Reading and Organizing The Data

For this project, the main issue is that the provided email files are not in tabular form and this makes it difficult to use functions, like read.table, to read in the information in a way we can manipulate it. To fix this problem, I used regular expressions to format the emails and generate variables, which I used to classify the ham and spam emails.

Storing the data was my first challenge. I decided on a list that contains three components(header, attachment(s), body message). To obtain this list, my plan included:

- Reading in the email
- Separating the three components
- Appending the components to the main list
- Repeating process for all files

Next, I read the prompt and viewed the first few files' patterns. I found that an empty string separated the header and body and I used this empty string to split each email. After the split, I found that the header keys are all capitalized and I used this pattern with gepl to document each line that has a header key. Next, I used cumsum and split to get each key with their values separated. Once separated, I used gsub to get the key names and substring to get the corresponding values. I placed the values into a vector and used the corresponding header keys as vector element names.

For the body and attachments, I had a conditional statement that would look to see if the Content-Type key in the header had an attachment boundary. If the boundary was missing in the Content-Type field, I read the email's entire body as the body message and placed an NA in the attachment component. If the boundary was present, I obtained the boundary and used it will grepl to document where the attachments started and ended. Similar to how I split the header, I used cumsum and split to get the attachments separated and organize the body. If the attachment is a "text/plain" type, I placed this attachment into the body message.

Irregular Email Files

Working through this project, I found that some emails did not have the same format as other emails. The first one I found was file SpamAssassinTrain/spam/0000.7b1b73cf36cf9dbc3d64e3f2ee2b91f1. This file has no header or body and, therefore, I removed this file from the list of files. The second problem I encountered is related to the prompt description that keys such as Date, From, and To(In-Reply-To, Bcc, cc) are mandatory. I used regular expressions to search for these header keys and I found that there are a few files that lacked a To(In-Reply-To, Bcc, cc) field. I believe that, without these keys, we don't know where the email is going. As a result, I removed these emails from the list of available emails. Another issue was a few files(I found 3) had a ">" in front of the "Received" keys. Since I did not use the "Received" key for my variables, my code does not adjust for this. In the improvement section below, I explain my actions towards this issue and how I would improve my code. The last email irregularity I found was some emails had boundaries that did not match the boundary specified in the header or there was no boundary specified in the header, but there were attachments present in the body. To resolve this issue, I used conditional statements in my code to identify these problems and I used agrep(max=4) to find the unspecified boundaries. Since I don't know the exact boundary format in those specific emails, I used agrep to allow R to be more general in searching for the boundaries. I set the argument for agrep's "max" parameter to 4 because I did not want to find matches that were not attachment boundaries.

Matching Patterns In Each File

Pattern matching and knowing which patterns will perform the task you want with the regular expression functions(grep, grepl, regexpr, regmatches, etc.) were this project's most difficult tasks. I found that observing the patterns in the file and testing those patterns on similar strings had the best results. For example, I formed a regular expression pattern based on the key in the header, I practiced with grepl on strings that were similar to the key/value pairs and, if I was sure the pattern would work, I tested that pattern on the actual emails. To ensure that my method worked, I looked at the list header component and confirmed that I identified all the key/value pairs.

When I used gsub, there were a few times where I had to change my initial pattern to be more general with the different email formats. For example, I used a pattern for gsub that was specific for quotation enclosed attachment boundaries. However, I found that some emails did not enclose their attachment boundary in quotations. I had to alter the pattern to accurately obtain boundaries that may or may not be enclosed by quotations.

Improvements

After working through this project, I would make a few improvements to my procedures. First, I would make my expressions more general to capture various email formats. A more general expression facilitates capturing different email components that don't conform to a specific format. For example, I can change my expression pattern to capture ">Received" as a key or ignore any non-alphabetic character that may precede a key. Another option is I can write code that detects the strings that don't conform with the pattern and, if necessary, I could alter those strings using gsub. Overall, my improvements would help the code be more versatile for emails that are not in this project.

Variable Creation

For my analysis, I decided to use these variables:

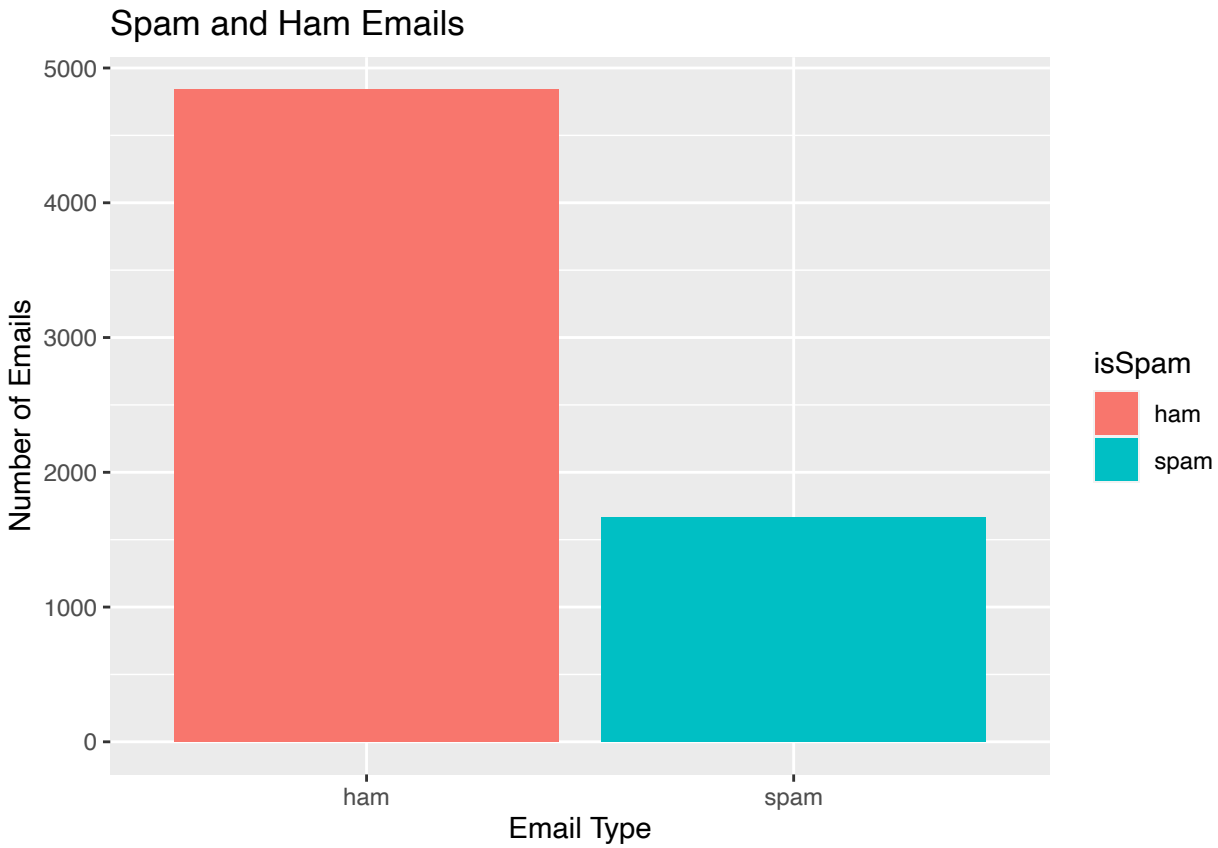
- isSpam
- isRe
- numLinesInBody
- bodyCharacterCount
- subjectExclamationCount
- numAttachments: From the prompt examples, I moved plain text attachments into the message body. A different approach may result in different attachment numbers.
- priorityt: If the X-Priority or X-Smell- Priority contained "high", "highest", 1, or 2, I classified emails as high priority.
- isInReplyTo
- multipartText: Searched for "mulipart". I made this change because I did not find any matches with "multipart/text." This may be very similar to number of attachments.
- subjectSpamWords
- percentSubjectBlanks

- messageIdHasNoHostname
- isYelling: If capital letters, exclamation points, and question marks fill 75
- fromNumericEnd
- numDollarSigns

