CS 505 Homework 02: Data Wrangling and BOW

Due Thursday 9/21 at midnight (1 minute after 11:59 pm) in Gradescope (with a grace period of 6 hours)

You may submit the homework up to 24 hours late (with the same grace period) for a penalty of 10%.

All homeworks will be scored with a maximum of 100 points; point values are given for individual problems, and if parts of problems do not have point values given, they will be counted equally toward the total for that problem.

Note: I strongly recommend you work in **Google Colab** (the free version) to complete homeworks in this class; in addition to (probably) being faster than your laptop, all the necessary libraries will already be available to you, and you don't have to hassle with `conda`, `pip`, etc. and resolving problems when the install doesn't work. But it is up to you! You should go through the necessary tutorials listed on the web site concerning Colab and storing files on a Google Drive. And of course, Dr. Google is always ready to help you resolve your problems.

I will post a "walk-through" video ASAP on my Youtube Channel (https://www.youtube.com/channel/UCfSqNB0yh99yuG4p4nzjPOA).

Submission Instructions

You must complete the homework by editing **this notebook** and submitting the following two files in Gradescope by the due date and time:

- A file `HW02.ipynb` (be sure to select `Kernel -> Restart and Run All` before you submit, to make sure everything works); and
- A file `HW02.pdf` created from the previous.

  For best results obtaining a clean PDF file on the Mac, select `File -> Print Review` from the Jupyter window, then choose `File-> Print` in your browser and then `Save as PDF`. Something similar should be possible on a Windows machine -- just make sure it is readable and no cell contents have been cut off. Make it easy to grade!

The date and time of your submission is the last file you submitted, so if your IPYNB file is submitted on time, but your PDF is late, then your submission is late.

Collaborators (5 pts)

Describe briefly but precisely

1. Any persons you discussed this homework with and the nature of the discussion;
2. Any online resources you consulted and what information you got from those resources; and
3. Any AI agents (such as chatGPT or CoPilot) or other applications you used to complete the homework, and the nature of the help you received.

A few brief sentences is all that I am looking for here.

```
ChatGPT: I requested GPT to debug multiple regex I've written that fails to match desired
patterns(for instance, the multi-layered filtering of conditions of punctuation removal in
1.D), to compare different regex which I believed to mean the same but actually produce di
fferent results, and to clarify certain concepts such as BOW model.

Websites I consulted: https://docs.python.org/3/library/re.html, Regex101, Python document
ation for list-operation functions like zip(), https://docs.python.org/3/library/collectio
ns.html
```

Overview

We are going to practice converting raw (string form) text into a useful data set using the script of *Pirates of the Caribbean: The Curse of the Black Pearl* (2003), the first in a series of PotC movies starring Johnny Depp. The script is part of the `webtext` corpus in NLTK.

Under the assumption that we wish to perform an analysis of the words spoken by the characters in the movie, we will convert the text in a series of steps from a raw string of ASCII characters into a dictionary holding a sparse BOW model of the words spoken by two of the main characters, Elizabeth Swann and Jack Sparrow. These dictionaries could be the data set for a classification task, or for creating a vector-space model for each character, which we will study later in the course. For this assignment, you will clean up, normalize and tokenize the text, create the dictionaries, and then simply print out the most common words spoken by the two characters.

Text normalization was covered in lecture on Tuesday 9/12 and the BOW model on Thursday 9/14. Before beginning the assignment, you should consult the following for information on using the Python regular expression library:

https://docs.python.org/3/library/re.html (https://docs.python.org/3/library/re.html)

Also useful is

https://docs.python.org/3/howto/regex.html (https://docs.python.org/3/howto/regex.html)

After reviewing the basic principles of regular expressions (which I will review in my walk-through video), read about the following useful functions:

```
result = re.split(...)

result = re.sub(...)
```

You may ONLY use standard functions from the `re` library for this homework, and you must perform your normalization starting with the string form of the script assigned below to the variable `pirates_txt` . You may NOT use indices of the string to perform your modifications (e.g., deleting the first line by counting how many characters to remove).

The point here is to use regular expressions to do the text wrangling. Don't worry, we shall use the `SpaCy` library later on the course to normalize text for a classification problem set.

