day_three

March 1, 2016

1 Text Data

1.1 Pre-introduction

We'll be spending a lot of time today manipulating text. Make sure you remember how to split, join, and search strings.

1.2 Introduction

We've spent a lot of time in python dealing with text data, and that's because text data is everywhere. It is the primary form of communication between persons and persons, persons and computers, and computers and computers. The kind of inferential methods that we apply to text data, however, are different from those applied to tabular data.

This is partly because documents are typically specified in a way that expresses both structure and content using text (i.e. the document object model).

Largely, however, it's because text is difficult to turn into numbers in a way that preserves the information in the document. Today, we'll talk about dominant language model in NLP and the basics of how to implement it in Python.

1.2.1 The term-document model

This is also sometimes referred to as "bag-of-words" by those who don't think very highly of it. The term document model looks at language as individual communicative efforts that contain one or more tokens. The kind and number of the tokens in a document tells you something about what is attempting to be communicated, and the order of those tokens is ignored.

To start with, let's load a document.

```
In [1]: import nltk
        #nltk.download('webtext')
        document = nltk.corpus.webtext.open('grail.txt').read()
```

```
/Users/dillon/anaconda/lib/python3.5/site-packages/sklearn/utils/fixes.py:64: DeprecationWarning: inspe
  if 'order' in inspect.getargspec(np.copy)[0]:
```

Let's see what's in this document

```
In [2]: len(document.split('\n'))
```

```
Out[2]: 1192
```

```
In [3]: document.split('\n')[0:10]
```

```
Out[3]: ['SCENE 1: [wind] [clop clop clop] ',
         'KING ARTHUR: Whoa there!  [clop clop clop] ',
         'SOLDIER #1: Halt!  Who goes there?',
```

```
'ARTHUR: It is I, Arthur, son of Uther Pendragon, from the castle of Camelot.  King of the Bri
'SOLDIER #1: Pull the other one!',
'ARTHUR: I am, ...  and this is my trusty servant Patsy.  We have ridden the length and breadt
'SOLDIER #1: What?  Ridden on a horse?',
'ARTHUR: Yes!',
"SOLDIER #1: You're using coconuts!",
'ARTHUR: What?']
```

It looks like we've gotten ourselves a bit of the script from Monty Python and the Holy Grail. Note that when we are looking at the text, part of the structure of the document is written in tokens. For example, stage directions have been placed in brackets, and the names of the person speaking are in all caps.

1.3 Regular expressions

If we wanted to read out all of the stage directions for analysis, or just King Arthur's lines, doing so in base python string processing will be very difficult. Instead, we are going to use regular expressions. Regular expressions are a method for string manipulation that match patterns instead of bytes.

```
In [4]: import re
        snippet = "I fart in your general direction! Your mother was a hamster, and your father smelt o
        re.search(r'mother', snippet)
```

```
Out[4]: <_sre.SRE_Match object; span=(39, 45), match='mother'>
```

Just like with `str.find`, we can search for plain text. But `re` also gives us the option for searching for patterns of bytes - like only alphabetic characters.

```
In [5]: re.search(r'[a-z]', snippet)
```

```
Out[5]: <_sre.SRE_Match object; span=(2, 3), match='f'>
```

In this case, we've told re to search for the first sequence of bytes that is only composed of lowercase letters between `a` and `z`. We could get the letters at the end of each sentence by including a bang at the end of the pattern.

```
In [6]: re.search(r'[a-z]!', snippet)
```

```
Out[6]: <_sre.SRE_Match object; span=(31, 33), match='n!'>
```

If we wanted to pull out just the stage directions from the screenplay, we might try a pattern like this:

```
In [7]: re.findall(r'[a-zA-Z]', document)[0:10]
```

```
Out[7]: ['S', 'C', 'E', 'N', 'E', 'w', 'i', 'n', 'd', 'c']
```

So that's obviously no good. There are two things happening here:

1. `[` and `]` do not mean 'bracket'; they are special characters which mean 'any thing of this class'
2. we've only matched one letter each

A better regular expression, then, would wrap this in escaped brackets, and include a command saying more than one letter.

Re is flexible about how you specify numbers - you can match none, some, a range, or all repetitions of a sequence or character class.

character	meaning
{x}	exactly x repetitions
{x,y}	between x and y repetitions
?	0 or 1 repetition
*	0 or many repetitions
+	1 or many repetitions

```
In [8]: re.findall(r'\[[a-zA-Z]+\]', document)[0:10]
```

```
Out[8]: ['[wind]',
         '[thud]',
         '[clang]',
         '[clang]',
         '[clang]',
         '[clang]',
         '[clang]',
         '[clang]',
         '[clang]',
         '[clang]']
```

This is better, but it's missing that `[clop clop clop]` we saw above. This is because we told the regex engine to match any alphabetic character, but we did not specify whitespaces, commas, etc. to match these, we'll use the dot operator, which will match anything expect a newline.

Part of the power of regular expressions are their special characters. Common ones that you'll see are:

character	meaning
`.`	match anything except a newline
`^`	match the start of a line
`$`	match the end of a line
`\s`	matches any whitespace or newline

Finally, we need to fix this `+` character. It is a 'greedy' operator, which means it will match as much of the string as possible. To see why this is a problem, try:

```
In [9]: snippet = 'This is [cough cough] and example of a [really] greedy operator'
        re.findall(r'\[.+\]', snippet)
```

```
Out[9]: ['[cough cough] and example of a [really]']
```

Since the operator is greedy, it is matching everything inbetween the first open and the last close bracket. To make `+` consume the least possible amount of string, we'll add a `?`.

```
In [10]: p = re.compile(r'\[.+?\]')
         re.findall(p, document)[0:10]
```

```
Out[10]: ['[wind]',
          '[clop clop clop]',
          '[clop clop clop]',
          '[clop clop clop]',
          '[thud]',
          '[clang]',
          '[clang]',
          '[clang]',
          '[clang]',
          '[clang]']
```

What if we wanted to grab all of Arthur's speech? This one is a little trickier, since:

1. It is not conveniently bracketed; and,
2. We want to match on ARTHUR, but not to capture it

If we wanted to do this using base string manipulation, we would need to do something like:

```
split the document into lines
create a new list of just lines that start with ARTHUR
create a newer list with ARTHUR removed from the front of each element
```

Regex gives us a way of doing this in one line, by using something called groups. Groups are pieces of a pattern that can be ignored, negated, or given names for later retrieval.

