0.0.1 Question 1c

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

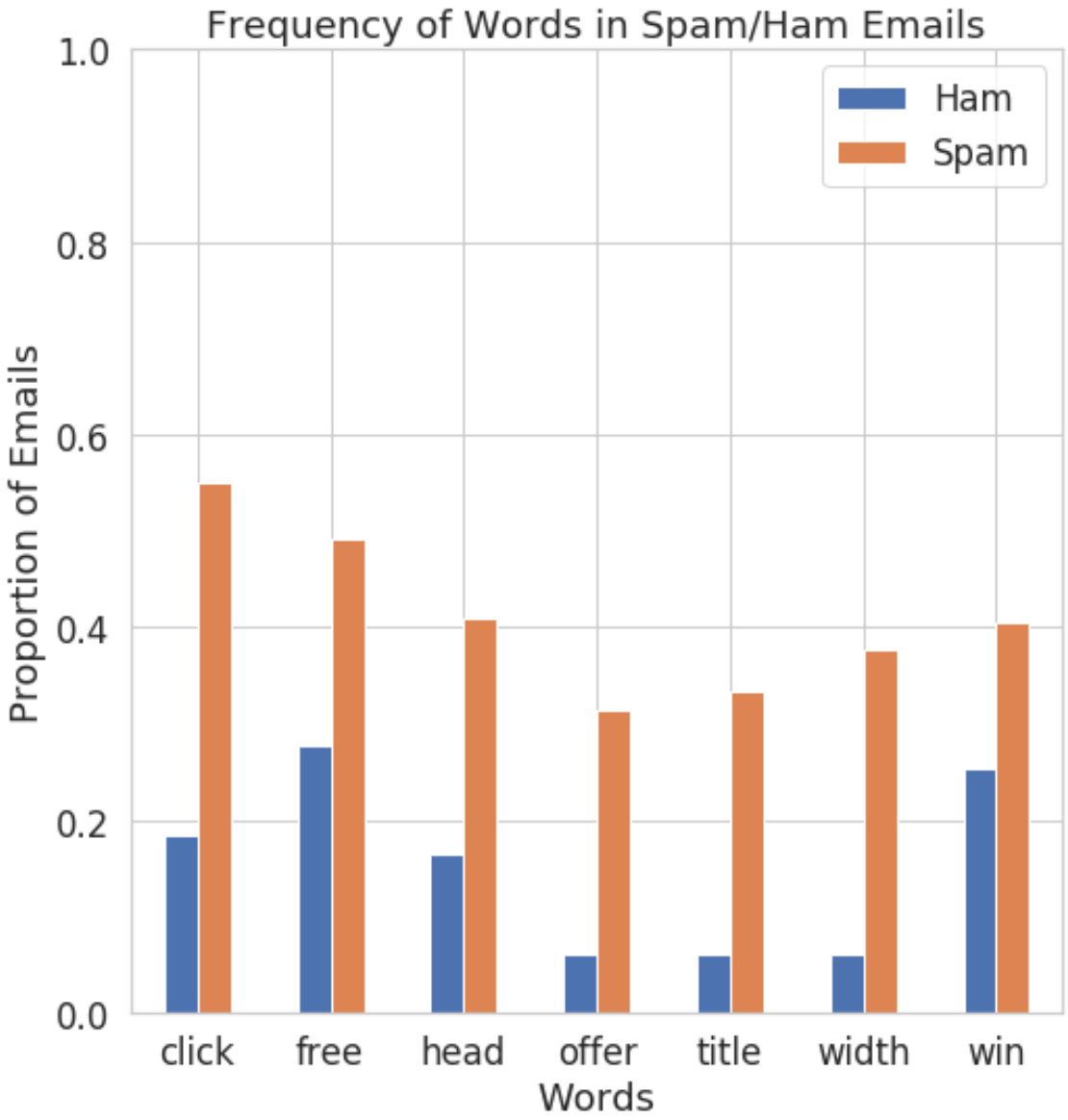
One major difference between the two emails is that the ham email looks like a lot of normal, human-readable text in the form of strings (although it includes links), whereas the spam email looks like html text.

