Classification of Spam and Ham Emails

Zeng Fung Liew

10/24/2020

I completed this assignment myself without consulting books or tutorials that specifically addressed this data set.

Create a data frame of "derived" variables that give various measures of email messages.

To make the process simple and easy to debug, I created the following functions:

- `read_files()`: takes in a directory as an input, and read all the emails into a "derived" data frame with the variables stated in Page 3.
- `derived_df()`: belongs in `read_files()` and is used to create a data frame with the variable names as columns.
- `get_row_data()`: belongs in `read_lines()`, for each email in the directory, we want to obtain the necessary data and record them in the data frame.
- `get_attachment_lines()`: takes in the header and body of the email, and extract the attachment "lines" from the body.
- a function for each variable in Page 3.

The functions are listed below (Explanations are being commented on top of each function):

```
# create an empty data frame with allocated variables
derived_df = function() {
  df = data.frame(isSpam = logical(),
                  isRe = logical(),
                  numLinesInBody = integer(),
                  bodyCharacterCount = integer(),
                  replyUnderline = logical(),
                  subjectExclamationCount = integer(),
                  subjectQuestCount = integer(),
                  numAttachments = integer(),
                  priority = logical(),
                  numRecipients = integer(),
                  percentCapitals = numeric(),
                  isInReplyTo = logical(),
                  sortedRecipients = logical(),
                  subjectPunctuationCheck = logical(),
                  hourSent = integer(),
                  multipartText = logical(),
```

```
                  containsImages = logical(),
                  isPGPsigned = logical(),
                  percentHTMLTags = numeric(),
                  subjectSpamWords = logical(),
                  percentSubjectBlanks = numeric(),
                  messageIdHasNoHostname = logical(),
                  fromNumericEnd = logical(),
                  isYelling = logical(),
                  percentForwards = numeric(),
                  isOriginalMessage = logical(),
                  isDear = logical(),
                  isWrote = logical(),
                  averageWordLength = numeric(),
                  numDollarSigns = integer())

  return (df)
}


# get data from file
# we first want to separate the email content into header and body+attachments
# to extract the data of each field, we call the respective functions
get_row_data = function(file){
  content = readLines(file, warn = FALSE)
  header = content[c(1:grep("^$", content)[1])]
  body = content[-c(1:grep("^$", content)[1])]
  return (list(is_spam(file),
               is_re(header),
               num_lines_in_body(header, body),
               body_character_count(header, body),
               reply_underline(header),
               subject_exclamation_count(header),
               subject_quest_count(header),
               num_attachments(header, body),
               is_priority(header),
               num_recipients(header),
               percent_capitals(header, body),
               is_in_reply_to(header),
               sorted_recipients(header),
               subject_punctuation_check(header),
               hour_sent(header),
               multipart_text(header),
               contains_images(header, body),
               is_PGP_signed(content),
               percent_HTML_tags(header, body),
               subject_spam_words(header),
               percent_subject_blanks(header),
               message_id_has_no_hostname(header),
               from_numeric_end(header),
               is_yelling(header),
               percent_forwards(header, body),
               is_original_message(header, body),
               is_dear(body),
```

```
              is_wrote(body),
              average_word_length(header, body),
              num_dollar_signs(body))
  )
}