In [142]:
```
import numpy as np
import nltk
import re

# The first time you will need to download the corpus:

nltk.download('webtext')

pirates_txt = nltk.corpus.webtext.raw('pirates.txt')
```

```
[nltk_data] Downloading package webtext to /Users/yfsong/nltk_data...
[nltk_data]   Package webtext is already up-to-date!
```

In [143]:
```
# raw string form
pirates_txt[:1000]
```

Out[143]:
```
"PIRATES OF THE CARRIBEAN: DEAD MAN'S CHEST, by Ted Elliott & Terry Rossio\n[view looking straig
ht down at rolling swells, sound of wind and thunder, then a low heartbeat]\nScene: PORT ROYAL\n
[teacups on a table in the rain]\n[sheet music on music stands in the rain]\n[bouquet of white o
rchids, Elizabeth sitting in the rain holding the bouquet]\n[men rowing, men on horseback, to th
e sound of thunder]\n[EITC logo on flag blowing in the wind]\n[many rowboats are entering the ha
rbor]\n[Elizabeth sitting alone, at a distance]\n[marines running, kick a door in] \n[a mule is
seen on the left in the barn where the marines enter]\n[Liz looking over her shoulder]\n[Elizabe
th drops her bouquet]\n[Will is in manacles, being escorted by red coats]\nELIZABETH SWANN: Wil
l...!\n[Elizabeth runs to Will]\nELIZABETH SWANN: Why is this happening? \nWILL TURNER: I don't
know. You look beautiful.\nELIZABETH SWANN: I think it's bad luck for the groom to see the bride
before the wedding.\n[marines cross their long axes to bar Go"
```

In [144]:
```
# printing it shows the formatting
print(pirates_txt[:1000])
```

```
PIRATES OF THE CARRIBEAN: DEAD MAN'S CHEST, by Ted Elliott & Terry Rossio
[view looking straight down at rolling swells, sound of wind and thunder, then a low heartbeat]
Scene: PORT ROYAL
[teacups on a table in the rain]
[sheet music on music stands in the rain]
[bouquet of white orchids, Elizabeth sitting in the rain holding the bouquet]
[men rowing, men on horseback, to the sound of thunder]
[EITC logo on flag blowing in the wind]
[many rowboats are entering the harbor]
[Elizabeth sitting alone, at a distance]
[marines running, kick a door in]
[a mule is seen on the left in the barn where the marines enter]
[Liz looking over her shoulder]
[Elizabeth drops her bouquet]
[Will is in manacles, being escorted by red coats]
ELIZABETH SWANN: Will...!
[Elizabeth runs to Will]
ELIZABETH SWANN: Why is this happening?
WILL TURNER: I don't know. You look beautiful.
ELIZABETH SWANN: I think it's bad luck for the groom to see the bride before the wedding.
[marines cross their long axes to bar Go
```

Problem One (35 points): Cleaning up the lines

The first task is to clean up the text so that at the conclusion of this problem, you will have a text with punctuation and extraneous characters removed, and each line consisting of a character's name (human or otherwise), a colon, and a sequence of words, ending in a newline.
(You will keep this as a single string until Problem 3, but we will refer to the "lines" spoken by each character -- also, I hope it will not be confusing to speak of (ASCII) characters and characters played by the actors in the script!)

Part 1.A (5 pts)

1. Convert the string into all lower-case letters.
2. Remove the first line which gives the title and authors. Print out the first 200 characters to show that you have done this.

Hint: Cut everything before the first '\n', using the 'beginning of string' special character `^` in the regular expression.

In [145]:
```
# Your code here
# convert the whole script to lower case
prts_lower = pirates_txt.lower()
# remove the metadata before (and including) the 1st newline character
prts_meta_removed = re.sub(r'^(.*)\n', '', prts_lower)
print(prts_meta_removed[:200])
```

```
[view looking straight down at rolling swells, sound of wind and thunder, then a low heartbeat]
scene: port royal
[teacups on a table in the rain]
[sheet music on music stands in the rain]
[bouquet of
```

Part 1.B (5 pts)

Cut out all the stage directions that are given in square brackets, including the newlines on those lines. Print out the first 200 characters as proof.

In [146]:
```
# Your code here
prts_nostage = re.sub(r'\[(.*)(\])?(\s)?\n', '', prts_meta_removed)
print(prts_nostage[:200])
```

```
scene: port royal
elizabeth swann: will...!
elizabeth swann: why is this happening?
will turner: i don't know. you look beautiful.
elizabeth swann: i think it's bad luck for the groom to see the brid
```

Part 1.C (5 pts)

Cut out the lines where the 'scene' is specified. Again, print out the first 200 characters.

