Build DAG Structure

July 13, 2021

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

import numpy as np
import pandas as pd
length = 1000
cols = ["Q", "X", "Y", "Z"]
mu = 0
sigma = 5


import pingouin
lst_dct = {col:[] for col in cols }
for i in range(length):
    lst_dct["Q"].append(50 + np.random.normal(mu, sigma))
    lst_dct["X"].append(5 * lst_dct["Q"][-1] + 10 + np.random.normal(mu, sigma␣
↪))
    lst_dct["Y"].append(lst_dct["Q"][-1] * -3 + 20 +  np.random.normal(mu,␣
↪sigma))
    lst_dct["Z"].append(5 * lst_dct["X"][-1] + 10 * lst_dct["Y"][-1] +  np.
↪random.normal(mu, 3 * sigma))

df = pd.DataFrame(lst_dct)
keys = ["X",
        "Y",
        "Z",
        "Q"]
dag_keys = keys
df
```

[1]:

	Q	X	Y	Z
0	52.457323	264.856404	-140.442207	-88.350736
1	55.187632	285.230763	-142.182892	10.818626
2	56.979403	297.170457	-149.832151	-59.109986
3	46.710283	239.312561	-127.561618	-72.351056
4	46.082279	241.174345	-121.159995	-4.737926

5	56.844132	289.864746	-148.758137	-29.840857
6	42.570988	228.502965	-112.082566	-9.250234
7	48.584345	246.648204	-118.891536	64.507319
8	49.071634	253.214566	-130.697490	-30.527712
9	45.754584	234.217265	-117.220257	0.102159
10	52.243100	268.527328	-132.298836	35.844453
11	46.688745	246.625787	-121.972997	15.593910
12	53.353651	273.902421	-135.720254	1.370654
13	54.540654	285.290418	-149.073016	-52.076949
14	50.948703	270.118997	-132.607422	49.175525
15	58.567293	299.091089	-155.177885	-45.588848
16	42.523717	221.000955	-114.798762	-68.229177
17	56.580527	287.768575	-151.256169	-80.483093
18	42.799918	225.697687	-108.320004	49.526891
19	45.071392	234.731725	-122.713110	-75.606396
20	48.984154	251.459296	-126.601926	-5.871169
21	53.333805	279.025306	-135.989947	43.698752
22	53.475110	276.223826	-140.270741	0.119387
23	39.914778	208.210433	-96.378864	56.709711
24	52.113612	269.495791	-139.435399	-38.003201
25	56.600207	289.387577	-139.526091	62.184516
26	45.841396	236.852090	-114.556367	40.981060
27	49.127282	258.069700	-128.771971	12.514957
28	43.405703	234.412522	-113.127494	48.194021
29	56.463465	300.876750	-148.977392	5.909910
..	…	…	…	…
970	40.959501	219.896385	-98.119874	140.221452
971	52.363482	268.293914	-139.858797	-42.521399
972	46.929016	250.230603	-118.932471	71.204668
973	44.712838	229.692369	-123.915555	-98.297387
974	54.937507	280.175298	-136.170448	39.729944
975	44.251748	228.736658	-102.338655	120.017808
976	53.485411	281.254092	-144.106622	-45.663720
977	46.799594	233.391561	-119.586966	-44.428083
978	50.167800	258.117773	-141.631598	-108.510803
979	46.805439	250.652399	-115.394886	82.467282
980	49.159098	247.665417	-132.094722	-89.887952
981	48.521564	261.736932	-130.795550	-35.055969
982	51.767786	261.037627	-134.113472	-45.789713
983	50.422784	265.265081	-115.844695	158.940732
984	49.052222	249.304513	-128.011815	-46.056844
985	45.394660	243.527921	-121.836632	-18.777373
986	54.042721	279.762656	-134.696460	73.342823
987	59.536165	304.414629	-163.810338	-127.328993
988	41.546015	226.441248	-106.091385	81.277787
989	44.081965	228.245579	-107.783295	67.746274
990	52.875136	281.023661	-134.699327	47.066682

```
991  54.733944  281.340658 -143.950623  -33.203364
992  47.055635  243.824300 -121.743851   -4.073991
993  42.499836  223.800593 -112.327021  -31.700728
994  53.450917  279.618887 -139.030525   11.653141
995  46.947367  255.982894 -124.125395   46.329974
996  53.587118  277.046160 -135.828283   54.491396
997  54.956602  284.959562 -147.914225  -53.486297
998  54.591523  288.134620 -144.527948   -7.834975
999  52.253658  273.646898 -136.490673  -13.327510

