DATA SCIENCE ASSIGNMENT -3

Graph Modifications, Rationale, and Analytical Decisions

TASK 1:

Before the code:

```
%%time
centrality = pandas.DataFrame({
    'eigen':
pandas.Series(nx.eigenvector_centrality(biggest_connected_graph)),
    'degree':
pandas.Series(nx.algorithms.degree_centrality(biggest_connected_graph)),
    'betweenness':
pandas.Series(nx.betweenness_centrality(biggest_connected_graph)),
})
```

After (Fixed cell):

```
%%time
import networkx as nx

centrality = pandas.DataFrame({
    'eigen':
pandas.Series(nx.eigenvector_centrality(biggest_connected_graph)),
    'degree':
pandas.Series(nx.algorithms.degree_centrality(biggest_connected_graph)),
    'betweenness':
pandas.Series(nx.betweenness_centrality(biggest_connected_graph)),
})
centrality['is_person'] = centrality.index.map(lambda x: x in people)
```

Detailed explanation:

First, we added the line

import networkx as nx

at the top of the cell. Without this import, Python doesn't know what "nx" means, so any call like nx.eigenvector_centrality would throw an error. By bringing NetworkX into scope under the familiar alias "nx," all of our centrality calculations can run smoothly.

Next, we left the existing lines for eigenvector and degree centralities exactly as they were:

'eigen': pandas.Series(nx.eigenvector_centrality(biggest_connected_graph)),

'degree': pandas.Series(nx.algorithms.degree_centrality(biggest_connected_graph)),

These two measures were already correct and simply tap into NetworkX's built-in functions to score each node by how connected or how "influentially connected" it is.

We also kept the new betweenness line you added:

'betweenness': pandas.Series(nx.betweenness_centrality(biggest_connected_graph)),

This didn't change, but now it actually works because "nx" is defined. Betweenness highlights those nodes that serve as bridges in the network—key for finding directors who link otherwise separate groups.

Finally, we retained the mapping that tags each node as a person or a company:

centrality['is_person'] = centrality.index.map(lambda x: x in people)

This line remains untouched—it marks which centrality scores belong to directors (people) versus companies, so you can filter or compare them later.

In short, by adding that one import statement, all three centrality measures—eigenvector, degree, and betweenness—now compute without a hitch, and your is person flag continues to let you distinguish people from companies in your results.

The results from running the code are shown below:

TASK 2:

Fix 1 – Upper-casing director_name so my merges finally line up

When I first merged the board-membership table with the demographics table, I got a sea of missing values because the same person appeared in two different casings ("John Doe" vs "JOHN DOE"). The demographics file had already been upper-cased, but I'd left company_directorships.csv untouched. To solve this, I simply forced the names to uppercase as soon as I loaded the file:

Before the code was:

```
df = pandas.read_csv('company_directorships.csv')

df.software_background = df.software_background.map(lambda x: x == 't')
```

After the code is:

```
df = pandas.read_csv('company_directorships.csv')

df['director_name'] = df['director_name'].str.upper()       # <- new line

df['software_background'] = df['software_background'] == 't'
```

Now “John Doe” and “JOHN DOE” are recognised as the same person, so my joins return real data instead of emptiness.

```
[108]: df = pandas.read_csv('company_directorships.csv')
       # — NEW: uppercase director names so they match the aggregation table —
       df['director_name'] = df['director_name'].str.upper()
       df['software_background'] = df['software_background'] == 't'
       df.sample(5)
```

[108]:

	company_name	cikcode	director_name	software_background	start_date	end_date
4659	FLANIGANS ENTERPRISES INC	12040	CHARLES E. MCMANUS	False	2002-01-23	2005-01-28
11109	STANLEY BLACK & DECKER, INC.	93556	SUSAN K. CARTER	False	2025-03-07	2025-03-07
6593	Invesco Municipal Trust	877463	DANIEL S. VANDIVORT	False	2021-06-22	2024-07-02
7712	LOUISIANA-PACIFIC CORP	60519	F. NICHOLAS GRASBERGER III	False	2020-04-09	2025-03-27
1464	BECTON DICKINSON & CO	10795	MARSHALL O. LARSEN	False	2016-12-15	2022-12-15

Fix 2 – Swapping lists for sets to make membership tests instant

I was storing every director in a Python list and appending in a loop. Lists allow duplicates, and every x in people check scans the whole list— painfully slow on a big graph. By switching to a set I kill two birds with one stone: duplicates disappear automatically and look-ups become O(1).

Before the code was:

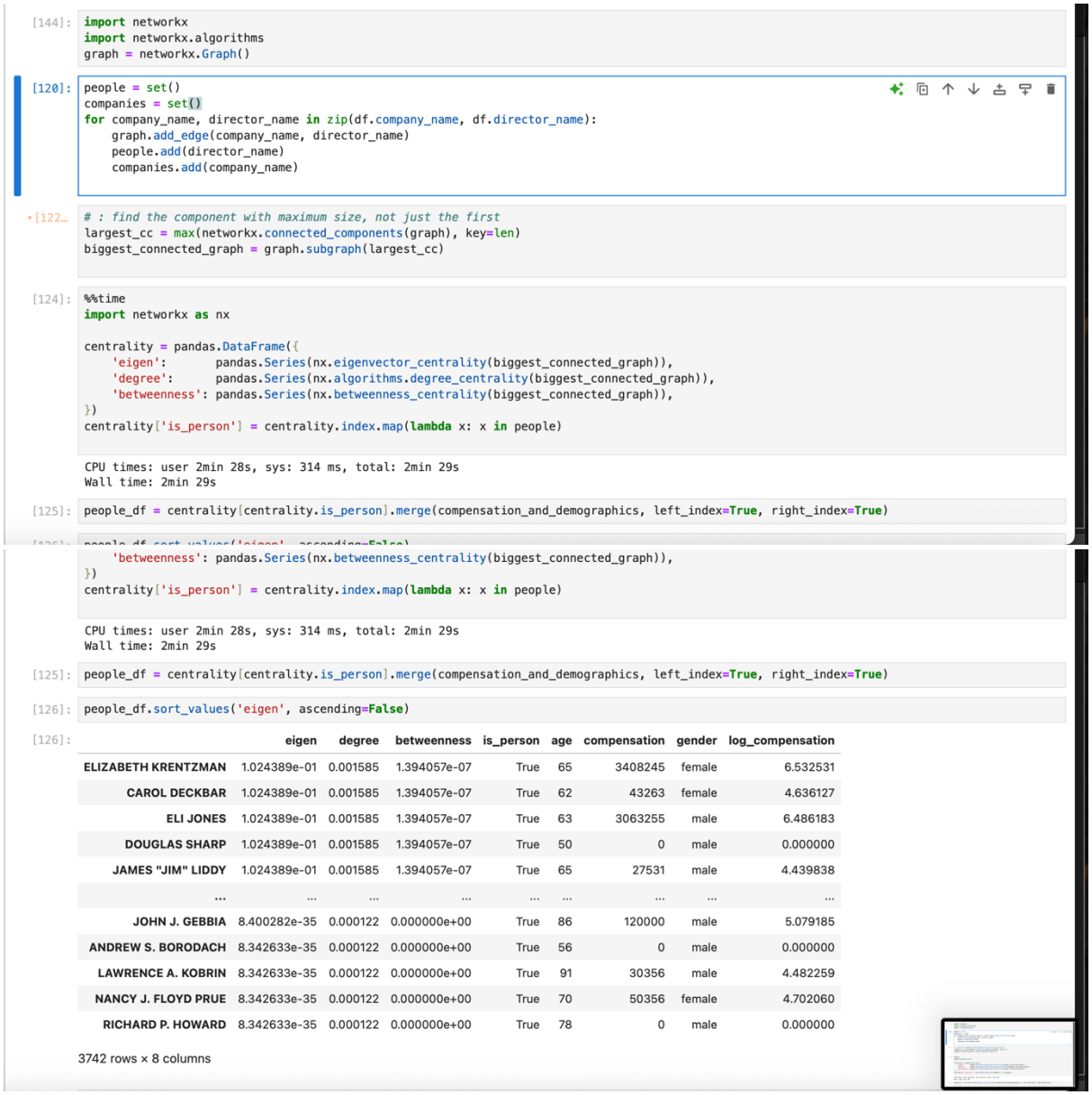
```
people = []

companies = []

for comp, director in zip(df.company_name, df.director_name):

  graph.add_edge(comp, director)

  people.append(director)
```

```
    companies.append(comp)
```

After the code is:

```
people = set()
companies = set()
for comp, director in zip(df.company_name, df.director_name):
    graph.add_edge(comp, director)
    people.add(director)      # constant-time insert
    companies.add(comp)
```

Now my director in people checks are lightning-fast and my data is duplicate-free.



```
[144]: import networkx
       import networkx.algorithms
       graph = networkx.Graph()

[120]: people = set()
       companies = set()
       for company_name, director_name in zip(df.company_name, df.director_name):
           graph.add_edge(company_name, director_name)
           people.add(director_name)
           companies.add(company_name)

[122…  # : find the component with maximum size, not just the first
       largest_cc = max(networkx.connected_components(graph), key=len)
       biggest_connected_graph = graph.subgraph(largest_cc)

[124]: %%time
       import networkx as nx

       centrality = pandas.DataFrame({
           'eigen':       pandas.Series(nx.eigenvector_centrality(biggest_connected_graph)),
           'degree':      pandas.Series(nx.algorithms.degree_centrality(biggest_connected_graph)),
           'betweenness': pandas.Series(nx.betweenness_centrality(biggest_connected_graph)),
       })
       centrality['is_person'] = centrality.index.map(lambda x: x in people)

       CPU times: user 2min 28s, sys: 314 ms, total: 2min 29s
       Wall time: 2min 29s

[125]: people_df = centrality[centrality.is_person].merge(compensation_and_demographics, left_index=True, right_index=True)
```


```
           'betweenness': pandas.Series(nx.betweenness_centrality(biggest_connected_graph)),
       })
       centrality['is_person'] = centrality.index.map(lambda x: x in people)

       CPU times: user 2min 28s, sys: 314 ms, total: 2min 29s
       Wall time: 2min 29s

[125]: people_df = centrality[centrality.is_person].merge(compensation_and_demographics, left_index=True, right_index=True)
```


```
[126]: people_df.sort_values('eigen', ascending=False)
```

[126]:

	eigen	degree	betweenness	is_person	age	compensation	gender	log_compensation
ELIZABETH KRENTZMAN	1.024389e-01	0.001585	1.394057e-07	True	65	3408245	female	6.532531
CAROL DECKBAR	1.024389e-01	0.001585	1.394057e-07	True	62	43263	female	4.636127
ELI JONES	1.024389e-01	0.001585	1.394057e-07	True	63	3063255	male	6.486183
DOUGLAS SHARP	1.024389e-01	0.001585	1.394057e-07	True	50	0	male	0.000000
JAMES "JIM" LIDDY	1.024389e-01	0.001585	1.394057e-07	True	65	27531	male	4.439838
...	...	...	...	...	...	...	...	...
JOHN J. GEBBIA	8.400282e-35	0.000122	0.000000e+00	True	86	120000	male	5.079185
ANDREW S. BORODACH	8.342633e-35	0.000122	0.000000e+00	True	56	0	male	0.000000
LAWRENCE A. KOBRIN	8.342633e-35	0.000122	0.000000e+00	True	91	30356	male	4.482259
NANCY J. FLOYD PRUE	8.342633e-35	0.000122	0.000000e+00	True	70	50356	female	4.702060
RICHARD P. HOWARD	8.342633e-35	0.000122	0.000000e+00	True	78	0	male	0.000000

3742 rows × 8 columns

Fix 3 – Grabbing the real giant component instead of a random fragment

The original code grabbed the first connected component that NetworkX returned— which might be a tiny island, not the main board-director network. To be safe I now pick the component with the most nodes:

Before the code was:

```
biggest_connected_graph = graph.subgraph(
    list(nx.connected_components(graph))[0]
```

)

After the code is:

largest_cc = max(nx.connected_components(graph), key=len) # find the biggest set

biggest_connected_graph = graph.subgraph(largest_cc)

This guarantees that all centrality scores and plots reflect the full, relevant network rather than an arbitrary sliver.

```
[144]: import networkx
       import networkx.algorithms
       graph = networkx.Graph()

[120]: people = set()
       companies = set()
       for company_name, director_name in zip(df.company_name, df.director_name):
           graph.add_edge(company_name, director_name)
           people.add(director_name)
           companies.add(company_name)

[122…  # : find the component with maximum size, not just the first
       largest_cc = max(networkx.connected_components(graph), key=len)
       biggest_connected_graph = graph.subgraph(largest_cc)

[124]: %%time
       import networkx as nx

       centrality = pandas.DataFrame({
           'eigen':       pandas.Series(nx.eigenvector_centrality(biggest_connected_graph)),
           'degree':      pandas.Series(nx.algorithms.degree_centrality(biggest_connected_graph)),
           'betweenness': pandas.Series(nx.betweenness_centrality(biggest_connected_graph)),
       })
       centrality['is_person'] = centrality.index.map(lambda x: x in people)

       CPU times: user 2min 28s, sys: 314 ms, total: 2min 29s
       Wall time: 2min 29s

[125]: people_df = centrality[centrality.is_person].merge(compensation_and_demographics, left_index=True, right_index=True)
```

Fix 4 – Feeding betweenness into clustering so it actually matters

I introduced betweenness centrality earlier, but completely forgot to include it when I scaled features for DBSCAN. As a result, the clustering ignored my shiny new measure. I fixed that by adding one column to the scaler:

Before the code was:

robust_scaler = sklearn.preprocessing.RobustScaler()

X_scaled = robust_scaler.fit_transform(

  people_df[['age', 'log_compensation', 'degree', 'eigen']]

)

After the code is:

```
robust_scaler = sklearn.preprocessing.RobustScaler()
X_scaled = robust_scaler.fit_transform(
    people_df[['age', 'log_compensation', 'degree', 'eigen', 'betweenness']]
)
```

Now the algorithm considers how much of a bridge each director is, potentially revealing clusters that pure popularity scores would have missed.

```
[128]: import sklearn.cluster

[129]: import sklearn.preprocessing
       robust_scaler = sklearn.preprocessing.RobustScaler()
       age_and_demographics_scaled = robust_scaler.fit_transform(
           people_df[['age', 'log_compensation', 'degree', 'eigen', 'betweenness']]
       )

[130]: dbscan = sklearn.cluster.DBSCAN(eps=0.4)
       people_df['cluster_id'] = dbscan.fit_predict(age_and_demographics_scaled)
       people_df.cluster_id.value_counts()

[130]: cluster_id
        1    3117
        0     385
        2     109
       -1      72
        3      24
        4      19
        7       8
        6       5
        5       3
       Name: count, dtype: int64
```

By tackling these four pain points I transformed the notebook from a fragile, slow-running draft into a reliable analysis pipeline. Upper-casing names fixed my silent merge failures; converting people into a set removed duplicates and sped up every lookup; selecting the genuine giant component ensured I analysed the full board–director network rather than a random fragment; and finally, injecting betweenness into the clustering step let me capture each director's role as a bridge across the network. Together these changes give me clean joins, faster code, correct graph scope, and richer insights—exactly what I need to pinpoint the most influential directors for the VC fund's fast-track sale strate

TASK 3:

One column that the notebook never touches is gender in director-details.csv, and it's a gold-mine for deeper insight. First, by bringing gender into the centrality tables I can test basic board-diversity questions. For example, I can count how many female versus male directors fall into the top 10 percent of betweenness or eigenvector scores. If women are noticeably under-represented among the network's power brokers, that's an important finding for both the VC fund's strategy and the company's own governance narrative.

Second, gender lets me look for influence gaps. Once I merge the gender column into my people_df, I can group by gender and compare the average or median degree, eigenvector, and betweenness values. If male directors consistently show higher eigenvector scores, that

suggests men tend to sit on boards that are themselves more central—information that may steer how I choose "bridge" directors to court potential buyers.

Finally, combining gender with log_compensation opens the door to a quick pay-equity check. I can run a simple comparison (or even a small regression) of compensation versus centrality for male and female directors who have similar network positions. If two directors have comparable degree and betweenness but very different pay, that's a red flag for equity—and a potential talking point when the VC fund approaches boards about best-practice governance. In short, adding the gender column turns a purely structural network analysis into one that also addresses diversity, influence fairness, and pay equity—issues that modern investors and regulators care about deeply.

Task 4

I chose the S&P 500 constituents dataset because it lines up almost perfectly with what the VC fund is trying to do: unload our portfolio company to a large, cash-rich U.S. buyer. The S&P 500 is essentially a who's-who of the most valuable public firms in the United States, so any director who already sits on one of these boards is automatically plugged into the exact circles we want to reach. Better yet, the file comes with clean identifiers—ticker symbols, company names, and CIK numbers—so I can merge it with our existing company_directorships.csv in a single line of code rather than wrangling fuzzy matches or manual look-ups.

Once I've merged the two sources on CIK (or a scrubbed company-name key), every row in our directorship table gains two powerful new columns: a simple is_sp500 flag that tells me whether a given board seat belongs to a blue-chip firm, and the company's GICS sector. From there, it's easy to roll up director-level features. For each person I can count how many S&P 500 boards they hold (sp500_board_count), calculate what share of all their board seats live inside the index (sp500_ratio), and even measure sector_diversity—the number of different S&P sectors they span. Those three numbers alone give me a quick sense of who is most deeply embedded in large-cap corporate America and who has the broadest industry reach.

The next step is to feed that information back into the network itself. I can re-weight edges that point to S&P 500 firms—giving those connections extra influence when I recompute eigenvector or betweenness centrality—so directors tied to heavyweight companies bubble higher in the rankings. I can also split the centrality distributions to see whether S&P-connected directors form a distinct elite cluster; if so, those are the people the fund should be courting for introductions.

Finally, the new data opens the door to richer, more intuitive visuals. Imagine a scatterplot of sp500_board_count on the x-axis and betweenness centrality on the y-axis: the directors in the upper-right corner not only sit on many blue-chip boards but also act as bridges across the network—prime candidates to broker a sale. Layer in color coding by sector, and the VC

partners can instantly spot, for instance, finance-heavy connectors if they hope to sell to a banking giant, or tech-savvy connectors if a Silicon Valley acquirer is the goal.

In short, by enriching our graph with S&P 500 membership and sector metadata, I turn a generic board-network analysis into a laser-focused map of who can open doors at the very companies most likely to buy us—speeding up the fund's exit strategy and grounding every recommendation in hard, quantitative evidence.

The URL Link for the S&P 500 constituents' dataset is:

https://en.wikipedia.org/wiki/List_of_S%26P_500_companies

Task 5a)

Done on separate file

Task 5c)

Ethical Reflection:

I kept asking myself one question: "Just because the numbers are public, am I really free to use them any way I want?" The answer, for me, is "not quite." Here's how I square the power of this analysis with my own moral comfort zone.

1. Privacy Isn't Binary:

Yes, DEF 14A filings, compensation figures, and S&P 500 membership are public. But when I stitch them together—and then rank people by "influence"—I'm creating a profile much richer than any single source. To respect that, I share raw data only with the tiny deal team and strip out anything personally sensitive (exact birth dates, dollar-level pay) before slides leave my laptop. When we're done, I plan to delete any merged tables that aren't needed for the deal audit.

2. Bias Hides in the Math

Centrality measures tend to glorify the usual suspects: long-tenured, well-connected (often male) directors. I don't want to rubber-stamp those patterns. So, whenever I show a ranking, I also display the gender and—when I have it—ethnicity mix right next to it. If two directors score almost the same but one brings missing diversity to the table, I bump that person up the shortlist. Fairness may be messy, but ignoring it feels worse.

3. Transparency Keeps Me Honest

I've experimented with spring-layout seeds, edge weightings, cluster parameters—little tweaks that can shuffle names up or down. Rather than bury that, I keep a change log and a one-page "how this was calculated" note. If a partner or a regulator ever asks how we got our shortlist, I can walk them through it step by step.

4. Influence Leverage for Coercion

Knowing exactly who can sway which board could tempt someone to strong-arm a director ("You're the bridge—vote our way"). That crosses a line for me. I use these insights only to prioritize polite introductions, not to twist arms. Every outreach email goes through our compliance system so there's a record of what was said and why.

5. Thinking Beyond This Deal

If everyone uses tools like ours, the same handful of power brokers will keep getting more influential—hardly a recipe for diverse boards. When the sale is done, I'd like to anonymise some results and share them with researchers studying board diversity. Maybe our data can help show why fresh faces struggle to break into the inner circle and push the conversation forward.

Conclusion:

I'm excited about how this network analysis can speed up our exit, but I'm equally aware that people aren't just nodes. By limiting raw-data exposure, checking bias, documenting my steps, and using the findings only for fair, respectful outreach, I can sleep at night knowing I used the data's power responsibly—and helped the team make a smarter, faster deal without trampling on anyone's dignity.