

Homework: *State Your Assumptions*

POLONIUS\ *What do you read, my lord?*

HAMLET\ *Words, words, words*

-- *Hamlet*, Act 2, Scene 2

This homework deals with the assumptions made when taking text from its original "raw" form into something more *computable*.

- Assumptions about the *shape* of text (e.g. how to break a corpus into documents)
- Assumptions about what makes a *token*, an *entity*, etc.
- Assumptions about what interesting or important *content* looks like, and how that informs our analyses.

There are three parts:

1. Splitting Lines from Shakespeare
2. Tokenizing and Aligning lines into plays
3. Assessing and comparing characters from within each play

NB This file is merely a *template*, with instructions; do not feel constrained to using it directly if you do not wish to.

Get the Data

Since the class uses `dvc`, it is possible to get this dataset either using the command line (e.g. `dvc import https://github.com/TLP-COI/text-data-course resources/data/shakespeare/shakespeare.txt`), or using the python api (if you wish to use python)

```
In [1]: from dvc.api import read, get_url
import pandas as pd

txt = read('resources/data/shakespeare/shakespeare.txt',
           repo='https://github.com/TLP-COI/text-data-course')

print(txt[:250])
```

First Citizen:
Before we proceed any further, hear me speak.

All:
Speak, speak.

First Citizen:
You are all resolved rather to die than to famish?

All:
Resolved. resolved.

First Citizen:
First, you know Caius Marcius is chief enemy to the people.

Make sure this works before you continue! Either way, it would likely be beneficial to have the data downloaded locally to keep from needing to re-download it every time.

Part 1

Split the text file into a *table*, such that

- each row is a single *line* of dialogue
- there are columns for
 1. the speaker
 2. the line number
 3. the line dialogue (the text)

Hint: you will need to use RegEx to do this rapidly. See the in-class "markdown" example!

Question(s):

- What assumptions have you made about the text that allowed you to do this?

```
In [2]: import pandas as pd
import numpy as np
import re
```

```
In [3]: # Use the first 250 characters as a sample
sample = txt[:250]
sample
```

```
Out[3]: 'First Citizen:\nBefore we proceed any further, hear me speak.\n\nAll:\nSpeak, speak.\n\nFirst Citizen:\nYou are all resolved rather to die than to famish?\n\nAll:\nResolved.
resolved.\n\nFirst Citizen:\nFirst, you know Caius Marcius is chief enemy to the peopl
e.\n'
```

Assumptions:

- the speaker begins with a capitalized letter and ends with a colon
- the dialogue begins with \n (new line) and ends with \n\n

```
In [4]: patt_try = re.compile(
    "^[A-Z].+?):$" # the speaker
    "\n{1}(.*)\n{2}", # the dialogue
    flags = re.S | re.M
)
```

```
In [5]: # Find `patt_try` such that:
matches_try = patt_try.findall(sample)
pd.DataFrame.from_records(matches_try, columns=['speaker', 'dialogue'])
```

```
Out[5]:
```

	speaker	dialogue
0	First Citizen	Before we proceed any further, hear me speak.
1	All	Speak, speak.
2	First Citizen	You are all resolved rather to die than to fam...
3	All	Resolved. resolved.

```
In [6]: # Check the last several lines to see whether they meet the assumptions
print(txt[-330:])
```

```
SEBASTIAN:
I do; and surely
It is a sleepy language and thou speak'st
Out of thy sleep. What is it thou didst say?
This is a strange repose, to be asleep
With eyes wide open; standing, speaking, moving,
And yet so fast asleep.
```

```
ANTONIO:
Noble Sebastian,
Thou let'st thy fortune sleep--die, rather; wink'st
Whiles thou art waking.
```

```
In [7]: txt[-330:]
```

```
Out[7]: "SEBASTIAN:\nI do; and surely\nIt is a sleepy language and thou speak'st\nOut of thy sle
ep. What is it thou didst say?\nThis is a strange repose, to be asleep\nWith eyes wide o
pen; standing, speaking, moving,\nAnd yet so fast asleep.\n\nANTONIO:\nNoble Sebastia
n,\nThou let'st thy fortune sleep--die, rather; wink'st\nWhiles thou art waking.\n"
```

Here, the last line will be omitted according to the current assumptions because it ends with `\n` instead of `\n\n`.

```
In [8]: # Fix the last line by adding an extra \n
new_line = '\n'
txt = txt + new_line
```

Another particular case that doesn't meet the assumptions is empty dialogue.

```
In [9]: # Example of empty dialogue
print(txt[11225:11536])
```

First Senator:
Your company to the Capitol; where, I know,
Our greatest friends attend us.

