0.0.1 Question 1c

Discuss one thing you notice that is different between the two emails that might relate to the identification of spam.

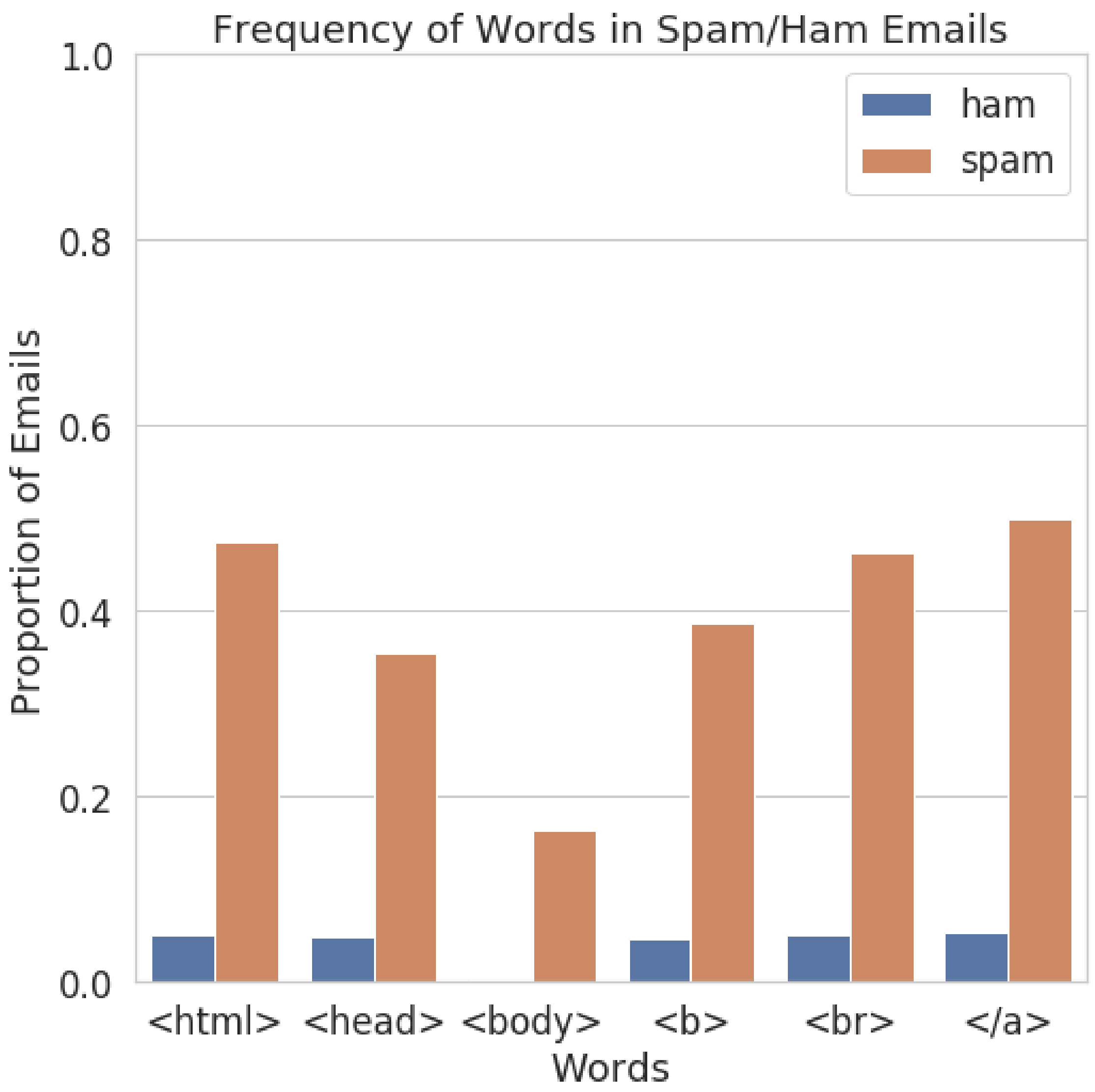
Spam is filled with HTML formatting, while Ham is plain text.

