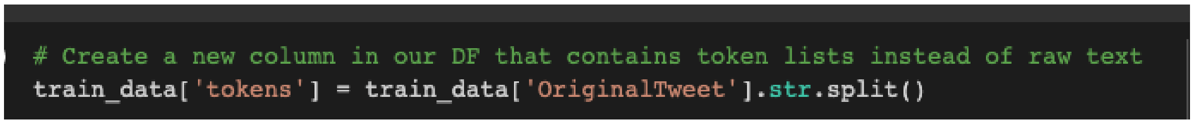
JSC270 Assignment 4

Author: Hongshuo Zhou, Chun Yin Yan
Contribution: All the work was done jointly with equal distribution.

Github link:
https://github.com/TheTonyZhou/JSC270-A4.git

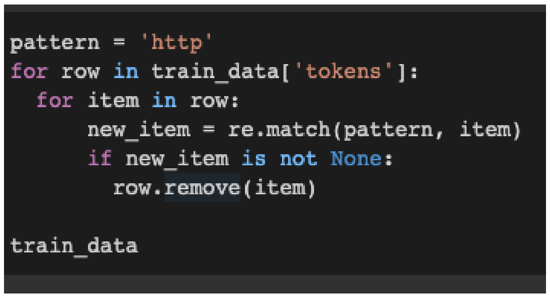
Part 1:
A:
There are 15397 negative observations, 7712 neutral observations, and 18042 positive observations in the training set. Thus, in the training data set, 48.84% are positive observations, 37.42% are negative observations, and 18.72% are neutral observations, which suggests the data is not well balanced.

B:



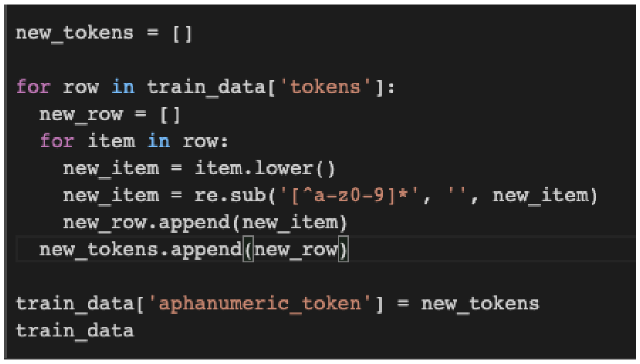
```
# Create a new column in our DF that contains token lists instead of raw text
train_data['tokens'] = train_data['OriginalTweet'].str.split()
```

C:



```
pattern = 'http'
for row in train_data['tokens']:
  for item in row:
      new_item = re.match(pattern, item)
      if new_item is not None:
        row.remove(item)

train_data
```

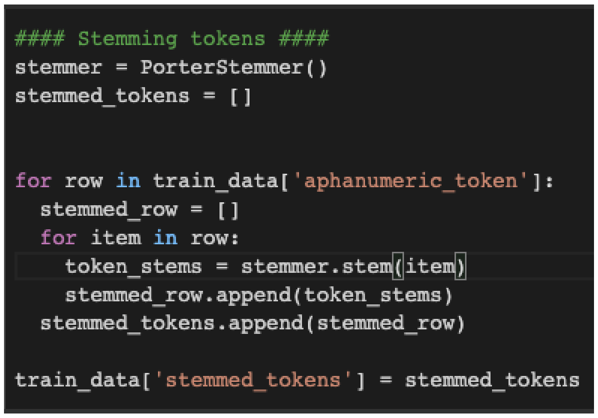
D:



```
new_tokens = []

for row in train_data['tokens']:
  new_row = []
  for item in row:
    new_item = item.lower()
    new_item = re.sub('[^a-z0-9]*', '', new_item)
    new_row.append(new_item)
  new_tokens.append(new_row)

train_data['aphanumeric_token'] = new_tokens
train_data
```

A scenario where we should keep some forms of punctuation is that at the end of a sentence, a question mark or exclamation mark can be helpful because they can suggest the sentiment of the text. So, it should be better to keep the punctuation in this case.

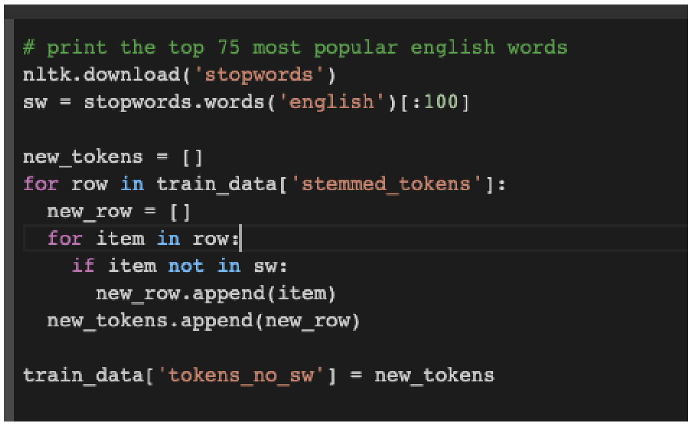
E:



```
#### Stemming tokens ####
stemmer = PorterStemmer()
stemmed_tokens = []


for row in train_data['aphanumeric_token']:
  stemmed_row = []
  for item in row:
    token_stems = stemmer.stem(item)
    stemmed_row.append(token_stems)
  stemmed_tokens.append(stemmed_row)

train_data['stemmed_tokens'] = stemmed_tokens
```

