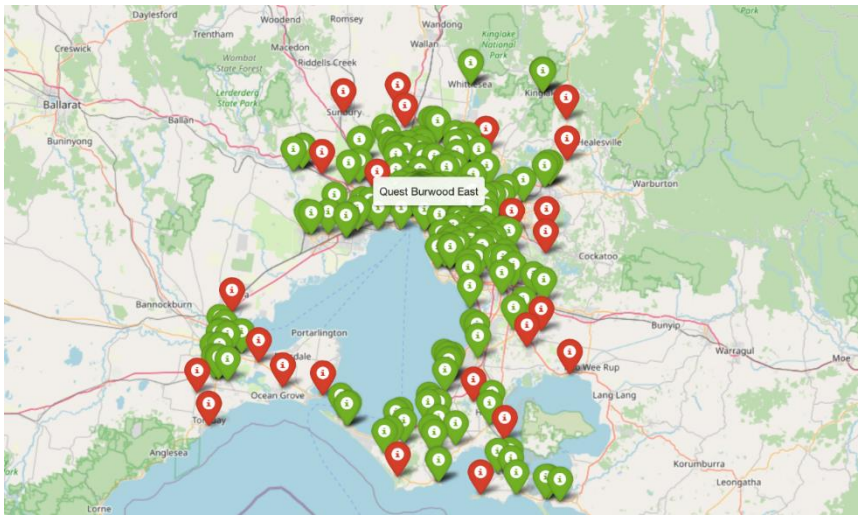
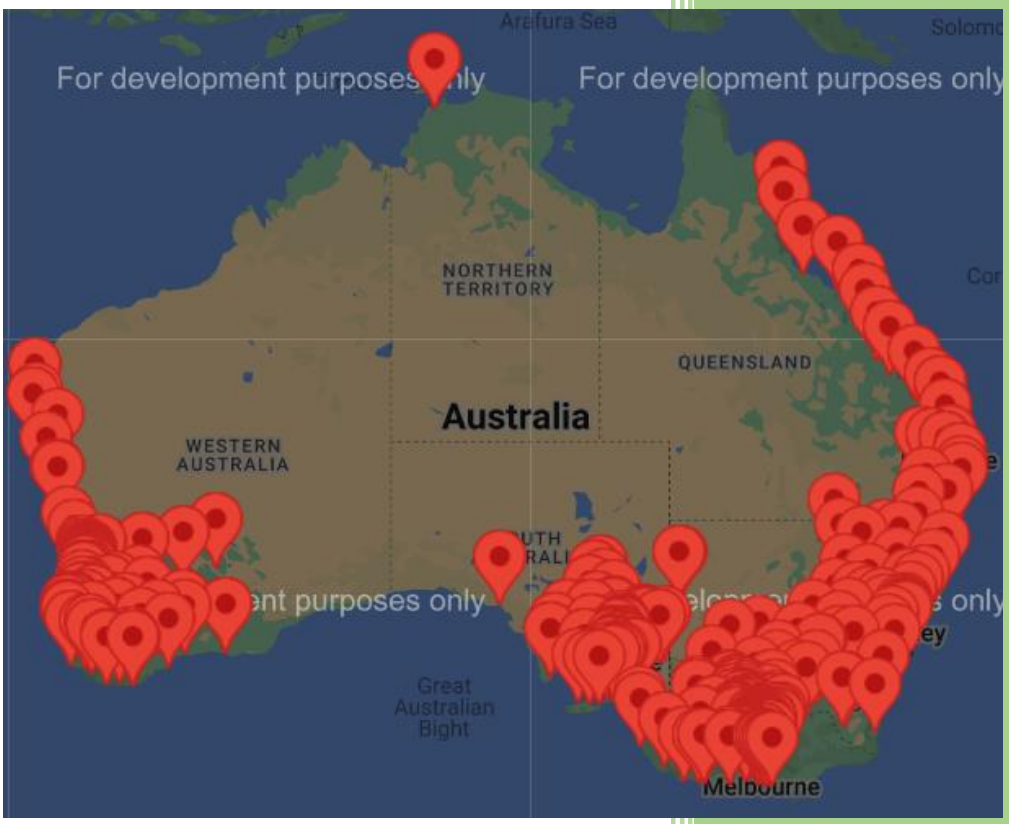


2022

EV Charger Forecasting and Location Optimisation (EVCFLO)

Data Analyst: Upskilling



MATTHEW ROBERT IREDALE (Project Lead)

Chameleon

9/25/2022

Contents

Disclaimer

This document serves for future data analysts working in EVCFLO, to hasten their upskilling process and familiarity with the project's workflow process.

Resources and processes provided in this document are subject to change in future trimesters, this document will remain 100% relevant for Tri-3 2022 only.

Any visual aids provided in this document may be unreadable, please use the zoom tool in the bottom right to enhance quality

Project EVCFLO

Our Vision

Chameleon provides services that promote the demand for environmentally friendly products to reduce the need for those which negatively impact the environment.

As part of the company Chameleon, project EVCFLO provides aid to this cause through two key services: **interactive maps using Google API,** and an **AI Prediction System,** that recommends new Electric Vehicle Charging Station (EVCS) locations which will be used effectively by the relevant population.

Our Objectives

Each trimester we aim to (at a minimum) achieve the following:

- Using community data to expand on our database through the addition of newly found datasets
- Narrow down to the key external factors which impact the usage and success of EVs and EVCSs
- Display newly discovered EVCS locations onto a visual map
- Integrate new EVCS locations onto our Google API interactive map (Web-Development team)

Our Goal

Through achieving these objectives, we will eventually have two things that represent the project being a success:

- An interactive map that covers all EVCS locations across the globe.
- An AI Prediction System that automatically and accurately recommends new EVCS locations for EVCS companies to utilize.

Our Process

Weekly update meetings to share progress on work, suggest ideas which will benefit the project and company, and resolve any conflicts.

Collecting community datasets which provide data on the usage and/or location of charging stations within a specified region.

Cleaning the datasets to ensure the data is free of irrelevant values, avoiding incorrect information and upholding accuracy.

Analysing cleaned datasets generating information that tells us anything noteworthy typically through detection of trends and/or outliers.

Generating Map Visualizations showing existing EVCS locations found through analysis and suggested (user-input OR algorithmic-computation) future EVCS locations that will prove effective.

Uploading sprint work is an essential step of the workflow process, this is a collaborative project after all where other members and future members will need to see your efforts, so that previous work can be referenced/expanded on.

At the end of each sprint, you are required to do the following:

- Any task that you are currently working on and yet to be finished are uploaded/updated onto the Microsoft Teams Channel filesystem (MTCF).
- Tasks that are marked as completed need to be uploaded to MTCF and the GitHub repository.

Sprints

Each trimester the project work is divided up into 3 sprints.

Sprint 1 (Weeks 2-4): Collection & Cleaning Phase

- Collect at least one dataset (ideally multiple)
- Import dataset into new Jupyter Notebook file (.ipynb)
- Clean the dataset removing any irrelevant data such as:
    - ‘0’ values
    - ‘Null’ cells
    - Incorrect records

Sprint 2 (Weeks 5-8): Analysis Phase

*NOTE intra-trimester break is between week 5-6 and it is not compulsory to conduct project work but is highly appreciated.

