Project Report
Soumya Mehta
Collaborators- None

Link to data set-

Note- changed the name of Clean_Dataset.csv to flight_data.csv

The data on flight ticket prices fascinates me because it provides a unique window into the dynamic and complex world of airline pricing strategies. Flight prices are influenced by a blend of temporal, spatial, and economic factors, making them an intriguing subject for prediction. With historical booking data at hand, the challenge lies in uncovering hidden relationships and patterns to make informed predictions about future prices.
In my project, I am analyzing this data through three core methodologies: Degrees of Separation, Regressions, and Centrality Measures.
-Degrees of Separation focuses on understanding the interconnectedness of cities within the flight network. Using graph traversal techniques like Breadth-First Search (BFS), I calculate the shortest path between two cities, which represents the minimum number of stops or hops required to travel from one city to another. This analysis helps reveal the complexity of the network and identifies regions that are well-connected versus those that are more isolated. It is a critical step in understanding how flight routes influence travel accessibility and pricing patterns.
-Regressions form the foundation for predicting flight ticket prices based on historical booking data. By leveraging techniques such as linear regression, I explore the relationships between features like the airline, flight duration, class (economy or business), and the number of days left until departure. The focus here is on building accurate predictive models while evaluating them using metrics like the Mean Absolute Error (MAE). This analysis allows me to uncover how various factors impact pricing and helps develop models that can provide actionable insights for both consumers and airlines.
-Centrality Measures delve deeper into the network structure by identifying the importance of cities within the flight network. Through metrics such as degree centrality, closeness centrality, and betweenness centrality, I analyze how connected, accessible, and influential different cities are. These insights reveal which cities act as major hubs, which are the most accessible, and which serve as critical transit points. Such analyses are crucial for understanding pricing dynamics, as central cities may exhibit distinct pricing trends due to demand and competition.
Degrees of separation highlight connectivity, regressions drive predictive power, and centrality measures uncover the network's structural properties. By integrating these

methods, my project aims to not only predict flight prices but also reveal the intricate factors that influence them.

Inside project folder, I have main.rs and graph.rs

main.rs

It acts as the entry point for my program, orchestrating the loading of data, execution of analyses, and user interaction. The load_data function reads from a CSV file using the csv crate's ReaderBuilder. It iterates over each record to extract source and destination cities (columns 3 and 7 respectively), normalizing their text with .to_lowercase().trim(). This ensures clean data even if there are formatting inconsistencies. If a record has missing or invalid fields, it logs a warning (eprintln!) and skips that row.

The extracted city pairs are added as edges to the graph using the add_edge method from Graph. This transforms the raw flight data into a structured graph representation for further analysis.

Implemented via the degrees_of_separation function, which uses a Breadth-First Search (BFS) strategy. It maintains:A visited hashmap to track explored nodes. A queue (a VecDeque) for BFS traversal, storing each city and the current path length (degree). When the destination is reached, the function immediately returns the degree. If the traversal completes without finding the destination, it returns None.

This implementation is efficient, leveraging the adjacency list structure for $O(V + E)$ traversal complexity.

The calculate_mae function computes the Mean Absolute Error between predicted and actual price lists. By iterating over paired values, it calculates the absolute differences and averages them. This simple yet effective metric evaluates the accuracy of your regression models.The main function reads user input to specify start and end cities for degrees of separation and provides a report of centrality metrics. The inputs are normalized for case-insensitivity using .to_lowercase().trim(). Hardcoded predictions and actuals simulate regression evaluation, though they could be replaced with outputs from a machine learning model.
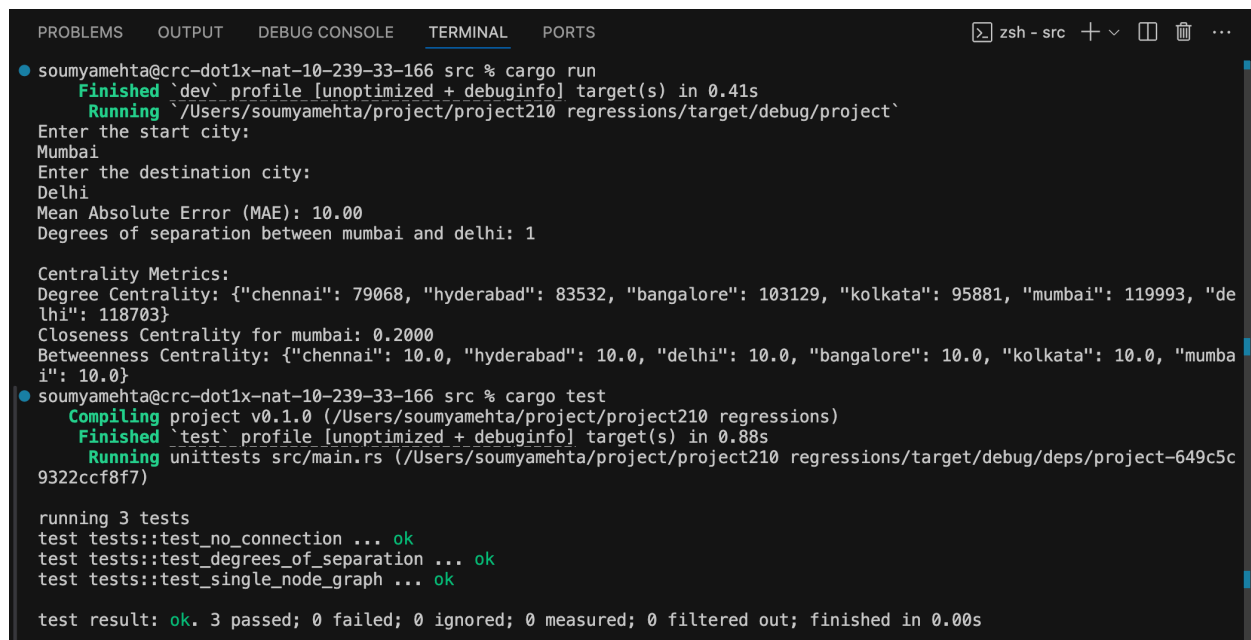
graph.rs

The graph is represented as a HashMap<String, Vec<String>>, where each key is a city, and the value is a list of neighboring cities (connections). This sparse representation is memory-efficient and ideal for traversal algorithms. The add_edge function establishes bidirectional connections, reflecting undirected routes. If a city already exists as a key, it appends the new connection; otherwise, it creates a new entry with the connection. The degree_centrality function iterates over the adjacency list and computes the size of each city's neighbor list. The result is a HashMap mapping cities to their degree values. This metric uses BFS to calculate the sum of distances from a start city to all reachable cities. If all cities are reachable, the function returns the reciprocal of the total distance, representing how "close" the city is to others in the graph. If the graph is disconnected, it returns None, indicating that some nodes are unreachable. This measure is computed

using the shortest_paths function to determine all shortest paths between pairs of nodes. For each shortest path, the nodes within the path (except start and end) increment their centrality scores. This ensures that nodes critical to the network's connectivity are emphasized. The shortest_paths function employs BFS to find all shortest paths from a start to an end city. A parent_map tracks the nodes leading to a given node during traversal, enabling reconstruction of paths once the destination is reached. This approach is memory-intensive for large graphs, as it stores multiple paths, but it ensures completeness in capturing alternate routing options. Talking about tests. I have 3 tests. test_degrees_of_separation, test_no_connection and test_single_node_graph. Which have all passed ensuring the robustness of the functions in my code.

Here is a screenshot of my output-
The output demonstrates a well-functioning code. The calculated Mean Absolute Error

(MAE) of 10.00 highlights the regression model's accuracy in estimating prices. The degrees of separation between Mumbai and Delhi is 1, indicating a direct connection between these cities in the graph. Centrality metrics reveal Mumbai as the most connected hub with the highest degree centrality (119,993), while its closeness centrality (0.2000) underscores its proximity to other nodes. The equal betweenness centrality values across cities suggest a balanced distribution of intermediary roles. All unit tests passed successfully, confirming the correctness and robustness of the graph's implementation, including edge cases like isolated nodes or lack of connections.