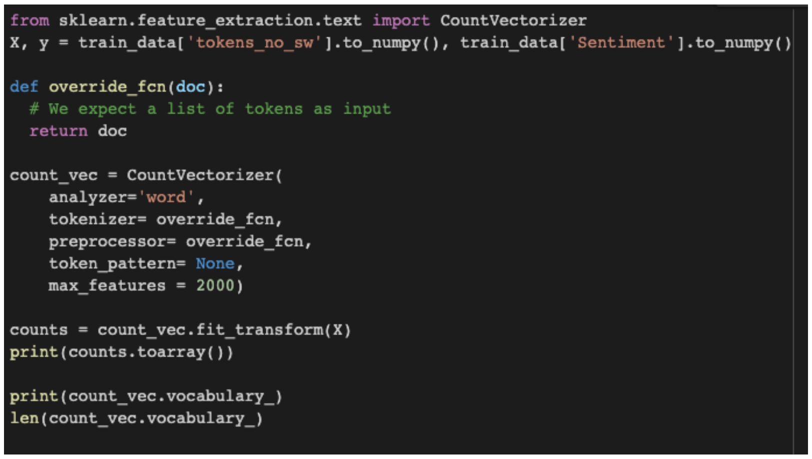
The stemmer we used is PorterStemmer.

F:



```
# print the top 75 most popular english words
nltk.download('stopwords')
sw = stopwords.words('english')[:100]

new_tokens = []
for row in train_data['stemmed_tokens']:
  new_row = []
  for item in row:
    if item not in sw:
      new_row.append(item)
  new_tokens.append(new_row)

train_data['tokens_no_sw'] = new_tokens
```

G:



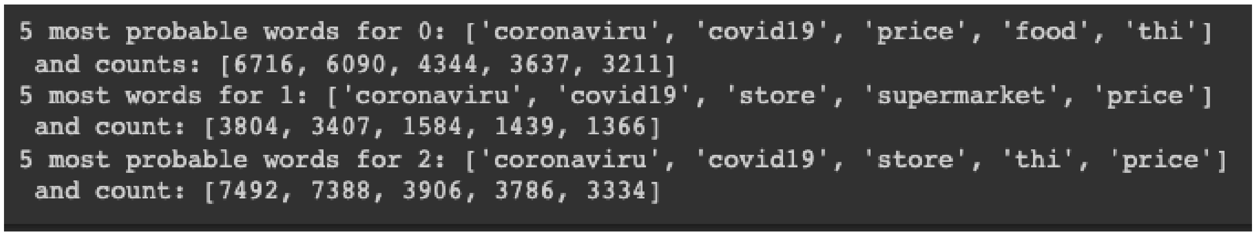
```python
from sklearn.feature_extraction.text import CountVectorizer
X, y = train_data['tokens_no_sw'].to_numpy(), train_data['Sentiment'].to_numpy()

def override_fcn(doc):
  # We expect a list of tokens as input
  return doc

count_vec = CountVectorizer(
    analyzer='word',
    tokenizer= override_fcn,
    preprocessor= override_fcn,
    token_pattern= None,
    max_features = 2000)

counts = count_vec.fit_transform(X)
print(counts.toarray())

print(count_vec.vocabulary_)
len(count_vec.vocabulary_)
```

The total length of our vocabulary is 52572, but we only used the top 2000 to save space and running time.

H:

The code is too long so please refer to the google colab. The training accuracy of this model is 0.71.20 and testing accuracy is 0.3873.

Top frequency words in each class:



```
5 most probable words for 0: ['coronaviru', 'covid19', 'price', 'food', 'thi']
 and counts: [6716, 6090, 4344, 3637, 3211]
5 most words for 1: ['coronaviru', 'covid19', 'store', 'supermarket', 'price']
 and count: [3804, 3407, 1584, 1439, 1366]
5 most probable words for 2: ['coronaviru', 'covid19', 'store', 'thi', 'price']
 and count: [7492, 7388, 3906, 3786, 3334]
```

I:

I think we should not fit an ROC curve in this scenario because we have three classes. However, the ROC curve can only be fitted between three classes. If one really wants to fit a ROC curve, they can use the approach like one class versus rest classes, but it is not very informative.

J:

The training accuracy is 0.6920 and the testing accuracy is 0.4104. Compared to count vectors, the training accuracy decreases but the testing accuracy increases. However, the overall changes are relatively small so the performance of the two are roughly the same.

K:

The training accuracy is 0.6889 and the testing accuracy is 0.4120. Compared to stemming, the training accuracy decreases but the testing accuracy increases. However, the overall changes are relatively small so the performance of the two are roughly the same.

Bonus:
Naive Bayes models are generative because they are based on the joint probability of the classes to build our model, which is a key feature of generative models.