0.1 Question 1

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

I found better features by using AB testing and testing to see if they were reduntant. I used features that I were useful and would highlight the extra junk that is included in spam emails.

I tried using the length of emails to separate spam from ham emails, and counts of things. For example, counts of excalamtion points, punctuation, capital letters, and other things. Those features weren't the best at creating this spam-ham model.

What I found works really well are ratios. I wrote some code that cleaned emails, and I used the ratio of counts found in the clean emails versus the uncleaned emails. This separated a lot of the noise, and really allowed the comparison of the content in spam emails to distinguish themselves from ham emails. This was surprising to me.

0.2 Question 2a

Generate your visualization in the cell below.

In [12]:
```
# Define your processing function, processed data, and model here.
# You may find it helpful to look through the rest of the questions first!

train = pd.read_csv('train.csv')

# file_path = 'vader_lexicon.txt'

# df = pd.read_csv(file_path, sep='\t', header=None)
# df.columns = ['symbol', 'score', 'score_sd', 'sample_scores']
# df.head()

# sentiment_dict = df.set_index('symbol')['score'].to_dict()
```

In [13]:
```
## implemeting feature selection:

def get_X(train):
    # 1. Email Length:
    train['length'] = train['email'].apply(lambda x: len(x))

    # 2. Word Selection:
    words = ['Free', 'Ad', 'Offer', 'Credit', 'Save', 'Click here',
       'Guaranteed', 'Money', 'Rates', 'Special', 'Refinance', 'Debt',
       'Cash', 'Quote', 'Easy', 'Loans', 'Secret', 'Limited', 'Removal',
       'Promotion']
    indicator_data = []

    for word in words:
        indicator_array = words_in_texts([word], train['email'])
        indicator_data.append(indicator_array[:, 0])  # Convert to 1D and append

    # Create a DataFrame from the indicator data
    indicator_df = pd.DataFrame(indicator_data).transpose()
    indicator_df.columns = words

    # Concatenate the new DataFrame with the original one
    train = pd.concat([train, indicator_df], axis=1)

    # 3. HTML Analysis:
    html_stuff = train.email.str.extractall('<([^>]+)>').reset_index()
    html_gb = html_stuff.groupby('level_0').agg('count')

    train['hmtl_counts'] = html_stuff.groupby('level_0').agg('count').iloc[:, 0]
```

```python
train.fillna(0)

# 4. Sentiment Analysis:
def strip_to_text(html_content):
    # List of punctuation characters to remove
    punctuation = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

    # Using regular expressions to remove HTML tags and URLs
    text = re.sub('<[^<]+?>', '', html_content)   # Remove HTML tags
    text = re.sub(r'http\S+', '', text)            # Remove URLs

    # Removing punctuation
    text = text.translate(str.maketrans('', '', punctuation))

    # Removing tabs and newline characters
    text = text.replace('\t', '').replace('\n', '')

    return text

def apply_sentiment_score(text):
    sentences = text.split('. ')
    words_in_sentences = [sent.split() for sent in sentences]
    scores = []
    for sentence in words_in_sentences:
        sentence_scores = []
        for wrd in sentence:
            try:
                sentence_scores.append(sentiment_dict[wrd])
            except:
                sentence_scores.append(0)
        if len(sentence_scores) == 0:
            scores.append(0)
        else:
            scores.append(np.mean(sentence_scores))
    if len(scores) == 0:
        return 0
    else:
        return np.mean(scores)

train['email_clean'] = train['email'].apply(strip_to_text)
# train['mean_sentiment'] = train['email_clean'].apply(apply_sentiment_score)


# 5. Punctuation Counter:
pattern = r'[!"#$%&\'()*+,-./:;<=>?@\[\\\]^_`{|}~]'
train['num_punc'] = train['email'].apply(lambda x: len(re.findall(pattern, x)))

# 6. Junk Char Ratio
def junk_char_ratio(row):
    return len(row.email_clean)/len(row.email)

train['junk_char_ratio'] = train.apply(lambda row: junk_char_ratio(row), axis = 1)

train.fillna(0)
```

```
    # 7. Number of Capital Letters:

    def count_capitals(text):
        count = 0
        for char in text:
            if char.isupper():
                count += 1
        return count

    train['num_caps'] = train['email'].apply(count_capitals)

    # 8. Detecting Replies
    def detect_reply(text):
        if 'wrote:\n \n' in text:
            return 0
        else:
            return 1

    train['is_reply'] = train.email.apply(detect_reply)

    # 9. Numer of Capitalized Letters/Length of Email
    def cap_ratio(text):
        count = 0
        for char in text:
            if char.isupper() == True:
                count += 1
        return count/len(text)

    train['cap_ratio'] = train.email.apply(cap_ratio)

    # Y_train = train['spam']
    # cols = words + ['mean_sentiment', 'num_punc', 'junk_char_ratio', 'num_caps', 'is_reply',
    cols = words + ['num_punc', 'junk_char_ratio', 'is_reply', 'cap_ratio']
    X_train = train[cols].fillna(0)
    return X_train
```

```
In [14]: X_train = get_X(train)
         Y_train = train['spam']
```

```
In [15]: len(X_train.columns)
```

```
Out[15]: 24
```

```
In [16]: model = LogisticRegression(max_iter=1000)
         model.fit(X_train, Y_train)
```

```
Out[16]: LogisticRegression(max_iter=1000)
```

