Charlotte Chozick
CDS DS 210 Final Project Write Up

For this project, I used a data set I obtained from the website Kaggle. The data set has data imported from the online movie database IMDB, and includes columns such as the name of the movie, the director of the movie, the star of the movie, what the rating is, the genre of the movie, and more. I was interested in doing this project because while I am a data science major, I am also a film major and therefore I thought this would be an interesting comparison of the connectedness between actors and directors.

There are three modules used in my code. The first module is called csv_loader. The main goal of this module is to read the data from the CSV file and then populate a graph called CollaborationGraph with nodes and edges based on the content and data of the file. This is necessary because it serves as the bridge between analysis used in the code and the actual contents of the file. This works by firstly importing the CollaborationGraph struct from the graph_analysis module. Crate simply ensures that the modules being used exist within the same project. Then I imported the error trait that exists within the standard rust library. This is necessary for handling any errors that may occur throughout, as this may occur especially because my project deals with an external CSV file.

Then I have my first function, populate_graph_from_csv. This function takes an input parameter of the path to the CSV as a string slice and takes a mutable reference to CollaborationGraph (which needs to be mutable because it will change depending on the data that ends up assembling the graph). This expects a return of Ok on success. If there is a failure (such as an inability to read a line) an error will occur. This works by having box which is a pointer that allocated data on the heap and then a trait object dyn (which is short for dynamic) that represents any error type. This is necessary because it handles multiple different error types which is useful specifically here because there is no need to custom define whether I am working with a struct or enum which gives more flexibility. I then have a let statement which creates a CSV reader from a file path. It does this by initializing a new reader instance by opening the file specified by the string from file_path. Then if the method is successful, the reader instance is configured which parses the file but if it is unsuccessful, an error will be returned, which is also why the ? operator is used. Reader is a mutable variable because it gets modified as the processing of the CSV file occurs.

I then wrote a for loop with an if statement inside. The for loop goes over each row of the CSV file and then either represents it as Ok(csv::StringRecord) if the string is successfully parsed or if it is not, produces an error. In every row of the CSV, the data is parsed, meaning it is converted into a string record. Then the specific data is extracted (specified by the numbers which demonstrate the column number data needs to be extracted from). Then missing and

invalid data is properly handled. Furthermore, this for loop also contains the crucial .to_lowercase method which homogenizes all string characters to lowercase. This is necessary because it means that case sensitive inputs and outputs will be void, allowing for stronger comparison and takes into account user behavior. The for loop also contains two if statements which function almost identically. They both refer to specific columns within the CSV file (director and star) and print a statement for if these columns are empty. This is needed because the comparison occurs between a star and a director and if nothing is there, the system cannot just panic. This allows for the code to understand there is nothing in this particular place but allows for the code to continue parsing and makes note of the empty spot.

Finally, an edge is added to the collaboration graph. The graph stores nodes (which are actors and directors) and then edges (which are the relationships between them. .add_edge is a method which adds the edges. This works by associating score with the weight of the edge. This is used later in the add edge function, but the method works by calling star and director (which have both already been converted to lowercase from earlier and a floating point number which is the collaboration between the actor (star) and director. “Ok” then marks successful completion of the populate_graph_from_csv function. The () refers to the fact that nothing needs to be returned from this because it is simply creating the graph.

Then my next module is all about graph analysis. I first imported petgraph. Petgraph was chosen to be imported because I wanted to create a graph with nodes and edges and petgraph is a module in rust that allows for that. The graph type is a genetic data structure that has nodes (vertices) and edges (connections). I imported it as undirected, because I wanted the relationships to be bidirectional. This is because while I wanted to find the relationship between certain directors and the stars of their movies, I did not want this value to change based on the role they had. I then imported the usage of hashmap from Rust’s standard library because this will allow my code to store key-value pairs. This is necessary because the code needs to map entity names (names of stars and directors) as strings to a corresponding node index. Furthermore, this allows for avoiding duplicate nodes and quick (O(1)) lookups.

I then did #[derive(Debug)] which is an attribute that automatically will implement the debug trait into the CollaborationGraph struct. This allows the struct to be printed without a compilation error. Then the struct is defined as a public struct meaning it can be accessed from outside of the module. This is needed because this is accessed in both of the other modules. The graph takes in strings for nodes (which are the actors and directors), floats of type f64 for edges (which is the collaboration score) and again denotes that it is bidirectional or undirected. The next line then maps names to their respective indices which is NodeIndex which allows for quick lookups and ensures nodes will not have duplicates when edges are added.

Then I implemented the struct and wrote a couple of functions needed for use within my implementation function. These methods can be used to manipulate the instances of CollaborationGraph. Within this implementation block is the new function which is a public function that returns a new instance of a struct. The return type is indicated as -> Self which means that the return is CollaborationGraph. Within the self method the fields of the struct are initialized, without having to expose their internal information.

Then I have a function add_node, that takes a mutable reference to self (CollaborationGraph), meaning the collaboration graph will be altered during this state (because the return of the function is NodeIndex, a part of CollaborationGraph). The other parameter of this is the name string slice of either the director or actor. This function includes an if statement which looks up if the name already exists in the node map. This means that if the name exists within the node_map, the name is retrieved from the index. This entire step is just to avoid duplicate nodes. However, the else statement works to add a new node if it is not found. It converts the name (either actor or director) to a string and then returns the node index of the newly added node. It also updates node_map with mapping for future lookups. The method then returns the newly created NodeIndex.

My next function is called add_edge which takes a reference to self, an actor, a director, and rating as parameters. This function works by ensuring that an actor exists as a node, and then retrieves their node index. If they do not exist, the actor is added as a new node and then their node index is retrieved. A similar thing happens with the let statement for directors. Then, an edge is added between actor_index and director_index. The edge is assigned the weight of their rating which quantifies how well the public receives how an actor and director worked together.

My final function is called predict rating which is an estimated collaboration score between any actor and any director based on their previous collaboration scores. The method takes a reference to self that is immutable because here all that is wanted is to use the values to query not add. Then two string slices, one actor and one director. Finally, the function returns a value of type f64 which is a float, which will only return if both the actor and director are present in the graph. The graph then normalizes the input names to make them lowercase so that they can match the strings in the actual CSV file. This does not account for misspellings of actors names, but does allow for the user to have a better case of matching with the program because it means that their input no matter the case will be matched because this line exists to convert and compare to the already converted CSV file. The node indices are then retrieved. If an actor or director is not found, the ? operator exists which returns None immediately.

Then the actual collaboration ratings are taken. These work by taking a let statement and iterating over all the edges connected to a certain actor in a graph. edge.weight retrieves the score of the collaboration. map(|edge| *edge.weight()) then collects the weight of each edge and stores

it as a float. This number is then collected into a vector. The same process is applied to a director and their director indices. The vectors are then averaged for both the director and actor. This is done by summing the vector and dividing by the length of it. If the vector is empty, it means there are no collaborations and the average is then defaulted to 0.0. This is done with an if statement. Finally, the actor average and director average are added up and their average is found, and returned as a type float 64.

I then have my main function and a couple of tests. The modules are declared at the beginning of the module that contains the main function. The line uses csv_loader then imports the populate_graph_from_csv from the csv_loader module, so that it can be used here. Then, I imported items from the standard Rust library, which include self (which allows for more calling) and write, which is used to write outputs.

My main function specifies a file path to my CSV file. This classifies what the data imputed to the CollaborationGraph will be. The next line creates a new and empty Collaboration Graph which will represent the graph of collaborations. This is mutable because the graph needs to be able to add edges and nodes later. I then have an if statement which attempts to load data from the CSV file. If it is successful, an Ok(()) indicating success is returned and if it isn't an error will appear. This uses eprintln!: which is from the standard error stream (hence proving that the imports were necessary). If the graph is successfully populated, the user is given a message written by a print statement. Then the user is given a loop that takes in an actors name as a string and a directors name, which they are instructed to do by a print statement. They are also informed that to exit the loop they must instruct the program by typing "exit." The line of code io::stdout().flush() ensures that the prompt is immediately displayed to the user, which is necessary because without it, the prompt may not appear until after the user provides input. The input is then read and stored in actor_name. Actor_name then goes through a process of having leading and trailing whitespace removed and is converted to lowercase. This input is then compared. A similar process occurs with the prompt for the director's name.

I then used a match expression which calls graph.predict_rating with parameters of actor name and director name (which are both given by the user's input). The predict rating function is called and returns a float value. If none is returned it means no data exists for either the actor or director. However, if predict_rating returns Some(predicted_rating) then the predicted_rating value is extracted and printed in a print statement for the user to see. This rating is given to 2 decimal places. In the case that predict_rating returns None, a print statement exists to inform the user that there is no collaboration data. The loop is then closed when the user inputs exit.

I then have three tests. The first one, test_incomplete_data_handlin, verifies that the graph ignores rows with missing actor or director names when adding edges. It checks that the node_map remains empty after attempting to add invalid data. My next test is

test_non_existant_nodes which makes sure that the predict_rating method returns None if the actor and or director names that are imputed by the user do not exist in the graph. This is used to ensure that no false predictions are conjured for nodes that do not exist. Finally, my test function test_valid_prediction tests that the predict_rating method does actually compute the average collaboration score. It ensures that the prediction is in an acceptable range.

Sample Output:
Skipping row due to missing actor: StringRecord(["The Business of Show Business", "", "History", "1983", "February 15, 1983 (Canada)", "8.3", "79.0", "Tom Logan", "Tom Logan", "", "Canada", "323562.0", "", "CTV", "55.0"])
Graph successfully populated!

Enter the name of the actor (or 'exit' to quit): Elisabeth Moss
Enter the name of the director: Robert Redford
Predicted rating for collaboration between 'elisabeth moss' and 'robert redford': 6.94

If I were to do this project again, I would attempt to find another data set that perhaps had more actors in each movie so that I wouldn't just have to go off the star. Furthermore, if I could maybe then rank these actors in each individual movie based on pertinence to the plot (which in it of itself would also have multiple aspects such as screen time, dialogue, mentions, etc.) and then use this rank to inform the weight of the edges. Additionally, other roles such as writer, producer, etc. could be added and ranked as well. Then the user could be asked to input multiple aspects of a movie.

In actuality, these ratings do not actually necessarily tell us the strength of what a certain actor and director working together would be, because there are many other factors that occur when recognizing what motivates viewers to rank a movie highly. However, it does provide insight into the relationship between actors and directors, and the potential of other future collaborations meaning this project is still worthwhile as it provides a foundation for future analyses and specific outlooks into very niche relationships.