0.0.2 Question 3a

Create a bar chart like the one above comparing the proportion of spam and ham emails containing certain words. Choose a set of words that are different from the ones above, but also have different proportions for the two classes. Make sure to only consider emails from `train`.

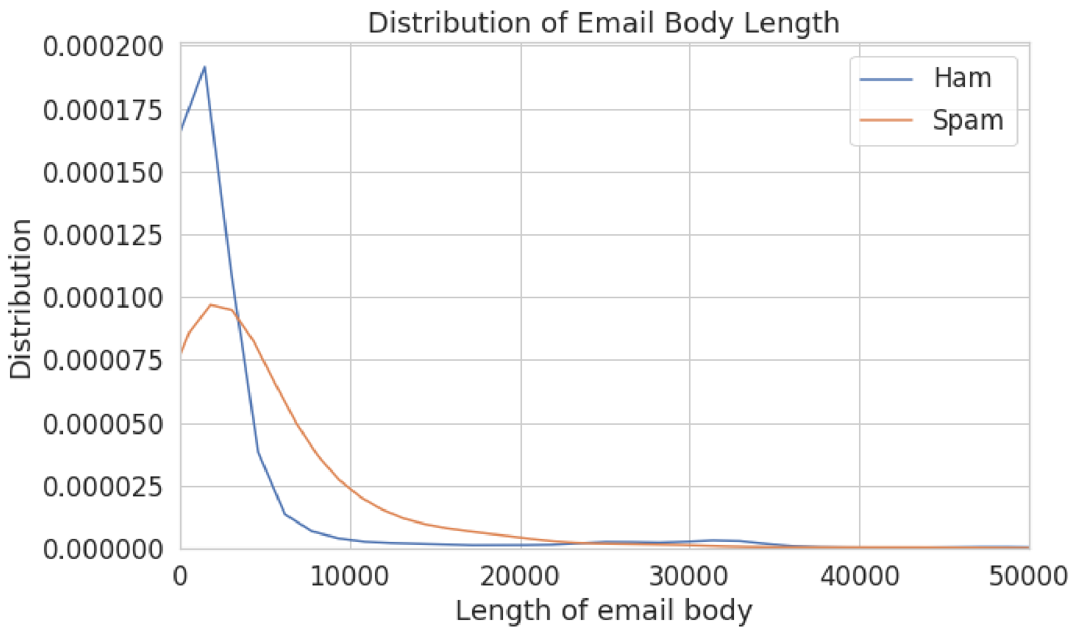
```
In [13]: train=train.reset_index(drop=True) # We must do this in order to preserve the ordering of emai

         test_words = ["<html>", "<head>", "<body>", "<b>", "<br>", "</a>"]
         frequency = words_in_texts(test_words, train["email"])
         df_test_words = pd.DataFrame(frequency, columns=test_words).assign(type=train["spam"]).replace
         df_test_words = df_test_words.melt("type")

         plt.figure(figsize=(8,8))
         sns.barplot(x="variable", y="value", hue="type", data=df_test_words, ci=None)
         plt.xlabel("Words")
         plt.ylabel("Proportion of Emails")
         plt.title("Frequency of Words in Spam/Ham Emails")
         plt.legend(title="")
         plt.ylim(0, 1)
```

```
Out[13]: (0, 1)
```


Frequency of Words in Spam/Ham Emails
ham
spam
Proportion of Emails
1.0
0.8
0.6
0.4
0.2
0.0
<html>
<head>
<body>
<b>

