1]][[1]]$header["Reply-To"]
    if (!is.na(reply_to)) {
      replyUnderline[i] <- grepl("\\w+", reply_to) && grepl("_", reply_to)
    } else {
      #NA is there isn't a reply-to section in the header
      replyUnderline[i] <- NA
    }
  }
  return(replyUnderline)
}

#length(count_reply_underline(hamemails))
```

```r
#subjectExclamationCount integer a count of the number of exclamation marks (!) in the subject of the m
library(stringr)

count_subject_exclamation <- function(emails) {
  subjectExclamationCount <- integer(length(emails))
  for (i in seq_along(emails)) {
    subjectline <- emails[[i]][[1]][[1]]$header["Subject"]
    subjectExclamationCount[i] <- sum(str_count(subjectline, fixed("!")))
  }
  return(subjectExclamationCount)
}

#length(count_subject_exclamation(hamemails))
```

```r
#subjectQuestCount integer the number of question marks in the subject
count_subject_questionmark <- function(emails) {
  subjectquestionmarkCount <- integer(length(emails))
  for (i in seq_along(emails)) {
    subjectline <- emails[[i]][[1]][[1]]$header["Subject"]
    subjectquestionmarkCount[i] <- sum(str_count(subjectline, fixed("!")))
  }
  return(subjectquestionmarkCount)
}

#length(count_subject_questionmark(hamemails))
```

```r
#numAttachments integer the number of attachments in the message
numAttachments <- function(emails) {
  attachmentsCount <- integer(length(emails))
  for (i in seq_along(emails)) {
    attachments <- emails[[i]][[1]][[1]][[length(emails[[i]][[1]][[1]])]]
    if (length(attachments) == 0) {
      attachmentsCount[i] <- 0
    } else {
      attachmentsCount[i] <- length(attachments)
    }
  }
  return(attachmentsCount)
}
#length(numAttachments(hamemails))
```

```r
#subjectSpamWords logical whether the subject contains one of the following phrases: viagra,pounds, fre

subjectSpamWords <- function(emails) {
  spam_words <- c("viagra", "pounds", "free", "weight", "guarantee", "millions",
                  "dollars", "credit", "risk", "prescription", "generic", "drug",
                  "money back", "credit card")

  has_spam <- logical(length(emails))

  for (i in seq_along(emails)) {
    subject <- tolower(emails[[i]][[1]][[1]]$header["Subject"])
    has_spam[i] <- any(grepl(paste(spam_words, collapse = "|"), subject))
  }

  return(has_spam)
}

#length(subjectSpamWords(hamemails))
```

```r
# isInReplyTo logical whether the header of the message has an In-Reply-To field

isInReplyTo <- function(emails) {
  has_in_reply_to <- logical(length(emails))

  for (i in seq_along(emails)) {
    header <- emails[[i]][[1]][[1]]$header
    has_in_reply_to[i] <- !is.null(header["In-Reply-To"])
  }

  return(has_in_reply_to)
}

#length(isInReplyTo(hamemails))
#check if the spam has any false, ham doesn't have any false
```

```r
#messageIdHasNoHostname logical whether the message identifier (id) that uniquely identifies the messag
messageIdHasNoHostname <- function(emails) {
  no_hostname <- logical(length(emails))
```

```
  for (i in seq_along(emails)) {
    message_id <- emails[[i]][[1]][[1]]$header["Message-ID"]
    no_hostname[i] <- is.na(message_id)
  }

  return(no_hostname)
}

#length(messageIdHasNoHostname(hamemails))
#any(messageIdHasNoHostname(hamemails))
```

```
fromNumericEnd <- function(emails) {
  numeric_end <- logical(length(emails))

  for (i in seq_along(emails)) {
    from_field <- emails[[i]][[1]][[1]]$header[["From"]]
    if (!is.na(from_field)) {
      # Extract user login from the From field
      user_login <- gsub(".*[<\"](.+)[>\"].*", "\\1", from_field)
      # Check if the user login ends with a number
      numeric_end[i] <- grepl("\\d$", user_login)
    } else {
      numeric_end[i] <- NA
    }
  }

  return(numeric_end)
}

#length(fromNumericEnd(hamemails))
#any(!fromNumericEnd(hamemails))
```

```
#isYelling logical whether the Subject of the mail is in capital letters
isYelling <- function(emails) {
  yelling <- logical(length(emails))
  for (i in seq_along(emails)) {
    subject <- emails[[i]][[1]][[1]]$header["Subject"]
    yelling[i] <- toupper(subject) == subject
  }
  return(yelling)
}
#length(isYelling(hamemails))
```

