Assignment 4

Nicholas Wengel, Aida Radu, Hugh Bagan

Overview

Our application provides an interface for obtaining information about a database. To use the application, the following dependencies are required:

- `Python 3`, `pip`, `pandas`, `folium`, and `matplotlib`. SQLite3 is also used, but comes bundled with Python 3.

There is a `setup.sh` provided that will inexhaustively install these dependencies on Debian-based Linux OSs.

To start the application, run the following command:

```
$ python3 main.py a4-sampled.db
```

(Note: This assumes that your database is named a4-sampled.db and is located in the same directory as main.py.)

A menu will appear with four options as well as the option to exit. Each option will return certain information from the database. Beside each task is a character in square brackets. Type this character to run the corresponding task, and follow any next instructions.

Task 1

Type in the start and end year (inclusive), and the crime type that you are interested in.The program will generate a bar plot of the total incidents of that crime type per month, within the given yearly range. This chart will be saved to your working directory with the title `Q1-<count>.png`. The count represents the number of times task 1 has been executed.

Task 2

Type in the number of most and least populous neighbourhoods that you want to map. The map will be saved to your working directory as `Q2-<count>.html` where `<count>` is incremented each time you run Task 2.

Task 3

Type in the start and end year (inclusive), the crime type, and how many neighbourhoods that you want to map. The program will map out the top neighbourhoods for occurences of that crime type and stop at the number of neighbourhoods you specified. The complete results will be outputted to your terminal as well as any problems that may have been encountered while creating the map. The map will be saved to your working directory as `Q3-<count>.html` where `<count>` is incremented each time you run Task 3.

Task 4

Type in the start and end year (inclusive), and how many neighborhoods that you want to map. The program will map out the selected number of neighborhoods with the top crime to population ratios, as well as the most common crime in that neighborhood. The map will be saved to your working directory as `Q4-<count>.html` where `<count>` is incremented each time you run Task 4.

Application Design

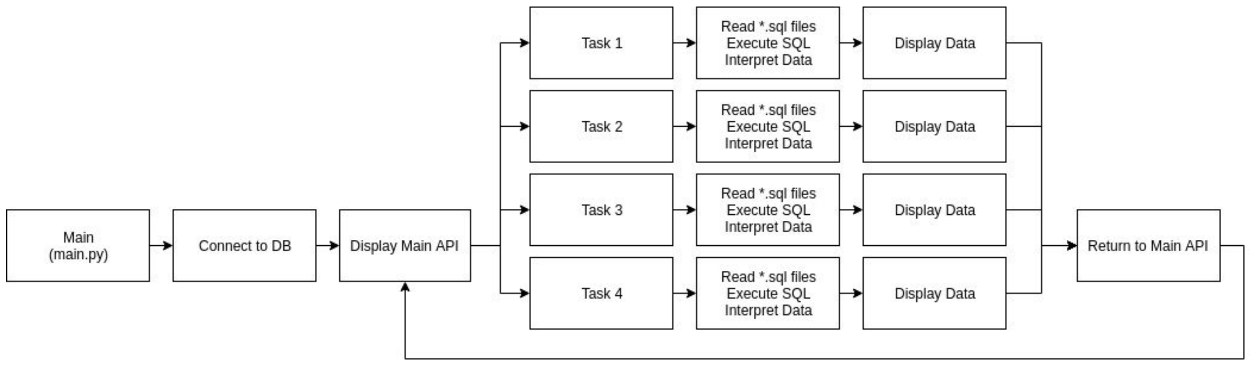
Python was used as the project's programming language, since it is high-level enough for fast software development and does not require a compile step. Python has support from libraries such as `pandas`, `folium` and `matplotlib` to ease data processing and visualization.

The database name is passed to the program as a command line argument so it doesn't have to be retyped at a dialog every time the program starts.

The core program is contained in `main.py`.The `main()` function sets up a database connection then enters a main loop that receives keyboard input from the user. The program API is displayed as a list of labelled tasks, which are executed as prompted by the user. Task selection is done using a dictionary of functions since Python does not support switch statements. Each task function receives only the connection to the database, making them pure.

SQL queries are stored as individual files to avoid cluttering `main.py`. They are not embedded, but are loaded as needed by the task functions.

Control flow diagram (made with draw.io):



Testing

Git and GitHub were used for developing and versioning the project. GitHub’s pull request system was used to ensure that new code was tested before being merged from feature branches onto our repository’s master branch. To enforce this, merging without an approving review was disabled, as was commiting directly to the master branch.

We did not create automated tests due to time constraints. The strategy was to ensure that bugs did not make it into the master branch to begin with thanks to peer reviews.

In our peer reviews, we tested invalid inputs such as: negative numbers, invalid letters and strings, and empty input. We also tested options that would result in a query returning no numeric data (for example, entering a non-existent crime type in task 1). In order to plot a bar plot with no numeric data, we used `pandas.DataFrame.fillna` to replace entire columns of null values with zeros.

To facilitate development and testing, the schema for `a4-sampled.db` was dumped into the file `schema.sql`.

Group Work Break-Down

Git was used for project versioning, allowing each project member to work on their own feature branches, and merge those features to a master branch later. This enabled us to keep track of who did what in a very exact manner.

To partition work, our Git project was hosted on GitHub, where tasks were made into Github issues. These issues were assigned to project members at random, but evenly.

Communication was facilitated through Slack and Discord for group calls.

This report was created using Google Docs to enable easy, equal collaboration.

Work distribution:

- Nicholas Wengel:
    - Q4 and and Python (20 hrs)
        - 4a.sql, 4b.sql, function task4() in main.py
- Hugh Bagan:
    - Q2 and Q3 SQL and Python (20 hrs)
        - 2a.sql, 2b.sql, 3.sql, functions task2(), task3() in main.py
- Aida Radu:
    - Q1 and Python (20 hrs)
        - 1a.sql, functions task1() in main.py
- All group members worked on:
    - main() function in main.py
    - Report
    - Testing (peer reviews), bug fixes

It’s unclear how time spent on tasks could be measured since the group did all work remotely. As well, it is likely that most tasks were not completed in one sitting, but were attacked slowly over the assignment period. As such, hour estimates are approximate.