Boston University DS 210 Final Project – Spring 2024

Kai Gallemore

Preface

The dataset I used is the passenger activity data from San Francisco International Airport (SFO). I am interested in this data set because it provides a comprehensive overview of passenger movements over time, including information such as dates,geo-regions of destinations, and passenger counts. Analyzing this data can reveal insights into passenger flow patterns and airline connections.

https://www.kaggle.com/datasets/rajsengo/sfo-air-traffic-passenger-and-cargo-statistics/data

I. Main function (main.rs)

This code establishes the framework for the other 2 major things I am accomplishing here. It processes flight data records at SFO international airport, and ensures that the data is ready to use for my future analysis. The major components of this code are as follows:

1. Read_csv function

   This function reads the flight data from our CSV file. It parses each record into a 'FlightRecord' struct, and uses Serde for deserialization and filering records with unknown activity types. Essentially cleaning and prepping data for analysis.

This code calls upon code located in separate files for analysis.

Challenges and highlights:

It was challenging to get flight data properly loaded into the struct, I tried many different methods for this including arrays and such. In addition, it was confusing for me when I divided my code into different files, as this was something I had little practice with since it is not a thing in Python.

II. Analyzing Centrality (georeg.rs)

This code is meant to analyze flight data records at SFO international airport and construct a graph representing the flow of passengers between different geographical regions. The major components of this code are as follows:

1. Flight record struct

   Although established in the main file, the flight record struct is imported. It contains information such as the geographical region (‘geo_region’), activity type (‘activity_type’), and passenger count(‘passenger_count’) of each flight through SFO.

2. Construct_graph function

   This function takes a slice of ‘FlightRecord’ instances and constructs a directed graph where nodes represent geographical regions and edges represent passenger flow between regions. It filters records based on activity types in ‘activity_type’, which are Enplaned, Deplaned, or Thru/Transfer, and assigns edge weights based on passenger counts.

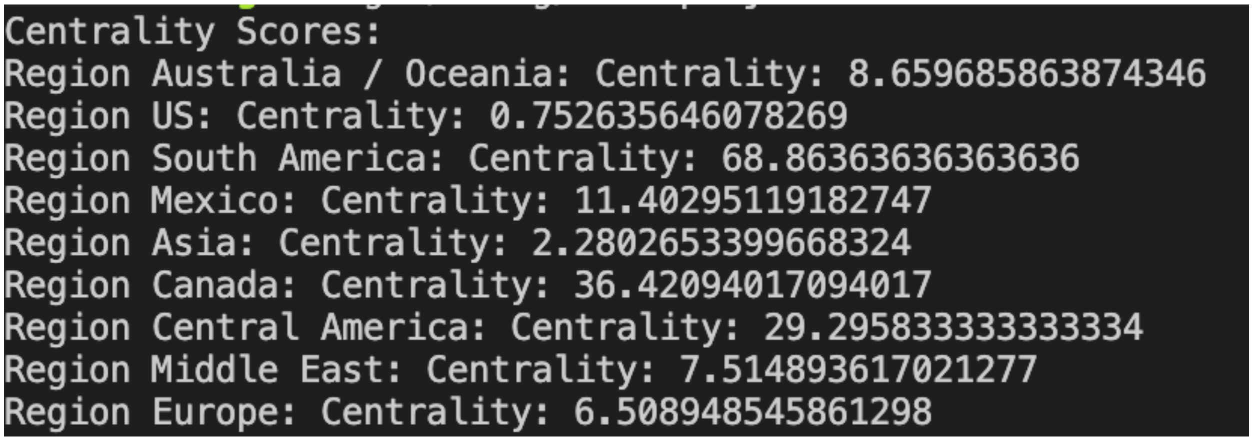
3. Analyze_centrality function

   This function analyzes the centrality (importance) of each geographical region in the constructed graph. It calculates the degree centrality for each node, which represents how connected a node is within the graph and prints it out for each region.

Challenges and highlights:

The most challenging part of this section to code was the centrality calculation. This was because I had to continuously trial and error this section and ensure that the calculation was behaving as expected. I had not actually coded a centrality function prior to this so there was a lot of referencing of lecture and class material used in this section. The HashMap was the easiest part, as the PageRank homework set me up for success. It was a similar type of implementation to that homework.

Output for this section:



```
Centrality Scores:
Region Australia / Oceania: Centrality: 8.659685863874346
Region US: Centrality: 0.752635646078269
Region South America: Centrality: 68.86363636363636
Region Mexico: Centrality: 11.40295119182747
Region Asia: Centrality: 2.2802653399668324
Region Canada: Centrality: 36.42094017094017
Region Central America: Centrality: 29.295833333333334
Region Middle East: Centrality: 7.514893617021277
Region Europe: Centrality: 6.508948545861298
```

III. Linear Regression (predcount.rs)

This code is meant to perform linear regression analysis on flight data records at SFO international airport to predict passenger counts for the year following the end of the data. The major components of this code are as follows:

1. Linear_regression function

   This function is a general purpose function to be called later. It takes two slices of values representing the independent variable (x) and dependent variable (y). It calculates the slope and intercept of the regression line using the least squares method, giving a linear approximation of the relationship between x and y.

2. Predict_passenger_counts_by_month function

   Predicts the passenger counts for each month of the following year based on the data in our CSV. It aggregates passenger counts for each month then uses our linear_regression function to fit a model to predict future counts. The predicted counts are stored in a HashMap, where the key is a string representing the year and month, and the value is the predicted passenger count.

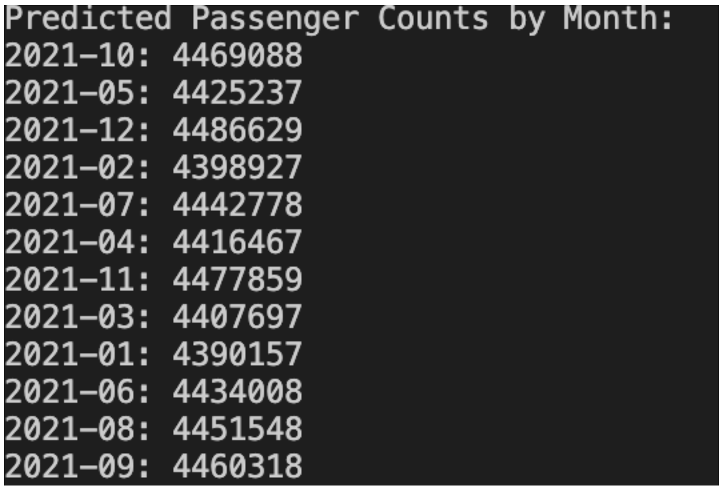
3. Flight record struct

   Like in georeg.rs, we also use the flight record struct which has attributes pertaining to activity date and passenger count, which are relevant here.

Challenges and highlights:

Turning the linear regression from a theory and an equation into functions of code was very challenging. Its not like python where we can import some library to do it for us in one line of code. Hard coding the regression was somewhat challenging for me.

Output for this section:



```
Predicted Passenger Counts by Month:
2021-10: 4469088
2021-05: 4425237
2021-12: 4486629
2021-02: 4398927
2021-07: 4442778
2021-04: 4416467
2021-11: 4477859
2021-03: 4407697
2021-01: 4390157
2021-06: 4434008
2021-08: 4451548
2021-09: 4460318
```

Work Cited

1. https://docs.rs/csv/latest/csv/struct.Reader.html#:~:text=The%20most%20flexible%20way%20to,custom%20struct%20automatically%20using%20Serde.
2. https://serde.rs/derive.html
3. https://doc.rust-lang.org/rust-by-example/error/multiple_error_types/boxing_errors.html
4. https://doc.rust-lang.org/rust-by-example/error/multiple_error_types/define_error_type.html
5. https://medium.com/@rithvik119am/building-a-linear-regression-model-in-rust-from-scratch-eee7f0da650f
6. https://www.codeproject.com/Articles/1271862/Simple-Linear-Regression-from-Scratch-in-Rust
7. https://doc.rust-lang.org/rust-by-example/std_misc/arg/matching.html
8. https://doc.rust-lang.org/std/collections/struct.HashMap.html
9. https://doc.rust-lang.org/rust-by-example/std/hash.html
10. https://web.mit.edu/rust-lang_v1.25/arch/amd64_ubuntu1404/share/doc/rust/html/book/second-edition/ch08-03-hash-maps.html
11. https://doc.rust-lang.org/std/collections/struct.HashSet.html
12. https://stackoverflow.com/questions/57857832/implement-a-graph-structure-in-rust
13. https://www.turing.com/kb/graph-centrality-measures
14. https://depth-first.com/articles/2020/02/03/graphs-in-rust-an-introduction-to-petgraph/
15. https://docs.rs/petgraph/latest/petgraph/
16. https://docs.rs/petgraph/latest/petgraph/graph/struct.Graph.html

17. https://doc.rust-lang.org/book/ch11-01-writing-tests.html

18. https://doc.rust-lang.org/rust-by-example/testing/unit_testing.html

19. https://doc.rust-lang.org/book/ch11-03-test-organization.html

20. https://web.mit.edu/rust-lang_v1.25/arch/amd64_ubuntu1404/share/doc/rust/html/book/first-edition/testing.html