[1000 rows x 4 columns]
```

[2]:

```python
import pingouin

undirected_graph = {key:[] for key in df.keys()}
for x in undirected_graph:
    remaining_vars = [y for y in df.keys() if y != x]
    for y in remaining_vars:
        undirected_graph[x].append(y)

undirected_graph
```

[2]:

```
{'Q': ['X', 'Y', 'Z'],
 'X': ['Q', 'Y', 'Z'],
 'Y': ['Q', 'X', 'Z'],
 'Z': ['Q', 'X', 'Y']}
```

[3]:

```python
import copy
import pingouin
p_val = .001
def build_skeleton(df, undirected_graph):
    def check_remaining_controls(control_vars, undirected_graph, x, y,␣
 ↪controls_used) :
        c_used = copy.copy(controls_used)
        for c_var in control_vars:
            if y not in undirected_graph[x]:
                break
            c_used.append(c_var)
            test = df.partial_corr(x = x, y = y, covar=c_used,
                                   method = "pearson")
            if test["p-val"].values[0] > p_val:

                undirected_graph[x].remove(y)
                #breakout of the for
                break
            else:
                remaining_controls = copy.copy(control_vars)
```

```
                remaining_controls.remove(c_var)
                check_remaining_controls(remaining_controls, undirected_graph,␣
↪x, y, c_used)
    d_sep = {}
    for x in df.keys():
        ys = undirected_graph[x]
        for y in df.keys():
            d_sep[(x,y)] = []

            if x != y:
            # first check for correlation with no controls
                print(x, y)
                test = df.partial_corr(x = x, y = y, covar = None,method =␣
↪"pearson")
                if test["p-val"].values[0] > p_val:
                    undirected_graph[x].remove(y)
            # if correlated check for deseparation controlling for other␣
↪variables
                else:
                ############## make recursive function ############

                    control_vars = [z for z in df.keys() if z != y and z != x]
                    check_remaining_controls(control_vars, undirected_graph, x,␣
↪y, [])
    return undirected_graph

undirected_graph = build_skeleton(df, undirected_graph)
undirected_graph
```

```
Q X
Q Y
Q Z
X Q
X Y
X Z
Y Q
Y X
Y Z
Z Q
Z X
Z Y
```

[3]:
```
{'Q': ['X', 'Y'], 'X': ['Q', 'Z'], 'Y': ['Q', 'Z'], 'Z': ['X', 'Y']}
```

[4]:
```
def check_colliders(df, undirected_graph):
    for x in undirected_graph.keys():
        # use copy.copy() to make a copy of the list
```

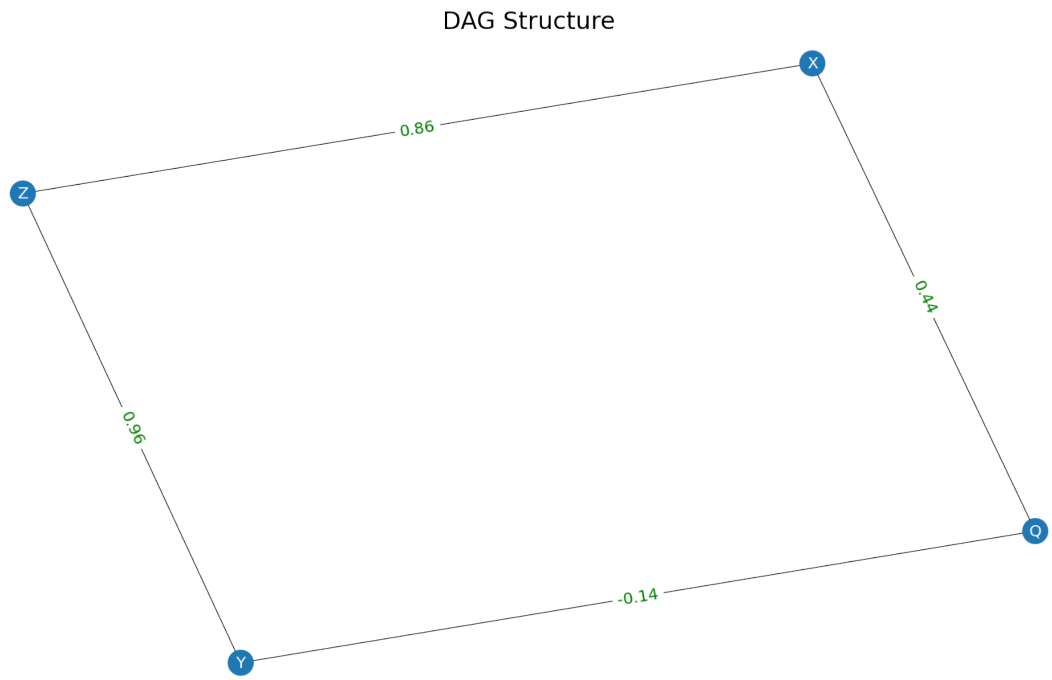
```
        # since we will likely remove links from the list
        ys = copy.copy(undirected_graph[x])
        for y1 in ys:
            remaining_ys = copy.copy(ys)
            remaining_ys.remove(y1)
            for y2 in remaining_ys:
                test = df.partial_corr(x = y1, y = y2, covar=[],
                                 method = "pearson")
                # mark previous significance as True or False for reference
                prev_sig = False if test["p-val"].values[0] > p_val else True