Variable Plots

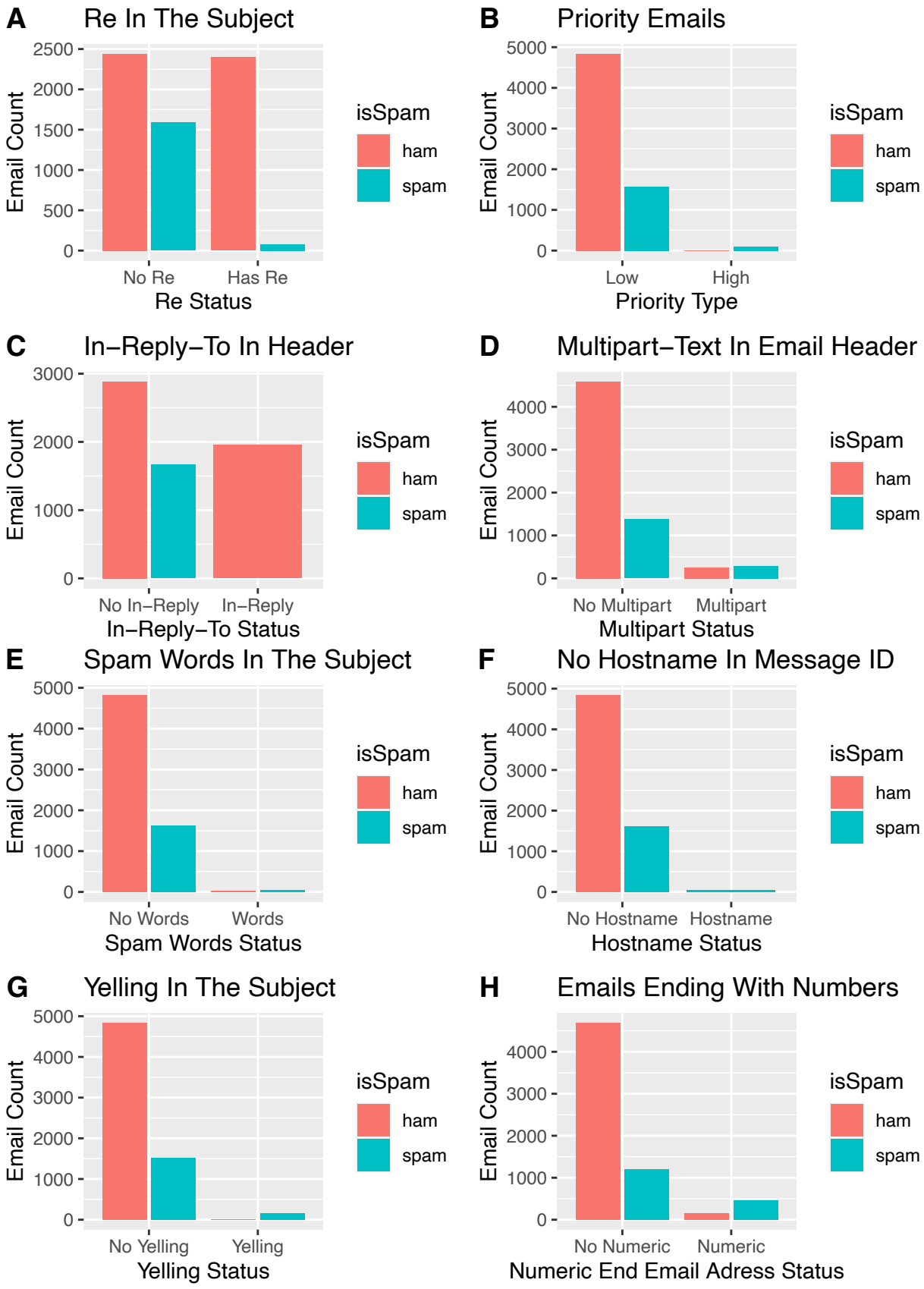
Categorical Plots:

The first plot below is the number of spam and ham in the files:



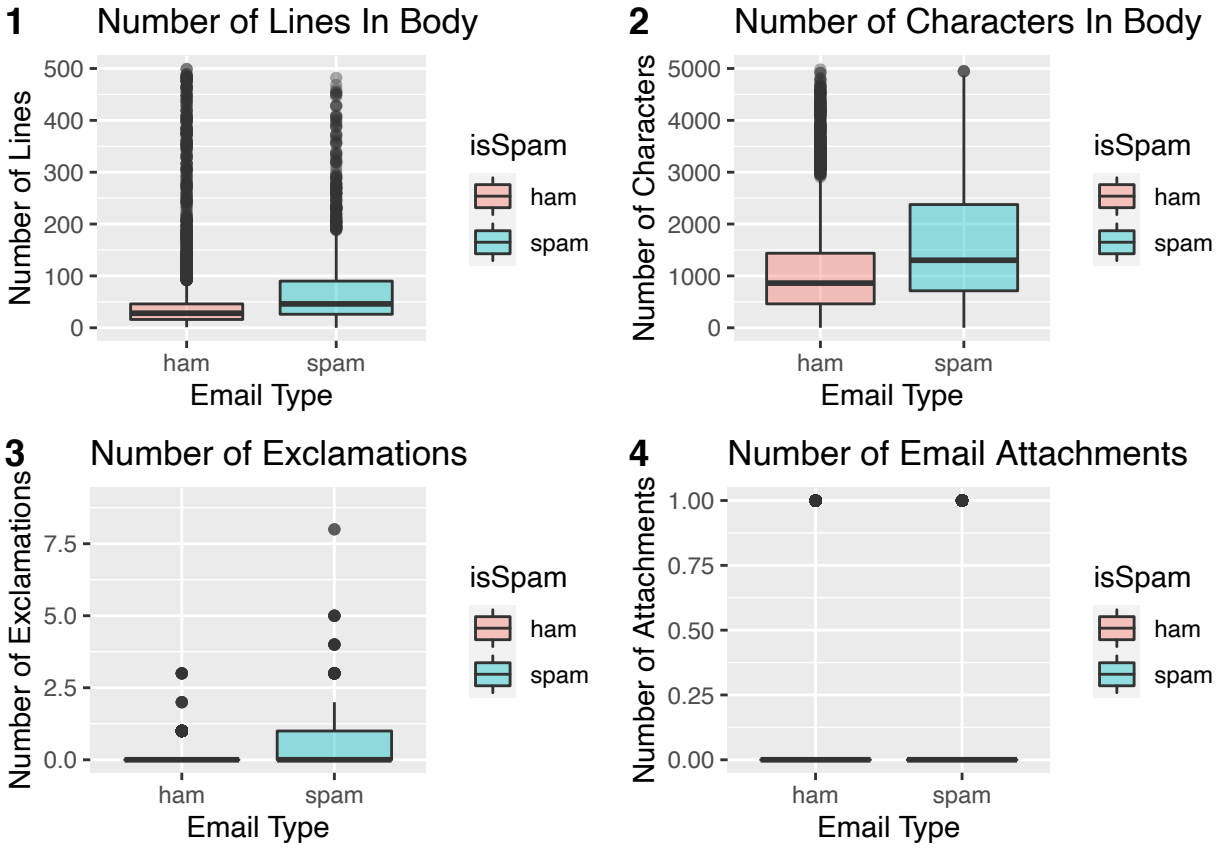
From the Spam and Ham Emails plot, we can see that the data consists largely of ham emails(4838). There are 1666 spam emails.

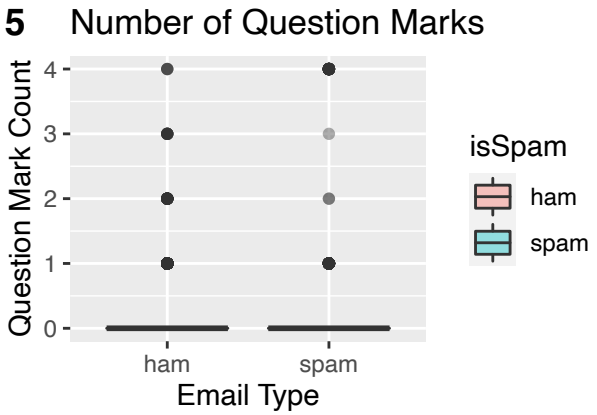
The following plots are of the categorical variables isRe, priority, isInReplyTo, multipartText, subjectSpamWords, messageIdHasNoHostname, isYelling, and fromNumericEnd. I used side-by-side bar plots to visualize the data. The turquoise bar represents spam emails and the red bar represents ham emails. To identify "good" classification predictors for spam, we want to see that there is a large difference between the number of spam and ham for when the predictor is true and false.

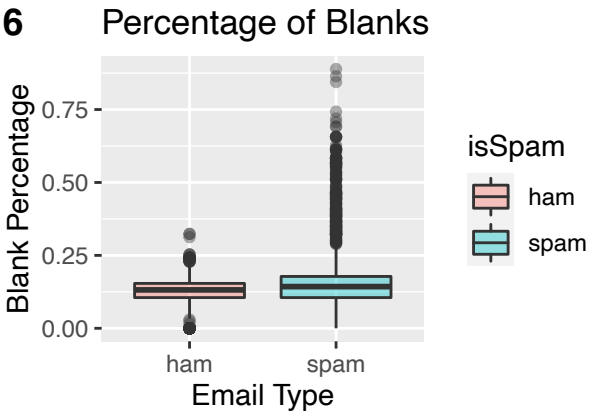

A
Re In The Subject
Email Count
2500
2000
1500
1000
500
0
No Re
Has Re
Re Status
isSpam
ham
spam
B
Priority Emails
Email Count
5000
4000
3000
2000
1000
0
Low
High
Priority Type
isSpam
ham
spam
C
In−Reply−To In Header
Email Count
3000
2000
1000
0
No In−Reply
In−Reply
In−Reply−To Status
isSpam
ham
spam
D
Multipart−Text In Email Header
Email Count
4000
3000
2000
1000
0
No Multipart
Multipart
Multipart Status
isSpam
ham
spam
E
Spam Words In The Subject
Email Count
5000
4000
3000
2000
1000
0
No Words
Words
Spam Words Status
isSpam
ham
spam
F
No Hostname In Message ID
Email Count
5000
4000
3000
2000
1000
0
No Hostname
Hostname
Hostname Status
isSpam
ham
spam
G
Yelling In The Subject
Email Count
5000
4000
3000
2000
1000
0
No Yelling
Yelling
Yelling Status
isSpam
ham
spam
H
Emails Ending With Numbers
Email Count
4000
3000
2000
1000
0
No Numeric
Numeric
Numeric End Email Adress Status
isSpam
ham
spam

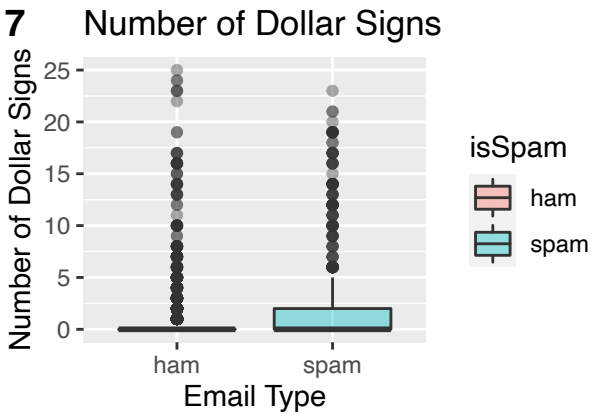
In plot A, we can see that more ham emails that have Re than spam emails. There are also fewer spam emails that don't have Re than ham emails that do have Re. Therefore, isRe may be a "good" predictor to use for classifying spam. For the priority emails plot(B), we see that no ham emails have high priority and just a few spam emails have high priority. The priority may have a slight impact on classifying spam emails. In plot C, we see that no spam emails have "In-Reply-To" in the header and slightly less than half of the ham emails have "In-Reply-To" in the header. This variable seems helpful in identifying what is not a spam email. In plot D, we see that more spam emails that have "multipart" than ham emails. We can also see that more ham emails do not have "multipart" than spam emails. Therefore, the multipart variable could be useful in classify spam. For plot E, both ham and spam emails have very few spam words in the subject. Since most observations have no spam words for both spam and ham, the subjectSpamWords does not seem to be useful in classify spam. The variables messageIdHasNoHostname and isYelling are similar to subjectSpamWords and, as a result, these variables may not be important in classifying spam. In plot H, we see that there are more spam emails than ham emails that have addresses with numeric ends. Ham emails have more email addresses than spam emails that have no numeric ends. In my opinion, the fromNumericEnd variable may be important in classify spam.

