Overview

This application provides a simple command line interface to allow simple queries on the crime dataset provided through the City of Edmonton's Open Data initiative. To provide an easier interpretation of the data, all queries either create a graph or plot data on an interactive map interface which is written to file in the directory "generated_files".

To run, the application expects the path to the database to be provided in the db_path argument. An example of such usage would be:

Python3 src/main.py --db_path PATH_TO_DATABASE

Upon running the program you will see a menu as follows:

---------- 1 --> Q1 (Bar plot of total crimes/month for year range) 2 --> Q2 (Generate map of N most populous neighbourhoods) ... ----------

Where each menu entry runs one of the four queries. All queries require additional query specific input.

This project has dependencies on the Python libraries Pandas (graphs) as well as Folium (plotting data on maps)

Modules

The program is divided into four separate modules. They are:

main:
- Handles initialization, argument parsing, and the main menu loop

menu_options:
- Functions that hold the specific logic for each query live here

a4_specific_utils:
- Key utility functions that are specific to this program

utils:
- Various functions that are used throughout the application

Control Flow

The program flow is fairly simple. A rough example is as follows:

- User picks an option from the main menu (main.py)
- The logic specific to that option executes (menu_options.py)
- Various functions specific to this lab are called (a4_specific_utils.py)
- Any bad user input is caught and the user is returned to the main menu
- A graph or plot is created on disk
- User is returned to the main menu

Testing Strategy

No automated tests were developed for this lab as it's difficult to programmatically verify that the output is correct. For example, how would you write a test to verify that a map or bar plot was generated correctly? While it's possible (although with great difficulty) to create a test for this, the test would also likely be extremely brittle, as tiny changes in how the output is displayed would likely cause the tests to fail.

Because of this, all testing was done by the manual inspection of output for a given input. Once a question seemed to produce acceptable output, edge cases were then tested. For example, most questions involve getting an N from the user. What if this N is very large? What if it's small? Will it handle tie condition correctly? As much as possible we tried to focus on these edge cases.

Debugging mainly pertained to sql queries and tracing through tables to see if values are being added up correctly. This was mostly done manually, and with inputs that included "edge" cases such as ties or 0 counts.

Group Work Strategy

Workload Distribution:

Yourui Dong: Q3, Q4
Brendan Gluth: Utility code, Q2
Allen Lu: Q1, Q4

Time Distribution:
Yourui Dong: ~ 7 hours

Brendan Gluth: ~10 hours
Allen Lu: ~7 hours

Coordination

Not too much to say here. Discord was used (a chat/voice service) to communicate, and we met once to discuss the project. Git version control was used and the project was hosted on GitHub. Video calling with screenshare