                test = df.partial_corr(x = y1, y = y2, covar=[x],
                                 method = "pearson")



check_colliders(df, undirected_graph)
```

[5]:

```
import matplotlib.pyplot as plt
import networkx as nx
def graph_DAG(undirected_graph, df, title = "DAG Structure"):

    # generate partial correlation matrix to draw values from
    # for graph edges
    pcorr_matrix = df.pcorr()
    graph = nx.Graph()
    edges = []
    edge_labels = {}
    for key in undirected_graph:
        for key2 in undirected_graph[key]:
            if (key2, key) not in edges:
                edge = (key.replace(" ","\n"), key2[0].replace(" ","\n"))
                edges.append(edge)
                # edge label is partial correlation between
                # key and key2
                edge_labels[edge] = str(round(pcorr_matrix.loc[key][key2],2))

    # edge format: ("i", "j") --> from node i to node j
    graph.add_edges_from(edges)
    color_map = ["C0" for g in graph]

    fig, ax = plt.subplots(figsize = (20,12))
    graph.nodes()
    plt.tight_layout()
    pos = nx.spring_layout(graph)#, k = 5/(len(sig_corr.keys())**.5))

    plt.title(title, fontsize = 30)
```

```
    nx.draw_networkx(graph, pos, node_color=color_map,
                     node_size = 1000,
                     with_labels=True,  arrows=False,
                     font_size = 20, alpha = 1,
                     font_color = "white",
                     ax = ax)
    nx.draw_networkx_edge_labels(graph,pos,
                                 edge_labels=edge_labels,
                                 font_color='green',
                                 font_size=20)
    plt.axis("off")
    plt.savefig("g1.png", format="PNG")
    # tell matplotlib you're done with the plot: https://stackoverflow.com/
↪questions/741877/how-do-i-tell-matplotlib-that-i-am-done-with-a-plot
    plt.show()
```

[6]:
```
graph_DAG(undirected_graph, df, title = "DAG Structure")
```



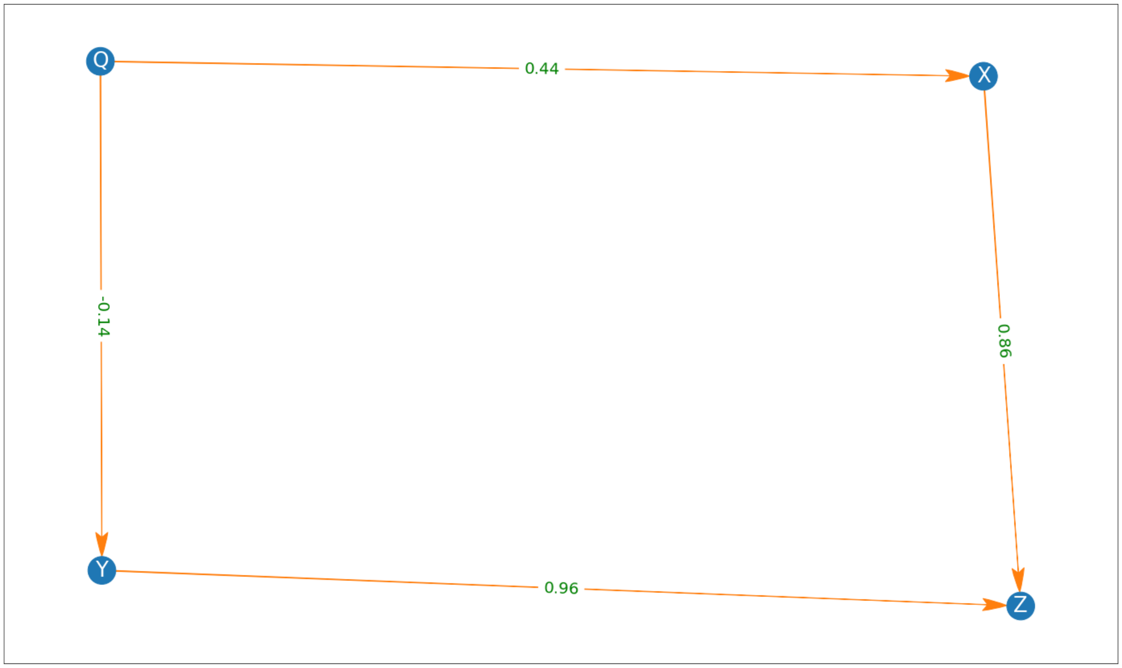
[7]:
```
from pgmpy.estimators import PC
c = PC(df)
max_cond_vars = len(keys) - 2


