Assignment1 (Score: 9.0 / 9.0)

1. Test cell (Score: 2.0 / 2.0)
2. Test cell (Score: 2.0 / 2.0)
3. Test cell (Score: 1.0 / 1.0)
4. Test cell (Score: 4.0 / 4.0)

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 `assets/Employee_Movie_Choices.txt` .

A second file, `assets/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 [1]:
```
import networkx as nx
import pandas as pd
import numpy as np


# 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, width=weights);
    else:
        nx.draw_networkx(G, pos,);
```

Question 1

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

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

In [2]: Student's answer (Top)

```
def answer_one() -> nx.Graph:

    emp_mv = pd.read_csv(r"assets/Employee_Movie_Choices.txt", delim
iter = "\t")
    emp_mv_graph = nx.from_pandas_edgelist(emp_mv, source = "#Employ
ee", target = "Movie", create_using = nx.Graph())
    return emp_mv_graph
```

In [3]: Grade cell: `cell-029237261317f603` Score: 2.0 / 2.0 (Top)

```
assert type(answer_one()) == nx.Graph , "Your return type should be
a Graph object"
```

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 bipartite networkx graph with node attributes `{'type': 'movie'}` or `{'type': 'employee'}`

In [4]: Student's answer (Top)

```
def answer_two():
    emp_mv_graph = answer_one()
    emp_mv_graph.add_nodes_from(employees, type = "employee")
    emp_mv_graph.add_nodes_from(movies, type = "movie")
    return emp_mv_graph
```

In [5]: Grade cell: cell-c0500f828d5662fb Score: 2.0 / 2.0 (Top)

```
assert type(answer_two()) == nx.Graph , "Your return type should be
a Graph object"
```

Question 3

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

This function should return a weighted projected graph.

In [6]: Student's answer (Top)

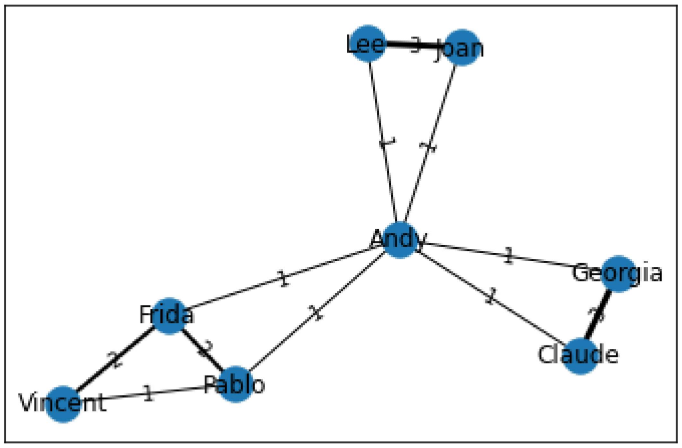
```
def answer_three():
    from networkx import bipartite
    bprojj = bipartite.weighted_projected_graph(answer_two(), nodes =
employees)
    return bproj
```

In [7]:

```
G = answer_three()
plot_graph(G, weight_name="weight")
G.edges(data = True)
```

Out[7]:

```
EdgeDataView([('Claude', 'Andy', {'weight': 1}), ('Claude', 'Georgia',
{'weight': 3}), ('Lee', 'Andy', {'weight': 1}), ('Lee', 'Joan', {'weig
ht': 3}), ('Frida', 'Vincent', {'weight': 2}), ('Frida', 'Andy', {'wei
ght': 1}), ('Frida', 'Pablo', {'weight': 2}), ('Andy', 'Pablo', {'weig
ht': 1}), ('Andy', 'Georgia', {'weight': 1}), ('Andy', 'Joan', {'weigh
t': 1}), ('Pablo', 'Vincent', {'weight': 1})])
```



In [8]: Grade cell: cell-2778b8d02bc4ca1c Score: 1.0 / 1.0 (Top)

```
assert type(answer_three()) == nx.Graph , "Your return type should b
e a Graph object"
```

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 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 [9]:

Student's answer (Top)

```
def answer_four():
    import scipy.stats as stats
    emp_relations = pd.read_csv(r"assets/Employee_Relationships.tx
t", delimiter = "\t", names = ["emp_1", "emp_2", "rel_score"])

    def fill_df(dframe: pd.DataFrame) -> pd.DataFrame:


        original_emp_pairs = list(zip(emp_relations.emp_1, emp_relat
ions.emp_2))
        data = dframe

        employees = list(zip(data.emp_1, data.emp_2))

        for reference_row in range(data.shape[0]):

            reference_pair = employees[reference_row]

            for lookup_offset in range(reference_row, data.shape
[0]):

                lookup_pair = employees[lookup_offset]

                if (reference_pair[0] == lookup_pair[1]) and (refere
nce_pair[1] == lookup_pair[0]):

                    if np.isnan(data.rel_score[reference_row]):
                        data.rel_score[reference_row] = data.rel_sco
re[lookup_offset]
                    elif np.isnan(data.rel_score[lookup_offset]):
                        data.rel_score[lookup_offset] = data.rel_sco
re[reference_row]

                    if np.isnan(data.n_common_movies[reference_ro
w]):
                        data.n_common_movies[reference_row] = data.n
_common_movies[lookup_offset]
                    elif np.isnan(data.n_common_movies[lookup_offse
t]):
                        data.n_common_movies[lookup_offset] = data.n
_common_movies[reference_row]

            if reference_pair not in original_emp_pairs:
                tmp = data.emp_1[reference_row]
                data.emp_1[reference_row] = data.emp_2[reference_ro
w]
                data.emp_2[reference_row] = tmp

        return data

    proj_graph_data = {
    "emp_1": [],
    "emp_2": [],
    "n_common_movies": []
    }
    for edge in answer_three().edges(data = True):
        proj_graph_data.get("emp_1").append(edge[0])
        proj_graph_data.get("emp_2").append(edge[1])
        proj_graph_data.get("n_common_movies").append(edge[2].get("w
eight"))

    proj_graph_df = pd.DataFrame(proj_graph_data)
    merge = pd.merge(emp_relations, proj_graph_df, left_on = ["emp_
1", "emp_2"], right_on = ["emp_1", "emp_2"], how = "outer")

    merge = fill_df(merge).drop_duplicates().fillna(0)

    return stats.pearsonr(merge.n_common_movies, merge.rel_score)[0]
```

In [10]: Grade cell: cell-b7b288e5ac139702 Score: 4.0 / 4.0 (Top)

```
ans_four = answer_four()
```

This assignment was graded by mooc_adswpy:63f4b23a9e38, v1.25.120622