ETL Report: Greater Melbourne Suburbs & Crime Data

Victorian Crime Data vs. Greater Melbourne Suburbs and Their Establishments

Authors: Jason Sutton, Tom Peddlesden, Welan Chu & Heesu Ha 8th March, 2021

Project Overview

This project seeks to enable exploratory data analysis of the relationship (if any) between crimes committed and the profile of the suburb based on types of establishments, within the Greater Melbourne area. The data captures types of crimes and the average frequency per year and place types including liquor stores, police stations and parks. (The average is based on the frequency recorded for years ending from 2011 to 2020.)

The Extract-Transform-Load (ETL) process was utilised for its effectiveness in collating the raw datasets for analysis (extraction) from various sources, and applying numerous transformations on each dataset in order to be loaded into SQL as a relational database. The transformations "cleaned" the data into a normalised and consistent format to allow for importing into SQL and subsequently allow any user to perform various join analyses applicable to one's needs.

Documented and described data sources

The following data sources were used in this project:

1. **Melbourne suburbs data** was collected from the website domain.com.au on 25th February 2021, as HTML code. Web-scraping using Python code library import BeautifulSoup collected the suburb names in Melbourne and were subsequently exported into a CSV which defined the scope of the rest of the data.
   https://www.domain.com.au/news/melbournes-321-suburbs-ranked-for-liveability-20151106-gkq447/#:~:text=Melbourne's%20321%20suburbs%20ranked%20for%20liveability

2. **Suburb latitude and longitude** coordinates data for each suburb was downloaded as CSV on the 25th February 2021.
   https://www.corra.com.au/australian-postcode-location-data/

3. **Local Government Area (LGA) and postcode** data from ABS archives as a CSV that linked LGA with suburbs on 25th February 2021.

https://www.abs.gov.au/AUSSTATS/abs@.nsf/Lookup/2905.0.55.001Main+Features1Aug%202006?OpenDocument

4. **Google Places data** obtained from Google Places API calls, retrieved as JSON responses for counts on search terms per suburbs coordinates on the 26th February 2021.
   https://developers.google.com/maps/documentation/places/web-service/search

5. **Victorian Government crime data** available from discover.data.vic.gov.au as an excel spreadsheet. The data describes the types of crimes and incident counts for each suburb and LGA for 2010 to September 2020. This was collected on 25th February 2021.
   https://discover.data.vic.gov.au/dataset/crime-by-location-data-table

Data Retrieval Process

Data was retrieved from the above data sources using the following methods:

Suburb Data	
Data Source	**Extraction Method**
1. Domain.com.au Greater Melbourne Area (GMA) Suburbs **Data type:** HTML **Source:** https://www.domain.com.au/news/melbournes-321-suburbs-ranked-for-liveability-20151106-gkq447/#:~:text=Melbourne's%20321%20suburbs%20ranked%20for%20liveability	Python imports: Requests, BeautifulSoup 1. Requests.get method used to request webpage, providing the web address response (URL); 2. BeautifulSoup used to convert the response to HTML code and find all suburbs through a find with tag div and class domain-article-content-wrap then a find_all for all h3 tags. This returns the names of suburbs in greater melbourne and placed arranged in a DataFrame. 3. This data was then saved as a CSV file.

2. Latitude and Longitude Coordinates Data for GMA Suburbs **Data type:** CSV **Source:** https://www.corra.com.au/australian-postcode-location-data/	Download CSV data direct from site ● From Python importing Pandas using pd.read_csv() method to import the data as a Pandas DataFrame.
3. LGA and Postcode data from Crime Data **Data type:** Excel Spreadsheet **Source:** https://discover.data.vic.gov.au/dataset/crime-by-location-data-table	Download Excel data direct from site ● From Python importing Pandas using pd.read_excel() method to import the data as a Pandas DataFrame.