character	meaning
`(x)`	match x
`(?:x)`	match x but don't capture it
`(?P<x>)`	match something and give it name x
`(?=x)`	match only if string is followed by x
`(?!x)`	match only if string is not followed by x

```
In [11]: p = re.compile(r'(?:ARTHUR: )(.+)')
         re.findall(p, document)[0:10]
```

```
Out[11]: ['Whoa there!  [clop clop clop] ',
          'It is I, Arthur, son of Uther Pendragon, from the castle of Camelot.  King of the Britons, de
          'I am, ...  and this is my trusty servant Patsy.  We have ridden the length and breadth of the
          'Yes!',
          'What?',
          'So?  We have ridden since the snows of winter covered this land, through the kingdom of Merc
          'We found them.',
          'What do you mean?',
          'The swallow may fly south with the sun or the house martin or the plover may seek warmer cli
          'Not at all.  They could be carried.']
```

Because we are using `findall`, the regex engine is capturing and returning the normal groups, but not the non-capturing group. For complicated, multi-piece regular expressions, you may need to pull groups out separately. You can do this with names.

```
In [12]: p = re.compile(r'(?P<name>[A-Z ]+)(?::)(?P<line>.+)')
         match = re.search(p, document)
         match
```

```
Out[12]: <_sre.SRE_Match object; span=(34, 77), match='KING ARTHUR: Whoa there!  [clop clop clop] '>
```

```
In [13]: match.group('name'), match.group('line')
```

```
Out[13]: ('KING ARTHUR', ' Whoa there!  [clop clop clop] ')
```

Now let's try a small challenge! To check that you've understood something about regular expressions, we're going to have you do a small test challenge. Partner up with the person next to you - we're going to do this as a pair coding exercise - and choose which computer you are going to use.

Then, navigate to `challenges/03_analysis/` and read through challenge A. When you think you've completed it successfully, run `py.test test_A.py` .

1.4 Tokenizing

Let's grab Arthur's speech from above, and see what we can learn about Arthur from it.

```
In [14]: p = re.compile(r'(?:ARTHUR: )(.+)')
         arthur = ' '.join(re.findall(p, document))
         arthur[0:100]
```

```
Out[14]: 'Whoa there!  [clop clop clop]  It is I, Arthur, son of Uther Pendragon, from the castle of Ca
```

In our model for natural language, we're interested in words. The document is currently a continuous string of bytes, which isn't ideal. You might be tempted to separate this into words using your newfound regex knowledge:

```
In [15]: p = re.compile(r'\w+', flags=re.I)
         re.findall(p, arthur)[0:10]
```

```
Out[15]: ['Whoa', 'there', 'clop', 'clop', 'clop', 'It', 'is', 'I', 'Arthur', 'son']
```

But this is problematic for languages that make extensive use of punctuation. For example, see what happens with:

```
In [16]: re.findall(p, "It isn't Dav's cheesecake that I'm worried about")
```

```
Out[16]: ['It',
          'isn',
          't',
          'Dav',
          's',
          'cheesecake',
          'that',
          'I',
          'm',
          'worried',
          'about']
```

The practice of pulling apart a continuous string into units is called "tokenizing", and it creates "tokens". NLTK, the canonical library for NLP in Python, has a couple of implementations for tokenizing a string into words.

```
In [17]: from nltk import word_tokenize
         word_tokenize("It isn't Dav's cheesecake that I'm worried about")
```

```
Out[17]: ['It',
          'is',
          "n't",
          'Dav',
          "'s",
          'cheesecake',
          'that',
          'I',
          "'m",
          'worried',
          'about']
```

The distinction here is subtle, but look at what happened to "isn't". It's been separated into "IS" and "N'T", which is more in keeping with the way contractions work in English.

```
In [18]: tokens = word_tokenize(arthur)
         tokens[0:10]
```

```
Out[18]: ['Whoa', 'there', '!', '[', 'clop', 'clop', 'clop', ']', 'It', 'is']
```

At this point, we can start asking questions like what are the most common words, and what words tend to occur together.

```
In [19]: len(tokens), len(set(tokens))
```

```
Out[19]: (2393, 596)
```

So we can see right away that Arthur is using the same words a whole bunch - on average, each unique word is used four times. This is typical of natural language.

> Not necessarily the value, but that the number of unique words in any corpus increases much more slowly than the total number of words.
>
> A corpus with 100M tokens, for example, probably only has 100,000 unique tokens in it.

For more complicated metrics, it's easier to use NLTK's classes and methods.

```
In [20]: from nltk import collocations
         fd = collocations.FreqDist(tokens)
         fd.most_common()[:10]
```

```
Out[20]: [(',', 135),
          ('.', 129),
          ('!', 119),
          ('the', 70),
          ('?', 61),
          ('you', 51),
          ('of', 45),
          (']', 38),
          ('[', 38),
          ('I', 34)]
```

```
In [21]: measures = collocations.BigramAssocMeasures()
         c = collocations.BigramCollocationFinder.from_words(tokens)
         c.nbest(measures.pmi, 10)
```

```
Out[21]: [("'Til", 'Recently'),
          ('ARTHUR', 'chops'),
          ('An', 'African'),
          ('BLACK', 'KNIGHT'),
          ('Bloody', 'peasant'),
          ('Castle', 'Aaagh'),
          ('Chop', 'his'),
          ('Cut', 'down'),
          ('Divine', 'Providence'),
          ('Eternal', 'Peril')]
```

```
In [22]: c.nbest(measures.likelihood_ratio, 10)
```

```
Out[22]: [('I', 'am'),
          ('Well', ','),
          ('boom', 'boom'),
          ('Run', 'away'),
          ('of', 'the'),
          ('Holy', 'Grail'),
          (']', '['),
          ('Brother', 'Maynard'),
          ('Jesus', 'Christ'),
          ('Round', 'Table')]
```

We see here that the collocation finder is pulling out some things that have face validity. When Arthur is talking about peasants, he calls them “bloody” more often than not. However, collocations like “Brother Maynard” and “BLACK KNIGHT” are less informative to us, because we know that they are proper names.

If you were interested in collocations in particular, what step do you think you would have to take during the tokenizing process?