```
averageWordLength <- function(emails) {
  word_lengths <- numeric(length(emails))
  for (i in seq_along(emails)) {
    body_text <- emails[[i]][[1]][[1]]$body
    words <- unlist(strsplit(body_text, "\\W+"))
    if (is.na(mean(nchar(words)))){
      word_lengths[i] <- 0
    }
    else{
```

```
      word_lengths[i] <- mean(nchar(words))
    }
  }
  return(word_lengths)
}
#length(averageWordLength(hamemails))
```

```
percentSubjectBlanks <- function(emails) {
  percent_blanks <- numeric(length(emails))
  for (i in seq_along(emails)) {
    subject <- emails[[i]][[1]][[1]]$header["Subject"]
    subject_length <- nchar(subject)
    blanks <- sum(substr(subject, 1, 1) == " ")
    percent_blanks[i] <- (blanks / subject_length) * 100
  }
  return(percent_blanks)
}
#any(percentSubjectBlanks(emails) > 0, na.rm=TRUE)
#length(percentSubjectBlanks(emails))
```

```
hourSent <- function(emails) {
  hour_sent <- integer(length(emails))
  for (i in seq_along(emails)) {
    date_string <- emails[[i]][[1]][[1]]$header["Date"]
    # Extract the hour part using regular expressions
    hour_match <- regmatches(date_string, regexpr("\\d{2}:", date_string))
    if (length(hour_match) > 0) {
      # Remove ":" and convert to integer
      hour_sent[i] <- as.integer(sub(":", "", hour_match))
    } else {
      # If hour is not found, set to NA
      hour_sent[i] <- NA
    }
  }
  return(hour_sent)
}
#hamemails[920]
#any(is.na(hourSent(hamemails)))
which(is.na(hourSent(hamemails)))
```

```
##  [1]  972 1762 1763 1896 2938 3009 3150 3349 3489 3617 3822 3844 3886 4115 4329
## [16] 4367 4517 4770 4796 4821 4828 4829 4846
```

```
#hamemails[[419]][[1]][[1]]$header
#fix the way I write the headers in, shouldn't be any NAs
```

```
subjectPunctuationCheck <- function(emails) {
  has_punctuation <- logical(length(emails))
  for (i in seq_along(emails)) {
    subject <- tolower(emails[[i]][[1]][[1]]$header["Subject"])
    # Check if the subject contains punctuation or digits surrounded by characters
    has_punctuation[i] <- grepl("\\b[a-zA-Z]+[a-zA-Z][[:punct:]\\d][a-zA-Z][a-zA-Z]+\\b", subject)
```

```
  }
  return(has_punctuation)
}
#any(is.na(subjectPunctuationCheck(emails)))
```

```
numDollarSigns <- function(emails) {
  dollarSignCount <- integer(length(emails))
  for (i in seq_along(emails)) {
    body_text <- emails[[i]][[1]][[1]]$body
    dollarSignCount[i] <- sum(gregexpr("\\$", body_text)[[1]] > 0)
  }
  return(dollarSignCount)
}
#any(numDollarSigns(emails)>0)
#check if spam has more than 0 if not something is wrong
```

```
numRecipients <- function(emails) {
  num_recipients <- integer(length(emails))
  for (i in seq_along(emails)) {
    to_field <- emails[[i]][[1]][[1]]$header["To"]
    cc_field <- if ("Cc" %in% names(emails[[i]][[1]][[1]]$header)) {
      emails[[i]][[1]][[1]]$header["Cc"]
    } else {
      NA
    }

    # Check if To and Cc fields exist
    if (is.na(to_field) && is.na(cc_field)) {
      num_recipients[i] <- 0
    } else {
      num_to <- ifelse(is.na(to_field), 0, length(unlist(strsplit(to_field, ","))))
      num_cc <- ifelse(is.na(cc_field), 0, length(unlist(strsplit(cc_field, ","))))
      num_recipients[i] <- num_to + num_cc
    }
  }
  return(num_recipients)
}
#length(numRecipients(emails))
```