In [147]:
```
# Your code here
prts_noscene = re.sub(r'(scene)(.*)\n', '', prts_nostage)  # review capturing group
print(prts_noscene[:200])
```

```
elizabeth swann: will...!
elizabeth swann: why is this happening?
will turner: i don't know. you look beautiful.
elizabeth swann: i think it's bad luck for the groom to see the bride before the weddi
```

Part 1.D (20 pts)

Now, we still have a lot of punctuation and some miscellaneous odd things occurring in this text, and we need to do further cleaning. But you will have to figure this out for yourself!

The main thing to do is to remove punctuation and anything that does not contribute to the goal of making a BOW model for our two characters.

But you can't just remove all non-word characters! Make sure you take account of the following:

1. You need to keep the character's names at the beginning of the line, so

do not remove the colon after the name (note that these always occur at the beginning of a line, i.e., at the very beginning or immediately after the newline from the previous line).

2. In the next problem we will normalize the words, so **we DON'T want to change anything that might be a word,** such as,

   `charges` `don't` `it's` `'er` `ah-ha` `ha-ha-ha-ha-ha` `stealin'`

3. After observing the caveats above, **remove all punctuation.**

4. There are some places where apparently the transcriber was not sure what the word was and gave alternatives:

   `weren't/wasn't`

   and some places where it is not clear what is intended:

   `oy /quick`

Just treat `/` like ordinary punctuation and replace it by a blank.

5. Finally, there are miscellanous weird things in the text, such as

   `###` `?:`

   (and possibly others) which **should be removed.**

How to proceed: To explore the data, print out the text after the modifications in Parts 1.A -- 1.D:

```
print(pirates_txt_01)
```

and think about what needs to be removed, paying careful attention to the comments above. (You could use the `Find` function in your browser to flip through various possibilities.)

After examining the text, **comment out the `print(pirates_txt_01)` so that we don't have to look at it!** This was just for exploration!

The result of your cleaning in this part should be assigned to `pirates_txt_01`.

Hint: At this stage, it might be better to **replace substrings with blanks** instead of deleting them (replacing with the empty string)

Part 1.D.1 (5 pts)

Write a short description here of what you removed, giving your reasoning. You must account for at least what is listed above, but you may find other things you want to change.

1. **for '/' between two characters, replace it with a space:**
   - replaces '/' with a space, but when '/' is between word characters or spaces.
   - This step is implemented first so that the remaining '/' can be removed without further scrutiny.
2. **don't change anything that has the following features inside a word:**
   - The patterns to reserve as I looked through the first 25000 characters:
     - contractions in the forms of "'t", "'s", "'re", "'n", "'ve", "'d", "'er", "o'", "'m", "'ll", "'til", "'im", "t'", "'em", "n'", "'b", "d'", "l'", "'n'", "i'"
     - crucial hyphens inside words, e.g. "ah-ha", "ha-ha-ha-ha-ha"
   - These are all negative cases that a complete punctuation removal does not want to include. To reduce the complexity of later filtering, I substituted the single-quoted patterns founds and the crucial hyphens by indices. When the large-scale removal of punctuation finalizes, I then substitute the corredponding patterns back.
3. **don't remove the colon after the name**
   - limit the colons after character names (i.e., the first colon in every line)

Also print out some portion of the text to show at least some of the changes you have made.
These are the differences my codes made:

before:
jack sparrow: tia **dalma!**
tia dalma: i always know de wind was **goin'** blow you back to me one day**.**
tia dalma: you**.** you have a touch of**...** destiny about ***you***, william **turner.**
will turner: you know **me?**
tia dalma: you want to know ***me*****.**
jack sparrow: **there'll** be no knowing **here. we've** come for help and **we're** not leaving without **it.**
jack sparrow: i thought ***i*** knew **you.**
tia dalma: not so well as i had **hoped. come.**
jack sparrow: **come.**
tia dalma: **what... service...** may i do **you? hmmm?** you know i demand **payment.**