0.0.2 Question 3a

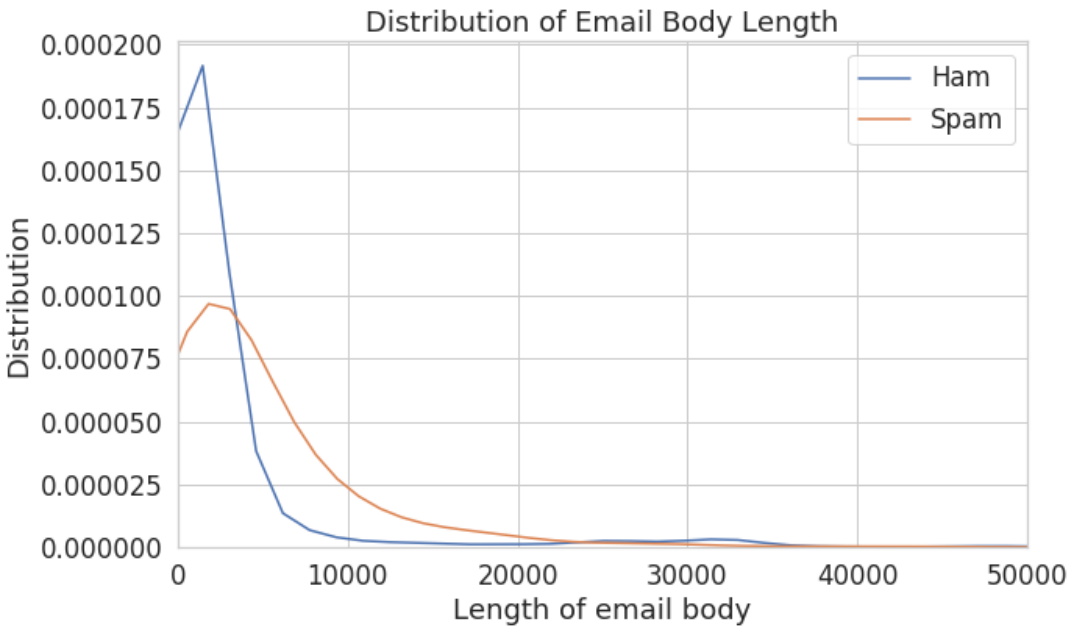
Create a bar chart like the one above comparing the proportion of spam and ham emails containing certain words. Choose a set of words that are different from the ones above, but also have different proportions for the two classes. Make sure to only consider emails from `train`.

In [203]:
```
train=train.reset_index(drop=True) # We must do this in order to preserve the ordering of ema


my_words = ['free', 'head', 'offer', 'title', 'click', 'win', 'width']
indicator = words_in_texts(my_words, train['email'])

spam_or_ham = train['spam'].replace({0: 'ham', 1: 'spam'})
my_df = pd.DataFrame(indicator, columns = my_words).assign(type=spam_or_ham)
my_df = pd.melt(my_df, id_vars = 'type', value_vars={'free', 'head', 'offer', 'title', 'click
my_df = my_df.groupby(['variable', 'type']).sum().unstack()
my_df = my_df['value'].assign(ham = my_df['value']['ham']/(train.shape[0]-train['spam'].sum()

my_df.plot(kind='bar', rot=0, figsize=(8, 8.5))
plt.ylim(0, 1)
plt.legend(['Ham', 'Spam'])
plt.xlabel('Words')
plt.ylabel('Proportion of Emails')
plt.title('Frequency of Words in Spam/Ham Emails')
```

