0.0.1 Question 1: 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 tried to find better features for my model by seeing if there was correlation between length of email and whether the email was spam or ham; I first thought that longer, messier emails would contain more characters and would probably be spam rather than ham. I also wanted to see if punctuation had correlation with spam; I think spam would have more punctuation generally because a lot of spam advertises things and would want to grab attention, so there would be more punctuation.
2. At first, I tried to graph the length by counting the number of characters within an email only, and then graphing this based on spam or ham. I realized that counting by characters, while helpful, had extremely small density values so I decided to also count by words. The small density values were still small but since there was less variation in the number of words, the density values were slightly bigger than graphing by number of characters. The distplot worked well because I was able to see a histogram and line graph as well as graph the mean on the graph.
3. For the punctuation feature, I tried to at first use the proportion of punctuation but I realized that spam emails tended to be longer and so the proportion of punctuation would not be a good classifier. This was surprising because I had not considered the length of email at first. I found that spam, on average, had more punctuation than ham emails and were also longer than ham emails.

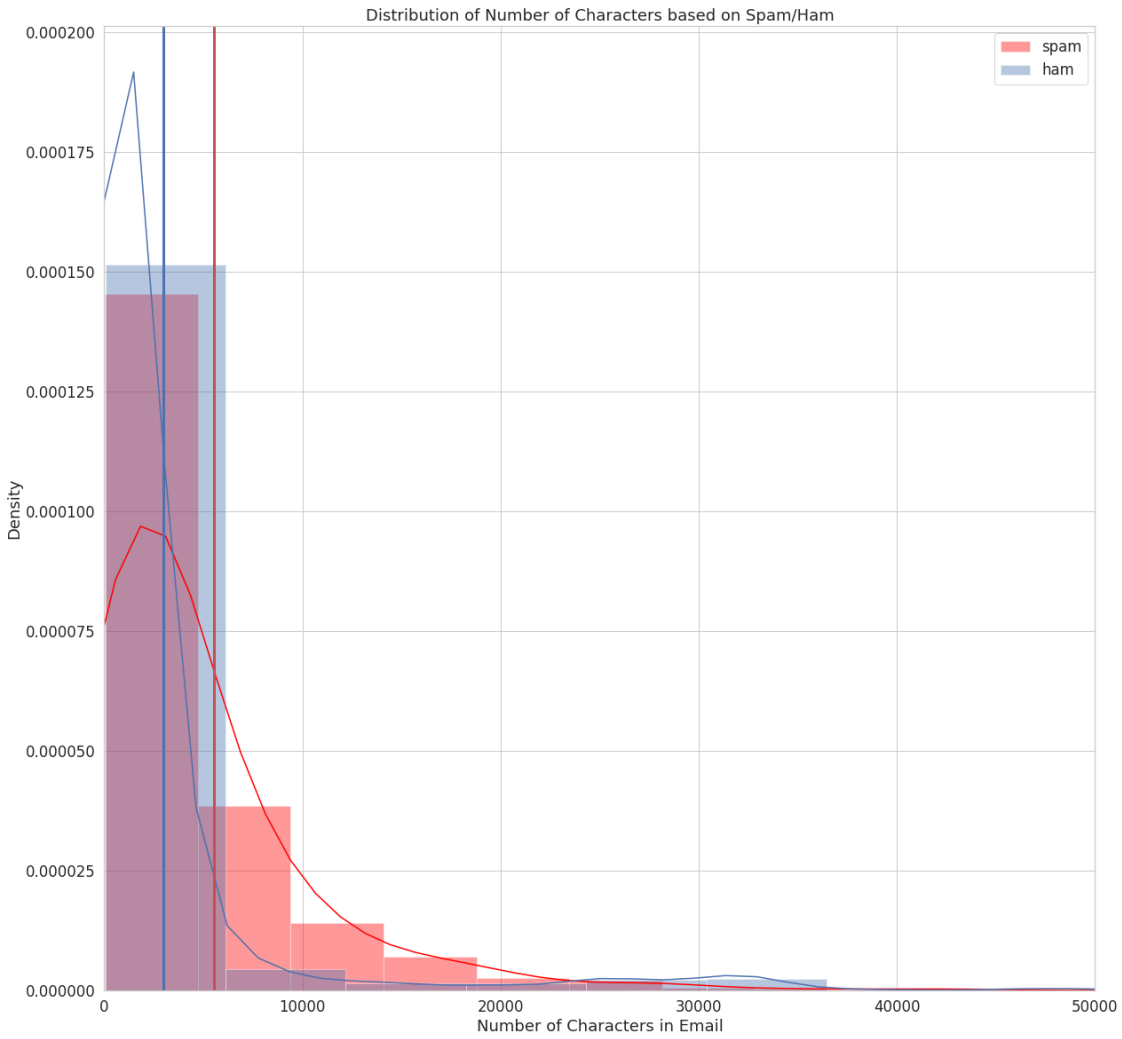
Question 2a Generate your visualization in the cell below.