# get indexes of body that is for attachments
# function runs through the following possible cases:
# - if header contains the field "MIME version", then it contains attachments,
#     and we will obtain the attachments via the boundary in the header
# - if the body contains some field "boundary", we get the boundary to obtain
#     find the attachements
# - we find the lines "+_NextPart_xxxx" from the body as that indicates that there
#     is an attachment underneath
# - otherwise, we look for the lines "= Multipart Boundary 0000" as that also
#     indicates that there is an attachment underneath
# after obtaining the boundaries, we find the lines of the body that contain the
#   boundaries
# then, if between every two boundary there contains the keyword "attachment", then
#   that chunk is an attachment, and we return those lines
get_attachment_lines = function(header, body){
  if (max(grepl("^MIME-Version=", header, ignore.case = TRUE)) == 1){
    b_line = header[grepl("boundary=.*", header)]
    b_stmt = regmatches(b_line, gregexpr("boundary=.*", b_line))[[1]]
    b_stmt = substring(b_stmt, 10)
    boundary = gsub("\"", "", b_stmt)
  } else if (max(grepl("boundary=", body, ignore.case = TRUE)) == 1){
    b_line = body[grepl("boundary=.*", body)]
    b_stmt = regmatches(b_line, gregexpr("boundary=.*\\b", b_line))[[1]]
    b_stmt = substring(b_stmt, 10)
    boundary = gsub("\"", "", b_stmt)
    boundary = gsub(";.*", "", boundary)
  } else if (max(grepl("=_NextPart_[0-9a-zA-Z]+", body, ignore.case = TRUE)) == 1){
    b_line = body[grepl("=_NextPart_[0-9a-zA-Z]+", body)]
    boundary = regmatches(b_line, gregexpr("=_NextPart_[0-9a-zA-Z]+", b_line))[[1]]
  } else{
    boundary = "= Multipart Boundary 0802010258"
  }

  bb = c(agrep(boundary, body, fixed = TRUE), length(body))
  ba = grep("attachment", body, ignore.case = TRUE)
  attachment_lines = integer(0)
  for (val in ba){
    k = min(which(val < bb)) # min index of bb for which val < bb[k]
    attachment_lines = c(attachment_lines, max(1,bb[k-1]):(bb[k]-1))
  }
  return (attachment_lines)
}

# returns TRUE if mail is spam and FALSE if mail is ham
is_spam = function(filename) {
  return (grepl("^[.]/.*spam.*/", filename))
}
```

```
# returns TRUE if Re: appears as the first word in subject and FALSE otherwise
is_re = function(header) {
  # check if subject field exists in header
  if (sum(grepl("^Subject:", header)) == 0) {
    return (NA)
  }

  subject = header[grep("^Subject:", header)]
  return (grepl("^Subject:.*[Rr]e:.+", subject))
}

# return number of lines in body
num_lines_in_body = function(header, body) {
  #remove attachment sections
  if (num_attachments(header, body) > 0) {
    al = get_attachment_lines(header, body)
    body = body[-al]
  }

  return (length(body))
}

# count the number of characters in body of email messages, exclude html tags
body_character_count = function(header, body) {
  # remove attachment sections
  if (num_attachments(header, body) > 0) {
    al = get_attachment_lines(header, body)
    bod = body[-al]
  }

  html_tags = gregexpr("<.*>", body)
  data = regmatches(body, html_tags, invert = TRUE)
  return (sum(nchar(data)))
}

# whether the Reply-To field in the header has an underline
# and numbers/letters
reply_underline = function(header) {
  if (sum(grepl("^Reply-To", header)) == 0) {
    return (NA)
  }

  reply_to = header[grep("^Reply-To", header)]
  return (grepl("([0-9a-zA-Z]*_[0-9a-zA-Z]+|[0-9a-zA-Z]+_[0-9a-zA-Z]*)",
                reply_to))
}

# count number of exclamation marks (!) in subject
subject_exclamation_count = function(header) {
  # check if subject field exists in header
  if (sum(grepl("^Subject:", header)) == 0) {
    return (NA)
  }
```

```
  subject = header[grep("^Subject:", header)]
  subject_vec = strsplit(subject, "")[[1]]
  return (sum(subject_vec == "!"))
}

# count number of question# function runs through the following possible cases:
# - if header contains the field "MIME version", then it contains attachments,
#     and we will obtain the attachments via the boundary in the header
# - if the body contains some field "boundary", we get the boundary to obtain
#     find the attachements
# - we find the lines "+_NextPart_xxxxx" from the body as that indicates that there
#     is an attachment underneath
# - otherwise, we look for the lines "= Multipart Boundary 0000" as that also
#     indicates that there is an attachment underneath
# marks (?) in subject
subject_quest_count = function(header) {