The following plots are of the numeric variables numLinesInBody, bodyCharacterCount, subjectExclamationCount, numAttachments, percentSubjectBlanks, and numDollarSigns. I used a box plot to visualize these variables. The turquoise density plot represents spam emails and the red density plot represents ham emails. To identify "good" classification predictors for spam, we hope to see that the distributions for the spam and ham emails cover different predictor values(the distributions should span different predictor values).









From plots 3,4,5, and 7, the variables subjectExclamationCount, numAttachments, subjectExclamationCount, and numDollarSign don't seem to be important in classifying spam because there doesn't appear to be an apparent difference between these variables' medians and distributions for spam and ham emails. Contrastly, numLinesInBody, bodyCharacterCount and percentSubjectBlanks seem to have slight differences between their ham and spam values.

Spam Classification Model

For this analysis, I choose to construct a classification tree to model the email data. I first formed an un-pruned tree.

```
## Call:
## rpart(formula = isSpam ~ ., data = spam_data, method = "class")
##   n= 6504
##
##           CP nsplit rel error    xerror       xstd
## 1 0.10264106      0 1.0000000 1.0000000 0.02113028
## 2 0.09183673      2 0.7947179 0.7767107 0.01932505
## 3 0.07022809      3 0.7028812 0.7100840 0.01867341
## 4 0.06542617      4 0.6326531 0.6824730 0.01838573
## 5 0.05282113      5 0.5672269 0.6422569 0.01794678
## 6 0.04621849      6 0.5144058 0.5654262 0.01703632
## 7 0.02821128      7 0.4681873 0.5078031 0.01628364
## 8 0.01260504      8 0.4399760 0.4741897 0.01581315
## 9 0.01000000      9 0.4273709 0.4609844 0.01562140
```

```
##
## Variable importance
##               isRe_vect             isInReplyTo subjectExclamationCount
##                      23                      13                      13
##    percentSubjectBlanks      bodyCharacterCount          numLinesInBody
##                       8                       8                       7
##           multipartText          fromNumericEnd          numAttachments
##                       7                       6                       6
##               isYelling                priority          numDollarSigns
##                       5                       4                       1
##
## Node number 1: 6504 observations,    complexity param=0.1026411
##   predicted class=ham   expected loss=0.2561501  P(node) =1
##     class counts:  4838  1666
##    probabilities: 0.744 0.256
##   left son=2 (2476 obs) right son=3 (4028 obs)
##   Primary splits:
##       isRe_vect               < 0.5        to the right, improve=403.5296, (0 missing)
##       isInReplyTo             < 0.5        to the right, improve=366.8008, (0 missing)
##       fromNumericEnd          < 0.5        to the left,  improve=338.6684, (0 missing)
##       subjectExclamationCount < 0.5        to the left,  improve=298.3276, (0 missing)
##       percentSubjectBlanks    < 0.2537009  to the left,  improve=271.3437, (0 missing)
##   Surrogate splits:
##       isInReplyTo < 0.5        to the right, agree=0.837, adj=0.572, (0 split)
##
## Node number 2: 2476 observations
##   predicted class=ham   expected loss=0.03150242  P(node) =0.3806888
##     class counts:  2398    78
##    probabilities: 0.968 0.032
##
## Node number 3: 4028 observations,    complexity param=0.1026411
##   predicted class=ham   expected loss=0.3942403  P(node) =0.6193112
##     class counts:  2440  1588
##    probabilities: 0.606 0.394
##   left son=6 (3538 obs) right son=7 (490 obs)
##   Primary splits:
##       subjectExclamationCount < 0.5        to the left,  improve=230.7186, (0 missing)
##       fromNumericEnd          < 0.5        to the left,  improve=178.8804, (0 missing)
##       percentSubjectBlanks    < 0.2537009  to the left,  improve=172.2135, (0 missing)
##       numLinesInBody          < 25.5       to the left,  improve=165.4479, (0 missing)
##       bodyCharacterCount      < 620.5      to the left,  improve=161.0795, (0 missing)
##   Surrogate splits:
##       numLinesInBody       < 0.5        to the right, agree=0.880, adj=0.012, (0 split)
##       bodyCharacterCount   < 17         to the right, agree=0.879, adj=0.004, (0 split)
##       percentSubjectBlanks < 0.5628283  to the left,  agree=0.879, adj=0.004, (0 split)
##       numDollarSigns       < 199.5      to the left,  agree=0.879, adj=0.004, (0 split)
##
## Node number 6: 3538 observations,    complexity param=0.09183673
##   predicted class=ham   expected loss=0.3312606  P(node) =0.5439729
##     class counts:  2366  1172
##    probabilities: 0.669 0.331
##   left son=12 (3381 obs) right son=13 (157 obs)
##   Primary splits:
##       percentSubjectBlanks < 0.2537009  to the left,  improve=141.4004, (0 missing)
```

```
##       fromNumericEnd       < 0.5        to the left,  improve=137.5803, (0 missing)
##       numLinesInBody       < 28.5       to the left,  improve=127.7571, (0 missing)
##       bodyCharacterCount   < 620.5      to the left,  improve=120.9195, (0 missing)
##       numDollarSigns       < 0.5        to the left,  improve=109.2814, (0 missing)
##
## Node number 7: 490 observations
##   predicted class=spam  expected loss=0.1510204  P(node) =0.07533825
##     class counts:    74   416
##    probabilities: 0.151 0.849
##
## Node number 12: 3381 observations,    complexity param=0.07022809
##   predicted class=ham   expected loss=0.3007986  P(node) =0.5198339
##     class counts:  2364  1017
##    probabilities: 0.699 0.301
##   left son=24 (2996 obs) right son=25 (385 obs)
##   Primary splits:
##       fromNumericEnd     < 0.5        to the left,  improve=107.1465, (0 missing)
##       multipartText      < 0.5        to the left,  improve=106.8824, (0 missing)
##       isYelling          < 0.5        to the left,  improve=104.0677, (0 missing)
##       bodyCharacterCount < 621        to the left,  improve=102.3830, (0 missing)
##       numLinesInBody     < 30.5       to the left,  improve=100.3948, (0 missing)
##   Surrogate splits:
##       numLinesInBody     < 460        to the left,  agree=0.897, adj=0.099, (0 split)
##       bodyCharacterCount < 18510.5    to the left,  agree=0.894, adj=0.065, (0 split)
##
## Node number 13: 157 observations
##   predicted class=spam  expected loss=0.01273885  P(node) =0.02413899
##     class counts:     2   155
##    probabilities: 0.013 0.987
##
## Node number 24: 2996 observations,    complexity param=0.06542617
##   predicted class=ham   expected loss=0.2556742  P(node) =0.4606396
##     class counts:  2230   766
##    probabilities: 0.744 0.256
##   left son=48 (2721 obs) right son=49 (275 obs)
##   Primary splits:
##       multipartText  < 0.5        to the left,  improve=118.58160, (0 missing)
##       numAttachments < 0.5        to the left,  improve=105.27300, (0 missing)
##       isYelling      < 0.5        to the left,  improve= 99.10955, (0 missing)
##       numLinesInBody < 32.5       to the left,  improve= 89.38102, (0 missing)
##       numDollarSigns < 0.5        to the left,  improve= 85.77132, (0 missing)
##   Surrogate splits:
##       numAttachments     < 0.5        to the left,  agree=0.989, adj=0.880, (0 split)
##       bodyCharacterCount < 48.5       to the right, agree=0.921, adj=0.142, (0 split)
##       numLinesInBody     < 3.5        to the right, agree=0.914, adj=0.062, (0 split)
##
## Node number 25: 385 observations,    complexity param=0.05282113
##   predicted class=spam  expected loss=0.3480519  P(node) =0.05919434
##     class counts:   134   251
##    probabilities: 0.348 0.652
##   left son=50 (106 obs) right son=51 (279 obs)
##   Primary splits:
##       bodyCharacterCount   < 11633      to the right, improve=94.064000, (0 missing)
##       numLinesInBody       < 242.5      to the right, improve=90.714290, (0 missing)
```