after:
jack sparrow: tia dalma
tia dalma: i always know de wind was goin' blow you back to me one day
tia dalma: you you have a touch of destiny about you william turner
will turner: you know me
tia dalma: you want to know me
jack sparrow: there'll be no knowing here we've come for help and we're not leaving without it
jack sparrow: i thought i knew you
tia dalma: not so well as i had hoped come
jack sparrow: come
tia dalma: what service may i do you hmmm you know i demand payment

before:
will turner: **move!**
cannibal island: throne

cannibal crowd: **ahhh! fye-fye!**
jack sparrow: well, go **on!** go get **them!** hay **ala!**
cannibal crowd: hay **ala! ala, ala!**
jack sparrow: **no! no no! oy! no no!**
jack sparrow: not **good.**
will turner: cut it **loose!** find a **rock!**
will turner: roll the **cage!**
will turner: lift the **cage! hurry!**
gibbs: come **on, men!** lift it like a **lady's skirt!**
gibbs: this **way, lads!**
cannibal boy: da **litozo!** da **litozo!**
cannibal woman: **a-geev-nee. uh-boogie?**

after:
will turner: move
cannibal island: throne
cannibal crowd: ahhh fye-fye
jack sparrow: well go on go get them hay ala
cannibal crowd: hay ala ala ala
jack sparrow: no no no oy no no
jack sparrow: not good
will turner: cut it loose find a rock
will turner: roll the cage
will turner: lift the cage hurry
gibbs: come on men lift it like a lady's skirt
gibbs: this way lads
cannibal boy: da litozo da litozo
cannibal woman: a-geev-nee uh-boogie

Part 1.D.2 (15 pts)

Write your code in the following cell. The result at the end should be stored in `pirates_txt_01` . Print out the first 2000 characters.

In [148]:
```
# review positive/negative lookbehind/lookahead
# use re.findall() to test the regex

# 1. For "/" between two characters, replace it with a space
prts_3 = re.sub(r'(?<=[\w\s\W])/(?=[\w\s])', ' ', prts_noscene)
```

In [149]:
```
# 2. meaningful hyphens inside a word
hyphen_reserve = r"(?<=\w)-(?=\w)" # matches - in the middle of words
crucial_sngl_hph = [f"sssHYPHEN_PAT{i}sss" for i, _ in enumerate(re.findall(hyphen_reserve, prts_3

# replace the crucial hyphens with their correponding indiced name
for idx in crucial_sngl_hph:
    prts_3 = re.sub(hyphen_reserve, idx, prts_3, count=1)

# 2. meaningful single quotes inside a word
crucial_sngl_qts = [r"'t", r"'s", r"'re", r"'n", r"'ve", r"'d",\
                    r"'e", r"o'", r"'m",r"'l", r"'i", r't'", r"n'", r"'b", r"d'", r"l'", r"i'"]

sngl_qts_patidxs = [f"sssSINGLE_QUOTE_PAT{i}sss" for i, _ in enumerate(crucial_sngl_qts)]

# replace the crucial single-quote patterns with their correponding indiced name
for pat, idx in zip(crucial_sngl_qts, sngl_qts_patidxs):
    prts_3 = re.sub(rf"(?<=[\w\s]){pat}(?=[\w\s])", idx, prts_3)

# 3. don't remove the colon after the name
colon_reserve = r"^.*?:"  # matches the first colon in every line lazily

# remove other punctuations
selected_pucts = r"[^\w\s:]"
pat = rf'({selected_pucts})(?!{colon_reserve})'
pirates_txt_01 = re.sub(pat, '', prts_3)

# replace the crucial hyphens and single-quote patterns back to where they were
for pat, idx in zip(crucial_sngl_qts, sngl_qts_patidxs):
    pirates_txt_01 = pirates_txt_01.replace(idx, pat)
for idx in crucial_sngl_hph:
    pirates_txt_01 = pirates_txt_01.replace(idx, '-', 1)

print(pirates_txt_01[:2000])
```

```
elizabeth swann: will
elizabeth swann: why is this happening
will turner: i don't know you look beautiful
elizabeth swann: i think it's bad luck for the groom to see the bride before the wedding
lord cutler beckett: governor weatherby swann it's been too long
lord cutler beckett: his lord now actually
lord cutler beckett: in fact i do mister mercer the warrant for the arrest of one william turner
lord cutler beckett: oh is it that's annoying my mistake arrest her
elizabeth swann: on what charges
will turner: no
lord cutler beckett: ah-ha here's the one for william turner and i have another one for a mister
james norrington is he present
elizabeth swann: what are the charges
lord cutler beckett: i don't believe that's the answer to the question i asked
will turner: lord beckett in the category of questions not answered
elizabeth swann: we are under the jurisdiction of the king's governor of port royal and you will
tell us what we are charged with
lord cutler beckett: for which the punishment regrettably is also death perhaps you remember a c
ertain pirate named jack sparrow
elizabeth swann: captain jack sparrow
lord cutler beckett: captain jack sparrow yes i thought you might
gibbs: fifteen men on a dead man's chest yo ho ho and a bottle of rum drink and the devil had do
ne for the rest yo ho ho and a bottle of rum ha-ha-ha-ha-ha
jack sparrow: sorry mate
jack sparrow: mind if we make a little side trip i didn't think so
gibbs: not quite according to plan
jack sparrow: complications arose ensued were overcome
gibbs: you got what you went in for then
jack sparrow: mm-hmm
gibbs: captain i think the crew meaning me as well were expecting something a bit more shiny wha
t with the isla de muerta going all pear shaped reclaimed by the sea and the treasure with it
leech: and the royal navy chasing us all around the atlantic
marty: and the hurricane aye
crew: aye aye
gibbs: all in all it's seems some time since we did a speck of honest pirating
jack sparrow: shiny
gibbs: ay
```