In [12]:
```
def split_char(s):
    return [c for c in s]
```

In [13]:
```
train["length_char"] = train["email"].apply(lambda x: len(split_char(str(x))))
train["length_word"] = train["email"].apply(lambda x: len(str(x).split(' ')))
spam = train[train["spam"] == 1]
ham = train[train["spam"] == 0]
```

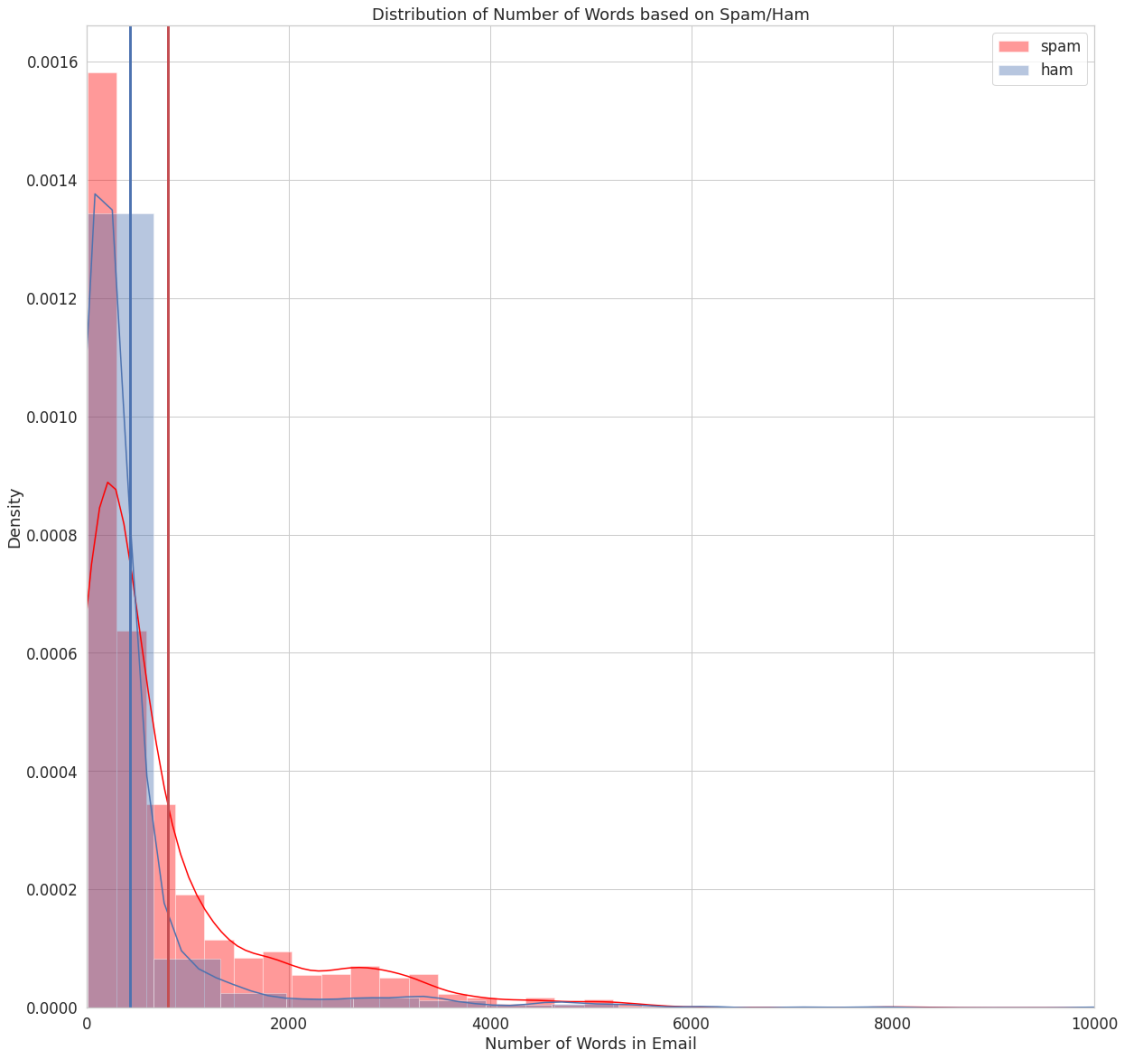
In [14]:
```
plt.figure(figsize=(20,20))
plt.title("Distribution of Number of Characters based on Spam/Ham")
sns.distplot(spam['length_char'], label= 'spam', color = 'red' )
sns.distplot(ham['length_char'], label= 'ham')
plt.xlabel("Number of Characters in Email")
plt.xlim([0, 50000])
plt.axvline(np.mean(spam["length_char"]), color = 'r', linewidth = 3)
plt.axvline(np.mean(ham["length_char"]), color = 'b', linewidth = 3)
plt.legend()
```