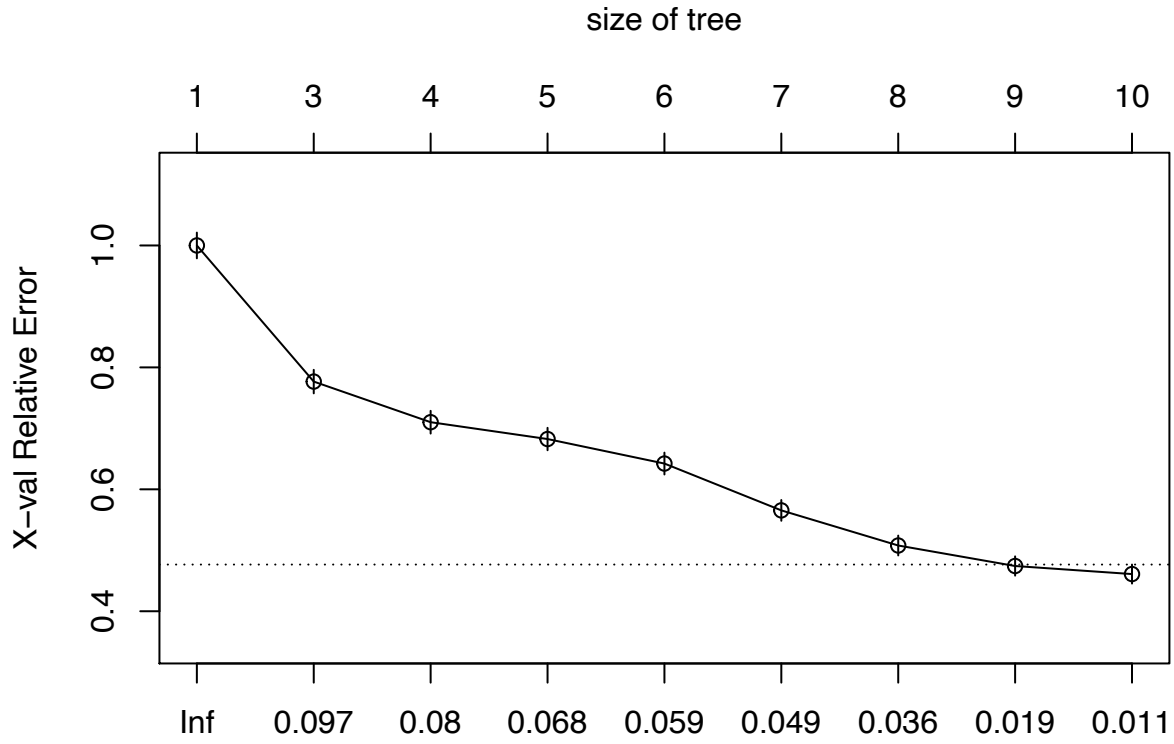
```
##       percentSubjectBlanks < 0.1024488  to the right, improve=14.046750, (0 missing)
##       isYelling            < 0.5        to the left,  improve= 4.309034, (0 missing)
##       numDollarSigns       < 3.5        to the right, improve= 3.586381, (0 missing)
##   Surrogate splits:
##       numLinesInBody < 237.5      to the right, agree=0.982, adj=0.934, (0 split)
##       numDollarSigns < 3.5        to the right, agree=0.753, adj=0.104, (0 split)
##
## Node number 48: 2721 observations,    complexity param=0.04621849
##   predicted class=ham   expected loss=0.2109519  P(node) =0.4183579
##     class counts:  2147   574
##    probabilities: 0.789 0.211
##   left son=96 (2636 obs) right son=97 (85 obs)
##   Primary splits:
##       isYelling          < 0.5        to the left,  improve=96.61118, (0 missing)
##       numDollarSigns     < 0.5        to the left,  improve=80.98710, (0 missing)
##       bodyCharacterCount < 625.5      to the left,  improve=79.02478, (0 missing)
##       numLinesInBody     < 32.5       to the left,  improve=71.44364, (0 missing)
##       priority           < 0.5        to the left,  improve=63.42517, (0 missing)
##
## Node number 49: 275 observations,    complexity param=0.01260504
##   predicted class=spam  expected loss=0.3018182  P(node) =0.04228167
##     class counts:    83   192
##    probabilities: 0.302 0.698
##   left son=98 (63 obs) right son=99 (212 obs)
##   Primary splits:
##       numLinesInBody       < 72.5       to the right, improve=21.756670, (0 missing)
##       bodyCharacterCount   < 5357.5     to the right, improve=18.888920, (0 missing)
##       percentSubjectBlanks < 0.09175084 to the right, improve= 5.040680, (0 missing)
##       numDollarSigns       < 3.5        to the right, improve= 4.265078, (0 missing)
##       isYelling            < 0.5        to the left,  improve= 2.286015, (0 missing)
##   Surrogate splits:
##       bodyCharacterCount < 3253       to the right, agree=0.942, adj=0.746, (0 split)
##       numDollarSigns     < 3.5        to the right, agree=0.833, adj=0.270, (0 split)
##
## Node number 50: 106 observations
##   predicted class=ham   expected loss=0.08490566  P(node) =0.01629766
##     class counts:    97     9
##    probabilities: 0.915 0.085
##
## Node number 51: 279 observations
##   predicted class=spam  expected loss=0.1326165  P(node) =0.04289668
##     class counts:    37   242
##    probabilities: 0.133 0.867
##
## Node number 96: 2636 observations,    complexity param=0.02821128
##   predicted class=ham   expected loss=0.1870258  P(node) =0.4052891
##     class counts:  2143   493
##    probabilities: 0.813 0.187
##   left son=192 (2589 obs) right son=193 (47 obs)
##   Primary splits:
##       priority           < 0.5        to the left,  improve=63.25498, (0 missing)
##       numDollarSigns     < 0.5        to the left,  improve=63.06305, (0 missing)
##       bodyCharacterCount < 625.5      to the left,  improve=62.52039, (0 missing)
##       numLinesInBody     < 32.5       to the left,  improve=59.52294, (0 missing)
```

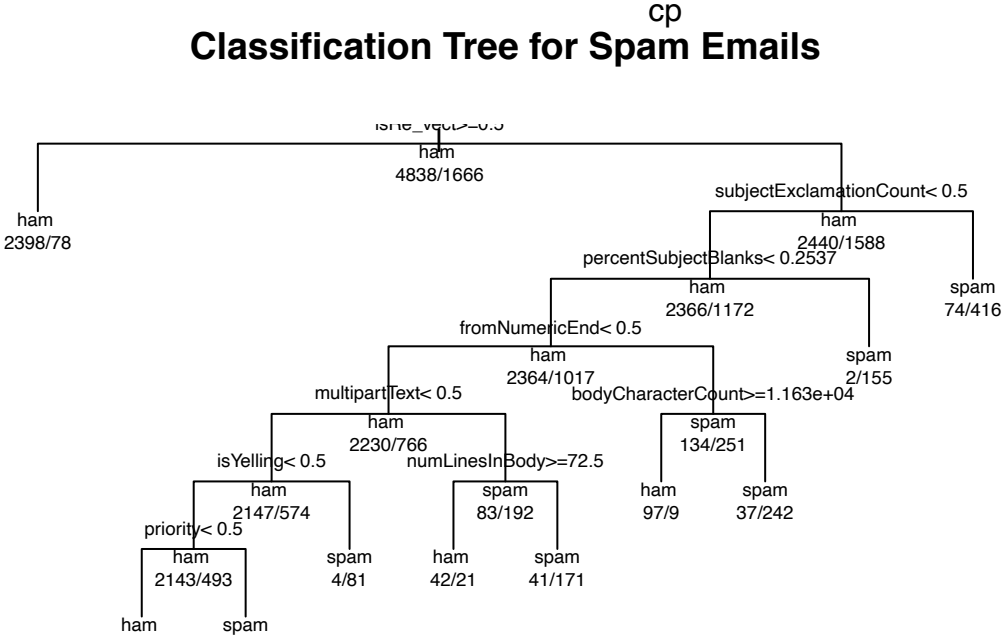
```
##       subjectQuestCount  < 3.5        to the left,  improve=28.45275, (0 missing)
##
## Node number 97: 85 observations
##   predicted class=spam  expected loss=0.04705882  P(node) =0.01306888
##     class counts:     4    81
##    probabilities: 0.047 0.953
##
## Node number 98: 63 observations
##   predicted class=ham   expected loss=0.3333333  P(node) =0.009686347
##     class counts:    42    21
##    probabilities: 0.667 0.333
##
## Node number 99: 212 observations
##   predicted class=spam  expected loss=0.1933962  P(node) =0.03259533
##     class counts:    41   171
##    probabilities: 0.193 0.807
##
## Node number 192: 2589 observations
##   predicted class=ham   expected loss=0.1722673  P(node) =0.3980627
##     class counts:  2143   446
##    probabilities: 0.828 0.172
##
## Node number 193: 47 observations
##   predicted class=spam  expected loss=0  P(node) =0.007226322
##     class counts:     0    47
##    probabilities: 0.000 1.000

##               isRe_vect             isInReplyTo subjectExclamationCount
##               403.52962               230.93759               230.71862
##    percentSubjectBlanks      bodyCharacterCount          numLinesInBody
##               142.34213               135.01147               130.34003
##           multipartText          fromNumericEnd          numAttachments
##               118.58162               107.14653               104.35183
##               isYelling                priority          numDollarSigns
##                96.61118                63.25498                16.57392

##
## Classification tree:
## rpart(formula = isSpam ~ ., data = spam_data, method = "class")
##
## Variables actually used in tree construction:
## [1] bodyCharacterCount      fromNumericEnd          isRe_vect
## [4] isYelling               multipartText           numLinesInBody
## [7] percentSubjectBlanks    priority                subjectExclamationCount
##
## Root node error: 1666/6504 = 0.25615
##
## n= 6504
##
##         CP nsplit rel error  xerror     xstd
## 1 0.102641      0   1.00000 1.00000 0.021130
## 2 0.091837      2   0.79472 0.77671 0.019325
## 3 0.070228      3   0.70288 0.71008 0.018673
## 4 0.065426      4   0.63265 0.68247 0.018386
```

```
## 5 0.052821      5   0.56723 0.64226 0.017947
## 6 0.046218      6   0.51441 0.56543 0.017036
## 7 0.028211      7   0.46819 0.50780 0.016284
## 8 0.012605      8   0.43998 0.47419 0.015813
## 9 0.010000      9   0.42737 0.46098 0.015621
```