In [17]:
```
y_pred = model.predict(X_train)
```

In [18]:
```
np.mean(y_pred == Y_train)
```

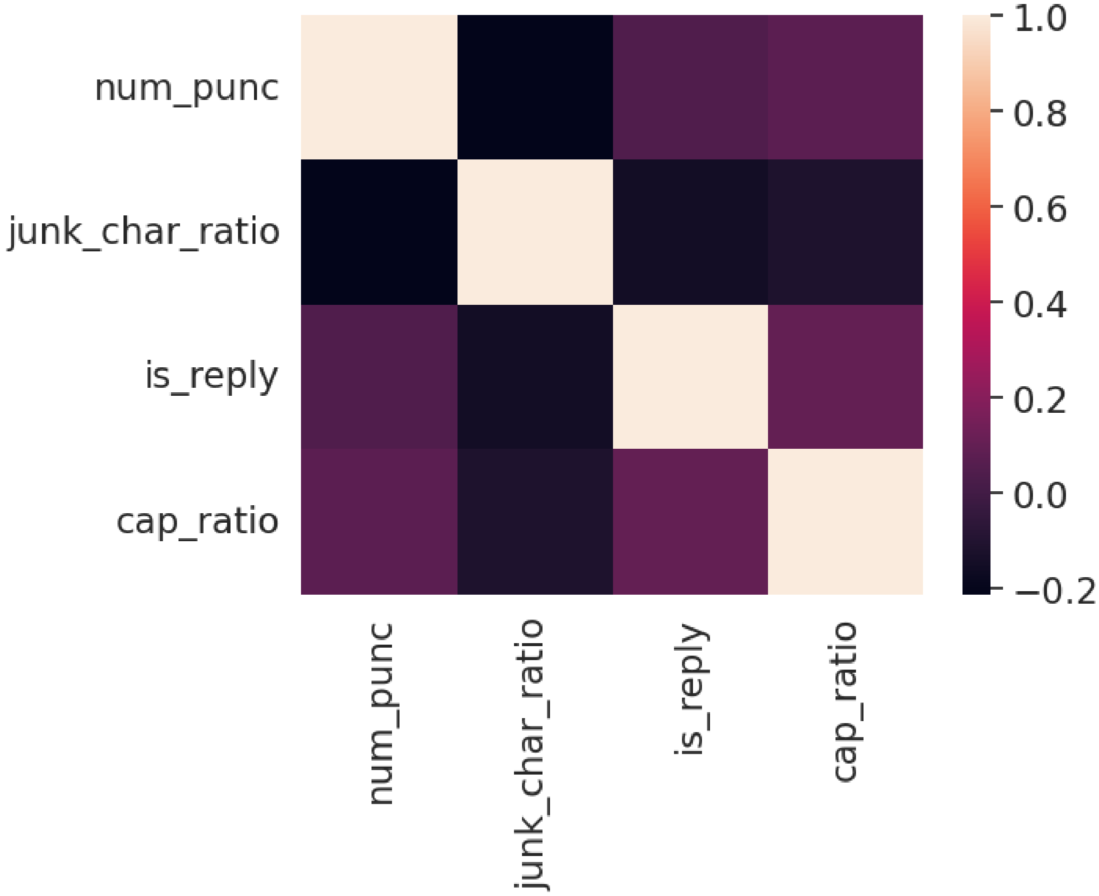
Out[18]:
```
0.860804983229516
```

In [19]:
```
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor

def VIF(df, columns):
    values = sm.add_constant(df[columns]).values
    num_columns = len(columns)+1
    vif = [variance_inflation_factor(values, i) for i in range(num_columns)]
    return pd.Series(vif[1:], index=columns)

VIF(X_train, X_train.columns)
```

Out[19]:
```
Free              1.249652
Ad                1.236748
Offer             1.161467
Credit            1.114013
Save              1.153130
Click here        1.084374
Guaranteed        1.082985
Money             1.090048
Rates             1.286453
Special           1.253901
Refinance         1.607395
Debt              1.249209
Cash              1.103276
Quote             1.140543
Easy              1.165459
Loans             1.588769
Secret            1.158504
Limited           1.112322
Removal           1.095155
Promotion         1.104827
num_punc          1.100907
junk_char_ratio   1.215312
is_reply          1.037932
cap_ratio         1.040852
dtype: float64
```

In [20]:
```
sns.heatmap(X_train[['num_punc', 'junk_char_ratio', 'is_reply', 'cap_ratio']].corr())
```