Google Places API Data	
Data Source	**Extraction Method**
4. Google Places: Place Types **Data type:** API **Source:** https://developers.google.com/maps/documentation/places/web-service/search	Python imports: Requests, Json 1. Requests method used to request Google Places API address (URL) with parameters of suburbs; 2. Google Places API were called for each Melbourne suburb's latitude and longitude, as well as a specified radius (1,000 m) and establishment type. The JSON response list of results length (counts) for each type was appended to the DataFrame. ● The radius distance chosen in the API call was decided to be 1000m as Victoria's grid *"layout of the inner suburbs is largely one-mile grid pattern"* This is less true for suburbs further away from Melbourne CBD which are generally larger. With more time, a data source with each suburb area may be taken into account and

	subsequently a radius search term may be appropriately made for each suburb. ○ *https://en.wikipedia.org/wiki/Melbourne*

Crime Data	
Data Source	**Extraction Method**
5. Victorian Government Department of Justice Crime Data from October 2010 - September 2020 **Data type:** Excel Spreadsheet **Source:** https://discover.data.vic.gov.au/dataset/crime-by-location-data-table	Download Excel data direct from site ● Python importing Pandas using pd.read_excel() method to import the data as a Pandas DataFrame by parsing the relevant sheet number ("Table 3")

Cleaning, Checks and Exploratory Data Analysis (EDA) process

Following the extraction process of retrieving all data and inputting into Jupyter Notebook, a series of transformations were undertaken, including cleaning and checking the dataset to ensure the final output to CSV could be imported successfully into an SQL schema.

Some EDA was also undertaken where relevant as a way to check the datasets and provide cursory analysis for the end user as examples of how the data can be analysed.

Suburb Data

Suburb data - suburb names:

Cleaning:

Scraped information was uppercase. Str.lower() was used to make all letters lowercase.

Exploratory Data Analysis:

Not applicable. This data type does not require any exploration as it is essentially an index data type to inform the Google Places and Victorian Government Crime data.

Suburb data - coordinate data:

Cleaning:

Coordinate data was retrieved from corra.com.au “Australian Postcodes” CSV and was already in a usable format. Further cleaning involved dropping data from other states, leaving a dataframe of only Victorian suburbs and postcodes.

Checks:

Using the .info() method to confirm the data types of each column and whether there were any null values. Critical checks included: postcodes must be integers, coordinates must be float, all others may be string.

```
In [17]: #Check data info for null fields - Total records are 354
         suburbpostcodemerge.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 354 entries, 1 to 483
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   suburb    354 non-null    object
 1   postcode  354 non-null    int32
 2   state     354 non-null    object
 3   dc        354 non-null    object
 4   type      354 non-null    object
 5   lat       354 non-null    float64
 6   lon       354 non-null    float64
dtypes: float64(2), int32(1), object(4)
memory usage: 20.7+ KB
```

Exploratory Data Analysis:

Not applicable. This data type does not require any exploration as it is more of a key table to inform the Google Places and Victorian Government Crime data.

Suburb data - LGA and suburb table (linking table):

Cleaning:

Pandas used to read in suburb_vs_crime CSV and used str.lower() to make lga_name column values all lower case. The suburb_vs_crime table was then filtered to only show lga_name, suburb and postcode columns. The linking table was created by creating an lga_id table and merging it with the previously filtered table on LGA names. The merged table was then filtered to give back LGA ID numbers (primary key) and suburbs (primary key).

Exploratory Data Analysis:

Not applicable. This data type does not require any exploration as it is more of a key table to inform the Google Places and Victorian Government Crime data.

Google Places Data

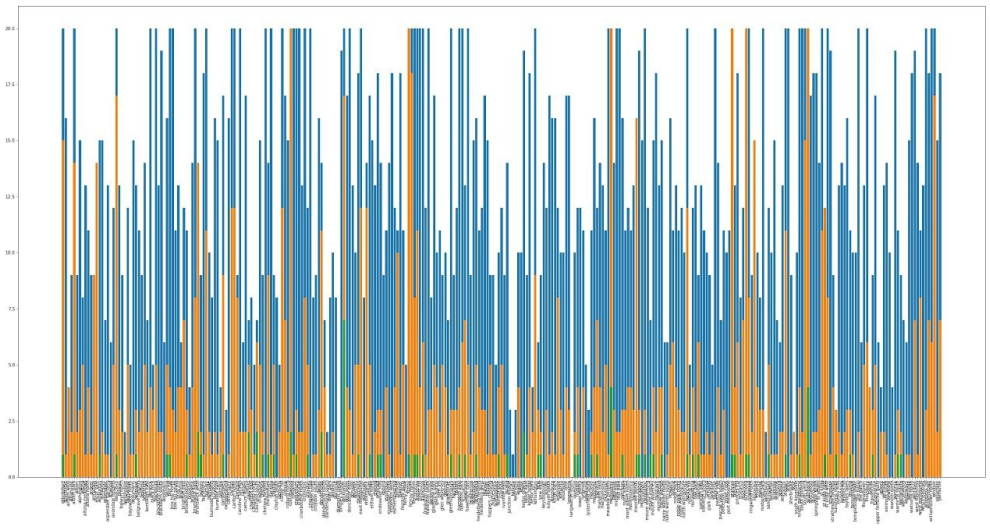
Cleaning:

Since the data was retrieved as an API call based on explicit search terms and aggregated using a for loop, there was no cleaning required of the data. The purpose of this data was to retrieve the count of specific establishment types (i.e. liquor store, police station and park) which is calculated within the for loop and thus does not require further transformations.

Checks:

A stacked bar plot created with the Python library matplotlib was used to check that the places counts produced a chart that made sense and as a way to confirm the presence of any anomalies or erroneous data. The stacked bar chart plotted over 300 suburb data points and appears to display correctly.

Exploratory Data Analysis:



A limitation of the Google Places API call limited the maximum number of results per type to 20, as evidenced in the peaks in the stacked bar chart. Further exploration of the API reveals a method for retrieving more results using 'pagetoken' to increase the results to a maximum of 60, however, due to time and credit constraints this was not found.

- Google Places API *"returns up to 20 results from a previously run search. Setting a pagetoken parameter will execute a search with the same parameters"* https://developers.google.com/maps/documentation/places/web-service/search

	suburb	postcode	state	dc	type	lat	lon	No liquor_store	No police	No park
29	bellfield	3381	VIC	STAWELL	Delivery Area	-37.228577	142.480053	0.0	0.0	0.0
99	diamond creek	3089	VIC	DIAMOND CREEK DC	Delivery Area	-37.642629	145.217595	0.0	0.0	0.0
153	hillside	3875	VIC	BAIRNSDALE	Delivery Area	-37.830936	147.470591	0.0	0.0	0.0
183	langwarrin south	3911	VIC	BAXTER DELIVERY ANNEXE	Delivery Area	-38.198507	145.182526	0.0	0.0	0.0

There were four suburbs found that have no search term results found in the Google Places API. This may change if each suburb area was taken into account to expand the radius of the API call. Due to the very small suburbs with no results, it was decided no further action was required.

```
 #   Column          Non-Null Count
---  ------          --------------
 0   suburb          316 non-null
 1   postcode        316 non-null
 2   state           316 non-null
 3   dc              316 non-null
 4   type            316 non-null
 5   lat             316 non-null
 6   lon             316 non-null
 7   No liquor_store 316 non-null
 8   No police       316 non-null
 9   No park         316 non-null
```

The data types for each column were also checked for any ‘NaN’ and/or erroneous values that would affect further extractions and transformations. None were found and it was deemed acceptable to proceed.

Victorian Crimes Data

Cleaning:

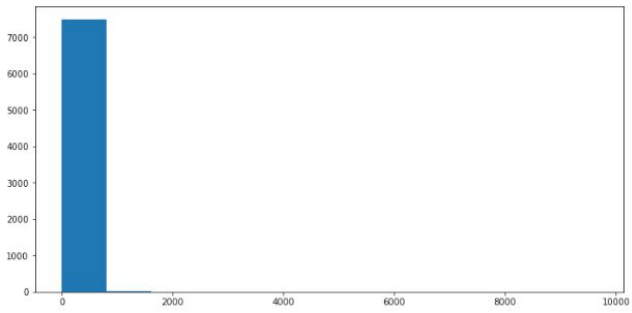
The excel data of Victorian crimes included information on the year of criminal incidents, location, number of incidents and three columns to specify types of crime committed. The types of crimes were broken down into three parts of increasing specificity: offence division, offence subdivision, and offence subgroup. The offence subdivision column was kept while the other two were dropped from the dataframe**.** The string data in the offence subdivision column was then split to isolate the offence type code and added to the dataframe to be later used as a primary key for offence types.
Values in the postcode column were converted to integer values and column headers and string values were canonicalised by converting strings from uppercase to lowercase lettering.