For the first tree, we see that the variables used in the tree are bodyCharacterCount, fromNumericEnd, isRe, isYelling, multipartText, numLinesInBody, percentSubjectBlanks, priority, and subjectExclamationCount. We can also see how important variables are. The most important variable appears to be isRe. Among

trees with 1 to 9 splits, the results show that a tree with 9 splits has the smallest complexity parameter and the smallest cross-validation error. A tree with 9 splits is easily interpretable but may not provide the accuracy we need to detect spam emails. The cross-validation error rate is 0.46459 and there is a possibility of improving this value. Therefore, I will grow a full grown-tree and prune the tree to see how the accuracy improves.

```
##
## Classification tree:
## rpart(formula = isSpam ~ ., data = spam_data, method = "class",
##     cp = -1)
##
## Variables actually used in tree construction:
##  [1] bodyCharacterCount      fromNumericEnd          isInReplyTo
##  [4] isRe_vect               isYelling               multipartText
##  [7] numAttachments          numDollarSigns          numLinesInBody
## [10] percentSubjectBlanks    priority                subjectExclamationCount
## [13] subjectQuestCount       subjectSpamWords
##
## Root node error: 1666/6504 = 0.25615
##
## n= 6504
##
##             CP nsplit rel error  xerror     xstd
## 1   0.10264106      0   1.00000 1.00000 0.021130
## 2   0.09183673      2   0.79472 0.78631 0.019414
## 3   0.07022809      3   0.70288 0.70648 0.018636
## 4   0.06542617      4   0.63265 0.67947 0.018354
## 5   0.05282113      5   0.56723 0.61885 0.017680
## 6   0.04621849      6   0.51441 0.53721 0.016676
## 7   0.02821128      7   0.46819 0.49880 0.016160
## 8   0.01260504      8   0.43998 0.47119 0.015770
## 9   0.00840336      9   0.42737 0.45138 0.015479
## 10  0.00660264     10   0.41897 0.43457 0.015225
## 11  0.00585234     11   0.41236 0.42137 0.015021
## 12  0.00570228     15   0.38896 0.41116 0.014860
## 13  0.00420168     17   0.37755 0.39736 0.014637
## 14  0.00360144     19   0.36915 0.39316 0.014568
## 15  0.00330132     22   0.35834 0.38956 0.014508
## 16  0.00300120     24   0.35174 0.38836 0.014489
## 17  0.00270108     27   0.34274 0.38295 0.014398
## 18  0.00264106     29   0.33733 0.38295 0.014398
## 19  0.00240096     35   0.32113 0.36855 0.014154
## 20  0.00225090     39   0.31152 0.36435 0.014081
## 21  0.00220088     45   0.29592 0.36435 0.014081
## 22  0.00210084     48   0.28932 0.36435 0.014081
## 23  0.00180072     50   0.28511 0.35234 0.013871
## 24  0.00160064     51   0.28331 0.35414 0.013903
## 25  0.00150060     57   0.27371 0.35114 0.013850
## 26  0.00120048     59   0.27071 0.34994 0.013828
## 27  0.00072029     76   0.24130 0.33914 0.013634
## 28  0.00060024     81   0.23770 0.33914 0.013634
## 29  0.00020008     88   0.23349 0.34154 0.013677
## 30  0.00015006     97   0.23169 0.33733 0.013601
## 31  0.00012005    105   0.23049 0.33613 0.013579
```

```
## 32  0.00000000    110   0.22989 0.33733 0.013601
## 33 -1.00000000    204   0.22989 0.33733 0.013601

##
## Classification tree:
## rpart(formula = isSpam ~ ., data = spam_data, method = "class",
##     cp = -1)
##
## Variables actually used in tree construction:
## [1] bodyCharacterCount      fromNumericEnd          isRe_vect
## [4] isYelling               multipartText           numLinesInBody
## [7] percentSubjectBlanks    priority                subjectExclamationCount
##
## Root node error: 1666/6504 = 0.25615
##
## n= 6504
##
##         CP nsplit rel error  xerror     xstd
## 1 0.102641      0   1.00000 1.00000 0.021130
## 2 0.091837      2   0.79472 0.78631 0.019414
## 3 0.070228      3   0.70288 0.70648 0.018636
## 4 0.065426      4   0.63265 0.67947 0.018354
## 5 0.052821      5   0.56723 0.61885 0.017680
## 6 0.046218      6   0.51441 0.53721 0.016676
## 7 0.028211      7   0.46819 0.49880 0.016160
## 8 0.012605      8   0.43998 0.47119 0.015770
## 9 0.010000      9   0.42737 0.45138 0.015479
```

First, we can see that using a fully grown tree improves the cross-validation error rate to 0.36074. However, this fully grown tree uses 204 splits, which is highly complex and over-fits the data. To avoid these problems, I pruned the tree and found that a tree with 76 splits provides better cross-validation error(0.35114) and is relatively more interpretable. Since predicting spam is most important, I would use the pruned tree to classify possible spam emails. The next step would be to test the model on external data and test other methods to see if we can reduce the error rate further.