Out[20]:
```
<Axes: >
```




```
In [21]: train = pd.read_csv('train.csv')
```

```
In [22]: train[train.spam == 0].email.values[165]
```

```
Out[22]: 'On Tue 30 Jul 2002 10:28, David Neary wrote:\n \n > I have 3 or 4 email addresses (which get
```

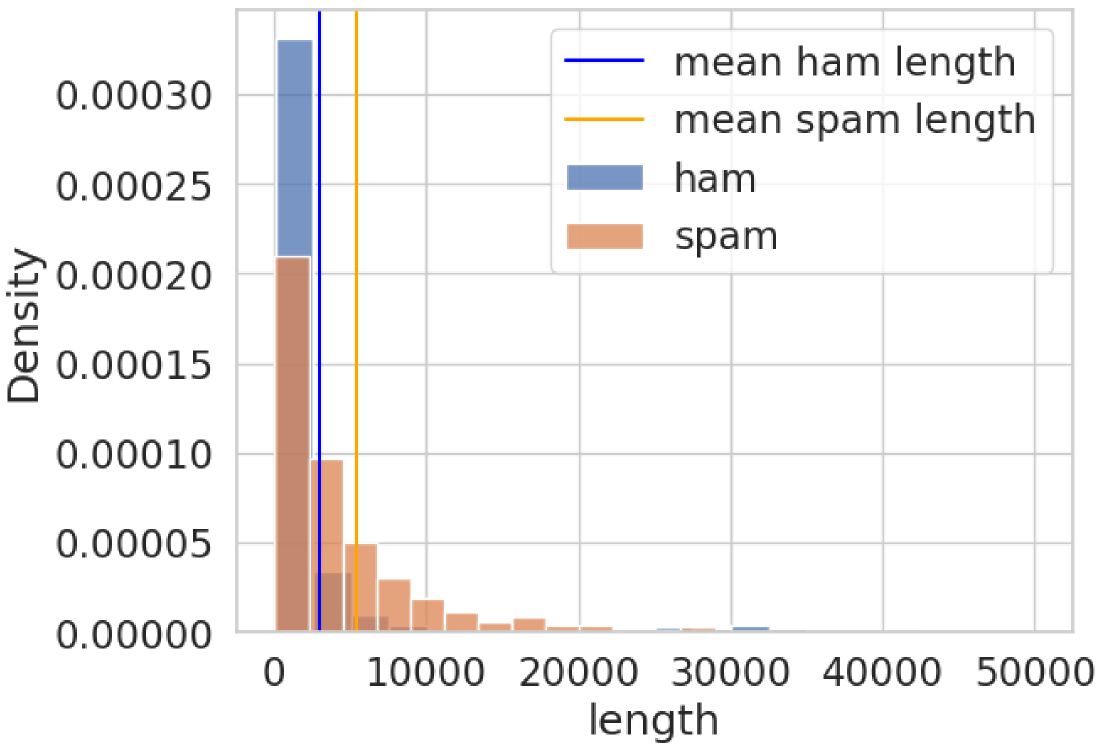
```
In [ ]:
```

0.3 Measuring the length of an email

```
In [23]: train['length'] = train['email'].apply(lambda x: len(x))
```

```
sns.histplot(data = train[(train.length < 50000) & (train.spam == 0)], x = 'length', label = 'l
sns.histplot(data = train[(train.length < 50000) & (train.spam == 1)], x = 'length', label = 's

plt.axvline(np.mean(train[train.spam == 0]['length']), label = 'mean ham length', color = 'blue
plt.axvline(np.mean(train[train.spam == 1]['length']), label = 'mean spam length', color = 'ora

plt.legend()
```

Out[23]: <matplotlib.legend.Legend at 0x7f752f60d610>



0.4 Word Selection

I found a website that had a bunch of spam phrases and words. Source: https://www.activecampaign.com/blog/spam-words

In [24]:
```
# sample_spam_words = pd.read_csv('spam_texts.csv')
# sample_spam_words.columns = ['spam_phrases']
# sample_spam_words
```

```
In [25]: train[train.spam == 1]['email'].values[0]
```

```
Out[25]: '<HTML>\n <HEAD>\n </HEAD>\n <BODY>\n <FONT SIZE=3D"4"><B> A man endowed with a 7-8" hammer is
```

```
In [26]: train[train.spam == 1]['email'].values[1002]
```

```
Out[26]: 'Please visit http://ukprankcalls.com to play a hilarious joke on your mates!\n \n \n'
```

