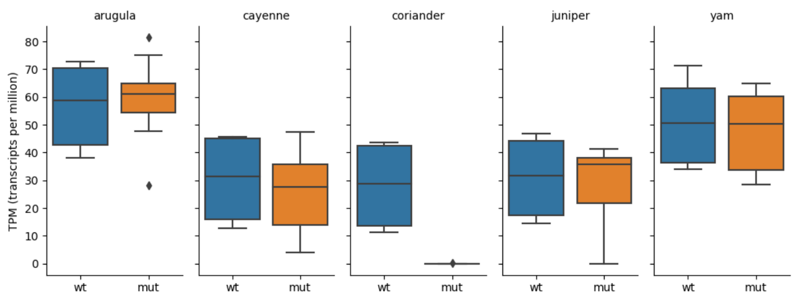
pset03: the adventure of the missing phenotype

The sand mouse gene *Coriander* is supposed to encode an important transcription factor. However, in a new paper from Lestrade *et al.* in the *Sand Mouse Journal*, an RNA-seq analysis of wild-type versus *Coriander* mutant sand mice shows that loss of *Coriander* function results in no significant effects on the mean expression of any sand mouse gene transcript (other than *Coriander* itself, of course).

For example, Figure 2 from Lestrade's paper shows mean expression levels and standard deviations for four sand mouse genes previously believed to be direct targets of the *Coriander* transcription factor (*Arugula*, *Cayenne*, *Juniper*, and *Yam*), showing no expression differences. Lestrade also showed that the *Coriander* mutant is indeed a knockout, null for RNA expression. The figure is based on RNA-seq measurements on ten biological replicates each (wow!), in wild type versus mutant sand mice.



You decide to dig a little deeper into Lestrade's data. Fortunately, Lestrade's lab, though somewhat bumbling, does seem to practice reproducible computational science, because they've made their data available in their

Supplementary Data Table 1: w03-data.tbl.

Download the data file, and have a look. It's a column-justified, whitespace-delimited file. Looks like the table starts with some comment lines and commented column headers, followed by one line of RNA-seq expression levels (in units of TPM, transcripts per million) per gene, with 21 fields per line: the gene name, 10 replicates for wild type, and 10 replicates for mutant.

1. downsample the data set by reservoir sampling

Write a script that takes a random sample of 10 data (non-comment) lines, using the reservoir sampling algorithm.

Obviously Pandas can take a random sample easily, with `pd.sample()`. You can use Pandas to read the table if you want, but implement your own reservoir sampling algorithm; don't use `pd.sample()`.)

2. look at outliers; validate the formatting; and clean the data

Write a script that checks over the whole file and:

- Find and print the data line that contains the maximum expression level.
- Find and print the data line that contains the minimum expression level.
- Find lines that don't have the right format. ("Right" is deliberately left vague. Validation is a creative art!)

Write a script that removes the data lines that have the problems you just detected.

(hint: there are two problems in the data file, affecting 35 data lines. Your cleaning should leave you with 19,996 data lines for that many sand mouse genes, not counting comment/header lines.)

3. tidy the data

Write a script that converts the data table to a new file in tidy data format.

Both Pandas and Seaborn (which we're about to use next) are happiest with tidy data.

4. visualize the data and explain your conclusions

Write a script to read your tidy data file and visualize the distribution of the raw data points using *beeswarm plots* (also known as *swarm plots*) -- that is, 20 points per gene, 10 biological replicates for wild type versus mutant -- for any list of chosen sand mouse genes, and use it to display a plot of 10 randomly sampled genes. The Seaborn package has a good swarm plot function... which is why we're using Seaborn.

Also have a look at Seaborn's *catplot* function too (i.e. a *categorical plot*, which they used to call a *factor plot*), which gives you the ability to plot lots of genes in a "facet grid" in the same figure.

Instead of a swarm plot, another way to do it (which I might actually prefer, now that I've been playing with it for a while) is a Seaborn *strip plot* with `jitter=True`.

(Helpful side note: we're only introducing you a wee bit to data plotting this week. Learning Pandas in some depth is the bigger goal. If you aren't familiar with Matplotlib or Seaborn for plotting, that's fine -- don't start reading their manuals or tutorials yet. Instead, skip to the "massive hint" below. All you're really going to have to do here is take a working Seaborn script and figure out how to get it to plot what you want.)

What's going on in the *Coriander* mutant?

turning in your work

Submit your Jupyter notebook page (a `.ipynb` file) to the course Canvas page under the Assignments tab. Please name your file `<LastName><FirstName>_<psetnumber>.ipynb`; for example, mine would be `EddySean_03.ipynb`.

Remember that we grade your psets as if they're lab reports. The quality of your text explanations of what you're doing (and why) are just as important as getting your code to work. We want to see you discuss both your analysis code, and your biological interpretations.

---

hints

- Lestrade *et al.* say they validated the data file, to make sure it contains no bad data, and they've even provided the script they used. This is another example (like last week) showing how to open a file, read it one line at a time, and split each line into whitespace-delimited fields that you can look at one at a time. (When you look at their script, you may also develop some worries about their data validation strategy.) This is a bare-bones starting point. Data science modules like Pandas give you more powerful tools for data file input and parsing.

- There's two reasonable ways to make the 'tidy' data set, as we discussed in lecture, with different advantages and disadvantages. One form ('wide'), with each individual gene as a variable, is only mostly-tidy, but makes more sense for many kinds of analysis; another form ('long') with `gene_name` as a variable is fully tidy in Wickham's sense, and is easier to manipulate and plot in a Seaborn factorplot, at least for this exercise. Hint: check out the Pandas `melt` method, which allows you to convert any number of columns (measured variables - like, oh, say, 20,000 columns labeled as gene names for example) into a single variable in one column ("gene", for example).

- Need an even more massive hint on using Seaborn to produce swarm plots and jittered strip plots? First, try your friend Google; you should be able to find examples of people using (struggling to use) Seaborn to make swarm plots or related plots such as boxplots. But if you get stuck, or if it's your first time plotting data with Python... Lestrade *et al.* have also provided the script they used to produce their boxplots, and a tidy data file that it will work on. This is a massive hint because it only takes a small tweak to make this produce your swarm plot -- so use it only if you're really stuck and frustrated.

- The script with the massive plotting hint also shows an example of using the Pandas `melt` method to convert from "wide" to "long" tidy format for the data.