Code Appendix

```
library(rlist)
library(stringr)
library(ggplot2)
library(ggpubr)
library(dplyr)
#Make the files easily attainable:
setwd("~/Desktop/STA 141B/Project 2/SpamAssassinTrain")
wd=setwd("~/Desktop/STA 141B/Project 2/SpamAssassinTrain")
folder_names = list.files(wd, full.names = TRUE)
all_files = unlist(lapply(folder_names, list.files, full.names = TRUE))

#Check to see the format of each first entry in the headers:
first_lines = sapply(all_files, readLines, n = 1)
test_first_lines = grepl("^[A-Z]|^[A-Z].*: ", first_lines)
```

```
all(test_first_lines == TRUE)
which(test_first_lines == FALSE)

#It appears as though the 4865 email does not have the same format as the
#rest of the first lines.
#To check it
readLines(all_files[4865])
#The 4865 file is not of the same format as the rest of the files:
#When writing the function we need a check for this file. Either remove or treat
#the entire email as the body.

#I will do a few more checks for the files
all_files[4865] #The file name doesn't indicate the file as a special file
readLines(all_files[4865])
#If we read the lines again in the 4865th file, we see that the each line
#starts with mv and the last line also has cmds.
#I will see if any other files have this pattern:
junk_file_ind = sapply(all_files, function(x) any(grepl("^(mv) ", readLines(x, warn = FALSE), useBytes =
#I got warnings about strings not in the locale, I turned  useBytes to TRUE to avoid the warning
#I also received warnings about incomplete final lines. It doesn't seem like this is a
#major issue.
any(junk_file_ind == TRUE)
which(junk_file_ind == TRUE)

#Now, I will check for headers missing "from": Date, From, and To (or In-Reply-To, or Bcc)
junk_file_ind1 = sapply(all_files, function(x) any(grepl("^From |^From: ", readLines(x, warn = FALSE, n=
junk_file_ind2 = sapply(all_files, function(x) any(grepl("^Date: ", readLines(x, warn = FALSE), useBytes
junk_file_ind3 = sapply(all_files, function(x) any(grepl("^To: |^In-Reply-To: |^Bcc: |^Delivered-To: |^
which(junk_file_ind1 == FALSE)
which(junk_file_ind2 == FALSE)
which(junk_file_ind3 == FALSE)
omit_files=which(junk_file_ind3 == FALSE)
#From the prompt, the To, In-Reply-To, or Bcc, keys are mandatory
#There are a few files that are missing these mandatory keys
#Since they are mandatory, I will not read these files.

#Lastly, I want to confirm the header format:
header_check = sapply(all_files, function(x) any(grepl("^[a-z].*", readLines(x, warn = FALSE, n = 6), us
any(header_check == TRUE)
which(header_check == TRUE) #Doesn't appear to be any header keys in non-capitals except for 4865

#Checking for files that don't have a blank line:
header_check = sapply(all_files, function(x) any(grepl("^$", readLines(x, warn = FALSE), useBytes = TRU
any(header_check == FALSE)
which(header_check == FALSE) #4865

header_check = sapply(all_files, function(x) any(which(readLines(x, warn = FALSE) == "") > 0))
any(header_check == FALSE)
which(header_check == FALSE)

#Check for non-alphabetic/numeric characters that start a header key:
header_check = sapply(all_files, function(x) any(grepl("^>[A-Z][a-z]+: ", readLines(x, warn = FALSE, n=
any(header_check == TRUE)
```

```
which(header_check == TRUE)
#783, 784, 2510  have a ">" character at the beginning of a key
#Since there are only 3 and I will not be using the Received: key for my
#Analysis, I will not change them.



#Now, I will test how I will break up the emails:
email_contents = readLines(all_files[4666])  #4666 for attachments

#According to the description and from observation, The first blank line splits
#the header from the rest of the body. So, I think the best way to go about this
#is to split the email into two parts and read the patterns separately.
header_body_split_pos = which(email_contents == "")[1]
header_split = email_contents[1:(header_body_split_pos-1)]
body_split = email_contents[(header_body_split_pos+1):length(email_contents)]

#Now, I want to split the header into a character vector as name:value pairs:
(header_pattern = grepl("^[A-Z]|^[A-Z].*: ", header_split)) #FIX THIS
#(header_pattern2 = grepl("^From |^[A-Z].*: ", header_split))
#I used grepl() to find each single header entry. Now, I will use cumsum() to
#group the entries and I will split the header by these groupings
header_groups = cumsum(header_pattern)
separated_header_entries = split(header_split, header_groups) #character vector
separated_header_entries = sapply(separated_header_entries, paste, collapse = " ") #Char Vector


#The next step would be to separate each entry title from the values
#To help facilitate the process, I will check if the first entry title is in the
#same format as the other titles and, if not, I will make it so. This will hopefully
#be useful when I want to split the entries:

title_pattern = regexpr("^[A-Z][^:]*:|^[A-Z][a-z]*[^[:space:]]", separated_header_entries)
header_titles_values = regmatches(separated_header_entries, title_pattern, invert = TRUE)
header_titles_values = matrix(unlist(header_titles_values), ncol = 2, byrow = 2)
headers_titles = regmatches(separated_header_entries, title_pattern, invert = FALSE)
headers_titles = gsub("^([A-Z][^[:space:]]*):", "\\1", headers_titles)
headers_values_matrix = cbind(headers_titles, header_titles_values[,2])
colnames(headers_values_matrix) = c("headers", "values")
#Create a vector:
header_values_vector = headers_values_matrix[,2]
names(header_values_vector) = headers_values_matrix[,1]

#Now, we will read the body:
#I think the best method here would to have a check on whether the email has
#attachments or not. I will first start with code that looks for the "--" at the
#beginning of each attachment to

email_attachments = list()
message_body = list()
if("Content-Type" %in% names(header_values_vector) &
  grepl(".*boundary=.+", header_values_vector["Content-Type"])){
    attachment_boundary = gsub('.*\\"(.+)\\"$', "\\1",  header_values_vector["Content-Type"]) #if issue
```

```
    #Use the boundary to find the attachments
    attachment_ind = grepl(paste0("^(--)", attachment_boundary, "-*"), body_split, useBytes = TRUE)
    #Get any text before attachments start
    #if(attachment_ind[1] != TRUE){
    # before_attachments=which(attachment_ind==TRUE)[1]
    #  message_body = append(message_body,body_split[1: before_attachments-1])
    #}
    #if(attachment_ind[length(attachment_ind)]!=TRUE){
    # end_attach_pos = which(attachment_ind==TRUE)[length(which(attachment_ind==TRUE))]
    # message_body = append(message_body, body_split[(end_attach_pos+1):length(attachment_ind)])
    #}
    #Group the attachments:
    attach_groups_nums = cumsum(attachment_ind)
    attachment_split = split(body_split, attach_groups_nums)
    #attachment_split = sapply(attachment_split, paste, collapse = " ") MAY NEED TO PAST TOGETHER
    remove_body_locate = sapply(1:length(attachment_split),
                                function(x) any(grepl('^Content-Type:', attachment_split[[x]], useBytes
    #HOW TO SEARCH THROUGH LIST^^
    remove_body_attach = sapply(1:length(attachment_split),
                                function(x) any(grepl("Content-Type: text/plain", attachment_split[[x]]
    attach_locate = which(remove_body_locate == TRUE & remove_body_attach == FALSE)
    find_body = which(remove_body_attach == FALSE & remove_body_locate==FALSE)
    body_text_attach=which(remove_body_attach == TRUE & remove_body_locate == TRUE)
    message_body = append(message_body,sapply(c(find_body, body_text_attach), function(x) attachment_spl
    email_attachments = list.append(email_attachments, sapply(attach_locate, function(x) attachment_spli
}else{
  email_attachments=NA
  message_body=body_split
}

#Now we list the elements together
#May want to unlist attachments*****************
return_value = list(header_values_vector, email_attachments , unlist(message_body))
rm(list = ls()[!(ls() %in% c('folder_names','all_files', 'omit_files'))])


##-------------------------------------------------------------------------------------------
#FUNCTION TO READ TEXT AND ANALYSIS
##-------------------------------------------------------------------------------------------
#Watch for line 4865!!!!!

#Omit these files that are missing mandatory fields
real_emails = all_files[-omit_files]

#Function to get the header:
get_header = function(header_split){
  #Split the header into a character vector as name:value pairs
  header_pattern = grepl("^[A-Z][^[:space:]]* ", header_split, useBytes = TRUE) #"^[A-Z]|^[A-Z].*: "
  #grepl("^From |^[A-Z].*: ", header_split))
  #Group the entries and I will split the header by these groupings:
  header_groups = cumsum(header_pattern)
  separated_header_entries = split(header_split, header_groups) #character vector
  separated_header_entries = sapply(separated_header_entries, paste, collapse = " ")
```

```
  #The next step would be to separate each entry title from the values
  #To help facilitate the process, I will check if the first entry title is in the
  #same format as the other titles and, if not, I will make it so. This will hopefully
  #be useful when I want to split the entries:
  title_pattern = regexpr("^[A-Z][^[:space:]]* ", separated_header_entries, useBytes = TRUE)
  header_titles_values = regmatches(separated_header_entries, title_pattern, invert = TRUE)
  header_titles_values = matrix(unlist(header_titles_values), ncol = 2, byrow = 2)
  headers_titles = regmatches(separated_header_entries, title_pattern, invert = FALSE)
  headers_titles = gsub("^([A-Z][^[:space:]]*):", "\\1", headers_titles, useBytes = TRUE) #"^([A-Z][^[:
  headers_values_matrix=cbind(headers_titles, header_titles_values[,2])
  colnames(headers_values_matrix)=c("headers", "values")
  #Create a vector:
  header_values_vector_f = headers_values_matrix[,2]
  names(header_values_vector_f) = substring(headers_values_matrix[,1], 1, nchar(headers_values_matrix[,
  return(header_values_vector_f)
}

body_attachment_split= function(attach_groups_nums, body_split){
  temp_body=list()
  temp_attach=list()
  attachment_split = split(body_split, attach_groups_nums)
  remove_body_locate = sapply(1:length(attachment_split),
                              function(x) any(grepl('^Content-Type:', attachment_split[[x]], useBytes =
  remove_body_attach = sapply(1:length(attachment_split),
                              function(x) any(grepl("Content-Type: text/plain", attachment_split[[x]],
  attach_locate=which(remove_body_locate == TRUE & remove_body_attach == FALSE)
  find_body = which(remove_body_attach == FALSE & remove_body_locate == FALSE)
  body_text_attach = which(remove_body_attach == TRUE & remove_body_locate == TRUE)
  temp_body = append(temp_body,sapply(c(find_body, body_text_attach), function(x) attachment_split[[x]]
  if(length(attach_locate)==0){
    temp_attach=list.append(temp_attach,NA)
  }else{
  temp_attach = list.append(temp_attach, sapply(attach_locate, function(x) attachment_split[[x]]))
       }
  return(list(temp_body,temp_attach))
}



get_body = function(body_split, header_items){