TITUS:

COMINIUS:
Noble Marcius!

First Senator:

MARCIUS:
Nay, let them follow:
The Volscies have much corn; take these rats thither
To gnaw their garners. Worshipful mutiners,
Your valour puts well forth: pray, follow.

My assumption is that the dialogue begins with `\n` and ends with `\n\n` (a total of three `\n`'s). However, an empty dialogue comes with only two `\n`'s.

```
In [10]: # Fix empty dialogue by adding an extra \n
txt = txt.replace(':', '\n\n')
```

```
In [11]: patt = re.compile(
    "^[A-Z].+?):$" # the speaker
    "\n{1}(.*)\n{2}", # the dialogue
    flags = re.S | re.M
)
```

```
In [12]: # Find `patt` such that:
matches = patt.findall(txt)
df = pd.DataFrame.from_records(matches, columns=['speaker', 'dialogue'])
df.tail()
```

```
Out[12]:
```

	speaker	dialogue
7217	ANTONIO	Nor I; my spirits are nimble.\nThey fell toget...
7218	SEBASTIAN	What, art thou waking?
7219	ANTONIO	Do you not hear me speak?
7220	SEBASTIAN	I do; and surely\nIt is a sleepy language and ...
7221	ANTONIO	Noble Sebastian,\nThou let'st thy fortune slee...

```
In [13]: # Create a new column for line number
df['line'] = df.index + 1
df
```

Out [13]:

	speaker	dialogue	line
0	First Citizen	Before we proceed any further, hear me speak.	1
1	All	Speak, speak.	2
2	First Citizen	You are all resolved rather to die than to fam...	3
3	All	Resolved. resolved.	4
4	First Citizen	First, you know Caius Marcius is chief enemy t...	5
...
7217	ANTONIO	Nor I; my spirits are nimble.\nThey fell toget...	7218
7218	SEBASTIAN	What, art thou waking?	7219
7219	ANTONIO	Do you not hear me speak?	7220
7220	SEBASTIAN	I do; and surely\nIt is a sleepy language and ...	7221
7221	ANTONIO	Noble Sebastian,\nThou let'st thy fortune slee...	7222

7222 rows × 3 columns

In [14]:

```
# Split dialogue text into multiple rows
# Ex. split row 7221 into multiple rows based on \n in the dialogue column
(df['dialogue'].str.split('\n', expand=True).stack()
 .reset_index(level=1, drop=True).rename('dialogue'))
```

Out [14]:

```
0      Before we proceed any further, hear me speak.
1      Speak, speak.
2      You are all resolved rather to die than to fam...
3      Resolved. resolved.
4      First, you know Caius Marcius is chief enemy t...
      ...
7220    With eyes wide open; standing, speaking, moving,
7220      And yet so fast asleep.
7221      Noble Sebastian,
7221    Thou let'st thy fortune sleep--die, rather; wi...
7221      Whiles thou art waking.
Name: dialogue, Length: 25680, dtype: object
```

In [15]:

```
# Join the splitted dialogue with the original dataframe
df = (df.drop('dialogue', axis=1)
      .join(df['dialogue'].str.split('\n', expand=True).stack()
            .reset_index(level=1, drop=True).rename('dialogue'))
      .reset_index(drop=True))
df
```

Out [15]:

	speaker	line	dialogue
0	First Citizen	1	Before we proceed any further, hear me speak.
1	All	2	Speak, speak.
2	First Citizen	3	You are all resolved rather to die than to fam...
3	All	4	Resolved. resolved.
4	First Citizen	5	First, you know Caius Marcius is chief enemy t...
...
25675	SEBASTIAN	7221	With eyes wide open; standing, speaking, moving,
25676	SEBASTIAN	7221	And yet so fast asleep.
25677	ANTONIO	7222	Noble Sebastian,
25678	ANTONIO	7222	Thou let'st thy fortune sleep--die, rather; wi...
25679	ANTONIO	7222	Whiles thou art waking.