Problem Two (30 points): Normalizing, Stemming, and Lemmatization

In this problem we are going to do **some** normalizing of the words, first of all to normalize certain words with apostrophes, and then performing stemming and lemmatization. We are not intended to be absolutely thorough here, just to try a few obvious possibilities.

Part 2.A Normalizing (15 pts)

There are several ways that apostophes (single quotes) are used to compress two words into one (to give a better sense for how they are pronounced, *especially by pirates*):

```
didn't  =  did not    we've  = we have       there'd  = there would
```

Your task: Find as many examples of these as you can, and replace the compressed word with the two-word phrase it represents.

Note: **Do NOT process any words with `'s`,** as these will be done in the next part.

Do NOT simply compile a list of specific examples, but look for general patterns for substitution, for example:

```
n't =>  _not       've =>  _have       # where _ represents a blank
```

Simply find as many examples which seem to have a general rule, and perform those substitutions, putting the result in `pirates_txt_02`.

Finally, print out the first 2000 characters.

In [150]:
```
# Your code here
# chatgpt: help me debug why re.sub() not working after making the raw string pattern as dict keys

# make a list of tuples specifying the normalizations
to_replace_lst = [
    (r"won't", r"will not"),
    (r"n't", r" not"),
    (r"'til", r"until"),
    (r"'re", r" are"),
    (r"(?<=\s)('n)(?=\s)", r"in"),
    (r"'n'", r" and"),
    (r"'ve", r" have"),
    (r"'d", r" would"),
    (r"(?<=\s)('er)(?=\s)", r"her"),  # if "'er" is surrounded by spaces
    (r"'m", r" am"),
    (r"'ll", r" will"),
    (r"'im", r"him"),
    (r"'em", r"them"),
    (r"n'(?!\w)", r"ng"),  # e.g. matched "goin'" but not "don't"
    (r"'b", r"about"),
    (r"d'", r"do "),
    (r"cap'n", r"captain"),  # special case 1: captain
    (r"be'er", r"better"),  # special case 2: better
    (r"wa'er", r"water"),  # special case 3: water
    (r"li'erally", r"it"),  # special case 4: literally
    (r"'course", r"of course"),  # special case 5: of course
    (r"o'", r"of"),  # special case 6: of
    (r": 's", r": it's"),  # special case 7: it's
    (r"'ello", r"hello"),  # special case 8: hello
    (r"'cause", r"because"),  # special case 9: because
    (r"wort'", r"worth"),  # special case 10: worth
    (r"t'inking", r"thinking"),  # special case 11: thinking
    (r"wit'", r"with"),  # special case 12: with
    (r"eart'", r"earth"),  # special case 13: earth
    (r"jones'", r"jones 's"),  # special case 14: jones 's
    (r"'e's", r"he is"),  # special case 15: he is
    (r"'twixt", r"betwixt")  # special case 16: betwixt
]

# create the normalization dict by recompiling each string to make it remain raw string
# to_replace = {re.compile(pat): rep for pat, rep in to_replace_lst}
pirates_txt_02 = pirates_txt_01
for (pat, rep) in to_replace_lst:
    pattern = re.compile(pat)
    pirates_txt_02 = re.sub(pattern, rep, pirates_txt_02)

print(pirates_txt_02[:2000])
```

```
elizabeth swann: will
elizabeth swann: why is this happening
will turner: i do not know you look beautiful
elizabeth swann: i think it's bad luck for the groom to see the bride before the wedding
lord cutler beckett: governor weatherby swann it's been too long
lord cutler beckett: his lord now actually
lord cutler beckett: in fact i do mister mercer the warrant for the arrest of one william turner
lord cutler beckett: oh is it that's annoying my mistake arrest her
elizabeth swann: on what charges
will turner: no
lord cutler beckett: ah-ha here's the one for william turner and i have another one for a mister
james norrington is he present
elizabeth swann: what are the charges
lord cutler beckett: i do not believe that's the answer to the question i asked
will turner: lord beckett in the category of questions not answered
elizabeth swann: we are under the jurisdiction of the king's governor of port royal and you will
tell us what we are charged with
lord cutler beckett: for which the punishment regrettably is also death perhaps you remember a c
ertain pirate named jack sparrow
elizabeth swann: captain jack sparrow
lord cutler beckett: captain jack sparrow yes i thought you might
gibbs: fifteen men on a dead man's chest yo ho ho and a bottle of rum drink and the devil had do
ne for the rest yo ho ho and a bottle of rum ha-ha-ha-ha-ha
jack sparrow: sorry mate
jack sparrow: mind if we make a little side trip i did not think so
gibbs: not quite according to plan
jack sparrow: complications arose ensued were overcome
gibbs: you got what you went in for then
jack sparrow: mm-hmm
gibbs: captain i think the crew meaning me as well were expecting something a bit more shiny wha
t with the isla de muerta going all pear shaped reclaimed by the sea and the treasure with it
leech: and the royal navy chasing us all around the atlantic
marty: and the hurricane aye
crew: aye aye
gibbs: all in all it's seems some time since we did a speck of honest pirating
jack sparrow: shiny
gibbs:
```

Part 2.B Stemming and Lemmatization (15 pts)

Stemming

There are multiple occurrence of the suffix `'s` in the text, some standing for a two word phrase:

```
he's = he is       it's = it is       what's = what is     here's = here is
```

and some being possessives:

```
jack's     man's       hangman's
```

In the first case, the word `is` is very common, and would be removed later when we remove "stop words"; in the second, we will assume there is little difference in the BOW model between a noun and its possessive. So we will remove the `'s` from all words.

Lemmatization

There are eight "official' different forms of the verb 'to be', all of which occur in the text. These must be replaced by the lemma 'be'. (These eight forms do not include modal expressions such as 'will be' or 'would be'.)

Your tasks:

1. Stem these words by removing all instances of `'s`.
2. Lemmatize all the 8 forms of the verb 'to be' by replacing them by their stem 'be'. Be sure to ONLY replace separate words, not substrings of other words, i.e., don't change 'mistake' to 'mbetake'!
3. Put the result in `pirates_txt_02` and print out the first 2000 characters.