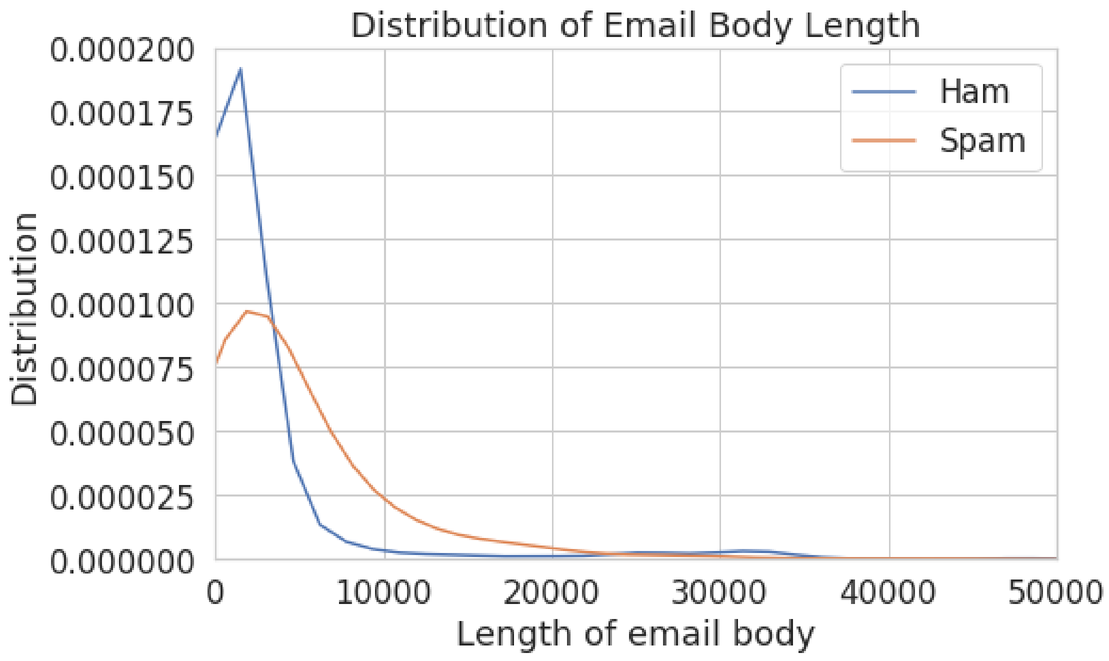
</a>
Words

0.0.3 Question 3b



Create a *class conditional density plot* like the one above (using `sns.distplot`), comparing the distribution of the length of spam emails to the distribution of the length of ham emails in the training set. Set the x-axis limit from 0 to 50000.

```
In [14]: train2 = train.copy()
         train2["Length"] = train2["email"].str.len()

         plt.figure(figsize=(8,5))
         sns.distplot(train2[train2["spam"]==0]["Length"], hist=False, label='Ham')
         sns.distplot(train2[train2["spam"]==1]["Length"], hist=False, label='Spam')
         plt.xlabel("Length of email body")
         plt.ylabel("Distribution")
         plt.title("Distribution of Email Body Length")
         plt.legend(title="")
         plt.xlim(0, 50000)
         plt.ylim(0, 0.0002)

         plt.savefig('training_conditional_densities.png')
```


Distribution of Email Body Length
Distribution
Length of email body
Ham
Spam
0.000200
0.000175
0.000150
0.000125
0.000100
0.000075
0.000050
0.000025
0.000000
0
10000
20000
30000
40000
50000

0.0.4 Question 6c

Provide brief explanations of the results from 6a and 6b. Why do we observe each of these values (FP, FN, accuracy, recall)?

A zero predictor will never predict positive, hence the 0 FP rate. Instead, every single Spam will be misidentified as Ham (100% FN). The accuracy of 74% simply reflects the ratio of Ham among all emails in the training set, but recall is 0 as none of the Spam is successfully captured.

0.0.5 Question 6e

Are there more false positives or false negatives when using the logistic regression classifier from Question 5?

There are much more false negatives (1699) compared to false positives (122) using logistic regression. Comparing logistic regression to the zero predictor (1918 fneg, 0 fpos), logistic regression has fewer false negatives and more false positives.

