

Download Chapter pdf

Download Chapter notebook (ipynb)

Mandatory Lesson Feedback Survey

- What are different ways of importing data in NetworkX?
- What are the common file import error?
- How to troubleshoot the import errors?
- Reviewing data import
- Understanding troubleshooting of common errors in data import
- Applying network concepts to real-world datasets

Prerequisites

- Networks - Part 1
- Networks - Part 2

```
1 import networkx as nx
2
3 from pandas import read_csv
4
5 from numpy import mean, fill_diagonal
6 from numpy.random import randint
7
8 from matplotlib.pyplot import subplots, show
```

Importing data into NetworkX

In the previous two lessons we have looked at the basics of NetworkX, created some networks, analysed various properties of networks, and practised these techniques with the pre-prepared *C. elegans* dataset. If you want to use NetworkX with data of your own, you'll need to import your data in a compatible way. Specifically, you need to be able to import data from different original data formats.

CSV Format

'csv' indicates a very common file format, comma separated values. We used it in the previous lesson to import the simplified *C. elegans* neuronal network.

The network data 'celegans131matrix_50.csv' is simply a large adjacency matrix of 0s and 1s, with no further information. The 'celegans131labels_50.csv' file contains the list of neuron names, which we

used to label the nodes. We'll have a look in more detail at how this import works. We use Pandas functions as a simple way to import a CSV file. (There are of course other ways.)

Here is the import of the data for the first 50 neurons.

```
1 neurons=read_csv('data/celegans131matrix_50.csv', header=None, dtype =
  "int")
2
3 neurons=neurons.to_numpy()
4
5 print(len(neurons))
```

```
1 50
```

We import the CSV network file as before. We specify with a keyword argument that there is no header, otherwise the first line in the file will be assumed to be a header and not parsed (handled) in the same way as the rest of the data. We also specify that the adjacency matrix is of data type `int`. (This does not necessarily have to be the case, see the Microbiome network below.) We then convert the `neurons` dataframe to a Numpy array. The `neuron_Names` can be imported in a similar way. They will need to be converted from a dataframe to a dictionary:

```
1 neuron_Names = read_csv('data/celegans131labels_50.csv', header=None)
2
3 neuronNames = neuron_Names.to_dict()
4
5 neuronLabels = neuronNames[0]
6
7 print(neuronLabels)
```

```
1 {0: 'ADFL', 1: 'ADFR', 2: 'ADLL', 3: 'ADLR', 4: 'AFDL', 5: 'AFDR', 6: '
  AIAL', 7: 'AIAR', 8: 'AIBR', 9: 'AINL', 10: 'AINR', 11: 'AIZL', 12:
  'AIZR', 13: 'ALA', 14: 'ASEL', 15: 'ASER', 16: 'ASGL', 17: 'ASGR',
  18: 'ASHL', 19: 'ASHR', 20: 'ASIL', 21: 'ASIR', 22: 'ASJL', 23: '
  ASJR', 24: 'ASKL', 25: 'ASKR', 26: 'AUAL', 27: 'AUAR', 28: 'AVAL',
  29: 'AVAR', 30: 'AVBL', 31: 'AVBR', 32: 'AVDL', 33: 'AVDR', 34: '
  AVEL', 35: 'AVER', 36: 'AVHL', 37: 'AVHR', 38: 'AVJL', 39: 'AVJR',
  40: 'AVL', 41: 'AWAL', 42: 'AWAR', 43: 'AWBL', 44: 'AWBR', 45: 'AWCL
  ', 46: 'AWCR', 47: 'BAGL', 48: 'BAGR', 49: 'CEPDL'}
```

Dictionaries associate keys with values. You may remember we previously created a dictionary to assign labels to nodes. Here we convert the `neuron_Names` dataframe into a dictionary the neuron (node) indices are the keys and the corresponding neuron (node) names are the values. As the function `to_dict` wraps the dictionary within a dictionary, we obtain the plain labels dictionary by referring to index '0'.

