0.0.1 Question 1: Feature/Model Selection Process

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

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

1. First I looked up common phrases in spam emails to get specific patterns that would have a high likeleyhood to be flagged as spam. I then chose likely errors or phrases that could also be seen as spam such as multiple spaces or repeated punctuation. The final method I tried was looking at the emails and subjects directly to see common words in ham emails and in spam emails.
2. The features that worked best were punctuation since excessive use of them tended to correlate with spam emails. Also emails with specific days in them were useful to track as that made them more likely to be ham. What did not work overall was trying to plug in random words since they had to be case sensitive and if they were too specific bareley any emails would contain them making it hard to train and if the words were too vauge it would be too included in all emails.
3. The most surprising search was repeating punctuation as that raised the accuracy by a lot. I was also surprised by how the total number of html links affected the model as I did not believe there was a correlation between links and spam/ham. I was a bit dissapointed in many words that I tried searching for such as "now", "immediatly", and "today" not being that useful since I assumed these words had a rushed tone making them more likely to be a scam.|

Question 2a Generate your visualization in the cell below.

In [12]:
```
import regex as re
```

In [13]:
```
def count_character(char, email):
    pattern = "\\" + char
    return len(re.findall(pattern, email))
def count_exclamation(email):
    return len(re.findall(r"\!", email))
def count_question(email):
    return len(re.findall(r"\?", email))
def count_dollar(email):
    return len(re.findall(r"\$", email))
def count_capital(email):
    return len(re.findall(r"[A-Z]", email))
def count_reply(email):
    return len(re.findall(r"reply", email))
def count_forward(email):
    return len(re.findall(r"forwarded", email))
def count_html(email):
    return len(re.findall(r"html", email))
def count_hashtag(email):
    return len(re.findall(r"#", email))
def count_body(email):
    return len(re.findall(r"body", email))
def count_equal(email):
    return len(re.findall(r"\=", email))
def count_percent(email):
    return len(re.findall(r"\%", email))
def count_numbers(email):
    return len(re.findall(r"[0-9]", email))
def count_special(email):
    return len(re.findall(r"[^\w]", email))
def count_period(email):
    return len(re.findall(r"\.", email))
```

In [14]:
```
def contains_date(email):
    email = email.lower()
    days = ['mon', 'tue', 'wed', 'thu', 'fri', 'sat', 'sun']
    for day in days:
        if day in email:
            return 1
    return 0
```

In [15]:
```
labels = ["!","?", "$", "#", "=", "s"]
email_labels = ["body", "html", "forward", "."]
funcs = [count_exclamation,count_question, count_dollar, count_special]
email_funcs = [count_body, count_html, count_forward, count_period]
```

```
email_words = ["buy", "reply", "click", "once", "sign", "  ", "update", "you", "hr", "%", "ord
subject_words = ["  ", "You"]
```

In [16]:
```
#training set

copy_data = train.copy()
train_email = words_in_texts(email_words, train['email'])
train_subject = words_in_texts(subject_words, train['subject'])
train_email_df = pd.DataFrame(train_email)
for i in range(len(email_words)):
    train_email_df = train_email_df.rename({i:email_words[i]}, axis = 1)
train_subject_df = pd.DataFrame(train_subject)
train_subject_df = train_subject_df.rename({0:subject_words[0], 1:subject_words[1]}, axis = 1)
for i in range(len(funcs)):
    train_email_df["num of " + labels[i]] = copy_data["email"].apply(funcs[i])
for i in range(len(funcs)):
    train_subject_df[labels[i] + " in subject"] = copy_data["subject"].apply(funcs[i])
for i in range(len(email_funcs)):
    train_email_df[email_labels[i]] = copy_data["email"].apply(email_funcs[i])
train_subject_df["numbers"] = copy_data["email"].apply(count_numbers)
train_subject_df["capital"] = copy_data["subject"].apply(count_capital)
train_df = pd.concat([train_email_df, train_subject_df], axis = 1)
for column in train_df.columns:
    train_df[column] = train_df[column]  / train_df[column].abs().max()
```

