Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [1]:
```
NAME = "Matthew Brennan"
COLLABORATORS = "Connor McCormick"
```

Homework 4: Spam/Ham Classification

Feature Engineering, Logistic Regression, Cross Validation

Due Date: 11/1/18, 11:59PM

Course Policies

Here are some important course policies. These are also located at http://www.ds100.org/fa18/ (http://www.ds100.org/fa18/).

Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the homework, we ask that you **write your solutions individually**. If you do discuss the assignments with others please **include their names** at the top of your solution.

This Assignment

In this homework, you will use what you've learned in class to create a classifier that can distinguish spam (junk or commercial or bulk) emails from ham (non-spam) emails. In addition to providing some skeleton code to fill in, we will evaluate your work based on your model's accuracy and your written responses in this notebook.

After this homework, you should feel comfortable with the following:

- Feature engineering with text data
- Using sklearn libraries to process data and fit models
- Validating the performance of your model and minimize overfitting
- Generating and analyzing precision recall curves

Warning

We've tried our best to filter the data for anything blatantly offensive as best as we can, but unfortunately there may still be some examples you may find in poor taste. If you encounter these examples and believe it is inappropriate for students, please let a TA know and we will try to remove it for future semesters. Thanks for your understanding!

Regarding Submissions - IMPORTANT, PLEASE READ

For this assignment and future assignments (homework and projects) you will also submit your free response and plotting questions to Gradescope. To do this, you can download as PDF; there are two ways to do this:

1. File > Print Preview ----> Save website as PDF
2. Control/Command + P ----> Save website as PDF

If you are having trouble with text being cut off in the generated PDF:

- For cells containing text surrounded in triple quotes (e.g. """ Hello world """), you can press enter in the middle of the string to push the text to a new line so that all the text stays within the box.

You are responsible for submitting and tagging your answers in Gradescope. For each free response and plotting question, please include:

1. Relevant code used to generate the plot or inform your insights
2. The written free response or plot

Part I - Initial Analysis

In [2]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set(style = "whitegrid",
        color_codes = True,
        font_scale = 1.5)
```

Loading in the Data

The dataset consists of email messages and their labels (0 for ham, 1 for spam).

Your labeled dataset contains 8348 labeled examples, and the test set contains 1000 unlabeled examples.

Run the following cells to load in the data into DataFrames.

The `train` DataFrame contains labeled data that you will use to train your model. It contains four columns:

1. `id` : An identifier for the training example.
2. `subject` : The subject of the email
3. `email` : The text of the email.
4. `spam` : 1 if the email was spam, 0 if the email was ham (not spam).

The `test` DataFrame contains another set of 1000 unlabeled examples. You will predict labels for these examples and submit your predictions to Kaggle for evaluation.

In [3]:
```
from utils import fetch_and_cache_gdrive
fetch_and_cache_gdrive('1SCASpLZFKCp2zek-toR3xeKX3DZnBSyp', 'train.csv')
fetch_and_cache_gdrive('1ZDFo9OTF96B5GP2Nzn8P8-AL7CTQXmC0', 'test.csv')

original_training_data = pd.read_csv('data/train.csv')
test = pd.read_csv('data/test.csv')

# Convert the emails to lower case as a first step to processing the text
original_training_data['email'] = original_training_data['email'].str.lower()
test['email'] = test['email'].str.lower()

original_training_data.head()
```

```
Using version already downloaded: Sat Nov  3 21:20:32 2018
MD5 hash of file: 0380c4cf72746622947b9ca5db9b8be8
Using version already downloaded: Sat Nov  3 21:20:31 2018
MD5 hash of file: a2e7abd8c7d9abf6e6fafc1d1f9ee6bf
```

Out[3]:

	id	subject	email	spam
0	0	Subject: A&L Daily to be auctioned in bankrupt...	url: http://boingboing.net/#85534171\n date: n...	0
1	1	Subject: Wired: "Stronger ties between ISPs an...	url: http://scriptingnews.userland.com/backiss...	0
2	2	Subject: It's just too small ...	<html>\n <head>\n </head>\n <body>\n <font siz...	1
3	3	Subject: liberal defnitions\n	depends on how much over spending vs. how much...	0
4	4	Subject: RE: [ILUG] Newbie seeks advice - Suse...	hehe sorry but if you hit caps lock twice the ...	0

Question 1a

First let's check if our data contains any nan values. Fill in the cell below to print whether any of the columns contain nan values. If there are nan values, replace them with the appropriate filler values. In other words, a nan value in the subject column should be replaced with an empty string.

Note that while there are no nan values in the spam column, we should be careful when replacing nan values when they are the labels. Doing so without consideration may introduce significant bias into our model when fitting.

In [4]:
```
pi = pd.isna(original_training_data[['subject']])
pi[pi['subject'] == True]
```

Out[4]:

	subject
271	True
1383	True
3605	True
3658	True
3734	True
7221	True

In [5]:
```
# YOUR CODE HERE
original_training_data.isnull().values.any()
original_training_data.fillna("", inplace = True)
#raise NotImplementedError()
```

Question 1b

In the cell below, print the text of the first ham and the first spam email in the original training set. Then, discuss one thing you notice that is different between the two that might relate to the identification of spam.

In [6]:
```
# Print the text of the first ham and the first spam emails. Then, fill in your response in the q01 var
first_ham = print(original_training_data[original_training_data['spam'] == 0]['email'].iloc[0])
first_spam = print(original_training_data[original_training_data['spam'] == 1]['email'].iloc[0])

# YOUR CODE HERE
#raise NotImplementedError()
```

```
url: http://boingboing.net/#85534171 (http://boingboing.net/#85534171)
 date: not supplied

 arts and letters daily, a wonderful and dense blog, has folded up its tent due
 to the bankruptcy of its parent company. a&l daily will be auctioned off by the
 receivers. link[1] discuss[2] (_thanks, misha!_)

 [1] http://www.aldaily.com/ (http://www.aldaily.com/)
 [2] http://www.quicktopic.com/boing/h/zlfterjnd6jf (http://www.quicktopic.com/boing/h/zlfterjnd6jf)



<html>
 <head>
 </head>
 <body>
 <font size=3d"4"><b> a man endowed with a 7-8" hammer is simply<br>
  better equipped than a man with a 5-6"hammer. <br>
 <br>would you rather have<br>more than enough to get the job done or fall =
 short. it's totally up<br>to you. our methods are guaranteed to increase y=
 our size by 1-3"<br> <a href=3d"http://209.163.187.47/cgi-bin/index.php?10=
 004">come in here and see how</a>
 </body>
 </html>
```

In [7]:
```
# This is a cell with just a comment but don't delete me if you want to get credit.
```

The first spam cell provides a perverted message in a very odd format whereas the first ham message contains a coherent message written in a normal format. For example, the spam message contains an html format with <> around many words suggesting that the message likely was computer automated rather than having a personalized touch. In contrast, the first ham message contains the name of an individual named "Misha" thus adding to the personalized element.

Training Validation Split

The training data we downloaded is all the data we have available for both training models and **validating** the models that we train. We therefore need to split the training data into separate training and validation datsets. You will need this **validation data** to validate your model once you are finished training. Note that we set the seed (random_state) to 42. This will produce a pseudo-random sequence of random numbers. Do not modify this in the following questions, as our assert statements depend on this random seed.

```
In [8]: from sklearn.model_selection import train_test_split

        [train, val] = train_test_split(original_training_data, test_size=0.1, random_state=42)
```

Basic Feature Engineering

We would like to take the text of an email and predict whether the text is ham or spam. This is a classification problem, so we can use logistic regression to make a classifier. Recall that to train an logistic regression model we need a numeric feature matrix Φ (pronounced phi as in wifi) and corresponding binary labels Y. Unfortunately, our data are text, not numbers. To address this, we can create numeric features derived from the email text and use those features for logistic regression.

Each row of Φ is derived from one email example. Each column of Φ is one feature. We'll guide you through creating a simple feature, and you'll create more interesting ones when you are trying to increase your accuracy.

Question 2

Create a function called `words_in_texts` that takes in a list of `words` and a pandas Series of email `texts` . It should output a 2-dimensional NumPy array containing one row for each email text. The row should contain either a 0 or a 1 for each word in the list: 0 if the word doesn't appear in the text and 1 if the word does. For example:

```
>>> words_in_texts(['hello', 'bye', 'world'],
                   pd.Series(['hello', 'hello world hello']))

array([[1, 0, 0],
       [1, 0, 1]])
```

In [9]:
```
def words_in_texts(words, texts):
    '''
    Args:
        words (list-like): words to find
        texts (Series): strings to search in

    Returns:
        NumPy array of 0s and 1s with shape (n, p) where n is the
        number of texts and p is the number of words.
    '''
    indicator_array = np.array([[w in t for w in words] for t in texts]).astype(int)
    # YOUR CODE HERE
    #raise NotImplementedError()
    return indicator_array
```

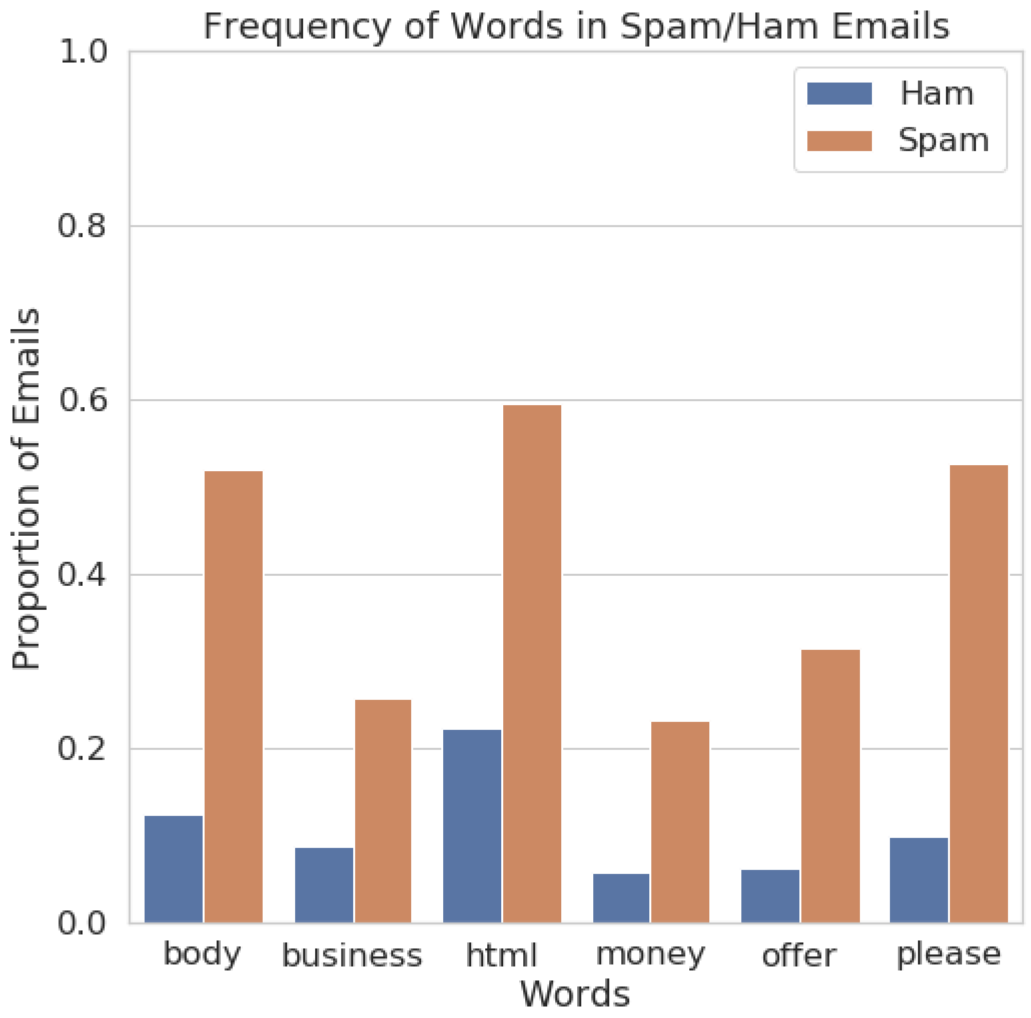
In [10]:
```
# If this doesn't error, your function outputs the correct output for this example
assert np.allclose(words_in_texts(['hello', 'bye', 'world'],
                                  pd.Series(['hello', 'hello world hello'])),
                   np.array([[1, 0, 0],
                             [1, 0, 1]]))

assert np.allclose(words_in_texts(['a', 'b', 'c', 'd', 'e', 'f', 'g'],
                                  pd.Series(['a b c d e f g', 'a', 'b', 'c', 'd e f g', 'h', 'a h'])),
                   np.array([[1,1,1,1,1,1,1],
                             [1,0,0,0,0,0,0],
                             [0,1,0,0,0,0,0],
                             [0,0,1,0,0,0,0],
                             [0,0,0,1,1,1,1],
                             [0,0,0,0,0,0,0],
                             [1,0,0,0,0,0,0]]))
```

Basic EDA

Now we need to identify some features that allow us to tell spam and ham emails apart. One idea is to compare the distribution of a single feature in spam emails to the distribution of the same feature in ham emails. If the feature is itself a binary indicator, such as whether a certain word occurs in the text, this amounts to comparing the proportion of spam emails with the word to the proportion of ham emails with the word.

Question 3a

Create a bar chart comparing the proportion of spam and ham emails containing certain words. It should look like the following plot (which was created using `sns.barplot`), but you should choose your own words as candidate features. Make sure to use the training set (after splitting).



Hint:

- You can use DataFrame's `.melt` method to "unpivot" a DataFrame. See the following code cell for example

In [11]:
```
from IPython.display import display, Markdown
df = pd.DataFrame({
    'word_1': [1, 0, 1, 0],
    'word_2': [0, 1, 0, 1],
    'type': ['spam', 'ham', 'ham', 'ham']
})
display(Markdown("> Our Original DataFrame has some words column and a type column. You can think of eac
display(df)
display(Markdown("> `melt` will turn columns into variale, notice how `word_1` and `word_2` become `var
display(df.melt("type"))
```

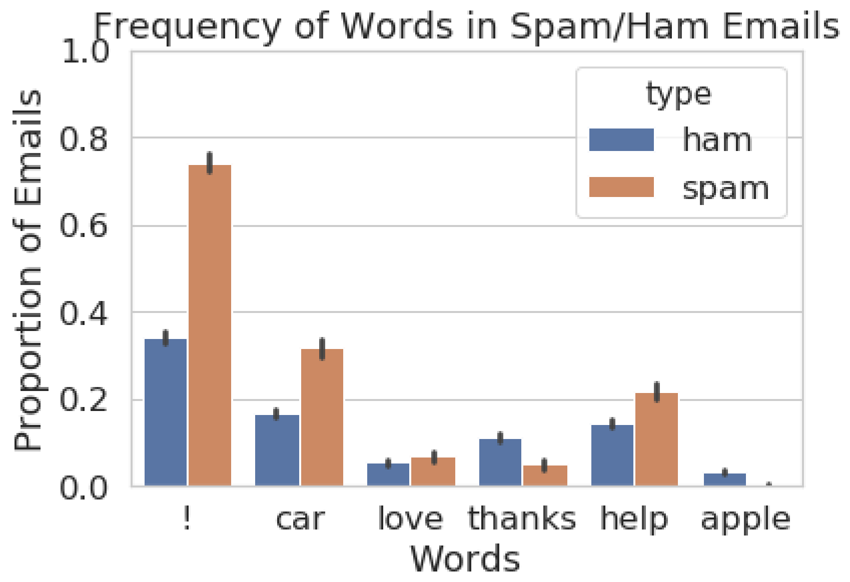
```
<IPython.core.display.Markdown object>
```

	word_1	word_2	type
0	1	0	spam
1	0	1	ham
2	1	0	ham
3	0	1	ham

```
<IPython.core.display.Markdown object>
```

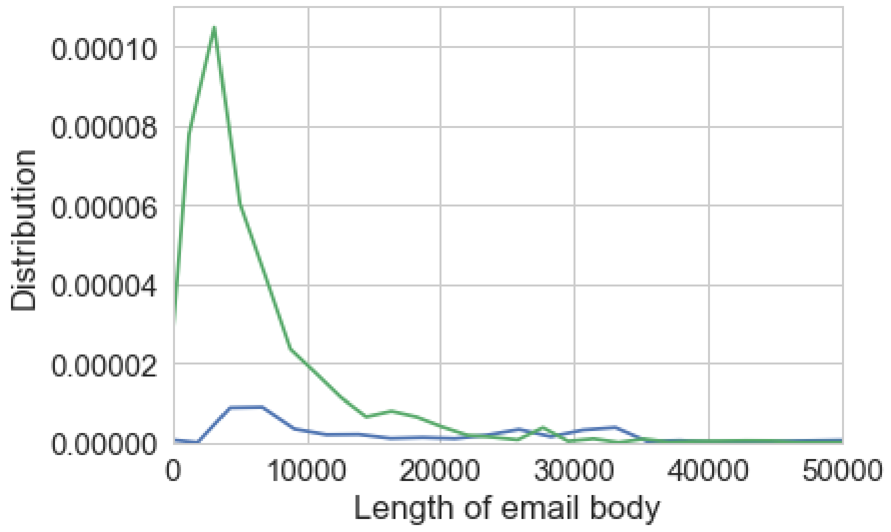
	type	variable	value
0	spam	word_1	1
1	ham	word_1	0
2	ham	word_1	1
3	ham	word_1	0
4	spam	word_2	0
5	ham	word_2	1
6	ham	word_2	0
7	ham	word_2	1

In [13]:
```
train=train.reset_index(drop = True)# We must do this in order to preserve the ordering of emails to lal
arr_of_vals = list(words_in_texts(["!","car", "love",
                    "thanks", "help", "apple"], train['email']))
spam_frame = pd.DataFrame({
    '!': [lst[0] for lst in arr_of_vals],
    'car': [lst[1] for lst in arr_of_vals],
    'love': [lst[2] for lst in arr_of_vals],
    'thanks': [lst[3] for lst in arr_of_vals],
    'help': [lst[4] for lst in arr_of_vals],
    'apple': [lst[5] for lst in arr_of_vals],
    'type': ['ham' if num == 0 else 'spam'for num in train['spam']]
})
melted_spam = spam_frame.melt(id_vars = 'type')
ax = sns.barplot(x = 'variable', y = 'value', hue = 'type', data = melted_spam)
ax.set_ylim(0.0, 1.0)
plt.ylabel('Proportion of Emails')
plt.xlabel('Words')
plt.title('Frequency of Words in Spam/Ham Emails');

#melted_spam
# YOUR CODE HERE
#raise NotImplementedError()
```

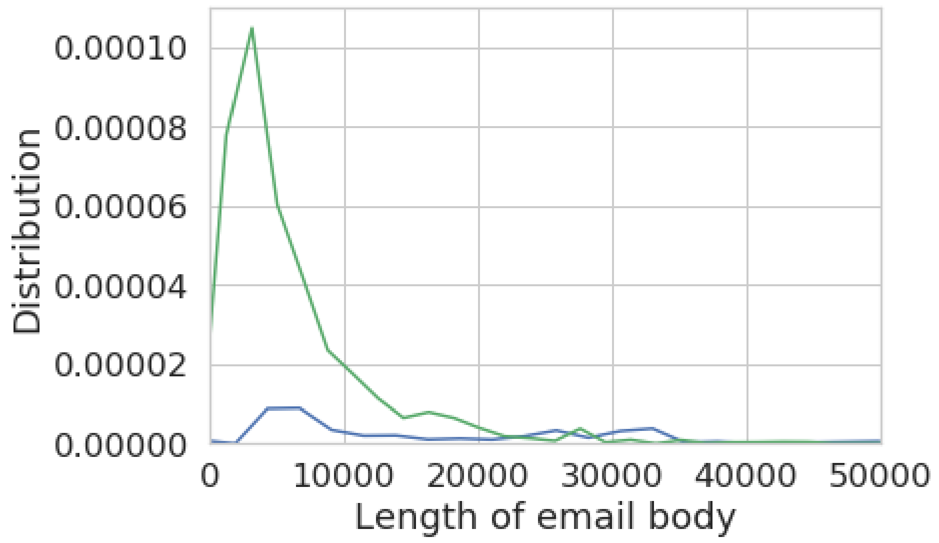


Question 3b

When the feature is binary, it makes sense (as in the previous question) to compare the proportion of 1s in the two classes of email. Otherwise, if the feature can take on many values, it makes sense to compare the distribution under spam to the distribution under ham. Create a class conditional density plot like the one below (which was created using `sns.distplot`), comparing the distribution of a feature among all spam emails to the distribution of the same feature among all ham emails. You should use your training set (after splitting). **You may use the length of the email body or create your own feature.** If using length of the email body, please set the xlim to 50000.



In [13]:
```
# YOUR CODE HERE
len_values_0 = [len(text) for text in train[train['spam'] == 0]['email']]
len_values_1 = [len(text) for text in train[train['spam'] == 1]['email']]
sns.distplot(len_values_0, hist = False, color = 'b')
sns.distplot(len_values_1, hist = False, color = 'g')
plt.xlim(0,50000)
plt.xlabel("Length of email body")
plt.ylabel("Distribution");
#raise NotImplementedError()
```



Basic Classification

Notice that the output of `words_in_texts(words, train['email'])` is a numeric matrix containing features for each email. This means we can use it directly to train a classifier!

Question 4

We've given you 5 words that might be useful as features to distinguish spam/ham emails. Use these words as well as the `train` DataFrame to create two NumPy arrays: `Phi_train` and `Y_train`.

`Phi_train` should be a matrix of 0s and 1s created by using your `words_in_texts` function on all the emails in the training set.

`Y_train` should be a vector of the correct labels for each email in the training set.

In [14]:
```
some_words = ['drug', 'bank', 'prescription', 'memo', 'private']

Phi_train = words_in_texts(some_words, train['email'])
Y_train = np.array(train['spam'])

# YOUR CODE HERE
#raise NotImplementedError()
Phi_train[:5], Y_train[:5]
```

Out[14]:
```
(array([[0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0]]), array([0, 0, 0, 0, 0]))
```

In [15]:
```
assert np.all(np.unique(Phi_train) == np.array([0, 1]))
assert np.all(np.unique(Y_train) == np.array([0, 1]))
assert Phi_train.shape[0] == Y_train.shape[0]
assert Phi_train.shape[1] == len(some_words)
```

Question 5

Now we have matrices we can give to scikit-learn! Using the `LogisticRegression` (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) classifier, train a logistic regression model using `Phi_train` and `Y_train`. Then, output the accuracy of the model (on the training data) in the cell below. You should get an accuracy of around 0.75.

In [16]:
```
from sklearn import linear_model
model = linear_model.LogisticRegression()
model.fit(Phi_train, Y_train)
y_pred = model.predict(Phi_train)
training_accuracy = sum(y_pred == Y_train)/len(Y_train)
training_accuracy


# YOUR CODE HERE
#raise NotImplementedError()
```

Out[16]:
```
0.7576201251164648
```

In [17]:
```
assert training_accuracy > 0.72
```

Question 6

That doesn't seem too shabby! But the classifier you made above isn't as good as this might lead us to believe. First, we are validating on the training set, which may lead to a misleading accuracy measure, especially if we used the training set to identify discriminative features. In future parts of this analysis, it will be safer to hold out some of our data for model validation and comparison.

Presumably, our classifier will be used for **filtering**, i.e. preventing messages labeled `spam` from reaching someone's inbox. Since we are trying There are two kinds of errors we can make:

- False positive (FP): a ham email gets flagged as spam and filtered out of the inbox.
- False negative (FN): a spam email gets mislabeled as ham and ends up in the inbox.

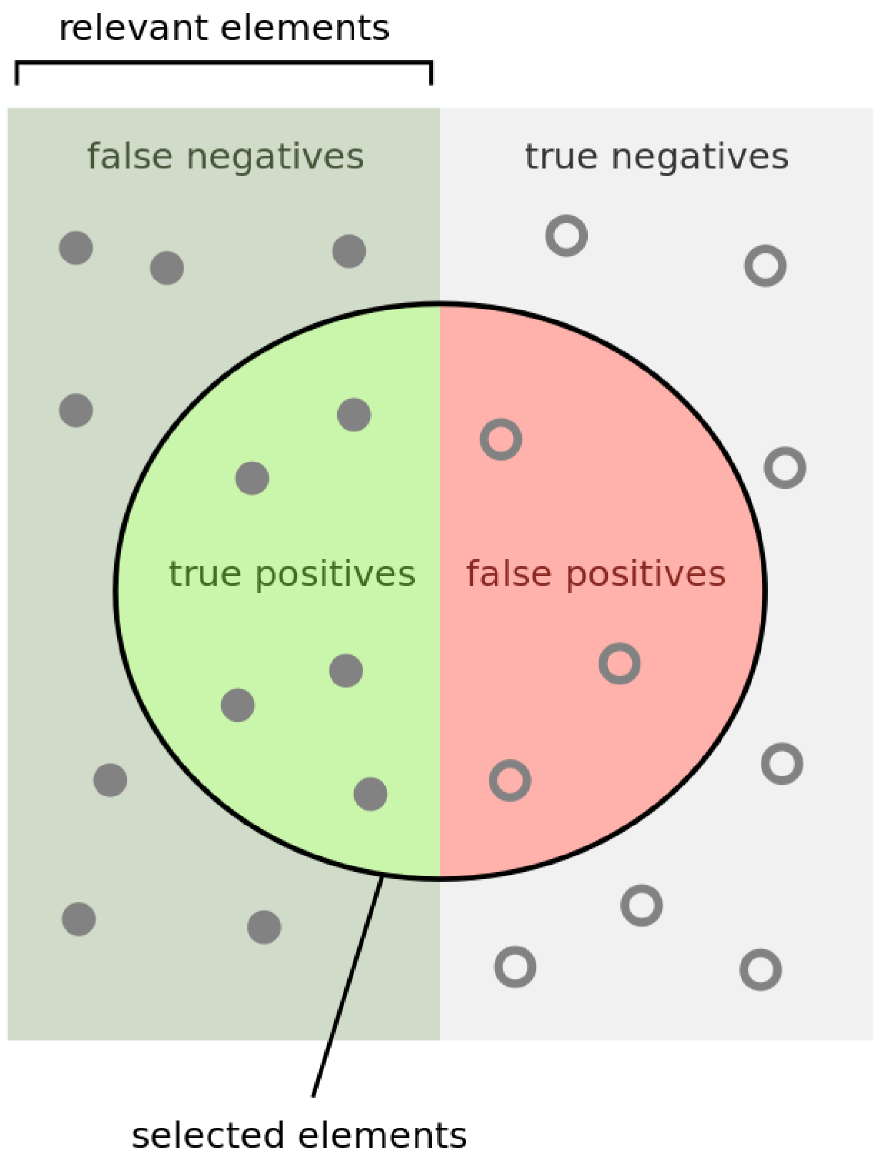
These definitions depend both on the true labels and the predicted labels. False positives and false negatives may be of differing importance, leading us to consider more ways of evaluating a classifier, in addition to overall accuracy:

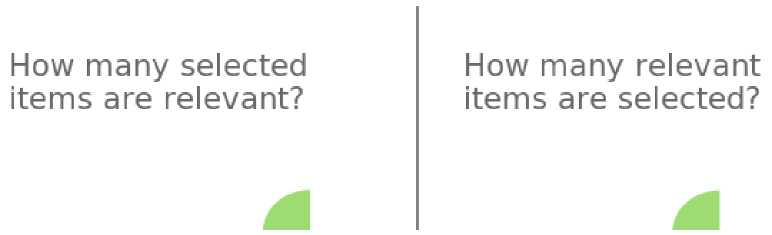
Precision measures the proportion $\frac{\text{TP}}{\text{TP+FP}}$ of emails flagged as spam that are actually spam.

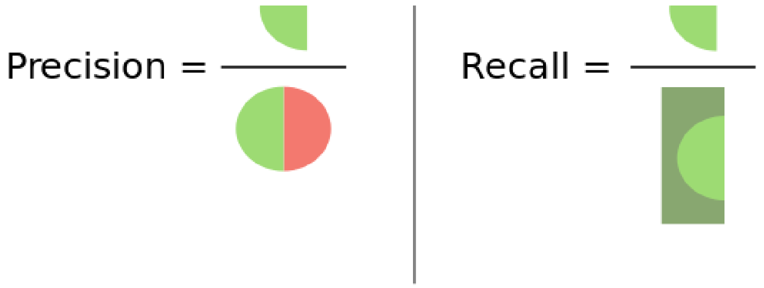
Recall measures the proportion $\frac{\text{TP}}{\text{TP+FN}}$ of spam emails that were correctly flagged as spam.

False-alarm rate measures the proportion $\frac{\text{FP}}{\text{FP+TN}}$ of ham emails that were incorrectly flagged as spam.

The following image might help:


relevant elements
false negatives
true negatives
true positives
false positives
selected elements


How many selected items are relevant?
How many relevant items are selected?



Note that a true positive (TP) is a spam email that is classified as spam, and a true negative (TN) is a ham email that is classified as ham. Answer the following questions in the cells below:

- (a) Suppose we have a classifier that just predicts 0 (ham) for every email. How many false positives are there? How many false negatives are there? Provide specific numbers using the training data from Question 4.
- (b) Suppose we have a classifier that just predicts 0 (ham) for every email. What is its accuracy on the training set? What is its recall on the training set?
- (c) What are the precision, recall, and false-alarm rate of the logistic regression classifier in Question 5? Are there more false positives or false negatives?
- (d) Our logistic regression classifier got 75.6% prediction accuracy (number of correct predictions / total). How does this compare with predicting 0 for every email?
- (e) Given the word features we gave you above, name one reason this classifier is performing poorly.
- (f) Which of these two classifiers would you prefer for a spam filter and why? (N.B. there is no "right answer" here but be thoughtful in your reasoning).

In [ ]:

```

```

In [18]:

```python
# provide number of FP and FN, respectively,
# for a classifier that always predicts 0 (never predicts positive...)
zero_predictor_fp = 0 #not possible
zero_predictor_fn = len(train[train["spam"] == 1])
# YOUR CODE HERE
#raise NotImplementedError()
```

In [19]:

```python
# This is a cell with just a comment but don't delete me if you want to get credit.
```

In [20]:
```
# provide training accuracy & recall, respectively,
# for a classifier that always predicts 0
zero_predictor_acc = len(train[train['spam'] == 0])/len(train)
zero_predictor_recall = 0 #no values were flagged as spam
zero_predictor_acc
# YOUR CODE HERE
#raise NotImplementedError()
```

Out[20]:
```
0.7447091707706642
```

In [21]:
```
# This is a cell with just a comment but don't delete me if you want to get credit.
```

In [22]:
```
# provide training accuracy & recall, respectively,
# for logistic regression classifier from question 5
tp = np.sum(y_pred & Y_train)
fp = np.sum(~Y_train & y_pred)
fn = np.sum(~y_pred & Y_train)
tn = -1*np.sum(~Y_train & ~y_pred)
logistic_predictor_precision = tp/(tp+fp)
logistic_predictor_recall = tp/(tp+fn)
logistic_predictor_far = fp/(fp+tn)
logistic_predictor_far

# YOUR CODE HERE
#raise NotImplementedError()
```

Out[22]:
```
0.012609819121447029
```

In [23]:
```
# This is a cell with just a comment but don't delete me if you want to get credit.
```

c) There are more false negatives as the value of the logistic_predictor precision is a larger value than the logisitic_predictor recall where the only discrepancy is one has a fp in the denominator whereas the other one has a fn. Since the one with fn is a smaller value this must mean that there are more false negatives than false postives.

d) The value for the zero_predictor_acc is about 74.5% so we see from this that it is not much worse than the value of our logistic regression classifier. From this we can conclude that our logistic regression classifier likely is not very good as it is comparable to a classifier that literally only predicts 'ham'.

e) The word list containing 'drug', 'bank', 'prescription', 'memo', 'private' which we used for the logistic regression classifier has a very low recall rate of about 11.4% meaning that only this percentage of spam is correctly identified as spam. For obvious reasons, we would want to have a model with a higher recall rate than this as spam should have discernable features to allow for better accuracy.

f) I would prefer the zero classifier despite the worse accuracy as there is less of an attempt to be correct whereas the logistic regression classifier attempts to predict but does so very inaccurately. The zero classifier actually depicts a more approachable classifier to work with as it is the most generic starting place.

Part II - Moving Forward

With this in mind, it is now your task to make the spam filter more accurate. In order to get full credit on the accuracy part of this assignment, you must get at least **88%** accuracy on the test set. To see your accuracy on the test set, you will use your classifier to predict every email in the `test` DataFrame and upload your predictions to Kaggle.

To prevent you from overfitting to the test set, you may only upload predictions to Kaggle twice per day. This means you should start early and rely on your **validation data** to estimate your Kaggle scores.

Here are some ideas for improving your model:

1. Finding better features based on the email text. Some example features are:
   A. Number of characters in the subject / body
   B. Number of words in the subject / body
   C. Use of punctuation (e.g., how many '!' were there?)
   D. Number / percentage of capital letters
   E. Whether the email is a reply to an earlier email or a forwarded email
2. Finding better words to use as features. Which words are the best at distinguishing emails? This requires digging into the email text itself.
3. Better data processing. For example, many emails contain HTML as well as text. You can consider extracting out the text from the HTML to help you find better words. Or, you can match HTML tags themselves, or even some combination of the two.
4. Model selection. You can adjust parameters of your model (e.g. the regularization parameter) to achieve higher accuracy. Recall that you should use cross-validation to do feature and model selection properly! Otherwise, you will likely overfit to your training data.

You may use whatever method you prefer in order to create features. However, **you are only allowed to train logistic regression models and their regularized forms**. This means no random forest, k-nearest-neighbors, neural nets, etc.

We will not give you a code skeleton to do this, so feel free to create as many cells as you need in order to tackle this task. However, answering questions 7, 8, and 9 should help guide you.

Note: You should use the validation data to evaluate your model and get a better sense of how it will perform on the Kaggle evaluation.

In [24]:
```
#where I come up with ideas
import re
my_array_words = ['html', 'body', 'please','urgent',
                  '$', "#", 'remove', 'font', 'href', 'nbsp', '>', '<']
train['small_len'] = [1 if len(msg) < 8000 else 0 for msg in train['email']]
train['capital_count'] = [1 if len(re.findall('[A-Z]',
                            text)) > 25 else 0 for text in train['subject']]
```

In [25]:
```
def words_in_texts_mod(words, texts, data):
    '''
    Args:
        words (list-like): words to find
        texts (Series): strings to search in

    Returns:
        NumPy array of 0s and 1s with shape (n, p) where n is the
        number of texts and p is the number of words.
    '''
    indicator_array = [[w in t for w in words] for t in texts]
    num = 0
    val = 0
    for i in data['small_len']:
        indicator_array[num].append(i)
        num += 1
    for i in data['capital_count']:
        indicator_array[val].append(i)
        val += 1
    # YOUR CODE HERE
    #raise NotImplementedError()
    return np.array(indicator_array).astype(int)
```

In [26]:
```
My_train = words_in_texts_mod(my_array_words, train['email'], train)
```

In [27]:
```
my_model = linear_model.LogisticRegression()
my_model.fit(My_train, Y_train)
my_y_pred = my_model.predict(My_train)
my_training_accuracy = sum(my_y_pred == Y_train)/len(Y_train)
my_training_accuracy
```

Out[27]:
```
0.8953813390123786
```

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 / didn't work?
3. What was surprising in your search for good features?

1. Using the suggestions that were given I was able to attempt to implement better features in order to improve my model. Initially, I started by implementing the fact that spam messages are more likely to occur at smaller email lengths and therefore assigned a penalty to emails of length less than 8000 characters. Additionally, I added a feature that calculates the number of capital letters as I noticed that spam emails are more likely to have a large amount of capital letters.
2. I found that my strategies from 1 were very effective and that looking for certain words that are associated with spam was very effective. In my efforts to improve my model I found that counting the number of '!' in each email was not a helpful means to improve my model and that assigning weights to different categories for my model did not work as planned.
3. The fact that adjusting the weights for the model wasn't effective seemed interesting as I attempted to assign a heavier factor to the features I implemented which were the length of the email and the number of capital letters. I later realized that this was ineffective as sklearn does this better than I could manually. Further, the specific words that seemed to represent higher spam likelihood were interesting as they oftentimes were words that coincide predominantly in the same emails such as "html" and "body", yet including both helped the model. Additionally, when I attempted to graph the distplots of a feature for both spam and ham I found that oftentimes the graphs resembled each other which made chossing the cutoff points for certain values very arbitrary.

Question 8 (EDA)

In the two cells below, show a visualization that you used to select features for your model. Include both

1. A plot showing something meaningful about the data that helped you during feature / model selection.
2. 2-3 sentences describing what you plotted and what its implications are for your features.

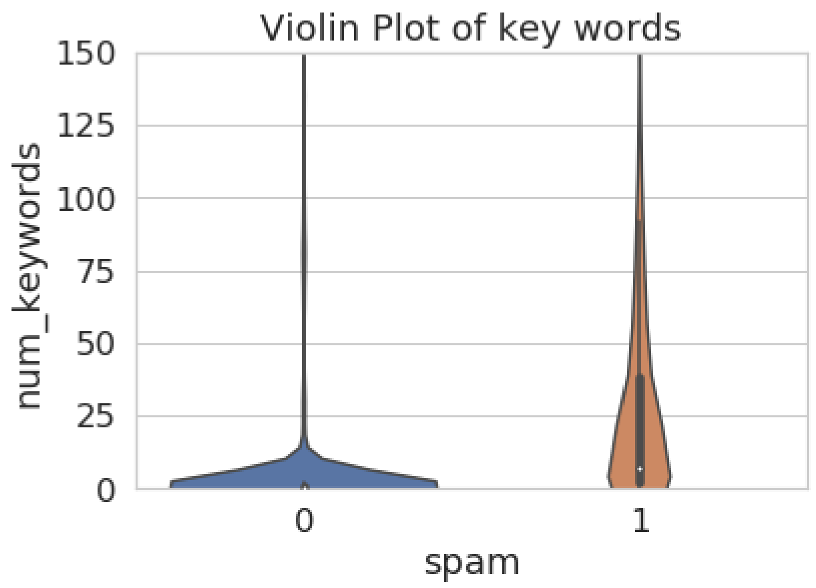
Feel to create as many plots as you want in your process of feature selection, but select one for the cells below.

You should not show us a visualization just like in question 3. Specifically, don't show us a bar chart of proportions, or a one-dimensional class conditional density plot. Any other plot is acceptable, as long as it comes with thoughtful commentary. Here are some ideas:

1. Consider the correlation between multiple features (look up correlation plots and `sns.heatmap`).
2. Try to show redundancy in a group of features (e.g. `body` and `html` might co-occur relatively frequently, or you might be able to design a feature that captures all html tags and compare it to these).
3. Use a word-cloud or another visualization tool to characterize the most common spam words.
4. Visually depict whether spam emails tend to be wordier (in some sense) than ham emails.

In [28]:
```
word_check = ['html', 'body', 'please','urgent','remove', 'font']
num_keywords = [sum([text.count(word) for
                     word in word_check]) for text in train['email']]
train['num_keywords'] = num_keywords
```

In [29]:
```
# YOUR CODE HERE
sns.violinplot(x = 'spam', y = 'num_keywords', data = train)
plt.ylim(0,150)
plt.title("Violin Plot of key words");
#raise NotImplementedError()
```



This violinplot displays the distribution of the number of keywords that occur in the spam and the ham categories, where the key words were choosen specifically to see their effectiveness. We can see from the diagram that the majority of ham emails do not possess any of these words whereas the expectation for a spam email would be that there is one of these words in the email reflecting their overal significance as features. Although we cannot see the effectiveness of an individual feature in this plot, we can see the effectiveness of a collection of features.

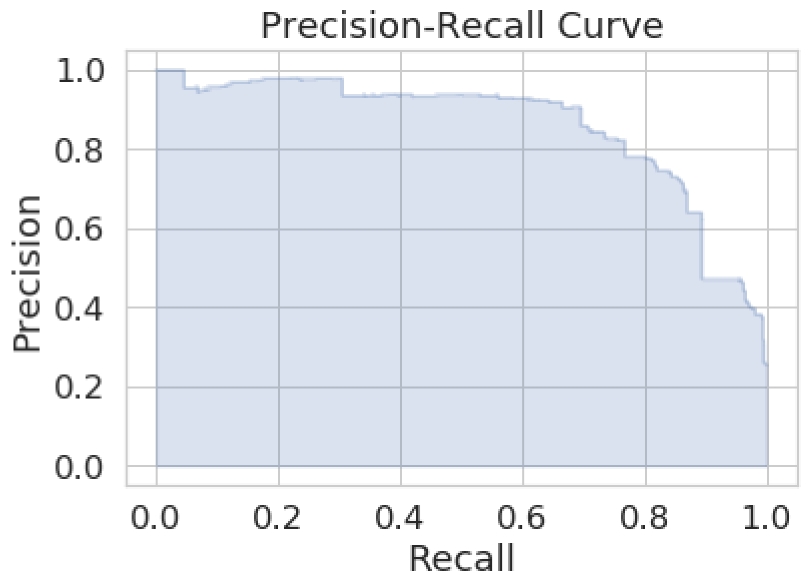
Question 9 (Making a Precision-Recall Curve)

We can trade off between precision and recall. In most cases we won't be able to get both perfect precision (i.e. no false positives) and recall (i.e. no 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 a disease until it's too late to treat, while a false positive means that a patient will probably have to take another screening.

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 precision-recall curve shows this trade off for each possible cutoff probability. In the cell below, plot a precision-recall curve (http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#plot-the-precision-recall-curve) for your final classifier (the one you use to make predictions for Kaggle).

In [30]:
```
from sklearn.metrics import precision_recall_curve
from sklearn.utils.fixes import signature