0.5 Selecting Random emails:

This code selects a random email and displays its contents.

```
In [27]: import random

         train[train.spam == 1]['email'].values[random.randint(0, len(train[train.spam == 1]))]
```

```
Out[27]: '<html>\n <head>\n <title>Digital Publishing Tools - Free Software Alert!</title>\n <meta http-
```

```
In [28]: words = ['Free', 'Ad', 'Offer', 'Credit', 'Save', 'Click here',
                 'Guaranteed', 'Money', 'Rates', 'Special', 'Refinance', 'Debt',
                 'Cash', 'Quote', 'Easy', 'Loans', 'Secret', 'Limited', 'Removal',
                 'Promotion', 'Opportunity', 'Income', 'Bonus', 'Save up to',
                 'Warranty', 'Unlimited', 'Membership', 'Deal', 'Billion',
                 'Premium']

         new_train = train.copy()

         indicator_data = []

         for word in words:
             indicator_array = words_in_texts([word], new_train['email'])
             indicator_data.append(indicator_array[:, 0])  # Convert to 1D and append

         # Create a DataFrame from the indicator data
         indicator_df = pd.DataFrame(indicator_data).transpose()
         indicator_df.columns = words

         # Concatenate the new DataFrame with the original one
         new_train = pd.concat([new_train, indicator_df], axis=1)
```

```
In [29]: # display(new_train.iloc[:, range(-1 * len(words), 0)])
         # display(new_train)
```

```
cols = ['spam'] + words

dat = new_train[cols].melt('spam')
dat.value_counts()
# dispaly(dat)
spam_dict = {0: 'ham', 1: 'spam'}
dat['label'] = dat['spam'].map(spam_dict)

pt = pd.pivot_table(dat, index = 'variable', columns = 'label', aggfunc = 'mean').reset_index(
pt.columns = ['variable', 'spam', 'spam', 'ham_perc', 'spam_perc']
pt['differential'] = pt.spam_perc - pt.ham_perc
# display(pt.head())
best_words = pt.sort_values('differential', ascending = False).variable.values[0:20]
best_words
```

Out[29]:
```
array(['Free', 'Ad', 'Offer', 'Credit', 'Save', 'Click here',
       'Guaranteed', 'Money', 'Rates', 'Special', 'Refinance', 'Debt',
       'Cash', 'Quote', 'Easy', 'Loans', 'Secret', 'Limited', 'Removal',
       'Promotion'], dtype=object)
```

In [30]:
```
pt.sort_values('differential', ascending = False)
```

Out[30]:

	variable	spam	spam	ham_perc	spam_perc	differential
9	Free	0	1	0.058956	0.178037	0.119081
0	Ad	0	1	0.125966	0.217290	0.091323
16	Offer	0	1	0.003866	0.074299	0.070433
5	Credit	0	1	0.001128	0.069159	0.068031
24	Save	0	1	0.005799	0.070093	0.064294
4	Click here	0	1	0.013853	0.071495	0.057642
10	Guaranteed	0	1	0.000000	0.055607	0.055607
15	Money	0	1	0.004832	0.050000	0.045168
21	Rates	0	1	0.001933	0.042991	0.041058
27	Special	0	1	0.020296	0.058879	0.038582
22	Refinance	0	1	0.000000	0.035047	0.035047
7	Debt	0	1	0.000322	0.030841	0.030519
3	Cash	0	1	0.001128	0.029439	0.028312
20	Quote	0	1	0.004994	0.033178	0.028184
8	Easy	0	1	0.012081	0.038785	0.026704
13	Loans	0	1	0.000000	0.026636	0.026636
26	Secret	0	1	0.005799	0.031776	0.025977
12	Limited	0	1	0.002094	0.026636	0.024541
23	Removal	0	1	0.000322	0.024766	0.024444
19	Promotion	0	1	0.001933	0.024766	0.022833
17	Opportunity	0	1	0.000966	0.022897	0.021931
11	Income	0	1	0.001933	0.023364	0.021431
2	Bonus	0	1	0.000805	0.021028	0.020223
25	Save up to	0	1	0.000161	0.018224	0.018063
29	Warranty	0	1	0.000000	0.017757	0.017757
28	Unlimited	0	1	0.000161	0.016822	0.016661

```
14   Membership     0     1  0.001289   0.016822      0.015534
6          Deal     0     1  0.006604   0.021963      0.015358
1       Billion     0     1  0.001289   0.016355      0.015066
18      Premium     0     1  0.000483   0.014953      0.014470
```

