01-Operationalizing

September 19, 2017

```
In [8]: # Show graphs in notebook
        %matplotlib inline

        # Import libraries
        import pandas as pd  # tabular data analysis library
        import numpy as np  # mathematical operations library
        import os  # library for manipulating the file system and Bash
        from sklearn.feature_extraction.text import CountVectorizer
        import re  # regular expressions library
        import matplotlib.pyplot as plt  # plotting base library
        import seaborn as sns  # plotting extension library
        from bs4 import BeautifulSoup  # html/lxml parsing library
        from datascience import *
```

1 Character Space

This notebook recreates results discussed in:

- Moretti, Franco. "'Operationalizing': or, the function of measurement in modern literary theory". Stanford Literary Lab Pamphlet 6. 2013

In Moretti's study, he offers several measures of the concept of character space. The simplest of these is to measure the relative dialogue belonging to each character in a play. Presumably the main characters will speak more and peripheral characters will speak less.

The statistical moves we will make here are not only counting the raw number of words spoken by each character, but also normalizing them. That is, converting them into a fraction of all words in the play.

In order to focus on the statistical tasks at hand, we need to parse raw text files to figure out who said what. Unfortunately, that's the hard part! We'll walk through the first one and I'll quickly do the ones after.

2 Jean Racine's *Phèdre*

```
In [9]: # Read the text of the play from its file on the hard-drive
        with open('data/phedre.txt', 'r') as f:
            phedre = f.read()
        print(phedre[:200])  # print first 200 characters
```

```
ACT I

SCENE I
HIPPOLYTUS, THERAMENES


HIPPOLYTUS
My mind is settled, dear Theramenes,
And I can stay no more in lovely Troezen.
In doubt that racks my soul with mortal anguish,
I grow ashamed of suc
```

```
In [10]: # Create a list, where each entry is a line from the play. We'll split on double line
         # Each line starts with the name of the speaker.
         phedre_list = phedre.split('\n\n')

         # Create a regex pattern to match words we don't want to start the line
         pattern = re.compile(r'ACT|SCENE|Scene')

         # Grab list of all the dialogue lines if they don't have the words above in them
         phedre_list = [x.strip() for x in phedre_list if re.match(pattern, x) == None and '\n

         # Print first three dialogue turns
         phedre_list[:3]
```

```
Out[10]: ['HIPPOLYTUS\nMy mind is settled, dear Theramenes,\nAnd I can stay no more in lovely T
          "THERAMENES\nAnd where, prince, will you look for him?\nAlready, to content your jus
          "HIPPOLYTUS\nCease, dear Theramenes, respect the name\nOf Theseus. Youthful errors ha
```

Now that we have the dialogue texts in a list, we can attribute dialogue words to each character.

> "character-space turns smoothly into “word-space”—“the number of words allocated to a particular character”—and, by counting the words each character utters, we can determine how much textual space it occupies." [2]

```
In [11]: # Create a dictionary where each key is the name of a character
         # and each entry is a single string of words spoken by them

         # Initiate empty dict
         dialogue_dict_phedre = {}

         # Iterate through list of turns in the dialogue list
         for line in phedre_list:

             # Get the name of the character
```

```
    char = line.split('\n')[0].split()[0]

    # Get the dialogue text
    dialogue = '\n'.join(line.split('\n')[1:])

    # Add dialogue text to that character
    if char not in dialogue_dict_phedre.keys():
        dialogue_dict_phedre[char] = dialogue
    else:
        dialogue_dict_phedre[char] += dialogue


# Print first 200 character's of Phaedra's dialogue
print(dialogue_dict_phedre['PHAEDRA'][:200])
```

```
We have gone far enough. Stay, dear Oenone;
Strength fails me, and I needs must rest awhile.
My eyes are dazzled with this glaring light
So long unseen, my trembling knees refuse
Support. Ah me!Ah, ho
```