    email_attachments = list()
    message_body = list()
    if("Content-Type" %in% names(header_items) &
              grepl(".*boundary=.+", header_items["Content-Type"], useBytes = TRUE)){
    attachment_boundary = gsub('.*boundary=[ \\"]*([^\\"]*).*', "\\1",  header_items["Content-Type"], us
    #Use the boundary to find the attachments
    attachment_ind = grepl(paste0("^(--)", attachment_boundary, "-*"), body_split, useBytes = TRUE)
    #Group the attachments:
    attach_groups_nums = cumsum(attachment_ind)
    if(all(attach_groups_nums == 0))
      { #This is for files that have different boundaries then what was indicated in the header
       attachment_ind=agrepl(paste0("^(--)", attachment_boundary, "-*"), body_split, max = 4, useBytes =
       attach_groups_nums = cumsum(attachment_ind)
```

```
       email_items=body_attachment_split(attach_groups_nums, body_split)
       message_body=email_items[[1]]
       email_attachments = email_items[[2]]
      }else{
        #attachment_split = split(body_split, attach_groups_nums)
       # remove_body_locate = sapply(1:length(attachment_split),
                              # function(x) any(grepl('^Content-Type:', attachment_split[[x]])))
        #HOW TO SEARCH THROUGH LIST^^
        #remove_body_attach = sapply(1:length(attachment_split),
                                #function(x) any(grepl("Content-Type: text/plain", attachment_split[[x]
        #attach_locate=which(remove_body_locate == TRUE & remove_body_attach == FALSE)
       # find_body = which(remove_body_attach == FALSE & remove_body_locate == FALSE)
       # body_text_attach = which(remove_body_attach == TRUE & remove_body_locate == TRUE)
       # message_body = append(message_body,sapply(c(find_body, body_text_attach), function(x) attachme
       # email_attachments = list.append(email_attachments, sapply(attach_locate, function(x) attachmen
        email_items=body_attachment_split(attach_groups_nums, body_split)
        message_body=email_items[[1]]
        email_attachments = email_items[[2]]
      }
    }else if(any(grepl("^(--).*[multi|next]part.*-*", body_split, ignore.case = TRUE, useBytes = TRUE)==
             any(grepl("^(--).*[=]+.*Exmh.*-*", body_split, ignore.case = TRUE, useBytes = TRUE)==TRUE))
       {
      attachment_ind = grepl("^(--).*[multi|next]part.*-*", body_split, useBytes = TRUE) # most I see h
      attach_groups_nums = cumsum(attachment_ind)
      email_items=body_attachment_split(attach_groups_nums, body_split)
      message_body=email_items[[1]]
      email_attachments = email_items[[2]]
        }else{
    email_attachments = NA
    message_body = body_split
  }
  return(list(email_attachments, message_body))
}
#email_file=real_emails[5145]
read_emails= function(email_file){
  email_contents = readLines(email_file, warn=FALSE)
  #Split the email into two parts: header and body
  header_body_split_pos = which(email_contents == "")[1]
  header_split = email_contents[1:(header_body_split_pos-1)]
  body_split = email_contents[(header_body_split_pos+1):length(email_contents)]
  #Read in the Header:
  header_values_vector = get_header(header_split)
  email_body = get_body(body_split, header_values_vector) ################################
  email_attachments = email_body[[1]]
  if(length(email_body[[2]])==0){
  email_message = ""
  }else{
    email_message = email_body[[2]]
  }
  return(list(header_values_vector,  email_attachments, unlist(email_message)))
}
```

```r
all_emails_list=lapply(real_emails, read_emails)

##-------------------------------------------------------------------------------------------------------------
##-------------------------------------------------------------------------------------------------------------
#VARIABLES CODE
##-------------------------------------------------------------------------------------------------------------
##-------------------------------------------------------------------------------------------------------------

#Emails marked as spam or ham
test_ham = grepl("_ham", all_files[-omit_files])
head(test_ham)
any(test_ham == FALSE)
isSpam= ifelse(!test_ham==TRUE, "spam", "ham")

#To confirm that ham and spam were marked correctly
length(which(isSpam == 'ham')) #Find the number of Ham emails(4838)

#First variable: find the subject(if it exists) and get if Re is in the beginning:---------------------
isRe_funct = function(file_num){
  if("Subject" %in% names(all_emails_list[[file_num]][[1]])
     & grepl("^Re:|^\\[.*\\][ ]*Re:", all_emails_list[[file_num]][[1]]["Subject"], ignore.case = TRUE))
    return(TRUE)
  }
     return(FALSE)
}
isRe_vect=sapply(1:length(all_emails_list), isRe_funct)


#Second Variable: Number of lines in Body-----------------------------------------------------------------
#To get an accurate count, I will have to remove attachment separations if they are present in body
numLinesInBody_funct = function(file_num){
  if("Content-Type" %in% names(all_emails_list[[file_num]][[1]]) &
     grepl(".*boundary=.+", all_emails_list[[file_num]][[1]]["Content-Type"], useBytes = TRUE)){
     added_boundary = gsub('.*boundary=[\\"]*([^\\"]*).*', "\\1",
                           all_emails_list[[file_num]][[1]]["Content-Type"])
     added_boundary = paste0("^(--)",  added_boundary, "-*")
     num_lines= sum(!grepl(added_boundary, all_emails_list[[file_num]][[3]], useBytes = TRUE))
     return(num_lines)
     }
  return(length(all_emails_list[[file_num]][[3]]))
}
numLinesInBody = sapply(1:length(all_emails_list), numLinesInBody_funct)

#Third Variable: body character count---------------------------------------------------------------------
bodyCharacterCount_funct= function(file_num){
  if("Content-Type" %in% names(all_emails_list[[file_num]][[1]]) &
     grepl(".*boundary=.+", all_emails_list[[file_num]][[1]]["Content-Type"], useBytes = TRUE) &
           length(all_emails_list[[file_num]][[3]])>1)
    {
    added_boundary = gsub('.*boundary=[\\"]*([^\\"]*).*', "\\1",
                          all_emails_list[[file_num]][[1]]["Content-Type"])
    added_boundary = paste0("^(--)",  added_boundary, "-*")
    lines_to_count = !grepl(added_boundary, all_emails_list[[file_num]][[3]], useBytes = TRUE)
```

```
    num_char = sum(sapply(all_emails_list[[file_num]][[3]][lines_to_count], nchar, type = "bytes"))
    return(num_char)
    }else if("Content-Type" %in% names(all_emails_list[[file_num]][[1]]) &
             grepl(".*boundary=.+", all_emails_list[[file_num]][[1]]["Content-Type"], useBytes = TRUE) &
                   length(all_emails_list[[file_num]][[3]])==1)
                   {
                     return(1)
                   }
  return(sum(sapply(all_emails_list[[file_num]][[3]], nchar, type = "bytes")))
}
bodyCharacterCount = sapply(1:length(all_emails_list), bodyCharacterCount_funct)


#Fourth Variable: subject exclamation point count-----------------------------------------------------
subjectExclamationCount_funct = function(file_num){
  if("Subject" %in% names(all_emails_list[[file_num]][[1]]) &
     gregexpr("!", all_emails_list[[file_num]][[1]]["Subject"])[[1]][1] != -1) #Can also use string cou
    {
    num_excla = length(gregexpr("!", all_emails_list[[file_num]][[1]]["Subject"])[[1]])
    return(num_excla)
     }
  return(0)
}
subjectExclamationCount=sapply(1:length(all_emails_list), subjectExclamationCount_funct)


#Fifth Variable: Number of attachments----------------------------------------------------------------
numAttachments_funct = function(file_num){
  if(any(!is.na(all_emails_list[[file_num]][[2]])==TRUE)){
    num_attachments = length(all_emails_list[[file_num]][[2]])
    return(num_attachments)
  }
  return(0)
}

numAttachments=sapply(1:length(all_emails_list), numAttachments_funct)

#Sixth Variable: Number of question marks in the subject----------------------------------------------
subjectQuestCount_funct = function(file_num){
  if("Subject" %in% names(all_emails_list[[file_num]][[1]]) &
     gregexpr("\\?", all_emails_list[[file_num]][[1]]["Subject"])[[1]][1] != -1) #Can also use string c
    {
    num_excla = length(gregexpr("\\?", all_emails_list[[file_num]][[1]]["Subject"])[[1]])
    return(num_excla)
     }
  return(0)
}
subjectQuestCount=sapply(1:length(all_emails_list), subjectQuestCount_funct)


#Seventh Variable: Determining Priority---------------------------------------------------------------
#Pre-specification:
#priority_vals=table(sapply(1:600, function(x) list_test[[x]][[1]]["X-Priority"]))
```

```r
#priority_vals=table(sapply(1:600, function(x) list_test[[x]][[1]]["X-Smell-Priority"]))
# 3 seems to indicate a normal priority
# 2 seems to indicate high
# 1 seems to indicate a highest priority

priority_funct = function(file_num){
  if("X-Priority" %in% names(all_emails_list[[file_num]][[1]])){
    priority_high=grepl("high|1|2", all_emails_list[[file_num]][[1]]["X-Priority"])
    return(priority_high)
  }else if("X-Smell-Priority" %in% names(all_emails_list[[file_num]][[1]])){
    priority_high=grepl("high|1|2", all_emails_list[[file_num]][[1]]["X-Smell-Priority"])
    return(priority_high)
  }
  return(FALSE)
}

priority=sapply(1:length(all_emails_list), priority_funct)

#Eighth Variable: is "In-reply-to" in the header?-----------------------------------------------------

isInReplyTo_funct = function(file_num){
  if("In-Reply-To" %in% names(all_emails_list[[file_num]][[1]])){
    return(TRUE)
  }
  return(FALSE)
}
isInReplyTo=sapply(1:length(all_emails_list), isInReplyTo_funct)


#Ninth Variable: Multipart text-----------------------------------------------------------------------
#I look for multipart because Multipart/text did not capture a hit
multipartText_funct= function(file_num){
  if("Content-Type" %in% names(all_emails_list[[file_num]][[1]]) &
     grepl("multipart", all_emails_list[[file_num]][[1]]["Content-Type"], ignore.case = TRUE))
  {
  return(TRUE)
  }
  return(FALSE)
}

multipartText = sapply(1:length(all_emails_list), multipartText_funct)


#Tenth Variable: Subject Spam Words text--------------------------------------------------------------
subjectSpamWords_funct = function(file_num){
  if("Subject" %in% names(all_emails_list[[file_num]][[1]]))
  {
    spam_words = "\\b(viagra|pounds|free|weight|guarantee|millions|dollars|credit|risk|prescription|gen
    spam_word_found=grepl(spam_words, all_emails_list[[file_num]][[1]]["Subject"])
    return(spam_word_found)
  }
  return(FALSE)
}
```

```
subjectSpamWords=sapply(1:length(all_emails_list), subjectSpamWords_funct)

which(subjectSpamWords==TRUE)

#Eleventh Variable: percent of blanks in the subject-----------------------------------------------------

percentSubjectBlanks_funct = function(file_num){
  if("Subject" %in% names(all_emails_list[[file_num]][[1]]) &
     gregexpr("[[:space:]]", all_emails_list[[file_num]][[1]]["Subject"])[[1]][1] != -1) #Can also use
  {
    num_spaces = length(gregexpr("[[:space:]]", all_emails_list[[file_num]][[1]]["Subject"])[[1]])