0.6 HMTL Analysis

In [31]:
```
# train.email[2]
```

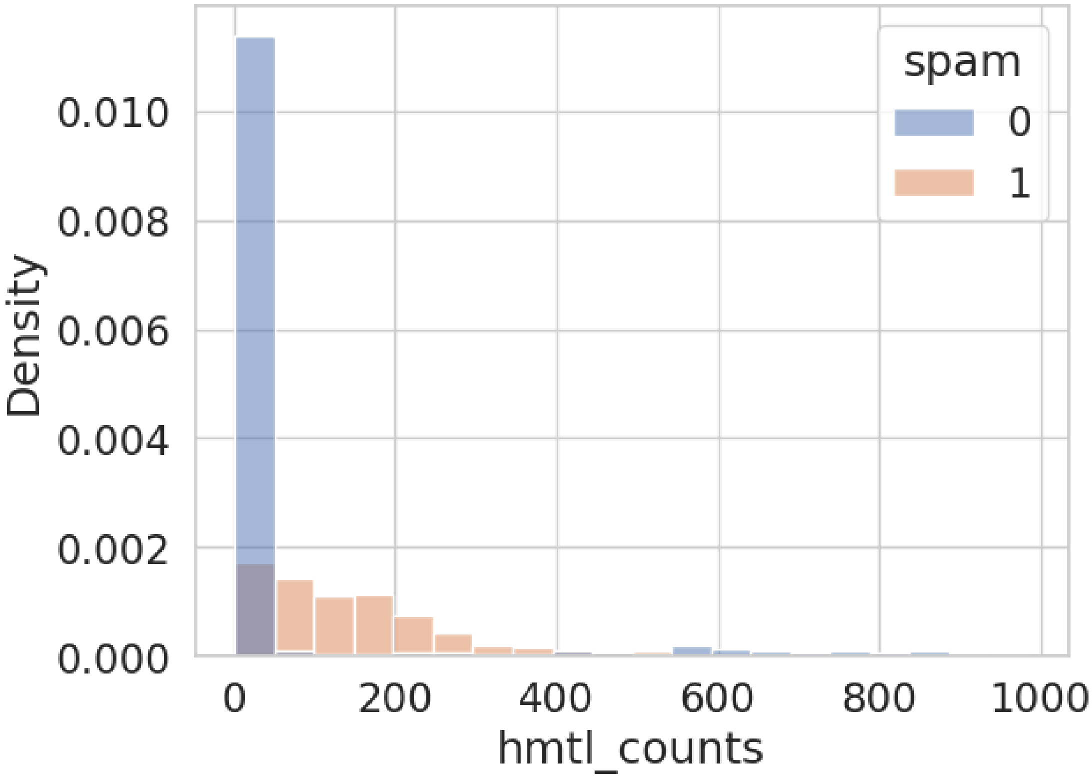
In [32]:
```
new_train.head()
```

Out[32]:
```
   id                                            subject  \
0   0  Subject: A&L Daily to be auctioned in bankrupt…
1   1  Subject: Wired: "Stronger ties between ISPs an…
2   2  Subject: It's just too small                    …
3   3                      Subject: liberal defnitions\n
4   4  Subject: RE: [ILUG] Newbie seeks advice - Suse…

                                               email  spam  length  Free  Ad  \
0  URL: http://boingboing.net/#85534171\n Date: N…     0     359     0   0
1  URL: http://scriptingnews.userland.com/backiss…     0     278     0   0
2  <HTML>\n <HEAD>\n </HEAD>\n <BODY>\n <FONT SIZ…     1     444     0   0
3  Depends on how much over spending vs. how much…     0    1500     0   0
4  hehe sorry but if you hit caps lock twice the …     0    2018     0   1

   Offer  Credit  Save  …  Opportunity  Income  Bonus  Save up to  Warranty  \
0      0       0     0  …            0       0      0           0         0
1      0       0     0  …            0       0      0           0         0
2      0       0     0  …            0       0      0           0         0
3      0       0     0  …            0       0      0           0         0
4      0       0     0  …            0       0      0           0         0

   Unlimited  Membership  Deal  Billion  Premium
0          0           0     0        0        0
1          0           0     0        0        0
2          0           0     0        0        0
3          0           0     0        0        0
4          0           0     0        0        0

[5 rows x 35 columns]
```

In [33]:
```
"""
First, we are detecting the number of HTML tags that exist in each email. It seems that way mo
in spam emails rather than regular emails.
"""
```

```
html_stuff = train.email.str.extractall('<([^>]+)>').reset_index()
html_gb = html_stuff.groupby('level_0').agg('count')

new_train = train.copy()
new_train['hmtl_counts'] = html_stuff.groupby('level_0').agg('count').iloc[:, 0]
new_train.fillna(0)

sns.histplot(data = new_train[new_train.hmtl_counts < 1000], hue = 'spam', x = 'hmtl_counts', 
```

```
Out[33]: <Axes: xlabel='hmtl_counts', ylabel='Density'>
```



Trying to understand how many 'head' and 'body' formatting to see if there's a pattern

```
In [34]: html_stuff['new_lowered'] = html_stuff[0].apply(lambda string: string.lower())
         html_stuff
         html_stuff[(html_stuff.level_0 == 7) & ((html_stuff.new_lowered.str.contains('body')) | (html_s
```

```
Out[34]:     level_0  match                                              0  \
```

```
16         7      0                                   body lang=EN-US
18         7      2  p class=MsoBodyText style='text-align:justify'
30         7     14  p class=MsoBodyText style='text-align:justify'
38         7     22  p class=MsoBodyText style='text-align:justify'
42         7     26  p class=MsoBodyText style='text-align:justify'
44         7     28  p class=MsoBodyText style='text-align:justify'
46         7     30  p class=MsoBodyText style='text-align:justify'
48         7     32  p class=MsoBodyText style='text-align:justify'
52         7     36  p class=MsoBodyText style='text-align:justify'
54         7     38                                p class=MsoBodyText
57         7     41                                              /body

                                         new_lowered
16                                 body lang=en-us
18  p class=msobodytext style='text-align:justify'
30  p class=msobodytext style='text-align:justify'
38  p class=msobodytext style='text-align:justify'
42  p class=msobodytext style='text-align:justify'
44  p class=msobodytext style='text-align:justify'
46  p class=msobodytext style='text-align:justify'
48  p class=msobodytext style='text-align:justify'
52  p class=msobodytext style='text-align:justify'
54                             p class=msobodytext
57                                           /body
```