Now we can create a graph, specify a layout, and plot the network.

```
1 neuronGraph = nx.from_numpy_matrix(neurons)
2
3 neuronLayout = nx.random_layout(neuronGraph, seed=123)
4
5 nx.draw(neuronGraph, neuronLayout,
6         node_size=1000,
7         labels = neuronLabels)
8
9 show()
```

In this particular case we also have other metadata that can be used for visualisation and analysis. In our case, we have the file 'celegans131positions_50.csv' - which contains information on how the nodes relate to each other in 2-D space. We can include this information to replace the layout.

```
1 neuronPos = read_csv('data/celegans131positions_50.csv', header=None)
2
3 neuronPositions = neuronPos.values
4
5 nx.draw(neuronGraph, neuronPositions,
6         node_size=1000,
7         labels = neuronLabels)
8
9 show()
```

The two BAG nodes to the right of the display are the (right and left) sensory neurons used to monitor oxygen and carbon dioxide.

Do it Yourself

Find the nodes indices of the sensory neurons named 'BAGL' and 'BAGR'.

```
1 for ind, name in enumerate(neuronLabels.values()):
2
3     if 'BAG' in name:
4
5         print(ind, neuronLabels[ind])
```

```
1 47 BAGL
2 48 BAGR
```

Networks Repository: List of Edges

Network files in the Network Repository are sourced from publications and provided in the 'edges' format. These are plain text files. Let's have a look at the network of a mouse visual cortex. You can

find this at MOUSE-VISUAL-CORTEX-1. You can either download the zip file from the database or use the file provided for this Lesson.

Place the file 'bn-mouse_visual-cortex_1.edges' in your working directory. It can be helpful to first open it in any text-editor to see what the data look like. In this case, it is a list of two numbers per row, separated by a space. This is the list of (directed) edges. The first number indicating 'from', the second 'to'. Nodes are not given explicitly but will be inferred from the indices.

Being a list of edges, we can import it into NetworkX as an *edgelist*, specifying that the nodes are given as integers. The result is a standard graph object which we can plot using *draw*.

```
1 MouseCortex = nx.read_edgelist("data/bn-mouse_visual-cortex_1.edges",
2                               nodetype=int)
3 MouseCortexLayout = nx.random_layout(MouseCortex, seed=111)
4
5 nx.draw(MouseCortex, MouseCortexLayout,
6         node_size=1000,
7         node_color='r',
8         with_labels=True)
9
10 show()
```

To interpret the network, you can check it against the source, MOUSE-VISUAL-CORTEX-1.

Do it Yourself

Network Repository provides some summary data on all their networks. Obtain the (i) number of nodes, (ii) number of edges, and (iii) average clustering coefficient and compare it to the data in their network summary.

```
1 print('Number of nodes:', len(MouseCortex))
2 print('')
3 print('Number of edges:', len(MouseCortex.edges))
4 print('')
5 print('Average clustering coefficient:', mean(list(nx.clustering(
6     MouseCortex).values())))
```

```
1 Number of nodes: 29
2
3 Number of edges: 44
4
5 Average clustering coefficient: 0.04942528735632184
```

JSON Format

JSON (JavaScript Object Notation) is an open standard file format which is language independent and widely used. Here is an example code to import and display the network that contains co-occurrences of characters in Victor Hugo's novel 'Les Misérables'.

```
1 import json
2
3 with open('data/miserables.json', 'r') as myfile:
4     data=myfile.read()
5
6
7
8 miserables = json.loads(data)
9
10 miserablesNetwork = nx.json_graph.node_link_graph(miserables)
11
12 miserablesNetworkLayout = nx.circular_layout(miserablesNetwork)
13
14 fig, ax = subplots(figsize=(12,12))
15
16 nx.draw(miserablesNetwork, miserablesNetworkLayout,
17         node_size=3000,
18         with_labels=True)
19
20 fig.tight_layout()
21
22 show()
```

As you can see, the 'data' variable contains the character names as node names.