Out[14]:
```
<matplotlib.legend.Legend at 0x7ff98c7c87c0>
```




```
In [15]: plt.figure(figsize=(20,20))
         plt.title("Distribution of Number of Words based on Spam/Ham")
         sns.distplot(spam['length_word'], label= 'spam', color = 'red' )
         sns.distplot(ham['length_word'], label= 'ham')
         plt.xlabel("Number of Words in Email")
         plt.xlim([0, 10000])
         plt.axvline(np.mean(spam["length_word"]), color = 'r', linewidth = 3)
         plt.axvline(np.mean(ham["length_word"]), color = 'b', linewidth = 3)
         plt.legend()
```

```
Out[15]: <matplotlib.legend.Legend at 0x7ff98a5d0bb0>
```

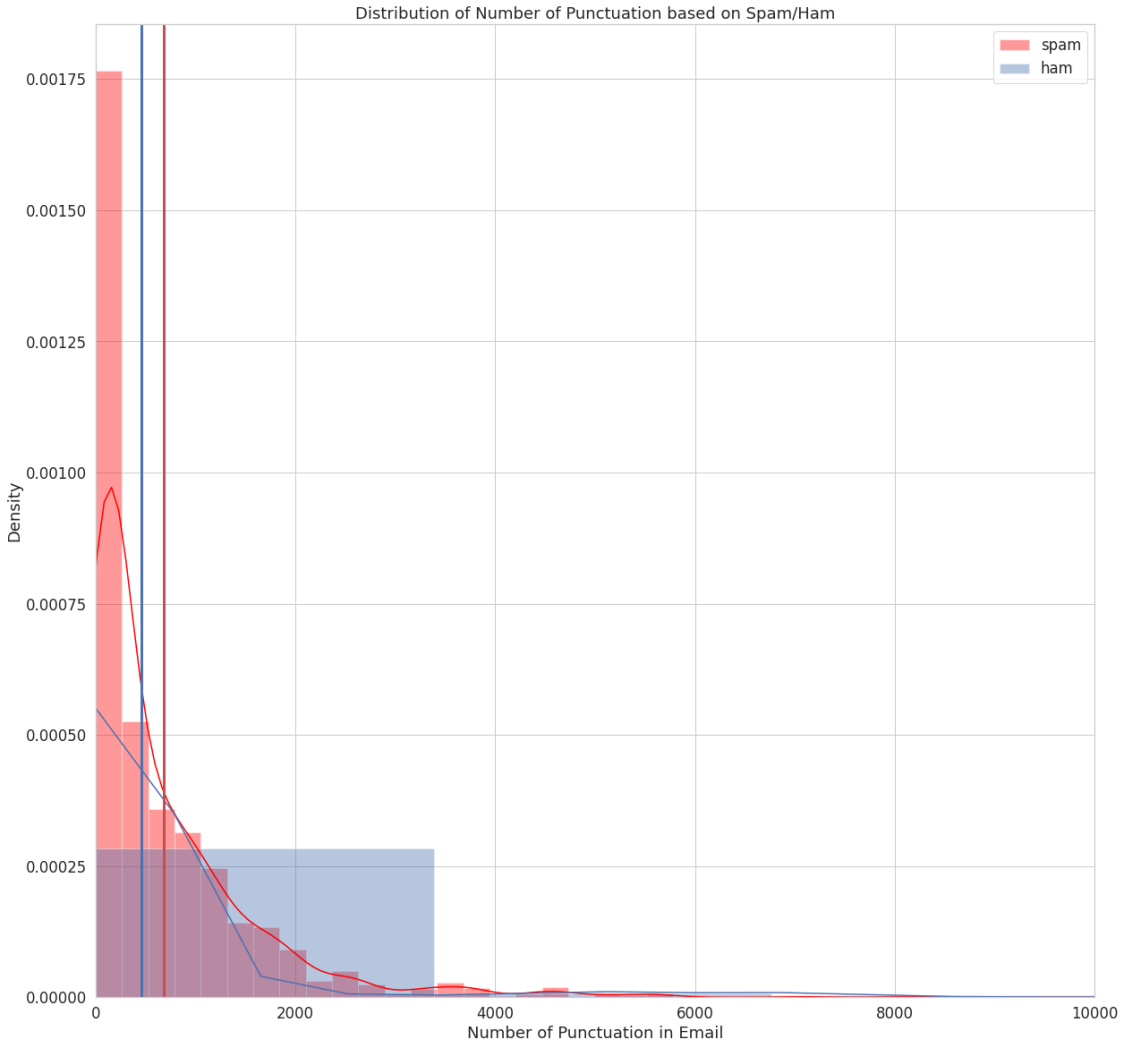


In [16]:
```
import string
#returns number of punctuation characters in email
def punct(s):
    punct = []
    chars = split_char(s)
    for c in chars:
        if c in string.punctuation:
            punct.append(c)
    return len(punct)
train["pn"] = train["email"].apply(lambda x: punct(x))
spam_pn = train[train["spam"] == 1]
ham_pn = train[train["spam"] == 0]
print(max(spam_pn["pn"]))
```