Checks:

Each row was checked for unique values to find and correct any spelling variations or mistakes as well as drop nonsensical values which would result in the removal of specific rows. Dataframe.info was used to check column values and to identify any NaN values.

Exploratory Data Analysis:

Early summary tables and visualisations indicated extreme outliers.



The outliers were determined and dropped from the dataframe to smooth the data.

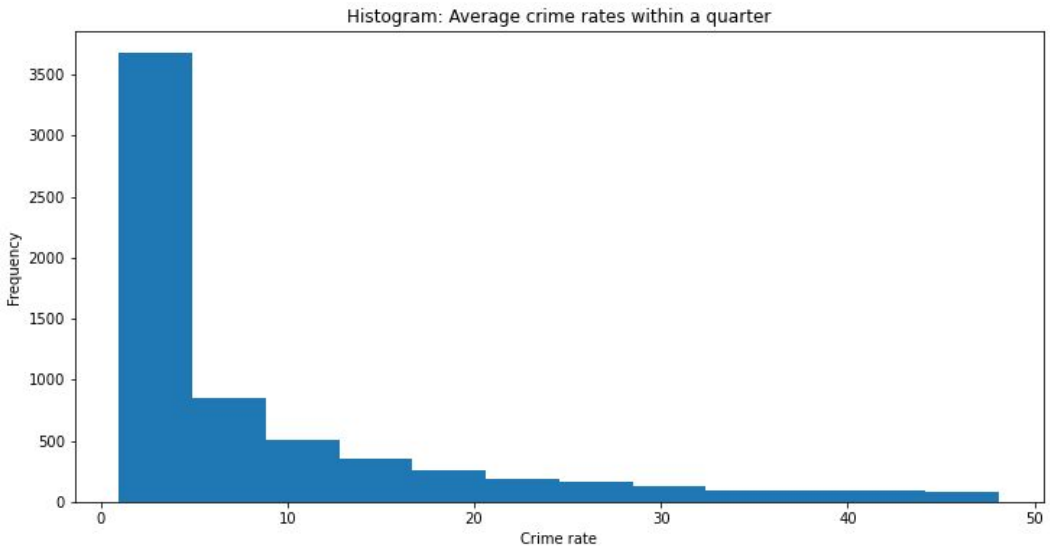
Summary statistics

Summary statistics were undertaken to determine if any results for the top ten or the button ten stood out as an issue.

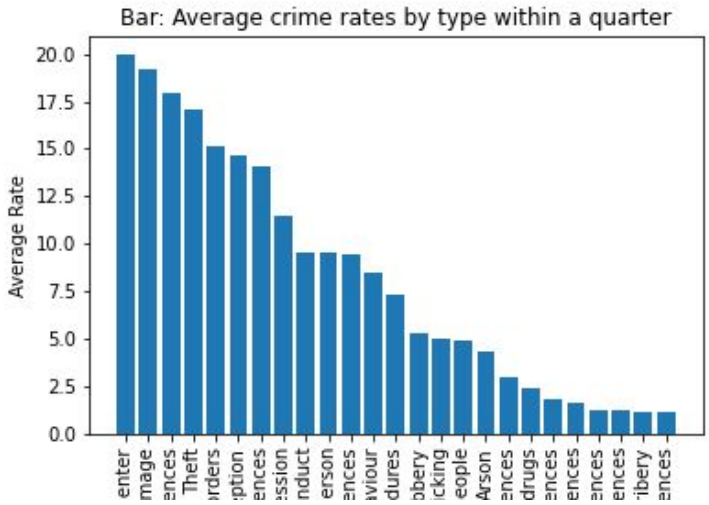
suburb	Mean crime rates per quarter	median	Variance crime rates per quarter	Standard Deviation crime rates per quarter	SEM crime rates per quarter
reservoir	22.441204	25.300000	291.070575	17.060791	4.925026
preston	21.787075	20.800000	316.699573	17.796055	4.756196
cranbourne	20.197778	19.400000	284.358169	16.862923	4.353988
hoppers crossing	20.088235	19.600000	255.889436	15.996545	3.879732
craigieburn	19.342222	15.000000	249.879122	15.807565	4.081496
broadmeadows	18.680741	14.100000	300.702548	17.340777	4.477369
berwick	17.957143	15.966667	173.736996	13.180933	3.522753
footscray	17.612755	10.850000	252.735355	15.897652	4.248826
springvale	17.418132	16.900000	148.646774	12.192078	3.381474
narre warren	17.209804	15.000000	264.197467	16.254152	3.942211