  # check if subject field exists in header
  if (sum(grepl("^Subject:", header)) == 0) {
    return (NA)
  }

  subject = header[grep("^Subject:", header)]
  subject_vec = strsplit(subject, "")[[1]]
  return (sum(subject_vec == "?"))
}

# returns number of attachments in body
# function runs through the following possible cases:
# - if header contains the field "MIME version", then it contains attachments,
#     and we will obtain the attachments via the boundary in the header
# - if the body contains some field "boundary", we get the boundary to obtain
#     find the attachements
# - we find the lines "+_NextPart_xxxxx" from the body as that indicates that there
#     is an attachment underneath
# - otherwise, we look for the lines "= Multipart Boundary 0000" as that also
#     indicates that there is an attachment underneath
# our output will be the number of  matches of these boundaries, since each boundary
#   is above an attachment
# if email has no attachments, our variable bb will be an empty vector, hence our
#   output will be 0.
num_attachments = function(header, body) {
  if (max(grepl("^MIME-Version=", header, ignore.case = TRUE)) == 1){
    b_line = header[grepl("boundary=.*", header)]
    b_stmt = regmatches(b_line, gregexpr("boundary=.*", b_line))[[1]]
    b_stmt = substring(b_stmt, 10)
    boundary = gsub("\"", "", b_stmt)
  } else if (max(grepl("boundary=", body, ignore.case = TRUE)) == 1){
    b_line = body[grepl("boundary=.*", body)]
    b_stmt = regmatches(b_line, gregexpr("boundary=.*\\b", b_line))[[1]]
    b_stmt = substring(b_stmt, 10)
    boundary = gsub("\"", "", b_stmt)
    boundary = gsub(";.*", "", boundary)
  } else if (max(grepl("=_NextPart_[0-9a-zA-Z]+", body, ignore.case = TRUE)) == 1){
```

```
    b_line = body[grepl("=_NextPart_[0-9a-zA-Z]+", body)]
    boundary = regmatches(b_line, gregexpr("=_NextPart_[0-9a-zA-Z]+", b_line))[[1]]
  } else{
    boundary = "= Multipart Boundary 0802010258"
  }

  bb = c(agrep(boundary, body, fixed = TRUE))
  return (length(bb))
}

# check if email is of high/highest priority
is_priority = function(header) {
  if (max(grepl("^X-Priority:", header)) == 1) {
    xp = tolower(header[grep("^X-Priority:", header)])
    if (grepl("(high|1|2)", xp) == TRUE) {
      return (TRUE)   #based on spam_2 dir, 1-highest, 2-high
    }
  }
  if (max(grepl("^X-Smell-Priority:", header)) == 1) {
    xsp = tolower(header[grep("^X-Smell-Priority:", header)])
    if (grepl("high", xsp) == TRUE) {
      return (TRUE)
    }
  }
  return (FALSE)
}

# check number of recipients from email via To, Cc, Bcc
num_recipients = function(header) {
  # return NA if To: field does not exist
  if (max(grepl("^(To|Cc):", header, ignore.case = TRUE)) == 0) {
    return (0)
  }

  # regex for email: local@dom
  local = "[A-Za-z0-9_-]+(\\.[A-Za-z0-9_-]+)*"
  dom = "([a-zA-Z0-9]([a-zA-Z0-9-]*[a-zA-Z0-9])*)(\\.[a-zA-Z0-9]([a-zA-Z0-9-]*[a-zA-Z0-9])*)+"
  pattern = paste0(local, "@", dom)

  to_rcp = character(0)  # declare to_rcp for sake of combining recipient list
  if (max(grepl("^To:", header, ignore.case = TRUE)) == 1) {
    to = header[grep("^To:", header)]
    to_getpattern = gregexpr(pattern, to)
    to_rcp = regmatches(to, to_getpattern)[[1]]
  }

  cc_rcp = character(0)  # declare cc_rcp for sake of combining recipient list
  if (max(grepl("^Cc:", header)) == 1) {
    cc = header[grep("^Cc:", header)]
    cc_getpattern = gregexpr(pattern, cc)
    cc_rcp = regmatches(cc, cc_getpattern)[[1]]
  }
```

```
  bcc_rcp = character(0)  # declare bcc_rcp for sake of combining recipient list
  if (max(grepl("^Bcc:", header)) == 1) {
    bcc = header[grep("^Bcc:", header)]
    bcc_getpattern = gregexpr(pattern, bcc)
    bcc_rcp = regmatches(bcc, bcc_getpattern)[[1]]
  }

  rcp_list = c(to_rcp, cc_rcp, bcc_rcp)
  return (length(unique(rcp_list)))
}

# returns percentage of the body that are capitals
percent_capitals = function(header, body) {
  # remove attachment sections
  if (num_attachments(header, body) > 0) {
    al = get_attachment_lines(header, body)
    bod = body[-al]
  }

  rm_tags = gsub("<.*>", "", body)
  letters = regmatches(rm_tags, gregexpr("[[:alpha:]]", rm_tags))