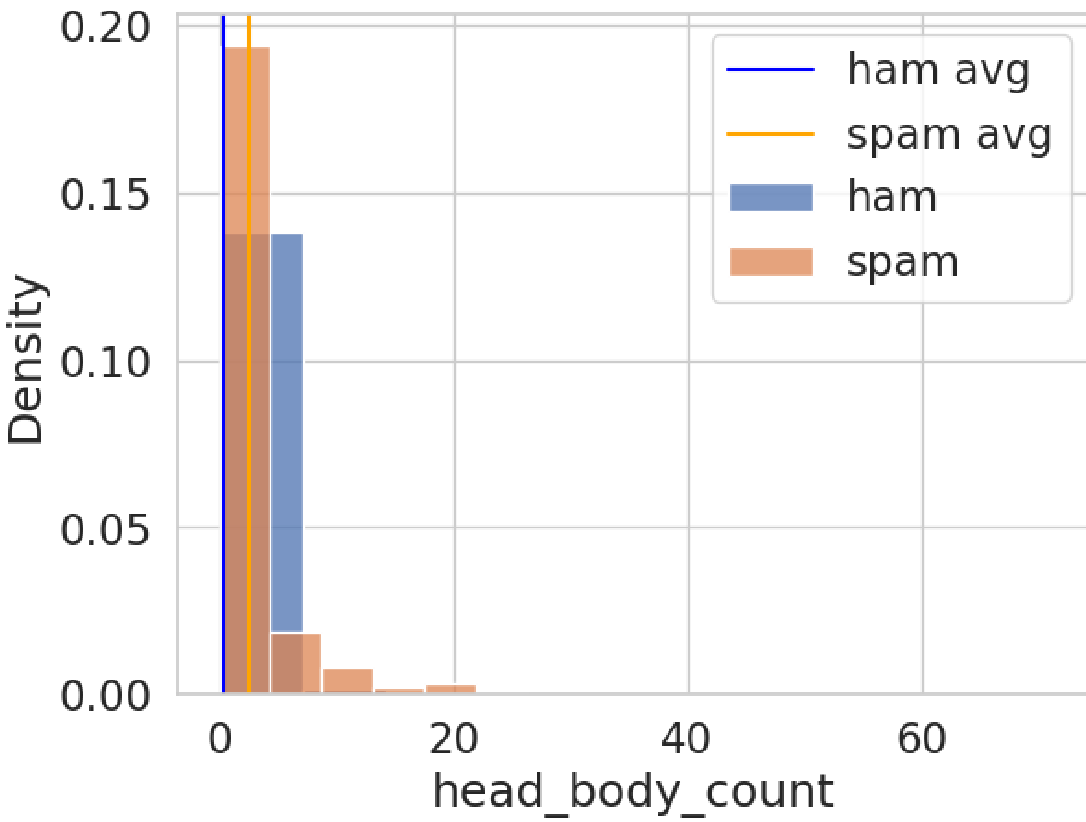
In [35]:
```
def head_body_count(df):
    return len(df[((df.new_lowered.str.contains('body')) | (df.new_lowered.str.contains('head'

df_test = html_stuff[html_stuff.level_0 == 1]
head_body_count(df_test)

hb_counts = html_stuff.sort_values('level_0', ascending = True).groupby('level_0').apply(lambda
# hb_counts
```

In [36]:
```
head_body_count_arr = []

for i in range(len(train)):
    if i in hb_counts.index:
        head_body_count_arr.append(hb_counts[i])
    else:
        head_body_count_arr.append(0)

train['head_body_count'] = head_body_count_arr
```

In [37]:
```
sns.histplot(data = train[train.spam == 0], x = 'head_body_count', stat = 'density', bins = 10
sns.histplot(data = train[train.spam == 1], x = 'head_body_count', stat = 'density', bins = 10

plt.axvline(np.mean(train[train.spam == 0].head_body_count), label = 'ham avg', color = 'blue'
plt.axvline(np.mean(train[train.spam == 1].head_body_count), label = 'spam avg', color = 'oran
```

```
plt.legend()
```

Out[37]: <matplotlib.legend.Legend at 0x7f752ddcd610>



0.7 Attempting Sentiment Analysis

In [38]:
```
# file_path = 'vader_lexicon.txt'

# df = pd.read_csv(file_path, sep='\t', header=None)
# df.columns = ['symbol', 'score', 'score_sd', 'sample_scores']
# df.head()
```