1.5 Stemming

This has gotten us as far identical tokens, but in language processing, it is often the case that the specific form of the word is not as important as the idea to which it refers. For example, if you are trying to identify the topic of a document, counting ‘running’, ‘runs’, ‘ran’, and ‘run’ as four separate words is not useful. Reducing words to their stems is a process called stemming.

A popular stemming implementation is the Snowball Stemmer, which is based on the Porter Stemmer. It’s algorithm looks at word forms and does things like drop final ’s’s, ’ed’s, and ’ing’s.

Just like the tokenizers, we first have to create a stemmer object with the language we are using.

```
In [23]: snowball = nltk.SnowballStemmer('english')
```

Now, we can try stemming some words

```
In [24]: snowball.stem('running')

Out[24]: 'run'

In [25]: snowball.stem('eats')

Out[25]: 'eat'

In [26]: snowball.stem('embarassed')

Out[26]: 'embarass'
```

Snowball is a very fast algorithm, but it has a lot of edge cases. In some cases, words with the same stem are reduced to two different stems.

```
In [27]: snowball.stem('cylinder'), snowball.stem('cylindrical')

Out[27]: ('cylind', 'cylindr')
```

In other cases, two different words are reduced to the same stem.

This is sometimes referred to as a ‘collision’

```
In [28]: snowball.stem('vacation'), snowball.stem('vacate')

Out[28]: ('vacat', 'vacat')

In [29]: snowball.stem('organization'), snowball.stem('organ')

Out[29]: ('organ', 'organ')

In [30]: snowball.stem('iron'), snowball.stem('ironic')

Out[30]: ('iron', 'iron')

In [31]: snowball.stem('vertical'), snowball.stem('vertices')

Out[31]: ('vertic', 'vertic')
```

A more accurate approach is to use an English word bank like WordNet to call dictionary lookups on word forms, in a process called lemmatization.

```
In [32]: # nltk.download('wordnet')
         wordnet = nltk.WordNetLemmatizer()

In [33]: wordnet.lemmatize('iron'), wordnet.lemmatize('ironic')

Out[33]: ('iron', 'ironic')

In [34]: wordnet.lemmatize('vacation'), wordnet.lemmatize('vacate')

Out[34]: ('vacation', 'vacate')
```

Nothing comes for free, and you've probably noticed already that the lemmatizer is slower. We can see how much slower with one of IPYthon's `magic functions`.

```
In [35]: %timeit wordnet.lemmatize('table')

100000 loops, best of 3: 5.45 s per loop

In [36]: 4.45 * 5.12

Out[36]: 22.784000000000002

In [37]: %timeit snowball.stem('table')

100000 loops, best of 3: 16.8 s per loop
```

Time for another small challenge! Switch computers for this one, so that you are using your partner's computer, and try your hand at challenge B!

1.6 Sentiment

Frequently, we are interested in text to learn something about the person who is speaking. One of these things we've talked about already - linguistic diversity. A similar metric was used a couple of years ago to settle the question of who has the largest vocabulary in Hip Hop.

> Unsurprisingly, top spots go to Canibus, Aesop Rock, and the Wu Tang Clan. E-40 is also in the top 20, but mostly because he makes up a lot of words; as are OutKast, who print their lyrics with words slurred in the actual typography

Another thing we can learn is about how the speaker is feeling, with a process called sentiment analysis. Before we start, be forewarned that this is not a robust method by any stretch of the imagination. Sentiment classifiers are often trained on product reviews, which limits their ecological validity.

We're going to use TextBlob's built-in sentiment classifier, because it is super easy.

```
In [38]: from textblob import TextBlob

In [39]: blob = TextBlob(arthur)

In [40]: for sentence in blob.sentences[10:25]:
             print(sentence.sentiment.polarity, sentence)
```

```
-0.3125 What do you mean?
0.8 The swallow may fly south with the sun or the house martin or the plover may seek warmer climes in w
0.0 Not at all.
0.0 They could be carried.
0.0 It could grip it by the husk!
0.0 Well, it doesn't matter.
0.0 Will you go and tell your master that Arthur from the Court of Camelot is here.
0.0 Please!
-0.15625 I'm not interested!
0.25 Will you ask your master if he wants to join my court at Camelot?!
0.125 Old woman!
0.0 Man.
-0.5 Sorry.
0.13636363636363635 What knight live in that castle over there?
0.0 I-- what?
```

1.7 Semantic distance

Another common NLP task is to look for semantic distance between documents. This is used by search engines like Google (along with other things like PageRank) to decide which websites to show you when you search for things like 'bike' versus 'motorcycle'.

It is also used to cluster documents into topics, in a process called topic modeling. The math behind this is beyond the scope of this course, but the basic strategy is to represent each document as a one-dimensional array, where the indices correspond to integer ids of tokens in the document. Then, some measure of semantic similarity, like the cosine of the angle between unitized versions of the document vectors, is calculated.

Luckily for us there is another python library that takes care of the heavy lifting for us.

```
In [41]: from gensim import corpora, models, similarities
```

We already have a document for Arthur, but let's grab the text from someone else to compare it with.

```
In [42]: p = re.compile(r'(?:GALAHAD: )(.+)')
         galahad = ' '.join(re.findall(p, document))
         arthur_tokens = tokens
         galahad_tokens = word_tokenize(galahad)
```

Now, we use gensim to create vectors from these tokenized documents:

```
In [43]: dictionary = corpora.Dictionary([arthur_tokens, galahad_tokens])
         corpus = [dictionary.doc2bow(doc) for doc in [arthur_tokens, galahad_tokens]]
         tfidf = models.TfidfModel(corpus, id2word=dictionary)
```

Then, we create matrix models of our corpus and query

```
In [44]: query = tfidf[dictionary.doc2bow(['peasant'])]
         index = similarities.MatrixSimilarity(tfidf[corpus])

WARNING:gensim.similarities.docsim:scanning corpus to determine the number of features (consider setting
```

And finally, we can test our query, "peasant" on the two documents in our corpus

```
In [45]: list(enumerate(index[query]))

Out[45]: [(0, 0.017683197), (1, 0.0)]
```

So we see here that "peasant" does not match Galahad very well (a really bad match would have a negative value), and is more similar to the kind of speach output that we see from King Arthur.

2 Tabular data

In data storage, data visualization, inferential statistics, and machine learning, the most common way to pass data between applications is in the form of tables (these are called tabular, structured, or rectangular data). These are convenient in that, when used correctly, they store data in a DRY and easily queryable way, and are also easily turned into matrices for numeric processing.