suburb	Mean crime rates per quarter	median	Variance crime rates per quarter	Standard Deviation crime rates per quarter	SEM crime rates per quarter
sherbrooke	1.505159	1.083333	0.579165	0.761029	0.219690
clematis	1.591667	1.600000	0.343125	0.585769	0.162463
the patch	1.929340	1.666667	0.896771	0.946980	0.236745
sassafras	2.043959	1.464286	2.936829	1.713718	0.403927
selby	2.134649	1.250000	3.143644	1.773032	0.406762
north warrandyte	2.182639	1.416667	4.473152	2.114983	0.528746
junction village	2.284226	1.775000	4.056224	2.014007	0.503502
belgrave heights	2.393170	1.333333	4.865610	2.205813	0.506048
strathmore heights	2.396508	1.333333	9.404859	3.066734	0.685743
langwarrin south	2.463095	1.708333	4.832550	2.198306	0.518146

Histogram of total crime rates



Bar chart of rate per type



Transformations Applied - Why these transformations add value

The following tables outline significant transformations undertaken for each dataset in order to arrive at its final, usable state as a relational database in SQL. Some data required more transformations than others, due to the nature of the raw dataset and the end purpose.

Suburb Data	
Transform applied	**Purpose/benefit**
Capitalised words from the scraped data changed from uppercase to lowercase using str.lower() method	Normalising data to avoid duplicates of data when merging with other tables since string data is case sensitive.
Filtered tables to only include columns that were needed by assigning the filtered DataFrame to another variable.	Allows for easier data handling and removes superfluous columns.
Isolate dataframe based on state column, find all values that do not equal “VIC” and drop them in place from the dataframe using .drop(index=True)	Cleaned data to be only Victorian data

Changed postcode data type from float to integer using .astype(int) on the postcode column.	This ensured the postcodes column was represented accurately (without decimal points) and could be further transformed as an integer type, instead of, say, a string type.
Using .drop() method, dropped all locations that do not have the type “Delivery Area” from the dataframe to get the final, unique Greater Melbourne Area (GMA) Suburbs data with the Latitude and Longitude Coordinates Data for GMA Suburbs	Created usable table based on determined list of suburbs to base API calling on and to filter crime rate data
Created usable table based on determined list of suburbs to base API calling on and to filter crime rate data	Reduced superfluous columns from Pandas DataFrame
Directory 004_Extract_Teansform_LGA_Suburb	
List comprehension to generate local government area IDs for column “lga_id.”	Required to assign unique IDs for each local government area.
Created a unique list of local government areas using the .unique() method on the series “lga_name” and wrapping this within a list() method.	Required to assign the local government area to each unique ID.
Used pd.DataFrame() to turn the two lists into a new pandas dataframe for export as CSV.	CSV generated to export into PostgreSQL schema.
.drop_duplicate(subset=”suburb”) method used on previously created dataframe in order to ensure only unique suburbs remained in the dataframe to be exported.	CSV generated to export into PostgreSQL schema.

Google Places Data	
Transform applied	**Purpose/benefit**
Importing suburb coordinate data for centre of each suburb to merge with suburb names used for Google Places API calls	Limit the API call to relevant suburbs for the scope of the project.
Added columns for each search term (No. Police, No. Liquor_Store, No. Park) to DataFrame from melbourne_suburbs_postcode.cs	Append count of establishment results from Google Places API to each suburb in the relevant column. This dataframe was then exported as “melb_suburbs_types.csv” for further transformations.