25680 rows × 3 columns

Part 2

You have likely noticed that the lines are not all from the same play! Now, we will add some useful metadata to our table:

- Determine a likely source title for each line
- add the title as a 'play' column in the data table.
- make sure to document your decisions, assumptions, external data sources, etc.

This is fairly open-ended, and you are not being judged completely on *accuracy*. Instead, think outside the box a bit as to how you might accomplish this, and attempt to justify whatever approximations or assumptions you felt were appropriate.

I found a Kaggle dataset about Shakespeare plays

(<https://www.kaggle.com/kingburrito666/shakespeare-plays>). It has the following columns:

1. Dateline
2. Play
3. PlayerLinenumbr
4. ActSceneLine
5. Player
6. PlayerLine

In [16]:

```
# Read in the data
shakespeare = pd.read_csv('Shakespeare_data.csv')
shakespeare[92333:92338]
```

Out [16]:

	Dataline	Play	Player	Linenumber	Act	Scene	Line	Player	PlayerLine
92333	92334	The Tempest		120.0	2.1.227	SEBASTIAN	With eyes wide open, standing, speaking, moving,		
92334	92335	The Tempest		120.0	2.1.228	SEBASTIAN	And yet so fast asleep.		
92335	92336	The Tempest		121.0	2.1.229	ANTONIO	Noble Sebastian,		
92336	92337	The Tempest		121.0	2.1.230	ANTONIO	Thou let'st thy fortune sleep--die, rather, wi...		
92337	92338	The Tempest		121.0	2.1.231	ANTONIO	Whiles thou art waking.		

I can determine the play by matching speaker to Player and dialogue to PlayerLine. For easier matches, I will drop the empty dialogues and transform the remaining to plain lowercase text.

In [17]:

```
# df: drop the empty dialogues
df = df[df['dialogue'] != ''].reset_index(drop=True)
```

In [18]:

```
# Remove punctuation and lowercase (example)
(shakespeare[92333:92338]['PlayerLine']
 .str.replace(r'^\w\s', '', regex=True)
 .str.lower())
```

Out[18]:

92333	with eyes wide open standing speaking moving
92334	and yet so fast asleep
92335	noble sebastian
92336	thou letst thy fortune sleepdie rather winkst
92337	whiles thou art waking

Name: PlayerLine, dtype: object

In [19]:

```
# df: create a new column for transformed dialogue
df['mat'] = (df['dialogue']
 .str.replace(r'^\w\s', '', regex=True)
 .str.lower())

df.tail(3)
```

Out[19]:

	speaker	line	dialogue	mat
25552	ANTONIO	7222	Noble Sebastian,	noble sebastian
25553	ANTONIO	7222	Thou let'st thy fortune sleep--die, rather; wi...	thou letst thy fortune sleepdie rather winkst
25554	ANTONIO	7222	Whiles thou art waking.	whiles thou art waking

In [20]:

```
# shakespeare: create a new column for transformed PlayerLine
shakespeare['che'] = (shakespeare['PlayerLine']
 .str.replace(r'^\w\s', '', regex=True)
 .str.lower())

shakespeare[92335:92338]
```

Out [20]:

	Dataline	Play	Player	Linenum	ActSceneLine	Player	PlayerLine	che
92335	92336	The Tempest		121.0	2.1.229	ANTONIO	Noble Sebastian,	noble sebastian
92336	92337	The Tempest		121.0	2.1.230	ANTONIO	Thou let'st thy fortune sleep--die, rather, wi...	thou letst thy fortune sleepdie rather winkst
92337	92338	The Tempest		121.0	2.1.231	ANTONIO	Whiles thou art waking.	whiles thou art waking

In [21]:

```
# Merge df and shakespeare
# ['speaker', 'mat'] == ['Player', 'che']
df = (df
      .merge(shakespeare[['Play', 'Player', 'che']],
              how = 'left',
              left_on = ['speaker', 'mat'],
              right_on = ['Player', 'che'])
      .drop(['Player', 'mat', 'che'], axis = 1)
      .rename(str.lower, axis='columns'))
df
```