- Define any functions relevant to dataset
- Perform search algorithms to read-in relevant data
- Generate graphs using read-in data for visual aid
- Analyse graph noting any trends and/or outliers
- Conclude on your findings

Sprint 3 (Weeks 9-11): Mapping Phase

- Import map visualisation library of your choosing
- Setup environment constraints such as:
    - Plot boundaries
    - Clusters
    - Marker colours
- Display map for existing EVCS
- Define decision criteria for new EVCS recommendations:
    - Minimum distance from existing EVCS's
    - Location is vacant (can be bought and built on)
- Display map with new EVCS recommendations

Resources:

You should have access to all the following resources, try accessing each of them before starting any sprint work.

If you still need access, please indicate what resource you need access to and provide necessary information to the relevant company member such as:

In descending order of priority…

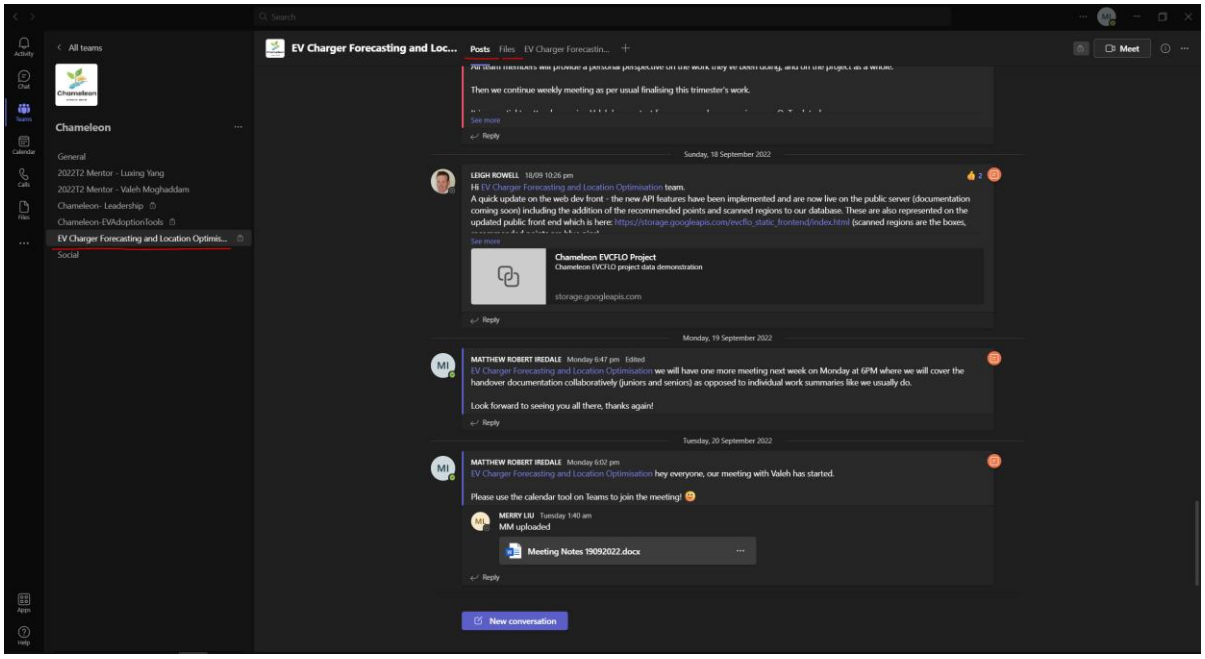
- The trimester's project leader/s
- The company director
- The unit-chair

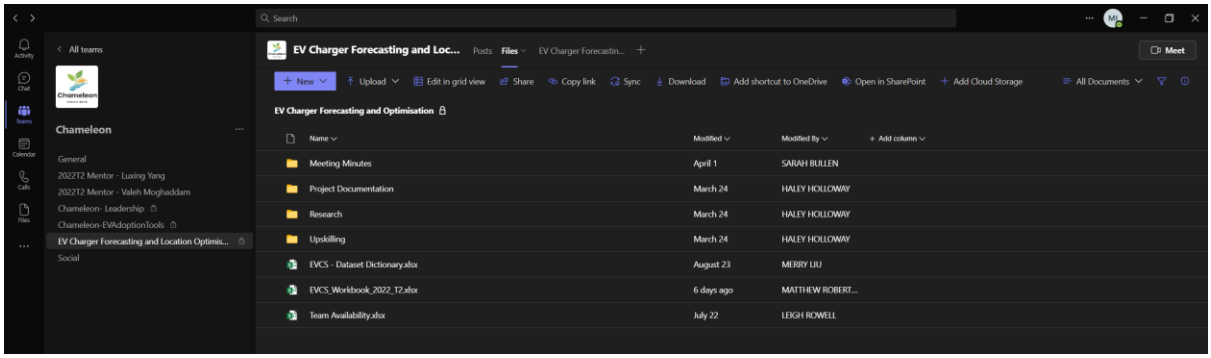
Microsoft Teams Channel & Filesystem (MTCF):

Temporary and permanent project workspace for sharing files.

Weekly project meetings and discussions are carried out in this channel, you can expect to spend the majority of your teamwork through this resource.

*See next page for visual references.





Meeting Minutes:

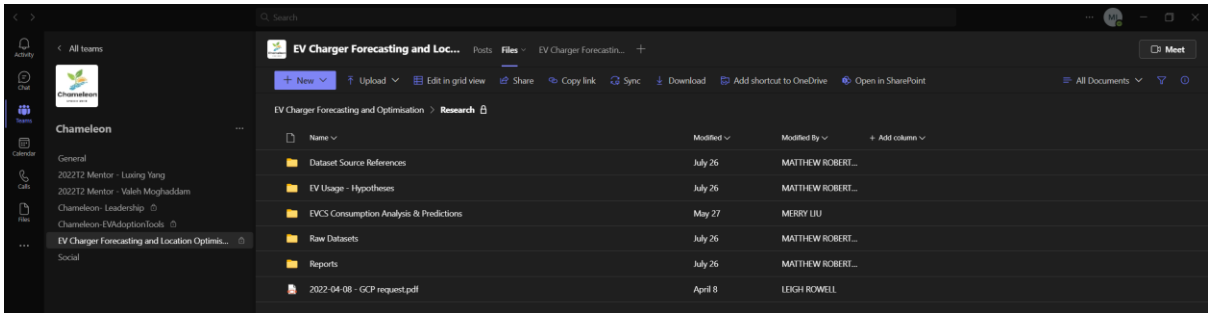
- Location for storing ‘minutes’ for each of the weekly meetings
- Contains previous/current/future meeting minutes
- Continuously refer to when unsure about sprint-work to be conducted
- If unable to attend meeting/s, useful for catching up on weekly update

Project Documentation:

- Stores documents for new project members to understand jargon
    - Such as a Glossary of Terms explaining key terms used (like EVCS)
- Records resource links used in the project

Research:

- <u>Your main workspace for uploading/updating files</u>



- **Dataset Source References:**
  - Research links for understanding EVs and EVCSs
  - Links with known and potential EV/EVCS datasets
  - Good starting point for collecting new datasets
- **EV Usage – Hypotheses:**
  - Types of questions to consider when conducting sprint-work
  - Proof/disproof of external factors that impact the usage of EVs
- **EVCS Consumption Analysis & Predictions:**
  - Jupyter Notebook files containing Analysis and Visualisations
- **Raw Datasets:**
  - Newly found datasets are uploaded here
  - Please place in correct folder based on data given for example…
    - If dataset only contains location information, then…
      - Place in ‘EVCS Locations’ folder
- **Reports:**
  - Any conclusions you make from analysis and create a report on goes here

