Central London EV Charging Points:

Data Pipeline & Insights

by Thilina Jayasinghe

Introduction:

This report focuses on EV charging points in Central London. An ETL pipeline was built to extract data from Open Charge Map and Google Places API, transform and clean it, and load it into a BigQuery table. The resulting dataset is visualized through a Google Looker Studio dashboard, enabling analysis and insights into the city's charging infrastructure.

Data sources:

- Google Places API by Google
- Open Charge Map (OCM): https://openchargemap.org/site

Google Places API

Strengths:

- It has a global coverage, including many countries and cities.
- Updated regularly by Google, with data often refreshed daily via user contributions, business listings, and automated sources.
- High reliability in terms of geolocation and basic metadata (especially the latitude and longitude values).
- Includes rich metadata, such as name, address, phone, website, and user ratings.
- Additionally, it's a well-documented API which supports search by keyword, type, location, and radius.

Limitations:

- It has some coverage gaps, especially in EV charging-specific attributes (charger type, network). Therefore, it essential to find alternative data source (like OCM) to obtain that data.
- May include unrelated places if keyword matching isn't precise (e.g., "charging station" might return hotels with "charging" in the name).
- User contributions can vary in quality.

Licensing / Cost:

- Paid beyond the free tier. The cost breakdown will discuss at the end of the report.
- Query limits per day; large-scale data collection may require careful handling or multiple API keys.

Open Charge Map (OCM)

Strengths:

- Mainly focused exclusively on EV vehicle charging stations.
- It has a global coverage, but is best in Europe and North America.
- Detailed EV-specific attributes are often included (eg: Operator, Location Name, Latitude, Longitude, Operational status, Equipment details, Usage and etc)
- Community-driven contributions allow relatively frequent updates.
- Geolocation accuracy is generally good.
- REST API available.
- Additionally, it's a well-documented API which supports schema like POI, DataProvider, OperatorInfo etc.

Limitations:

- Data may be incomplete in regions with fewer contributors.
- Some small or new chargers may be missing.
- Need to perform more data preprocessing (Some addressess, location names are not in a proper format)

Licensing:

- Open data (CC BY-SA 4.0 license).
- No API cost

Summary of the data sources:

Dimensions	Google Places API	OCM
Coverage	Very broad: leverages Google's global Places database, so charging stations that appear on Google Maps are included	Global registry. It has combination of Community-driven data and imported from public/official sources.
Data Quality	High-quality place data, metadata (address, name, types, possibly EV-connector types now) via the new Places API.	Mixed. Because OCM is community-driven, data quality varies per entry. However, their API has a "DataQualityLevel" attribute where we can filter the data.
Cost	Paid API	Free to use public API (with API key).
Accuracy	Very good for location, basic place metadata.	Accuracy varies: since it's crowdsourced, some entries may be duplicated, or missing connector details.

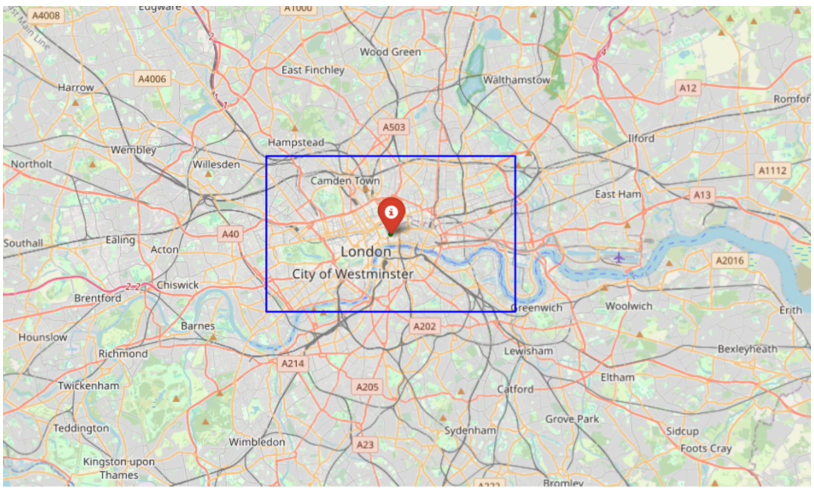
Assumptions:

01. Assumes that the central London area covers the coordinates below

Minimum latitude, Maximum latitude = 51.48, 51.55

Minimum Longitude, Maximum Longitude = -0.20, -0.02

Area of Interest:



02. Assume that the Google Place API can covers all the EV charging points in the central London area (Its depending on the grid size. Eg: 0.001 grid size will capture more EV charging points compared to 0.01).