Out [21]:

	speaker	line	dialogue	play
0	First Citizen	1	Before we proceed any further, hear me speak.	Coriolanus
1	All	2	Speak, speak.	Coriolanus
2	First Citizen	3	You are all resolved rather to die than to fam...	Coriolanus
3	All	4	Resolved. resolved.	Coriolanus
4	First Citizen	5	First, you know Caius Marcius is chief enemy t...	Coriolanus
...
25633	SEBASTIAN	7221	With eyes wide open; standing, speaking, moving,	The Tempest
25634	SEBASTIAN	7221	And yet so fast asleep.	The Tempest
25635	ANTONIO	7222	Noble Sebastian,	The Tempest
25636	ANTONIO	7222	Thou let'st thy fortune sleep--die, rather; wi...	The Tempest
25637	ANTONIO	7222	Whiles thou art waking.	The Tempest

25638 rows × 4 columns

In [22]:

```
# Check the unmatched dialogues
df[df['play'].isnull()]
```


Out [22]:

	speaker	line	dialogue	play
789	CORIOLANUS	228	But then Aufidius was within my view,	NaN
942	VOLUMNIA	274	Nay,'tis true.	NaN
1569	BRUTUS	455	And	NaN
1570	BRUTUS	455	Twice being	NaN
1605	LARTIUS	466	On safe-guard he came to me; and did curse	NaN
...
21617	ANGELO	5878	Happy return be to your royal grace!	NaN
22975	PETRUCHIO	6265	The youngest daughter whom you hearken for	NaN
22995	GRUMIO	6269	O excellent motion! Fellows, let's be gone.	NaN
23326	GREMIO	6383	Amen, say we: we will be witnesses.	NaN
23860	ALL SERVING-MEN	6570	Here, here, sir; here, sir.	NaN

64 rows × 4 columns

The number of rows increased from 25554 to 25638, indicating that some dialogues got matched more than once. In addition, there are 64 unmatched dialogues.

Assumptions:

- If an unmatched and a match belong to the same line, they should correspond to the same play.
- If the dialogue before an unmatched has a corresponding play and the dialogue after belong to the same play, the unmatched should come from the same play.

In [23]:

```
# Example of the first assumption
# line 455 belong to Coriolanus
# row 1569 and 1570 should come from the same play
df[1568:1573]
```

Out [23]:

	speaker	line	dialogue	play
1568	BRUTUS	455	That our beat water brought by conduits hither;	Coriolanus
1569	BRUTUS	455	And	NaN
1570	BRUTUS	455	Twice being	NaN
1571	BRUTUS	455	Was his great ancestor.	Coriolanus
1572	SICINIUS	456	One thus descended,	Coriolanus

In [24]:

```
for i in range(1,len(df)):
    if df.loc[i,'play'] is np.nan:
        if ((df.loc[i,'line'] == df.loc[i-1,'line']) and # i and i-1 have the same line
            (df.loc[i-1,'play'] is not np.nan)): # i-1 belongs to a play
            df.loc[i,'play'] = df.loc[i-1,'play'] # assign i with the same play
```

In [25]:

```
# Example of the second assumption
# row 2746 and row 2748 belong to the same play
# => row 2747 should come from the same play
df[2745:2750]
```

Out [25]:

	speaker	line	dialogue	play
2745	First Servingman	843	Ay, and for an assault too.	Coriolanus
2746	Third Servingman	844	O slaves, I can tell you news,-- news, you ras...	Coriolanus
2747	First Servingman	845	What, what, what? let's partake.	NaN
2748	Third Servingman	846	I would not be a Roman, of all nations; I had as	Coriolanus
2749	Third Servingman	846	lieve be a condemned man.	Coriolanus

In [26]:

```
for i in range(1,len(df)-1):
    if df.loc[i,'play'] is np.nan:
        if ((df.loc[i-1,'play'] is not np.nan) and # i-1 belongs to a play
            (df.loc[i-1,'play'] == df.loc[i+1,'play'])): # i+1 belongs to the same play
            df.loc[i,'play'] = df.loc[i-1,'play'] # assign i with the same play
```