0.0.6 Question 6f

1. Our logistic regression classifier got 75.76% prediction accuracy (number of correct predictions / total). How does this compare with predicting 0 for every email?
2. Given the word features we gave you above, name one reason this classifier is performing poorly. Hint: Think about how prevalent these words are in the email set.
3. Which of these two classifiers would you prefer for a spam filter and why? Describe your reasoning and relate it to at least one of the evaluation metrics you have computed so far.

1. The zero predictor was ~74% accurate, so the regression classifier is only marginally better by a little over 1%.

2. The words provided probably do not have a strongly skewed split in the ratio between Ham/Spam, as many, such as ”memo,” ”bank,” or ”private,” could have been in legitimate emails, which would mean they make for poor indicators.

3. I would prefer logistic regression, as it is still an improvement in accuracy overall, even if marginal, and does capture some email as Spam, which is serving its function rather than the naive zero predictor being the same as no Spam filter whatsoever.

0.0.7 Question 7: Feature/Model Selection Process

In this following cell, describe the process of improving your model. You should use at least 2-3 sentences each to address the follow questions:

1. How did you find better features for your model?
2. What did you try that worked or didn't work?
3. What was surprising in your search for good features?

1. I began with the HTML tags, as in the beginning of the project, added some of the given words (body, offer, money), and then in trying to add words, I opened train.csv to look through the emails for frequently occuring words in the texts and tried adding them to test for accuracy.

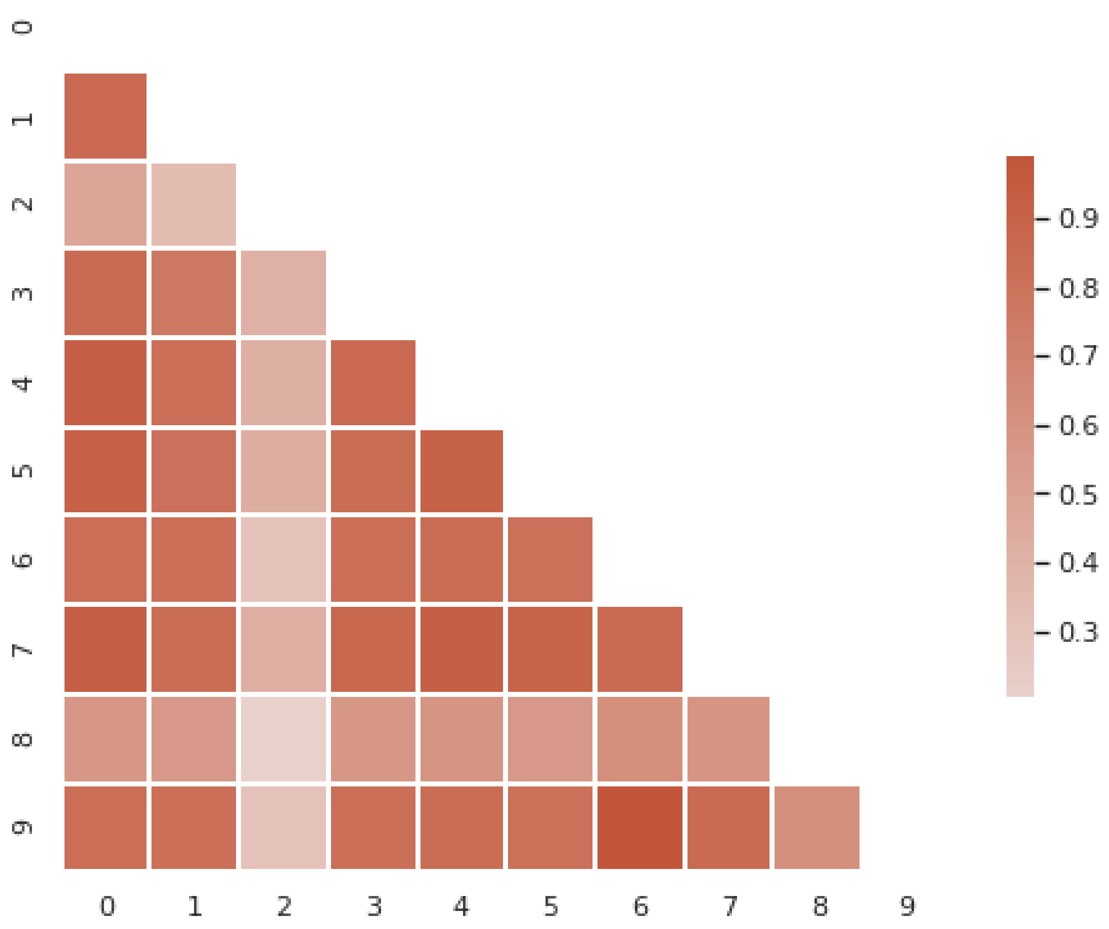
2. I tried to combine the Subject and Email body into one combined column for more words in checking, like with Fwd: or Re:, but it didn't seem to improve the accuracy of my model, so I did not include it. Most HTML tags were good, but there was major diminishing returns as HTML-heavy Spam tended to use a lot of it, whereas some did not, so it captured only one particular class of Spam.