> note - it is sometimes tempting to refer to N-dimensional matrices as arrays, following the numpy naming convention, but these are not the same as arrays in C++ or Java, which may cause confusion

It is common in enterprise applications to store tabular data in a SQL database. In the sciences, data is typically passed around as comma separated value files (.csv), which you have already been dealing with over the course of the last two days.

For this brief introduction to analyzing tabular data, we'll be using the scipy stack, which includes numpy, pandas, scipy, and "scikits" like sk-learn and sk-image.

```
In [46]: import pandas as pd
```

You might not have seen this `as` convention yet. It is just telling python that when we import `pandas`, we don't want to access it in the namespace as `pandas` but as `pd` instead.

2.1 Pandas basics

We'll start by making a small table to practice on. Tables in pandas are called data frames, so we'll start by making an instance of class `DataFrame`, and initialize it with some data.

> note - pandas and R use the same name for their tables, but their behavior is often very different

```
In [47]: table = pd.DataFrame({'id': [1,2,3], 'name':['dillon','juan','andrew'], 'age':[47,27,23]})
         print(table)
```

```
age  id    name
0    47   1  dillon
1    27   2    juan
2    23   3  andrew
```

Variables in pandas are represented by a pandas-specific data structure, called a `Series`. You can grab a `Series` out of a `DataFrame` by using the slicing operator with the name of the variable that you want to pull.

```
In [48]: table['name'], type(table['name'])
```

```
Out[48]: (0    dillon
          1      juan
          2    andrew
          Name: name, dtype: object, pandas.core.series.Series)
```

We could have made each variable a `Series`, and then put it into the DataFrame object, but it's easier in this instance to pass in a dictionary where the keys are variable names and the values are lists. You can also modify a data frame in place using similar syntax:

```
In [49]: table['fingers'] = [9, 10, None]
```

If you try to run that code without the `None` there, pandas will return an error. In a table (in any language) each column must have the same number of rows.

We've entered `None`, base python's missingness indicator, but pandas is going to swap this out with something else:

```
In [50]: table['fingers']

Out[50]: 0     9
         1    10
         2   NaN
         Name: fingers, dtype: float64
```

You might be tempted to write your own control structures around these missing values (which are variably called `NaN`, `nan`, and `NA`), but this is always a bad idea:

```
In [51]: table['fingers'][2] == None

Out[51]: False

In [52]: table['fingers'][2] == 'NaN'

Out[52]: False

In [53]: type(table['fingers'][2]) == str

Out[53]: False
```

None of this works because the pandas `NaN` is a subclass of numpy's double precision floating point number. However, for ambiguous reasons, even numpy.nan does not evaluate as being equal to itself.

To handle missing data, you'll need to use the pandas method `isnull`.

```
In [54]: pd.isnull(table['fingers'])

Out[54]: 0    False
         1    False
         2     True
         Name: fingers, dtype: bool
```

In the same way that we've been pulling out columns by name, you can pull out rows by index. If I want to grab the first row, I can use:

```
In [55]: table[:1]

Out[55]:    age  id    name  fingers
         0   47   1  dillon        9
```

Recall that indices in python start at zero, and that selecting by a range does not include the final value (i.e. `[ , )`).

Unlike other software languages (R, I'm looking at you here), row indices in pandas are immutable. So, if I rearrange my data, the index also get shuffled.

```
In [56]: table.sort_values('age')

Out[56]:    age  id    name  fingers
         2   23   3  andrew      NaN
         1   27   2    juan       10
         0   47   1  dillon        9
```

Because of this, it's common to set the index to be something like a timestamp or UUID.

We can select parts of a `DataFrame` with conditional statements:

```
In [57]: table[table['age'] < 40]

Out[57]:    age  id    name  fingers
         1   27   2    juan       10
         2   23   3  andrew      NaN
```

2.2 Merging tables

As you might expect, tables in pandas can also be merged by keys. So, if we make a new dataset that shares an attribute in common:

```
In [58]: other_table = pd.DataFrame({
                  'name':['dav', 'juan', 'dillon'],
                  'languages':['python','python','python']})
```

```
In [59]: table.merge(other_table, on='name')
```

```
Out[59]:    age  id    name  fingers languages
         0   47   1  dillon        9    python
         1   27   2    juan       10    python
```

Note that we have done an "inner join" here, which means we are only getting the intersection of the two tables. If we want the union, we can specify that we want an outer join:

```
In [60]: table.merge(other_table, on='name', how='outer')
```

```
Out[60]:    age  id    name  fingers languages
         0   47   1  dillon        9    python
         1   27   2    juan       10    python
         2   23   3  andrew      NaN       NaN
         3  NaN NaN     dav      NaN    python
```

Or maybe we want all of the data from `table`, but not `other_table`

```
In [61]: table.merge(other_table, on='name', how='left')
```

```
Out[61]:    age  id    name  fingers languages
         0   47   1  dillon        9    python
         1   27   2    juan       10    python
         2   23   3  andrew      NaN       NaN
```

2.3 Reshaping

To make analysis easier, you may have to reshape your data. It's easiest to deal with data when each table meets the follwing criteria:

1. Each row is exactly one observation
2. Each column is exactly one kind of data
3. The table expresses one and only one relationship between observations and variables

This kind of format is easy to work with, because:

1. It's easy to update when every piece of data exists in one and only one place
2. It's easy to subset conditionally across rows
3. It's easy to test across columns

To make this more concrete, let's take an example table.

	name	city1	city2	population
dillon	williamsburg	berkeley	110	
juan	berkeley	berkeley	110	
dav	cambridge	berkeley	110	

This table violates all three of the rules above. Specifically, it:

1. each row is about two observations
2. two columns are about the same kind of date (city), while another datatype (time) has been hidden in the column names
3. it expresses the relationship between people and where they live; and, cities and their population

In this particular example, our data is too wide. If we create that dataframe in pandas

In [62]:
```
wide_table = pd.DataFrame({'name' : ['dillon', 'juan', 'dav'],
                           'city1' : ['williamsburg', 'berkeley', 'cambridge'],
                           'city2' : ['berkeley', 'berkeley', 'berkeley'],
                           'population' : [110, 110, 110]
                          })
wide_table
```

Out[62]:
```
          city1     city2    name  population
0  williamsburg  berkeley  dillon         110
1      berkeley  berkeley    juan         110
2     cambridge  berkeley     dav         110
```