```
priority <- function(emails) {
  has_high_priority <- logical(length(emails))

  for (i in seq_along(emails)) {
    header <- emails[[i]][[1]][[1]]$header

    if (!is.null(header["X-Priority"]) || !is.null(header["X-Msmail-Priority"])) {
      if (!is.null(header["X-Priority"])) {
        has_high_priority[i] <- grepl("high", header["X-Priority"], ignore.case = TRUE)
      } else {
        has_high_priority[i] <- grepl("high", header["X-Msmail-Priority"], ignore.case = TRUE)
      }
    } else {
      has_high_priority[i] <- FALSE
```

```
    }
  }

  return(has_high_priority)
}
#any(!priority(emails))
#length(priority(emails))
```

```
percentCapitals <- function(emails) {
  percent_capitals <- numeric(length(emails))

  for (i in seq_along(emails)) {
    body_text <- paste(emails[[i]][[1]][[1]]$body, collapse = " ")  # Collapse lines into a single stri

    if (anyNA(body_text)) {
      percent_capitals[i] <- 0
      next
    }

    total_chars <- nchar(gsub("[^A-Za-z]", "", body_text))  # Count only letters
    uppercase_chars <- nchar(gsub("[^A-Z]", "", body_text))  # Count uppercase letters

    if (total_chars > 0) {
      percent_capitals[i] <- (uppercase_chars / total_chars) * 100
    } else {
      percent_capitals[i] <- 0
    }
  }

  return(percent_capitals)
}

length(percentCapitals(hamemails))
```

```
## [1] 4864
```

```
containsImages <- function(emails) {
  contains_images <- logical(length(emails))

  for (i in seq_along(emails)) {
    body_text <- paste(emails[[i]][[1]][[1]]$body, collapse = " ")

    # Check if body_text contains NA, if so, set contains_images to FALSE and move to the next email
    if (anyNA(body_text)) {
      contains_images[i] <- FALSE
      next
    }

    # Check if body_text contains the <img> tag
    contains_images[i] <- grepl("\\<img\\>", body_text, ignore.case = TRUE)
  }
```

```
  return(contains_images)
}
any(containsImages(hamemails))
```

```
## [1] TRUE
```

```
# Assuming you already have emails loaded and the functions defined

# Create an empty dataframe
hamdf <- data.frame(
  isSpam = rep(FALSE, length(hamemails)),
  body_lengths = count_body_lines(hamemails),
  isRe = find_Re_subject(hamemails),
  bodyLineCount = count_body_lines(hamemails),
  replyUnderline = count_reply_underline(hamemails),
  subjectExclamationCount = count_subject_exclamation(hamemails),
  subjectQuestionmarkCount = count_subject_questionmark(hamemails),
  numAttachments = numAttachments(hamemails),
  priority = priority(hamemails),
  subjectSpamWords = subjectSpamWords(hamemails),
  isInReplyTo = isInReplyTo(hamemails),
  messageIdHasNoHostname = messageIdHasNoHostname(hamemails),
  fromNumericEnd = fromNumericEnd(hamemails),
  isYelling = isYelling(hamemails),
  averageWordLength = averageWordLength(hamemails),
  percentSubjectBlanks = percentSubjectBlanks(hamemails),
  hourSent = hourSent(hamemails),
  subjectPunctuationCheck = subjectPunctuationCheck(hamemails),
  numDollarSigns = numDollarSigns(hamemails),
  numRecipients = numRecipients(hamemails),
  percentCapitals = percentCapitals(hamemails),
  containsImages = containsImages(hamemails)
)

head(hamdf)
```

```
##   isSpam body_lengths isRe bodyLineCount replyUnderline subjectExclamationCount
## 1  FALSE          184 TRUE           184             NA                       0
## 2  FALSE            9 TRUE             9          FALSE                       0
## 3  FALSE            9 TRUE             9          FALSE                       0
## 4  FALSE            9 TRUE             9          FALSE                       0
## 5  FALSE            9 TRUE             9             NA                       0
## 6  FALSE            9 TRUE             9          FALSE                       0
##   subjectQuestionmarkCount numAttachments priority subjectSpamWords isInReplyTo
## 1                        0              0    FALSE           FALSE        TRUE
## 2                        0              2    FALSE           FALSE        TRUE
## 3                        0              2    FALSE           FALSE        TRUE
## 4                        0              2    FALSE           FALSE        TRUE
## 5                        0              2    FALSE           FALSE        TRUE
## 6                        0              2    FALSE           FALSE        TRUE
##   messageIdHasNoHostname fromNumericEnd isYelling averageWordLength
## 1                   TRUE          FALSE     FALSE          4.265585
## 2                   TRUE          FALSE     FALSE          8.000000
```

