DESIGN DOCUMENT

Crime Cruiser

Team London

OVERVIEW & PURPOSE

The purpose of this document is to explain what choices we made when building *Crime Cruiser*, and why we made them. It will explore decisions made regarding data management, analysis and visualization and how these decisions impacted the design of the program. This includes how we parsed the large file, the types of questions we chose to address, and how we chose to display our answers to the user in a useful way. It will also explore improvements that could be made in the next version of *Crime Cruiser*.

DATA MANAGEMENT

Crime Cruiser uses the entire crime data file found on the Stats Canada website. This CSV file contains 6 columns which describe the occurrences of different violations in specific locations in Canada between 1998 and 2015.

The 3 columns we used for *Crime Cruiser* are: Year, Coordinate and Value. The coordinate column was particularly useful as it contains the location, violation and statistic all in one field ("LOCATION.VIOLATION.STATISTIC"). Because of this, we only had to store 3 fields as we read through the file instead of storing all of them.

The location, violation and statistic are each simply encoded by a number, so by splitting the coordinate it is easy to search through the coordinates and match what we are looking for.

We decided to only look at the "Rate per 100,000 Population" statistic (#2) because we felt it would allow accurate comparison between locations of the actual rate of a violation, since it is balanced by the population. Focusing on just one statistic type allowed us to store much less data which simplified the parsing of the file.

To parse the file, *Crime Cruiser* does the following:

- Opens the file and stores the contents in an array
- Loops through each line of the file (each element of the array) and parses it by comma into 6 different fields (1 for each column)
- Using a regular expression, it checks the coordinate field (field #5) for a coordinate of the from “ANYTHING.2” (since we only look at statistic #2, the rate)
- If the coordinate for that line matches the form we are looking for, we store the YEAR, COORDINATE and VALUE for that line in a hash.
- Each hash is pushed onto an array.

Parsing the entire file takes 20-25 seconds on the Mac lab computers. We end parsing with an array. Each element of that array is a hash for each line of the file whose coordinate ends in ‘.2’. The hash contains the year, coordinate and value for that line.

DATA ANALYSIS

Once *Crime Cruiser* has parsed the data into an array of hashes, we analyze the data through 5 different question types. The user may answer one question type at a time. Below are the 5 types. Fields to be filled in by the user are *in italics.*

Question 1: What *province/city* had the *highest/lowest* rate per 100,000 population of a *violation* in a *year*?

Question 2: What *province/city* had the *highest/lowest* percent change of a *violation* from a *start year* to an *end year*?

Question 3: What *provinces/cities* had the *top/bottom n* rates per 100,000 population of a *violation* in a *year*?

Question 4: What *provinces/cities* had the *top/bottom n* percent changes of a *violation* from a *start year* to an *end year*?

Question 5: Construct a line graph of the rate per 100,000 population of all violations in a *specific location* from a *start year* to an *end year*.

The first 4 questions involve finding the *location* with the highest or lowest rate of a specific violation over a given time frame.

The results for **questions 1 and 3** are calculated using the same function. Once the user

enters all the required fields…

- Some of their entries are passed to a function
    - PROVINCE OR CITY, VIOLATION, YEAR
- If the user is searching for a province, the program builds a set of coordinates of the form: “PROVINCE.VIOLATION.2” and stores them in an array
    - Violation is always the same (the number they entered), but a different coordinate is created for each province (13 total coordinates are made).
    - The same process occurs for cities.
- The program then loops through the array of hashes that was created when we parsed the file at the start
    - For each hash in that array, the program checks if the year entered by the user equals the year in the hash
    - If so, it loops through each coordinate that was built from the user’s entries at the start of the subroutine.
        - If the coordinate in the hash is equal to one of the user-built coordinates, the program stores the value for that hash (the rate per 100,000 population) in a new array.
        - An array of the coordinates is simultaneously created so we know not only what the value is, but where it occurred.
- Once every value (one for each location) has been found and pushed onto an array, the value/coordinate arrays are sorted simultaneously using a bubble sort in descending order.
- These arrays are passed by reference out of the function to be visualized.

The results for **questions 2 and 4** are calculated in the same function, which is similar to the function for questions 1 and 3 except a percent change is calculated.

- The same process is followed as for the first function, but this time TWO years have been passed in (a start year and an end year)
- So, the year-matching loop happens twice: once for the start year and once for the end year. This results in 4 arrays (start values, end values, start coordinates and end coordinates)
- A percent change is calculated for each location by subtracting the value for the start year from the value for the end year and dividing by 100.
- The percent changes for each location are stored in an array, then bubble sorted in descending order along with their coordinates.
- The 2 sorted arrays (values/coordinates) are returned from the function

The **fifth question** is a graphing question. The user provides 3 fields: a specific province or city, a start year and an end year. The program…

- Builds a coordinate for each violation using the user's location number:
    - "LOCATION.VIOLATIONS.2"
    - Where location is constant but violations cycle (1 coordinate per violation)
    - These coordinates are stored in an array
- The program loops through each year from the start year to the end year
    - For each year, the program loops through the array of hashes created from the crime data while parsing
    - If the current year equals the year inside the hash, the program proceeds to loop through the array of coordinates built from the user's entries
    - If the current coordinate equals the coordinate in the current hash, that coordinate is split (by period).
        - The violation number (the second part of the coordinate) is extracted and its corresponding name is found.
- This information is written to a uniquely named text file, in this format:
    - "Violation", "Year", "Rate"
- Once every value has been found and written to the text file, a line graph is built from the text file using R, which plots all the rates found in each line of the text file. This graph is created in a pdf file.

DATA VISUALIZATION

- For **questions 1 and 2**, the result is outputted in a sentence format. *Crime Cruiser* first displays the user's question to make sure they know exactly what they're asking. Then it tells them the answer in sentence format.
- For **questions 3 and 4**, the result is visualized in a list format. It is showing the top/bottom n values, so an indexed list makes the result easy to read and interpret
- For **question 5**, the result is visualized as a line graph constructed using R. The user is provided with the name of the PDF file containing the graph, and must navigate to the project folder to find it.

DESIGN DECISIONS

The design decisions made when designing *Crime Cruiser* all revolved around making the program as useful as possible with the least amount of effort. We tried to maximize the

amount of questions our program could answer, while minimizing the amount of code we had to write. By allowing the user to pick and choose so many factors (province or city, highest or lowest, 15 different violations, and any year) we multiplied the usefulness of our program exponentially.

All fields are filled in using numbers instead of words. For example, the user types in 1 for province or 2 for city, and they type in the violation number as opposed to the name. This is not only helpful for the user to speed up the interface, but it also makes it easier for us to interpret their answer and validate their input. The user interface was designed with numbers so it is clean, simple and effortless. It is pretty hard to mess up when filling in the fields, and if the user does the program just prompts them to enter their answer again. Crime Cruiser also displays the full question the user asked directly above their result, which is useful because it ensures they are really getting the answer they wanted and didn't make a mistake.

The choice to use hashes was motivated by the fact that they are unordered (unlike arrays) so they are easier to search through and manipulate. This made parsing and finding the data we needed for each question a lot easier and avoided the need for splitting the data into files.

NEXT STEPS

- Allow the user to choose which type of statistic they would like to look at (not just the Rate per 100,000 Population).
- Instead of reading through the file line-by-line each time the program is started, we could store the data in files to make this step much faster.
- The modularity of the code could be improved. We used one function to answer question 1 & 3, one to answer questions 2 & 4, and one to answer question 5 (3 total). However, the first two functions start similarly and could probably be combined into one to simplify the code!
- For the graphing question, we could allow the user to pick and choose which violations they would like to display so the graphs would be less crowded.
- Allow the user to exit from the question they are currently filling in.
- Add a snappy GUI with drop down menus and pretty graphics!
- Produce bar graphs for the first 4 questions.

Thank you for reading our design document! We hope you enjoyed using *Crime Cruiser*.