3. I thought times, "am" and "pm," would have been useful, but it was fairly evenly split between the two. Emoji ":)" did actually help, as they tended to be Ham. Funny enough, "sincerely" is only associated with Spam.

Generate your visualization in the cell below and provide your description in a comment.

```
In [25]: # Write your description (2-3 sentences) as a comment here:
         #  HTML appears to be used widely in Spam, but HTML formatting also
         #  tends to be all-or-nothing. That is, Spam using HTML will have a lot,
         #  but some Spam won't use much, if at all. The aim of using a
         #  correlation map between HTML terms is to see which specific features
         #  are worth using, and which are simply redundant.

         # Write the code to generate your visualization here:
         #   followed suggested example and used code here from:
         #   https://seaborn.pydata.org/examples/many_pairwise_correlations.html
         html_words = ["<html>", "<head>", "<body>", "<b>", "<br>", "</a>", "</table>", "</font>", "</c
         model_xhtml = pd.DataFrame(words_in_texts(html_words, train["email"]))

         sns.set_theme(style="white")
         corr = model_xhtml.corr()
         mask = np.triu(np.ones_like(corr, dtype=bool))
         f, ax = plt.subplots(figsize=(8, 8))
         cmap = sns.diverging_palette(230, 20, as_cmap=True)
         sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.99, center=0,
                     square=True, linewidths=1, cbar_kws={"shrink": .5})
```

```
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7f19b5ddcd00>
```