  caps = regmatches(rm_tags, gregexpr("[A-Z]", rm_tags))
  return (sum(lengths(caps)) / sum(lengths(letters)))
}

# whether the header of the message has an In-Reply-To field
is_in_reply_to = function(header) {
  return (max(grepl("^In-Reply-To:", header)) == 1)
}

# check if email addresses are sorted
sorted_recipients = function(header) {
  # return NA if To: field does not exist
  if (max(grepl("^(To|Cc):", header, ignore.case = TRUE)) == 0) {
    return (0)
  }

  to_rcp = character(0)  # declare to_rcp for sake of combining recipient list
  if (max(grepl("^To:", header, ignore.case = TRUE)) == 1) {
    to = header[grep("^To:", header)]
    to_getpattern = gregexpr("[^[:space:]]+@[a-zA-Z]+(\\.[a-zA-Z]+)+", to)
    to_rcp = regmatches(to, to_getpattern)[[1]]
  }

  cc_rcp = character(0)  # declare cc_rcp for sake of combining recipient list
  if (max(grepl("^Cc:", header)) == 1) {
    cc = header[grep("^Cc:", header)]
    cc_getpattern = gregexpr("[^[:space:]]+@[a-zA-Z]+(\\.[a-zA-Z]+)+", cc)
    cc_rcp = regmatches(cc, cc_getpattern)[[1]]
  }

  if ((length(to_rcp) <= 2) & (length(cc_rcp) <= 2)){
    return (NA)
```

```
  } else{
    return((!is.unsorted(to_rcp) | !is.unsorted(rev(to_rcp))) &
             (!is.unsorted(cc_rcp) | !is.unsorted(rev(cc_rcp))))
  }
}

# whether the subject has punctuation or digits surrounded by characters
subject_punctuation_check = function(header) {
  # check if subject field exists in header
  if (max(grepl("^Subject:", header)) == 0) {
    return (NA)
  }

  subject = header[grep("^Subject:", header)]
  return (grepl("[a-zA-Z]+[0-9[:punct:]]+[a-zA-Z]+",
                subject))
}

# the hour in the day the mail was sent (0-23)
hour_sent = function(header) {
  # check if subject field exists in header
  if (max(grepl("^Date:", header)) == 0) {
    return (NA)
  }

  date = tolower(header[grep("^Date:", header)])
  pattern = regexpr("[0-9]{1,2}:[0-9]{1,2}", date)
  time = regmatches(date, pattern)
  hour = strsplit(time, ":")[[1]][1]
  return (as.integer(hour))
}

# whether the header states that the message is a multipart/text
multipart_text = function(header) {
  # check if subject field exists in header
  if (max(grepl("^Content-[Tt]ype:", header)) == 0) {
    return (NA)
  }

  content_type = header[grep("^Content-[Tt]ype:", header)]
  #print(content_type)
  return (grepl("multipart", content_type))
}

# returns TRUE if html body contains images, ie. <img...>
contains_images = function(header, body) {
  # returns NA if body of message is not html
  if (max(grepl("Content-[Tt]ype:.*html", header)) == 0) {
    return (NA)
  }
  if (max(grepl("<img.*>", body)) == 1){
    return (TRUE)
  } else{
```

```
    return (FALSE)
  }
}

# returns TRUE if PGP or GPG signed
is_PGP_signed = function(content) {
  if (max(grepl("(PGP|GPG) SIGNATURE", toupper(content))) == 1){
    return (TRUE)
  } else{
    return (FALSE)
  }
}

# return the proportion of any HTML text
percent_HTML_tags = function(header, body) {
  # remove attachment sections
  if (num_attachments(header, body) > 0) {
    al = get_attachment_lines(header, body)
    bod = body[-al]
  }

  if (max(grepl("^Content-[Tt]ype:.*[Hh][Tt][Mm][Ll]", header)) == 0){
    return (NA)
  }
  get_pattern = gregexpr("<.*>", body)
  not_tags = regmatches(body, get_pattern, invert = TRUE)
  html_tags = regmatches(body, get_pattern, invert = FALSE)
  return (sum(nchar(html_tags)) / (sum(nchar(not_tags)) + sum(nchar(html_tags))) * 100)
}

# check if subject line contains spam words
subject_spam_words = function(header) {
  # check if subject field exists in header
  if (max(grepl("^Subject:", header)) == 0) {
    return (NA)
  }

  subject = tolower(header[grep("^Subject:", header)])
  spam1 = "(viagra|pounds|free|weight|guarantee|millions|dollars|credit)"
  spam2 = "(risk|prescription|generic|drug|money back|credit card)"
  match1 = grepl(spam1, subject)
  match2 = grepl(spam2, subject)
  return (match1 | match2)
}

# percentage of blanks in the subject
percent_subject_blanks = function(header) {
  # check if subject field exists in header
  if (max(grepl("^Subject:", header)) == 0) {
    return (NA)
  }

  subject = header[grep("^Subject:", header)]
```

```
  blanks = length(gregexpr(" ", subject)[[1]]) - 1
  m = regexpr(" ", subject)