In [151]:
```
# Your code here
# remove all 's if it's preceded by a word char and followed by a space
prts_tobe_remove = re.sub(r"(?<=\w)'s(?=[\s\w])", ' be', pirates_txt_02)

# lemmatize all 8 forms of 'to be': be, am, is, are, was, were, being, been
# /b denotes substrings before which is a word char and after which is a non-word char
pirates_txt_02 = re.sub(r"(?<=\s)(be|am|is|are|was|were|being|been)(?=\s)", 'be', prts_tobe_remove

print(pirates_txt_02[:2000])
```

```
elizabeth swann: will
elizabeth swann: why be this happening
will turner: i do not know you look beautiful
elizabeth swann: i think it be bad luck for the groom to see the bride before the wedding
lord cutler beckett: governor weatherby swann it be be too long
lord cutler beckett: his lord now actually
lord cutler beckett: in fact i do mister mercer the warrant for the arrest of one william turner
lord cutler beckett: oh be it that be annoying my mistake arrest her
elizabeth swann: on what charges
will turner: no
lord cutler beckett: ah-ha here be the one for william turner and i have another one for a miste
r james norrington be he present
elizabeth swann: what be the charges
lord cutler beckett: i do not believe that be the answer to the question i asked
will turner: lord beckett in the category of questions not answered
elizabeth swann: we be under the jurisdiction of the king be governor of port royal and you will
tell us what we be charged with
lord cutler beckett: for which the punishment regrettably be also death perhaps you remember a c
ertain pirate named jack sparrow
elizabeth swann: captain jack sparrow
lord cutler beckett: captain jack sparrow yes i thought you might
gibbs: fifteen men on a dead man be chest yo ho ho and a bottle of rum drink and the devil had d
one for the rest yo ho ho and a bottle of rum ha-ha-ha-ha-ha
jack sparrow: sorry mate
jack sparrow: mind if we make a little side trip i did not think so
gibbs: not quite according to plan
jack sparrow: complications arose ensued be overcome
gibbs: you got what you went in for then
jack sparrow: mm-hmm
gibbs: captain i think the crew meaning me as well be expecting something a bit more shiny what
with the isla de muerta going all pear shaped reclaimed by the sea and the treasure with it
leech: and the royal navy chasing us all around the atlantic
marty: and the hurricane aye
crew: aye aye
gibbs: all in all it be seems some time since we did a speck of honest pirating
jack sparrow: shiny
gibbs:
```

Problem Three (30 points): Removing Stop Words, Tokenizing, and Creating the BOW Models

3.A Removing Stop Words (10 pts)

"Stop words" are common words which do not give much information about a text, since they occur in almost all texts. There is a standard set of such words which can be accessed through NLTK (notice that these include some with apostrophes, which we will have already removed):

In [152]:
```
import nltk
from nltk.corpus import stopwords

nltk.download('stopwords')
print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/yfsong/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