```
## 3                   TRUE          FALSE     FALSE          8.000000
## 4                   TRUE          FALSE     FALSE          8.000000
## 5                   TRUE          FALSE     FALSE          8.000000
## 6                   TRUE          FALSE     FALSE          8.000000
##   percentSubjectBlanks hourSent subjectPunctuationCheck numDollarSigns
## 1                    0       19                    TRUE              0
## 2                    0       10                    TRUE              0
## 3                    0       10                    TRUE              0
## 4                    0       10                    TRUE              0
## 5                    0       11                    TRUE              0
## 6                    0       10                    TRUE              0
##   numRecipients percentCapitals containsImages
## 1             2        3.712153          FALSE
## 2             2        2.247191          FALSE
## 3             3        2.247191          FALSE
## 4             2        2.247191          FALSE
## 5             3        2.247191          FALSE
## 6             2        2.247191          FALSE
```

```
spamdf <- data.frame(
  isSpam = rep(TRUE, length(spamemails)),
  body_lengths = count_body_lines(spamemails),
  isRe = find_Re_subject(spamemails),
  bodyLineCount = count_body_lines(spamemails),
  replyUnderline = count_reply_underline(spamemails),
  subjectExclamationCount = count_subject_exclamation(spamemails),
  subjectQuestionmarkCount = count_subject_questionmark(spamemails),
  numAttachments = numAttachments(spamemails),
  priority = priority(spamemails),
  subjectSpamWords = subjectSpamWords(spamemails),
  isInReplyTo = isInReplyTo(spamemails),
  messageIdHasNoHostname = messageIdHasNoHostname(spamemails),
  fromNumericEnd = fromNumericEnd(spamemails),
  isYelling = isYelling(spamemails),
  averageWordLength = averageWordLength(spamemails),
  percentSubjectBlanks = percentSubjectBlanks(spamemails),
  hourSent = hourSent(spamemails),
  subjectPunctuationCheck = subjectPunctuationCheck(spamemails),
  numDollarSigns = numDollarSigns(spamemails),
  numRecipients = numRecipients(spamemails),
  percentCapitals = percentCapitals(spamemails),
  containsImages = containsImages(spamemails)
)
head(spamdf)
```

```
##   isSpam body_lengths  isRe bodyLineCount replyUnderline
## 1   TRUE          103 FALSE           103             NA
## 2   TRUE           42 FALSE            42             NA
## 3   TRUE           31 FALSE            31             NA
## 4   TRUE           48 FALSE            48             NA
## 5   TRUE          606 FALSE           606             NA
## 6   TRUE           41 FALSE            41             NA
##   subjectExclamationCount subjectQuestionmarkCount numAttachments priority
## 1                       0                       0              0    FALSE
```

```
## 2                       0                         0              0    FALSE
## 3                       0                         0              0    FALSE
## 4                       0                         0              0    FALSE
## 5                       0                         0              0    FALSE
## 6                       1                         1              0    FALSE
##   subjectSpamWords isInReplyTo messageIdHasNoHostname fromNumericEnd isYelling
## 1            FALSE        TRUE                  FALSE          FALSE     FALSE
## 2             TRUE        TRUE                   TRUE          FALSE     FALSE
## 3             TRUE        TRUE                   TRUE          FALSE     FALSE
## 4            FALSE        TRUE                  FALSE          FALSE     FALSE
## 5            FALSE        TRUE                   TRUE          FALSE     FALSE
## 6            FALSE        TRUE                   TRUE          FALSE      TRUE
##   averageWordLength percentSubjectBlanks hourSent subjectPunctuationCheck
## 1          3.616618                    0       20                   FALSE
## 2          3.934211                    0        6                   FALSE
## 3          4.297521                    0        9                    TRUE
## 4          4.231111                    0       10                   FALSE
## 5          3.700691                    0       10                   FALSE
## 6          6.964844                    0       13                   FALSE
##   numDollarSigns numRecipients percentCapitals containsImages
## 1              0             1       37.970254          FALSE
## 2              0             1        8.043876          FALSE
## 3              0             1        6.584362          FALSE
## 4              0             1       24.702194          FALSE
## 5              0             1        5.912294           TRUE
## 6              0            11       18.407311          FALSE
```