In [39]:
```
# sentiment_dict = df.set_index('symbol')['score'].to_dict()
```

In [40]:
```
# train.email.values[5]
```

In [41]:
```
def strip_to_text(html_content):
    # List of punctuation characters to remove
    punctuation = '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'

    # Using regular expressions to remove HTML tags and URLs
    text = re.sub('<[^<]+?>', '', html_content)   # Remove HTML tags
    text = re.sub(r'http\S+', '', text)           # Remove URLs

    # Removing punctuation
    text = text.translate(str.maketrans('', '', punctuation))

    # Removing tabs and newline characters
    text = text.replace('\t', '').replace('\n', '')

    return text

train['email_clean'] = train['email'].apply(strip_to_text)
train['email_clean'][0]
```

Out[41]:
```
'URL  Date Not supplied  Arts and Letters Daily a wonderful and dense blog has folded up its t
```

In [42]:
```
# def apply_sentiment_score(text):
#     sentences = text.split('. ')
#     words_in_sentences = [sent.split() for sent in sentences]
#     scores = []
#     for sentence in words_in_sentences:
#         sentence_scores = []
#         for wrd in sentence:
#             try:
#                 sentence_scores.append(sentiment_dict[wrd])
#             except:
#                 sentence_scores.append(0)
#         if len(sentence_scores) == 0:
#             scores.append(0)
#         else:
#             scores.append(np.mean(sentence_scores))
#     if len(scores) == 0:
#         return 0
#     else:
#         return np.mean(scores)
```

In [43]:
```
# train['mean_sentiment'] = train['email_clean'].apply(apply_sentiment_score)
```

In [44]:
```
# sns.histplot(data = train[train.spam == 0], x = 'mean_sentiment', stat = 'density', bins = 1
# sns.histplot(data = train[train.spam == 1], x = 'mean_sentiment', stat = 'density', bins = 1
```

```
# plt.axvline(np.mean(train[train.spam == 0].mean_sentiment), label = 'ham avg', color = 'blue
# plt.axvline(np.mean(train[train.spam == 1].mean_sentiment), label = 'spam avg', color = 'ora

# plt.legend()
```

In [45]:
```
train.sample(n = len(train), replace = False).head()['spam']
```

Out[45]:
```
30      0
6101    0
1564    0
5149    0
3250    0
Name: spam, dtype: int64
```

In [46]:
```
def count_capitals(text):
    count = 0
    for char in text:
        if char.isupper():
            count += 1
    return count

train['num_caps'] = train['email'].apply(count_capitals)
train
```

Out[46]:
```
        id                                            subject  \
0        0  Subject: A&L Daily to be auctioned in bankrupt…
1        1  Subject: Wired: "Stronger ties between ISPs an…
2        2  Subject: It's just too small                  …
3        3                       Subject: liberal defnitions\n
4        4  Subject: RE: [ILUG] Newbie seeks advice - Suse…
…      …                                              …
8343  8343               Subject: Re: ALSA (almost) made easy\n
8344  8344                Subject: Re: Goodbye Global Warming\n
8345  8345                                    Subject: hello\n
8346  8346  Subject: Your application is below. Expires Ju…
8347  8347                Subject: Re: [SAtalk] CONFIDENTIAL\n

                                                  email  spam  length  \
0     URL: http://boingboing.net/#85534171\n Date: N…     0     359
1     URL: http://scriptingnews.userland.com/backiss…     0     278
2     <HTML>\n <HEAD>\n </HEAD>\n <BODY>\n <FONT SIZ…     1     444
3     Depends on how much over spending vs. how much…     0    1500
4     hehe sorry but if you hit caps lock twice the …     0    2018
…                                                   …   …     …
8343  Thanks for this, I'm going to give them anothe…     0    2287
8344  Thanks for the link - I'm fascinated by archae…     0    6463
8345  WE NEED HELP.  We are a 14 year old fortune 50…     1     881
```

```
8346  <html>\n \n \n <HEAD> \n <META charset=3DUTF-8…     1    2723
8347  On Wed, 2002-08-21 at 06:42, Craig R.Hughes wr…     0     863

      head_body_count                                        email_clean  \
0                   0  URL  Date Not supplied  Arts and Letters Daily…
1                   0  URL  Date Wed 25 Sep 2002 153310 GMT  Wired1 S…
2                   4       A man endowed with a 78 hammer is simply …
3                   0  Depends on how much over spending vs how much …
4                   0  hehe sorry but if you hit caps lock twice the …
…                 …                                                  …
8343                0  Thanks for this Im going to give them another …
8344                0  Thanks for the link  Im fascinated by archaeol…
8345                0  WE NEED HELP  We are a 14 year old fortune 500…
8346                7                               Your application …
8347                0  On Wed 20020821 at 0642 Craig RHughes wrote  O…

      num_caps
0           22
1           18
2           51
3           26
4           90
…          …
8343        87
8344       167
8345       203
8346       371
8347        37

[8348 rows x 8 columns]
```