If you are curious, open the file with a text editor and check the entries.

E.g. our file starts like this:

```
1 {
2
3     "nodes": [
4
5         {"id": "Myriel", "group": 1},
```

As you can see, the style with curly brackets and colons immediately suggests the use of associative arrays (dictionary) once the data are within Python. The function `node_link_graph` provides an interface to handle the data as a NetworkX graph.

There are many other files formats that contain network data. For a list of NetworkX functions that allow import from and export to other file formats, see Reading and writing graphs.

NetworkX troubleshooting

There are many errors that may occur in the context of dealing with network data. We exemplify some common problems, to help find out how to diagnose and solve them.

FileNotFoundError

Here we try to import a CSV file which contains an adjacency matrix. We have checked that the file is present in the working directory.

```
1 neurons = read_csv('data/celegans131mtrix_50.csv', header=None, dtype="float64")
```

```
1 Error: FileNotFoundError: [Errno 2] No such file or directory: 'data/celegans131mtrix_50.csv'
```

Do it Yourself

What has gone wrong, and what is the solution?

You'll see this error if something has gone wrong in the way you've instructed Python to find your file. This may be that you've mis-typed the file name, or the file isn't in your working directory. Here, the file is present in the working directory, but there's a typo in the file name.

```
1 neurons = read_csv('data/celegans131matrix_50.csv', header=None, dtype='int')
```

KeyError

Here we generate a random adjacency matrix, specify how it should be plotted, and try to visualise it.

```
1 rm = randint(0, 2, size=(5, 5))
2
3 thisgraph = nx.from_numpy_matrix(rm)
4
5 thisgraphLayout = nx.spiral_layout(thisgraph)
6
7 thisgraphLabels = {
8     0: 'A',
9     1: 'B',
10    2: 'C',
11    3: 'D',
```

```
12     4: 'E',
13     5: 'F',
14 }
15
16 nx.draw(thisgraph, thisgraphLayout,
17         labels=thisgraphLabels)
18
19 show()
```

```
1 Error: KeyError: 5
```

Do it Yourself

Why does the call of `draw` throw an error, and what is the solution?

If you look at the error message, it first flags **5: 'F'**, and ends with an error about labels. We've included too many node labels: 6 labels for five nodes. This is easily corrected by taking one label out.

```
1  rm = randint(0, 2, size=(5, 5))
2
3  thisgraph = nx.from_numpy_matrix(rm)
4
5  thisgraphLayout = nx.spiral_layout(thisgraph)
6
7  thisgraphLabels = {
8      0: 'A',
9      1: 'B',
10     2: 'C',
11     3: 'D',
12     4: 'E',
13 }
14
15 nx.draw(thisgraph, thisgraphLayout,
16         node_size=1000,
17         node_color='tomato',
18         labels=thisgraphLabels)
19
20 show()
```

Graph is not bipartite

Here we generate a bipartite graph, and try to plot it.

```
1  thisBipartite = nx.Graph()
2
3  # Add nodes with the node attribute "bipartite"
```

```

4 thisBipartite.add_nodes_from(['a', 'b', 'c', 'd', 'e', 'f'], bipartite
    =0)
5 thisBipartite.add_nodes_from(['m', 'n', 'o', 'p', 'q', 'r'], bipartite
    =1)
6
7 # Add edges only between nodes of opposite node sets
8 thisBipartite.add_edges_from([('a', 'm'), ('a', 'n'),
9                               ('b', 'n'), ('b', 'o'),
10                              ('c', 'o'), ('d', 'm'),
11                              ('a', 'p'), ('a', 'd'),
12                              ('f', 'p'), ('d', 'r'),
13                              ('a', 'q'), ('e', 'n')])