In [27]:

```
# Check the unmatched dialogues now
df[df['play'].isnull()]
```

Out [27]:

	speaker	line	dialogue	play
1823	Senators, &C	539	Weapons, weapons, weapons!	NaN
1824	Senators, &C	539	'Tribunes!' 'Patricians!' 'Citizens!' 'What, ho!'	NaN
1825	Senators, &C	539	'Sicinius!' 'Brutus!' 'Coriolanus!' 'Citizens!'	NaN
1826	Senators, &C	539	'Peace, peace, peace!' 'Stay, hold, peace!'	NaN

In [28]:

```
# Manually find the play for the 4 unmatched dialogues
df = df.fillna('Coriolanus')
df[1823:1827]
```

Out [28]:

	speaker	line	dialogue	play
1823	Senators, &C	539	Weapons, weapons, weapons!	Coriolanus
1824	Senators, &C	539	'Tribunes!' 'Patricians!' 'Citizens!' 'What, ho!'	Coriolanus
1825	Senators, &C	539	'Sicinius!' 'Brutus!' 'Coriolanus!' 'Citizens!'	Coriolanus
1826	Senators, &C	539	'Peace, peace, peace!' 'Stay, hold, peace!'	Coriolanus

In [29]:

```
# Resulting dataframe
df
```

Out [29]:

	speaker	line	dialogue	play
0	First Citizen	1	Before we proceed any further, hear me speak.	Coriolanus
1	All	2	Speak, speak.	Coriolanus
2	First Citizen	3	You are all resolved rather to die than to fam...	Coriolanus
3	All	4	Resolved. resolved.	Coriolanus
4	First Citizen	5	First, you know Caius Marcius is chief enemy t...	Coriolanus
...
25633	SEBASTIAN	7221	With eyes wide open; standing, speaking, moving,	The Tempest
25634	SEBASTIAN	7221	And yet so fast asleep.	The Tempest
25635	ANTONIO	7222	Noble Sebastian,	The Tempest
25636	ANTONIO	7222	Thou let'st thy fortune sleep--die, rather; wi...	The Tempest
25637	ANTONIO	7222	Whiles thou art waking.	The Tempest

25638 rows × 4 columns

Part 3

Pick one or more of the techniques described in this chapter:

- keyword frequency
- entity relationships
- markov language model
- bag-of-words, TF-IDF
- semantic embedding

make a case for a technique to measure how *important* or *interesting* a speaker is. The measure does not have to be both important *and* interesting, and you are welcome to come up with another term that represents "useful content", or tells a story (happiest speaker, worst speaker, etc.)

Whatever you choose, you must

1. document how your technique was applied
2. describe why you believe the technique is a valid approximation or exploration of how important, interesting, etc., a speaker is.
3. list some possible weaknesses of your method, or ways you expect your assumptions could be violated within the text.

This is mostly about learning to transparently document your decisions, and iterate on a method for operationalizing useful analyses on text. Your explanations should be understandable; homeworks will be peer-reviewed by your fellow students.

In [30]:

```
# Create a function for generating word clouds
import matplotlib.pyplot as plt
def value_ct_wordcloud(s: pd.Series):
    from wordcloud import WordCloud
    wc = (WordCloud(background_color="white", max_words=50)
          .generate_from_frequencies(s.to_dict()))
    plt.figure()
```

```
plt.imshow(wc, interpolation="bilinear")
plt.axis("off")
plt.show()
```

```
In [31]: # Visualize types of play
(df.play.explode().value_counts()
 .pipe(value_ct_wordcloud))
```



```
In [32]: # Visualize types of speaker
(df.speaker.explode().value_counts()
 .pipe(value_ct_wordcloud))
```



```
In [33]: # Tokenize dialogue
from nltk.tokenize import word_tokenize
df["tokens"] = (df['dialogue']
 .str.replace(r'^\w\s', '', regex=True) # remove punctuation
 .str.lower() # lower case
 .apply(word_tokenize))

df.head(3)
```

	speaker	line	dialogue	play	tokens
0	First Citizen	1	Before we proceed any further, hear me speak.	Coriolanus	[before, we, proceed, any, further, hear, me, ...]
1	All	2	Speak, speak.	Coriolanus	[speak, speak]
2	First Citizen	3	You are all resolved rather to die than to fam...	Coriolanus	[you, are, all, resolved, rather, to, die, tha...]

```
In [34]: # Remove stopwords from tokens
from nltk.corpus import stopwords
sw = stopwords.words("english")
df['tokens'] = df['tokens'].apply(lambda x: [tkn for tkn in x if tkn not in sw])
df.head(3)
```