Upskilling:

- Resources that assist in upskilling new/future members can be found here
- Such as the document you are viewing right now

EVCS – Dataset Dictionary.xlsx:

- VERY IMPORTANT RESOURCE
- You will use this when collecting and analysing datasets yourself
- Used for checking if a region has:
  - A location dataset
    - Analysed Y/N?
  - A usage dataset
    - Analysed Y/N?
  - Additional notes

EVCS_Workbook_[Year]_[Trimester].xlsx:

- Used for recording any unit work including:
  - Project work
  - Attending meetings
  - Attending/watching lectures
  - Completing OnTrack tasks

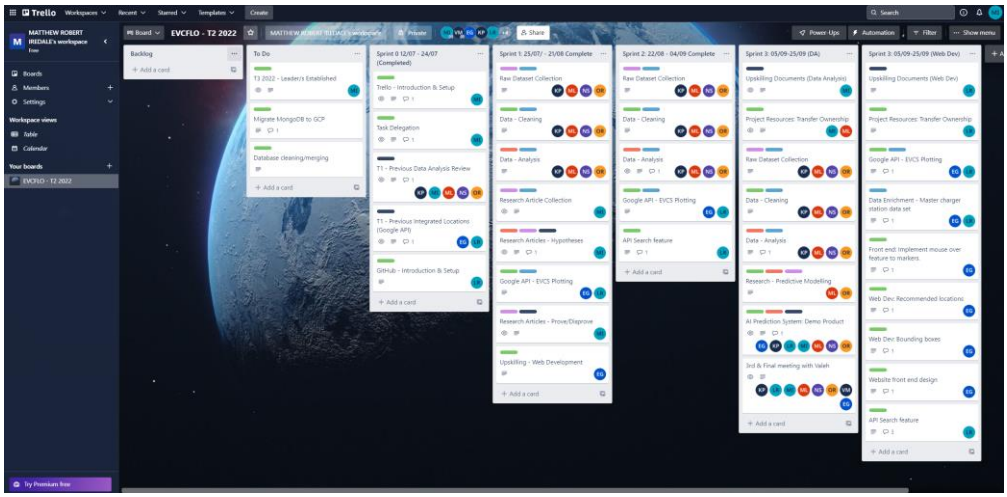
Team Availability.xlsx:

- Used to organise appropriate times to host weekly update meetings
  - First document to fill-in before entering sprint 1
  - Indicate your student credentials for reference
  - Add your weekly availability

GitHub Repository:

- Permanent project workspace for uploading finished work at sprint-end
- The filesystem has the same structure as the MTCF with the exception off files relevant to the web-development team only (which you can ignore)
- A short video guide to help you get started working with GitHub:
  - https://www.youtube.com/watch?v=iv8rSLsi1xo

Trello Board:

- Dashboard used to indicate all work to be done for each sprint
- Your best friend for keeping up-to-date on project work
- Project leader/s will be mainly editing the dashboard, but you have the freedom to add cards/comments of your own if necessary
- Visual reference for T2 – 2022's Trello Board
- 

- A short video guide to help you get started working with Trello:
- https://www.youtube.com/watch?v=geRKHFzTxNY

Website:

- Public display of our project (no web-design has been implemented yet)
- This feature is mainly concerned with the Web-Development Team
- However, any project work you have conducted and finished…
    - Can be displayed on the ‘View Research’ page if you make a request to the current web-dev team to update with your work
- 

Pre-requisite Software:

Anaconda Navigator is a free-to-use dashboard with all the IDEs you could possibly need for this project including Jupyter Notebook.

In order to clean/analyse/map any datasets you must have this software.

Before sprint 1 you should have this installed and ready to go.

Follow this guide for help with installation and a basic tutorial:

- https://www.youtube.com/watch?v=3C9E2yPBw7s

Collecting Data

It is imperative that before performing any cleaning/analysing/mapping that new data upholds to the minimum standard.

The two main dataset file formats we are looking for are:

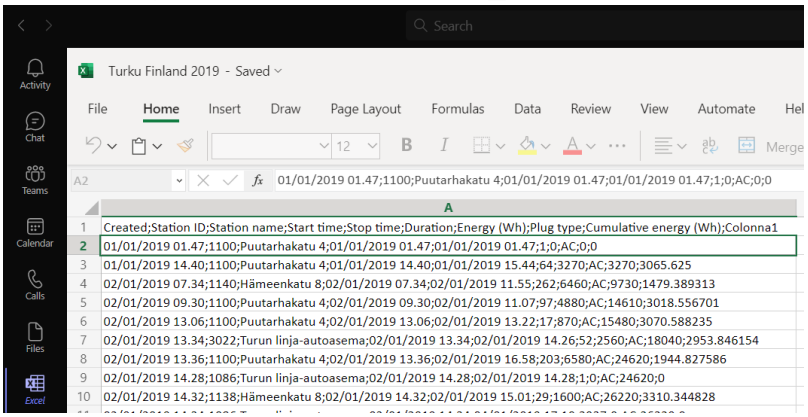
- **.csv** – Comma Separated Values
  - each record is in one cell, but each value is separated by a comma
- **.xlsx** – Traditional excel file format
  - values are separated using the columns/rows table provided by excel

However, community data comes in many file formats where we are required to convert datasets into .csv or .xlsx.

This is not covered in this document but many guides on the internet can be found to assist in converting file formats to others.

When considering a dataset, we can categorize it as either 'Usage', 'Location' or both (Location & Usage)

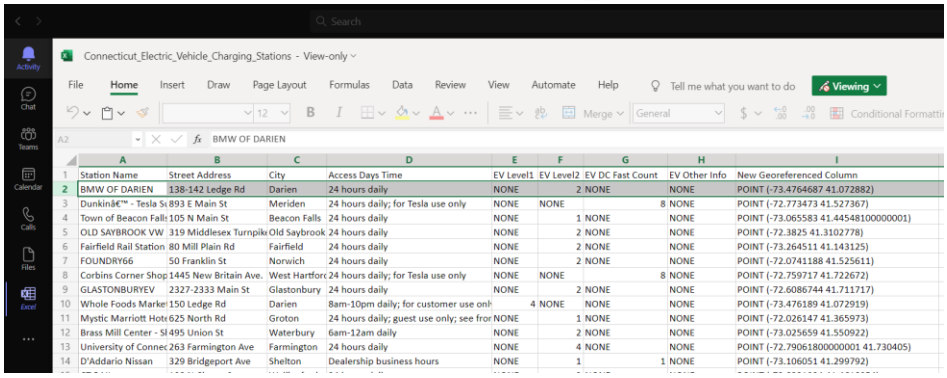
- **Usage:**
  - Contains usage data on EV users visiting EVCSs.
  - Sample record (green box) from Turku_Finland 2019.csv dataset:



    - Date Created = 01/01/2019 01.47
    - Station ID = 1100
    - Station Name = Puutarhakatu 4
    - Charging start time = 01/01/2019 01.47
    - Charging stop time = 01/01/2019 01.47
    - Duration spent charging = 1 (min)
    - Energy provided during charging = 0 (Wh)
    - Plug Type = AC
    - Cumulative Energy = 0 (Wh)

- At a minimum we require:
  - Station ID
  - Station Name
  - Start Time/Stop Time OR duration
  - Energy Provided
  - Plug Type
- *If this dataset didn't meet the minimum requirements, it would be rejected (not enough useful data)