```

```
1 nx.is_connected(thisBipartite)
```

```
1 True
```

```

1 basegroup = nx.bipartite.sets(thisBipartite)[0]
2
3 bipartiteLayout = nx.bipartite_layout(thisBipartite, basegroup)
4
5 nx.draw(thisBipartite, bipartiteLayout,
6         node_size=2000,
7         with_labels=True)
8 show()

```

```
1 Error: networkx.exception.NetworkXError: Graph is not bipartite.
```

```
1 Error: NameError: name 'basegroup' is not defined
```

```
1 Error: NameError: name 'bipartiteLayout' is not defined
```

Do it Yourself

Why did this happen, what check can you do to avoid the problem, and how do you debug it?

In this instance, the error message is very clear: you can't plot a graph with a bipartite layout if the graph isn't bipartite. This would have also been evident if I hadn't just checked that the graph was connected, but also that it was bipartite.

```
1 nx.bipartite.is_bipartite(thisBipartite)
```

```
1 False
```

There is one edge connecting two nodes within one group. If we either remove that edge, or change one of the nodes to the other group, it will be bipartite.


```

1 thisBipartite = nx.Graph()
2
3 # Add nodes with the node attribute "bipartite"
4 thisBipartite.add_nodes_from(['a', 'b', 'c', 'd', 'e', 'f'], bipartite
    =0)
5 thisBipartite.add_nodes_from(['m', 'n', 'o', 'p', 'q', 'r'], bipartite
    =1)
6
7 # Add edges only between nodes of opposite node sets
8 thisBipartite.add_edges_from([('a', 'm'), ('a', 'n'),
9                               ('b', 'n'), ('b', 'o'),
10                              ('c', 'o'), ('d', 'm'),
11                              ('a', 'p'), ('a', 'n'),
12                              ('f', 'p'), ('d', 'r'),
13                              ('a', 'q'), ('e', 'n')])

```

```

1 nx.bipartite.is_bipartite(thisBipartite)

```

```

1 True

```

```

1 basegroup = nx.bipartite.sets(thisBipartite)[0]
2
3 bipartiteLayout = nx.bipartite_layout(thisBipartite, basegroup)
4
5 fig = nx.draw(thisBipartite, bipartiteLayout,
6               node_size=2000,
7               node_color='tomato',
8               node_shape='h',
9               with_labels=True)
10
11 show()

```

Example: Full *C. elegans* Neural Network

In the final part, we look at two publicly available networks that are of biomedical interest.

In this first example we return to data about the neuronal connections in *C. elegans*. This time we are using the full 277 neuron dataset, which includes 131 frontal neurons. For the original version of the data, go to the (Dynamic Connectome) website and select the Resources tab.

The data in the 277×277 network matrix in the *celegans277.zip* file contain the connections between the 131 neurons in the frontal network, and 146 in the remainder of the network. If the connections *within* each of these groups are severed, and the only remaining edges are between the two groups, this takes the form of a *bipartite graph*.

We have made available this modified version of the 277 neuron matrix, with no edges within the 131

or the 146 neuron group, see file 'celegans277matrix_bipartite.csv'. Note that bipartite networks do not work in NetworkX if any of the nodes are not connected. Also available for download is a list of neuron labels and spatial positions, files 'celegans277labels.csv' and 'celegans277positions.csv'.

```
1 neurons = read_csv('data/celegans277matrix_bipartite.csv', header=None,
2                     dtype="int")
3 neuronNames = read_csv('data/celegans277labels.csv', header=None)
4
5 neurons= neurons.to_numpy()
6
7 neuronNames= neuronNames.to_dict()
8
9 myBipartiteLabels = neuronNames[0]
10
11 myBipartite = nx.from_numpy_matrix(neurons)
```

Here we import the bipartite module from NetworkX, and then import the bipartite 277 neuron adjacency matrix and list of neuron names, in the same method as previously. Next, we use the bipartite convention in NetworkX to designate which nodes are in the 0 group, and which are in the 1 group. There are several ways to make a graph bipartite, here this is done by listing the node numbers of the 146 neurons and adding the attribute '*bipartite*' = 0, and listing the node numbers of the 131 neurons and adding the attribute '*bipartite*' = 1.

```
1 myBipartite.add_nodes_from([
2     0, 1, 2, 3, 13, 14, 17, 21, 22, 23, 24, 25,
3     26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 59,
4     60, 61, 66, 67, 69, 78, 79, 84, 85, 86, 87, 88,
5     89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100,
6     101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112,
7     125, 126, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142,
8     143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154,
9     155, 156, 157, 158, 159, 160, 161, 162, 170, 171, 172, 173,
10    180, 195, 196, 203, 204, 205, 206, 207, 235, 236, 237, 238,
11    239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250,
12    251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262,
13    263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274,
14    275, 276], bipartite=0)
15
16 myBipartite.add_nodes_from([
17     4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 16,
18     18, 19, 20, 37, 38, 39, 40, 41, 42, 43, 44,
19     45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
20     56, 57, 58, 62, 63, 64, 65, 68, 70, 71, 72,
21     73, 74, 75, 76, 77, 80, 81, 82, 83, 113, 114,
22     115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 127,
23     128, 129, 130, 131, 132, 163, 164, 165, 166, 167, 168,
24     169, 174, 175, 176, 177, 178, 179, 181, 182, 183, 184,
25     185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 197,
```

```

26 198, 199, 200, 201, 202, 208, 209, 210, 211, 212, 213,
27 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224,
28 225, 226, 227, 228, 229, 230, 231, 232, 233, 234], bipartite=1)

```

We can confirm that the network is bipartite.

```

1 nx.bipartite.is_bipartite(myBipartite)

```

```

1 True

```

Next, we continue to use the bipartite convention introduced above to define the two bipartite groups, and set up a custom colouring list by group.

```

1 top = nx.bipartite.sets(myBipartite)[0]
2 pos = nx.bipartite_layout(myBipartite, top)
3
4 color_dictionary = {0:'gold',1:'lavender'}
5
6 color_list = [color_dictionary[i[1]] for i in myBipartite.nodes.data('
    bipartite')]
7
8 fig, ax = subplots(figsize=(10, 30))
9
10 nx.draw(myBipartite, pos,
11         node_size=1200,
12         labels=myBipartiteLabels,
13         node_color=color_list)
14 show()

```

The graph shows in a preliminary way that there are certain preferred connections between the two sets, in this case between the frontal group and the rest of the neurons.

Do it Yourself

Draw the bipartite neural network, coloured by group and positioned using the positional information from Dynamic Connectome in the file 'celegans277positions.csv'.

The 2D positions are contained in the file 'celegans277positions.csv'. After the import of the file, they can be extracted into a variable using `values`. The positions then need to replace the original layout values (called 'pos' above).

```

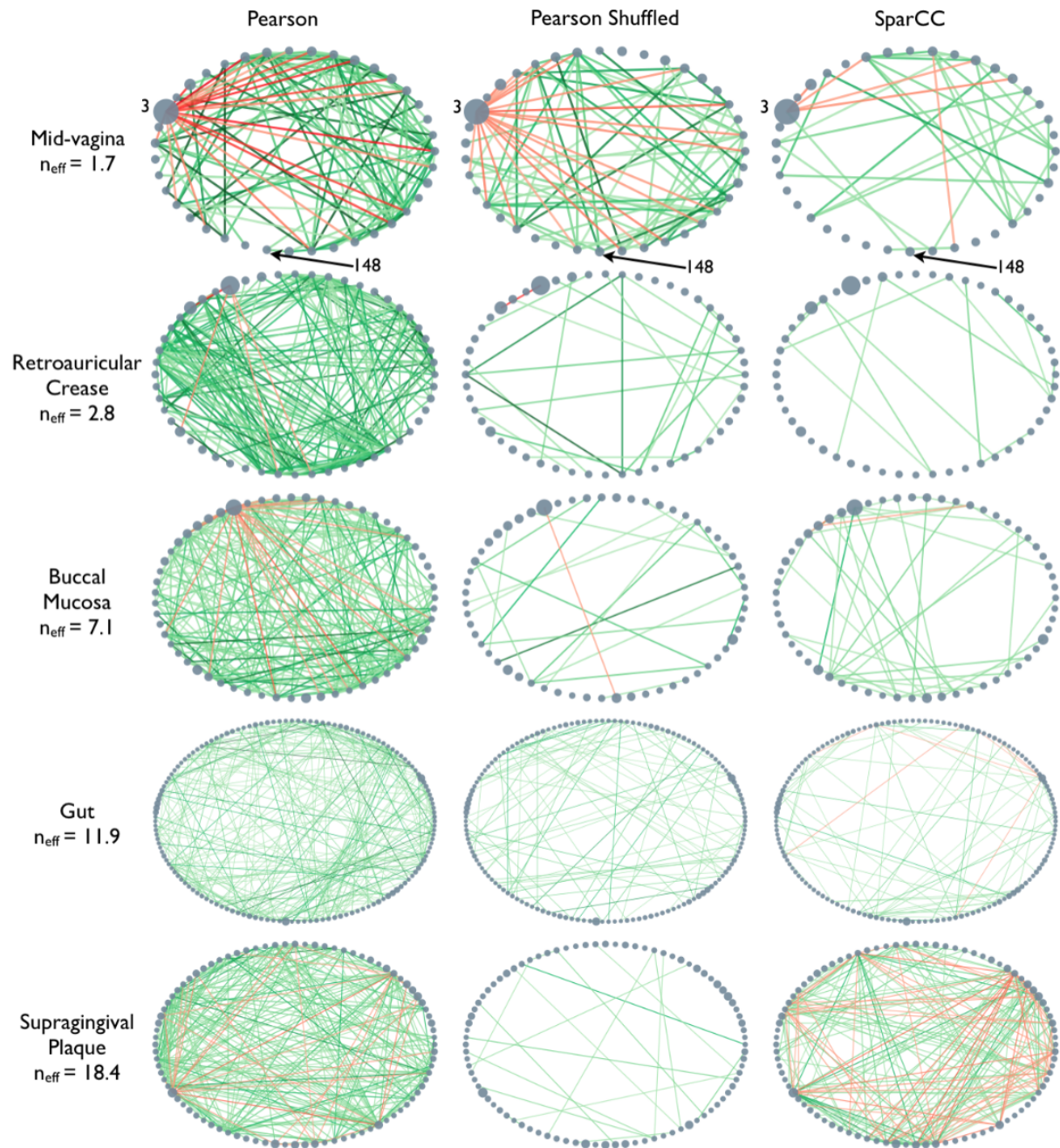
1 neuronPos = read_csv('data/celegans277positions.csv', header=None)
2
3 neuronPositions = neuronPos.values
4
5 fig, ax = subplots(figsize=(10, 20))
6

```

```
7 nx.draw(myBipartite, neuronPositions,  
8         node_size=3000,  
9         labels=myBipartiteLabels,  
10        node_color=color_list)  
11  
12 fig.tight_layout()  
13  
14 show()
```

Example: Microbiome network

Microbiome data can be collected from a wide variety of environments, and is generally considered to be the bacterial, fungal, and viral components of a given environment. Friedman and Alm (2012) used data from the Human Microbiome Project to calculate pairwise interaction correlations for each operational taxonomic unit (OTU). They then used Python and NetworkX to analyse and visualise several of these correlation datasets for each human body site. The challenge here is to extract network information about a graph from bivariate quantities like the correlation coefficient.