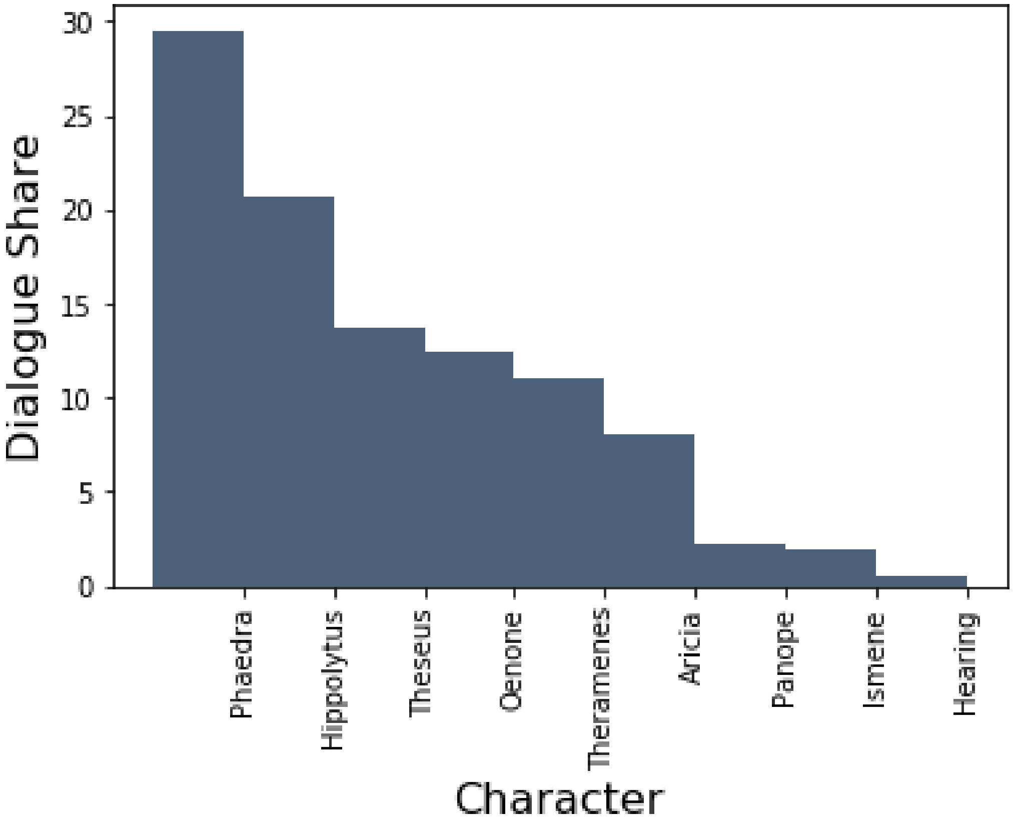
```
In [12]: def plot_character_space(dialogue):

             # Create counter to get all words in all dialogue
             total_words = 0
             for char in dialogue.keys():
                 total_words += len(dialogue[char].split())

             # Create dict to record share of dialogue for each character
             dialogue_share = []
             for char in dialogue.keys():
                 dialogue_share.append({'Character': char.title(), 'Dialogue Share': len(dialo

             my_table = Table.from_records(dialogue_share).sort('Dialogue Share', descending=T
             my_table.bar(column_for_categories='Character')
             plt.xticks(range(len(my_table.columns[0])), my_table.columns[0], rotation=90)

In [13]: plot_character_space(dialogue_dict_phedre)
```