Out[203]:
```
Text(0.5, 1.0, 'Frequency of Words in Spam/Ham Emails')
```

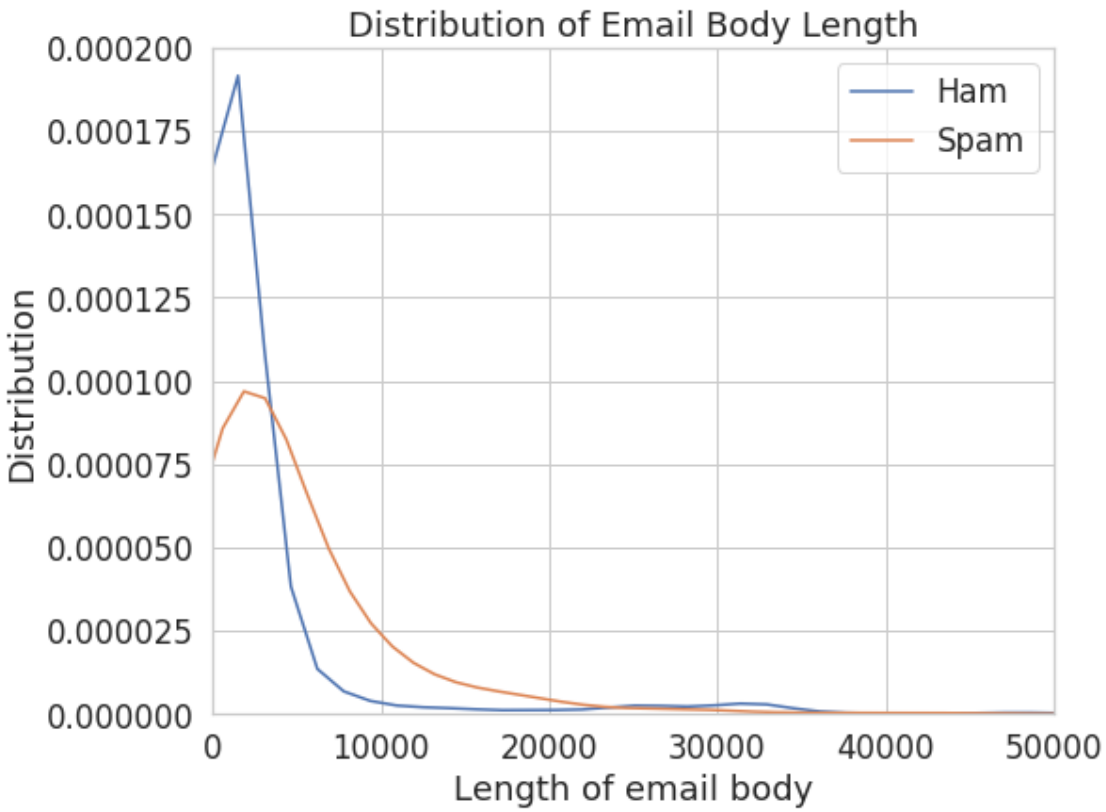

Frequency of Words in Spam/Ham Emails
Ham
Spam
Proportion of Emails
1.0
0.8
0.6
0.4
0.2
0.0
click
free
head
offer
title
width
win
Words

0.0.3 Question 3b



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

```
In [204]: my_plot_df = train.assign(Length = train['email'].apply(len))
          spam = my_plot_df[my_plot_df['spam']==1]['Length']
          ham = my_plot_df[my_plot_df['spam']==0]['Length']

          plt.figure(figsize=(8, 6.5))
          sns.distplot(ham, hist=False)
          sns.distplot(spam, hist=False)
          plt.xlim(0, 50000)
          plt.ylim(0, 0.000200)
          plt.xlabel('Length of email body')
          plt.ylabel('Distribution')
          plt.title('Distribution of Email Body Length')
          plt.legend(['Ham', 'Spam'])
          plt.savefig('training_conditional_densities.png')
```


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

0.0.4 Question 6c

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

FP: False positives refer to a "false alarm" i.e. when our prediction is positive, but the variable actually ends up being negative. In our case, we predict NONE of the predictions to be positive i.e. NONE of the emails to be spam. This is because we are using a zero predictor i.e. a classifier that always predicts negative. Thus, FP = 0 in our case.