0
1
2
3
4
5
6
7
8
9
0.9
0.8
0.7
0.6
0.5
0.4
0.3

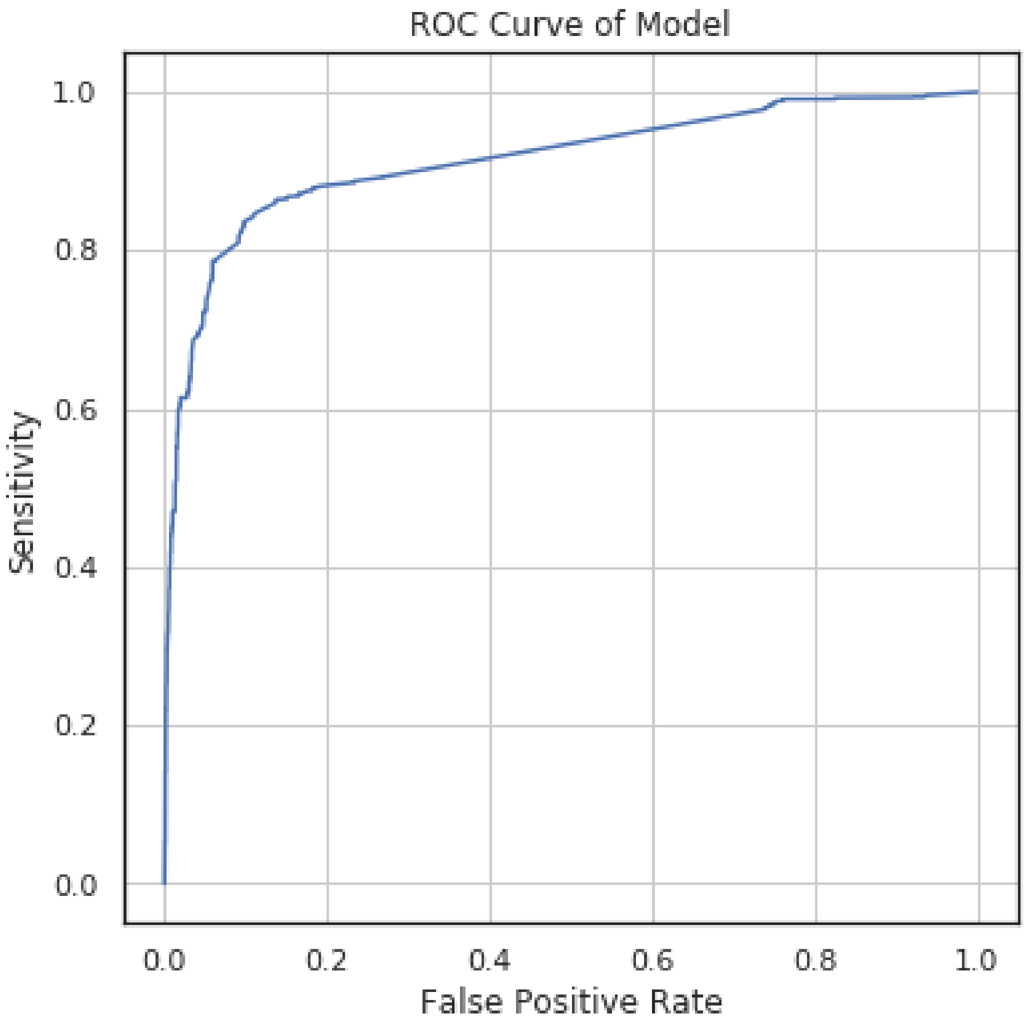
0.0.8 Question 9: ROC Curve

In most cases we won't be able to get 0 false positives and 0 false negatives, so we have to compromise. For example, in the case of cancer screenings, false negatives are comparatively worse than false positives — a false negative means that a patient might not discover that they have cancer until it's too late, whereas a patient can just receive another screening for a false positive.

Recall that logistic regression calculates the probability that an example belongs to a certain class. Then, to classify an example we say that an email is spam if our classifier gives it ≥ 0.5 probability of being spam. However, *we can adjust that cutoff*: we can say that an email is spam only if our classifier gives it ≥ 0.7 probability of being spam, for example. This is how we can trade off false positives and false negatives.

The ROC curve shows this trade off for each possible cutoff probability. In the cell below, plot a ROC curve for your final classifier (the one you use to make predictions for Gradescope) on the training data. Refer to Lecture 19 or Section 17.7 of the course text to see how to plot an ROC curve.

```
In [28]: from sklearn.metrics import roc_curve

         # Note that you'll want to use the .predict_proba(...) method for your classifier
         # instead of .predict(...) so you get probabilities, not classes

         model_words = ["<html>", "<head>", "<body>", "<b>", "<br>", "</a>", "</table>", "</font>", "</
         model_X_fit = pd.DataFrame(words_in_texts(model_words, train["email"]))
         model_Y_fit = train["spam"]
         model.fit(model_X_fit, model_Y_fit)

         # code taken and adapted from Sec 17.7, but grid disappeared
         false_positive_rate_values, sensitivity_values, thresholds = roc_curve(train["spam"], model.pr
         plt.figure(figsize=(6,6))
         plt.plot(false_positive_rate_values, sensitivity_values)
         plt.xlabel("False Positive Rate")
         plt.ylabel("Sensitivity")
         plt.title("ROC Curve of Model")
         plt.grid()
```


ROC Curve of Model
Sensitivity
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate