In [1]:
```
from client.api.notebook import Notebook
ok = Notebook('proj1.ok')
```

```
=====================================================================
Assignment: proj1
OK, version v1.18.1
=====================================================================
```

# Project 1: Spam/Ham Classification

## Due Date: Friday 5/21, 11:59 PM

**Collaboration Policy**

Data science is a collaborative activity. While you may talk with others about the project, 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 notebook.

**Collaborators**: *list collaborators here*

## This Assignment

In this project, 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 project, 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 minimizing 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!

## Score Breakdown

| Question | Points |
|---|---|
| 1a | 1 |
| 1b | 1 |
| 1c | 2 |
| 2 | 3 |
| 3 | 2 |
| 4 | 2 |
| 5 | 2 |
| 6a | 1 |
| 6b | 1 |
| 6c | 2 |
| 6d | 2 |
| 6e | 2 |
| 7 | 6 |
| 8 | 6 |
| 9 | 6 |
| 10 | 6 |
| Total | 45 |

# 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

In email classification, our goal is to classify emails as spam or not spam (referred to as "ham") using features generated from the text in the email.

The dataset consists of email messages and their labels (0 for ham, 1 for spam). Your labeled training 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 is spam, 0 if the email is ham (not spam)

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

In [3]:
```
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()
```

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 missing values.

- Fill in the cell below to print the number of NaN values in each column.
- If there are NaN values, replace them with appropriate filler values (i.e., NaN values in the `subject` or `email` columns should be replaced with empty strings).
- Print the number of NaN values in each column after this modification to verify that there are no NaN values left.

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

*The provided test checks that there are no missing values in your dataset.*

In [4]:
```
# BEGIN YOUR CODE
# ----------------------
print('Before imputation:')
print(original_training_data.isnull().sum())
original_training_data = original_training_data.fillna('')
print('------------')
print('After imputation:')
print(original_training_data.isnull().sum())
# ----------------------
# END YOUR CODE
```

```
Before imputation:
id         0
subject    6
email      0
spam       0
dtype: int64
------------
After imputation:
id         0
subject    0
email      0
spam       0
dtype: int64
```

In [5]:
```
ok.grade("q1a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooooook] 100.0% passed
```

## Question 1b

In the cell below, print the text of the first ham and the first spam email in the original training set.

*The provided tests just ensure that you have assigned `first_ham` and `first_spam` to rows in the data, but only the hidden tests check that you selected the correct observations.*

In [6]:
```
# BEGIN YOUR CODE
# ----------------------
first_ham = original_training_data['email'][0]
first_spam = original_training_data['email'][2]
# ----------------------
# END YOUR CODE

print('The text of the first Ham:')
print('------------')
print(first_ham)

print('The text of the first Spam:')
print('------------')
print(first_spam)
```

```
The text of the first Ham:
------------
url: 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 b
y the
 receivers. link[1] discuss[2] (_thanks, misha!_)

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



The text of the first Spam:
------------
<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]:
```
ok.grade("q1b");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

---

### Question 1c

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

Answer: `Spam email uses HTML, the standard markup language for Web pages. In the contrast, ham email just uses plain texts.`

# 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 assess the performance of your classifier 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 that is the same for every student. **Do not modify this in the following questions, as our tests 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 email is **ham** or **spam**. This is a *classification* problem, and here we use logistic regression to train a classifier.

Recall that to train an logistic regression model we need:

- a numeric feature matrix $X$
- a vector of 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 $X$ is an email.
- Each column of $X$ contains one feature for all the emails.

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 worldhello']))

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

**Hint**: pandas.Series.str.contains (https://pandas.pydata.org/docs/reference/api/pandas.Series.str.contains.html)

*The provided tests make sure that your function works correctly, so that you can use it for future questions.*

In [12]:
```
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.
    '''
    # BEGIN YOUR CODE
    # ----------------------
    indicator_array = []
    for t in texts:
        word= []
        for w in words:
            if w in t:
                word.append(1)
            else:
                word.append(0)
        indicator_array.append(word)
    # ----------------------
    # END YOUR CODE

    return indicator_array
```

In [13]:
```
ok.grade("q2");
```

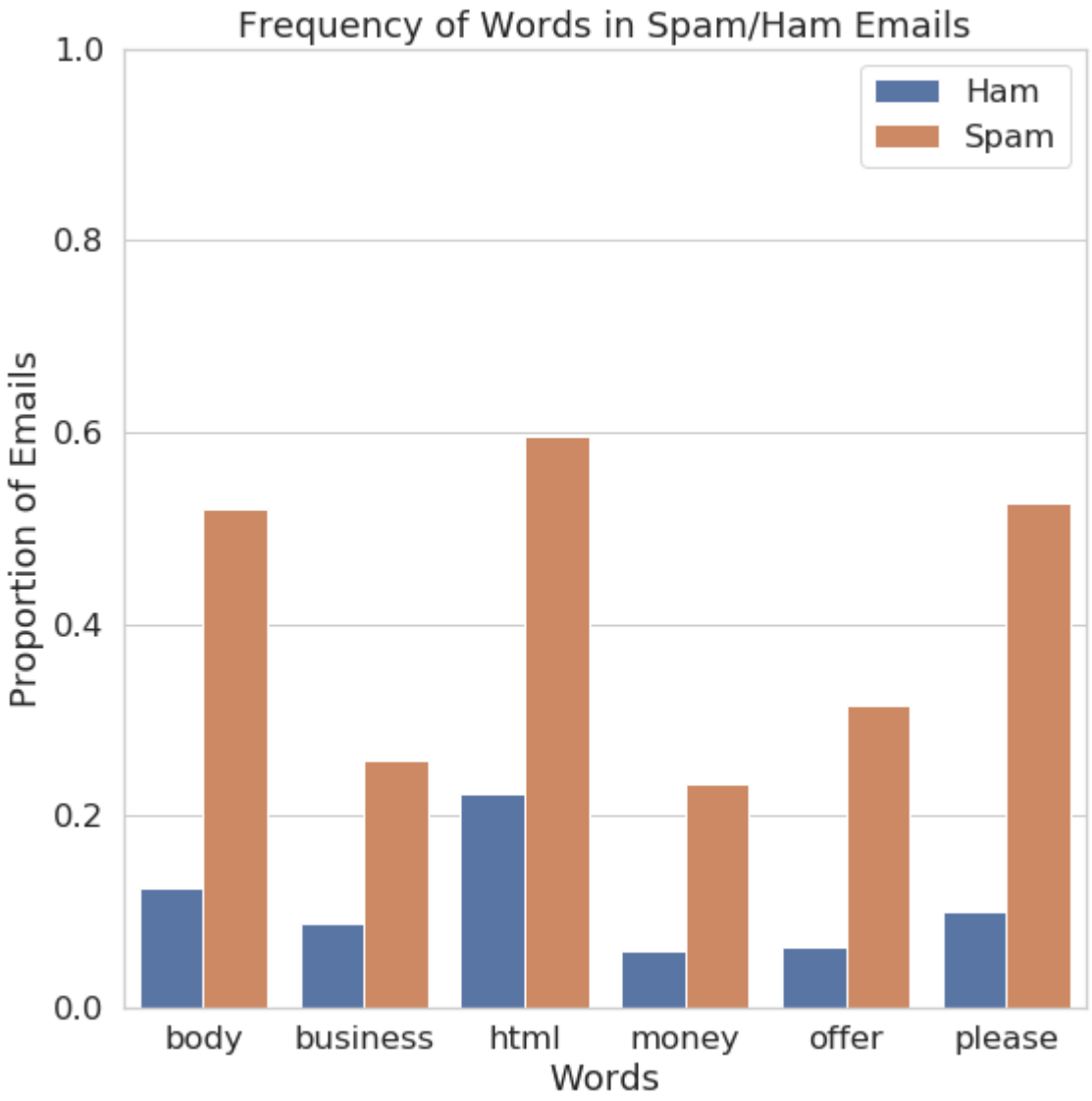
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

# Basic EDA

We need to identify some features that allow us to distinguish spam emails from ham emails. 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.

The following plot (which was created using `sns.barplot`) compares the proportion of emails in each class containing a particular set of words.



In [14]:
```
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 each row is a sentence, and the value of 1 or 0 indicates the number of occurances of the word in this sentence."))
display(df);
display(Markdown("> `melt` will turn columns into variale, notice how `word_1` and `word_2` become `variable`, their values are stoed in the value column"))
display(df.melt("type"))
```

> Our Original DataFrame has some words column and a type column. You can think of each row is a sentence, and the value of 1 or 0 indicates the number of occurances of the word in this sentence.

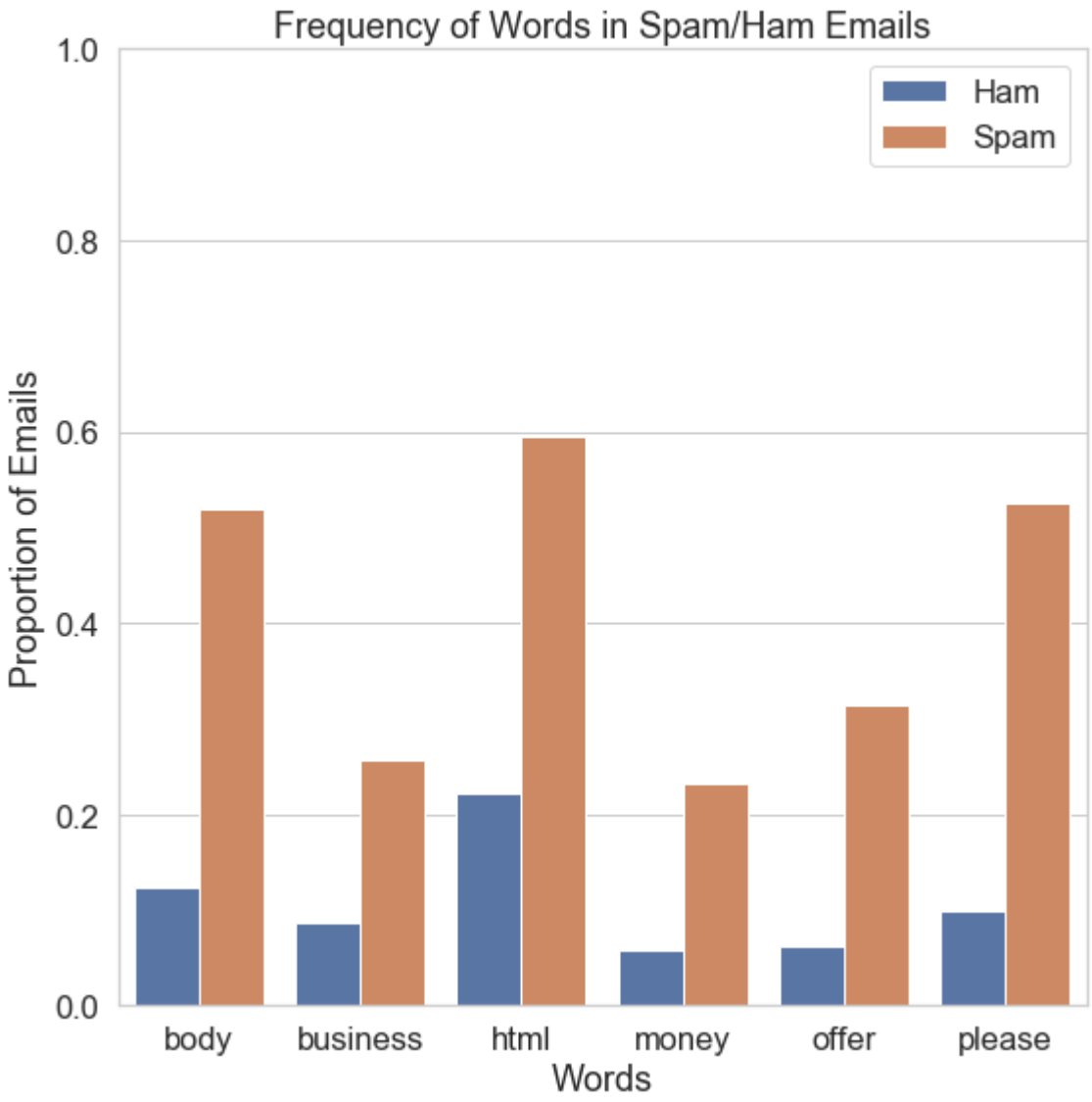
| | word_1 | word_2 | type |
|---|---|---|---|
| **0** | 1 | 0 | spam |
| **1** | 0 | 1 | ham |
| **2** | 1 | 0 | ham |
| **3** | 0 | 1 | ham |

> `melt` will turn columns into variale, notice how `word_1` and `word_2` become `variable`, their values are stoed in the value column

| | 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 |

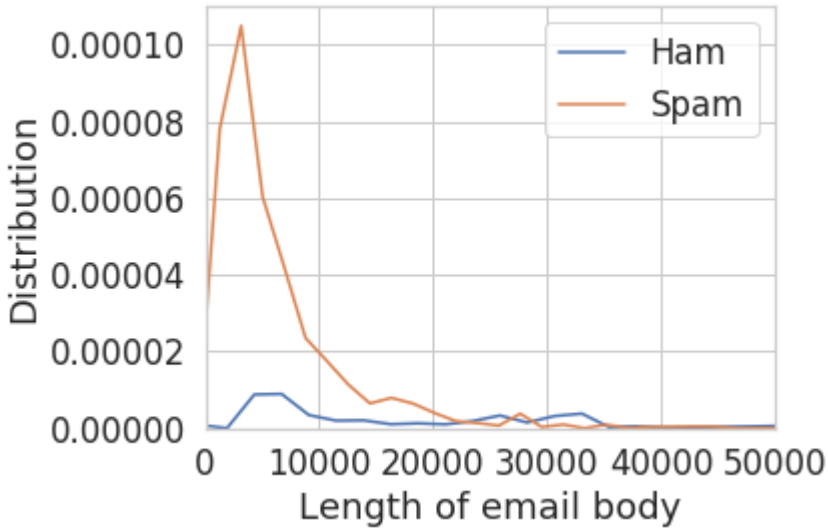
We can create a bar chart like the one above comparing the proportion of spam and ham emails containing certain words. Choose a set of words that are different from the ones above, but also have different proportions for the two classes. Make sure that we only consider emails from `train`.

In [15]:
```
# We must do this in order to preserve the ordering of emails to labels for
words_in_texts
train=train.reset_index(drop=True)

some_words = ['body', 'html', 'please', 'money', 'business', 'offer']
Phi_train = words_in_texts(some_words, train['email'])

df = pd.DataFrame(data = Phi_train, columns = some_words)
df['label'] = train['spam']

plt.figure(figsize=(8,8))
sns.barplot(x = "variable",
            y = "value",
            hue = "label",
            data = (df
                    .replace({'label':
                                {0 : 'Ham',
                                 1 : 'Spam'}})
                    .melt('label')
                    .groupby(['label', 'variable'])
                    .mean()
                    .reset_index()))

plt.ylim([0, 1])
plt.xlabel('Words')
plt.ylabel('Proportion of Emails')
plt.legend(title = "")
plt.title("Frequency of Words in Spam/Ham Emails")
plt.tight_layout()
plt.show()
```


Frequency of Words in Spam/Ham Emails
Ham
Spam
Proportion of Emails
1.0
0.8
0.6
0.4
0.2
0.0
body
business
html
money
offer
please
Words

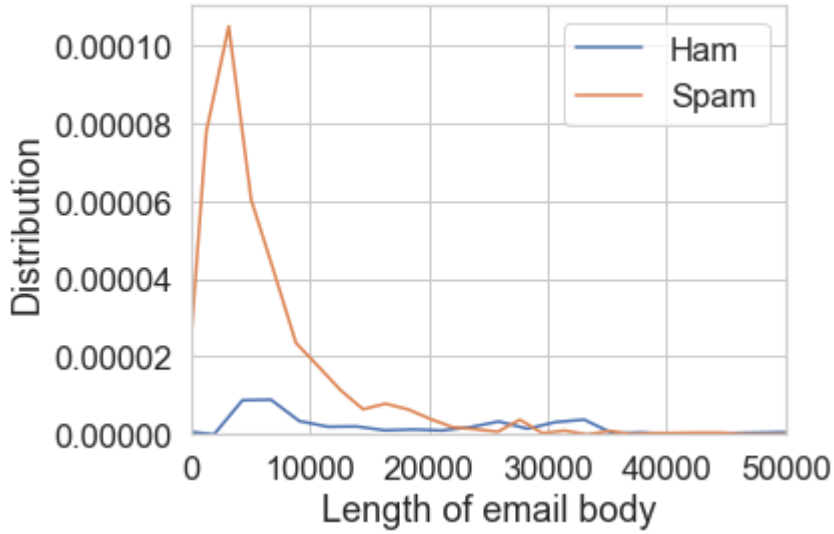
## Question 3

When the feature is binary, it makes sense to compare its proportions across classes (as in the previous question). Otherwise, if the feature can take on numeric values, we can compare the distributions of these values for different classes.




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 [16]:
```
# BEGIN YOUR CODE
# ----------------------
tmp = train.copy()
tmp['length'] = tmp['email'].str.len()
sns.distplot(tmp.loc[tmp['spam'] == 0, 'length'],hist=False, label='Ham')
sns.distplot(tmp.loc[tmp['spam'] == 1, 'length'],hist=False, label='Spam')
plt.xlabel('Length of email body')
plt.ylabel('Distribution')
plt.xlim((0,50000))
plt.tight_layout()
# ----------------------
# END YOUR CODE
```




## 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: `X_train` and `Y_train`.

- `X_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.

*The provided tests check that the dimensions of your feature matrix (X) are correct, and that your features and labels are binary (i.e. consists of 0 and 1, no other values). It does not check that your function is correct; that was verified in a previous question.*

In [17]:
```
some_words = ['drug', 'bank', 'prescription', 'memo', 'private']

# BEGIN YOUR CODE
# ------------------------
X_train = np.array(words_in_texts(some_words, train['email']))
Y_train = np.array(train['spam'].tolist())
# ------------------------
# END YOUR CODE

X_train[:5], Y_train[:5]
```

Out[17]:
```
(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 [18]:
```
ok.grade("q4");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooooook] 100.0% passed
```

## 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 `X_train` and `Y_train`.
- Then, output the accuracy of the model (on the training data) in the cell below. You should get an accuracy around 0.75.

*The provided test checks that you initialized your logistic regression model correctly.*

In [19]:
```
from sklearn.linear_model import LogisticRegression

# BEGIN YOUR CODE
# ----------------------
model = LogisticRegression().fit(X_train, Y_train)
Y_pred = model.predict(X_train)
training_accuracy = model.score(X_train, Y_train)
# ----------------------
# END YOUR CODE

print("Training Accuracy: ", training_accuracy)
```

```
Training Accuracy:  0.757620125116
```

In [20]:
```
ok.grade("q5");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooook] 100.0% passed
```

# Evaluating Classifiers

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 evaluating accuracy 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. 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.

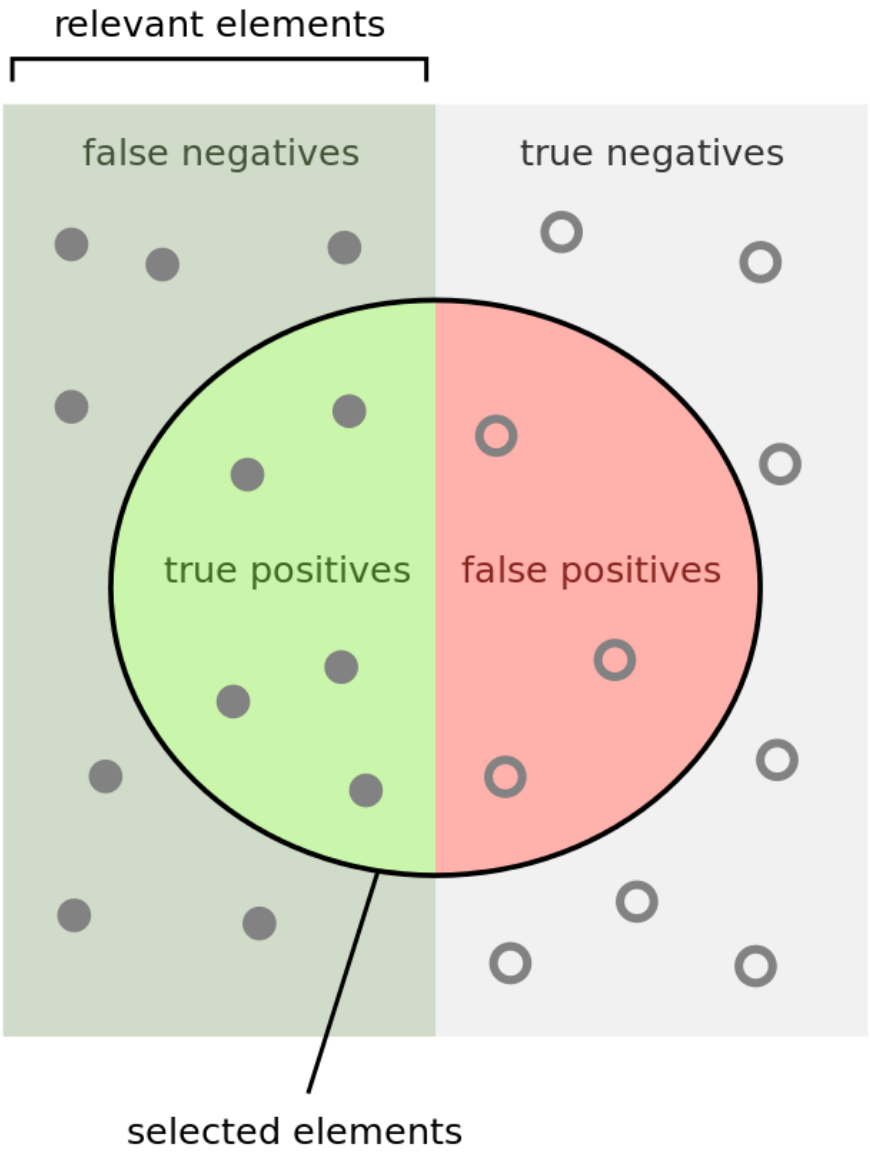
These definitions depend both on the true labels and the predicted labels. False positives and false negatives may be of differing importance, leading us to consider more ways of evaluating a classifier, in addition to overall accuracy:

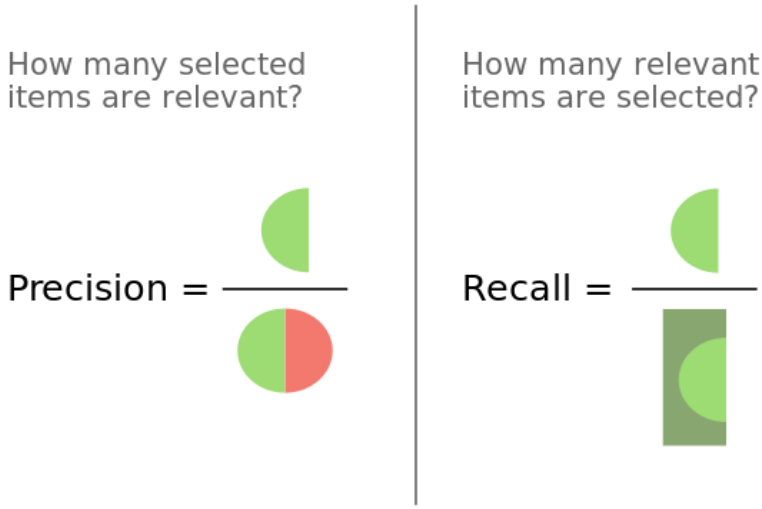
**Precision** measures the proportion $\frac{\text{TP}}{\text{TP}+\text{FP}}$ of emails flagged as spam that are actually spam.

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

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

The following image might help:




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.

## Question 6a

Suppose we have a classifier `zero_predictor` that always predicts 0 (never predicts positive). How many false positives and false negatives would this classifier have if it were evaluated on the training set and its results were compared to `Y_train`? Fill in the variables below (answers can be hard-coded):

*Tests in Question 6 only check that you have assigned appropriate types of values to each response variable, but do not check that your answers are correct.*

In [21]:
```
# BEGIN YOUR CODE
# ----------------------
zero_predictor_fp = 0
zero_predictor_fn = sum(Y_train==1)
# ----------------------
# END YOUR CODE
```

In [22]:
```
ok.grade("q6a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooook] 100.0% passed
```

## Question 6b

What are the accuracy and recall of `zero_predictor` (classifies every email as ham) on the training set? Do NOT use any `sklearn` functions.

In [23]:
```
# BEGIN YOUR CODE
# ----------------------
zero_predictor_acc = (len(Y_train)-Y_train.sum())/len(Y_train)
zero_predictor_recall = 0
# ----------------------
# END YOUR CODE
```

In [24]:
```
ok.grade("q6b");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 100.0% passed
```

## Question 6c

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

Answer: Because zero_predictor always predicts 0 (classifies every email as ham), therefore, the False Positive will always be 0. The False Negative here is 1918, meaning there are 1918 spam emails being mislabeled as ham emails.Because the zero_predictor always predict 0, therefore the True Positive is always 0, then the recall of this model is 0.The accuracy of this zero_predictor is around 0.74, which represents the proportion of ham emails predicted as ham emails.

## Question 6d

Compute the precision, recall, and false-alarm rate of the `LogisticRegression` classifier created and trained in Question 5. **Note: Do NOT use any `sklearn` functions.**

In [31]:
```
# BEGIN YOUR CODE
# ----------------------
logistic_predictor_precision = sum((Y_train==1) & (Y_pred==1))/sum((Y_pred==1))
logistic_predictor_recall = sum((Y_train==1) & (Y_pred==1))/sum((Y_train==1))
logistic_predictor_far = sum((Y_train==0) & (Y_pred==1))/sum((Y_train==0))
# ----------------------
# END YOUR CODE
```

In [32]:
```
ok.grade("q6d");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooooook] 100.0% passed
```

## Question 6e

1. Our logistic regression classifier got 75.6% 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.

Answer: `

1. The logistic regression classifier has a slightly better prediction accuracy in comparison to the zero-predictor classifier.
2. "drug" and "prescription" in some_words won't appear frequently in emails so they are not good features to distinguish spam/ham emails, which makes this classifier perform poorly.
3. I prefer the logistic regression classifier because the training accuarcy is slightly higher than the zero_predictor and the recall of the logistic regression classifier is higher, which means the proportion of spam emails that were correctly flagged as spam is higher. `

# 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 **77%** 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.

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.

ou may use whatever method you prefer in order to create features, but **you are not allowed to import any external feature extraction libraries**. In addition, **you are only allowed to train logistic regression models**. No random forests, k-nearest-neighbors, neural nets, etc.

We have not provided any code 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 [35]:
```
# Split training data into spam emails and ham emails.
spam=train[train['spam']==1]
ham=train[train['spam']==0]

