Lab: gene expression

Assignment 4

Due 17 February

Compare gene expression before and after year-long exercise program

- 20 people with high insulin and glucose levels
- RNA-seq to measure gene expression
- One year of exercise program
- Same 20 people now have normal insulin and glucose levels
- RNA-seq again to measure gene expression

Assumptions of gene expression

- Assume that gene expression levels correspond to function product levels
- Assume that a normal cell has a standard expression profile/signature
- Assume that changes in expression profile indicate that something important is happening

Data set contains raw RNA-seq counts for each person Before and After

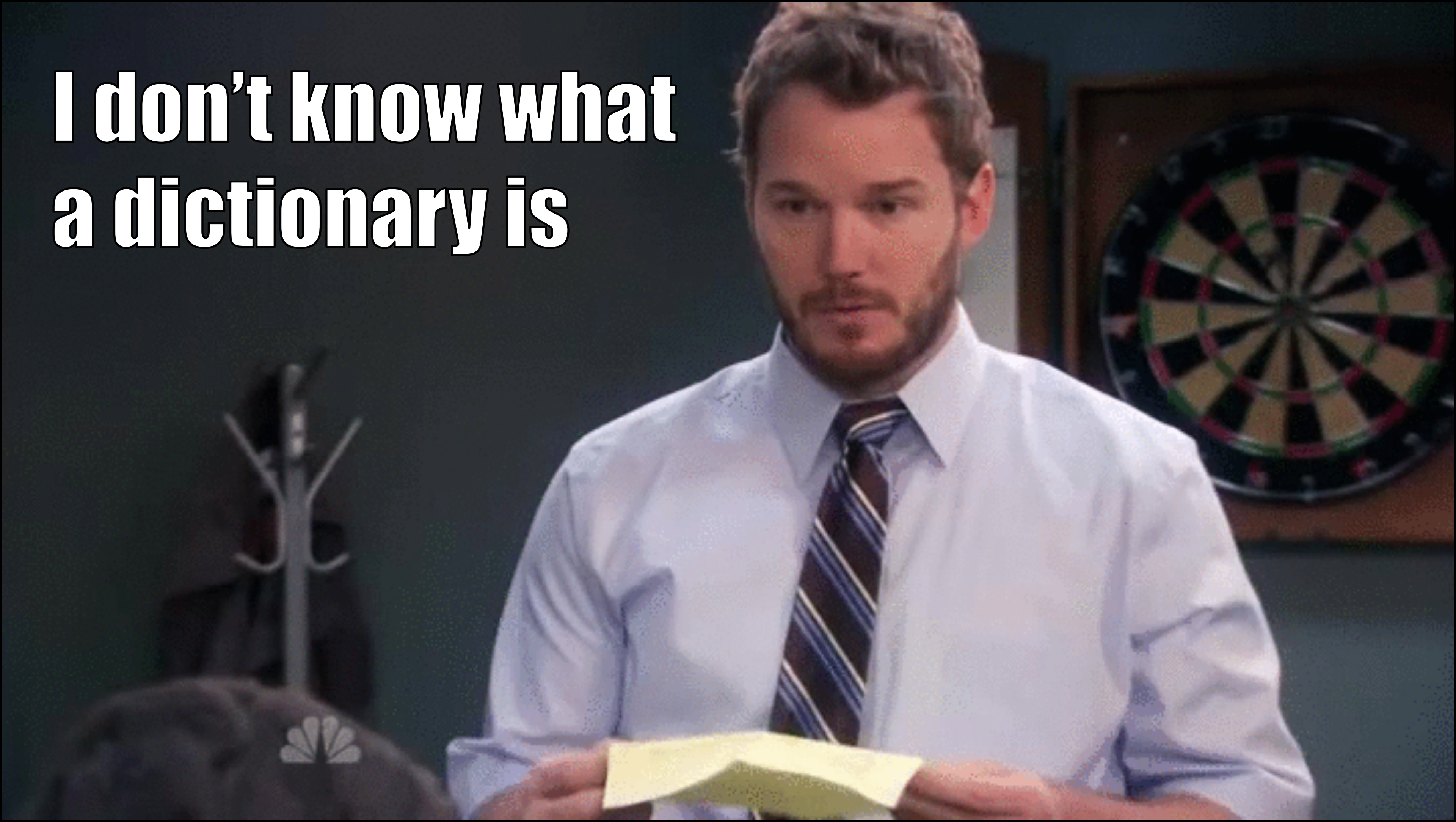
Gene	**Before1**	**Before2**	**…**	**Before20**	**After1**	**After2**	**…**	**After20**
A1BG	7	3	…	7	13	4	…	13
A1BG-AS1	10	4	…	13	8	5	…	5
…	…	…	…	…	…	…	…	…
ZZZ3	742	554	…	645	593	405	…	632

- 27012 genes at the start
- Filter out genes with little to no information about gene expression
- Normalize data before comparing before and after groups

Where to start your analysis

- Your labmate did a similar project but only has an incomplete version of the script they used
- You need to fill in the blanks and add a new function to complete your assignment

- Guess what! You're going to get a lot of practice working with dictionaries.


I don’t know what a dictionary is

Two approaches to filtering data

1. Remove genes from an existing dictionary if they do not pass a filter
   - Python gets upset if you remove something from a dictionary while iterating through it
   - Make a list of keys you want to remove and go back later to remove them
   - Remove a key from a dictionary: `gene_list.pop(“A1BG”,None)`
2. Copy genes (and counts) from an existing dictionary to another dictionary if they DO pass a filter
   - This approach is simpler when iterating through a dictionary testing if genes pass filter or not
   - Could be inefficient with larger data sets, etc.

Upper quartile normalization

Gene	**Before1**	**Before2**	**…**	**Before20**	**After1**	**After2**	**…**	**After20**
A1BG	7	3	…	7	13	4	…	13
A1BG-AS1	10	4	…	13	8	5	…	5
…	…	…	…	…	…	…	…	…
ZZZ3	742	554	…	645	593	405	…	632
75th pctile	D1	D2	…	D20	D21	D22	…	D40

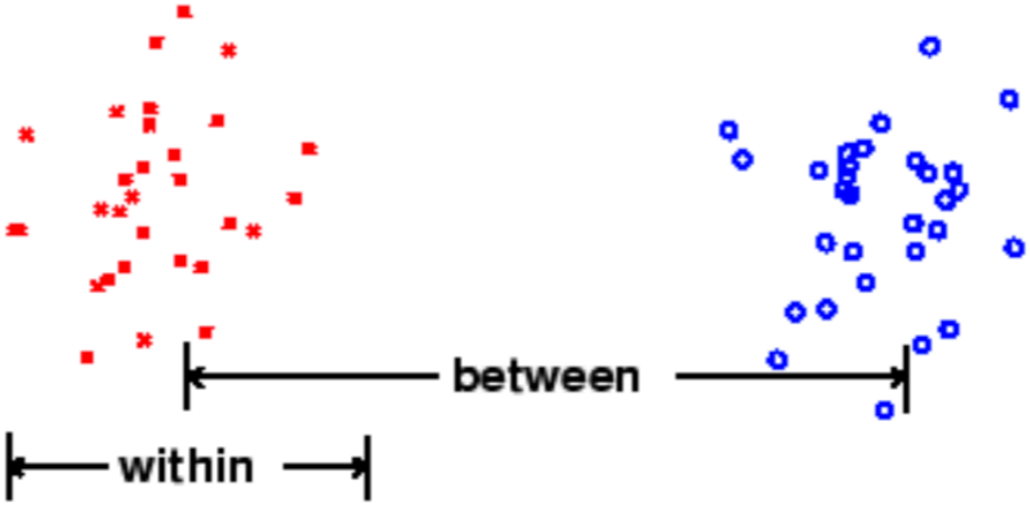
- D1 = 75th percentile of [7, 10, …, 742]
- D = [D1, D2, ..., D40]
- Multiply each column *i* by mean(D)/D(*i*)

Fisher's Linear Discriminant

- Measure how separated two groups are
- Two groups (before and after exercise) may differ in the expression level of some genes
- Use numpy module to calculate mean and standard deviation

$$FLD(j) = \frac{(m_1 - m_2)^2}{(s_1)^2 + (s_2)^2}$$

Where:
m_1 = the mean value of group 1
m_2 = the mean value of group 2
s_1 = the standard deviation of group 1
s_2 = the standard deviation of group 2



Making bar charts

```
# At top of script
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# When you want to plot something
fig = plt.figure() #start a new figure
plt.bar(left=[x-axis values], height=[y-axis values], align='center')
plt.xlabel('Your x-axis label')
plt.ylabel('Your y-axis label')
plt.suptitle('Title of chart')
fig.savefig('your_file_name.png')
```

Look here to add error bars, change color of error bars, and change tick mark labels:
http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.bar
http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.xticks

Dictionaries in python

- Dictionaries store data as key and value pairs
- Then, you can "look up" the value of a key
- Example:
  - Key = "A1BG"
  - Value = [7, 3, …, 13]
- Initialize a dictionary with curly brackets
  - `gene_dict = {}`
- Add a new key and value to a dictionary
  - `gene_dict["A1BG"] = [7, 3, …, 13]`
- Change the value associated with a key
  - `gene_dict["A1BG"].append(2)`
  - Works the same as whatever data type the value has
- Look up the value of a key
  - `gene_dict["A1BG"]`

For loop through dictionaries

- Dictionaries are iterable
- You can iterate through them with a for loop
- The general format could look like this:

```
for k,v in gene_dict.items():
    #k is key (type = string)
    #v is value (type = list)
    #k and v are variables
    print(k,“:”,v[0:2])
```

- Return the keys in alphabetical order:
  - `sorted(gene_dict.keys())`
- Return the keys, but sorted highest to lowest by value
  - `sorted(gene_dict, key=gene_dict.get, reverse=True)`