model = c.estimate(return_type = "dag", significance_level = 0.01,
                   max_cond_vars = max_cond_vars, ci_test = "pearsonr")
```

```python
edges = model.edges()
pcorr = df.pcorr()
weights = {}
for edge in edges:
    print(edge, ":",pcorr[edge[0]].loc[edge[1]])
```

```
Working for n conditional variables: 2:
100%|                        | 2/2 [00:00<00:00,
19.87it/s]C:\ProgramData\Anaconda3\lib\site-packages\pgmpy\estimators\PC.py:369:
UserWarning: Reached maximum number of allowed conditional variables. Exiting
  warn("Reached maximum number of allowed conditional variables. Exiting")
Working for n conditional variables: 2:
100%|                        | 2/2 [00:00<00:00, 16.45it/s]

('X', 'Z') : 0.8555687048039682
('Y', 'Z') : 0.9606500083793457
('Q', 'X') : 0.43924930511512955
('Q', 'Y') : -0.1425291548117913
```

[8]:
```python
from matplotlib.patches import ArrowStyle

def graph_DAG(edges, df, title = ""):
    pcorr = df.pcorr()
    graph = nx.DiGraph()
    edge_labels = {}
    for edge in edges:
        edge_labels[edge] = str(round(pcorr[edge[0]].loc[edge[1]],2))

    graph.add_edges_from(edges)
    color_map = ["C0" for g in graph]

    fig, ax = plt.subplots(figsize = (20,12))
    graph.nodes()
    plt.tight_layout()
    pos = nx.spring_layout(graph)#, k = 5/(len(sig_corr.keys())**.5))

    plt.title(title, fontsize = 30)
    nx.draw_networkx(graph, pos, node_color=color_map, node_size = 1200,
                     with_labels=True,  arrows=True,
                     font_color = "white",
                     font_size = 26, alpha = 1,
                     width = 1, edge_color = "C1",
                     arrowstyle=ArrowStyle("Fancy, head_length=3, head_width=1.
 ↪5, tail_width=.1"), ax = ax)
    nx.draw_networkx_edge_labels(graph,pos,
                                 edge_labels=edge_labels,
```

```
                                    font_color='green',
                                    font_size=20)

graph_DAG(edges, df)
```



[9]:
```
def graph_stats(df, edges):
    statistics = {}
    for (node1, node2) in edges:
        covar = [node for node in df.keys() if node not in [node1, node2]]
        statistics[(node1, node2)] = df.partial_corr(x = node1, y = node2,␣
 ↪covar=covar,
                        method = "pearson")
        statistics[(node1, node1)] = statistics[(node1, node2)]

        print(node1, node2, statistics[(node1, node2)], sep = "\n")
graph_stats(df, edges)
```

```
X
Z
            n         r          CI95%          p-val
pearson  1000  0.855569  [0.84, 0.87]  4.821511e-287
Y
Z
            n        r          CI95%  p-val
pearson  1000  0.96065  [0.96, 0.97]    0.0
```

```
Q
X
            n         r          CI95%          p-val
pearson  1000  0.439249  [0.39, 0.49]  2.496327e-48
Q
Y
            n         r           CI95%     p-val
pearson  1000 -0.142529  [-0.2, -0.08]  0.000006
```