- **Location:**
  - Contains location data on EVCSs
  - Sample record (green box) from Connecticut_EVCS.csv dataset:



Station Name	Street Address	City	Access Days Time	EV Level1	EV Level2	EV DC Fast Count	EV Other Info	New Georeferenced Column
BMW OF DARIEN	138-142 Ledge Rd	Darien	24 hours daily	NONE	2	NONE	NONE	POINT (-73.4764687 41.072882)
Dunkinâ€™ - Tesla S	893 E Main St	Meriden	24 hours daily; for Tesla use only	NONE	NONE	8	NONE	POINT (-72.773473 41.527367)
Town of Beacon Falls	105 N Main St	Beacon Falls	24 hours daily	NONE	1	NONE	NONE	POINT (-73.065583 41.44548100000001)
OLD SAYBROOK VW	319 Middlesex Turnpik	Old Saybrook	24 hours daily	NONE	2	NONE	NONE	POINT (-72.3825 41.3102778)
Fairfield Rail Station	80 Mill Plain Rd	Fairfield	24 hours daily	NONE	2	NONE	NONE	POINT (-73.264511 41.143125)
FOUNDRY66	50 Franklin St	Norwich	24 hours daily	NONE	2	NONE	NONE	POINT (-72.0741188 41.525611)
Corbins Corner Shop	1445 New Britain Ave.	West Hartfor	24 hours daily; for Tesla use only	NONE	NONE	8	NONE	POINT (-72.759717 41.722672)
GLASTONBURYEV	2327-2333 Main St	Glastonbury	24 hours daily	NONE	2	NONE	NONE	POINT (-72.6086744 41.711717)
Whole Foods Marke	150 Ledge Rd	Darien	8am-10pm daily; for customer use onl	4	NONE	NONE	NONE	POINT (-73.476189 41.072919)
Mystic Marriott Hot	625 North Rd	Groton	24 hours daily; guest use only; see fro	NONE	1	NONE	NONE	POINT (-72.026147 41.365973)
Brass Mill Center - Sl	495 Union St	Waterbury	6am-12am daily	NONE	2	NONE	NONE	POINT (-73.025659 41.550922)
University of Connec	263 Farmington Ave	Farmington	24 hours daily	NONE	4	NONE	NONE	POINT (-72.79061800000001 41.730405)
D'Addario Nissan	329 Bridgeport Ave	Shelton	Dealership business hours	NONE	1	1	NONE	POINT (-73.106051 41.299792)

    - Station Name = BMW OF DARIEN
    - Street Address = 138-142 Ledge Rd
    - City = Darien
    - Access Days Time = 24 hours daily
    - EV Level1 = NONE
    - EV Level2 = 2
    - EV DC Fast Count = NONE
    - EV Other Info = NONE
    - New Georeferenced Column = POINT (-73.476 41.073)
  - At a minimum we require:
    - Station Name and Street Address
      - OR
    - Georeferenced OR latitude and longitude
    - Plug types (Level 1, Level 2, DC, DC Fast, etc.)
  - *If we have the name and address of the charging station, we can manually find the latitude and longitude coordinates and add them into a new column/s like the example above

- **Both:**
  - Contains minimum data requirements of location and usage

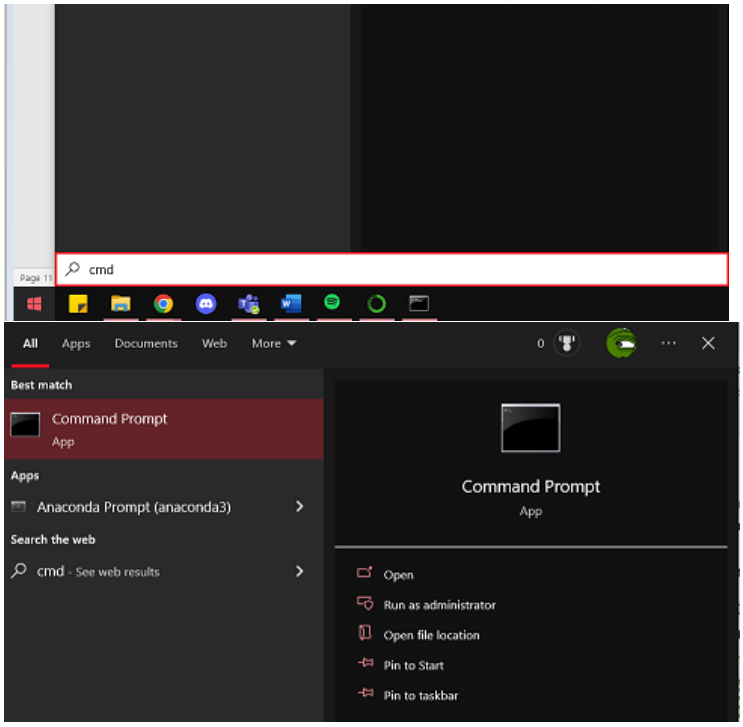
Cleaning Data

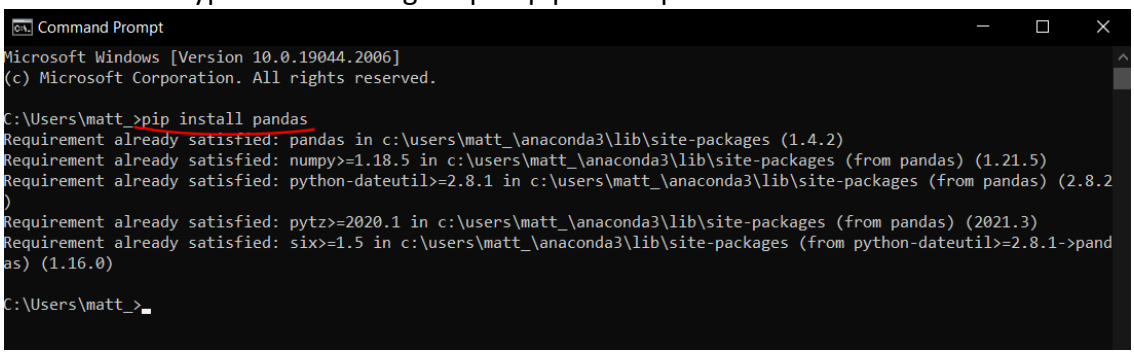
Import Libraries:

- First, we must import libraries before reading data into our notebook

Import Libraries

```
In [399]: import pandas as pd
          import numpy as np
          import matplotlib.pyplot as plt
          import matplotlib.ticker as ticker
          import datetime
          from datetime import datetime as dt
          import warnings
          import seaborn as sns
          import os
          warnings.filterwarnings("ignore")
```

-
- Pandas is used to read in a .csv file, if you have never used pandas before you will have to install it manually using Command Prompt or PowerShell
- 

- Run cmd and type the following script – pip install pandas
- 

- This can be done for any library so – pip install [library name]
- Now pandas is installed, you can read in any .csv dataset as seen below:

Read in the dataset/s:

Read in the dataset "Turku Finland"

```
In [369]: df = pd.read_csv("Turku Finland 2019.csv", encoding="ISO-8859-1", sep = ";")
          df.head()
```

Check and remove null values:

Check for null values

```
In [370]: df.isnull().any()
Out[370]: Created                 False
          Station ID              False
          Station name            False
          Start time              False
          Stop time               False
          Duration                False
          Energy (Wh)             False
          Plug type               False
          Cumulative energy (Wh)  False
          Colonna1                False
          dtype: bool
```