# Note that you'll want to use the .predict_proba(...) method for your classifier
# instead of .predict(...) so you get probabilities, not classes
my_model.fit(My_train, Y_train)
my_y_pred_prob = my_model.predict_proba(My_train)[:,1]
precision, recall, thresholds = precision_recall_curve(Y_train, my_y_pred_prob)
step_kwargs = ({'step': 'post'}
               if 'step' in signature(plt.fill_between).parameters
               else {})
plt.step(recall, precision, color='b', alpha=0.2, where='post')
plt.fill_between(recall, precision, alpha=0.2, color='b', **step_kwargs)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve');
# YOUR CODE HERE
#raise NotImplementedError()
```



Question 10: Submitting to Kaggle

The following code will write your predictions on the test dataset to a CSV, which you can submit to Kaggle. You may need to modify it to suit your needs.

Save your predictions in a 1-dimensional array called `test_predictions` . Even if you are not submitting to Kaggle, please make sure you've saved your predictions to `test_predictions` as this is how your grade for this part will be determined.

Remember that if you've performed transformations or featurization on the training data, you must also perform the same transformations on the test data in order to make predictions. For example, if you've created features for the words "drug" and "money" on the training data, you must also extract the same features in order to use scikit-learn's `.predict(...)` method.

You should submit your CSV files to https://www.kaggle.com/t/d9a7013e7fd048c291ff7efe6e1ac25e (https://www.kaggle.com/t/d9a7013e7fd048c291ff7efe6e1ac25e)

In [31]:
```
test.fillna("", inplace = True)
test['capital_count'] = [1 if len(re.findall('[A-Z]',
                    text)) > 25 else 0 for text in test['subject']]
test['small_len'] = [1 if len(msg) < 8000 else 0 for msg in test['email']]
testPhi = words_in_texts_mod(my_array_words, test['email'], test)
```

In [32]:
```
# CHANGE ME (Currently making random predictions)
test_predictions = my_model.predict(testPhi)

# YOUR CODE HERE
#raise NotImplementedError()
```

In [33]:
```
# must be ndarray of predictions
assert isinstance(test_predictions, np.ndarray)

# must be binary labels (0 or 1) and not probabilities
assert np.all((test_predictions == 0) | (test_predictions == 1))

# must be the right number of predictions
assert test_predictions.shape == (1000, )
```

In [34]:
```
# Please do not modify this cell
```

The following saves a file to submit to Kaggle.

```
In [35]: from datetime import datetime

         # Assuming that your predictions on the test set are stored in a 1-dimensional array called
         # test_predictions. Feel free to modify this cell as long you create a CSV in the right format.

         # must be ndarray of predictions
         assert isinstance(test_predictions, np.ndarray)

         # must be binary labels (0 or 1) and not probabilities
         assert np.all((test_predictions == 0) | (test_predictions == 1))

         # must be the right number of predictions
         assert test_predictions.shape == (1000, )

         # Construct and save the submission:
         submission_df = pd.DataFrame({
             "Id": test['id'],
             "Class": test_predictions,
         }, columns=['Id', 'Class'])
         timestamp = datetime.isoformat(datetime.now()).split(".")[0]
         submission_df.to_csv("submission_{}.csv".format(timestamp), index=False)

         print('Created a CSV file: {}.'.format("submission_{}.csv".format(timestamp)))
         print('You may now upload this CSV file to Kaggle for scoring.')
```

```
Created a CSV file: submission_2018-11-08T01:50:29.csv.
You may now upload this CSV file to Kaggle for scoring.
```

Submission

You're done!

Before submitting this assignment, ensure to:

1. Restart the Kernel (in the menubar, select Kernel->Restart & Run All)
2. Validate the notebook by clicking the "Validate" button

Finally, make sure to **submit** the assignment via the Assignments tab in Datahub