  sub_len = nchar(regmatches(subject, m, invert = TRUE)[[1]][2])
  return (blanks / sub_len * 100)
}

# no message id hostname
message_id_has_no_hostname = function(header) {
  message_id = header[grep("^Message-Id", header)[1]]
  pattern = "^Message-Id: <.+>"
  return (grepl(pattern, message_id) == FALSE)
}

# returns TRUE if From: field ends with numbers
from_numeric_end = function(header) {
  # check if subject field exists in header
  if (max(grepl("^Subject:", header)) == 0) {
    return (NA)
  }

  subject = header[grep("^Subject:", header)]
  return (grepl("[0-9]$", subject))
}

# check if subject of the mail is in caps
is_yelling = function(header) {
  # check if subject field exists in header
  if (max(grepl("^Subject:", header)) == 0) {
    return (NA)
  }

  subject = header[grep("^Subject:", header)]
  pattern = "^Subject:.*([Rr]e)*[^a-z]+$"
  return (grepl(pattern, subject))
}

# returns percentage of body that is a forwarded message
percent_forwards = function(header, body) {
  if (max(grepl("^Subject:.*[Ff]wd", header))== 1){
    return (100)
  } else if (max(grepl("\\b(begin|end) forward", tolower(body))) == 1){
    fwd_index = grep("\\b(begin|end) forward", tolower(body))
    fwd_lines = max(fwd_index) - min(fwd_index)
    return (fwd_lines / num_lines_in_body(header, body) * 100)
  } else if (max(grepl("^Subject:.*[Ff]wd", body))== 1){
    if (max(grepl("^Content-[Tt]ype:.*multipart", header)) == 1){
      boundary = strsplit(header[grepl("boundary=", header)], "=")[[1]][2]
      boundary = trimws(boundary, whitespace = '\"')
      idx = grep(boundary, body)
      for (i in length(idx):1){
        chunk = body[-c(1:idx[i])]
        is_fwd = (max(grepl("^Subject:.*[Ff]wd", chunk)) == 1)
        if (is_fwd == TRUE){
```

```
          return (length(chunk) / num_lines_in_body(header, body) * 100)
        }
      }
    }
  }
  return (0)
}

# returns true if message is an original message
is_original_message = function(header, body) {
  # remove attachment sections
  if (num_attachments(header, body) > 0) {
    al = get_attachment_lines(header, body)
    bod = body[-al]
  }

  if (max(grepl("original message", body, ignore.case = TRUE)) == 0) {
    return (TRUE)
  } else{
    return (FALSE)
  }
}

# returns TRUE if message contains Dear as a form of intro
is_dear = function(body) {
  if (max(grepl("^[[:space:]]?[Dd]ear", body)) == 1){
    return (TRUE)
  } else {
    return (FALSE)
  }
}

# returns TRUE if message consists of format <person> wrote: <content>
is_wrote = function(body) {
  if (max(grepl("[Ww]rote:", body)) == 1) {
    return (TRUE)
  } else{
    return (FALSE)
  }
}

# returns average word length in body
average_word_length = function(header, body) {
  # remove attachment sections
  if (num_attachments(header, body) > 0) {
    al = get_attachment_lines(header, body)
    bod = body[-al]
  }

  # remove emails
  body = gsub("[a-zA-Z0-9[:punct:]]+@[a-zA-Z0-9]+(\\.[a-zA-Z0-9]+)+", "", body)
  # remove websites
  body = gsub("(https?:\\/\\/)?(www\\.)[a-zA-Z0-9[:punct:]]+", "", body)
```

```
  # remove html tags
  body = gsub("<[a-zA-Z0-9[:punct:]]+>", "", body)

  get_pattern = gregexpr("[[:alpha:][:digit:]]+", body)
  words = regmatches(body, get_pattern)
  nwords = sum(lengths(words))        # total number of words
  ncharacter = 0                      # total number of alphanumeric characters
  for (i in 1:length(words)){
    ncharacter = ncharacter + sum(nchar(words[[i]]))
  }
  return (ncharacter / nwords)
}

# returns number of dollar signs in body of message
num_dollar_signs = function(body) {
  get_pattern = gregexpr("\\$", body)
  dollars = regmatches(body, get_pattern)
  return (sum(lengths(dollars)))
}

# read files in a directory and output a data frame with relevant data
read_files = function(path) {
  # get list of files in folder
  files = list.files(path)
  # allocate space for data frame
  df = derived_df()
  # from each file, get the necessary details (write in another function)
  for (i in 1:length(files)) {
    # append the results into data frame
    file = paste(path, "/", files[i], sep = "")
    if (sum(grepl("^[Ff]rom", readLines(file, warn = FALSE))) == 0){
      next
    }
    df[i,] = get_row_data(file)
  }
  # return data frame
  return (df)
}
```