#use split()to isolate the words
#Here is a reference to what I learned in the ML class before
#The regular expression 0-9a-zA-Z is used to match numbers or letters
split2spam=spam['email'].str.replace(r'[^0-9a-zA-Z]',' ').str.split()
split2ham = ham['email'].str.replace(r'[^0-9a-zA-Z]',' ').str.split()

dict4spam={}#spam words dictionary
dict4ham={}#ham words dictionary

def count_word_spam(tmp_lst):  #coint spam words in email
    for x in tmp_lst:
        if x not in dict4spam:
            dict4spam[x] = 1
        else:
            dict4spam[x] = dict4spam[x]+1

def count_word_ham(tmp_lst): #count ham words in eamil
    for x in tmp_lst:
        if x not in dict4ham:
            dict4ham[x] = 1
        else:
            dict4ham[x] = dict4ham[x]+1

#create a custom function into series.apply to update the dic each time it
 sees a word
split2spam.apply(count_word_spam)
split2ham.apply(count_word_ham)

#sort the key and value according to the value in descending order for two
 dic
sorted_spam_dict = sorted(dict4spam.items(), key=lambda x: x[1],reverse=Tru
e)
sorted_ham_dict = sorted(dict4ham.items(), key=lambda x: x[1],reverse=True)

sorted_spam_dict,sorted_ham_dict

spam1000 = [x[0] for x in sorted_spam_dict[:1000]]#1000words of spam dic
ham1000 = [x[0] for x in sorted_ham_dict[:1000]]#1000words of ham dic

#find the words in spam dictionary ,not in ham dictionary
tmp_lst=[]
for x in spam1000:
    if x in ham1000:
        pass
    else:
        tmp_lst.append(x)


#training
x_train=words_in_texts(tmp_lst, train['email'])
y_train= train['spam']
```

```
my_model=LogisticRegression().fit(x_train,y_train)
y_pred = my_model.predict(x_train)

training_accuracy = my_model.score(x_train, y_train)
print("Accuracy: ", training_accuracy)
```

```
Accuracy:  0.972314654599
```

## Question 7: EDA

In the cell 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 free to create as many plots as you want in your process of feature selection, but select one for the response cell below.

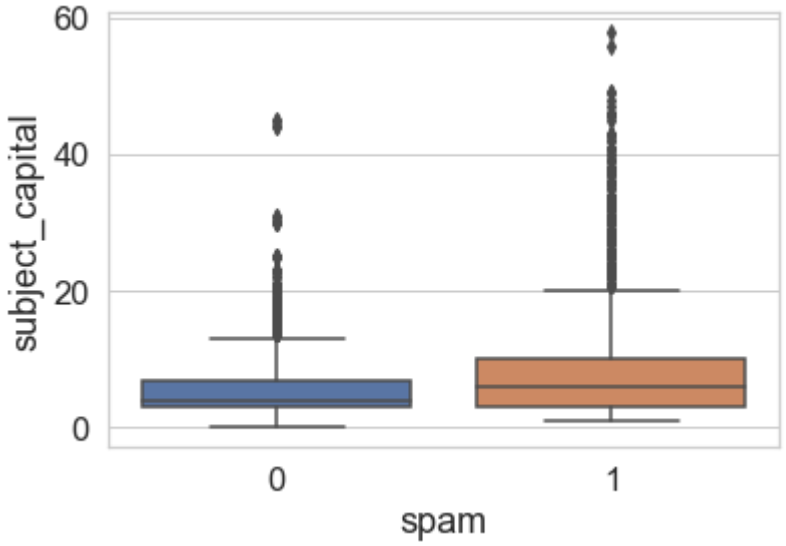
**You should not just produce an identical visualization to question 3.** Specifically, don't show us a bar chart of proportions, or a one-dimensional class-conditional density plot. Any other plot is acceptable, as long as it comes with thoughtful commentary. Here are some ideas:

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. Visualize which words have high or low values for some useful statistic.
4. Visually depict whether spam emails tend to be wordier (in some sense) than ham emails.

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

In [41]:
```
# Write your description (2-3 sentences) as a comment here:
#This plot shows the number of capital letters in the subject of spam (1) a
nd ham (0) emails.
#We can see that the spam emails have greater number of capital letters in
 their subjects.
#