In [17]:
```
x_train = train_df.to_numpy()
y_train = train['spam'].to_numpy()
model2 = LogisticRegression(solver = 'liblinear', max_iter=10000, penalty = "l1")
model2.fit(x_train, y_train)

training_accuracy = model2.score(x_train, y_train)
print("Training Accuracy: ", training_accuracy)
```

```
Training Accuracy:  0.9251963263676295
```

In [18]:
```
#Val Set


val_copy = val.copy().replace({'':'z'}).reset_index().drop("index", axis = 1)
val_email = words_in_texts(email_words, val['email'])
val_subject = words_in_texts(subject_words, val['subject'])
val_email_df = pd.DataFrame(val_email)
val_subject_df = pd.DataFrame(val_subject)
val_subject_df = val_subject_df.rename({0:subject_words[0], 1:subject_words[1]}, axis = 1)
for i in range(len(funcs)):
    val_email_df[labels[i]] = val_copy["email"].apply(funcs[i])
for i in range(len(funcs)):
    val_subject_df[labels[i]+'2'] = val_copy["subject"].apply(funcs[i])
```

```
        for i in range(len(email_funcs)):
            val_email_df[email_labels[i]] = val_copy["email"].apply(email_funcs[i])
        val_subject_df["c"] = val_copy["email"].apply(count_numbers)
        val_subject_df["capital"] = val_copy["subject"].apply(count_capital)
        val_df = pd.concat([val_email_df, val_subject_df], axis = 1)
        for column in val_df.columns:
            val_df[column] = val_df[column]  / val_df[column].abs().max()
```

In [19]:
```
valx = val_df.to_numpy()
valy = val['spam'].to_numpy()
val_pred = model2.predict(valx)
val_accuracy = sum(val_pred == valy) / len(valy)
print("Val Accuracy: ", val_accuracy)
```

```
Val Accuracy:  0.9101796407185628
```