To create the full data frame, we should just combine the data frames obtained from each directory, as shown below:

```
dirs = list.dirs(path = ".", recursive = FALSE)

df = rbind(read_files(dirs[2]),
           read_files(dirs[3]),
           read_files(dirs[4]),
           read_files(dirs[5]),
           read_files(dirs[6]))
```

Data analysis

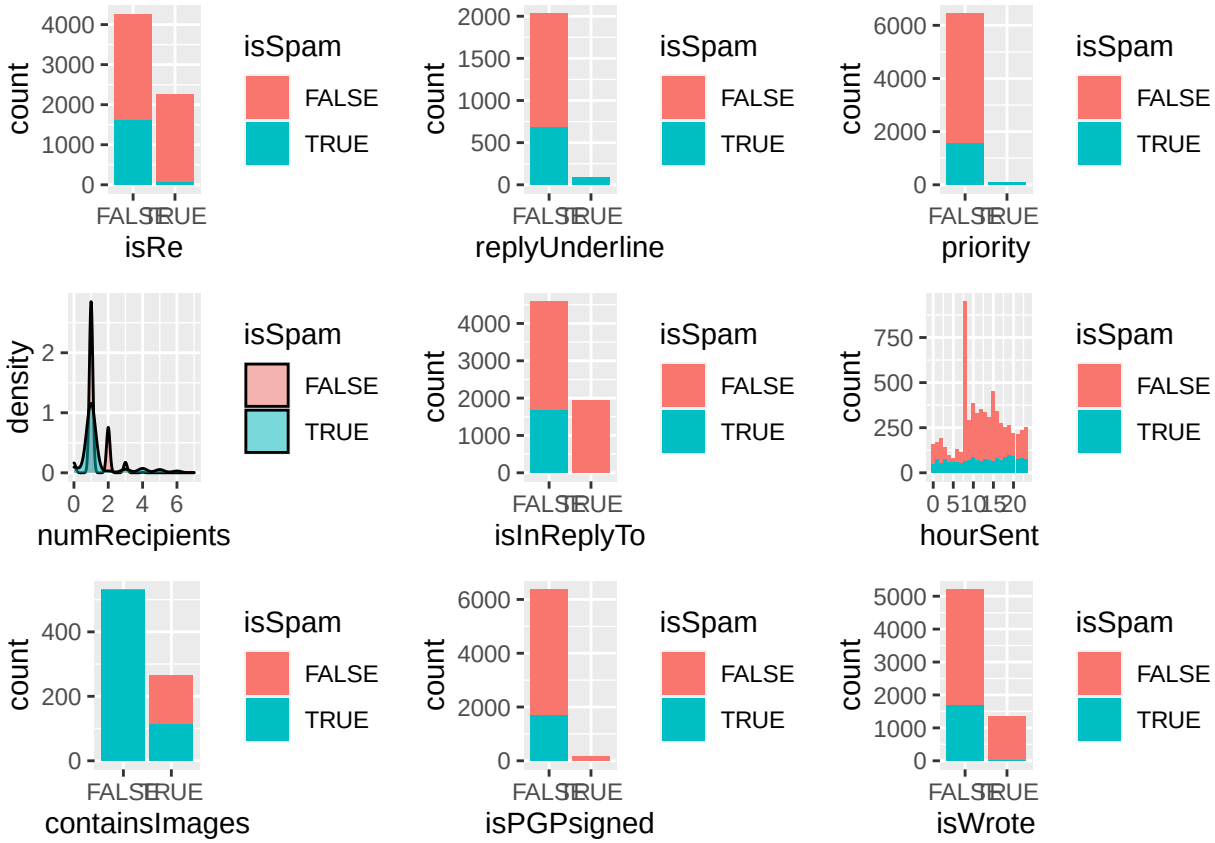
The following are some significant findings from this data set:

- Approximately 97% of emails with subjects starting with "Re:" are not spam emails.
- Approximately 97% of emails with subjects an underline in the "Reply-To" field are spam emails.
- Approximately 97% of high priority messages are spam emails, as compared to 75% of non-high priority messages are non-spam emails. However, there is a lack of evidence to fully support this due to the small number of high priority emails.
- Most non-spam emails are only sent to 1 or 2 recipients, as compared to spam emails that are sent to up to 7 recipients.
- All emails with "In-Reply-To:" fields are not spams.
- As for the time that emails are being sent, we find that the number of spam emails sent throughout the day is evenly spread out. On the other hand, for non-spam emails, there are significantly less emails being sent between 12-8am as compared to other times of the day. In fact, most non-spam emails are being sent between 8-9am (possibly due to most people starting work?).
- Given that an email is of HTML format, more than 99% of emails that contain no images are spam.
- All PGP signed emails are not spam. However, the number of emails that are PGP signed is very low.
- If the body of a message contains a line indicating an included message as identified by the word "wrote:", it is definitely not a spam.

[To be continued...]

```
library(ggplot2)
library(gridExtra)

# isRe barplpot
isRe_bar = ggplot(subset(df, !is.na(isSpam | isRe)),
                  aes(x = isRe, fill = isSpam, na.rm = TRUE)) +
           geom_bar(na.rm = TRUE)

# replyUnderline barplot
replyUnderline_bar = ggplot(subset(df, !is.na(replyUnderline)),
                            aes(x = replyUnderline, fill = isSpam, na.rm = TRUE)) +
                     geom_bar(na.rm = TRUE)

# priority barplot
priority_bar = ggplot(subset(df, !is.na(priority)),
                      aes(x = priority, fill = isSpam, na.rm = TRUE)) +
                geom_bar(na.rm = TRUE)

# numRecipients barplot
numRecipients_bar = ggplot(subset(df, !is.na(numRecipients)),
                      aes(x = numRecipients, fill = isSpam, na.rm = TRUE)) +
                    geom_density(alpha = 0.5)

# isInReplyTo barplot
isInReplyTo_bar = ggplot(subset(df, !is.na(isInReplyTo)),
                            aes(x = isInReplyTo, fill = isSpam, na.rm = TRUE)) +
                      geom_bar(na.rm = TRUE)

# hourSent barplot
```

```
hourSent_bar = ggplot(subset(df, !is.na(hourSent)),
                      aes(x = hourSent, fill = isSpam, na.rm = TRUE)) +
                 geom_bar(na.rm = TRUE)

# barplot of whether HTML email contains images
images_bar = ggplot(subset(df, !is.na(containsImages)),
                    aes(x = containsImages, fill = isSpam, na.rm = TRUE)) +
               geom_bar(na.rm = TRUE)

# barplot of whether emails are PGP signed
PGP_bar = ggplot(subset(df, !is.na(isPGPsigned)),
                 aes(x = isPGPsigned, fill = isSpam, na.rm = TRUE)) +
          geom_bar(na.rm = TRUE)

# isWrote barplot
wrote_bar = ggplot(subset(df, !is.na(isWrote)),
                   aes(x = isWrote, fill = isSpam, na.rm = TRUE)) +
            geom_bar(na.rm = TRUE)

grid.arrange(isRe_bar, replyUnderline_bar, priority_bar,
             numRecipients_bar, isInReplyTo_bar, hourSent_bar,
             images_bar, PGP_bar, wrote_bar,
             nrow = 3)
```



- A huge proportion of the emails do not contain forwarded parts.

- Spam emails are either regular spam emails, or forwarded spam emails. They do not contain additional content as do some non-spam emails.

[To be continued...]

```
percentageForwards_disc = ifelse(df$percentForwards == 0,
                                 "0",
                                 ifelse(df$percentForwards < 100,
                                        "0.1-99.9",
                                        "100"))
table(df$isSpam, percentageForwards_disc,
      dnn = c("isSpam", "percentForwards"))
```

```
##        percentForwards
## isSpam     0 0.1-99.9  100
##   FALSE 4783       25   56
##   TRUE  1668        0    8
```

- When comparing the subject field in spam and non-spam emails, there is a larger proportion (97%) of non-spam emails that do not contain exclamtion marks as compared to spam emails(75%).
- Additionally, non-spam emails do not contain more than 3 exclamation marks in their subject field.