For the first part of this problem, you must **remove all stop words from the text,** and store the result in `pirates_txt_03`. However, since `will` is the name of a character in the script, **do NOT remove the stopword `will`** ! Make SURE that you only remove words, and not substrings of larger words, e.g., do not remove all occurrences of the character `i` from the text just because the word `i` is a stop word! Replace stop words with single blanks to preserve the word boundaries.

Put your code in the next cell and print out the first 2000 characters.

In [153]:
```
# Your code here
# Replace stop words with single blanks to preserve the word boundaries
# don't remove 'will'
# remove 'i' if 'i' is NOT a part of a longer word
stopwrds_all = stopwords.words('english')
stopwrds_remove = ['will', 'i']
stopwrds = list(filter(lambda x: x not in stopwrds_remove, stopwrds_all))
stopwrds += [r"(?<=[\s\W])i(?=\s)"]

pirates_txt_03 = pirates_txt_02
for w in stopwrds:
    pirates_txt_03 = re.sub(rf"(?<=[\s]){w}(?=\s)", ' ', pirates_txt_03)

print(pirates_txt_03[:2000])
```

```
elizabeth swann: will
elizabeth swann:       happening
will turner:       know   look beautiful
elizabeth swann:   think     bad luck     groom   see   bride     wedding
lord cutler beckett: governor weatherby swann         long
lord cutler beckett:   lord   actually
lord cutler beckett:   fact     mister mercer   warrant     arrest   one william turner
lord cutler beckett: oh         annoying   mistake arrest
elizabeth swann:     charges
will turner:
lord cutler beckett: ah-ha       one   william turner       another one     mister james norring
ton     present
elizabeth swann:       charges
lord cutler beckett:       believe       answer     question   asked
will turner: lord beckett     category   questions   answered
elizabeth swann:         jurisdiction     king   governor   port royal     will tell us       ch
arged
lord cutler beckett:       punishment regrettably   also death perhaps   remember   certain pira
te named jack sparrow
elizabeth swann: captain jack sparrow
lord cutler beckett: captain jack sparrow yes   thought   might
gibbs: fifteen men     dead man   chest yo ho ho     bottle   rum drink     devil   done     res
t yo ho ho     bottle   rum ha-ha-ha-ha-ha
jack sparrow: sorry mate
jack sparrow: mind     make   little side trip       think
gibbs:   quite according   plan
jack sparrow: complications arose ensued   overcome
gibbs:   got     went
jack sparrow: mm-hmm
gibbs: captain   think   crew meaning     well   expecting something   bit   shiny       isla de
muerta going   pear shaped reclaimed     sea     treasure
leech:     royal navy chasing us   around   atlantic
marty:     hurricane aye
crew: aye aye
gibbs:           seems   time since       speck   honest pirating
jack sparrow: shiny
gibbs: aye shiny
jack sparrow:             feeling   perhaps dear old jack     serving   best interests   captain
cotton   parrot: awk walk   plank
jack sparrow:       bird say
leech:     blame   bird show us         piece   cloth
jack sp
```

3.B Tokenizing and Creating the BOW Dictionary (20 pts)

What we wish to do is to create a BOW model with a dictionary for two characters in the script, `elizabeth swann` and `jack sparrow` .

Part 3.B.1 (2 pts)

Using `split(...)` , split the text on the newlines `\n` to get a list of each line as a string. Print out the first 10 lines.

In [154]:
```
# Your code here
prts_split = re.split(r"\n", pirates_txt_03)
print(prts_split[:10])
```