Using the dataframe from “melb_suburbs_types.csv” from the previous notebook and isolating the establishment types column headers (e.g. liquor_store, police, park)	Required to extract the establishment types and save it as its own DataFrame that would be imported directly into SQL
Loop through column headers to strip “No.” from establishment types and save into a list. Extraneous columns (e.g. suburb, postcode, state, etc.) were excluded from the list.	Required to extract the place types and exclude the unnecessary column headers.
Reset index and rename the new column as “place_id” to generate unique ID numbers for each place type.	Required for the SQL table as a unique ID column that allows for linking to other tables.

Victorian Crimes Data	
Transform applied	**Purpose/benefit**
Filter out data for greater Victoria to include only metropolitan Melbourne suburbs using the merged final Greater Melbourne Area (GMA) Suburbs data with the Latitude and Longitude Coordinates Data for GMA Suburbs	Define definite lists of suburbs recorded to align data with SQL table schema
Assigned reduced dataframe to new variable using the double-square brackets method with required column headers to create new dataframe.	Reducing the existing dataframe to only relevant columns for further analysis and eventually for import into SQL.
Aggregate crimes into **Offence Subdivision** column using .groupby() method	Extracted data included three levels of crime categories. Lower level was cumbersome due to the large number of categories. As the second level was still descriptive, combining the lower level frequency to the second layer made more readable and manageable data.
Aggregate crimes by year and local government area using .groupby() method and .sum() to calculate total incidents recorded per year, local government area, postcode and suburb for each offence subdivision.	Aggregating by year allows for the potential to analyse differences in crime statistics over a period of time, such as determining trends and whether crimes on aggregate and/or by offence subdivision is consistent, increasing or decreasing over time.
Using .groupby() method and .mean() on	Removes the time aspect of the data to

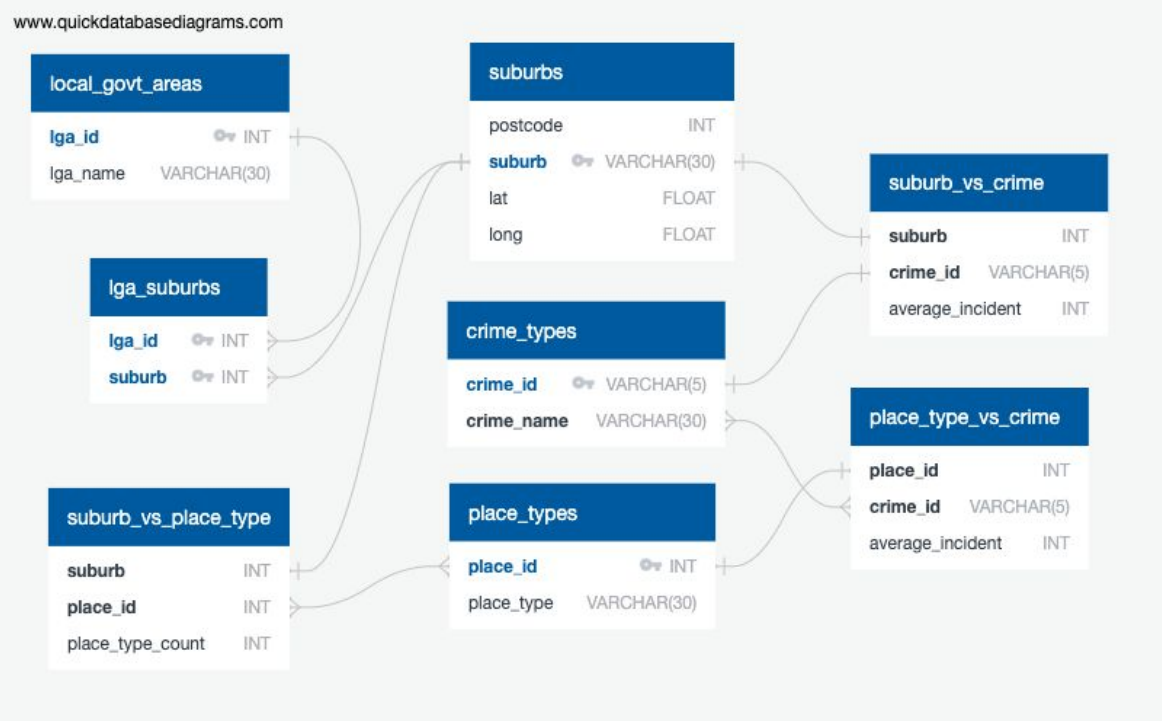
dataframe to get average incidents reported for each offence subdivision per suburb.	assess each suburb on an averaged, single value per offence subdivision, allowing for simpler comparison between suburbs/local government areas rather than comparisons per year, which may fluctuate.
Calculate average yearly crimes for the year 2011-2020	Provide succinct data for different crime types per suburb
Splitting the offence ID from the offence name to create a new dataframe with two columns: the crime_id and crime_name.	Allows for a separate table containing crime_id as a primary key column and corresponding to crime_name, and linking other tables by crime_id.

Loading Datasets into PostgreSQL from Jupyter Notebook

- As mentioned above, PostgreSQL relational database management system was used. PostgreSQL was the most meaningful way of illustrating the relationships between the data we obtained.
- The tables have been transformed to allow additional tables of relevant data to be included for further insight into the project. This may include but is not limited to area size, average income and election results by suburb or LGA.
- The final PostgreSQL table schema was defined in pgAdmin using the sql query file based on the below Entity Relationship Diagram.
- The final Jupyter Notebook pushed the datasets into the PostgreSQL database in pgAdmin using the python library SQL Alchemy. An engine is created and connects to PostgreSQL and enables Jupyter Notebook to write directly to the database tables.
- In order to do this in an efficient manner, all pandas dataframes were set to have the same column headers as in the SQL tables, and the indexes from the pandas dataframes dropped during the process of pushing the data to SQL using the .to_sql() method in pandas.

The SQL schema consists of eight tables: *suburbs, crime, place* and *local_govt_areas* that hold specific data for each topic. These datasets are then linked in the tables *suburb_vs_crime, place_vs_crime and suburb_vs_place,* allowing for immediate analysis through comparison and join queries to find relationships.

SQL Table Schema



Examples of Analysis

The following queries were tested in PostgreSQL as examples of potential analyses and lines of enquiry.

Query: total average incidents per local government area and crime type, ordered by local government area in ascending order

```
SELECT
        sc.lga_name,
        sc.crime_name,
        SUM(sc.average_incident) AS total_average_incidents
FROM suburb_vs_crime as sc
GROUP BY
        sc.lga_name,
        sc.crime_name
ORDER BY sc.lga_name ASC;
```

Query: total average incidents per local government area and crime type, ordered by total average incident in descending order

```
SELECT
        sc.lga_name,
        sc.crime_name,
        SUM(sc.average_incident) AS total_average_incidents
FROM suburb_vs_crime as sc
GROUP BY
        sc.lga_name,
        sc.crime_name
ORDER BY total_average_incidents DESC;
```

Query: average incidents per suburb and crime type vs. establishment types

```
SELECT
        l.lga_name,
        s.suburb,
        c.crime_name,
        c.average_incident,
        p.place_type,
        sp.place_type_count
FROM local_govt_areas AS l
INNER JOIN lga_suburbs AS s ON s.lga_id = l.lga_id
INNER JOIN suburb_vs_crime AS c ON c.suburb = s.suburb
INNER JOIN suburb_vs_place AS sp ON sp.suburb = s.suburb
INNER JOIN place AS p ON p.place_id = sp.place_id
ORDER BY c.average_incident DESC;
```

Query: total incidents per suburb vs. number and types of establishment

```
SELECT
        sc.suburb,
        p.place_type,
        sp.place_type_count,
        SUM(sc.average_incident) AS total_average_incident
FROM suburb_vs_crime AS sc
INNER JOIN suburb_vs_place AS sp ON sp.suburb = sc.suburb
INNER JOIN place AS p ON p.place_id = sp.place_id
GROUP BY
        sc.suburb,
        sp.place_type_count,
        p.place_type
ORDER BY total_average_incident DESC;
```