Out [34]:

	speaker	line	dialogue	play	tokens
0	First Citizen	1	Before we proceed any further, hear me speak.	Coriolanus	[proceed, hear, speak]
1	All	2	Speak, speak.	Coriolanus	[speak, speak]
2	First Citizen	3	You are all resolved rather to die than to fam...	Coriolanus	[resolved, rather, die, famish]

In [35]:

```
# Count the number of types
df.tokens.explode().value_counts().head(30)
```

Out [35]:

thou	1405
thy	1059
shall	845
thee	760
good	671
lord	632
come	628
sir	592
well	565
would	532
ill	467
hath	453
king	448
say	445
let	437
one	423
go	421
love	410
may	406
us	401
make	397
upon	390
yet	386
like	376
must	370
know	350
man	343
tis	328
see	322
death	304

Name: tokens, dtype: int64

In [36]:

```
# Add more stopwords
extra = ['thou','thy','shall','thee','would','hath',
        'let','one','may','us','upon','yet','tis']
sw = sw + extra
```

In [37]:

```
# Remove extra stopwords from tokens
df['tokens'] = df['tokens'].apply(lambda x: [tkn for tkn in x if tkn not in sw])
df.tail(3)
```

Out [37]:

	speaker	line	dialogue	play	tokens
25635	ANTONIO	7222	Noble Sebastian,	The Tempest	[noble, sebastian]
25636	ANTONIO	7222	Thou let'st thy fortune sleep--die, rather; wi...	The Tempest	[letst, fortune, sleepdie, rather, winkst]
25637	ANTONIO	7222	Whiles thou art waking.	The Tempest	[whiles, art, waking]

In [38]:

```
# Generate word clouds for the 10 most frequent speakers
speakers = df.speaker.explode().value_counts().head(10).index.tolist()
```

```
Out[38]:
```

GLOUCESTER	904
DUKE VINCENTIO	824
KING RICHARD II	760
LEONTES	679
CORIOLANUS	676
ROMEO	604
MENENIUS	587
PETRUCHIO	568
JULIET	551
QUEEN MARGARET	498
Name: speaker, dtype: int64	

```
Out[39]: 'GLOUCESTER'
```

```
In [41]: # Visualize keywords of a speaker
for name in speakers:
    print(name)
    speaker_wordcloud(name)
```

handface thus come mine
cousin make
way man time
blood crown know away live kings see tears
name good like love must
men ill eyes many god go great give heart
norfolk fair proud bolingbroke little think lord thoughts
king well

LEONTES

bear father take thine art come
done speak true eyes wife hast think
polixenes thought like might thing
life bastard boy mine nothing
ill even well thus know away much
hermione man look play great honour
good camillo
truth make paulina give hence shes

CORIOLANUS

peace must ill maker rome go
tongue show power hear whose heart know
noble done away gods wife true thus
like made sir love give come good
rather never part pray service say mine well
people aufidius eyes stand still common

ROMEO

light farewell look beauty give mercutio
stay hand death eyes go
poison lips well man world
dear joy come must mine night
banished loves never come take soul thine
ill art wilt tybalt love good think
juliet live tell lady doth
heaven rich sweet like name nurse father

MENENIUS

make tribunes work nay gods belly fair like friends
men know home say little
fellow made news leave go
hear must good two pray matter
rome noble thus time hessee
coriolanus that cominius speak come tell
marcius much general think could

PETRUCHIO



JULIET



QUEEN MARGARET



The word clouds above explore how interesting the speakers are. Juliet, for example, is all about love and Romeo, while Queen Margaret mentions a ton about the King and Edward. One weakness of this method is that some keywords do not necessarily represent meanings, but rather the name of the person talking to the speaker.