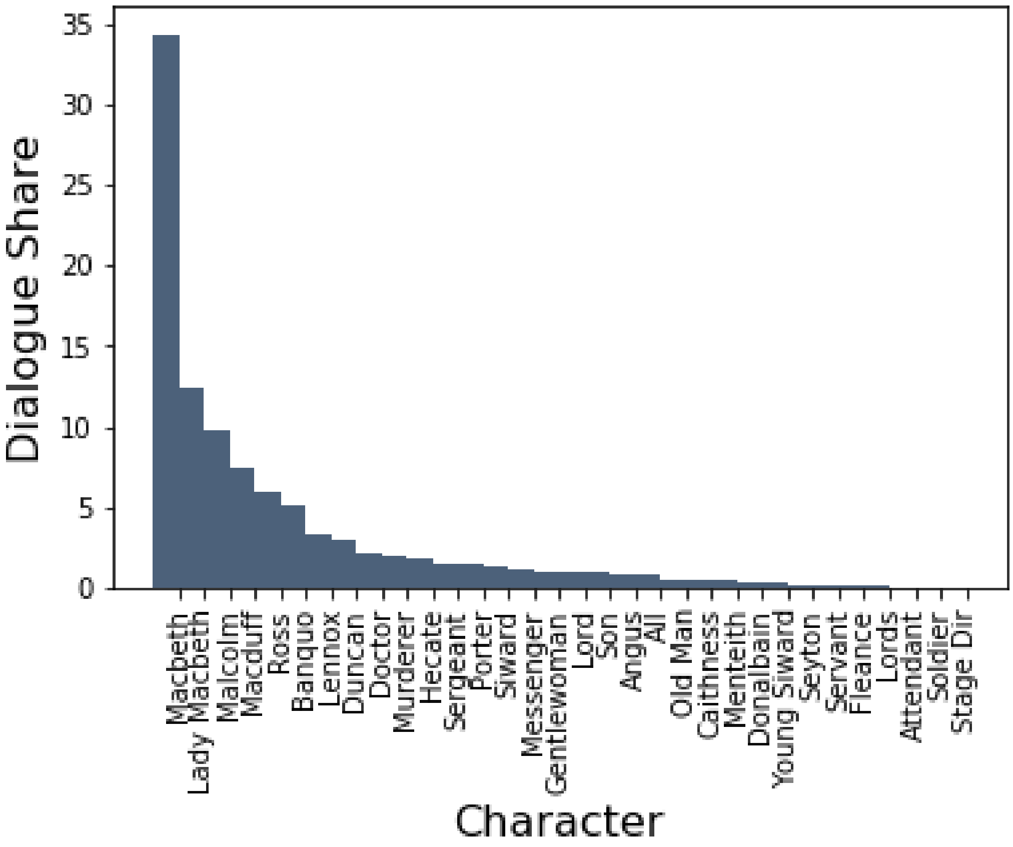
3 Macbeth

```
In [14]: # Read in text
         with open('data/macbeth.txt', 'r') as f:
             macbeth = f.read()

         # Get cast
         pattern = re.compile(r'<[A-Z ]*>')
         cast = list(set(re.findall(pattern, macbeth)))
         cast = [x.replace('>', '').replace('<', '') for x in cast]

         # Make dialogue dict
         soup = BeautifulSoup(macbeth, 'lxml')
         dialogue_dict_macbeth = {}
         for c in cast:
             dialogue = [x.text for x in soup.find_all(c.lower().split()[0])]
             dialogue = '\n'.join([re.sub(r'<.*>', '', x).strip() for x in dialogue])
             dialogue_dict_macbeth[c] = dialogue
```

```
# Plot
plot_character_space(dialogue_dict_macbeth)
```



4 Othello

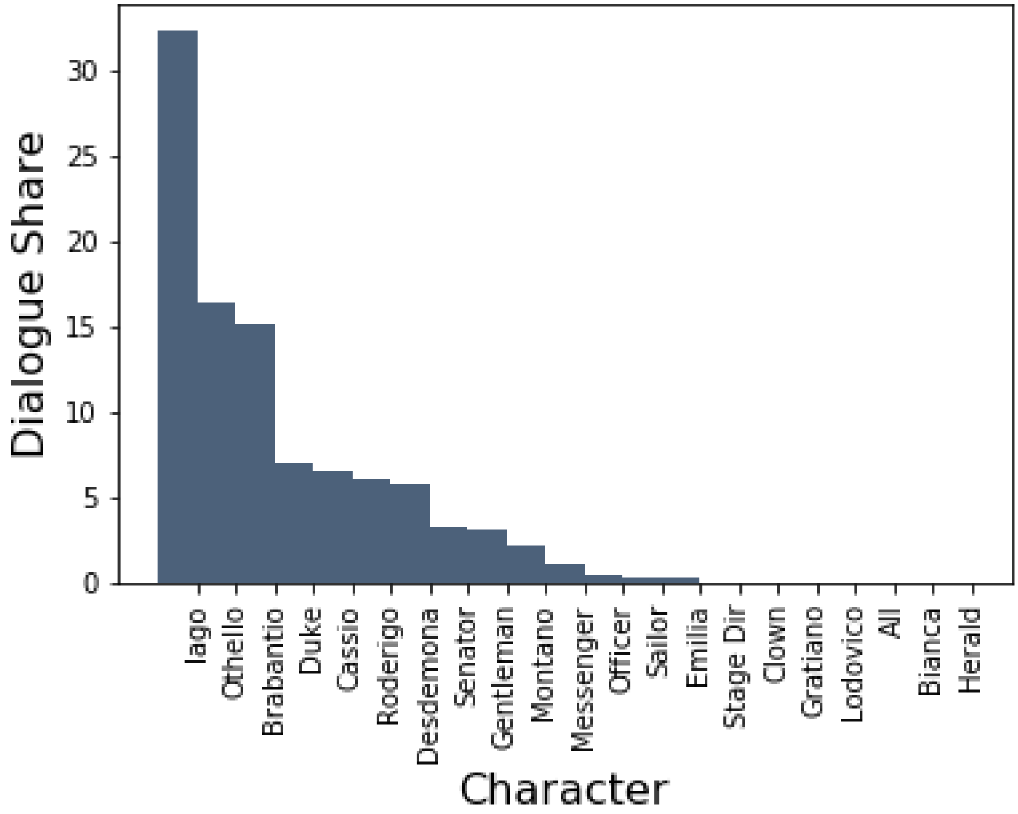
```
In [15]: # Read in text
         with open('data/othello.txt', 'r') as f:
             othello = f.read()

         # Get cast
         pattern = re.compile(r'<[A-Z ]*>')
         cast = list(set(re.findall(pattern, othello)))
         cast = [x.replace('>', '').replace('<', '') for x in cast]

         # Make dialogue dict
         soup = BeautifulSoup(othello, 'lxml')
         dialogue_dict_othello = {}
         for c in cast:
             dialogue = [x.text for x in soup.find_all(c.lower().split()[0])]
```

```python
    dialogue = '\n'.join([re.sub(r'<.*>', '', x).strip() for x in dialogue])
    dialogue_dict_othello[c] = dialogue

# Plot
plot_character_space(dialogue_dict_othello)
```



5 Antigone

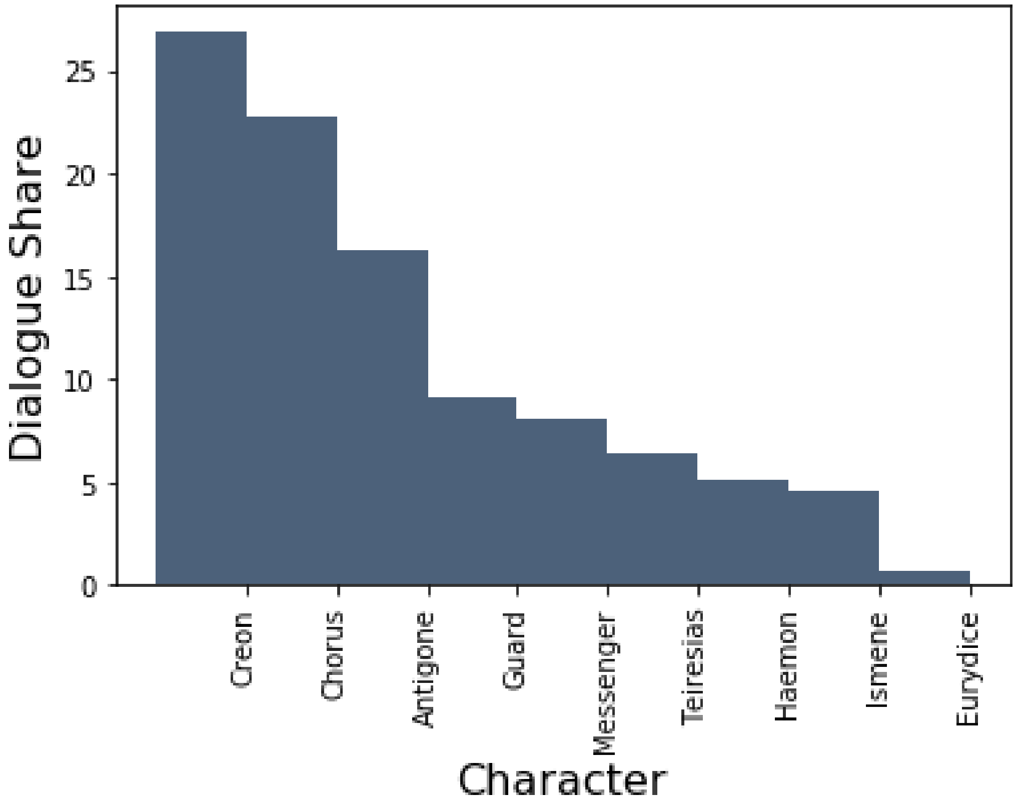
In [16]:

```python
# Read in text
with open('data/antigone.txt', 'r') as f:
    antigone = f.read()

# Split lines
antigone_list = antigone.split('\n\n')

# Make dialogue dict
dialogue_dict_antigone = {}
for line in antigone_list:
    dex = line.index(' ')
    char = line[:dex]
    if char not in dialogue_dict_antigone.keys():
        dialogue_dict_antigone[char] = line[dex:]
    else:
        dialogue_dict_antigone[char] += line[dex:]

# Plot
plot_character_space(dialogue_dict_antigone)
```



6 Operationalizing Tragic Collision: Most Distinctive Words

The code below looks complicated, but all it does is count how many times each character said each word in the entire text. If the character didn't say the word, it just gets tallied as a 0. We then sum all of these counts to get the number of times each word is spoken in the text. If we're intested in the most distinctive words, we'd want to know how many times a character said a specific word compared to how many times it was spoken in the entire text.

We'll make an 'EXPECTED' column that tells us if the word was distributed evenly amongst characters, how many times our target character should have said it. Then we'll add a column for the ratio between the observed occurences and the expected occurences.

TLDR: This code will tell us which words a specific character used more or less frequently than average for a character in a text.

"To do this, the Literary Lab follows an approach (which we call Most Distinctive Words) in several steps. First, we establish how often a word occurs in the corpus, and hence how often a specific character is expected to use it given the amount of words at its disposal; then we count how often the character actually utters the word, and calculate the ratio between actual and expected frequency; the higher the ratio, the greater the deviation from the average, and the more typical the word is of that character." [10]

```
In [17]: def get_mdw(dialogue_dict, character, group=False):
             # Boot up the dtm-maker
             cv = CountVectorizer()

             # Create the dtm
             dtm = cv.fit_transform(dialogue_dict.values()).toarray()

             # Put the dtm into human-readable format
             word_list = cv.get_feature_names()

             dtm_df = pd.DataFrame(dtm, columns = word_list, index = dialogue_dict.keys())

             # Create new dataframe
             mdw_df = pd.DataFrame()

             # Add a column for her observed word counts
             mdw_df[character] = dtm_df.loc[character]

             if group == False:
                 # Add a column for the total counts of each word in the play
                 mdw_df['WORD_TOTAL'] = dtm_df.sum()
             else:
                 # Add a column for the total counts of each word for the characters in the de
                 mdw_df['WORD_TOTAL'] = dtm_df.loc[group].sum()

             # Calculate Antigone's share of the total dialogue
             char_space = sum(mdw_df[character])/float(sum(mdw_df['WORD_TOTAL']))

             # Add a new column in which we calculate an "expected" number of times
             # Antigone would utter each word, based on its overall use in the play
             # and her share of the dialogue.

             mdw_df[character + '_EXPECTED'] = mdw_df['WORD_TOTAL']*char_space

             # How much more/less frequently does Antigone use the word than expected?
             mdw_df['OBS-EXP_RATIO'] = mdw_df[character]/(mdw_df[character + '_EXPECTED'])

             # Sort the dataframe by the Observed/Expected Ratio to show
             # Antigone's 20 "Most Distinctive Words"
             return mdw_df[(mdw_df['OBS-EXP_RATIO']>1)&(mdw_df['WORD_TOTAL']>5)].sort_values('
In [18]: get_mdw(dialogue_dict_antigone, 'ANTIGONE')
```

```
Out[18]:           ANTIGONE  WORD_TOTAL  ANTIGONE_EXPECTED  OBS-EXP_RATIO
         brother         10          14           2.234963       4.474348
         aught            4           6           0.957841       4.176058
         suffer           4           7           1.117481       3.579478
         mother           7          14           2.234963       3.132043
         nor              7          14           2.234963       3.132043
         home             3           6           0.957841       3.132043
         heaven           3           6           0.957841       3.132043
         knew             3           6           0.957841       3.132043
         mine            12          24           3.831365       3.132043
         another          3           6           0.957841       3.132043
         most             3           6           0.957841       3.132043
         edict            3           6           0.957841       3.132043
         could            6          13           2.075323       2.891117
         share            4           9           1.436762       2.784038
         living           4           9           1.436762       2.784038
         hades            4           9           1.436762       2.784038
         fear             3           7           1.117481       2.684609
         marriage         3           7           1.117481       2.684609
         knowest          3           7           1.117481       2.684609
         daughter         3           7           1.117481       2.684609
```

```
In [19]: get_mdw(dialogue_dict_antigone, 'CREON')
```

```
Out[19]:         CREON  WORD_TOTAL  CREON_EXPECTED  OBS-EXP_RATIO
         let          7           9        2.424587       2.887090
         sayest       6           8        2.155188       2.783980
         woman        8          11        2.963384       2.699617
         your         6           9        2.424587       2.474649
         dost         8          12        3.232782       2.474649
         lead         4           6        1.616391       2.474649
         grave        4           6        1.616391       2.474649
         ye          16          26        7.004361       2.284291
         friend       5           9        2.424587       2.062207
         woe          8          15        4.040978       1.979719
         shalt        4           8        2.155188       1.855987
         side         4           8        2.155188       1.855987
         taken        4           8        2.155188       1.855987
         burial       4           8        2.155188       1.855987
         edict        3           6        1.616391       1.855987
         every        3           6        1.616391       1.855987
         ruin         3           6        1.616391       1.855987
         behold       3           6        1.616391       1.855987
         before       6          12        3.232782       1.855987
         wife         3           6        1.616391       1.855987
```

Here's what Moretti had as most distinctive words:

But Moretti notes that these are Antigone's and Creon's most distinctive words as compared to the rest of the text (al the characters in the text). What we are interested in only the relationship between the two characters? We can look at the most distinctive words given the dialogue of only Antigone and Creon the same way, just leaving out the rest of the dialogue:

```
In [20]: get_mdw(dialogue_dict_antigone, 'ANTIGONE', group=['ANTIGONE', 'CREON'])
```

```
Out[20]:          ANTIGONE  WORD_TOTAL  ANTIGONE_EXPECTED  OBS-EXP_RATIO
         nor             7           8           2.976705       2.351594
         mother          7           8           2.976705       2.351594
         brother        10          13           4.837145       2.067335
         could           6           8           2.976705       2.015652
         whom            5           7           2.604617       1.919668
         suffer          4           6           2.232529       1.791690
         share           4           6           2.232529       1.791690
         last            4           6           2.232529       1.791690
         die             4           6           2.232529       1.791690
         go              4           6           2.232529       1.791690
         living          4           6           2.232529       1.791690
         mine           12          18           6.697586       1.791690
         dead            9          14           5.209233       1.727701
         had             5           8           2.976705       1.679710
         wilt            8          13           4.837145       1.653868
         they            9          15           5.581321       1.612521
         gods            7          12           4.465057       1.567729
         such            7          12           4.465057       1.567729
         thee           14          24           8.930114       1.567729
         thus            4           7           2.604617       1.535735
```

```
In [21]: get_mdw(dialogue_dict_antigone, 'CREON', group=['ANTIGONE', 'CREON'])
```

```
Out[21]:         CREON  WORD_TOTAL  CREON_EXPECTED  OBS-EXP_RATIO
         sayest      6           6        3.767471       1.592580
         she        16          16       10.046590       1.592580
         woman       8           8        5.023295       1.592580
         woe         8           8        5.023295       1.592580
         let         7           7        4.395383       1.592580
         we          8           9        5.651207       1.415627
         dost        8           9        5.651207       1.415627
         evil        7           8        5.023295       1.393508
         man        13          15        9.418679       1.380236
         even        6           7        4.395383       1.365069
         this       45          54       33.907243       1.327150
         art         5           6        3.767471       1.327150
         indeed      5           6        3.767471       1.327150
         away        5           6        3.767471       1.327150
         men         9          11        6.907031       1.303020
         son         9          11        6.907031       1.303020
         his        21          26       16.325709       1.286315
```

```
her          25          31      19.465269      1.284339
at           14          18      11.302414      1.238673
them          7           9       5.651207      1.238673
```

Here's what Moretti had:

6.1 Challenge

Experiment with looking at the most distinctive words for characters in the other plays we looked at (*Phèdre*, *Macbeth*, and Othello).

HINT: You should only have to write one line per text!

```
In [24]: get_mdw(dialogue_dict_phedre, 'Phèdre')
```

```
    ---------------------------------------------------------------------------

    KeyError                                  Traceback (most recent call last)

    /srv/app/venv/lib/python3.6/site-packages/pandas/core/indexing.py in _has_valid_type(se
   1410                 if key not in ax:
-> 1411                     error()
   1412             except TypeError as e:


    /srv/app/venv/lib/python3.6/site-packages/pandas/core/indexing.py in error()
   1405                 raise KeyError("the label [%s] is not in the [%s]" %
-> 1406                                (key, self.obj._get_axis_name(axis)))
   1407


    KeyError: 'the label [Phèdre] is not in the [index]'


During handling of the above exception, another exception occurred:


    KeyError                                  Traceback (most recent call last)

    <ipython-input-24-59e84ff253b1> in <module>()
----> 1 get_mdw(dialogue_dict_phedre, 'Phèdre')


    <ipython-input-17-31cae786bfd9> in get_mdw(dialogue_dict, character, group)
     15
     16     # Add a column for her observed word counts
---> 17     mdw_df[character] = dtm_df.loc[character]
     18
     19     if group == False:
```

```
/srv/app/venv/lib/python3.6/site-packages/pandas/core/indexing.py in __getitem__(self,
1310             return self._getitem_tuple(key)
1311         else:
-> 1312             return self._getitem_axis(key, axis=0)
1313
1314     def _getitem_axis(self, key, axis=0):


/srv/app/venv/lib/python3.6/site-packages/pandas/core/indexing.py in _getitem_axis(sel
1480
1481         # fall thru to straight lookup
-> 1482         self._has_valid_type(key, axis)
1483         return self._get_label(key, axis=axis)
1484


/srv/app/venv/lib/python3.6/site-packages/pandas/core/indexing.py in _has_valid_type(s
1417                 raise
1418             except:
-> 1419                 error()
1420
1421         return True


/srv/app/venv/lib/python3.6/site-packages/pandas/core/indexing.py in error()
1404                                        "key")
1405                 raise KeyError("the label [%s] is not in the [%s]" %
-> 1406                                (key, self.obj._get_axis_name(axis)))
1407
1408             try:


KeyError: 'the label [Phèdre] is not in the [index]'
```

What are each Phèdre, Macbeth, and Othello's most distinctive words? If you've read the text, does this confirm your opinion of it? Does it add anything new?

```
In [ ]: Macbeth get_mdw(dialogue_dict_macbeth, 'Macbeth')
        Othello get_mdw(dialogue_dict_othello , 'Othello')The most disinctive word for Phèdre=
```

If you've already taken Data 8, or your Python text parsing skills are already advanced, try this one:

I've placed two more text files in the data folder for the two remaining dramas Moretti plots: Friedrich Schiller's *Don Carlos* and Henrik Ibsen's *Ghosts*. Write some code to plot the character space!

```
In [ ]: !ls data
```

```
In [ ]: ## YOUR CODE HERE
```

```
In [ ]: plot_character_space(dialogue_dict_doncarlos)
        plot_character_space(dialogue_dict_ghosts)
```