INFO1903: RAINFALL VS FATALITIES

INFO1903 Project

Section I: Data analysis

Situation

For this project, I wanted to analyse state rainfall data alongside road fatalities, with the primary aim of finding a correlation between the two.

Data Sources

Online Sources

The Australian Road Deaths Database which contains information about the crashes and the fatalities.
The Bureau of Meteorology's Daily Rainfall Data for the station of Flemington in Victoria.

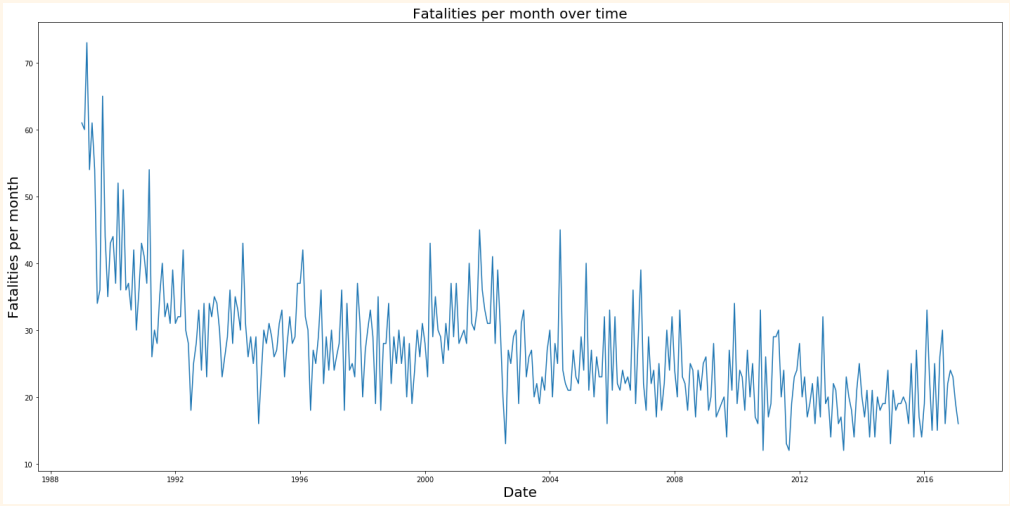
Download Links:

Crash database (csv)
Crash database legend (pdf)
Rain database (csv)

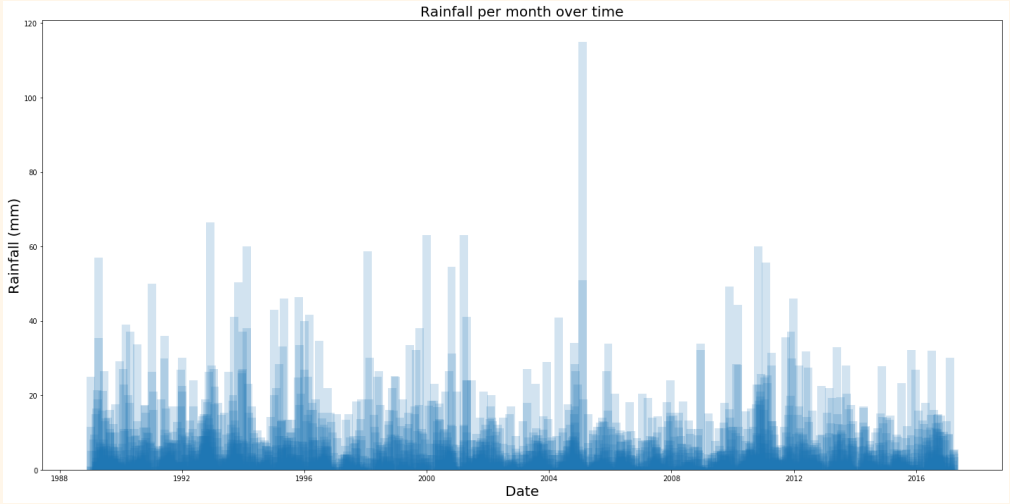
Graphs

After setting up the data, I first graphed the rainfall over time and the crashes over time to see if I could spot any trends among the separate graphs:

Car Crashes per Month

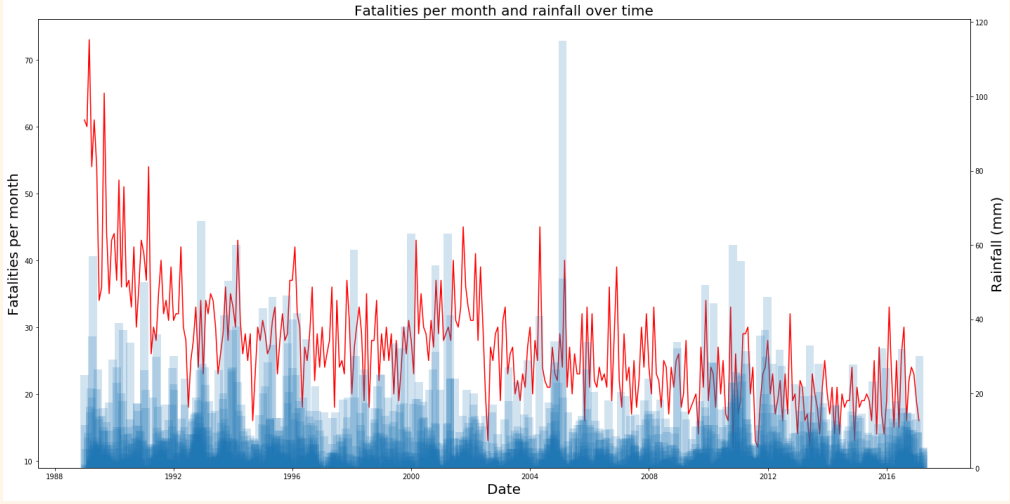


Monthly Rainfall



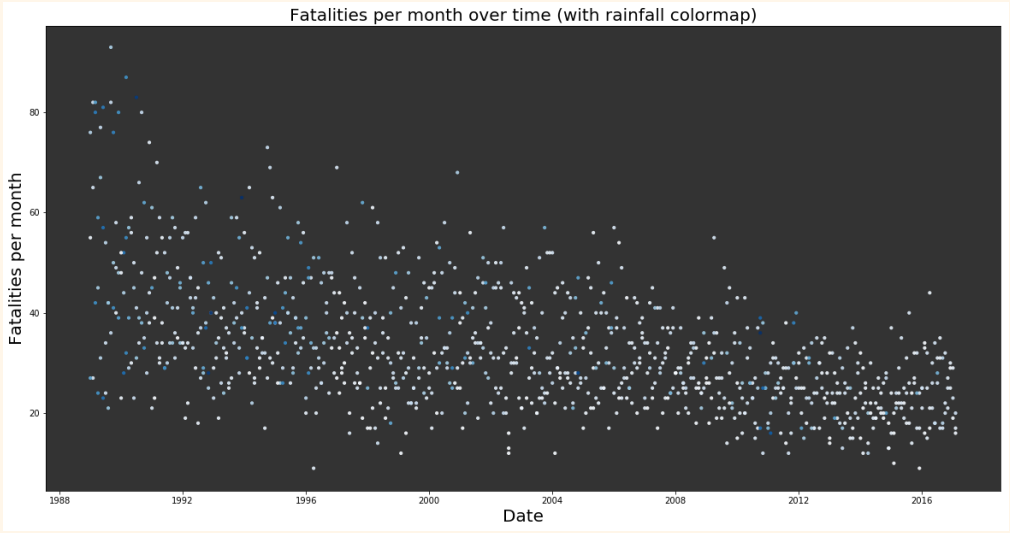
Crashes and Rainfall over Time

After having graphed the data separately, I graphed them on top of each other to get a better idea at a correlation:



This final graph is the same crashes/rainfall data but in a different style: the data points are the fatalities per month, but the rainfall is instead represented as the colour of the points.

I thought that this would help visualise the correlation, but it doesn't really work.



Discussion

Trends

The crashes/time graph shows that the average number of fatal car crashes per month has decreased since 1989, something which I expected to see. This is most likely due to an increase in car safety, road rules, and law enforcement since 1989.

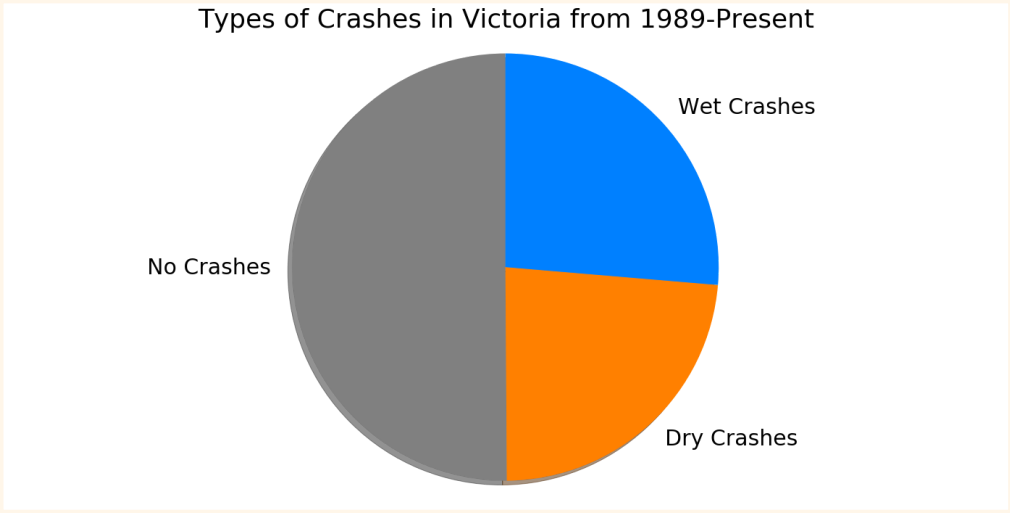
The spike at the start is because of the Kempsey Bus Crash, cited as the most deadly road accident in Australia's history. (Wikipedia Article)

In the rainfall/time graph, there is an outlier around 2005 - a heavy rain event. (BOM report)

Results

There did seem to be some correlation between rainfall and crash rates, just by looking at the graph. However, the correlation is vague, and I decided that graphing over time wasn't going to work.

After doing some calculations (described in Section II), I generated this pie chart:



This chart shows that there are indeed more crashes when it is raining then there are dry crashes, if only slightly. This was pretty much what I expected, although I expected the difference to be more prominent.

Further Research

The obvious option for further research is to expand the amount of data analysed. I only examined Victorian data; you could easily expand the queries to visualise data from other states. I don't know what governmental data is provided overseas, but that is definitely an option for further research.

The crash data also gave information on age and gender, which is another path of research you could take.

Section II: Data Generation

Getting the data

The website had two datasets available: one for each crash, and one for each fatality. I chose to use the one per crash, as I was not interested in statistics such as gender or age. However, the process required to obtain and store this data is available with the methods for the other files.

Due to the nature of the weather, I decided that getting the "total national rainfall" was not precise enough. Because the crash data sorted by state (and did not give a precise location), I decided to pick a state and use only crash data from that state.

The BOM data provides rainfall data for every weather station back to the 19th century. As my crash data location had state-level precision, I reasoned that if I chose a weather station in the middle of a state, it would give me the best approximate for "average statewide weather".

I chose Victoria, as it is a relatively small state with a weather station (Flemington station) somewhat near both the center of the state and the capital city, where I reasoned the most crashes would occur.

Storing in PostgreSQL

Database Schema

`crashes` **Table**

Column	Type	Description
`crashid`	`character(13)`	Internal crash ID
`state`	`character varying(3)`	State that the crash occured in
`day`	`integer`	Day of the crash
`month`	`character varying(10)`	Month of the crash (long name format)
`year`	`integer`	Year of the crash
`hour`	`integer`	Hour of the crash
`minute`	`integer`	Minute of the crash
`crashtype`	`character varying(16)`	Internal type of the crash
`fatalities`	`integer`	Number of fatalities
`bus`	`boolean`	Was a bus involved?
`heavytruck`	`boolean`	Was a heavy truck involved?
`articulatedtruck`	`boolean`	Was an articulated truck involved?
`speedlimit`	`integer`	The speed limit of the crash

`rainfall` **Table**

Column	Type	Description
`year`	`integer`	Year of the measurement
`month`	`integer`	Month of the measurement (in integer form)
`day`	`integer`	Day of the measurement
`rainfall`	`double precision`	Amount of rainfall
`period`	`integer`	Period measured
`quality`	`character(1)`	Quality of data

Entering into database

After creating the tables, I needed to copy the data from the `.csv` files. I tried the usual `COPY [table name] FROM csv WITH CSV HEADER;` command, but it complained about file permissions. After some reading of Postgres documentation (and some helpful StackOverflow articles), I found that Postgres provides the `\copy` command within the `psql` prompt. Fixing

this bug was as simple as putting a backslash in front of `copy`, and it had the correct permissions to import.

Issues

Date Formatting

Looking at the above schema, you might notice: the `month` field of the rainfall table is an `integer` type, but `crashes.month` is a `varchar(10)`. `crashes.month` is a long month name, e.g. `January`, `February`.

This was a problem. I had two different formats of data that were needed to do an SQL JOIN. Luckily, PostgreSQL has a very good set of date formatting commands.

I decided to leave the tables as they were, and convert the data on the fly when doing the SQL JOIN. The following SQL functions will convert a date in long format to an integer:

```
extract(MONTH from to_date(concat(crashes.month, ' 2000'), 'M
```

Querying

The queries I used were fairly simple. I mostly used implicit joining to join the rainfall and crashes databases (with the above `extract` function to integrate the dates:

```
select rainfall, fatalities, rainfall.year, rainfall.month, r
  from crashes, rainfall
 where rainfall.year = crashes.year
   and rainfall.month = extract(MONTH from to_date(concat(cra
   and rainfall.day = crashes.day
```

This query selects rainfall, fatalities, and date from the database, joining on dates.

I eventually added a `and crashes.state = [state]` line in, replacing `[state]` with the state I was querying.

Graphing

Graphing was the stage of the project that took the longest. Learning and becoming familiar with Matplotlib and the psycopg2 postgres library for python took some time, but the majority of my time was spent thinking about how I could organise my data and then graph it in the best format.

The psycopg2 library connects to your database (using `psycopg2.connect(dbname, user)`) and gives you a cursor. The cursor is the

object from which you can perform queries on your data, and it acts as a proxy between your Python code and your database.

The first issue I had was that `cursor.execute('query')` wasn't returning anything. I later learnt that `cursor.execute()` does not return data, just executes the query, and that I had to `cursor.fetchall()` to recieve the data in a list of tuples.

I used *many* list comprehensions, and I am very glad that I was already comfortable with how to use them from previous Python/Haskell experience. List comprehensions allowed me to filter and modify lists in place, as opposed to having to write a full for loop for each one.

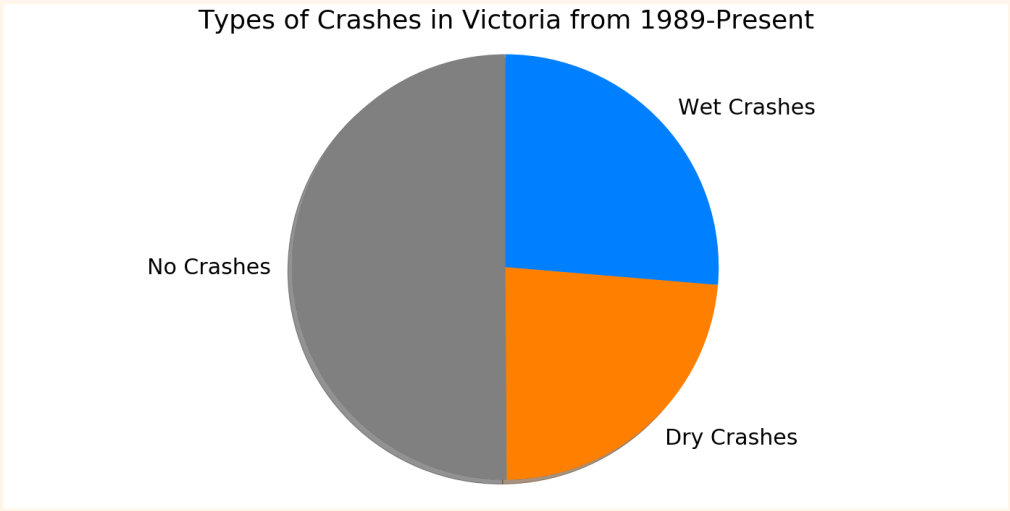
I at first graphed the number of fatalities vs rainfall, as opposed to the number of crashes vs rainfall. I then realised that that was not the graph that I wanted, although the graphs aren't very different side by side.

These graphs showed some correlation, but I decided to do some further calculations to figure out if there really was a connection.

I created a list of ints from the datasets. Each day had a corresponding list item according to this key:

> If there was no crash: 0
> If there was a crash and it wasn't raining: 1
> If there was a crash and it was raining: 2

I then graphed these and labelled accordingly to get the pie chart that I used as my final result:



Issues

I had *many* problems with this part; thinking about how to manipulate the data was the most challenging part of the whole project.

Invalid Graphs/Data

> My first graphs were not very successful. I originally graphed fatalities on the y axis vs the amount of rainfall on the x axis. There was no correlation and I soon realised that I was graphing the wrong properties.

I was also originally graphing the number of fatalities, instead of the number of crashes. This wouldn't have made much of a difference, but still an error that might have affected the validity when used on other datasets.
A couple of times I forgot to limit my queries to Victorian crashes only, and this turned out some graphs that (although interesting) were not valid.
Forgetting to sort my data before graphing.

Library Problems

It took a bit of time to understand how psycopg2 works (and also how to pronounce 'psycopg2')
Matplotlib and PyPlot are *huge*, and learning how each part integrates with each other part, and the other parts of the SciPy stack, was challenging.

Notebook

Here is the Jupyter Notebook that contains code for querying and visualising the data.

Want to see the code? This project can be found on **GITHUB**