FN: False negatives refer to our failure to detect something i.e. when our prediction is negative, but the variable actually ends up being positive. Since ALL our predictions are negative predictions, the number of false negatives is equal to the number of real positives. The number of real positives is given by the number of emails that were actually spam. We can find this value by summing the "spam" column of the train table.

Accuracy: The accuracy refers to the proportion of points we classified correctly and is given by $\frac{TP+TN}{N}$. Since we predicted none of them to be positive, $TP = 0$. Thus, our accuracy becomes $\frac{TN}{N}$. The number of true negatives are equal to the number of emails that were ham, which can be given by the total number of emails minus number of spam emails. The denominator of the expression is the total number of emails. This leads us to our expression: $\frac{len(train['spam'])-train['spam'].sum()}{len(train['spam'])}$.

Recall: The recall can be expressed in words by the following statement- of all observations that were actually 1, what proportion did we predict to be 1? The answer is zero. Since we used a zero predictor, we predicted NONE of our observations of be 1, and all of them to be 0. Thus, the numerator of the recall expression becomes 0, making our recall value 0.

0.0.5 Question 6e

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

```
In [215]: print(fp, fn)
```

```
122 1699
```

There are more false negatives (1699) as compared to false positives (122) when using the logistic regression classifier from Question 5. This means that a spam email is more likely to get mislabeled as ham and end up in the inbox as opposed to a ham email getting flagged as spam and filtered out of the inbox.

0.0.6 Question 6f

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

1) The zero_predictor got an accuracy of 74.47%, which is lower than the accuracy of the logistic regression classifier. However, this is not by much, so the logistic regression classifier does only a slightly better job.

2) The words we chose for our classifier ('drug', 'bank', 'prescription', 'memo', 'private') are not particularly common words, so they are unlikely to appear in either email, whether ham or spam. Due to this factor, they cannot classify the emails too accurately. There are wide reasons for this: these words are unlikely to appear in ham emails because ham emails are usually more professional, and these words are also unlikely to appear in spam emails because a lot of spam emails avoid these words in order to avoid getting tracked since these words are generally deemed as suspicious to authorities.

3) When deciding which classifier to choose, it is important to consider what our goal for the classifier is. Since the goal of our classifier is to filter out as many spam emails as possible, we want to look at our recall. Our recall is a good metric because it provides an answer to the following question- of all observations that were actually 1, what proportion did we predict to be 1 i.e. of all emails that were actually spam, what proportion did we successfully filter as out as spam? In the case of our zero_predictor, we got a recall of 0, so I would not choose that as a classifier. On the other hand, in the case of our logistic regression classifier, we got a recall of 11.4% (which isn't great, but it's better than the zero_predictor). Hence, I would choose the logistic regression classifier over the zero_predictor.

0.0.7 Question 7: Feature/Model Selection Process

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

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

```
In [227]: avg_num_capitals_in_spam = np.mean(num_capitals(my_train[my_train['spam']==1]['email']))
          avg_num_commas_in_spam = np.mean(num_symbols(my_train[my_train['spam']==1]['email'], ','))
          print("The average number of Uppercase Letters in spam emails is: ", avg_num_capitals_in_spam)
          print("The average number of commas in spam emails is: ", avg_num_commas_in_spam)
```

```
The average number of Uppercase Letters in spam emails is:  896.7194994786236
The average number of commas in spam emails is:  21.325860271115747
```

```
In [228]: avg_num_capitals_in_ham = np.mean(num_capitals(my_train[my_train['spam']==0]['email']))
          avg_num_commas_in_ham = np.mean(num_symbols(my_train[my_train['spam']==0]['email'], ','))
          print("The average number of Uppercase Letters in ham emails is: ", avg_num_capitals_in_ham)
          print("The average number of commas in ham emails is: ", avg_num_commas_in_ham)
```

```
The average number of Uppercase Letters in ham emails is:  181.33154602323503
The average number of commas in ham emails is:  16.11974977658624
```

1) I found better features for my models by going through the spam emails and observing that a lot of them were HTML-text heavy. Using this information, I created a better list of words (better_words) that contained primarily HTML words and tags e.g. 'html', 'div', 'body', 'head'. I also created functions to count the number of words in each email, the number of different symbols e.g. '!', '<' and '>' (due to the wide prevalence of these HTML characters in the spam emails). Furthermore, I also created functions to gauge whether each email was a reply or a forward, or none of those.