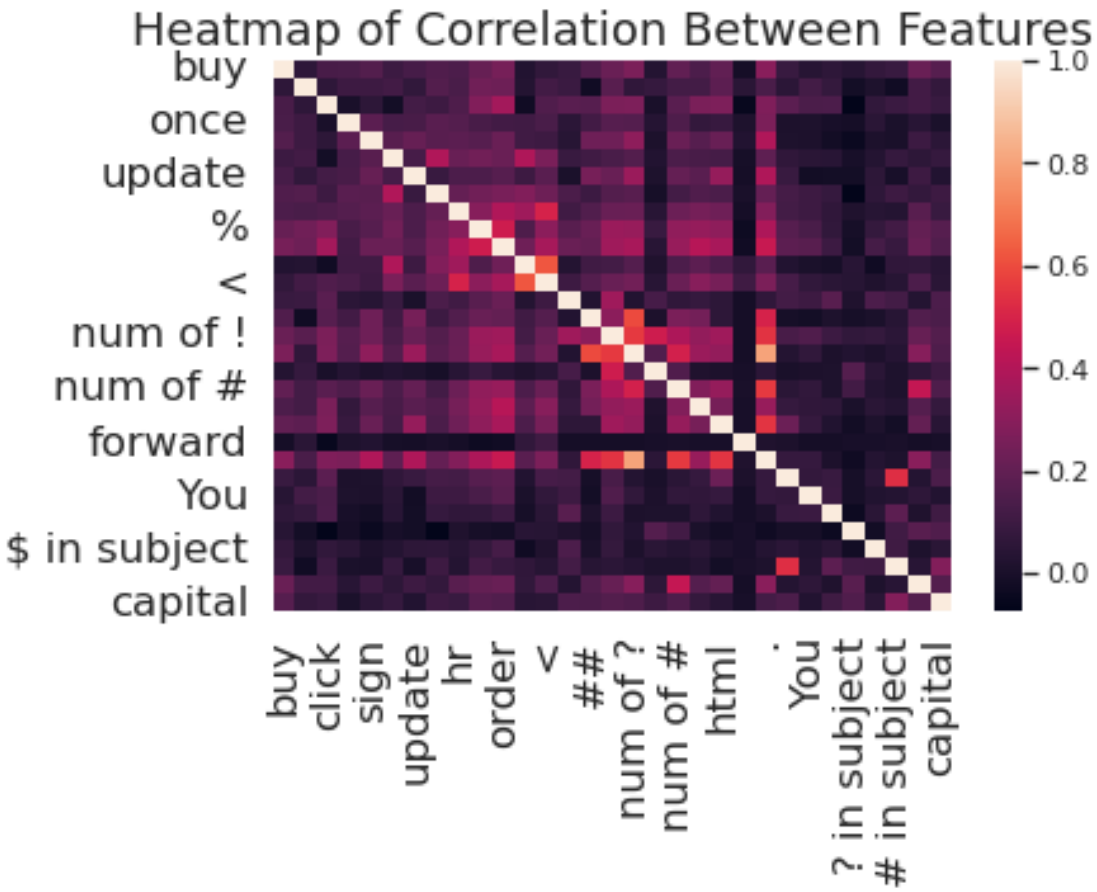
In [20]:
```
#Test data


test_copy = test.copy().fillna('')
test_email = words_in_texts(email_words, test_copy['email'])
test_subject = words_in_texts(subject_words, test_copy['subject'])
test_email_df = pd.DataFrame(test_email)
test_subject_df = pd.DataFrame(test_subject)
for i in range(len(funcs)):
    test_email_df["num of " + labels[i]] = test_copy["email"].apply(funcs[i])
for i in range(len(funcs)):
    test_subject_df[labels[i] + " in subject"] = test_copy["email"].apply(funcs[i])
for i in range(len(email_funcs)):
    test_email_df[email_labels[i]] = test_copy["email"].apply(email_funcs[i])
test_subject_df["numbers"] = test_copy["email"].apply(count_numbers)
test_subject_df["capital"] = test_copy["subject"].apply(count_capital)
test_df = pd.concat([test_email_df, test_subject_df], axis = 1)
for column in test_df.columns:
    test_df[column] = test_df[column]  / test_df[column].abs().max()
```

In [21]:
```
predict_train_df = train_df.copy()
predict_train_df["y"] = y_train
corr = train_df.corr()
```

In [22]:
```
plt.title("Heatmap of Correlation Between Features")
sns.set(rc={'figure.figsize':(20.7,8.27)})
sns.heatmap(corr)
```

Out[22]:
```
<AxesSubplot:title={'center':'Heatmap of Correlation Between Features'}>
```


Heatmap of Correlation Between Features
buy
once
update
%
<
num of !
num of #
forward
You
$ in subject
capital
buy
click
sign
update
hr
order
<
##
num of ?
num of #
html
.
You
? in subject
in subject
capital
1.0
0.8
0.6
0.4
0.2
0.0

Question 2b Write your commentary in the cell below.

Following the bottom row we can see what features correlate best with the actual spam and ham emails. The darker or brighter shows that these are good correlation features as darker would indicate spam and bright features would be ham. There also is deccent correlation between punctuations as the usage of them is somewhat correlated.

0.0.2 Question 3: ROC Curve

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

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

The ROC curve shows this trade off for each possible cutoff probability. In the cell below, plot a ROC curve for your final classifier (the one you use to make predictions for Gradescope) on the training data. Refer to Lecture 20 to see how to plot an ROC curve.

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

In [23]:
```
from sklearn.metrics import roc_curve

fpr, tpr, threshold = roc_curve(train["spam"],
                                model2.predict_proba(train_df.to_numpy())[:, 1])
```

In [ ]:
```
import plotly.express as px
fig = px.line(x=fpr, y = tpr, hover_name=threshold)
fig.update_xaxes(title="False Positive Rate")
fig.update_yaxes(title="True Positive Rate")
fig
```