# Write the code to generate your visualization here:
num_of_capital=train.subject.apply(lambda x: str(x)).str.findall("[A-Z]").a
pply(lambda x:len(x))
train["subject_capital"]=num_of_capital
sns.boxplot(x="spam",y="subject_capital",data=train)
plt.figure(figsize=(7,5))
```

Out[41]: <matplotlib.figure.Figure at 0x116d85f5160>




<matplotlib.figure.Figure at 0x116d85f5160>

## Question 8: 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.

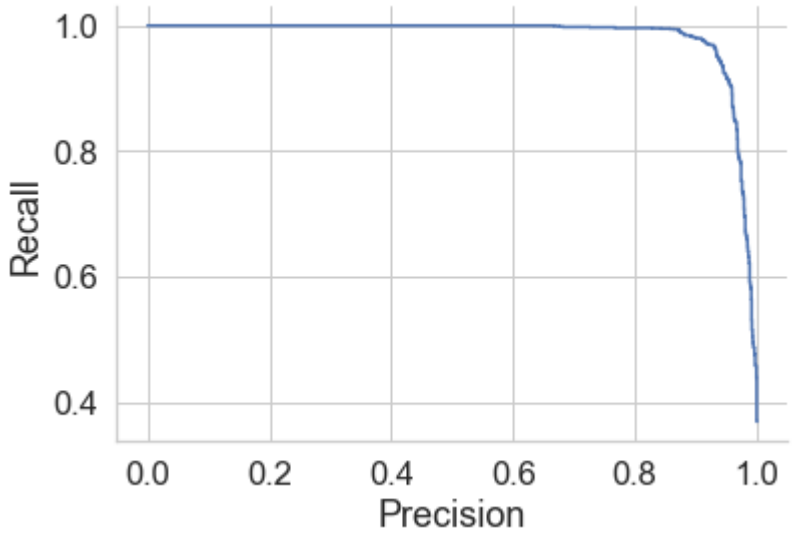
Recall that logistic regression calculates the probability that an example belongs to a certain class.

- Then, to classify an example we say that an email is spam if our classifier gives it $\geq 0.5$ probability of being spam.
- However, *we can adjust that cutoff*: we can say that an email is spam only if our classifier gives it $\geq 0.7$ probability of being spam.

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.

In [43]:
```
from sklearn.metrics import precision_recall_curve

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

# BEGIN YOUR CODE
# ----------------------
y_predict = my_model.predict_proba(x_train)[:,1]
y_predict

recall, precision, thresholds = precision_recall_curve(y_train, y_predict)
with sns.axes_style("whitegrid"):
    plt.plot(precision, recall)

sns.despine()
plt.xlabel('Precision')
plt.ylabel('Recall');
# ----------------------
# END YOUR CODE
```



## Question 9: 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 score for this question 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/c/cose471sp21project1 (https://www.kaggle.com/c/cose471sp21project1)

*The provided tests check that your predictions are in the correct format, but you must submit to Kaggle to evaluate your classifier accuracy.*

In [38]:
```
# BEGIN YOUR CODE
# ------------------------
test_predictions = my_model.predict(words_in_texts(tmp_lst, test['email']))
test_predictions = np.array(test_predictions.tolist())
# ------------------------
# END YOUR CODE
```

In [39]:
```
ok.grade("q9");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooooook] 100.0% passed
```

In [40]:
```
from datetime import datetime

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

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


print('You may now upload this CSV file to Kaggle for scoring.')
```

```
You may now upload this CSV file to Kaggle for scoring.
```

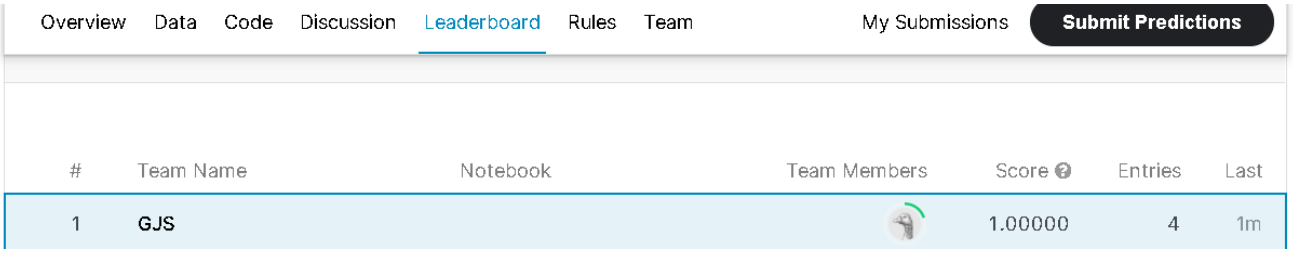
---

# Question 10: Attach Your Leaderboard Screenshot

Take a screenshot of your submission to Kaggle as follows. This screenshot should contain your testing score.

**You should replace `images/leaderboard_example.png` with your screenshot!**

Note that, in order to get full credit on the accuracy part of this assignment, you must get at least **88%** accuracy on the test set.



## Congratulations! You have completed Project 1.

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output.,

**Please save before submitting!**

Please generate pdf as follows and submit it to Gradescope.