```
emails_df <- rbind(spamdf, hamdf)
head(emails_df)
```

```
##   isSpam body_lengths  isRe bodyLineCount replyUnderline
## 1   TRUE          103 FALSE           103             NA
## 2   TRUE           42 FALSE            42             NA
## 3   TRUE           31 FALSE            31             NA
## 4   TRUE           48 FALSE            48             NA
## 5   TRUE          606 FALSE           606             NA
## 6   TRUE           41 FALSE            41             NA
##   subjectExclamationCount subjectQuestionmarkCount numAttachments priority
## 1                       0                         0              0    FALSE
## 2                       0                         0              0    FALSE
## 3                       0                         0              0    FALSE
## 4                       0                         0              0    FALSE
## 5                       0                         0              0    FALSE
## 6                       1                         1              0    FALSE
##   subjectSpamWords isInReplyTo messageIdHasNoHostname fromNumericEnd isYelling
## 1            FALSE        TRUE                  FALSE          FALSE     FALSE
## 2             TRUE        TRUE                   TRUE          FALSE     FALSE
## 3             TRUE        TRUE                   TRUE          FALSE     FALSE
## 4            FALSE        TRUE                  FALSE          FALSE     FALSE
## 5            FALSE        TRUE                   TRUE          FALSE     FALSE
## 6            FALSE        TRUE                   TRUE          FALSE      TRUE
##   averageWordLength percentSubjectBlanks hourSent subjectPunctuationCheck
## 1          3.616618                    0       20                   FALSE
## 2          3.934211                    0        6                   FALSE
```

```
## 3          4.297521                   0        9                    TRUE
## 4          4.231111                   0       10                   FALSE
## 5          3.700691                   0       10                   FALSE
## 6          6.964844                   0       13                   FALSE
##   numDollarSigns numRecipients percentCapitals containsImages
## 1              0             1       37.970254          FALSE
## 2              0             1        8.043876          FALSE
## 3              0             1        6.584362          FALSE
## 4              0             1       24.702194          FALSE
## 5              0             1        5.912294           TRUE
## 6              0            11       18.407311          FALSE
```

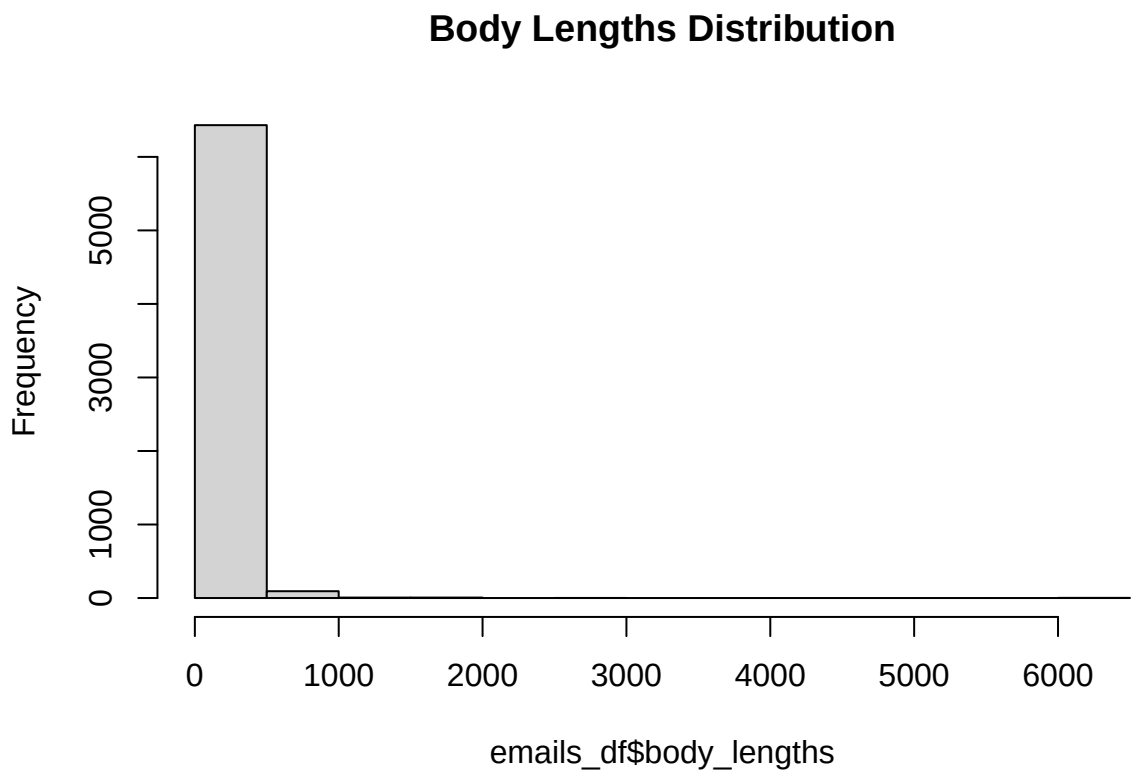
I wanted to visualize some of the variables in the dataframe to check if it looked accurate and do some more data exploration. In the future, I would do some data cleaning realted to NA values before utilizing this information in any predictive models or other data exploration strategies.

