NIAPythonDay4

March 28, 2019

1 Day 1: IDE, basic data types, operators

2 Day 2: Slicing, NumPy array

3 Day 3: Pandas DataFrame

4 Day 4: Case Study: Microarray Data analysis and Visualization

- file: ExampleMicroarrayData.xls
- Data: brain region microarray data (measures gene expression levels in a tissue)
- Experimental variables = tissue type
  - hippocampus & cerebral cortex
- Columns: 4 repeats for each tissue type
- Rows: 1 row per read; 1 or more rows per gene. Redundant reads need to be merged.

4.1 Analysis Method

1. Import file
2. For a given gene with redundant reads, take maximum value (collapse multiple rows into one row)
3. Log transform
4. Z-score
5. For each comparison, take difference of means of repeats and p-value
6. Create ribbon plot for genes with fold change of 1 and p-value < 0.05

4.2 Import libraries

```
In [1]: import pandas as pd
        import numpy as np
```

4.3 Optional: set Pandas display precision

Use TAB key to show you what options are available

```
In [ ]: pd.options.display
```

```
In [2]: pd.options.display.precision = 3
```

4.4 Read in Excel File

```
In [3]: microa = pd.read_excel( 'ExampleMicroarrayData.xls')
```

4.5 See what we got

```
In [4]: microa.head()
```

```
Out[4]:    ArrayID   Symbol      BR1_1      BR1_2      BR1_3      BR1_4      BR2_1  \
        0        1      NA1  24540.450  43972.090  32894.380  39717.850  37762.740
        1        2      NA2      3.384      3.420      3.184      3.504      2.218
        2        3      NA3      3.402      3.452      3.215      3.549      2.244
        3        4  Tbc1d19    497.424    704.163    598.064    770.349    318.999
        4        5     Cfc1      3.436     14.535      3.271      3.640      2.296

              BR2_2      BR2_3      BR2_4
        0  58349.680  20307.620  25211.600
        1      2.300      2.404      2.307
        2      2.327      2.434      2.334
        3    315.689    329.867    342.941
        4      2.374     10.220      2.384
```

```
In [5]: microa.shape
```

```
Out[5]: (59734, 10)
```

```
In [6]: microa.columns
```

```
Out[6]: Index(['ArrayID', 'Symbol', 'BR1_1', 'BR1_2', 'BR1_3', 'BR1_4', 'BR2_1',
               'BR2_2', 'BR2_3', 'BR2_4'],
              dtype='object')
```

4.6 Review: Get basic statistics across all columns using .describe()

```
In [7]: microa.describe()
```

```
Out[7]:           ArrayID       BR1_1       BR1_2       BR1_3       BR1_4       BR2_1  \
        count   59734.000   59734.000   59734.000   59734.000   59734.000   59734.000
        mean    31524.803    3826.969    4417.788    4440.600    5140.971    2811.512
        std     18161.786   12949.801   14653.350   14482.841   16651.936   11257.903
        min         1.000       3.126       2.979       2.860       3.060       2.048
        25%     15839.250      18.760      22.889      21.042      24.194      13.127
        50%     31507.500     292.898     361.491     363.980     418.769     205.459
        75%     47259.750    2143.263    2576.601    2603.771    3029.693    1387.381
        max     62976.000  309771.300  351175.700  340936.200  377660.600  548163.100

                    BR2_2       BR2_3       BR2_4
        count    59734.000   59734.000   59734.000
        mean      3070.629    3017.429    3422.667
```

```
std     11271.983   11149.905   12554.989
min         2.024       2.049       2.084
25%        13.279      10.264      12.211
50%       226.891     187.594     224.595
75%      1625.061    1499.185    1726.906
max    327943.800  381527.600  343815.500
```