2) One thing that I tried that really worked was creating a function that gauged if the email was a reply or not. Initially, I only tried including forwards, but using replies worked quite well because most of the emails that were replies were ham emails. This makes sense, because a lot of the ham emails tend to be professional so there is a lot of "back-and-forth" between co-workers.

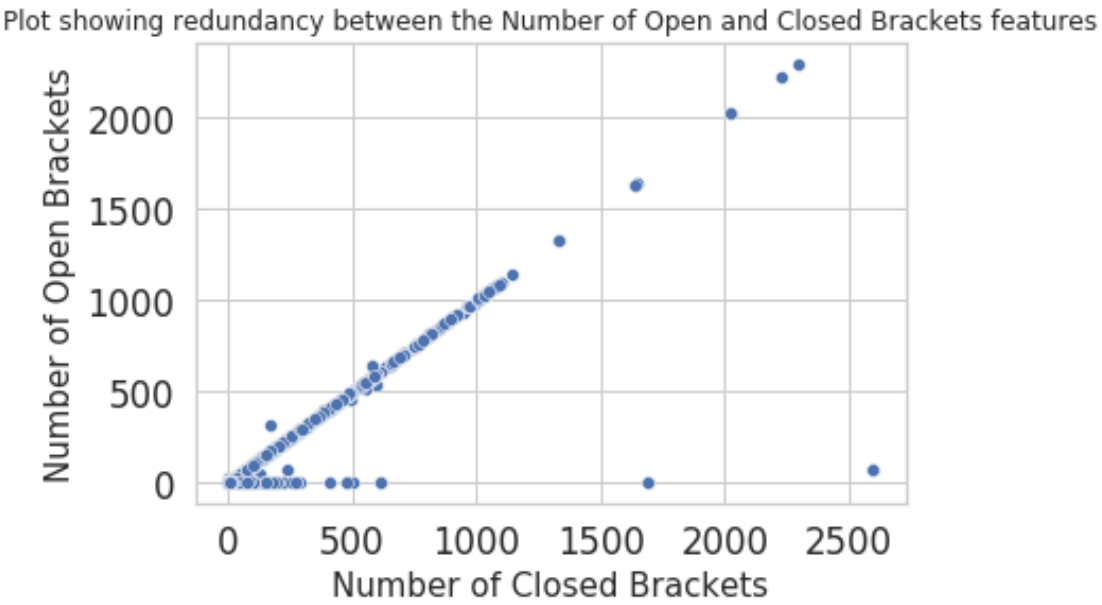
On the other hand, something that didn't work as well was including the number of '$' (dollar signs). I initially thought that this work well because a lot of spam emails tend to be offers about the cheapest new deals or some kind of money-related content, but that ended up lowering the accuracy of the model.

3) Something that was surprising in my search for good features is illustrated by the two cells above this cell. They show that the average number of Uppercase Letters and commas is higher in spam emails rather than ham emails. This is surprising because I initially thought that these would be higher in ham emails since they tend to have "better" punctuation as they are used for more professional work. However, quite the opposite turned out to be true.