```
summary(emails_df)
```

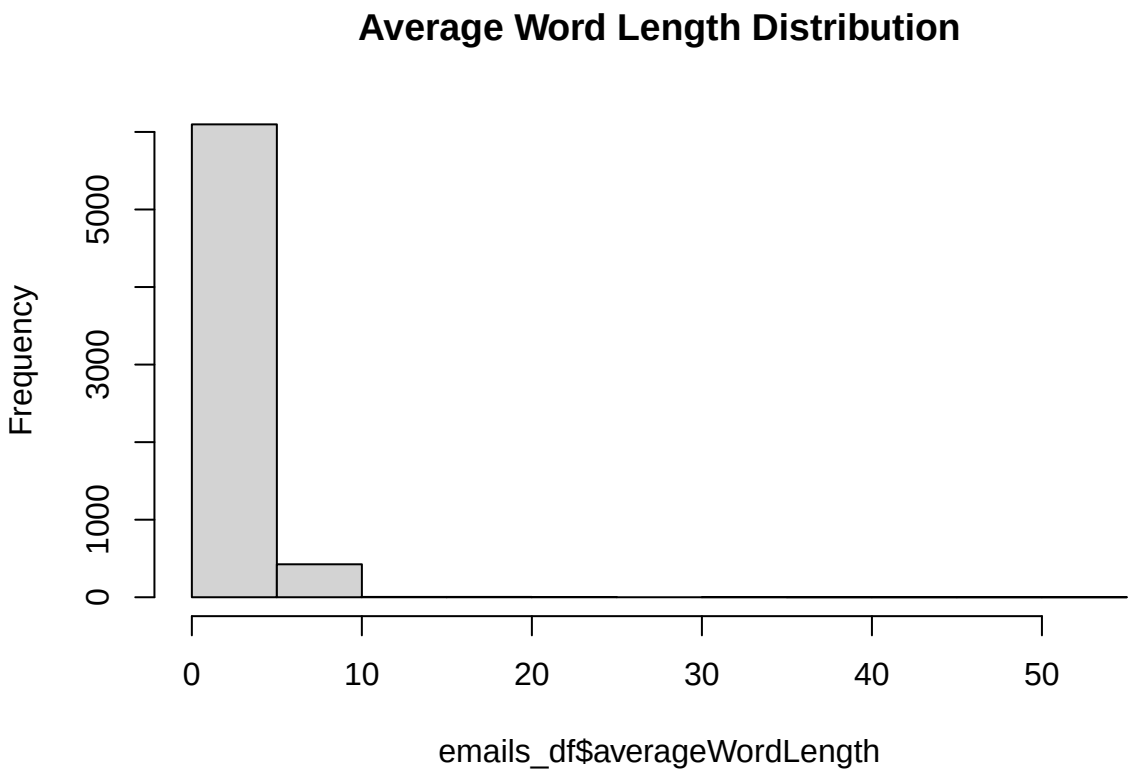
```
##    isSpam         body_lengths        isRe         bodyLineCount
##  Mode :logical   Min.   :   2.00   Mode :logical   Min.   :   2.00
##  FALSE:4864      1st Qu.:  16.00   FALSE:4462      1st Qu.:  16.00
##  TRUE :1676      Median :  31.00   TRUE :2078      Median :  31.00
##                  Mean   :  64.62                   Mean   :  64.62
##                  3rd Qu.:  55.00                   3rd Qu.:  55.00
##                  Max.   :6321.00                   Max.   :6321.00
##
##  replyUnderline  subjectExclamationCount subjectQuestionmarkCount
##  Mode :logical   Min.   :0.0000          Min.   :0.0000
##  FALSE:2027      1st Qu.:0.0000          1st Qu.:0.0000
##  TRUE :96        Median :0.0000          Median :0.0000
##  NA's :4417      Mean   :0.1146          Mean   :0.1146
##                  3rd Qu.:0.0000          3rd Qu.:0.0000
##                  Max.   :8.0000          Max.   :8.0000
##                  NA's   :4               NA's   :4
##  numAttachments    priority       subjectSpamWords isInReplyTo
##  Min.   :0.0000   Mode :logical   Mode :logical    Mode:logical
##  1st Qu.:0.0000   FALSE:6533      FALSE:6263       TRUE:6540
##  Median :0.0000   TRUE :7         TRUE :277
##  Mean   :0.1437
##  3rd Qu.:0.0000
##  Max.   :6.0000
##
##  messageIdHasNoHostname fromNumericEnd  isYelling       averageWordLength
##  Mode :logical          Mode :logical   Mode :logical   Min.   : 0.000
##  FALSE:585              FALSE:6510      FALSE:6386      1st Qu.: 3.889
##  TRUE :5955             TRUE :30        TRUE :150       Median : 4.182
##                                         NA's :4         Mean   : 4.216
##                                                         3rd Qu.: 4.506
##                                                         Max.   :50.570
##
##  percentSubjectBlanks    hourSent     subjectPunctuationCheck
##  Min.   :  0.0000     Min.   : 0.00   Mode :logical
##  1st Qu.:  0.0000     1st Qu.: 8.00   FALSE:4164
```