    total_characters = nchar(all_emails_list[[file_num]][[1]]["Subject"], type = "bytes")
    return(num_spaces/total_characters)
  }
  return(0)
}
percentSubjectBlanks=sapply(1:length(all_emails_list), percentSubjectBlanks_funct)


#Twelfth Variable: No hostname---------------------------------------------------------------------------
messageIdHasNoHostname_funct = function(file_num){
  if("Message-Id" %in% names(all_emails_list[[file_num]][[1]]) &
     grepl("[^@#]+[^-.@]@[^@#]+.[^@#]+$", all_emails_list[[file_num]][[1]]["Message-Id"])==TRUE) #Or us
  {
    return(FALSE)
  }else if("Message-ID" %in% names(all_emails_list[[file_num]][[1]]) &
           grepl("[^@#]+[^-.@]@[^@#]+.[^@#]+$", all_emails_list[[file_num]][[1]]["Message-ID"])==TRUE)
    {
    return(FALSE)
    }
  return(TRUE)
}
messageIdHasNoHostname=sapply(1:length(all_emails_list), messageIdHasNoHostname_funct)


#Thirteenth Variable: Is the subject in all capital Letters----------------------------------------------
#For this variable, I will say that a subject that has 75% or more caps is true
#I count ! and ? because they are related to yelling
isYelling_funct = function(file_num){
  if("Subject" %in% names(all_emails_list[[file_num]][[1]]))
  {
    num_spaces=length(gregexpr("[^][a-z 0-9@#]", all_emails_list[[file_num]][[1]]["Subject"])[[1]])  #"
    total_characters = nchar(all_emails_list[[file_num]][[1]]["Subject"], type = "bytes")
    if((num_spaces/total_characters)>.75){
      return(TRUE)
    }
  }
  return(FALSE)
}
isYelling=sapply(1:length(all_emails_list), isYelling_funct)

#Fourteenth Variable: when the user login ends in numbers-----------------------------------------------
fromNumericEnd_funct = function(file_num){
```

```
  if("From" %in% names(all_emails_list[[file_num]][[1]]) &
     grepl("^[^@]*[0-9]+@[^@]+", all_emails_list[[file_num]][[1]]["From"]) == TRUE)
  {
    return(TRUE)
  }
  return(FALSE)
}
fromNumericEnd=sapply(1:length(all_emails_list), fromNumericEnd_funct)


#Fifteenth Variable: number of dollar signs in the body of the message--------------------------------
numDollarSigns_funct = function(file_num){
    if(any(str_count(all_emails_list[[file_num]][[3]], "\\$")!=0))
    {
      num_dollar_signs = sum(str_count(all_emails_list[[file_num]][[3]], "\\$"))
      return(num_dollar_signs)
    }
    return(0)
  }
numDollarSigns = sapply(1:length(all_emails_list), numDollarSigns_funct)


#Put all variables in a data frame:
spam_data=data.frame(isSpam, isRe_vect, numLinesInBody,
           bodyCharacterCount, subjectExclamationCount,
           numAttachments, subjectQuestCount, priority, isInReplyTo, multipartText,
           subjectSpamWords, percentSubjectBlanks, messageIdHasNoHostname,isYelling,
           fromNumericEnd, numDollarSigns)

##-------------------------------------------------------------------------------------------------------
##-------------------------------------------------------------------------------------------------------
#Plotting and Forming a model:
##-------------------------------------------------------------------------------------------------------
##-------------------------------------------------------------------------------------------------------

#Plots:
#For numeric variables, I will plot a density plot
#For categorical variables, I will generate a side-by-side bar graph

isSpam_plot = ggplot(spam_data)+
  geom_bar(mapping = aes(x = isSpam, fill=isSpam)) +
                               xlab("Email Type") +
                                ylab("Number of Emails")+
                                labs(title = "Spam and Ham Emails")

isRe_vect_plot = ggplot(spam_data) +
                geom_bar(mapping= aes(x = isRe_vect, fill = isSpam),position = "dodge2") +
                scale_x_discrete(labels=c("No Re", "Has Re")) +
                xlab("Re Status") +
                ylab("Email Count")+
                labs(title = "Re In The Subject")
```

```
priority_plot =  ggplot(spam_data) +
                 geom_bar(mapping = aes(x =  priority, fill = isSpam), position = "dodge2") +
                 scale_x_discrete(labels=c("Low", "High")) +
                 xlab("Priority Type") +
                 ylab("Email Count") +
                 labs(title = "Priority Emails")


isInReplyTo_plot = ggplot(spam_data) +
                  geom_bar(aes(x = isInReplyTo, fill = isSpam),position="dodge2") +
                  scale_x_discrete(labels=c("No In-Reply", "In-Reply")) +
                  xlab("In-Reply-To Status") +
                  ylab("Email Count") +
                  labs(title = "In-Reply-To In Header")


multipartText_plot =  ggplot(spam_data) +
                      geom_bar(aes(x = multipartText, fill = isSpam), position = "dodge2") +
                      scale_x_discrete(labels=c("No Multipart", "Multipart")) +
                      xlab("Multipart Status") +
                      ylab("Email Count")+
                      labs(title = "Multipart-Text In Email Header")

subjectSpamWords_plot = ggplot(spam_data) +
                       geom_bar(aes(x = subjectSpamWords, fill = isSpam), position = "dodge2") +
                      scale_x_discrete(labels=c("No Words", "Words")) +
                      xlab("Spam Words Status") +
                      ylab("Email Count")+
                      labs(title = "Spam Words In The Subject")

messageIdHasNoHostname_plot = ggplot(spam_data) +
                        geom_bar(aes(x = messageIdHasNoHostname, fill = isSpam), position = "dodge2") +
                        scale_x_discrete(labels=c("No Hostname", "Hostname")) +
                        xlab("Hostname Status") +
                        ylab("Email Count")+
                        labs(title = "No Hostname In Message ID")

isYelling_plot = ggplot(spam_data) +
                geom_bar(aes(x = isYelling, fill = isSpam), position = "dodge2") +
                scale_x_discrete(labels=c("No Yelling", "Yelling")) +
                xlab("Yelling Status") +
                ylab("Email Count")+
                labs(title = "Yelling In The Subject")

fromNumericEnd_plot = ggplot(spam_data) +
                      geom_bar(aes(x = fromNumericEnd, fill = isSpam), position = "dodge2") +
                      scale_x_discrete(labels=c("No Numeric", "Numeric")) +
                      xlab("Numeric End Email Adress Status") +
                      ylab("Email Count")+
                      labs(title = "Emails Ending With Numbers")

numLinesInBody_plot = ggplot(spam_data, aes(x = isSpam, y=numLinesInBody, fill=isSpam)) +
                      #geom_histogram(aes(y=..density..), alpha=0.5,
```

```
                        #position="identity") +
                        geom_boxplot(alpha=0.4) +
                        ylim(0, 500)+
                        xlab("Email Type") +
                        ylab("Number of Lines")+
                        labs(title = "Number of Lines In Body")


bodyCharacterCount_plot = ggplot(spam_data, aes(x = isSpam, y = bodyCharacterCount, fill=isSpam)) +
                          #geom_histogram(aes(y=..density..), alpha=0.5,
                          #position="identity") +
                          geom_boxplot(alpha=0.4) +
                          ylim(0, 5000)+
                          xlab("Email Type") +
                          ylab("Number of Characters")+
                          labs(title = "Number of Characters In Body")


subjectExclamationCount_plot = ggplot(spam_data, aes(x = isSpam, y= subjectExclamationCount, fill=isSpa
                               #geom_histogram(aes(y=..density..), alpha=0.5,
                               #position="identity") +
                               geom_boxplot(alpha=0.4) +
                               ylim(0, 9)+
                               xlab("Email Type") +
                               ylab("Number of Exclamations") +
                               labs(title = "Number of Exclamations")

numAttachments_plot =  ggplot(spam_data, aes(x = isSpam, y = numAttachments, fill=isSpam)) +
                       #geom_histogram(aes(y=..density..), alpha=0.5,
                       #position="identity") +
                       geom_boxplot(alpha=0.4) +
                       xlab("Email Type") +
                       ylab("Number of Attachments")+
                       labs(title = "Number of Email Attachments")

subjectQuestCount_plot =  ggplot(spam_data, aes(x = isSpam, y = subjectQuestCount, fill=isSpam)) +
                          #geom_histogram(aes(y=..density..), alpha=0.5,
                          #position="identity") +
                          geom_boxplot(alpha=0.4) +
                          ylim(0, 4)+
                          xlab("Email Type") +
                          ylab("Question Mark Count")+
                          labs(title = "Number of Question Marks")

percentSubjectBlanks_plot =  ggplot(spam_data, aes(x = isSpam, y = percentSubjectBlanks, fill=isSpam)) +
                             #geom_histogram(aes(y=..density..), alpha=0.5,
                             #position="identity") +
                             geom_boxplot(alpha=0.4) +
                             xlab("Email Type") +
                             ylab("Blank Percentage")+
                             labs(title = "Percentage of Blanks")

numDollarSigns_plot = ggplot(spam_data, aes(x = isSpam, y = numDollarSigns, fill=isSpam)) +
```

```
                  #geom_histogram(aes(y=..density..), alpha=0.5,
                  #position="identity") +
                  geom_boxplot(alpha=0.4) +
                  ylim(0, 25) +
                  xlab("Email Type") +
                  ylab("Number of Dollar Signs")+
                  labs(title = "Number of Dollar Signs")



#Is spam plot
isSpam_plot
#categorical Data Plots
suppressWarnings(ggarrange(isRe_vect_plot, priority_plot, isInReplyTo_plot, multipartText_plot, ncol = 1
#Part 2
suppressWarnings(ggarrange(subjectSpamWords_plot,messageIdHasNoHostname_plot, isYelling_plot, fromNumeri
#Numeric variables:
suppressWarnings(ggarrange(numLinesInBody_plot, bodyCharacterCount_plot, subjectExclamationCount_plot, n

suppressWarnings(ggarrange(subjectQuestCount_plot, percentSubjectBlanks_plot , numDollarSigns_plot,
         ncol = 2, nrow=2, labels = c("5", "6", "7")))
library(rpart)
#Decision tree 1:
spam_decision=invisible(rpart(isSpam~., data=spam_data, method="class"))
#View the results:
tree_summar=invisible(summary(spam_decision))
tree_summar$variable.importance
printcp(spam_decision)
#plot the cross validation results:
plotcp(spam_decision)
#Plot the decision tree:
plot(spam_decision, uniform=TRUE,
     main="Classification Tree for Spam Emails")
text(spam_decision, use.n=TRUE, all=TRUE, cex=.60)


#Full grown tree
spam_decision2 = invisible(rpart(isSpam~., data=spam_data, method="class", cp=-1))
printcp(spam_decision2)
#Prune the tree to prevent overfitting
spam_decision2_pruned = invisible(prune(spam_decision2, cp=spam_decision$cptable[which.min(spam_decisio
printcp(spam_decision2_pruned)
```