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

In [229]:
```
# Write your description (2-3 sentences) as a comment here:
# The plot below illustrates the redundancy in the number_of_closed_brackets feature and the
# feature. The points seem to lie on almost a perfectly straight line hinting towards a very
# correlation. However, it is also important to note that the correlation is not perfect i.e.
# lie on the line. There are sufficient outliers, indicating that HTML tags are not the only
# ''>' symbols are used. They may have various other applications, e.g. they may be used for
# "greater than" or "less than". Another interesting observation is that there are more point
# than below i.e. there are often more closed brackets than open brackets. This may be becaus
# i.e. the greater than symbol is be used for indentation in reply and forward emails. For an
# exactly these frustrate users, here is an article!
# https://answers.microsoft.com/en-us/outlook_com/forum/oemail-osend/removing-the-greater-tha

# Write the code to generate your visualization here:


sns.scatterplot(data=my_X_train, x='number_of_closed_brackets', y='number_of_open_brackets')
plt.xlabel('Number of Closed Brackets', fontsize=15)
plt.ylabel('Number of Open Brackets', fontsize=15)
plt.title('Plot showing redundancy between the Number of Open and Closed Brackets features', f
```

Out[229]:
```
Text(0.5, 1.0, 'Plot showing redundancy between the Number of Open and Closed Brackets featur
```



0.0.8 Question 9: ROC Curve

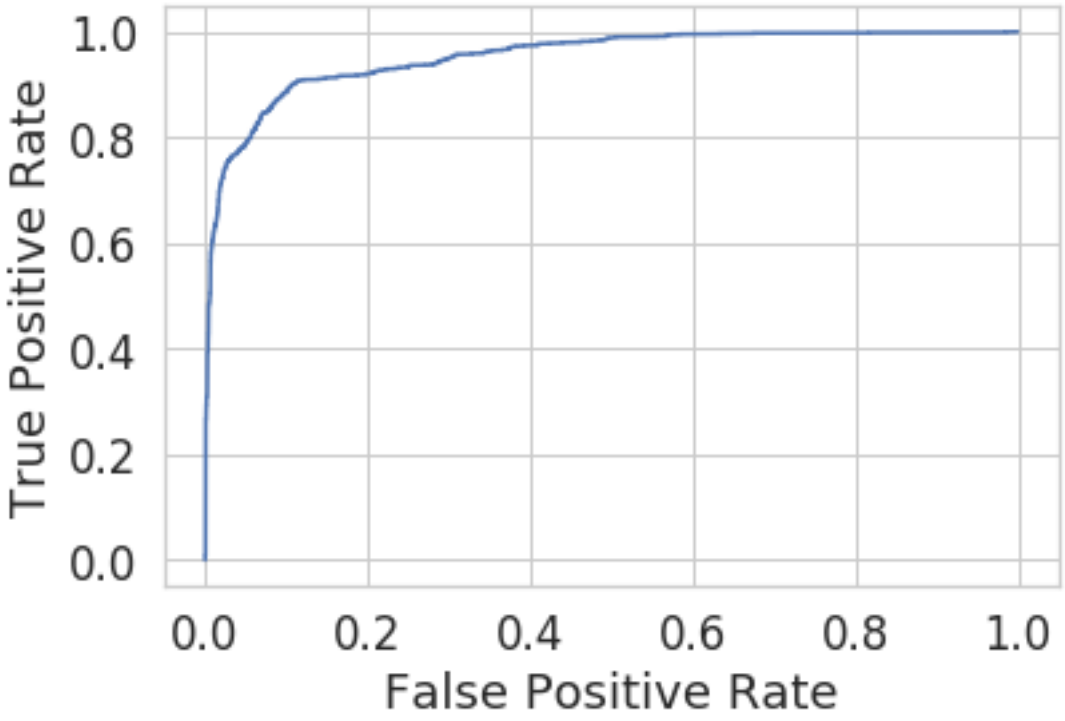
In most cases we won't be able to get 0 false positives and 0 false negatives, so we have to compromise. For example, in the case of cancer screenings, false negatives are comparatively worse than false positives — a false negative means that a patient might not discover that they have cancer until it's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

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

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

In [230]:
```
from sklearn.metrics import roc_curve

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


fpr, tpr, threshold = roc_curve(my_Y_train, my_model.predict_proba(my_X_train_scaled)[:, 1])
plt.plot(fpr, tpr)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

Out[230]:
```
Text(0, 0.5, 'True Positive Rate')
```


True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0