- None detected (False), no need to remove null values

Check and remove 0 values:

Remove all 0 values from energy and duration column

```
In [373]: #Values of 0 for 'Energy (Wh)' and 'Duration' is considered inaccurate data and must be removed...
          #An EV is not charging if it recieved 0 energy from the charging station
          #An EV was never at the charging station if it spent 0 time there
          df.drop(df[df['Energy (Wh)'] == 0].index, inplace = True)
          df.drop(df[df['Duration'] == 0].index, inplace = True)
          df.head()
```

Out[373]:

	Created	Station ID	Station name	Start time	Stop time	Duration	Energy (Wh)	Plug type	Cumulative energy (Wh)	Colonna1
1	01/01/2019 14.40	1100	Puutarhakatu 4	01/01/2019 14.40	01/01/2019 15.44	1.066667	3270	AC	3270	3065.625
2	02/01/2019 07.34	1140	Hämeenkatu 8	02/01/2019 07.34	02/01/2019 11.55	4.366667	6460	AC	9730	1479.389313
3	02/01/2019 09.30	1100	Puutarhakatu 4	02/01/2019 09.30	02/01/2019 11.07	1.616667	4880	AC	14610	3018.556701
4	02/01/2019 13.06	1100	Puutarhakatu 4	02/01/2019 13.06	02/01/2019 13.22	0.283333	870	AC	15480	3070.588235
5	02/01/2019 13.34	3022	Turun linja-autoasema	02/01/2019 13.34	02/01/2019 14.26	0.866667	2560	AC	18040	2953.846154

Check data types and size:

Check data types and size

```
In [371]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7737 entries, 0 to 7736
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Created                 7737 non-null   object
 1   Station ID              7737 non-null   int64
 2   Station name            7737 non-null   object
 3   Start time              7737 non-null   object
 4   Stop time               7737 non-null   object
 5   Duration                7737 non-null   int64
 6   Energy (Wh)             7737 non-null   int64
 7   Plug type               7737 non-null   object
 8   Cumulative energy (Wh)  7737 non-null   int64
 9   Colonna1                7737 non-null   object
dtypes: int64(4), object(6)
memory usage: 604.6+ KB
```

- Its good practice to perform this check, understanding what data types you're working with is useful when performing algorithmic analysis

Check for any format errors:

Change Duration time from minutes to hourly

In [372]:
```
#Energy was calculated in watts per hour (Wh), changing the duration to hours is appropriate for analysis
df['Duration'] = df['Duration'] / 60
df.head()
```

Out[372]:

	Created	Station ID	Station name	Start time	Stop time	Duration	Energy (Wh)	Plug type	Cumulative energy (Wh)	Colonna1
0	01/01/2019 01.47	1100	Puutarhakatu 4	01/01/2019 01.47	01/01/2019 01.47	0.016667	0	AC	0	0
1	01/01/2019 14.40	1100	Puutarhakatu 4	01/01/2019 14.40	01/01/2019 15.44	1.066667	3270	AC	3270	3065.625
2	02/01/2019 07.34	1140	Hämeenkatu 8	02/01/2019 07.34	02/01/2019 11.55	4.366667	6460	AC	9730	1479.389313
3	02/01/2019 09.30	1100	Puutarhakatu 4	02/01/2019 09.30	02/01/2019 11.07	1.616667	4880	AC	14610	3018.556701
4	02/01/2019 13.06	1100	Puutarhakatu 4	02/01/2019 13.06	02/01/2019 13.22	0.283333	870	AC	15480	3070.588235

Remove irrelevant data:

Remove all durations greater than 24 hours

In [374]:
```
#EVs staying at charging stations for longer than 24 hours generated outliers...
#This was ineffective for finding trends, thus any durations over were removed
df.drop(df[df['Duration'] > 24].index, inplace = True)
df.head()
```

Out[374]:

	Created	Station ID	Station name	Start time	Stop time	Duration	Energy (Wh)	Plug type	Cumulative energy (Wh)	Colonna1
1	01/01/2019 14.40	1100	Puutarhakatu 4	01/01/2019 14.40	01/01/2019 15.44	1.066667	3270	AC	3270	3065.625
2	02/01/2019 07.34	1140	Hämeenkatu 8	02/01/2019 07.34	02/01/2019 11.55	4.366667	6460	AC	9730	1479.389313
3	02/01/2019 09.30	1100	Puutarhakatu 4	02/01/2019 09.30	02/01/2019 11.07	1.616667	4880	AC	14610	3018.556701
4	02/01/2019 13.06	1100	Puutarhakatu 4	02/01/2019 13.06	02/01/2019 13.22	0.283333	870	AC	15480	3070.588235
5	02/01/2019 13.34	3022	Turun linja-autoasema	02/01/2019 13.34	02/01/2019 14.26	0.866667	2560	AC	18040	2953.846154

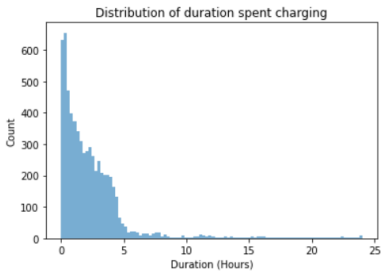
Analysing Data

Generating Visualisations (Graphs):

- Distribution:

Distribution of duration spent charging

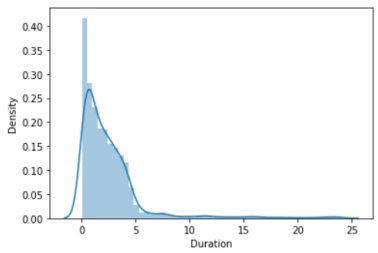
In [375]:
```
plt.hist(df["Duration"], bins=100, alpha=0.6)
plt.title("Distribution of duration spent charging")
plt.xlabel("Duration (Hours)")
plt.ylabel("Count")
plt.show()
```



  - As seen most people in Turku Finland charge their EVs for 1-2 hours
- Distribution with trendline:

Same distribution with trendline

```
In [376]: sns.distplot(df["Duration"])
Out[376]: <AxesSubplot:xlabel='Duration', ylabel='Density'>
```



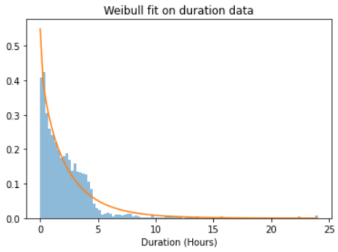

```
In [377]: print(df["Duration"].mean())
          print(df["Duration"].median())
          print(df["Duration"].max())
          print(df["Duration"].min())

2.4768065268065267
1.7333333333333334
24.0
0.016666666666666666
```

- Using the median, it can be seen most people spend roughly 1.45 hours charging their EVs

- Weibull Distribution:

Weibull Distribution of duration spent charging

```
In [378]: from scipy.stats import weibull_min
          plt.hist(df["Duration"], bins=100, density=True, alpha=0.5)
          shape, loc, scale = weibull_min.fit(df["Duration"], floc=0)
          x = np.linspace(df["Duration"].min(), df["Duration"].max(), 100)
          plt.plot(x, weibull_min(shape, loc, scale).pdf(x))
          plt.title("Weibull fit on duration data")
          plt.xlabel("Duration (Hours)")
          plt.show()
```