```
##  Median :  0.0000     Median :12.00   TRUE :2376
##  Mean   :  0.1713     Mean   :12.32
##  3rd Qu.:  0.0000     3rd Qu.:17.00
##  Max.   :100.0000     Max.   :95.00
##  NA's   :4            NA's   :155
##  numDollarSigns     numRecipients    percentCapitals   containsImages
##  Min.   :0.000000   Min.   : 0.000   Min.   :  0.000   Mode :logical
##  1st Qu.:0.000000   1st Qu.: 1.000   1st Qu.:  4.585   FALSE:6047
##  Median :0.000000   Median : 1.000   Median :  6.444   TRUE :493
##  Mean   :0.002294   Mean   : 1.839   Mean   :  9.380
##  3rd Qu.:0.000000   3rd Qu.: 1.000   3rd Qu.:  9.769
##  Max.   :2.000000   Max.   :74.000   Max.   :100.000
##
```

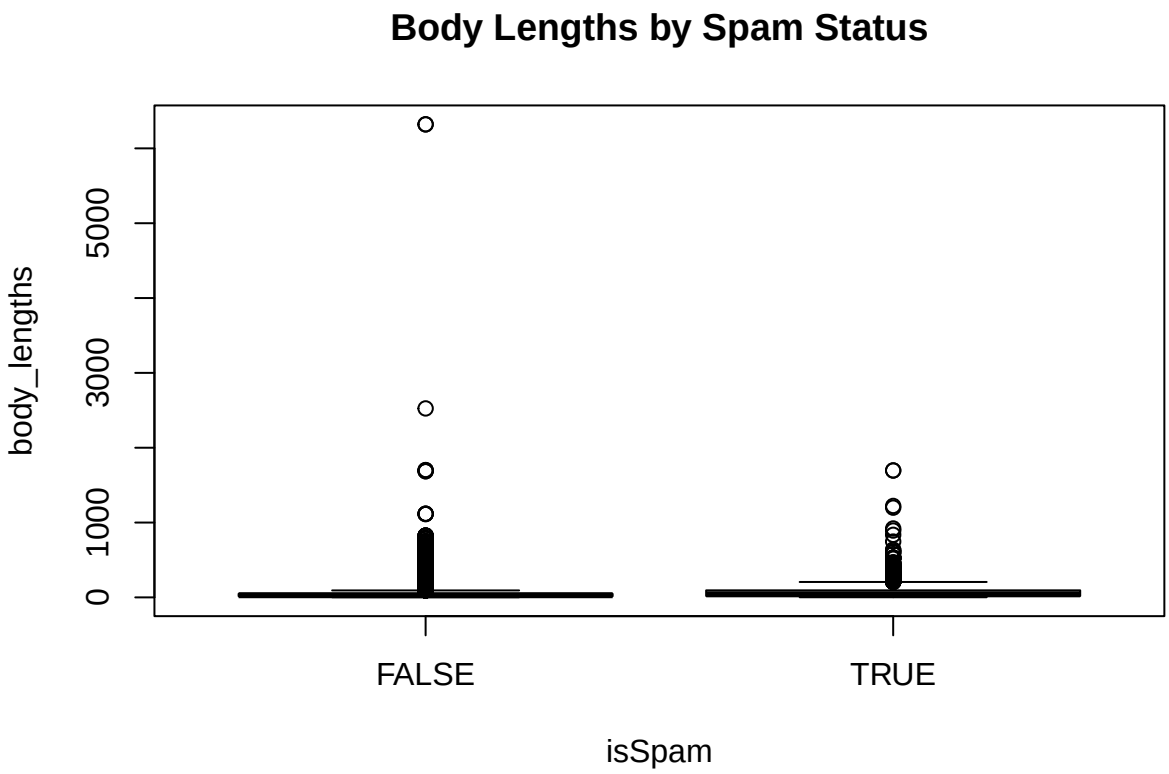
```
hist(emails_df$body_lengths, main = "Body Lengths Distribution")
```



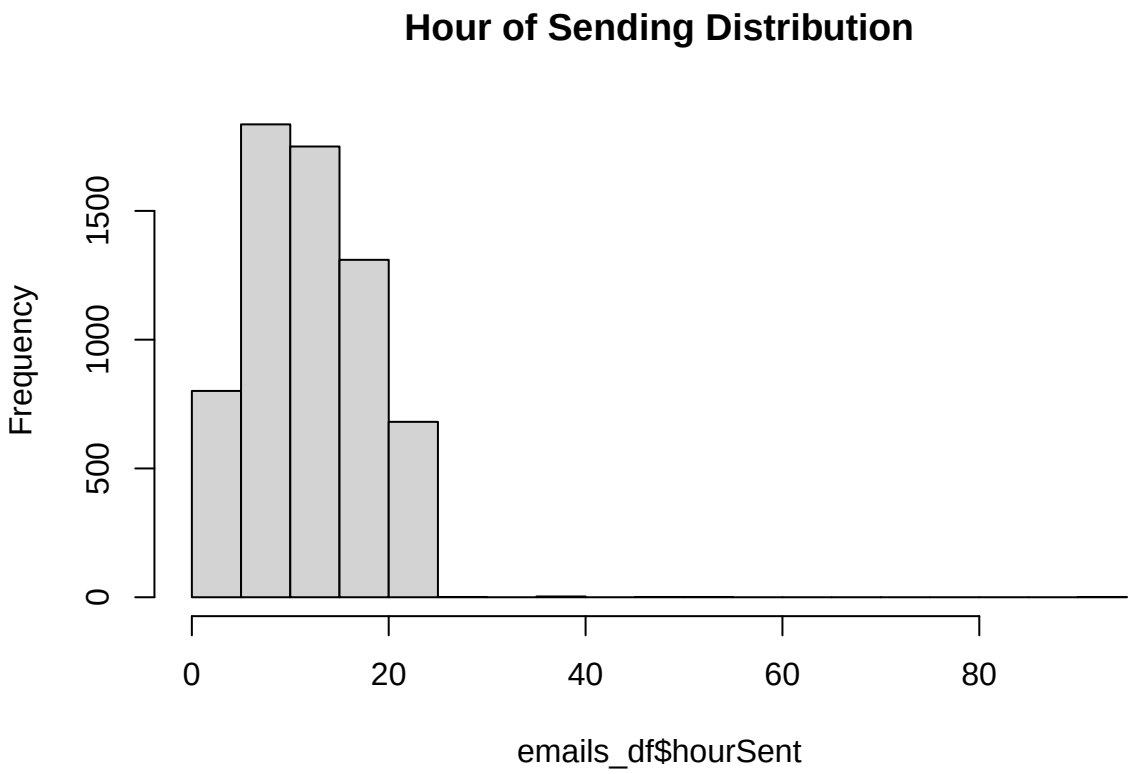

```
hist(emails_df$averageWordLength, main = "Average Word Length Distribution")
```




```r
boxplot(body_lengths ~ isSpam, data = emails_df, main = "Body Lengths by Spam Status")
```




```r
hist(emails_df$hourSent, main = "Hour of Sending Distribution")
```


Hour of Sending Distribution
Frequency
0
500
1000
1500
0
20
40
60
80
emails_df$hourSent