Pandas: Python Programming for Spreadsheets

Pamela Wu

Sept. 17th 2015

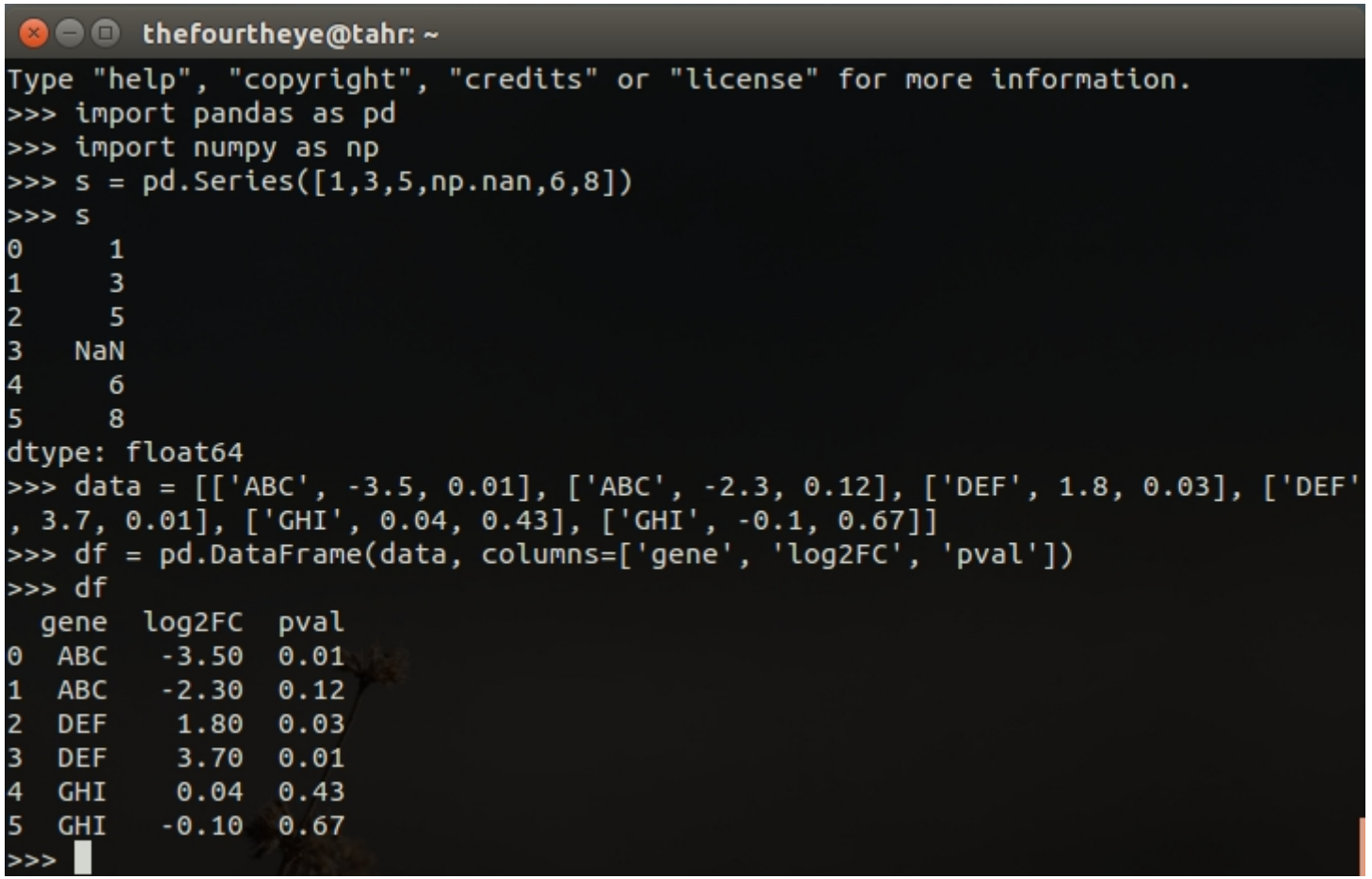
Getting Started

- Excel is easy to use, but scientists need more powerful tools
- GraphPad can only handle so many rows before crashing
- Today we'll learn how to
  - Quickly get stats on all of your samples
  - Merge data from multiple rows (i.e. transcripts to a gene)
  - Filter data by various criteria
  - Merge data from multiple sheets (i.e. UCSC annotation to HUGO)
- All of this comes from a module called “pandas”, which is included in Anaconda, so it should already be installed on your machine
- `import pandas as pd`
- `import numpy as np`

Pandas Objects

- Like lists, dictionaries, etc., Pandas has two objects:
  - **Series:** like a column in a spreadsheet
  - **DataFrame:** like a spreadsheet – a dictionary of Series objects
- Let's make some columns and spreadsheets
- `s = pd.Series([1,3,5,np.nan,6,8]) # What is np.nan?`
- `data = [['ABC', -3.5, 0.01], ['ABC', -2.3, 0.12], ['DEF', 1.8, 0.03], ['DEF', 3.7, 0.01], ['GHI', 0.04, 0.43], ['GHI', -0.1, 0.67]]`
- `df = pd.DataFrame(data, columns=['gene', 'log2FC', 'pval'])`
- Now type `s` and `df` into your terminal and see what it outputs

Pandas Objects



Viewing Data

- Try the following:
- `df.head()`
- `df.tail()`
- `df.tail(2)`
- `df['log2FC']`
- `df.columns`
- `df.index`
- `df.values`
- You should see, in order: the first 5 lines, the last 5 lines, the last 2 lines, only the column 'log2FC', the columns, the indices, and the data

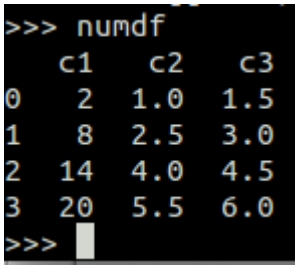
- Unlike other Python data objects, if you print a Pandas object to the terminal, it won't flood your screen because it was designed to be readable
- What you'll find in the following sections is that Pandas objects have a logic that is quite different from regular Python
- For example, operations happen on entire columns and rows
- The new Pandas rules exist to make your life easier, but it means you have to hold two sets of rules in your head

Basic Operations

- We'll go back to our `df` in a moment, but first, create this spreadsheet:
- `nums = [[1, 2], [4, 5], [7, 8], [10, 11]]`
- `numdf = pd.DataFrame(nums, columns=['c1', 'c2'])`
- Add a column:
- `numdf['c3'] = [3, 6, 9, 12]`
- Multiply all elements of a column (give just the name of the column):
- `numdf['c1'] = numdf['c1']*2`
- Divide all elements of multiple columns (give the DF a list of columns):
- `numdf[['c2', 'c3']] = numdf[['c2', 'c3']]/2`

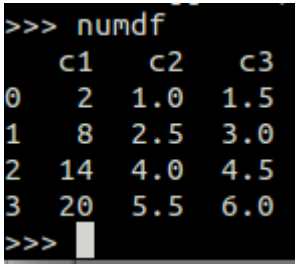
Basic Metrics

- Your DataFrame should look like this:
- Now try the following:
- `numdf.describe()`
- `save_stats = numdf.describe()`
- What if you want to calculate those yourself?
- `numdf.max(axis=0) # across all rows: the default`
- `numdf.max(axis=1) # across all columns`
- Now try the above for `numdf.min()`, `numdf.mean()`, `numdf.std()`, `numdf.median()`, and `numdf.sum()`
- Use what we learned to normalize all columns:
- `normdf = (numdf - numdf.mean())/numdf.std()`



Indexing and Iterating

- Remember indexing? How does it work with DFs?
- `numdf.ix[1, 'c2']`
- `numdf.ix[1, ['c1', 'c2']]`
- `numdf.ix[1]`
- `numdf.ix['c2'] # error`
- Exercise: get me 14 from `numdf`
- Exercise: get me the column `c2` for real. Hint: on another slide
- `numdf.ix[1, 'c2'] = 5.0`
- `numdf['c2'][1] = 5.0 # How are they different?`



Filtering Data

- Let's go back to our original DF, `df`
- We only want to see the p-values that passed
- `df['pval'] < 0.05 # this is a boolean Series`
- `df[df['pval'] < 0.05] # this is called boolean indexing`
- `df['gene'].isin(['ABC', 'GHI'])`
- `df[df['gene'].isin(['ABC', 'GHI'])]`
- `df[(df['pval'] < 0.05) & (df['gene'].isin(['ABC', 'GHI']))]`
- How do you save these to variables?
- Boolean indexing can also do assignments
- `df.ix[df['pval'] > 0.05, 'log2FC'] = np.nan`

Concat and Merge

- If two DFs share the same columns, they can be concatenated:
- pd.concat([numdf, numdf])
- More interestingly, two DFs can be joined by column values:
- `annodf = pd.DataFrame([['DEF', 'Leppard'], ['GHI', 'Ghost Hunters International'], ['ABC', 'Always Be Closing']], columns=['acronym', 'association'])`
- `resdf = df.merge(annodf, left_on='gene', right_on='acronym')`
- Write `resdf` in your console. You've just annotated your dataset!

Sort and Groupby

- Let's sort by p-value:
- `df = df.sort(columns='pval') # apparently deprecated, but the documentation on sort_values isn't up yet`
- Exercise: sort it by gene name
- In our data example, we have multiple rows with the same name. How do we easily combine their values?
- `smdf = df.groupby('gene').mean()`

Input and Output

- How do you get data into and out of Pandas as spreadsheets?
- Pandas can now work with XLS or XLSX files (they didn't use to)
  - A tab looks like this: `'\t'`, but on your file it looks like a big space
  - *Can also be comma-delimited, but bioinformatics people always like to use tabs because there are sometimes commas in our data*
  - **Check which delimiter your file is using before import!**
- Import to Pandas:
- `df = pd.read_csv('data.csv', sep='\t', header=0) # or header=None if there is no header`
- For Excel files, it's the same thing but with `read_excel`
- Export to text file:
- `df.to_csv('data.csv', sep='\t', header=True, index=False) # the values of header and index depend on if you want to print the column and/or row names`

Assignment Time!

- You should have two data files, data.csv and anno.csv
- I want you to:
  - Import data.csv and anno.csv
  - Annotate the data with gene names and loci
  - Assign all log fold changes with unpassed q-values to NaN
  - Obtain the mean fold change for each unique gene
  - Make one DataFrame with positive log fold change and q-value < 0.05, sort it by gene name, and save it to file
- Each step should only be one or two lines, and the trick is looking up and executing the correct commands