A. Project Overview

- **Goal**: What is the overlap between precincts and the frequency of different crimes?
- **Dataset**:NYPD Arrest Data (Year to Date)

B. Data Processing

- **Loading:** I used a csv reader after downloading the cleaned version in Python.
- **Cleaning:** I dropped any NaN values, and I also removed unnecessary columns like coordinates that I knew I wasn't going to use irregardless of what I was going to do. The python code I used to do this is located in the repo as a jupyter notebook.

Modules

- Module csv.rs - used to read the csv and create more space between the csv code and the graph code.
- Module graph.rs - used to implement the graph and clustering functions that are all implemented under the Graph struct.

Key Functions & Types (Structs, Enums, Traits, etc)

- ArrestRecord
    - Used to represent each row in the csv (each arrest done by the NYPD). It removes unnecessary information like other columns and makes it easy to identify the values and column names.
    - Input - CSV data, or more specifically each row as seen in read_csv(). Output is the struct, which is used to create the graph.
    - ArrestRecord parses the data using serde and provides the precinct number and description of the reason behind the arrest.
- Graph
    - Data structure to hold all the rows and run all important graph functions. The nodes are the precincts, and the edges are overlaps in crimes and the frequency of them.
    - Input - ArrestRecords - which is individually added in read_csv(). In dfs(), you also input the threshold, which is any number of your choosing that the weights must be above to create a connection between precincts. Output is a printed list of the weights between each precinct and another - which is just the # of overlapping arrests based on similar crimes (print_weighted_edges()). You can also get the output of clusters, if you run dfs and print_clusters() on the graph. The clusters are created based on whether the edges are above the threshold, otherwise the precinct isn't added to the cluster.
    - Precinct_frequency - creates a HashMap that stores the precinct(u32) with another nested HashMap that stores the offenses(String) and their respective number of arrests(u32).
    - Edge_weights - creates a HashMap that stores the edge weights between two precincts(u32, u32), based on the shared offenses, and the minimum sum of the number of arrests based on those shared offenses(u32).
    - Dfs - performs depth-first search to find connections between precincts based on whether edges are greater than the set threshold.

Main Workflow

- Graph's functions and structs(Graph, ArrestRecord) are used in csv.rs to create a graph immediately straight from read.csv(). Then the created graph can use all the implemented functions in graph.rs like print_weighted_edges() and dfs().

D. Tests

```
running 3 tests
test graph::frequency ... ok
test graph::test_add_single_row ... ok
test test_disconnected_nodes ... ok

test result: ok. 3 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; finished in 0.00s
```

-
    - Test frequency() - checks that the weights function grabs the minimum of shared values between precincts (1). This is necessary because we want to know that it is adjusting the weights correctly so when dfs checks the connections based on the weight and threshold, it is labeled correctly.
    - Test add_single_row() - checks that the rows are correctly added to the graph. This is the backbone of the code, because if the graph isn't made correctly nothing else will work.
    - Test disconnected_nodes() - checks that a graph's clusters are created correctly, so we know that the output is correct and not just a mistake.

E. Results

```
Weighted Precinct Graph Edges:
average edge weight:1795
```

- **Interpretation** A lot of the precincts have similar overlaps in terms of frequencies of similar offenses, although there are a couple outliers. I didn't dig too deep into why the outliers weren't in the main cluster, but if I had more time I'd want to look at the number of offenses and break it down more per offense.

```
Precinct 88 ↔ 23  → Weight: 1408
Precinct 76 ↔ 24  → Weight: 1068
Precinct 19 ↔ 17  → Weight: 1028
Precinct 48 ↔ 120  → Weight: 3350
Precinct 50 ↔ 78  → Weight: 959
Precinct 121 ↔ 61  → Weight: 2048
Precinct 60 ↔ 13  → Weight: 2390
Precinct 100 ↔ 77  → Weight: 1086
Precinct 94 ↔ 62  → Weight: 1056
Precinct 70 ↔ 44  → Weight: 3497
Precinct 112 ↔ 110  → Weight: 1685
Precinct 84 ↔ 25  → Weight: 2004
Precinct 24 ↔ 10  → Weight: 1202
Precinct 78 ↔ 113  → Weight: 1397
average edge weight:1795
Cluster 1: [483]
Cluster 2: [115, 67, 106, 30, 48, 60, 75, 66, 14, 6, 83, 43, 81, 40, 25, 105, 19, 49, 104, 109, 34, 7, 23, 1, 32, 33, 121, 102, 108, 69, 107, 73, 50, 47, 24, 41, 1
20, 88, 71, 110, 84, 113, 46, 122, 13, 70, 18, 62, 9, 5, 114, 90, 61, 44, 52, 78, 103, 42, 63, 10, 72, 79, 28, 26, 45, 112, 68, 76, 77, 101]
Cluster 3: [22]
Cluster 4: [123]
Cluster 5: [17]
Cluster 6: [20]
Cluster 7: [94]
Cluster 8: [100]
Cluster 9: [116]
Cluster 10: [111]
```

- **Building and running:** To build it, first start off with the graph functions that will define the outputs and inputs. Then you can tie it into the read_csv function and put it all together. To run the code, just do cargo run in the terminal. You can change the threshold of the weights for the dfs function in line 135 of graph.rs, and adjust any other columns you want to look at by swapping the input in line 17 csv.rs and line 29 of graph.rs. I did this with ofns_desc and pd_desc to see if the different (but similar) columns would shift the clusters by a lot.
- **Terminal interaction:** No user interaction in the terminal, just cargo run.
- **Runtime:** Around 5 seconds to run and print everything.
- **HELPFUL LINKS**
    - https://doc.rust-lang.org/std/collections/struct.VecDeque.html
    - https://serde.rs/field-attrs.html