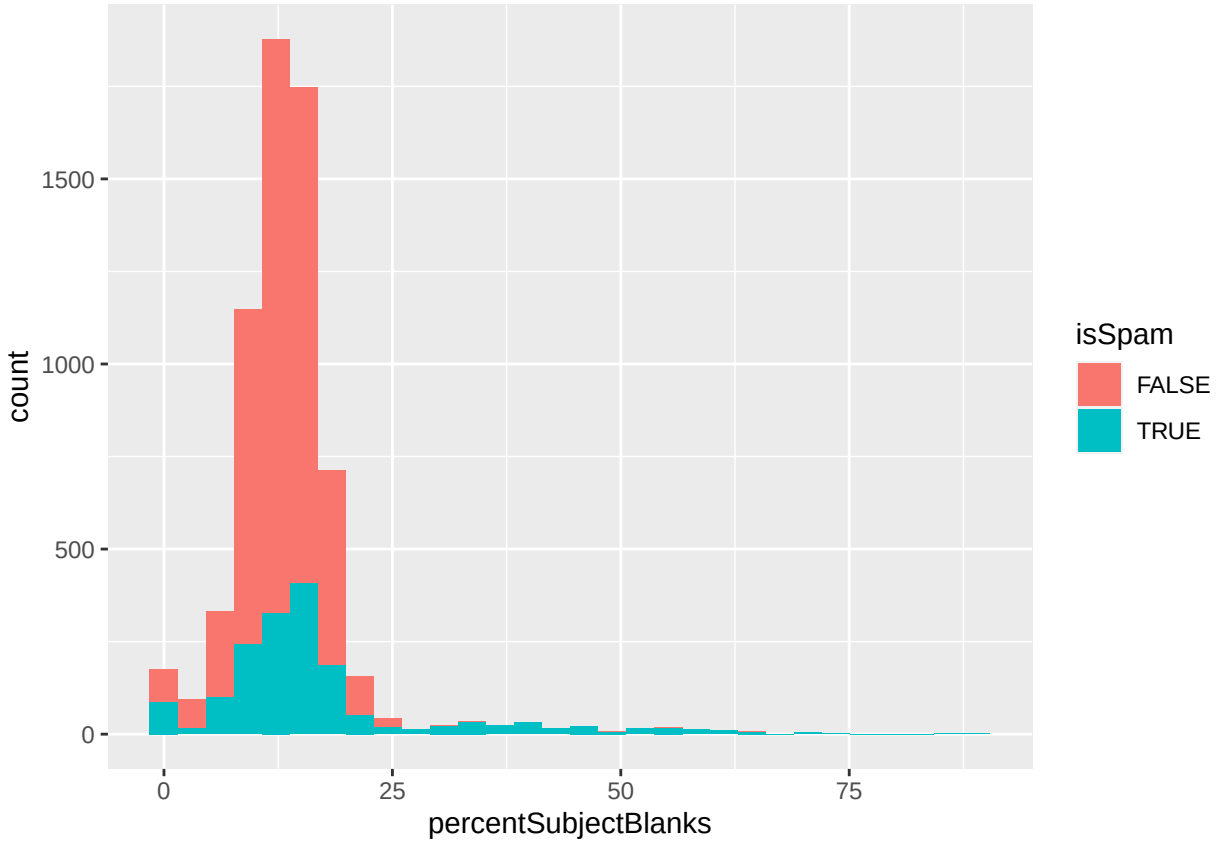
[To be continued...]

```
subjectExclamationCount =
  ifelse(df$subjectExclamationCount == 0,
         "0",
         ifelse(df$subjectExclamationCount <= 3,
                "1-3",
                "more than 3"))

table(df$isSpam, subjectExclamationCount,
      dnn = c("isSpam", "subjectExclamationCount"))
```

```
##        subjectExclamationCount
## isSpam     0  1-3 more than 3
##   FALSE 4714  146           0
##   TRUE  1249  406          21
```

- Subject fields with more than 25% blanks are all spam emails.

```
percentSubjectBlanks_hist = ggplot(subset(df, !is.na(percentSubjectBlanks)),
                             aes(x = percentSubjectBlanks, fill = isSpam, na.rm = TRUE)) +
                           geom_histogram()

percentSubjectBlanks_hist
```


1500
1000
500
0
count
0
25
50
75
percentSubjectBlanks
isSpam
FALSE
TRUE