Report:

The goal of my project was to look at a dataset of actors and perform bfs on that dataset (as well as on actors within specific categories). In order to get sufficient data, I used a data set that had actors and their birth and death years (so I could determine their ages), as well as a data set that contained movies and several actors that had participated in those movies, as well as the genres of those movies. Because each movie listed several actors, this allowed me to identify which actors had collaborated. Both datasets were csvs, so I read them as such and then used a modified version of the dataframe homework to process them (i.e. separating the headers, trimming values and setting strings to lowercase, and either setting empty values to 0 or skipping their rows, depending on the context). Overall, though, the datasets were both pretty clean and didn't require much processing.

My csv module allowed me to read a dataframe and process it as necessary. I created the enum ColumnVal so that I could categorize values in a column, and then manually implied equality, ordering, hashing, and display for that enum. I also created a struct DataFrame which contained the different elements of a dataframe. New allows me to create a new dataframe, and outputs self. Read_csv allows me to read a csv. It takes as input self, a path as a string, and a vector of types, and outputs a result containing a boxed dynamic error. Finally, get_column takes as input self and a column name and outputs a result containing a vector of all of the values in that column and a boxed dynamic error.

Graph allowed me to perform graph operations, including processing my dataframe into a graph, and turning my graph into a csv. I created a struct graph which contains vertex labels and a list of edges. Reverse_edges takes as input a list of edges, and outputs the reverse of those edges. Add_directed_edges modifies self and takes as input self and a list of edges. Sort_graph_lists sorts self and deduplicates it. Create_directed takes as input self and a list of edges and outputs a directed graph. Create_undirected takes as input self and a list of edges and outputs an undirected graph. BFS implements a breadth first search. It takes as input self and outputs a vector of tuples that contain start node, end node, and distance, as well as the average distance. Export_to_csv takes as input self and a path and creates a csv titled path that contains the values in self. It outputs a result containing a boxed dynamic error. Connections stores actors' connections as graph. It takes as input a dataframe, and outputs a hashmap containing an actor and their collaborators. Hash_graph turns the values in a hashmap into a graph. It takes as input a hashmap and outputs a graph.

Age.rs has a bunch of functions. Age takes as input a dataframe and returns a vector containing actors and their ages (by subtracting an actors' birth year from either the death year or the current year if they're still alive). Extract_val is a helper function that helps me process an option value as an i32. It takes as input Option<&(String, Option<ColumnVal>)> and outputs a result containing an i32 and a boxed dynamic error. Ages_bfs sorts the actors by ages, divides them into age brackets, and then computes bfs for actors in each age bracket. It takes as input a dataframe and a hashmap of actors and their ages and outputs actor graphs, actors BFS results and tuples containing youngest and oldest actors in each bracket. Within that is the function build_connections which builds connections for actors in each age bracket. It takes as inputs a vector of actors in the given age bracket and a hashmap of all the actors and their ages, and outputs a hashmap of actors and their connections within a bracket.

Likewise genre.rs has a couple functions. Genre takes a dataframe and returns a hashmap of each genre and the actors within it, by iterating through the actors in each row and the genres of the movies they're in. Genres_bfs then runs a bfs within each genre. It takes as input a dataframe and a hashmap (of actors and their connections) and then outputs a hashmap of hashmaps. The hashmap contains genres as keys whose are then the graph, bfs, and average distance for that genre, as well as a hashmap containing each actor and their connections.

Finally, main.rs prints all of these results. It reads the combined_csv and top_1000 csv, and then prints out the average distance between all of the actors, the average distance within a user-inputted age bracket, and the average distance within a user-inputted genre. It also turns the hashmap of all of the actors and their connections into a csv that it exports (I plotted that csv in python).

For my tests, I created a small_csv whose bfs results could be calculated manually, and confirmed that the average distance was 1. I then tested an arbitrary age range, the oldest one, and confirmed that it correctly calculated the average distance as 5, and then finally picked an arbitrary genre (comedy), and confirmed that its average distance was 5.

The results of my program show that while the average distance between all actors is 6, the distance between actors within a category tends to be lower. For example, the average distance between actors in the second youngest bracket was 5, and between action actors was 4. This makes sense, as these actors are more similar and thus are likely cast in more of the same movies.

In order to run my code, I used cargo run –release as run would have been very slow. When asked for a number from 1 through 4, that's asking for an age category (1 is the youngest and 4 is the oldest). Likewise, when asked to input a genre, a valid genre should be inputted (case insensitive), such as drama, sci-fi, romance, or western. Additionally, in order to plot the graph, I ran my code, and then uploaded the generated csv to google colab.

Datasets:
https://www.kaggle.com/datasets/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows
https://www.kaggle.com/datasets/rishabjadhav/imdb-actors-and-movies

Sources:
https://networkx.org/

Python: notebook
https://colab.research.google.com/drive/13HmjXv4sUu75so6akHcHIY26LNXhEBVk

```
The average number of connections between actors is: 6
Please enter a number from 1 to 4
2
The second youngest actors are between 44 and 58, and have 5 connections to each other on average.
Please enter a genre
action
Actors in the "action" genre have 4 connections to each other on average
```

```
running 3 tests
test small_test ... ok
test comedy_test ... ok
test oldest_test ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.78s
```

Actors graph plotted: You can see that there’s a bubble of super connected actors surrounded by a ring of actors who are more connected to each other than the bubbles, as well as some small separate groups.