13138

In [17]:
```
plt.figure(figsize=(20,20))
plt.title("Distribution of Number of Punctuation based on Spam/Ham")
sns.distplot(spam_pn['pn'], label= 'spam', color = 'red' )
sns.distplot(ham_pn['pn'], label= 'ham')
plt.xlabel("Number of Punctuation in Email")
plt.xlim([0, 10000])
plt.axvline(np.mean(spam_pn["pn"]), color = 'r', linewidth = 3)
plt.axvline(np.mean(ham_pn["pn"]), color = 'b', linewidth = 3)
plt.legend()
```

Out[17]: <matplotlib.legend.Legend at 0x7ff98a445850>



In [18]:
```
print("spam mean char:" , np.mean(spam["length_char"]), "ham mean char:" , np.mean(ham["length
print("spam mean word:", np.mean(spam["length_word"]), "ham mean word:", np.mean(ham["length_w
print("spam mean punct number:", np.mean(spam_pn["pn"]), "ham mean punct number:", np.mean(ham
```

```
spam mean char: 5528.656412930136 ham mean char: 2986.5805183199286
spam mean word: 798.9431699687174 ham mean word: 428.9304736371761
spam mean punct number: 682.6897810218978 ham mean punct number: 455.3971403038427
```

Question 2b Write your commentary in the cell below.

As seen in the visualizations above, spam emails are generally longer and contain more punctuation than ham emails. In all of the graphs, the distribution is right skewed but the mean values for spam was greater than the mean for ham emails.

0.0.2 Question 3: 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 20 to see how to plot an ROC curve.

Hint: You'll want to use the `.predict_proba` method for your classifier instead of `.predict` so you get probabilities instead of binary predictions.

In [21]:
```
import plotly.express as px
```

In [22]:
```
print(train[train["spam"] == 1]["email"])
```

```
5       --===_secatt_000_1fuklemuttfusq\n content-type…
13      ####################################\n \n   fre…
15      <table width="600" border="20" align="center" …
18      <html>\n <head>\n </head>\n <body>\n <center>\…
19      <html>\n <head>\n </head>\n <center>\n <h1>\n …
                               …
7502    <html>\n <body bgcolor=3d"#003300">\n <p align…
7503    below is the result of your feedback form.  it…
7505    <html>\n <table width="350" border="0" cellspa…
7507    \n mr. ayanda maredi\n department of minerals …
7509    \n dear consumers, increase your business sale…
Name: email, Length: 1918, dtype: object
```

In [23]:
```
new_words = ['drug', 'bank', 'prescription', 'memo', 'private',
                     'money', 'guaranteed', 'deposit', 'html', 'click', 'url', 'buy','sale','body',
                     'subscribe', 'discount', 'loan', "free", "spam", "income", "offer", "please", "!"
                     "#", 'order', "consumers", "business", "sale", "<head>", '<html>\n']
        print(new_words)
```

```
['drug', 'bank', 'prescription', 'memo', 'private', 'money', 'guaranteed', 'deposit', 'html', 'click',
```

In [24]:
```
x_train2 = words_in_texts(new_words, train["email"]) #0 if word not in text, 1 if word in text
y_train2 = np.array(train["spam"])
x_train2[:5], y_train2[:5]
```

Out[24]:
```
(array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
         1, 1, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0,
         0, 1, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
         1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
         1, 0, 0, 0, 0, 1, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0,
         1, 1, 1, 0, 1, 0, 1, 1]]),
 array([0, 0, 0, 0, 0]))
```

In [25]:
```
model = LogisticRegression(random_state=0).fit(x_train2, y_train2)
training_accuracy = np.mean(y_train2 == model.predict(x_train2))
print("Training Accuracy: ", training_accuracy)
```

```
Training Accuracy:  0.8912551577266072
```

In [26]:
```
from sklearn.metrics import roc_curve
#plot TPR on x vs FPR on y
model_probabilities = model.predict_proba(x_train2)[:,1]
fpr, tpr, threshold = roc_curve(y_train2, model_probabilities)
fig = px.line(x=fpr, y = tpr, hover_name=threshold, title = "ROC Curve")
fig.update_xaxes(title="False Positive Rate")
fig.update_yaxes(title="True Positive Rate")
fig
```