We can make this longer in pandas using the `melt` function

In [63]:
```
long_table = pd.melt(wide_table, id_vars = ['name'])
long_table
```

Out[63]:
```
     name    variable         value
0  dillon       city1  williamsburg
1    juan       city1      berkeley
2     dav       city1     cambridge
3  dillon       city2      berkeley
4    juan       city2      berkeley
5     dav       city2      berkeley
6  dillon  population           110
7    juan  population           110
8     dav  population           110
```

We can make the table wider using the pivot method

> side note - this kind of inconsistency between `melt` and `pivot` is un-pythonic and should not be emulated

In [64]:
```
long_table.pivot(columns='variable')
```

Out[64]:
```
            name                                value
variable   city1   city2 population          city1     city2 population
0         dillon     NaN        NaN   williamsburg       NaN        NaN
1           juan     NaN        NaN       berkeley       NaN        NaN
2            dav     NaN        NaN      cambridge       NaN        NaN
3            NaN  dillon        NaN            NaN  berkeley        NaN
4            NaN    juan        NaN            NaN  berkeley        NaN
5            NaN     dav        NaN            NaN  berkeley        NaN
6            NaN     NaN     dillon            NaN       NaN        110
7            NaN     NaN       juan            NaN       NaN        110
8            NaN     NaN        dav            NaN       NaN        110
```

WHOA

One of the really cool things about pandas is that it allows you to have multiple indexes for rows and columns. Since pandas couldn't figure out what do with two kinds of value variables, it doubled up our column index. We can fix this by specifying that we only want the 'values' values

```
In [65]: long_table.pivot(columns='variable', values='value')

Out[65]: variable          city1     city2 population
         0           williamsburg       NaN        NaN
         1               berkeley       NaN        NaN
         2              cambridge       NaN        NaN
         3                    NaN  berkeley        NaN
         4                    NaN  berkeley        NaN
         5                    NaN  berkeley        NaN
         6                    NaN       NaN        110
         7                    NaN       NaN        110
         8                    NaN       NaN        110
```

Challenge time! Switch computers *again* so that you are working on the first computer of the day, and have a look at challenge C. This will have you practice reading and merging tables. Again, when you are finished, check your work by running `py.test test_C` in a shell.

2.4 Descriptive statistics

Single descriptives have their own method calls in the `Series` class.

```
In [66]: table['fingers'].mean()

Out[66]: 9.5

In [67]: table['fingers'].std()

Out[67]: 0.70710678118654757

In [68]: table['fingers'].quantile(.25)

Out[68]: 9.25

In [69]: table['fingers'].kurtosis()

Out[69]: nan
```

You can call several of these at once with the `describe` method

```
In [70]: table.describe()

Out[70]:              age   id    fingers
         count   3.000000  3.0   2.000000
         mean   32.333333  2.0   9.500000
         std    12.858201  1.0   0.707107
         min    23.000000  1.0   9.000000
         25%    25.000000  1.5   9.250000
         50%    27.000000  2.0   9.500000
         75%    37.000000  2.5   9.750000
         max    47.000000  3.0  10.000000
```

2.5 Inferential statistics

pandas does not have statistical functions baked in, so we are going to call them from the `scipy.stats` library and the `statmodels` scikit.

We are also going to load in an actual dataset, as stats examples aren't very interesting with tiny bits of fake data.

```
In [71]: from scipy import stats
         data = pd.read_csv('../data/03_feedback.csv')
```

Using what you've learned so far about manipulating pandas objects, how would you find out the names of the variables in this dataset? Their datatypes? The distribution of their values?

2.5.1 Comparisons of group means

A common statistical procedure is to look for differences between groups of values. Typically, the values are grouped by a variable of interest, like sex or age. Here, we are going to compare the barriers of access to technology that people experience in the D-Lab compared to the world outside.

If you only have two groups in your sample, you can use a t-test:

```
In [72]: i = data['inside.barriers'].dropna()
         o = data['outside.barriers'].dropna()
         stats.ttest_ind(i, o)
```

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

        KeyError                                  Traceback (most recent call last)

        <ipython-input-72-4bf9b8c7e60c> in <module>()
    ----> 1 i = data['inside.barriers'].dropna()
          2 o = data['outside.barriers'].dropna()
          3 stats.ttest_ind(i, o)


        /Users/dillon/anaconda/lib/python3.5/site-packages/pandas/core/frame.py in __getitem__(self, key)
       1912             return self._getitem_multilevel(key)
       1913         else:
    -> 1914             return self._getitem_column(key)
       1915
       1916     def _getitem_column(self, key):


        /Users/dillon/anaconda/lib/python3.5/site-packages/pandas/core/frame.py in _getitem_column(self,
       1919         # get column
       1920         if self.columns.is_unique:
    -> 1921             return self._get_item_cache(key)
       1922
       1923         # duplicate columns & possible reduce dimensionaility


        /Users/dillon/anaconda/lib/python3.5/site-packages/pandas/core/generic.py in _get_item_cache(sel
       1088         res = cache.get(item)
       1089         if res is None:
    -> 1090             values = self._data.get(item)
       1091             res = self._box_item_values(item, values)
```

```
   1092             cache[item] = res


    /Users/dillon/anaconda/lib/python3.5/site-packages/pandas/core/internals.py in get(self, item, f
   3100
   3101             if not isnull(item):
-> 3102                 loc = self.items.get_loc(item)
   3103             else:
   3104                 indexer = np.arange(len(self.items))[isnull(self.items)]


    /Users/dillon/anaconda/lib/python3.5/site-packages/pandas/core/index.py in get_loc(self, key, me
   1690                 raise ValueError('tolerance argument only valid if using pad, '
   1691                                  'backfill or nearest lookups')
-> 1692             return self._engine.get_loc(_values_from_object(key))
   1693
   1694         indexer = self.get_indexer([key], method=method,


    pandas/index.pyx in pandas.index.IndexEngine.get_loc (pandas/index.c:3979)()


    pandas/index.pyx in pandas.index.IndexEngine.get_loc (pandas/index.c:3843)()


    pandas/hashtable.pyx in pandas.hashtable.PyObjectHashTable.get_item (pandas/hashtable.c:12265)()


    pandas/hashtable.pyx in pandas.hashtable.PyObjectHashTable.get_item (pandas/hashtable.c:12216)()


    KeyError: 'inside.barriers'
```

Notice that here, we are passing in two whole columns, but we could also be subsetting by some other factor.

If you have more than two groups (or levels) that you would like to compare, you'll have to use something like an ANOVA:

```
In [73]: m = data[data.gender == "Male/Man"]['outside.barriers'].dropna()
         f = data[data.gender == "Female/Woman"]['outside.barriers'].dropna()
         q = data[data.gender == "Genderqueer/Gender non-conforming"]['outside.barriers'].dropna()
         stats.f_oneway(m, f, q)
```

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

    KeyError                                  Traceback (most recent call last)

    <ipython-input-73-557bc7cc4363> in <module>()
----> 1 m = data[data.gender == "Male/Man"]['outside.barriers'].dropna()
      2 f = data[data.gender == "Female/Woman"]['outside.barriers'].dropna()
      3 q = data[data.gender == "Genderqueer/Gender non-conforming"]['outside.barriers'].dropna()
      4 stats.f_oneway(m, f, q)
```

```
/Users/dillon/anaconda/lib/python3.5/site-packages/pandas/core/frame.py in __getitem__(self, key)
   1912             return self._getitem_multilevel(key)
   1913         else:
-> 1914             return self._getitem_column(key)
   1915
   1916     def _getitem_column(self, key):


/Users/dillon/anaconda/lib/python3.5/site-packages/pandas/core/frame.py in _getitem_column(self,
   1919         # get column
   1920         if self.columns.is_unique:
-> 1921             return self._get_item_cache(key)
   1922
   1923         # duplicate columns & possible reduce dimensionaility


/Users/dillon/anaconda/lib/python3.5/site-packages/pandas/core/generic.py in _get_item_cache(sel
   1088         res = cache.get(item)
   1089         if res is None:
-> 1090             values = self._data.get(item)
   1091             res = self._box_item_values(item, values)
   1092             cache[item] = res


/Users/dillon/anaconda/lib/python3.5/site-packages/pandas/core/internals.py in get(self, item, f
   3100
   3101             if not isnull(item):
-> 3102                 loc = self.items.get_loc(item)
   3103             else:
   3104                 indexer = np.arange(len(self.items))[isnull(self.items)]


/Users/dillon/anaconda/lib/python3.5/site-packages/pandas/core/index.py in get_loc(self, key, me
   1690                 raise ValueError('tolerance argument only valid if using pad, '
   1691                                  'backfill or nearest lookups')
-> 1692             return self._engine.get_loc(_values_from_object(key))
   1693
   1694         indexer = self.get_indexer([key], method=method,


pandas/index.pyx in pandas.index.IndexEngine.get_loc (pandas/index.c:3979)()


pandas/index.pyx in pandas.index.IndexEngine.get_loc (pandas/index.c:3843)()


pandas/hashtable.pyx in pandas.hashtable.PyObjectHashTable.get_item (pandas/hashtable.c:12265)()


pandas/hashtable.pyx in pandas.hashtable.PyObjectHashTable.get_item (pandas/hashtable.c:12216)()


KeyError: 'outside.barriers'
```

Linear relationships Another common task is to establish if/how two variables are related across linear space. This could be something, for example, like relating shoe size to height. Here, we are going to ask whether barriers to access to technology inside and outside of the D-Lab are related.

One implementation of linear relationships is correlation testing:

```
In [74]: intermediate = data.dropna(subset=['inside.barriers', 'outside.barriers'])
         stats.pearsonr(intermediate['outside.barriers'], intermediate['inside.barriers'])
```

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

        KeyError                                  Traceback (most recent call last)

        <ipython-input-74-bea4991243a0> in <module>()
    ----> 1 intermediate = data.dropna(subset=['inside.barriers', 'outside.barriers'])
          2 stats.pearsonr(intermediate['outside.barriers'], intermediate['inside.barriers'])


        /Users/dillon/anaconda/lib/python3.5/site-packages/pandas/core/frame.py in dropna(self, axis, h
       2899                 check = indices == -1
       2900                 if check.any():
    -> 2901                     raise KeyError(list(np.compress(check,subset)))
       2902                 agg_obj = self.take(indices,axis=agg_axis)
       2903


        KeyError: ['inside.barriers', 'outside.barriers']
```

At this point, we're going to pivot to using `statsmodels`

```
In [75]: import statsmodels.formula.api as smf
```

The formulas module in statsmodels lets us work with pandas dataframes, and linear model specifications that are similar to R and other variants of statistical software, e.g.:

```
outcome ~ var1 + var2
```

```
In [76]: model_1 = smf.ols("inside_barriers ~ outside_barriers", data=data).fit()
         model_1

Out[76]: <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x10fc06828>
```

To get a summary of the test results, call the model's `summary` method

```
In [77]: model_1.summary()

Out[77]: <class 'statsmodels.iolib.summary.Summary'>
         """
                                     OLS Regression Results
         ==============================================================================
         Dep. Variable:        inside_barriers   R-squared:                       0.215
         Model:                            OLS   Adj. R-squared:                  0.214
         Method:                 Least Squares   F-statistic:                     242.0
```

```
Date:                Tue, 01 Mar 2016   Prob (F-statistic):          2.08e-48
Time:                        16:39:18   Log-Likelihood:               -807.74
No. Observations:                 884   AIC:                            1619.
Df Residuals:                     882   BIC:                            1629.
Df Model:                           1
Covariance Type:            nonrobust
=====================================================================================
                      coef    std err          t      P>|t|      [95.0% Conf. Int.]
-------------------------------------------------------------------------------------
Intercept           0.7529      0.038     19.599      0.000         0.678     0.828
outside_barriers    0.2464      0.016     15.558      0.000         0.215     0.277
==============================================================================
Omnibus:                      389.637   Durbin-Watson:                   1.865
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1871.839
Skew:                           2.026   Prob(JB):                         0.00
Kurtosis:                       8.865   Cond. No.                         5.17
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
"""
```

Since Python does not have private data or hidden attributes, you can pull out just about any intermediate information you want, including coefficients, residuals, and eigenvalues

Raymond Hettinger would say that Python is a "consenting adult language"

In [78]:
```
model_1.params['outside_barriers']
```

Out[78]:
```
0.24638149915096663
```

`statsmodels` also exposes methods for validity checking your regressions, like looking for outliers by influence statistics

In [79]:
```
model_1.get_influence().summary_frame()
```

Out[79]:
```
     dfb_Intercept  dfb_outside_barriers      cooks_d    dffits  \
0        -0.007772              0.000628  9.397542e-05 -0.013703
1         0.000048             -0.000032  1.258195e-09  0.000050
2         0.000048             -0.000032  1.258195e-09  0.000050
3         0.002573             -0.020131  5.793345e-04 -0.034033
4         0.000048             -0.000032  1.258195e-09  0.000050
5         0.000048             -0.000032  1.258195e-09  0.000050
6         0.000048             -0.000032  1.258195e-09  0.000050
8         0.000048             -0.000032  1.258195e-09  0.000050
9         0.000048             -0.000032  1.258195e-09  0.000050
10        0.031181             -0.062463  2.801289e-03 -0.074872
11        0.000048             -0.000032  1.258195e-09  0.000050
12        0.000048             -0.000032  1.258195e-09  0.000050
13        0.000048             -0.000032  1.258195e-09  0.000050
14        0.000048             -0.000032  1.258195e-09  0.000050
16        0.000048             -0.000032  1.258195e-09  0.000050
18        0.002573             -0.020131  5.793345e-04 -0.034033
20        0.002573             -0.020131  5.793345e-04 -0.034033
21        0.000048             -0.000032  1.258195e-09  0.000050
22        0.000048             -0.000032  1.258195e-09  0.000050
```

23	0.031181	-0.062463	2.801289e-03	-0.074872
24	0.002573	-0.020131	5.793345e-04	-0.034033
26	0.000048	-0.000032	1.258195e-09	0.000050
27	0.000048	-0.000032	1.258195e-09	0.000050
28	0.023881	-0.001929	8.858263e-04	0.042104
29	0.023881	-0.001929	8.858263e-04	0.042104
30	0.031181	-0.062463	2.801289e-03	-0.074872
32	0.031181	-0.062463	2.801289e-03	-0.074872
33	0.000048	-0.000032	1.258195e-09	0.000050
34	-0.007772	0.000628	9.397542e-05	-0.013703
35	0.000048	-0.000032	1.258195e-09	0.000050
...	...	...	...	...
1032	0.000048	-0.000032	1.258195e-09	0.000050
1033	-0.007772	0.000628	9.397542e-05	-0.013703
1034	0.023881	-0.001929	8.858263e-04	0.042104
1035	0.000048	-0.000032	1.258195e-09	0.000050
1036	0.002573	-0.020131	5.793345e-04	-0.034033
1037	0.000048	-0.000032	1.258195e-09	0.000050
1038	0.000048	-0.000032	1.258195e-09	0.000050
1039	-0.007772	0.000628	9.397542e-05	-0.013703
1040	0.000048	-0.000032	1.258195e-09	0.000050
1041	0.000048	-0.000032	1.258195e-09	0.000050
1042	0.031181	-0.062463	2.801289e-03	-0.074872
1043	0.002573	-0.020131	5.793345e-04	-0.034033
1044	-0.002656	0.020780	6.172843e-04	0.035131
1045	0.000048	-0.000032	1.258195e-09	0.000050
1046	-0.007772	0.000628	9.397542e-05	-0.013703
1047	0.000048	-0.000032	1.258195e-09	0.000050
1048	-0.007772	0.000628	9.397542e-05	-0.013703
1049	0.000048	-0.000032	1.258195e-09	0.000050
1050	0.002573	-0.020131	5.793345e-04	-0.034033
1051	0.055758	-0.004503	4.791375e-03	0.098308
1052	0.000048	-0.000032	1.258195e-09	0.000050
1053	0.000048	-0.000032	1.258195e-09	0.000050
1054	0.000048	-0.000032	1.258195e-09	0.000050
1055	0.031181	-0.062463	2.801289e-03	-0.074872
1056	0.002573	-0.020131	5.793345e-04	-0.034033
1057	0.000048	-0.000032	1.258195e-09	0.000050
1058	0.031181	-0.062463	2.801289e-03	-0.074872
1059	-0.007772	0.000628	9.397542e-05	-0.013703
1060	0.000048	-0.000032	1.258195e-09	0.000050
1061	0.000048	-0.000032	1.258195e-09	0.000050

	dffits_internal	hat_diag	standard_resid	student_resid
0	-0.013710	0.001134	-0.406955	-0.406762
1	0.000050	0.001902	0.001149	0.001149
2	0.000050	0.001902	0.001149	0.001149
3	-0.034039	0.001740	-0.815305	-0.815150
4	0.000050	0.001902	0.001149	0.001149
5	0.000050	0.001902	0.001149	0.001149
6	0.000050	0.001902	0.001149	0.001149
8	0.000050	0.001902	0.001149	0.001149
9	0.000050	0.001902	0.001149	0.001149
10	-0.074850	0.003721	-1.224749	-1.225096

11	0.000050	0.001902	0.001149	0.001149
12	0.000050	0.001902	0.001149	0.001149
13	0.000050	0.001902	0.001149	0.001149
14	0.000050	0.001902	0.001149	0.001149
16	0.000050	0.001902	0.001149	0.001149
18	-0.034039	0.001740	-0.815305	-0.815150
20	-0.034039	0.001740	-0.815305	-0.815150
21	0.000050	0.001902	0.001149	0.001149
22	0.000050	0.001902	0.001149	0.001149
23	-0.074850	0.003721	-1.224749	-1.225096
24	-0.034039	0.001740	-0.815305	-0.815150
26	0.000050	0.001902	0.001149	0.001149
27	0.000050	0.001902	0.001149	0.001149
28	0.042091	0.001134	1.249433	1.249831
29	0.042091	0.001134	1.249433	1.249831
30	-0.074850	0.003721	-1.224749	-1.225096
32	-0.074850	0.003721	-1.224749	-1.225096
33	0.000050	0.001902	0.001149	0.001149
34	-0.013710	0.001134	-0.406955	-0.406762
35	0.000050	0.001902	0.001149	0.001149
...	...	...	...	...
1032	0.000050	0.001902	0.001149	0.001149
1033	-0.013710	0.001134	-0.406955	-0.406762
1034	0.042091	0.001134	1.249433	1.249831
1035	0.000050	0.001902	0.001149	0.001149
1036	-0.034039	0.001740	-0.815305	-0.815150
1037	0.000050	0.001902	0.001149	0.001149
1038	0.000050	0.001902	0.001149	0.001149
1039	-0.013710	0.001134	-0.406955	-0.406762
1040	0.000050	0.001902	0.001149	0.001149
1041	0.000050	0.001902	0.001149	0.001149
1042	-0.074850	0.003721	-1.224749	-1.225096
1043	-0.034039	0.001740	-0.815305	-0.815150
1044	0.035136	0.001740	0.841585	0.841446
1045	0.000050	0.001902	0.001149	0.001149
1046	-0.013710	0.001134	-0.406955	-0.406762
1047	0.000050	0.001902	0.001149	0.001149
1048	-0.013710	0.001134	-0.406955	-0.406762
1049	0.000050	0.001902	0.001149	0.001149
1050	-0.034039	0.001740	-0.815305	-0.815150
1051	0.097892	0.001134	2.905821	2.918175
1052	0.000050	0.001902	0.001149	0.001149
1053	0.000050	0.001902	0.001149	0.001149
1054	0.000050	0.001902	0.001149	0.001149
1055	-0.074850	0.003721	-1.224749	-1.225096
1056	-0.034039	0.001740	-0.815305	-0.815150
1057	0.000050	0.001902	0.001149	0.001149
1058	-0.074850	0.003721	-1.224749	-1.225096
1059	-0.013710	0.001134	-0.406955	-0.406762
1060	0.000050	0.001902	0.001149	0.001149
1061	0.000050	0.001902	0.001149	0.001149

[884 rows x 8 columns]

If, at this stage, you suspect that one or more outliers is unduly influencing your model fit, you can transform your results into robust OLS with a method call:

```
In [80]: model_1.get_robustcov_results().summary()
```

```
Out[80]: <class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:        inside_barriers   R-squared:                       0.215
Model:                            OLS   Adj. R-squared:                  0.214
Method:                 Least Squares   F-statistic:                     101.0
Date:                Tue, 01 Mar 2016   Prob (F-statistic):           1.40e-22
Time:                        16:39:18   Log-Likelihood:                -807.74
No. Observations:                 884   AIC:                             1619.
Df Residuals:                     882   BIC:                             1629.
Df Model:                           1
Covariance Type:                  HC1
===================================================================================
                      coef    std err          t      P>|t|      [95.0% Conf. Int.]
-----------------------------------------------------------------------------------
Intercept           0.7529      0.036     21.041      0.000         0.683     0.823
outside_barriers    0.2464      0.025     10.052      0.000         0.198     0.294
==============================================================================
Omnibus:                      389.637   Durbin-Watson:                   1.865
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1871.839
Skew:                           2.026   Prob(JB):                         0.00
Kurtosis:                       8.865   Cond. No.                         5.17
==============================================================================

Warnings:
[1] Standard Errors are heteroscedasticity robust (HC1)
"""
```

This isn't very different, so we're probably okay.

If you want to add more predictors to your model, you can do so inside the function string:

```
In [81]: smf.ols("inside_barriers ~ outside_barriers + gender", data=data).fit().summary()
```

```
Out[81]: <class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:        inside_barriers   R-squared:                       0.235
Model:                            OLS   Adj. R-squared:                  0.233
Method:                 Least Squares   F-statistic:                     86.65
Date:                Tue, 01 Mar 2016   Prob (F-statistic):           7.02e-49
Time:                        16:39:18   Log-Likelihood:                -748.48
No. Observations:                 848   AIC:                             1505.
Df Residuals:                     844   BIC:                             1524.
Df Model:                           3
Covariance Type:            nonrobust
==========================================================================================
                                                    coef    std err          t      P>|t|      [9
------------------------------------------------------------------------------------------
```

```
Intercept                                           0.6550      0.045     14.680      0.000
gender[T.Genderqueer/Gender non-conforming]        -0.9480      0.588     -1.611      0.108
gender[T.Male/Man]                                  0.1808      0.043      4.209      0.000
outside_barriers                                    0.2586      0.016     16.106      0.000
==============================================================================
Omnibus:                      358.447   Durbin-Watson:                   1.947
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1650.675
Skew:                           1.940   Prob(JB):                         0.00
Kurtosis:                       8.626   Cond. No.                         76.2
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
"""
```

Note that our categorical/factor variable has been automatically one-hot encoded as treatment conditions. There's not way to change this within `statsmodels`, but you can specify your contrasts indirectly using a library called (`Patsy`)[http://statsmodels.sourceforge.net/stable/contrasts.html].

To add interactions to your model, you can use `:`, or `*` [for full factorial]

In [82]:
```python
smf.ols("inside_barriers ~ outside_barriers * gender", data=data).fit().summary()
```

Out[82]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:        inside_barriers   R-squared:                       0.267
Model:                            OLS   Adj. R-squared:                  0.264
Method:                 Least Squares   F-statistic:                     76.92
Date:                Tue, 01 Mar 2016   Prob (F-statistic):           1.26e-55
Time:                        16:39:18   Log-Likelihood:                -730.40
No. Observations:                 848   AIC:                             1471.
Df Residuals:                     843   BIC:                             1495.
Df Model:                           4
Covariance Type:            nonrobust
==================================================================================================
                                                                      coef    std err          t
--------------------------------------------------------------------------------------------------
Intercept                                                           0.7909      0.049     16.101
gender[T.Genderqueer/Gender non-conforming]                        -0.0303      0.022     -1.364
gender[T.Male/Man]                                                 -0.2132      0.077     -2.753
outside_barriers                                                    0.1993      0.019     10.755
outside_barriers:gender[T.Genderqueer/Gender non-conforming]       -0.1514      0.111     -1.364
outside_barriers:gender[T.Male/Man]                                 0.2124      0.035      6.061
==============================================================================
Omnibus:                      351.976   Durbin-Watson:                   1.962
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             1805.642
Skew:                           1.851   Prob(JB):                         0.00
Kurtosis:                       9.115   Cond. No.                     3.96e+16
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 3.92e-30. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
"""
```

3 Practice

In the time remaining, pull up a dataset that you have, and that you'd like to work with in Python. The instructors will be around to help you apply what you've learned today to problems in your data that you are dealing with.

If you don't have data of your own, you should practice with the test data we've given you here. For example, you could try to figure out:

1. Is King Arthur happier than Sir Robin, based on his speech?
2. Which character in Monty Python has the biggest vocabulary?
3. Do different departments have the same gender ratios?
4. What variable in this dataset is the best predictor for how useful people find our workshops to be?