4.7 Review: subselect rows by boolean criterion using brackets []

```
In [8]: microa[ microa.Symbol == 'Mfsd9' ]
```

```
Out[8]:        ArrayID Symbol     BR1_1     BR1_2     BR1_3     BR1_4    BR2_1  \
3692      3920  Mfsd9   968.993  1010.708  1007.123  1130.436  434.947
14504    15379  Mfsd9   949.827  1056.157   979.577  1163.150  482.391
15412    16338  Mfsd9    23.193    18.875    22.034    24.326    8.435
19317    20435  Mfsd9   891.462  1036.658   991.143  1173.474  452.980
21174    22376  Mfsd9   920.478   998.698  1052.486  1231.313  434.002
26077    27531  Mfsd9    18.640    24.579    12.145    28.158    9.539
32783    34582  Mfsd9   854.077  1062.900  1008.647  1180.013  435.248
33005    34823  Mfsd9   949.943   985.132  1079.542  1201.548  463.207
34495    36380  Mfsd9   856.771  1024.499   914.007  1123.651  473.591
48207    50847  Mfsd9   955.400  1029.965  1039.952  1146.253  471.125
57835    60983  Mfsd9   915.912  1021.728   947.302  1168.269  421.222
58099    61263  Mfsd9  1051.273  1004.055   996.786  1133.530  448.804

          BR2_2    BR2_3    BR2_4
3692    589.708  600.502  621.537
14504   543.322  601.314  674.765
15412     9.239    7.162    9.467
19317   555.692  622.570  606.833
21174   547.448  637.982  625.079
26077    12.697    8.741   11.155
32783   577.290  663.561  615.494
33005   530.134  596.087  659.742
34495   585.492  618.376  644.952
48207   596.579  627.870  621.421
57835   589.248  632.728  660.304
58099   589.341  651.024  605.704
```

4.8 Review: subselect one or more columns using brackets []

```
In [ ]: microa[ 'Symbol' ]
```

```
In [ ]: microa[ ['ArrayID', 'Symbol' ] ]
```

4.9 For subselecting rows and columns at the same time, use .loc[] (or.iloc[])

```
In [24]: microa.loc[ microa.Symbol == 'Mfsd9',
                     ['BR1_1', 'BR2_1' ] ]
```

```
Out[24]:          BR1_1    BR2_1
         3692    968.993  434.947
         14504   949.827  482.391
         15412    23.193    8.435
         19317   891.462  452.980
         21174   920.478  434.002
         26077    18.640    9.539
         32783   854.077  435.248
         33005   949.943  463.207
         34495   856.771  473.591
         48207   955.400  471.125
         57835   915.912  421.222
         58099  1051.273  448.804
```

4.10 Note: Difference between .loc[] and .iloc[]

- Use .loc[] to slice by row/column *NAMES* or *BOOLEANS*
- Use .iloc[] to slice by row/column *INDICES* (Like we used with NumPy)

```
In [ ]: microa.loc[ :10, -4: ]
```

```
In [26]: microa.iloc[ :10, -4: ]
```

```
Out[26]:        BR2_1      BR2_2      BR2_3      BR2_4
         0  37762.740  58349.680  20307.620  25211.600
         1      2.218      2.300      2.404      2.307
         2      2.244      2.327      2.434      2.334
         3    318.999    315.689    329.867    342.941
         4      2.296      2.374     10.220      2.384
         5     60.898     79.251     88.181     83.621
         6    337.046    246.692    264.554    373.699
         7  14415.300  15068.380  15317.930  18569.540
         8  21151.840  23888.320  25168.000  28457.830
         9    266.326    225.108    254.067    244.691
```

4.11 How many unique gene symbols do we have?

```
In [27]: len( microa.Symbol )
```

```
Out[27]: 59734
```

```
In [28]: microa.Symbol.unique()
```

```
Out[28]: array(['NA1', 'NA2', 'NA3', ..., 'NA11372', 'NA11373', 'NA11374'],
               dtype=object)
```

```
In [29]: len(microa.Symbol.unique())
```

```
Out[29]: 29784
```

```
In [ ]: microa.Symbol.value_counts()

In [31]: microa.Symbol.value_counts().value_counts()

Out[31]: 1     14855
         2     11361
         4      1162
         10      948
         3       932
         12      211
         11      130
         5       117
         6        28
         13       16
         20        8
         7         7
         14        3
         9         3
         8         2
         16        1
         Name: Symbol, dtype: int64
```

4.12 Group rows by gene symbol using .groupby()

- Pass the function the name of the column containing the values you want to group by

```
In [32]: grouped = microa.groupby('Symbol')

In [33]: type( grouped)

Out[33]: pandas.core.groupby.groupby.DataFrameGroupBy

In [ ]: grouped.mean()
```

4.12.1 Get a certain group using .get_group()

```
In [39]: grouped.get_group( 'Mfsd9' )

Out[39]:        ArrayID     BR1_1     BR1_2     BR1_3     BR1_4    BR2_1    BR2_2  \
         3692      3920   968.993  1010.708  1007.123  1130.436  434.947  589.708
         14504    15379   949.827  1056.157   979.577  1163.150  482.391  543.322
         15412    16338    23.193    18.875    22.034    24.326    8.435    9.239
         19317    20435   891.462  1036.658   991.143  1173.474  452.980  555.692
         21174    22376   920.478   998.698  1052.486  1231.313  434.002  547.448
         26077    27531    18.640    24.579    12.145    28.158    9.539   12.697
         32783    34582   854.077  1062.900  1008.647  1180.013  435.248  577.290
         33005    34823   949.943   985.132  1079.542  1201.548  463.207  530.134
         34495    36380   856.771  1024.499   914.007  1123.651  473.591  585.492
         48207    50847   955.400  1029.965  1039.952  1146.253  471.125  596.579
```

```
57835    60983   915.912  1021.728   947.302  1168.269  421.222  589.248
58099    61263  1051.273  1004.055   996.786  1133.530  448.804  589.341

         BR2_3    BR2_4
3692   600.502  621.537
14504  601.314  674.765
15412    7.162    9.467
19317  622.570  606.833
21174  637.982  625.079
26077    8.741   11.155
32783  663.561  615.494
33005  596.087  659.742
34495  618.376  644.952
48207  627.870  621.421
57835  632.728  660.304
58099  651.024  605.704
```

```
In [40]: max_expression = microa.groupby('Symbol').max()

In [41]: max_expression.shape

Out[41]: (29784, 9)

In [44]: max_expression.head()

Out[44]:         ArrayID     BR1_1     BR1_2     BR1_3     BR1_4    BR2_1    BR2_2  \
         Symbol
         A1bg      47917     6.551     9.730     6.253    13.327    3.550    2.836
         A1cf      50995     3.718     3.788     3.795     5.505    2.599    2.552
         A1i3      59848     3.740     7.137     3.697     4.011    2.605    2.541
         A26c2     52963     3.738     3.796     3.797    11.602    5.992    3.765
         A2m       36219  6142.893  5840.628  6192.190  6855.576  533.450  408.658

                   BR2_3    BR2_4
         Symbol
         A1bg      4.228    4.405
         A1cf      2.595    2.649
         A1i3      2.601    2.648
         A26c2     2.597    8.467
         A2m     355.236  371.571

In [43]: max_expression.iloc[ 4, 1]

Out[43]: 6142.893

In [ ]: max_expression.loc[ 'A2m', 'BR1_1']
```

4.13 Combine reads

Take maximum expression level for a given gene.

```
In [45]: max_expression = grouped.max()

In [46]: max_expression.shape

Out[46]: (29784, 9)
```

5 Drop the Array ID Column

```
In [47]: max_expression.head()
```

Out[47]:

Symbol	ArrayID	BR1_1	BR1_2	BR1_3	BR1_4	BR2_1	BR2_2	BR2_3	BR2_4
A1bg	47917	6.551	9.730	6.253	13.327	3.550	2.836	4.228	4.405
A1cf	50995	3.718	3.788	3.795	5.505	2.599	2.552	2.595	2.649
A1i3	59848	3.740	7.137	3.697	4.011	2.605	2.541	2.601	2.648
A26c2	52963	3.738	3.796	3.797	11.602	5.992	3.765	2.597	8.467
A2m	36219	6142.893	5840.628	6192.190	6855.576	533.450	408.658	355.236	371.571

```
In [ ]: max_expression.drop?

In [48]: max_expression.drop( columns='ArrayID', inplace=True )

In [49]: max_expression.head()
```

Out[49]:

Symbol	BR1_1	BR1_2	BR1_3	BR1_4	BR2_1	BR2_2	BR2_3	BR2_4
A1bg	6.551	9.730	6.253	13.327	3.550	2.836	4.228	4.405
A1cf	3.718	3.788	3.795	5.505	2.599	2.552	2.595	2.649
A1i3	3.740	7.137	3.697	4.011	2.605	2.541	2.601	2.648
A26c2	3.738	3.796	3.797	11.602	5.992	3.765	2.597	8.467
A2m	6142.893	5840.628	6192.190	6855.576	533.450	408.658	355.236	371.571

5.1 .apply() a log transformation function over all expression values

```
In [ ]: np.log2?
```

```
In [51]: np.log2( [1, 10, 100, 1000] )
```

```
Out[51]: array([0.        , 3.32192809, 6.64385619, 9.96578428])
```

```
In [52]: max_expression.shape
```

```
Out[52]: (29784, 8)
```

```
In [53]: log_trans = max_expression.apply( np.log2 )
```

```
In [ ]: log_trans
```

5.2 .apply() a Z-score transformation function over all genes

```
In [55]: from scipy.stats import zscore
```

```
In [ ]: zscore?
```

```
In [57]: zscore( [ 1,2,3,4,5,6,7,8,9,10] )
```

```
Out[57]: array([-1.5666989 , -1.21854359, -0.87038828, -0.52223297, -0.17407766,
                0.17407766,  0.52223297,  0.87038828,  1.21854359,  1.5666989 ])
```

```
In [58]: z_trans = log_trans.apply(zscore)
```

```
In [60]: log_trans.head()
```

```
Out[60]:          BR1_1   BR1_2   BR1_3   BR1_4  BR2_1  BR2_2  BR2_3  BR2_4
         Symbol
         A1bg     2.712   3.283   2.644   3.736  1.828  1.504  2.080  2.139
         A1cf     1.895   1.921   1.924   2.461  1.378  1.352  1.376  1.405
         A1i3     1.903   2.835   1.886   2.004  1.381  1.345  1.379  1.405
         A26c2    1.902   1.924   1.925   3.536  2.583  1.913  1.377  3.082
         A2m     12.585  12.512  12.596  12.743  9.059  8.675  8.473  8.537
```

```
In [ ]: log_trans.apply?
```

```
In [61]: log_trans.shape
```

```
Out[61]: (29784, 8)
```

```
In [62]: z_trans.shape
```

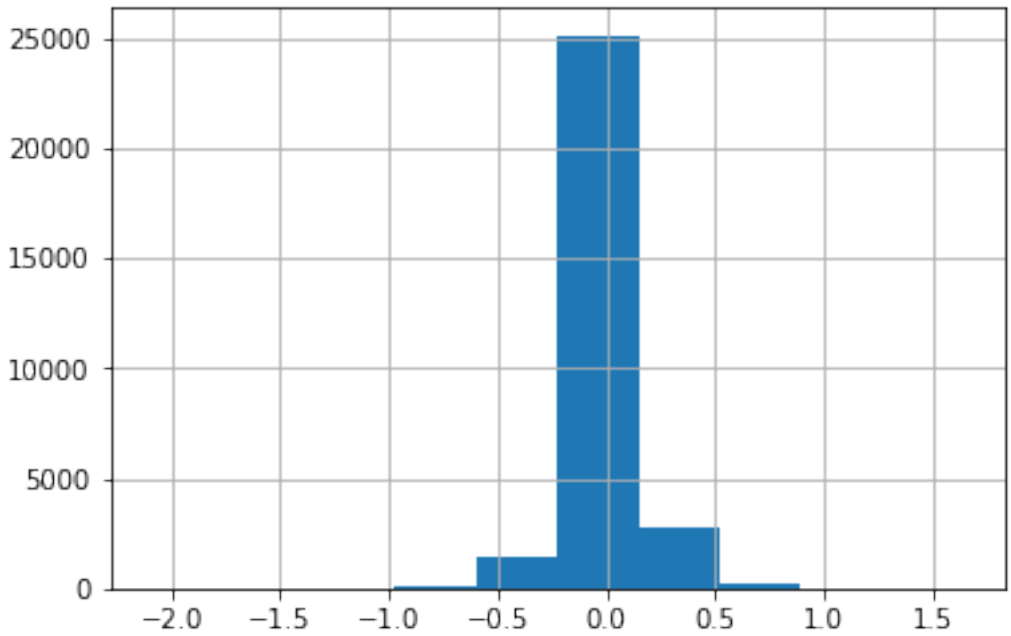
```
Out[62]: (29784, 8)
```

```
In [63]: z_trans.head()
```

```
Out[63]:          BR1_1  BR1_2  BR1_3  BR1_4  BR2_1  BR2_2  BR2_3  BR2_4
         Symbol
         A1bg   -1.104 -1.023 -1.147 -0.921 -1.198 -1.271 -1.085 -1.105
         A1cf   -1.301 -1.350 -1.317 -1.219 -1.307 -1.307 -1.250 -1.276
         A1i3   -1.299 -1.131 -1.326 -1.326 -1.306 -1.308 -1.250 -1.276
         A26c2  -1.299 -1.349 -1.317 -0.968 -1.017 -1.174 -1.250 -0.886
         A2m     1.273  1.188  1.203  1.185  0.541  0.422  0.413  0.383
```

5.3 Compare: Make a histogram of fold change

```
In [64]: br1 = z_trans[ ['BR1_1','BR1_2', 'BR1_3', 'BR1_4' ] ]

In [65]: br1.shape

Out[65]: (29784, 4)

In [66]: br2 = z_trans[ ['BR2_1','BR2_2', 'BR2_3', 'BR2_4' ] ]

In [67]: br2.shape

Out[67]: (29784, 4)

In [68]: br1_mean = br1.mean(axis=1)
         br2_mean = br2.mean(axis=1)

In [69]: br2_mean.shape

Out[69]: (29784,)

In [70]: diff = br2_mean - br1_mean

In [71]: diff.shape

Out[71]: (29784,)

In [72]: # tell Jupyter Notebook to show the figure after it made it
         %matplotlib inline

In [73]: diff.hist()

Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x112b8fac8>
```



5.4 Subselect genes based on having fold change > 1

```
In [ ]: diff[ diff.abs() > 1 ]
```

5.5 Get gene expression difference and p-value for desired comparisons

Use Wilcoxon rank-sum statistic for two samples to test if the reads for a given gene across comparison groups come from different distributions.

```
In [75]: from scipy.stats import ranksums

In [76]: ranksums( [1,2,3,4], [5,6,7,8])

Out[76]: RanksumsResult(statistic=-2.3094010767585034, pvalue=0.020921335337794014)

In [77]: ranksums( [1,2,3,4], [50,60,70,80])

Out[77]: RanksumsResult(statistic=-2.3094010767585034, pvalue=0.020921335337794014)

In [78]: ranksums( [1,2,3,5], [4,6,7,8])

Out[78]: RanksumsResult(statistic=-2.0207259421636903, pvalue=0.043308142810791955)
```

5.5.1 Go row-by-row doing significance test

```
In [79]: import time

In [80]: # create an empty list onto which we can append p-values
         pval_list = []

In [81]: t1 = time.time()

         # iterate over every gene
         for gene in br1.index:

             # subselect the expression values for the corresponding gene
             vals1 = br1.loc[ gene ]
             vals2 = br2.loc[ gene ]

             # Do the statistical test for this gene
             statistic, pvalue = ranksums( vals1, vals2 )

             # save the p-value
             pval_list.append( pvalue )

         t2 = time.time()
         print( "This operation took", t2-t1, "seconds.")

This operation took 13.183716058731079 seconds.
```

```
In [82]: len(pval_list)

Out[82]: 29784

In [83]: # One-liner!
         t1 = time.time()
         pval_list = [ ranksums(a,b)[1] for a, b in zip( br1.as_matrix(), br2.as_matrix() ) ]
         t2 = time.time()
         print( "This operation took", t2-t1, "seconds.")

/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:3: FutureWarning: Method .as_matri
  This is separate from the ipykernel package so we can avoid doing imports until


This operation took 4.83489203453064 seconds.
```

5.6 Combine fold change and p-value into one DataFrame

- Problem 1: the diff object is just one column (a Pandas "Series" object), not a full-fledged DataFrame
- Problem 2: the p-values are stored in a simple Python list
- Solution: use the to_frame() function to turn the Series into a DataFrame, then add the p-values as a new column to that DataFrame.

```
In [84]: type( diff )

Out[84]: pandas.core.series.Series

In [85]: diff.head()

Out[85]: Symbol
         A1bg    -0.116
         A1cf     0.012
         A1i3    -0.015
         A26c2    0.151
         A2m     -0.772
         dtype: float64

In [86]: combined = diff.to_frame()

In [87]: combined.head()

Out[87]:              0
         Symbol
         A1bg    -0.116
         A1cf     0.012
         A1i3    -0.015
         A26c2    0.151
         A2m     -0.772
```

```
In [88]: combined = diff.to_frame( name='fold')

In [89]: combined.head()

Out[89]:          fold
         Symbol
         A1bg   -0.116
         A1cf    0.012
         A1i3   -0.015
         A26c2   0.151
         A2m    -0.772

In [90]: combined[ 'pvals' ] = pval_list

In [91]: combined.head()

Out[91]:          fold  pvals
         Symbol
         A1bg   -0.116  0.149
         A1cf    0.012  0.564
         A1i3   -0.015  0.564
         A26c2   0.151  0.149
         A2m    -0.772  0.021

In [92]: combined.shape

Out[92]: (29784, 2)
```

5.7 Subselect genes with fold change > 1

```
In [93]: combined[ combined.fold.abs() > 1 ]

Out[93]:             fold  pvals
         Symbol
         Adam33     1.231  0.021
         C1ql2     -1.092  0.021
         Cd3e      -1.243  0.021
         Chst9     -1.406  0.021
         Cox6a2     1.153  0.021
         Csap1     -1.359  0.021
         Cxcr1     -1.511  0.021
         Cyp11b1    1.640  0.021
         Dpp4       1.057  0.021
         Fer1l4     1.212  0.021
         Fibcd1    -1.098  0.021
         Frem3     -1.127  0.021
         Gpat2      1.229  0.021
         Htr5b     -1.612  0.021
         Klk8      -1.738  0.021
```

```
LOC688459 -1.045  0.021
Lhx9      -1.354  0.021
Lrrc10b   -1.367  0.021
Meox1     -1.040  0.021
Meox2     -1.028  0.021
NA1086     1.151  0.021
NA1528     1.069  0.083
NA3871     1.197  0.021
NA3959     1.206  0.021
NA4283     1.426  0.021
NA5561    -1.124  0.021
Ndst4     -1.346  0.021
Nhlh1     -1.090  0.021
Ntf3      -1.082  0.021
Ntrk1     -1.386  0.021
Nts       -2.095  0.021
Ocm2       1.288  0.021
Olr315     1.008  0.083
Olr59      1.267  0.021
Qrfpr      1.016  0.021
Slc9a4    -1.078  0.021
Sprr1a     1.146  0.021
Upb1      -1.026  0.021
```

5.8 Subselect genes with fold change > 1 AND p-value < 0.05

```
In [94]: combined[ (combined.fold.abs() > 1) & (combined.pvals < 0.05) ]
```

```
Out[94]:             fold  pvals
         Symbol
         Adam33     1.231  0.021
         C1ql2     -1.092  0.021
         Cd3e      -1.243  0.021
         Chst9     -1.406  0.021
         Cox6a2     1.153  0.021
         Csap1     -1.359  0.021
         Cxcr1     -1.511  0.021
         Cyp11b1    1.640  0.021
         Dpp4       1.057  0.021
         Fer1l4     1.212  0.021
         Fibcd1    -1.098  0.021
         Frem3     -1.127  0.021
         Gpat2      1.229  0.021
         Htr5b     -1.612  0.021
         Klk8      -1.738  0.021
         LOC688459 -1.045  0.021
         Lhx9      -1.354  0.021
         Lrrc10b   -1.367  0.021
```

```
Meox1     -1.040  0.021
Meox2     -1.028  0.021
NA1086     1.151  0.021
NA3871     1.197  0.021
NA3959     1.206  0.021
NA4283     1.426  0.021
NA5561    -1.124  0.021
Ndst4     -1.346  0.021
Nhlh1     -1.090  0.021
Ntf3      -1.082  0.021
Ntrk1     -1.386  0.021
Nts       -2.095  0.021
Ocm2       1.288  0.021
Olr59      1.267  0.021
Qrfpr      1.016  0.021
Slc9a4    -1.078  0.021
Sprr1a     1.146  0.021
Upb1      -1.026  0.021
```

```
In [95]: plot_these = combined[ (combined.fold.abs() > 1) & (combined.pvals < 0.05) ]
```

```
In [96]: len(plot_these)
```

```
Out[96]: 36
```

Turn the row labels into variables in their own right:

```
In [97]: plot_these['Gene'] = plot_these.index
```

```
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.
```

```
In [98]: plot_these.sort_values(by='fold', inplace=True)
```

```
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.
```

5.9 Generate RibbonPlot

5.9.1 Load some of Python's figure-making libraries:

```
In [99]: import seaborn as sns
         import matplotlib.pyplot as plt
```

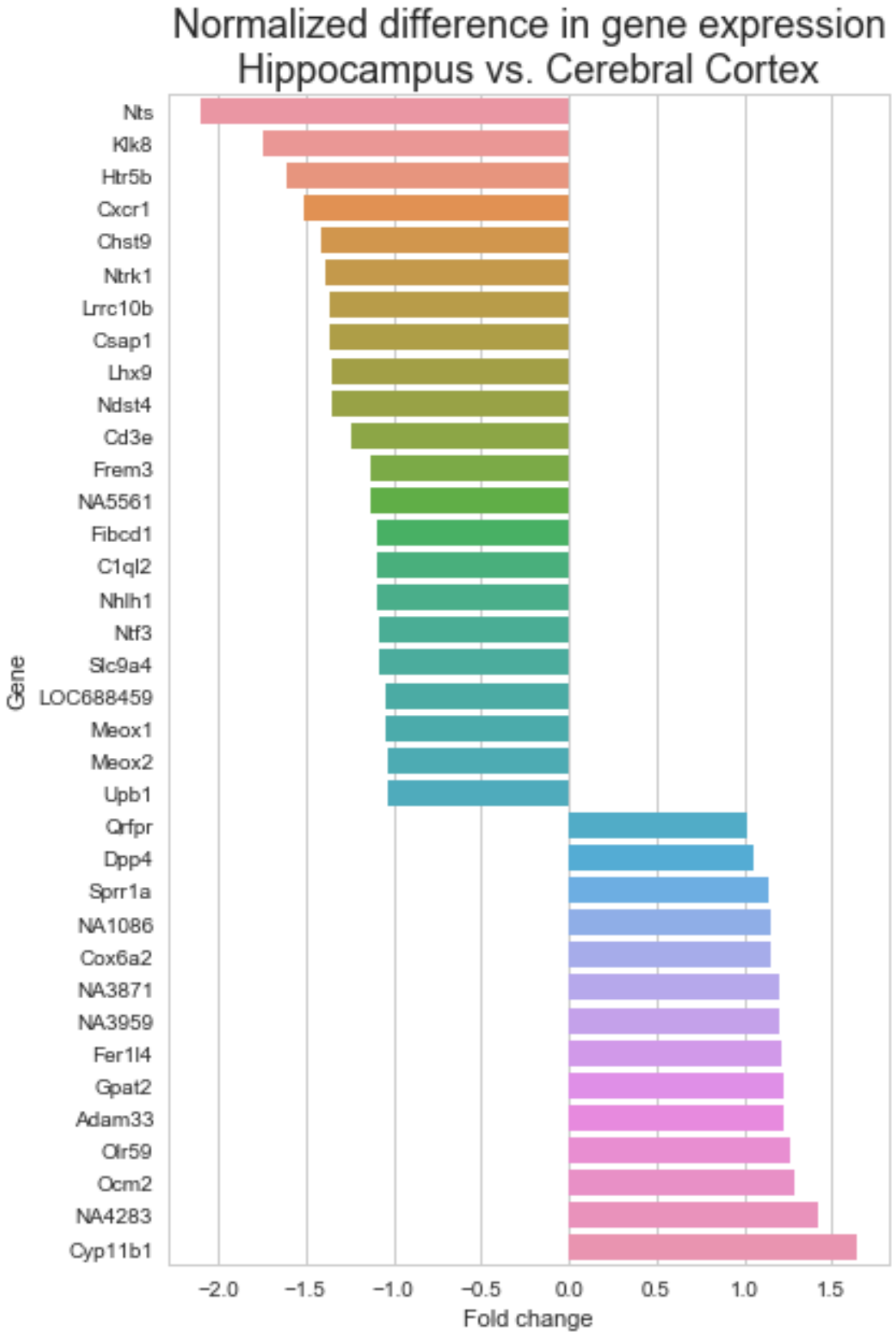
5.9.2 Set the style of the figure

When using the plotting package seaborn, there are five figure styles to choose from: 1. darkgrid 2. whitegrid 3. dark 4. white 5. ticks

```
In [100]: sns.set( style="whitegrid" )
```

5.9.3 Making the figure

```
In [101]: # I want this figure to be 6 inches wide and 10 inches tall
          fig_dimensions=(6, 10)

          # Create a blank figure to hang the data off of
          figure, axes = plt.subplots( figsize=fig_dimensions )

          # Plot the data onto the figure
          sns.barplot( data=plot_these, x="fold", y="Gene" )

          # Assign a title to the figure
          chart_title = """Normalized difference in gene expression
          Hippocampus vs. Cerebral Cortex"""
          axes.set_title( chart_title, size=18 )

          # Assign a label to the x-axis
          axes.set_xlabel( "Fold change" )

Out[101]: Text(0.5,0,'Fold change')
```


Normalized difference in gene expression
Hippocampus vs. Cerebral Cortex
Nts
Klk8
Htr5b
Cxcr1
Chst9
Ntrk1
Lrrc10b
Csap1
Lhx9
Ndst4
Cd3e
Frem3
NA5561
Fibcd1
C1ql2
Nhlh1
Ntf3
Slc9a4
LOC688459
Meox1
Meox2
Upb1
Qrfpr
Dpp4
Sprr1a
NA1086
Cox6a2
NA3871
NA3959
Fer1l4
Gpat2
Adam33
Olr59
Ocm2
NA4283
Cyp11b1
Gene
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Fold change

5.9.4 Save the figure as a PDF:

```
In [ ]: figure.savefig( "ribbonplot.pdf")
```