Assignment 1

October 7, 2020

---

*You are currently looking at **version 1.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

---

1 Assignment 1 - Creating and Manipulating Graphs

Eight employees at a small company were asked to choose 3 movies that they would most enjoy watching for the upcoming company movie night. These choices are stored in the file `Employee_Movie_Choices.txt`.

A second file, `Employee_Relationships.txt`, has data on the relationships between different coworkers.

The relationship score has value of `-100` (Enemies) to `+100` (Best Friends). A value of zero means the two employees haven't interacted or are indifferent.

Both files are tab delimited.

```
In [2]: import networkx as nx
        import pandas as pd
        import numpy as np
        from networkx.algorithms import bipartite


        # This is the set of employees
        employees = set(['Pablo',
                         'Lee',
                         'Georgia',
                         'Vincent',
                         'Andy',
                         'Frida',
                         'Joan',
                         'Claude'])

        # This is the set of movies
        movies = set(['The Shawshank Redemption',
                      'Forrest Gump',
```

```
                   'The Matrix',
                   'Anaconda',
                   'The Social Network',
                   'The Godfather',
                   'Monty Python and the Holy Grail',
                   'Snakes on a Plane',
                   'Kung Fu Panda',
                   'The Dark Knight',
                   'Mean Girls'])


    # you can use the following function to plot graphs
    # make sure to comment it out before submitting to the autograder
    #def plot_graph(G, weight_name=None):
    #     '''
    #     G: a networkx G
    #     weight_name: name of the attribute for plotting edge weights (if G is weighted)
    #     '''
    #     %matplotlib notebook
    #     import matplotlib.pyplot as plt

    #     plt.figure()
    #     pos = nx.spring_layout(G)
    #     edges = G.edges()
    #     weights = None

    #     if weight_name:
    #         weights = [int(G[u][v][weight_name]) for u,v in edges]
    #         labels = nx.get_edge_attributes(G,weight_name)
    #         nx.draw_networkx_edge_labels(G,pos,edge_labels=labels)
    #         nx.draw_networkx(G, pos, edges=edges, width=weights);
    #     else:
    #         nx.draw_networkx(G, pos, edges=edges);
```

1.0.1 Question 1

Using NetworkX, load in the bipartite graph from `Employee_Movie_Choices.txt` and return that graph.

This function should return a networkx graph with 19 nodes and 24 edges

```
In [3]: # !cat Employee_Movie_Choices.txt

In [4]: def answer_one():

            df = pd.read_csv('Employee_Movie_Choices.txt', names = ['Employee', 'Movie'], delimi

            B = nx.Graph()
```

```
        edges = [(df.loc[i,'Employee'], df.loc[i, 'Movie']) for i in df.index]

        B.add_nodes_from(df['Employee'], bipartite=0)
        B.add_nodes_from(df['Movie'], bipartite=1)
        B.add_edges_from(edges)



        return B
    answer_one()
```

Out[4]: <networkx.classes.graph.Graph at 0x7fb475041c50>

1.0.2   Question 2

Using the graph from the previous question, add nodes attributes named `'type'` where movies
have the value `'movie'` and employees have the value `'employee'` and return that graph.

    This function should return a networkx graph with node attributes {'type': 'movie'} or {'type': 'employee'}

```
In [5]: def answer_two():

            df = pd.read_csv('Employee_Movie_Choices.txt', names = ['Employee', 'Movie'], delimi

            B = nx.Graph()

            edges = [(df.loc[i,'Employee'], df.loc[i, 'Movie']) for i in df.index]

            B.add_nodes_from(df['Employee'], bipartite=0, type = 'employee')
            B.add_nodes_from(df['Movie'], bipartite=1, type = 'movie')
            B.add_edges_from(edges)

            return B

        answer_two()
```

Out[5]: <networkx.classes.graph.Graph at 0x7fb472919550>

1.0.3   Question 3

Find a weighted projection of the graph from `answer_two` which tells us how many movies differ-
ent pairs of employees have in common.

    This function should return a weighted projected graph.

```
In [6]: def answer_three():

            return nx.bipartite.weighted_projected_graph(answer_two(), employees)

        answer_three()
```

Out[6]: <networkx.classes.graph.Graph at 0x7fb475041710>

1.0.4 Question 4

Suppose you'd like to find out if people that have a high relationship score also like the same types of movies.

Find the Pearson correlation (using `DataFrame.corr()`) between employee relationship scores and the number of movies they have in common. If two employees have no movies in common it should be treated as a 0, not a missing value, and should be included in the correlation calculation.

This function should return a float.

```
In [7]: # !cat Employee_Relationships.txt

In [34]: def answer_four():

            rel_df = pd.read_csv('Employee_Relationships.txt', names = ['Employee 1', 'Employee

            R = nx.Graph()

            R.add_nodes_from(employees)

            rel_edges = [(rel_df.loc[i,'Employee 1'], rel_df.loc[i,'Employee 2']) for i in rel_

            rel_weight = dict(zip(rel_edges, rel_df['Relationship']))

            R.add_edges_from(rel_edges)

            nx.set_edge_attributes(R, values = rel_weight, name = 'relationship')

            X = nx.compose(answer_three(), R)

            df = pd.DataFrame(X.edges(data=True), columns = ['e1', 'e2', 'rel'])

            df['# movie'] = df['rel'].apply(lambda x: x.get('weight')).fillna(0)

            df['rel'] = df['rel'].apply(lambda x: x.get('relationship')).fillna(0)

            return  df.corr(method = 'pearson').iloc[0,1]

        answer_four()

Out[34]: 0.78839622217334737

In [ ]:
```