```
['elizabeth swann: will', 'elizabeth swann:       happening ', 'will turner:       know   look b
eautiful', 'elizabeth swann:   think     bad luck     groom   see   bride     wedding', 'lord cu
tler beckett: governor weatherby swann         long', 'lord cutler beckett:   lord   actually',
'lord cutler beckett:   fact     mister mercer   warrant     arrest   one william turner', 'lord
cutler beckett: oh         annoying   mistake arrest  ', 'elizabeth swann:     charges', 'will t
urner:  ']
```

Part 3.B.2 (18 pts)

Create a dictionary to hold the BOW models for these two characters, each being a `defaultdict` with a default value of 0 (this is a representation of the sparce matrix representing the BOW for the character).

Then go through the lines and calculate the frequency of each word spoken by that character. Print out the 20 most common words spoken by each character and the number of times spoken.

Hint: Scan through the lines created in 3.B.1, and just check if the line contains that character's name. Hint: you can use `in` to check if a substring occurs in a string:

```
'wayne:' in 'wayne snyder: hi there folks!'   =>  True
```

Then split the line on blanks, and add all but the first two words (the name of the character) to the BOW for that character. If the empty word '' occurs, ignore it (do not add it to the BOW).

In [155]:
```
# Elizabeth Swann's BOW (9 pts)

# Your code here
from collections import defaultdict, Counter

Swann_dict = defaultdict(int)  # default value = 0
pat = r"^elizabeth swann:"
Swann_lines = [re.sub(pat, '', l) for l in prts_split if re.match(pat, l)]
Swann_lines_split = [w for ws in Swann_lines for w in ws.split()]


Swann_dict.update(Counter(Swann_lines_split))  # https://docs.python.org/3/library/collections.htm

sorted_Swann = sorted(Swann_dict.items(), key=lambda v: v[1], reverse=True)[:20]
print("Word    Frequency")
for w, freq in sorted_Swann:
    print(f"{w:{10}}{freq}")
```

```
Word    Frequency
will      23
jack      12
find      7
know      7
oh        7
want      6
man       6
good      5
sparrow   4
would     4
something 4
chance    4
us        3
captain   3
compass   3
give      3
going     3
came      3
way       3
yes       3
```

In [156]:
```python
# Jack Sparrows's BOW   (9 pts)

# Your code here
from collections import defaultdict
Jack_dict = defaultdict(int)  # default value = 0
pat = r"^jack sparrow:"
Jack_lines = [re.sub(pat, '', l) for l in prts_split if re.match(pat, l)]
Jack_lines_split = [w for ws in Jack_lines for w in ws.split()]

Jack_dict.update(Counter(Jack_lines_split))  # https://docs.python.org/3/library/collections.html

sorted_Jack = sorted(Jack_dict.items(), key=lambda v: v[1], reverse=True)[:20]
print("Word    Frequency")
for w, freq in sorted_Jack:
    print(f"{w:{10}}{freq}")
```

```
Word    Frequency
want      15
come      11
will      10
know      9
oh        9
bugger    8
dirt      8
hey       7
jones     7
one       7
love      7
mate      6
captain   6
key       6
would     6
oy        6
save      6
jar       6
chest     6
much      5
```

Optional:

Take a look at the most common words spoken by each; they include names. Who does each mention the most and what does this say about the characters?