03. Assume that the EV points identified from Google Place API for EV charging points in the central London area exactly (or almost same with minor changes – at the last decimal points of latitude, longitude values can be vary) match the OCM location data (latitude, longitude values - Manually checked, and it has more than 95% accuracy). This assumption will hold in ETL Option 01 because it uses both the Google Places API and OCM as data sources. The relationship between the two datasets is built using the latitude and longitude values.

04. Charging types were classified using below criteria

Charging types	Power (kW)
Slow	< 7 kW
Fast	7–22 kW
Rapid	> 22 kW

ETL Pipeline (Option 01):

In this option, it will utilize both **Google Place API** and **OCM** as the data sources. In option 02, it will discuss how to use only OCM for the entire pipeline.

First, we have to create a GCP account and enable the Google Places API service. We then need to obtain an Place API key. After that, we need to create a dataset in the BigQuery end and a cloud storage bucket. As an example, we can use the names below (those names align with the code).

- BigQuery dataset: “ev_data_ocm_gpa”
- Cloud Storage Bucket Name: “ev-tracker-data-london-ocm-gpa”

After that, we need to navigate to the Cloud Run functions and create a Cloud Function for our ETL process.

> *Cloud Run Function → Write a Function → select "Use an inline editor to create a function" → Give a service name → select a region → Select the runtime as "Python: 3.11" → Containers → Add the environment variables ("**GOOGLE_API_KEY**" and "**OCM_API_KEY**") → Create*

It will create a Cloud Run function for the ETL process. After completing the initialization, replace the "main.py" file and "requirements.txt" file with the own code. Run the cloud function by clicking the cloud run URL. It will store data in the created BigQuery table and store new CSV file in the cloud storage bucket. Finally, we can build a Google Cloud Scheduler to schedule our ETL cloud function. Then there will be no manual handling in the entire process (No need to click the cloud url to run the cloud function after scheduling a cloud scheduler).

Extract:

- First, it verifies that the required environment variables (GOOGLE_API_KEY, OCM_API_KEY) are present.
- Initialize debug/logging so each major action is recorded.
- Define geographic scan bounds and a grid of latitude/longitude points to query.
- For each grid point, call the "Google Places Nearby Search API" to find nearby EV charging places.
- Handle pagination of nearby search results and accumulate all returned place entries.
- For each place returned, skip if that "place_id" has already been processed in this run.
- For each new "place_id", call the "Google Places Details API" to fetch full place information (name, address, geometry, types)
- If place details are missing or invalid, log and skip that place.
- For each valid place location, call the Open Charge Map (OCM) API to fetch connector types and power/connection details.

- Handle OCM request errors or malformed JSON and substitute empty connector/power lists if needed.