As the correlation data are open access, we can access them and derive the network.

Data are imported from text files. In this case, it is not a CSV file. Rather, the data are tab separated. We can adjust for this by specifying the keyword argument `delimiter` for tab. As the correlation coefficients are stored as decimal point numbers, we specify the keyword argument `dtype` as floating point 'float64'.

If you check the file, you will see that the first entry is '1.0'. This is because all self-correlations are

(trivially) equal to one. We replace those diagonal values with 0 to avoid (nonsensical) self-connections in the network.

```
1 gutbact      = read_csv('data/Stool_sparse_adj_matrix.txt',
2                          header=None, delimiter='\t', dtype="float64")
3
4 gutbactNames = read_csv('data/Stool_sparse_matrix_names.txt',
5                          header=None)
6
7 gutbact= gutbact.to_numpy()
8
9 fill_diagonal(gutbact, 0)
10
11 gutbactNames= gutbactNames.to_dict()
12
13 gutbactLabels = gutbactNames[0]
```

```
1 gutbactGraph = nx.from_numpy_matrix(gutbact)
2
3 gutbactLayout = nx.circular_layout(gutbactGraph)
4
5 fig, ax = subplots(figsize=(16,16))
6
7 nx.draw(gutbactGraph, gutbactLayout,
8         node_size=3000,
9         node_color='r',
10        labels=gutbactLabels)
11
12 show()
```

This gives us the correlation network from the stool samples, with nodes labelled by the OTU number. However, this doesn't look much like the data from the article, partly as there are so many edges. The reason is that any number in the network matrix that is not equal to zero is interpreted as an edge. Thus, essentially the graph is fully connected.

Functional Networks

In the paper, Friedman and Alm (2012), the edges are taken from the correlations coefficients of the correlation matrix. I.e. they define the correlation matrix as a network matrix. But the entries of the correlation matrix are not 0s and 1s as we have used so far. Instead they are real numbers between -1 and 1. What Friedman and Alm did to obtain meaningful edges was to pick a threshold correlation, e.g. $|cc| = 0.3$ and set any matrix entry with absolute value smaller than the threshold as 0. All other values are left as they are. Because anything non-zero will be interpreted as an edge, this means that they only plot edges where the correlation coefficient is greater than 0.3 or less than -0.3. Such networks are referred to as `_functional networks`.

To threshold the correlation matrix in Python, we select matrix values between -0.3 and 0.3, and replace them with 0. Then we convert this thresholded network matrix to a graph and plot it.

```
1 from numpy import where, logical_and
2
3 threshold = 0.3
4
5 gutbact_threshold = where(logical_and(gutbact>=-threshold , gutbact<=
    threshold), 0, gutbact)
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

This network only displays edges where strong correlations were found and this is how Friedman and Alm obtained their graphs. Note that the network structure now is a function of the chosen threshold. Smaller thresholds yield more edges, higher thresholds fewer edges. Choosing the right threshold may be difficult and needs further thought.

The creation of networks from functional data opens a wide field of research: *any* interrelation matrix obtained with whatever metric can be interpreted as a network matrix and converted into a graph. As an example, see Figure 1 in *The Brain as a Complex System: Using Network Science as a Tool for Understanding the Brain* which describes the procedure for brain imaging.

This concludes our set of Lessons on network handling in Python. For practice, try to find a database that touches on topics of your own interest. As an example, in the context of the human brain, there is a rich database at the Human Connectome Database. Each database will present its own challenges and obstacles. Nevertheless, it is worth to overcome those to be able to work with the data in Python. Eventually, all functions and dysfunctions observed in living systems are a consequence of interactions between components.

Keypoints

- `.csv` is a common file format when importing network data.
- `JSON` is another widely used standard file format which is language independent.
- `FileNotFoundError`, `KeyError` and `Graph is not bipartite` are the most common file import errors.