Overview / User Guide

After launching the program, the user is prompted to enter the name of a database to query from. If the name entered is not an existing database, a new database file will be created. After the database is selected, a textual interface (the main menu) appears that lists the available functions and gives the option to quit the program. Some of these functions generate html files as output. When this happens, the html filename appears in the format “QX.html” if it is the first time the function was run and “QX-n.html” otherwise where X is the question number and n is the number denoting the first available filename in the above format. No files will be overwritten when this is done, but if a file is deleted by the user its filename may be re-used. The following options are shown to the user, and may be selected by entering into the prompt the number shown to their left:

show_barplot_range

This function shows a barplot of the number of incidents of a certain crime type that occured in each month of a given range of years. When this function is selected, the user is prompted for a valid range of years [year1, year2] inclusive of the boundary years. Then the user is asked to select the crime type which can be done by entering the number that appears to the left of each crime type into the console. If the user selects a crime where there is no crime incidents during the timeframe, a text would be output explaining this situation. After this is done the bar plot appears on screen as well as being saved as a png, and once the bar plot window is closed the program will return to the main menu.

most_least_populous

The purpose of this function is to generate a map showing the n most populous and n least populous neighbourhoods from the database (including ties). On selection of this function, the user is prompted for an integer n that determines the number of most/least populous neighbourhoods to show on the map. Once the number is entered, the output map is saved to a html file and the program will return to the main menu. The circles in the output map will show the neighbourhood name and the population in that neighbourhood when clicked.

top_n_with_crime

This function shows the n neighbourhoods that have the largest amount of a given crime within a range of years, including ties. When this function is selected, it first prompts for a range of years, with inclusive bounds. Then the program asks for a crime type to be selected, which can be done by entering the number that appears to the left of the desired crime type. Next, the program asks for the number of neighbourhoods to show on the map, after which the program will generate the output and return to the main menu. When the circles on the map

are clicked, the neighbourhood name and number of crimes of the selected type within the given range of years is shown.

n_highest_crime_population_ratios

This function generates a map showing the n neighbourhoods with the highest crime/population ratios within a range of years (including ties) as well as the most common crime/s within the same neighbourhood and range of years. After selection of this function, the user will be prompted to enter a range of years [year1, year2] inclusive of the boundary years. Afterwards, the user is prompted to input the amount of neighbourhoods (n) they want to see information for on the map. If any neighbourhoods have the same crime ratio as the n^{th} neighbourhood, they are also shown. After the number of neighbourhoods to show is input, the output will be generated and can be found as an html file. When opened, the coloured circles can be clicked on to show the name of the neighbourhood, the most common crime/s, and the crime/population ratio for the given range of years. After generating the output file, the main menu is returned to.

Software Design

First the **main** function was designed, which presents a generic user interface allowing access to the functionality of the program. This function first gets a database connection from user input and the **connect** function which simply opens the database. Then it allows the user to select one of **show_barplot_range** (Q1), **most_least_populous** (Q2), **top_n_with_crime** (Q3), or **n_highest_crime_population_ratios** (Q4). Several of these procedures share functionality, which necessitated the creation of the functions **get_filename** which takes a base filename and a file extension to find a valid filename in the format outlined in the spec, **first_n_with_ties** which finds the index in a list to obtain the first n items as well as any items that are equivalent to the n^{th} item, **get_range_years** which gets a range of years from the user inclusive of both bounds, and **get_crime_type** which gets a list of all possible crime types and allows the user to select one, then returns the selected crime type.

The function **show_barplot_range** first calls **get_range_years,** then calls **get_crime_type.** Then **show_barplot_range** uses the crime type and the range of years to query the database and show a bar plot of the amount of that crime type in each month. Afterwards, the plot is saved as a png using a name obtained from calling the function **get_filename**.

The function **most_least_populous** first gets a number from the user, then queries the database to obtain a list of the neighbourhoods along with their populations and coordinates. Places with population ‘0’ as well as places with latitude and longitude as ‘0’ are ignored. Then **first_n_with_ties** is called to find the n most populous areas and n

least populous areas from this list. The sum of all the populations in the selected areas is taken to be used to scale the circles in the output. The output is generated and saved as an html file using a filename obtained from **get_filename**.

The next function (**top_n_with_crime**) first calls **get_range_years** and **get_crime_type,** then uses the output of these functions to find the number of incidents of the given type of crime in each neighbourhood within the range of years and save it into a list. Then the user is prompted for a int which is passed to **first_n_with_ties** to find the n neighbourhoods with the highest number of crime counts from the generated list. Afterwards, the map is created and saved with a filename obtained from the **get_filename** function.

The last function (**n_highest_crime_population_ratios**) first calls **get_range_years** and then asks the user for an int n. The year range is used in a sql query that finds the crime/population ratios, most common crime, and coordinates of each neighbourhood. The output of this query is converted to a list, then passed to another utility function called **collapse_index** that takes in a list of tuples and an index, and groups the values at the index in the tuples into a list, grouping when all other values of the tuple are equal. In this case **collapse_index** was used to account for ties in the most common crime within a neighbourhood. After collapsing the index denoting the most common crime, the n neighbourhoods with the highest crime/population ratios are selected from the list with the collapsed index using **first_n_with_ties** and then the output is generated and saved with **get_filename**.

Testing Strategy

We partially tested our questions as we were working on them, and then once they were mostly finished we met up in the 291 lab room to make sure everything worked on the lab machines, as for the last assignment we had a piece of code that worked on our machines but not on the lab computers. We systematically tested all the functionality for each question, making sure that each function acted as outlined in the spec. Additionally, we were sure to “mis-type” or otherwise give invalid input to ensure all potential errors were handled so that bad input could not crash the program or be accepted by the program and cause it to return faulty results. During testing an input bug was found where one of the questions would accept substrings of crime types as input, and another bug was found in question 4 where counts were used instead of sums in an sql query, causing the question to show incorrect crime ratios. There was also bugs found where certain SQL queries accepted rows with zeroed coordinates.

Group Work Breakdown

We distributed the questions such that each person completed an amount of work worth roughly 20 marks. Lidia was tasked with question 1 as well as helping Payas with

questions 3. Payas worked on questions 2 and 3. Emery did question 4 (as well as some of the utility functions it uses) and a lot of the design document. We coordinated through email and meetings, which allowed us to efficiently distribute the work and collaborate on the design of the program. In the meetings, we established deadlines for the completion of each component of the assignment to keep our work synchronized and on schedule. Overall, everyone completed their portions of the assignment in a timely manner. Each of us put upwards of 12 hours into the assignment.