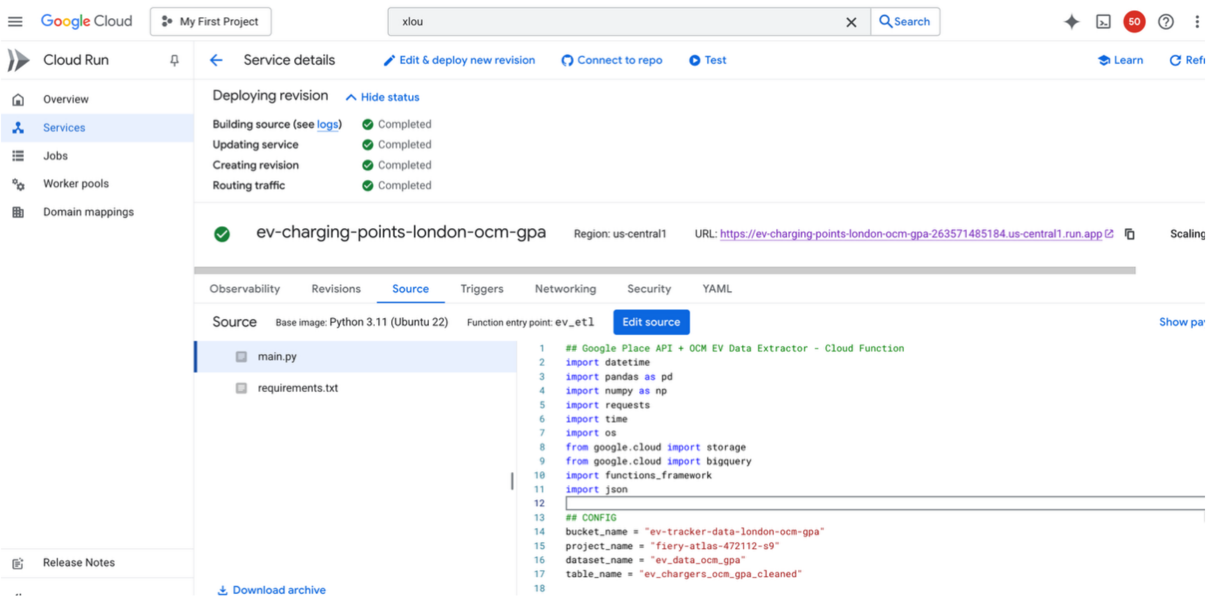
Transform:

- Load all collected API records into a pandas data frame and break down nested JSON fields into separate columns.
- For each EV station:
  - Count the number of connectors
  - Collect power ratings
  - Find min/max power
  - Categorize each connector into: (using the assumptions mentioned above)
    - Slow / Fast or Rapid
  - Determine the maximum charging type per station (Indicate if Rapid/Fast/Slow is available)
  - Extract connection types and current types
  - Check duplicate entries
  - Drop rows missing any of:
    - Number of connectors
    - Min power and Max power
    - Latitude and Longitude

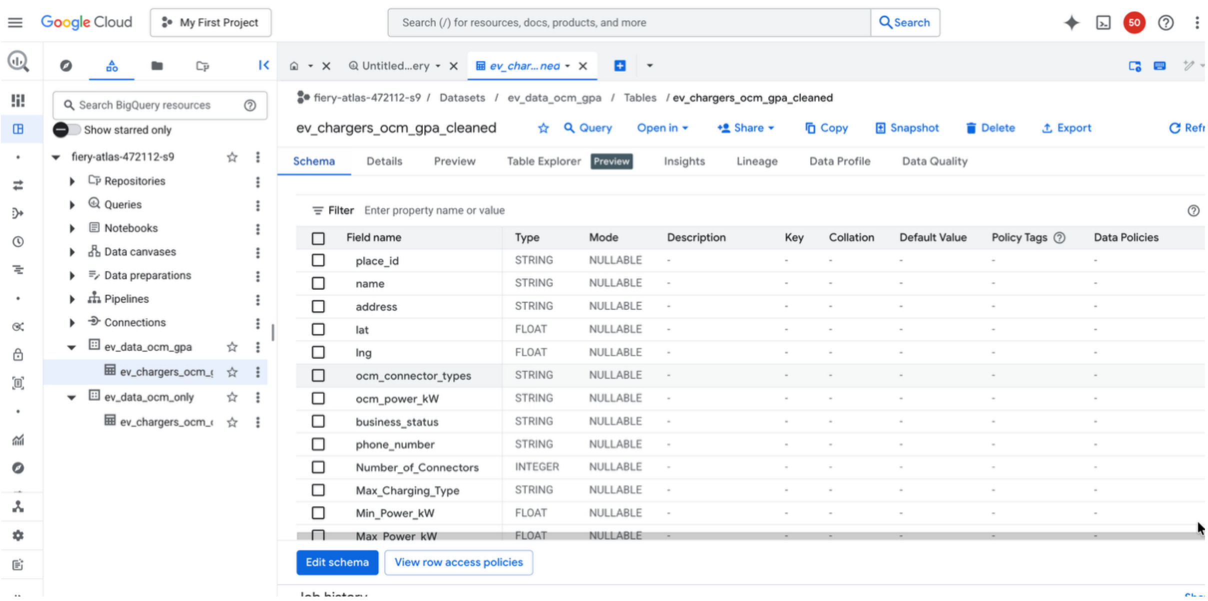
Load:

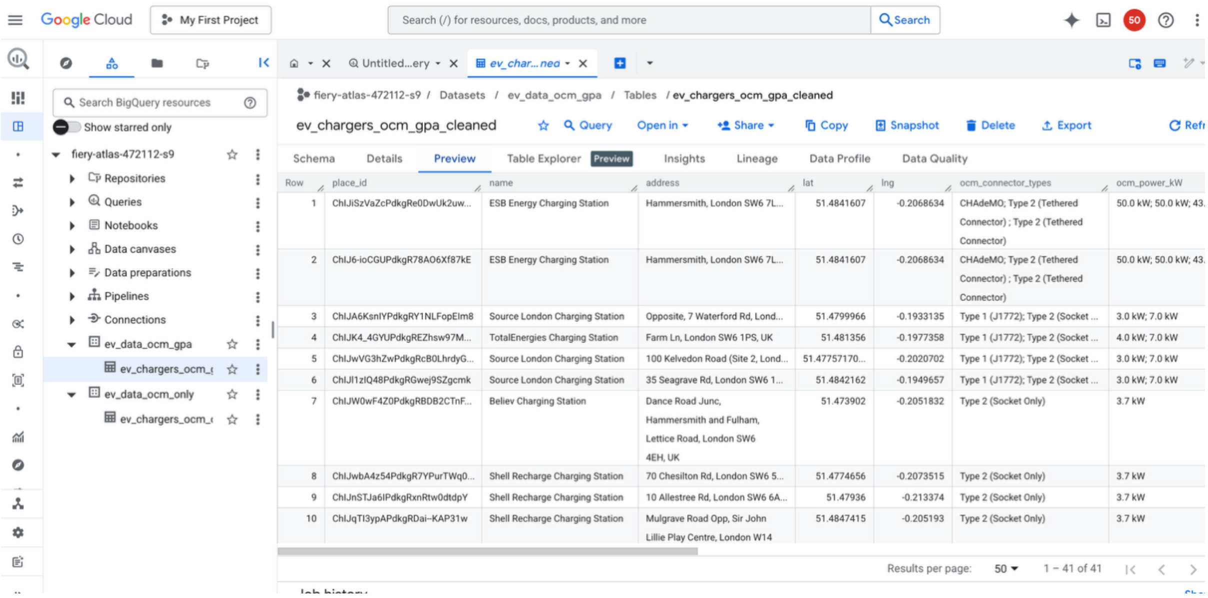
- Save the cleaned dataset to Cloud Storage and upload to the bucket.
- Load the cleaned dataset into BigQuery (It first checks whether the BigQuery table exists. Then it removes records that have a "Place_ID" already in the table. Finally, it loads only the new rows into BigQuery (creates the table if missing).

Cloud Run Job:



BQ Table:





Columns in the final BQ table:

Variable Name	Type	Definition
place_id	string	Unique ID for the charging point
name	string	Name of the Charging Point
address	string	Address of the Charging Point
lat	float	Latitude coordinate of the Charging Point
lng	float	Longitude coordinate of the Charging Point
ocm_connector_types	string	Connector type (Type 01, Type 02 etc)
ocm_power_kW	string	Power list in kW
bussiness_status	string	Operational type of the Charging Point (eg: operational, Partly operational, unknown etc)
Phone number	string	Phone number of the provider
Number_of_Connectors	integer	Number of connectors at the charging point
Min_Power_kW	integer	Minimum power generated from the charging point
Max_Power_kW	integer	Maximum power generated from the charging point
Max_Charging_type	string	Highest charging type (Slow, Fast or Rapid) according to the assumptions
Rapid_Charge_Available	string	Is rapid charge avialble at the point (Yes, No)
Fast_Charge_Available	string	Is fast charge avialble at the point (Yes, No)
Slow_Charge_Available	string	Is slow charge avialble at the point (Yes, No)

ETL Pipeline (Option 02):

In this option, it will discuss how to use only OCM for the entire pipeline.

Like in option 01, we need to create a dataset in the BigQuery end and a cloud storage bucket. As an example, we can use the names below (those names align with the code).

- BigQuery dataset: "ev_data_ocm_only"
- Cloud Storage Bucket Name: "ev-tracker-data-london-ocm-only"

After that, we need to navigate to the Cloud Run functions and create a Cloud Function for our ETL process.

> *Cloud Run Function → Write a Function → select "Use an inline editor to create a function" → Give a service name → select a region → Select the runtime as "Python: 3.11" → Containers → Add the environment variable (" "**OCM_API_KEY**") → Create*

Extract:

- First, define the data source
    - Use the Open Charge Map (OCM) API as the source for EV charging stations.
    - Set the Central London bounding box using the minimum and maximum latitudes and longitudes.
- Then, create a grid of small bounding boxes
- Split the full Central London box into smaller 0.01° × 0.01° grid cells and loop over each grid cell (smaller grid cells (e.g.: 0.001) will capture more EV points).
- After creating the grids, make API calls safely with 5 retry attempts with exponential backoff retry logic.
- In data extraction, a large list of raw, nested OCM records that includes address, operator, status, connection details, usage type, etc.

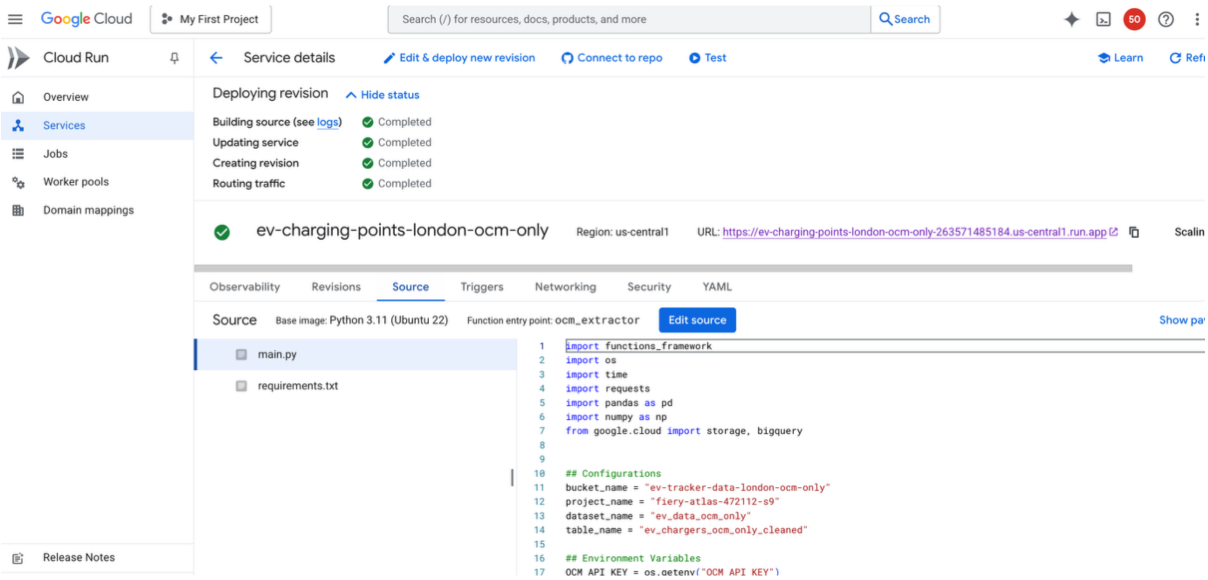
Transform:

- Load all collected API records into a Pandas DataFrame and break down nested JSON fields into separate columns.
- For each EV station:
    - Count the number of connectors
    - Collect power ratings
    - Find min/max power
    - Categorize each connector into: (using the assumptions mentioned above)
        - Slow / Fast or Rapid
    - Determine the maximum charging type per station (Indicate if Rapid/Fast/Slow is available)
    - Extract connection types and current types
    - Check duplicate entries
    - Data conversions: For example, in the “Operator” column, values such as "Unknow", "Unknown Operator" were renamed to "Unknown"
    - Drop rows missing any of:
        - Number of connectors
        - Min power and Max power
        - Latitude and Longitude

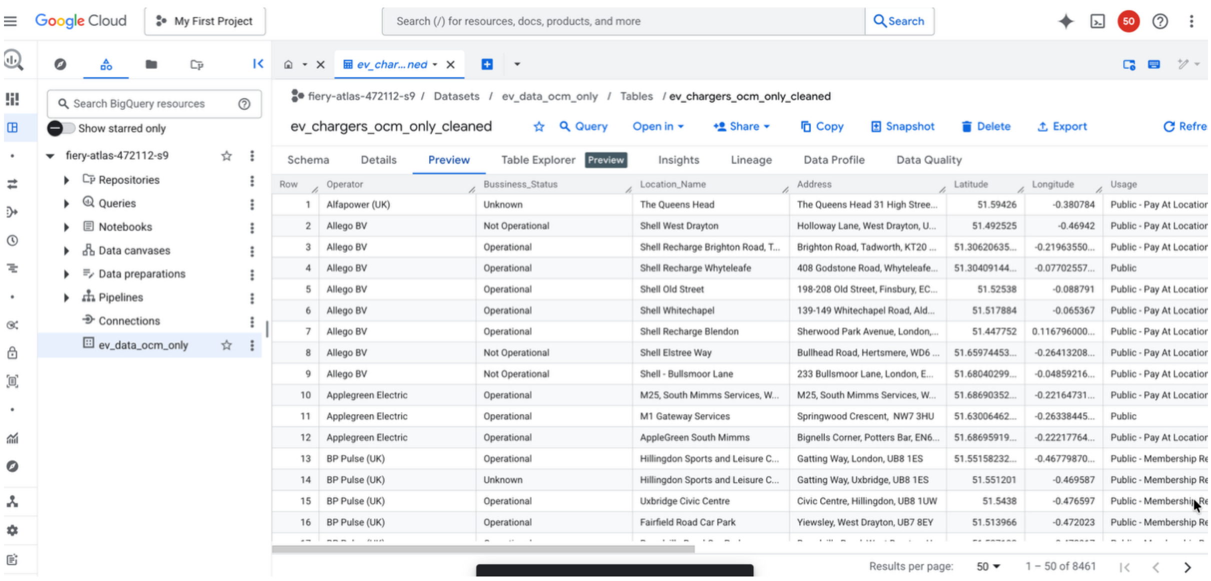
Load:

- Save the cleaned dataset to Cloud Storage and upload to the bucket.
- Load the cleaned dataset into BigQuery (It first checks whether the BigQuery table exists. Then it removes records that have a “Place_ID” already in the table. Finally, it loads only the new rows into BigQuery (creates the table if missing).

Cloud Run Job:

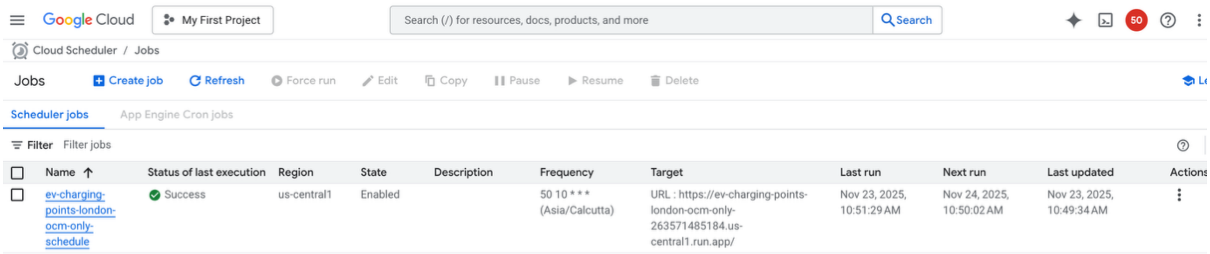


BQ Table:



Cloud Scheduler:

- The code will run every day at 10:50 am. (It will append new data to the existing BQ table)

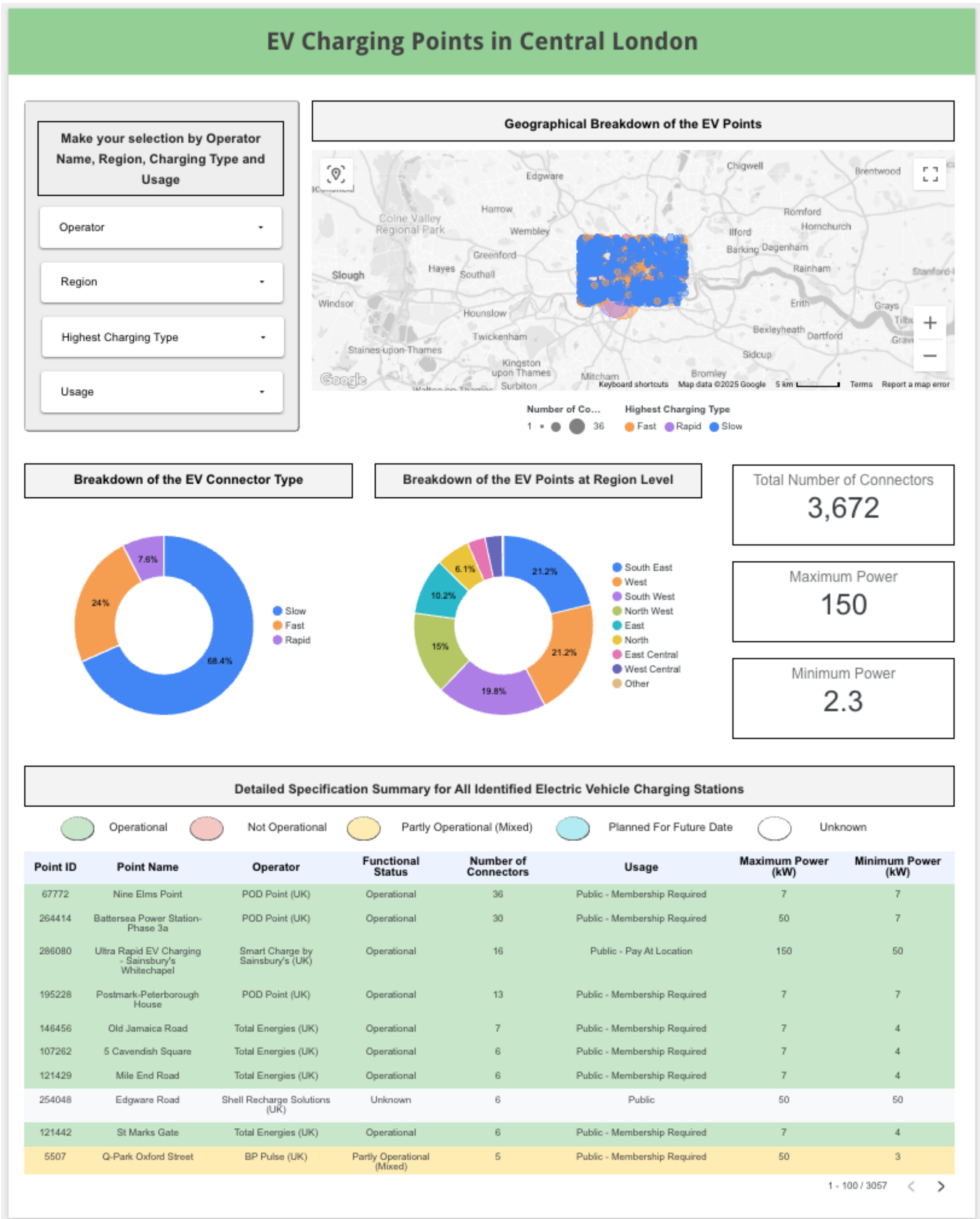


Columns in the final BQ table:

Variable Name	**Type**	**Definition**
Place_id	string	Unique ID for the charging point
Operator	string	Name of the Charging Point Operator
Bussiness_Status	string	Operational type of the Charging Point (eg: operational, Partly operational, unknown etc)
Location_Name	string	Location Name of the Charging Point
Address	string	Address of the Charging Point
Latitude	float	Latitude coordinate of the Charging Point
Longitude	float	Longitude coordinate of the Charging Point
Usage	string	Usage Type (eg: public, private, unknown etc)
Number_of_Connectors	integer	Number of connectors at the charging point
Min_Power_kW	integer	Minimum power generated from the charging point
Max_Power_kW	integer	Maximum power generated from the charging point
Max_Charging_type	string	Highest charging type (Slow, Fast or Rapid) according to the assumptions
Rapid_Charge_Available	string	Is rapid charge avialble at the point (Yes, No)
Fast_Charge_Available	string	Is fast charge avialble at the point (Yes, No)
Slow_Charge_Available	string	Is slow charge avialble at the point (Yes, No)

Dashboard:

The dashboard was built using the Google Looker Studio. It selected Looker Studio as the platform to make a dashboard because all the EV data is in a BQ table. It's very easy to connect the BQ table to Google Looker Studio. Here is the [link]() to the dashboard. The dashboard was built using the data obtained by running ETL pipeline 02.



Newly created fields for the dashboard:

- Extracting the Area Code
  - Create a field named “Area_Code” by extracting the final string after the last comma in the address (e.g., SW6 1PS from 10 Farm Lane, LONDON, SW6 1PS).
  - Used the REGEXP_EXTRACT function, which is the most robust way to find patterns in Looker Studio.

```
REGEXP_EXTRACT( Address , ',\\s*([^,]*)$')
```

- Splitting the Area Code
  - This step splits the extracted “Area_Code” (e.g., SW6 1PS) into two components using the space as a delimiter and rename the first part before the space as “Region_Code” (e.g., SW6)
  - Used REGEXP_EXTRACT to capture all non-space characters from the start of the string.

```
REGEXP_EXTRACT( Area_Code , '^(\\S+)')
```

- Mapping to Full Region Name
  - Create a descriptive field named “Region_Code_Name” by converting the two- or one-character prefix of the “Region_Code” (e.g., NW, SW) into a full name (e.g., NW6: North West, NW1: North West, SW: South West).
  - Used a CASE statement combined with the STARTS_WITH function.
  - The order of the WHEN clauses was corrected to place specific (multi-character) codes before general (single character) codes to ensure accurate mapping (e.g., check for SE before checking for E)

Findings:

- Most of the EV charging points in Central London are operated by **Shell Recharge Solutions**.
- Comparatively, the **Southeast, West, Southwest, and Northwest** London areas have more EV charging points than other London regions.
- Around **7.5%** of the chargers are rapid chargers, while **more than 68%** of them are slow chargers.
- The highest-power EV charging point provides **150 kW**, whereas the minimum is **2.3 kW**.

Cost

The main cost for the project comes from the Google Places API. Here is the breakdown for the Google Places API

- Nearby Search (fetch_nearby): This call is highly optimized for searching and costs $32 per 1,000 requests. Each call for a next_page_token is also a new, billable nearby search request.
- Total Nearby Search Calls: $1 initial call + any calls for next_page_token.

Suggesions:

- **Cost Optimization:** The primary cost driver for ETL pipeline - Option 01 is the Google Places API. Consider optimizing API usage, such as batching requests, caching results, or limiting the frequency of updates to reduce costs.
- **Data Accuracy Improvement:** A hybrid approach combining Google Places API and Open Charge Map (OCM) data (like in ETL pipeline – Option 02) can improve accuracy and coverage. Google Places provides updated business locations, while OCM focuses on EV-specific charging data. Relying solely on OCM may miss new or unregistered locations.
- Currently, a bounding box is used for Central London. Using a more precise shapefile for the area would increase accuracy, as bounding boxes may exclude border points or include irrelevant areas.

- Employing smaller grid sizes (e.g., 0.001° × 0.001° instead of 0.1° × 0.1°) increases spatial accuracy and can capture more EV charging points, providing finer granularity for analysis.

Conclusion

In this project, an ETL pipeline was developed to extract, clean, and load EV charging point data in central london area from Open Charge Map and Google Places API into BigQuery, followed by visualization in Looker Studio. Analysis shows that most chargers in Central London are operated by Shell Recharge Solutions, with the Southeast, West, Southwest, and Northwest areas having higher concentrations. The majority are slow chargers (over 68%), while only 7.5% are rapid, with power ranging from 2.3 kW to 150 kW.