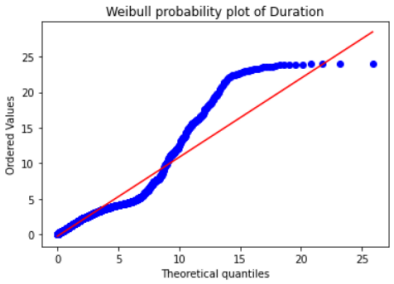


- Similar story, cross-check satisfied (validation)

- Weibull Probability:

Weibull Probability of duration spent charging

```
In [380]: from scipy.stats import probplot, weibull_min
          probplot(df["Duration"], dist=weibull_min(shape,loc,scale), plot=plt.figure().add_subplot(111))
          plt.title("Weibull probability plot of Duration")
          plt.show()
```

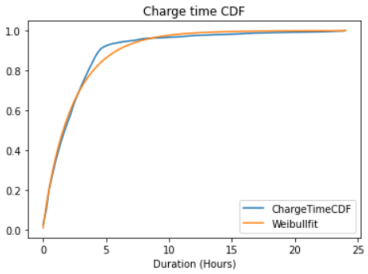


- Check the likeliness of different charge-time durations
- As seen, between 1-3 hours is highly probable
- Outliers detected! (Top right corner of graph / end of curve)

- CDF vs Weibull:

Cumulative Distribution Function against Weibull

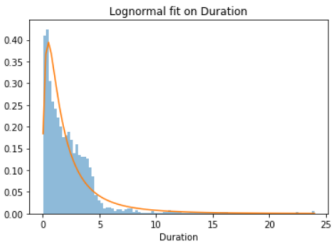
In [379]:
```
import statsmodels.distributions
ecdf=statsmodels.distributions.ECDF(df["Duration"])
plt.plot(x,ecdf(x),label="ChargeTimeCDF")
plt.plot(x,weibull_min(shape,loc,scale).cdf(x), label="Weibullfit")
plt.title("Charge time CDF")
plt.xlabel("Duration (Hours)")
plt.legend()
plt.show()
```



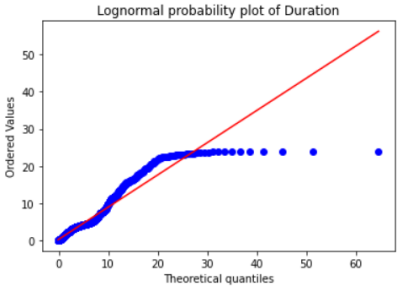
  - Using CDF, we checked the data against itself and was satisfied
  - Curves remain similar, therefore trendline exists
- Lognormal Distribution:

Lognormal distribution and probability

In [382]:
```
shape,loc,scale=scipy.stats.lognorm.fit(df["Duration"])
fitted=scipy.stats.lognorm(shape,loc,scale)
plt.hist(df["Duration"],bins=100,density=True,alpha=0.5)
x=np.linspace(df["Duration"].min(),df["Duration"].max(),100)
plt.plot(x,fitted.pdf(x))
plt.title("Lognormal fit on Duration")
plt.xlabel("Duration")
scipy.stats.probplot(df["Duration"],dist=fitted, plot=plt.figure().add_subplot(111))
plt.title("Lognormal probability plot of Duration")
plt.show()
```



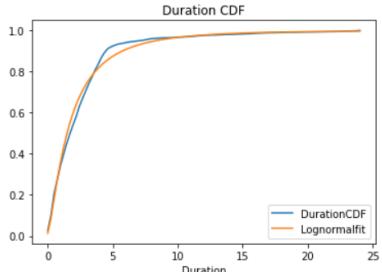
  - Similar story, another cross-check satisfied (validation)
- Lognormal Probability:



  - Check the likeliness of different charge-time durations
  - Different frequency used; 1-5 hours now seems highly probable
    - *Further investigation required
  - Outliers detected! (Middle right of graph / end of curve)
- CDF vs Lognormal:

CDF against Lognormal fit for Duration

```
In [383]: ecdf=statsmodels.distributions.ECDF(df["Duration"])
          plt.plot(x,ecdf(x),label="DurationCDF")
          plt.plot(x,scipy.stats.lognorm(shape,loc,scale).cdf(x), label="Lognormalfit")
          plt.title("Duration CDF")
          plt.xlabel("Duration")
          plt.legend()
          plt.show()
```

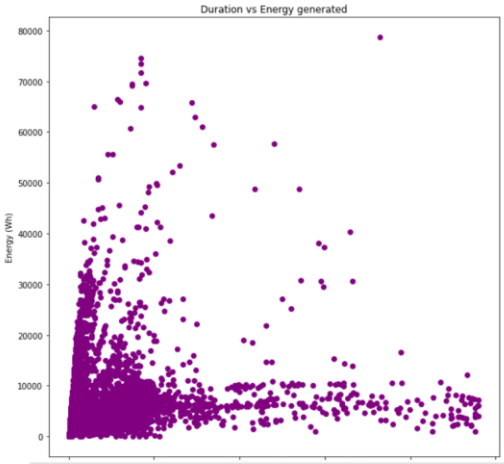


- Similar story to CDF vs Weibull
- Cross-check satisfied (validation)

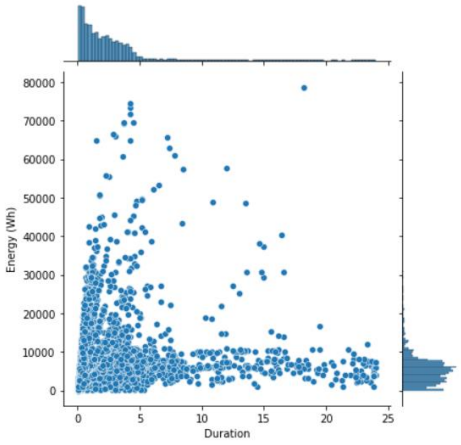
- Scatter Plot:

Scatter plot duration against energy generated

```
In [396]: fig, ax = plt.subplots(figsize=(10, 10))
          ax.scatter(df["Duration"].values,
                     df["Energy (Wh)"],
                     color='purple')
          ax.set(xlabel="Duration",
                 ylabel="Energy (Wh)",
                 title="Duration vs Energy generated")
          plt.show()
```




```
In [397]: sns.jointplot(x="Duration", y="Energy (Wh)", data=df)
Out[397]: <seaborn.axisgrid.JointGrid at 0x24e81667820>
```



- Clear correlation seen between duration and energy between 1-5 hours and 10,000-30,000 (Wh)

Mapping Data

Import libraries for map generation:

- In [1]:
```
import pandas as pd
import folium
from geopy.distance import distance
import warnings
warnings.filterwarnings("ignore")
```

Read in dataset:

- In [2]:
```
stations = pd.read_csv("/Users/merry/Desktop/Greater_Melbourne_and_Geelong.csv")
stations.head().T
```

Out[2]:

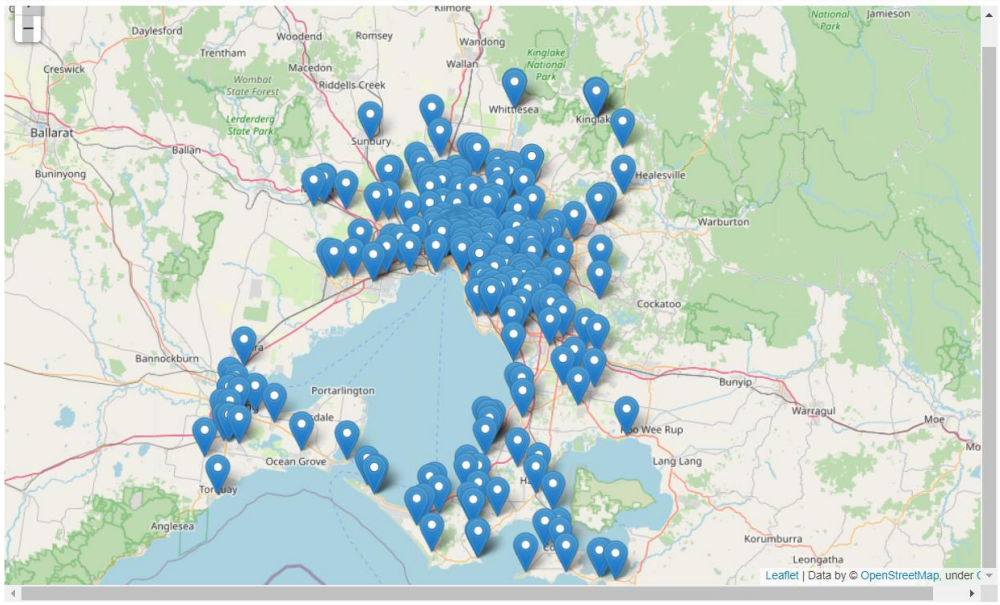
	0	1	2	3	4
name	Geelong Supercharger	Penguin Parade Visitor's Centre	Mornington Supercharger	Kingston Village Square (DC Fast Charger)	BIG4 Phillip Island Caravan Park
address	470-510 Princes Hwy, Geelong, VIC, Australia, ...	995 Ventnor Road Summerlands VIC 3922, Australia	75 Mornington-Tyabb Rd, Mornington VIC 3931, A...	Kingston Village Square, Grubb Rd, Ocean Grove...	24 Old Bridge Drive, Newhaven VIC 3925, Australia
longitude	144.382	145.148	145.051	144.54	145.356
latitude	-38.0652	-38.5055	-38.2343	-38.2467	-38.5168
description	NaN	NaN	NaN	NaN	NaN
parking	NaN	NaN	NaN	NaN	NaN
pricing	NaN	NaN	NaN	NaN	NaN
contact	61280152834	NaN	61280152834	1300518038	0359567227
networks	['Supercharger']	['Non-networked']	['Supercharger']	['Chargefox']	['Non-networked']
total_plugs	6	2	6	4	2

Initial map:

- Boundaries defined
- Latitude and longitude values passed in from dataset

- In [3]:
```
my_map = folium.Map(location = [-37.884254, 144.736465], width = 1000, height = 600)

for _, station in stations.iterrows():
    folium.Marker(location = [station["latitude"], station["longitude"]],
                  popup = station["name"],
                  tooltip = station["name"]).add_to(my_map)

my_map
```

Out[3]:

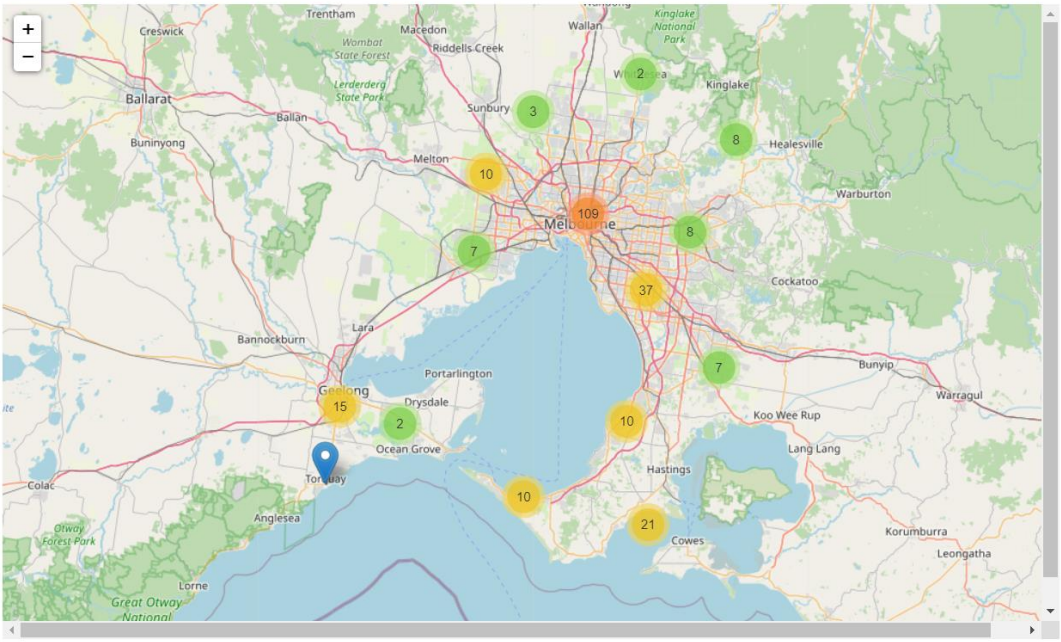


Clustered map:

- Similar to initial map with addition of clustering library and…
- child variable to indicate clusters/neighbours

In [6]:
```
from folium import Marker
from folium.plugins import MarkerCluster

my_map = folium.Map(location = [-37.884254, 144.736465], width = 1000, height = 600)
mc = MarkerCluster()

for _, station in stations.iterrows():
    mc.add_child(Marker(location = [station["latitude"], station["longitude"]],
                 popup = station["name"], tooltip = station["name"]))

my_map.add_child(mc)

my_map
```

Out[6]:



Map Distinguishing lonely EVCSs:

- Create coordinate array for storing distance between stations
- Iterate through stations nearby based on max distance from given EVCS
- If within range of max distance, count is added to nearby stations for EVCS

In [7]:
```
coordinates = []
for _, station in stations.iterrows():
    location = [[station["latitude"], station["longitude"]]]
    coordinates += location
```

In [8]:
```
stations["whether_evcs"] = ""
for i in range(len(coordinates)):
    location1 = coordinates[i]
    evcs_counts = 0
    for j in range(len(coordinates)):

        if i == j:
            pass
        elif distance(coordinates[j], location1) < 5:
            evcs_counts += 1
        else:
            evcs_counts += 0

    if evcs_counts > 0:
        stations["whether_evcs"].iloc[i] = "no"
    else:
        stations["whether_evcs"].iloc[i] = "yes"

stations.head()
```

- Out[8]:

- Indicate using colour which stations are lonely or not
- Red = Lonely / Green = Not lonely