In [47]:
```
def mean_diff(df, feature):
    return np.mean(df[df.shuffled_spam == 0][feature]) - np.mean(df[df.shuffled_spam == 1][fea

def shuffle_and_result(df, feature):
    new_label = 'shuffled_spam'
    df[new_label] = df.sample(n = len(df), replace = False)['spam'].values
    return df[['spam', new_label, feature]]

def ab_testing(df, feature):
    h0 = np.mean(df[df.spam == 0][feature]) - np.mean(df[df.spam == 1][feature])
    diffs = []
    for el in range(1000):
        s_df = shuffle_and_result(df, feature)
        diffs.append(mean_diff(s_df, feature))
    # plt.hist(diffs)
    # plt.axvline(h0)
    p_value = np.mean(h0 > diffs)
    print("P Value for " + feature + " " + str(p_value))
    return diffs, h0

# diffs, h0 = ab_testing(train, 'mean_sentiment')
```

In [ ]:
```
diffs, h0 = ab_testing(train, 'length')
```

In [ ]:
```
pattern = r'[!"#$%&\'()*+,-./:;<=>?@\[\\\]^_`{|}~]'

# patten = r'[!?;#$&]'

train['num_punc'] = train['email'].apply(lambda x: len(re.findall(pattern, x)))
train.head()
```

In [ ]:
```
def cap_ratio(text):
    count = 0
    for char in text:
        if char.isupper() == True:
            count += 1
    return count/len(text)

train['cap_ratio'] = train.email.apply(cap_ratio)

diff, h0 = ab_testing(train, 'cap_ratio')
```

In [ ]:
```
bruh = train.sample(n = 1)
email = bruh.email
print(bruh.spam)
[print(el) for el in email]
```

In [ ]:
```
len(train.iloc[1234, :].email_clean)/(len(train.iloc[1234, :].email))
```

In [ ]:
```
def junk_char_ratio(row):
    return len(row.email_clean)/len(row.email)

train['junk_char_ratio'] = train.apply(lambda row: junk_char_ratio(row), axis = 1)
train.head()
```

In [ ]:
```
diffs, h0 = ab_testing(train, 'junk_char_ratio')
```

In [ ]:
```
X_train.columns
```

In [ ]:
```
new_df = X_train.copy()
new_df['spam'] = Y_train
for col in X_train.columns:
    ab_testing(new_df, col)
```

In [ ]:
```
def detect_reply(text):
    if 'wrote:\n \n' in text:
        return 0
    else:
        return 1

train['is_reply'] = train.email.apply(detect_reply)
diffs, h0 = ab_testing(train, 'is_reply')
```

0.8 Question 2b

Write your commentary in the cell below.

List of Features That I Implemented: 1. Word selection 2. Sentiment Analysis 3. Number of puncutation symbols 4. Number of extra characters that are not readable words : Total number of characters 6. If an email is a reply 7. Number of capital letters : Total number of characters

Word Selection I wrote some code that randomly selects an emails and displays the contents. After a lot of re-running the cell, I found that most spam emails are usually trying to sell you something or click on a link. I generated a list of about 30 words that I found that came up and are associated with claiming some sort of prize, cash, or compensation. I sorted these words based on the difference between the percentage of times they show up in the spam versus ham emails.

Sentiment Analysis: I used the vader_leixcon file from HW three to assess the mean sentiment score of each email. My approach was to average the total sentiment of each sentence. I formatted each email so that is removes all of the excess punctuation and URLs, and then I applied the sentiment analysis.

Number of Punctuation Symbols: I summed the number of punctuation symbols in each email.

Junk Character Ratio: Using each cleaned email, I calculated the number of characters in the cleaned email to the uncleaned email. This would highlight all of the extra stuff that's included in formatting a spam email.

Reply Detection: I found that a lot of ham emails have a piece of text that tell who exactly sent an email. I used the syntax to detect ham emails, and implemented this as a True/False column. I could go deeper into this, but for now the model seems fine.

Number of Capital Letters: Spam emails seem to have more capital letters, which might be used to attract the readers attention. I counted the number of captial letters in each email, and compared it to the total number of characters. This might highlight the junk characters/word that might be present in spam emails.

0.9 Feature Performance Analysis:

I wrote some code that performs AB testing on each individual feature. I selected features that have statistically significant p-values to ensure each feature is distinct in their spam and ham performance.

0.10 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. In contrast, a patient can receive another screening for a false positive.

Recall that logistic regression calculates the probability that an example belongs to a particular class. 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 threshold**: 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 Receiver Operating Characteristic (ROC) curve shows this trade-off for each possible cutoff probability. In the cell below, plot an ROC curve for your final classifier (the one you use to make predictions for Gradescope) on the training data. Refer to Lecture 23 to see how to plot an ROC curve.

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

In [ ]:
```
from sklearn import metrics

#define metrics
y_pred_proba = model.predict_proba(X_train)[::,1]
fpr, tpr, _ = metrics.roc_curve(Y_train,  y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr, label = 'model performance')

x = np.arange(0, 1, 0.01)
x_ideal = [0, 0, 1]
y_ideal = [0, 1, 1]

plt.plot(x, x, color = 'red', label = 'random case')
plt.plot(x_ideal, y_ideal, color = 'green', label = 'best case')

plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
```