DS210 Final Project Report

Project Overview

- The goal of my project was to analyze and visualize a set of bank loan data to see what impact getting approved or denied for a loan has within the data set.

- https://www.kaggle.com/datasets/udaymalviya/bank-loan-data

Data Processing

- I first downloaded the files from Kaggle and found the data was stored within a CSV file. So I created a read_loans function that took in the file from Kaggle (in CSV format) and stored the output of reading the CSV as a vector;
- I didn't immediately clean the data, but I took precautions for reading and converting the CSV file to a vector. If a row of the data is missing information or does not match the struct that I made, which has a value for every value in the data, then skip the row and add the rest of the fully complete rows to the vector.

Code Structure

1. **Modules**
    - **Module 1–adjacency:** holds the functions/methods for all the adjacency node calculations and degree distributions
    - **Module 2–plot:** creates a dot plot from the results of the adjacency module.

    Why? I separated my code into these two modules because I thought it would create an easier understanding of my code. It kept the two main 'methods' of my project separate. Lastly, once both modules are implemented, it's easy to see the functioning and output of each module when they are in main.

2. **Key Functions & Types (Structs, Enums, Traits, etc)**

    - struct LoanRecord (mod adjacency)

        - Match the data types from the CSV file and use them to store those values in the custom datatype
        - Inputs: all column labels from csv, outputs: none

- struct LoanGraph (mod adjacency)
    - Be able to create new adjacency lists that are composed of a HashMap and a HashSet
    - Inputs: edges found from the build_from_data function, outputs: none
- fn new (mod adjacency)
    - Create and return a new LoanGraph made up of an empty HashMap
    - Inputs: none, outputs: LoanGraph
    - Logic: be able to create new LoanGraphs
- fn build_from_data (mod adjacency)
    - Loop through all data points of two people's education and their loan intent, and create edges from those two data points
    - Inputs: data (converted from csv file), outputs: a new LoanGraph with edges and nodes.
    - Logic: create edges from education and loan intent using a double for loop, interset values into the graph, and return a new LoanGraph datatype.
- fn compute_degree_distribution (mod adjacency)
    - Be able to calculate the degree distributions from the adjacency list
    - Inputs: self (associated with the LoanGraph impl), outputs: HashMap of the degree distributions that were calculated
    - Logic: loop through all the neighbors of the adjacency list and increment the count of all the associated neighbors
- fn read_loans (mod adjacency)
    - Read and convert the data from loan_data.csv to a vector
    - Inputs: file path to CSV, outputs: converted data vector
    - Logic: loop through all the data points and if they match the format from the parse_record() function, then push it to the vector, else skip the row because the format does not match (data checking)
- fn parse_record (mod adjacency)
    - Helper function→ checks the format of the data from the CSV file before adding to the converted vector
    - Inputs: row from the CSV file (record), outputs: LoanRecord
    - Logic: match each row to the data stored in the LoanRecord struct to make sure the row of data is full–not missing any inputs.

- fn plot_degree_distribution (mod plot)
    - Creates a graph from the output of the degree distribution
    - Inputs: distribution HashMap and output path for the PNG, output: PNG file
    - Logic: Use the Plotters Rust package and create a chart from the degree distribution. Added a title, description of x and y axis, margin, and sized the data to the chart (min and max size).

3. **Main Workflow**
    - Mod adjacency creates the degree distribution, passes the results to mod plot creates a graph to visualize the data.

Tests

- `cargo test` **output** (paste logs or provide screenshots).

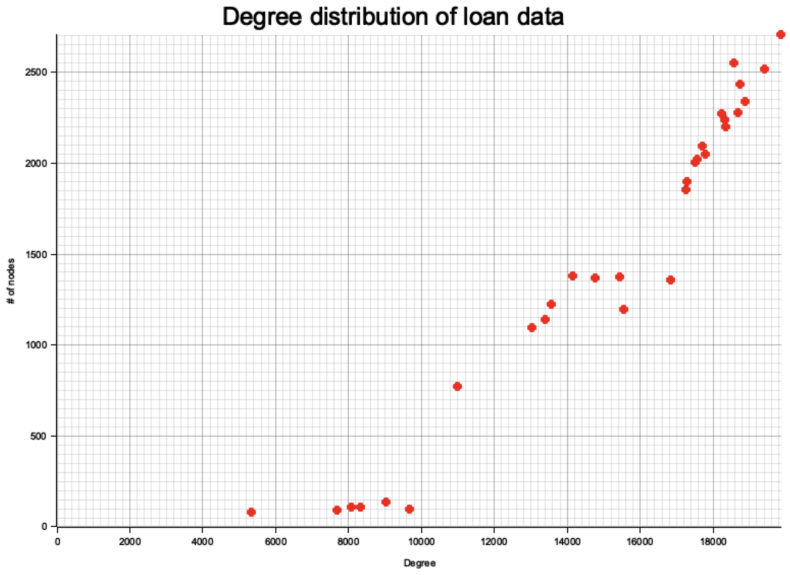
```
running 2 tests
test tests::test_degree_distribution ... ok
test tests::test_build_graph ... ok

test result: ok. 2 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

- **Test 1:** The first text I wrote creates two sample loan records, both with the same education level. The test adds them both to an adjacency list using the build_from_data from mod adjacency, and both of the sample records should be connected because they share the same education level. This is important because if this isn't calculated correctly, then the output of my program would have no meaning and be incorrect with the scope of my data.
- **Test 2:** The second test I wrote consists of checking to make sure the degree distribution is calculated correctly. I created three sample nodes that should all be connected and then ran the compute_degree_distribution function from mod adjacency to make sure the degrees of the sample adjacencies were calculated correctly; they should all be connected by 2 degrees, and they were. This is important for my data analysis because if the degree distribution is calculated incorrectly, then I would not be able to see the connections between the loan applicants.

Results

- All program outputs (screenshots or pasted).

```
Using 45000 loan records
Degree distribution:
Degree 5320: 83 nodes
Degree 7672: 93 nodes
Degree 8062: 109 nodes
Degree 8332: 107 nodes
Degree 9034: 134 nodes
Degree 9678: 95 nodes
Degree 10989: 772 nodes
Degree 13028: 1095 nodes
Degree 13391: 1138 nodes
Degree 13570: 1227 nodes
Degree 14148: 1378 nodes
Degree 14762: 1369 nodes
Degree 15434: 1376 nodes
Degree 15558: 1196 nodes
Degree 16825: 1356 nodes
Degree 17260: 1856 nodes
Degree 17271: 1901 nodes
Degree 17519: 2003 nodes
Degree 17554: 2024 nodes
Degree 17698: 2092 nodes
Degree 17794: 2052 nodes
Degree 18244: 2275 nodes
Degree 18334: 2241 nodes
Degree 18343: 2200 nodes
Degree 18574: 2550 nodes
Degree 18672: 2277 nodes
Degree 18746: 2434 nodes
Degree 18876: 2341 nodes
Degree 19426: 2520 nodes
Degree 19846: 2705 nodes
```



- I found that for the loan data I analyzed using a degree distribution that many loan applicants are similar in their education level and loan intent. It can be reasonably concluded that the data is highly dense and that most loan applicants are interconnected. It can also be said that education level and loan intent do not yield many different results; therefore, many people shared similar values, creating a very dense graph with large number of edges being connected.

Usage Instructions

- Cargo run –release

- Average runtime: 3 minutes