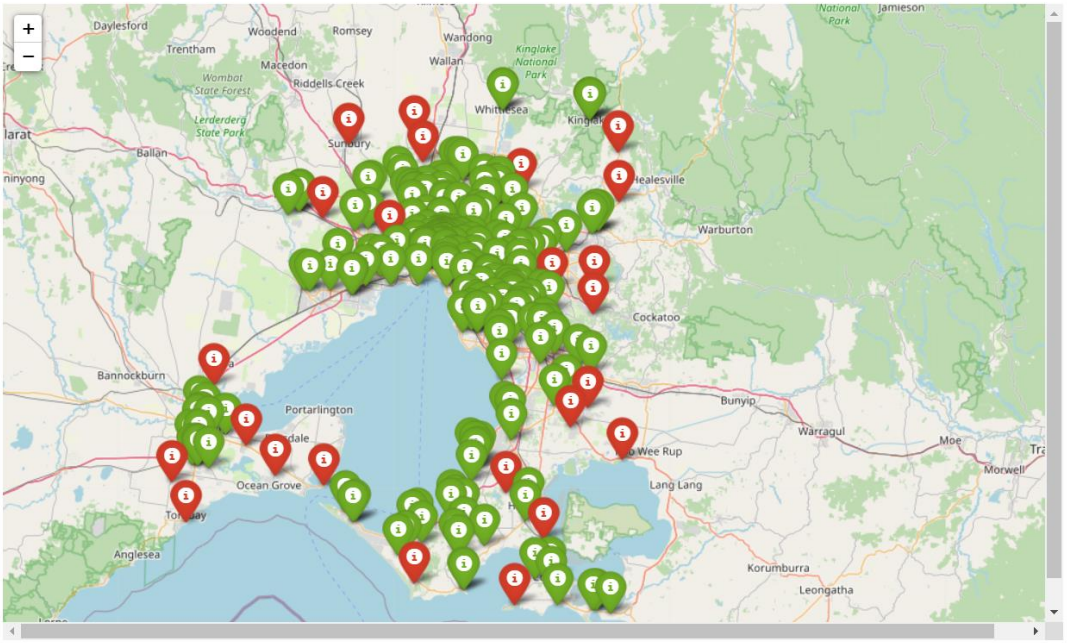
In [9]:
```
def select_marker_color(row):
    if row["whether_evcs"] == "yes":
        return "red"
    elif row["whether_evcs"] == "no" and row["total_plugs"] == "['unknown']":
        return "grey"
    return "green"
```

In [10]:
```
stations["colour"] = stations.apply(select_marker_color, axis = 1)
stations.head()
```

- Out[10]:
- Same as initial map, the map is created

In [13]:
```
my_map = folium.Map(location = [-37.884254, 144.736465], width = 1000, height = 600)

for _, station in stations.iterrows():
    folium.Marker(location = [station["latitude"], station["longitude"]],
                  popup = [station["name"], station["total_plugs"]],
                  tooltip = station["name"],
                  icon = folium.Icon(color = station["colour"])).add_to(my_map)

my_map
```

Out[13]:



- Here we can see based on location and distance there are some EVCSs (red) that could use expansion or additional nearby EVCSs
- Recommendation for where new EVCSs are to be located is required
- There are many ways to approach this such as using other datasets:
    - Foot traffic in area
    - Nearby district (entertainment/work/shopping)
    - Frequent travel routes
- Checking the red EVCSs in particular to see if they are always at max capacity or at least during certain time periods
- In this example its purely based off location and distance, in tandem with a dataset with parking lot locations that could be converted into EVCSs

Map for Recommending new EVCS locations:

- Based off distance between existing EVCS and region's parking lots
-
```
In [104]: parking = pd.read_csv("/Users/merry/Desktop/Off-street_car_parking_2020.csv")
          parking.head()
```
- Assign and check shape of parking dataset
- Create new array for storing parking lot coordinates
- Check EVCS coordinates against parking lot coordinates
    - If coordinates of parking lot and EVCS are less than 2
        - EVCS count is increased by 1
    - Else potential future EVCS detected
    - Check EVCS count
        - If > 0 then future EVCS not recommended
        - Else yes, it is recommended

```
In [106]: parking = parking.loc[parking["Parking spaces"] > 100]
          parking.shape
Out[106]: (442, 10)
```

```
In [107]: parking_coordinates = []

          for _, park in parking.iterrows():
              p = [[park["y coordinate"], park["x coordinate"]]]
              parking_coordinates += p
```

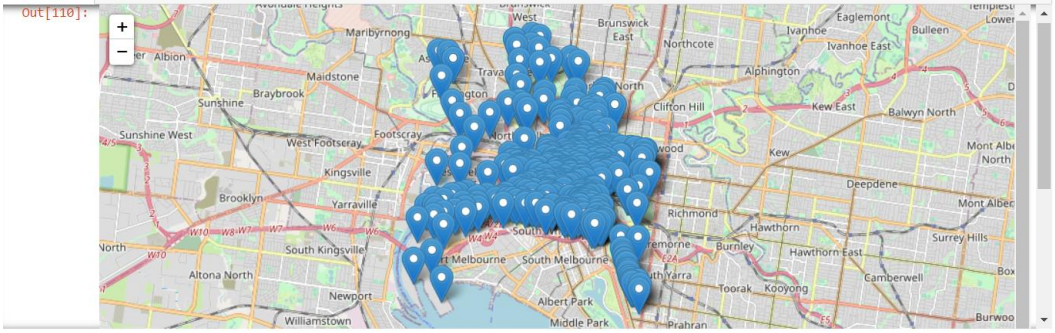
```
In [108]: parking["future_ev_site"] = ""

          for i in range(len(parking_coordinates)):
              location2 = parking_coordinates[i]
              evcs_counts = 0

              for j in range(len(coordinates)):

                  if i == j:
                      pass
                  elif distance(coordinates[j], location2) < 2:
                      evcs_counts += 1
                  else:
                      evcs_counts += 0

              if evcs_counts > 0:
                  parking["future_ev_site"].iloc[i] = "n"
              else:
                  parking["future_ev_site"].iloc[i] = "y"

          parking.head()
```

-
- Mapping the parking lot dataset

```
In [109]: parking["future_ev_site"].unique()
Out[109]: array(['n', 'y'], dtype=object)
```

```
In [110]: parking_map = folium.Map(location = [-37.82098, 144.95651], width = 1000, height = 600)

          for _, park in parking.iterrows():
              folium.Marker(location = [park["y coordinate"], park["x coordinate"]],
                            popup = park["Property ID"]).add_to(parking_map)

          parking_map
```

Out[110]:




```
In [111]: newdata = stations[["name", "address", "longitude", "latitude", "whether_evcs", "colour"]]
          newdata.head()
```

-

- Future EVCS recommendation are seen as blue

```
In [113]: a = parking.loc[parking["future_ev_site"] == "y"]
          a = a[["Block ID", "Building address", "x coordinate", "y coordinate", "future_ev_site"]]
          a["colour"] = "blue"
          a.head()
```

- Append new colour feature

```
In [114]: a.rename(columns = {"Block ID": "name", "Building address": "address", "x coordinate": "longitude",
                              "y coordinate": "latitude", "future_ev_site": "whether_evcs", "colour": "colour"},
                  inplace = True)
          a
```

- Rearrange data frame

```
frames = [newdata, a]

result = pd.concat(frames)
result
```

- Colour and icon selection functions for map generation

```
In [119]: def select_marker_icon(row):
              if row["colour"] == "red":
                  return "star"
              elif row["colour"] == "blue":
                  return "heart"
              return "flash"
```

```
In [120]: result["icon"] = result.apply(select_marker_icon, axis = 1)
          result.head()
```

Out[120]:

```
In [119]: def select_marker_icon(row):
              if row["colour"] == "red":
                  return "star"
              elif row["colour"] == "blue":
                  return "heart"
              return "flash"
```

```
In [120]: result["icon"] = result.apply(select_marker_icon, axis = 1)
          result.head()
```

- New map with future EVCS recommendations

```
In [121]: new_map = folium.Map(location = [-37.884254, 144.736465], width = 1000, height = 600)

          for _, r in result.iterrows():
              folium.Marker(location = [r["latitude"], r["longitude"]],
                            popup = [r["name"], r["address"]],
                            tooltip = r["name"],
                            icon = folium.Icon(icon = r["icon"], color = r["colour"])).add_to(new_map)

          new_map
```

Out[121]: