




1 ΣΕΠΤΕΜΒΡΙΟΥ 2024

# GEOSPATIAL DATA MANAGEMENT AND ANALYTICS

MICHALIS KOVAIOS

[mixalis.koveos@gmail.com](mailto:mixalis.koveos@gmail.com)



## Πίνακας περιεχομένων

Introduction .....	2
Q1: Data Extraction and Table Creation.....	2
Python Code:.....	2
SQL Query: .....	3
Results Screenshot: .....	4
Q2: Updating Location Data .....	5
SQL Query: .....	5
Results Screenshot: .....	5
Q3: Deleting Out-of-Bounds Data.....	5
SQL Query: .....	5
Results Screenshot: .....	6
Q4: Removing Duplicate or Near-Duplicate Records.....	6
SQL Query: .....	6
Results Screenshot: .....	7
Q5: Speed Calculation and Removing Outliers.....	7
SQL Query: .....	7
Results Screenshot: .....	8
Q6: Filtering Data by Time .....	9
Python Code:.....	9
Results Screenshot: .....	9
Q7: Identifying Close Encounters Between Moving Vehicles.....	10
Python Code:.....	10
Results Screenshot: .....	11
Q8: Calculating Taxi Behavior Statistics .....	12
Python Code:.....	12
Results Screenshot: .....	13
Q9: Creating Trips and SpeedLimits Tables .....	13
SQL Query: .....	14
Results Screenshot: .....	16
Results Screenshot: .....	16
Q10: Identifying Speed Violations .....	16
SQL Query: .....	16
Results Screenshot: .....	18
Q11: Analyzing Trip Data.....	18
SQL Query: .....	18

Results Screenshot: .....	19
Results Screenshot: .....	20
Q12: Kmeans .....	20
SQL Query: .....	20
Results Screenshot: .....	24

## Introduction

This project focuses on managing and analyzing GPS data from yellow cabs in San Francisco. The dataset contains over 11 million records. The project involves tasks such as data extraction, database creation, data cleaning, and conducting spatial queries. Each task is broken down into distinct steps aimed at extracting valuable insights from the data.

## Q1: Data Extraction and Table Creation

In this task, we wrote a Python script to combine data from multiple TXT files into a single CSV file. Each TXT file represents data for a specific taxi, with information like latitude, longitude, occupancy status, and a timestamp. The script loops through all the TXT files in a specified folder, extracts the data, and writes it to the CSV file. We used basic error handling to skip any lines with missing or invalid data and simplified the code to make it easier to understand.

Also, we created a table called Positions to store information about taxi positions. Each record in the table contains details such as the taxi's unique identifier (`taxi_id`), the time the position was recorded (timestamp), the latitude and longitude of the taxi's location, and whether the taxi was occupied at that time (occupancy). Additionally, we included a location column that stores the geographic point using the GEOMETRY type, which allows us to work with the actual spatial data more effectively. We used the EPSG:4326 coordinate system since it's commonly used for GPS coordinates. The primary key is a combination of `taxi_id` and timestamp, ensuring each position is uniquely identified by both the taxi and the exact time.

To populate the table, we used the COPY command to quickly load data from a CSV file. This method is faster than inserting records one by one and helps with importing large datasets efficiently. By using GEOMETRY and bulk loading, the table is well-prepared for spatial queries, allowing us to easily analyze the movement of taxis based on their locations.

### Python Code:

```
import os

import csv

input_folder = "E:/Downloads/GeoData"

output_file = 'C:/Users/Feuer_Frei/Desktop/TaxiData.csv'

with open(output_file, 'w', newline='') as csvfile:

    writer = csv.writer(csvfile)
```

```
writer.writerow(['taxi_id', 'timestamp', 'latitude', 'longitude', 'occupancy'])
```

```
for txt_file in os.listdir(input_folder):  
    if txt_file.endswith(".txt"):  
        full_path = os.path.join(input_folder, txt_file)  
        taxi_id = os.path.splitext(txt_file)[0]  
  
        with open(full_path, 'r') as f:  
            for line in f:  
                parts = line.strip().split()  
  
                if len(parts) != 4:  
                    print(f"Skipping invalid line in {txt_file}: {line.strip()}")  
                    continue  
  
                try:  
                    lat = float(parts[0])  
                    lon = float(parts[1])  
                    occ = int(parts[2])  
                    time = int(parts[3])  
                except ValueError:  
                    print(f"Skipping line in {txt_file} due to invalid data.")  
                    continue  
  
                writer.writerow([taxi_id, time, lat, lon, occ])  
  
print(f"Data saved to {output_file}")
```

SQL Query:

```

CREATE TABLE Positions (
    taxi_id VARCHAR(50), Taxi identifier
    timestamp BIGINT, Time the position was recorded
    latitude DOUBLE PRECISION, Latitude
    longitude DOUBLE PRECISION, Longitude coordinate
    occupancy SMALLINT, Whether the taxi is occupied or not (1 or 0)
    location GEOMETRY(Point, 4326), Geospatial point for the location (using EPSG:4326 for GPS)
    PRIMARY KEY (taxi_id, timestamp) Composite primary key
);

```

```

COPY Positions(taxi_id, timestamp, latitude, longitude, occupancy)
FROM 'C:/Users/Feuer_Frei/Desktop/DataGeospatial.csv'
DELIMITER ','
CSV HEADER;

```

Results Screenshot:

	taxi_id [PK] character varying (50)	timestamp [PK] bigint	latitude double precision	longitude double precision	occupancy smallint	location geometry
1	new_abboip	1213084687	37.75134	-122.39488	0	[null]
2	new_abboip	1213084659	37.75136	-122.39527	0	[null]
3	new_abboip	1213084540	37.75199	-122.3946	0	[null]
4	new_abboip	1213084489	37.7508	-122.39346	0	[null]
5	new_abboip	1213084237	37.75015	-122.39256	0	[null]
6	new_abboip	1213084177	37.75454	-122.39227	0	[null]
7	new_abboip	1213084172	37.75901	-122.3925	0	[null]
8	new_abboip	1213084092	37.77053	-122.39788	0	[null]
9	new_abboip	1213084032	37.77669	-122.39382	0	[null]
10	new_abboip	1213083971	37.78194	-122.38844	0	[null]
11	new_abboip	1213083910	37.78999	-122.38909	0	[null]
12	new_abboip	1213083855	37.79728	-122.39609	0	[null]
13	new_abboip	1213083811	37.79838	-122.40239	0	[null]
14	new_abboip	1213083736	37.79779	-122.40647	0	[null]
15	new_abboip	1213083715	37.79779	-122.40646	1	[null]
16	new_abboip	1213083655	37.79657	-122.40521	1	[null]
17	new_abboip	1213083600	37.79305	-122.40471	1	[null]
18	new_abboip	1213083535	37.78945	-122.40405	1	[null]
Total rows: 1000 of 11219955    Query complete 00:00:06.570    Ln 64, Col 1						

This table displays the taxi position data after being extracted from the raw text files and stored in the database. Each row represents a unique record for a taxi, showing the taxi's unique identifier (taxi\_id), timestamp, latitude, longitude, occupancy status, and a geometry field (currently empty in this view).

## Q2: Updating Location Data

In this task, we updated the 'location' column in the Positions table to store the geographic point data based on the latitude and longitude values already present in the table. Specifically, we used the ST\_MakePoint function to create a Point geometry using the longitude and latitude of each record. The ST\_SetSRID function then assigns the Spatial Reference System Identifier (SRID) of 4326, which is the standard for GPS coordinates.

SQL Query:

## UPDATE Positions

```
SET location = ST_SetSRID(ST_MakePoint(longitude, latitude), 4326);
```

### Results Screenshot:

	<b>taxi_id</b> [PK] character varying (50)	<b>timestamp</b> [PK] bigint	<b>latitude</b> double precision	<b>longitude</b> double precision	<b>occupancy</b> smallint	<b>location</b> geometry
1	new_abboip	1213084687	37.75134	-122.39488	0	0101000020E6100000C76C3B645995EC09C16BCE82BE04240
2	new_abboip	1213084659	37.75136	-122.39527	0	0101000020E6100000C1C58A1A4C995EC0E3C281902CE04240
3	new_abboip	1213084540	37.75199	-122.3946	0	0101000020E61000001AC05B2041995ECD0A375543541E04240
4	new_abboip	1213084489	37.7508	-122.39346	0	0101000020E61000003012DA722E995EC01CEBE2361AE04240
5	new_abboip	1213084237	37.75015	-122.39256	0	0101000020E6100000F06DFAB31F995EC0158CA4EA04E04240
6	new_abboip	1213084177	37.75454	-122.39227	0	0101000020E6100000EC4CA1F31A995EC00DC347C494E04240
7	new_abboip	1213084172	37.75901	-122.3925	0	0101000020E610000085EB51B81E995EC0224B583D27E14240
8	new_abboip	1213084092	37.77053	-122.39788	0	0101000020E6100000E1EEACDD76995EC01F4B1FBAA0E24240
9	new_abboip	1213084032	37.77669	-122.39382	0	0101000020E6100000B020CD5834995EC0AC90F2936AE34240
10	new_abboip	1213083971	37.78194	-122.38844	0	0101000020E6100000541D7233DC985EC095B7239C16E44240
11	new_abboip	1213083910	37.78999	-122.38909	0	0101000020E6100000D74CBED9E6985EC062156F641EE54240
12	new_abboip	1213083855	37.79728	-122.39609	0	0101000020E61000007311DF8959995EC0A0E062450DE64240
13	new_abboip	1213083811	37.79838	-122.40239	0	0101000020E6100000328FFCC1C0995EC0E7E3DA5031E64240
14	new_abboip	1213083736	37.79779	-122.40647	0	0101000020E61000008733BF9A039A5EC0B58993FB1DE64240
15	new_abboip	1213083715	37.79779	-122.40646	1	0101000020E610000075C8CD70039ASEC0B58993FB1DE64240
16	new_abboip	1213083655	37.79657	-122.40521	1	0101000020E6100000C780ECF5EE995EC0C47C7901F6E54240
17	new_abboip	1213083600	37.79305	-122.40471	1	0101000020E61000004E97C5C4E6995EC0E10B93A982E54240
18	new_abboip	1213083555	37.79045	-122.40495	1	0101000020E6100000D0E0875AD0995EC0F5B08E30C54240
Total rows: 1000 of 11219955      Query complete 00:02:11.075      Ln 96, Col 41						

In this screenshot, the location column has been successfully updated to store the geographic point information in binary format using the GEOMETRY type. The latitude and longitude values have been converted into spatial points, allowing for more efficient spatial queries and analyses in subsequent tasks.

### Q3: Deleting Out-of-Bounds Data

Here, we deleted records from the Positions table where the latitude and longitude values fall outside a specified area. The query removes rows with latitudes outside 37.707 to 37.811 and longitudes outside -122.514 to -122.358. This ensures that we only keep records relevant to a specific geographic region, cleaning up outliers and focusing the dataset on a defined area.

SQL Query:

DELETE FROM Positions

WHERE latitude < 37.707 OR latitude > 37.811

OR longitude < -122.514 OR longitude > -122.358;

### Results Screenshot:

	taxi_id [PK] character varying (50)	timestamp [PK] bigint	latitude double precision	longitude double precision	occupancy smallint	location geometry
1	new_abboip	1213084687	37.75134	-122.39488	0	0101000020E6100000C76C3B645995EC09C16BCE82BE04240
2	new_abboip	1213084659	37.75136	-122.39527	0	0101000020E6100000C1C58A1A4C995EC0E3C281902CE04240
3	new_abboip	1213084540	37.75199	-122.3946	0	0101000020E61000001AC05B2041995EC0A375543541E04240
4	new_abboip	1213084489	37.7508	-122.39346	0	0101000020E61000003012DA722E995EC01CEBE2361AE04240
5	new_abboip	1213084237	37.75015	-122.39256	0	0101000020E6100000F06DFAB31F995EC0158C4AEA04E04240
6	new_abboip	1213084177	37.75454	-122.39227	0	0101000020E6100000EC4CA1F31A995EC00DC347C494E04240
7	new_abboip	1213084172	37.75901	-122.3925	0	0101000020E610000085EB51B81E995EC022AB5B3D27E14240
8	new_abboip	1213084092	37.77053	-122.39788	0	0101000020E6100000E1EEACDD76995EC01F4B1FBAA0E24240
9	new_abboip	1213084032	37.77669	-122.39382	0	0101000020E6100000B020CD5834995EC0AC90F2936AE34240
10	new_abboip	1213083971	37.78194	-122.38844	0	0101000020E6100000541D7233DC985EC095B7239C16E44240
11	new_abboip	1213083910	37.78999	-122.38909	0	0101000020E6100000D74CBED9E6985EC062156F641EE54240
12	new_abboip	1213083855	37.79728	-122.39609	0	0101000020E61000007311DF8959995EC0A0E062450DE64240
13	new_abboip	1213083811	37.79838	-122.40239	0	0101000020E6100000328FFCC1C0995EC0E7E3DA5031E64240
14	new_abboip	1213083736	37.79779	-122.40647	0	0101000020E61000008733BF9A039A5EC0B58993FB1DE64240
15	new_abboip	1213083715	37.79779	-122.40646	1	0101000020E610000075C8CD70039A5EC0B58993FB1DE64240
16	new_abboip	1213083655	37.79657	-122.40521	1	0101000020E6100000C780ECF5EE995EC0C47C7901F6E54240
17	new_abboip	1213083600	37.79305	-122.40471	1	0101000020E61000004E97C5C4E6995EC0E10B93A982E54240
18	new_abboip	1213083535	37.78845	-122.40405	1	0101000020E6100000BFC67E4D8005EC0F3F005F320CF54240
Total rows: 1000 of 9914327    Query complete 00:00:10.579    Ln 81, Col 1						

This screenshot shows the cleaned dataset after irrelevant records were deleted based on latitude and longitude boundaries. Only taxi records within the specified geographic area are retained.

### Q4: Removing Duplicate or Near-Duplicate Records

We removed records where consecutive GPS positions for the same taxi were recorded less than 60 seconds apart. First, we created a common table expression (CTE) that compares each timestamp to the previous one for each taxi using the LAG function. Since all the data is in descending order for each taxi, this ensures we correctly identified the time differences between consecutive records. If the difference between two timestamps is less than 60 seconds, the record was deleted.

#### SQL Query:

WITH CTE AS (

SELECT

taxi\_id,

timestamp AS timestamp1,

LAG(timestamp) OVER (PARTITION BY taxi\_id ORDER BY timestamp) AS timestamp2

FROM Positions

)

DELETE FROM Positions

WHERE (taxi\_id, timestamp) IN (

SELECT taxi\_id, timestamp1

FROM CTE

WHERE (timestamp1 - timestamp2) < 60

);

### Results Screenshot:

	taxi_id [PK] character varying (50)	timestamp [PK] bigint	latitude double precision	longitude double precision	occupancy smallint	location geometry
1	new_abboip	1213084659	37.75136	-122.39527	0	0101000020E6100000C1C58A1A4C995EC0E3C281902CE042...
2	new_abboip	1213084489	37.7508	-122.39346	0	0101000020E61000003012DA722E995EC01CEBE2361AE042...
3	new_abboip	1213084237	37.75015	-122.39256	0	0101000020E6100000F06DFAB31F995EC0158C4AEA04E042...
4	new_abboip	1213084172	37.75901	-122.3925	0	0101000020E610000085EB51B81E995EC022AB5B3D27E142...
5	new_abboip	1213084092	37.77053	-122.39788	0	0101000020E6100000E1EEACDD76995EC01F4B1FBA0E242...
6	new_abboip	1213084032	37.77669	-122.39382	0	0101000020E6100000B020CD5834995EC0AC90F2936AE342...
7	new_abboip	1213083971	37.78194	-122.38844	0	0101000020E6100000541D7233DC985EC095B7239C16E442...
8	new_abboip	1213083811	37.79838	-122.40239	0	0101000020E6100000328FFCC1C0995EC0E7E3DA5031E642...
9	new_abboip	1213083715	37.79779	-122.40646	1	0101000020E610000075C8CD70039A5EC0B58993FB1DE642...
10	new_abboip	1213083600	37.79305	-122.40471	1	0101000020E61000004E97C5C4E6995EC0E10B93A982E54240
11	new_abboip	1213083535	37.78945	-122.40405	1	0101000020E6100000B9FC87F4DB995EC0E2E995B20CE542...
12	new_abboip	1213083475	37.78833	-122.40859	1	0101000020E610000040DEAB56269A5EC0543A58FFE7E442...
13	new_abboip	1213083384	37.78719	-122.41689	0	0101000020E6100000E2016553AE9A5EC07FDE54A4C2E442...
14	new_abboip	1213083323	37.78477	-122.42106	0	0101000020E6100000D769A4A5F29A5EC0E370E65773E442...
15	new_abboip	1213083263	37.7799	-122.42184	0	0101000020E61000004209336DFF9A5EC0401361C3D3E34240
16	new_abboip	1213083147	37.7748	-122.42452	0	0101000020E6100000DE1FEF552B9B5EC06C78A75A52CE342...
17	new_abboip	1213083083	37.77403	-122.43021	0	0101000020E6100000611A868F889B5EC0B80F406A13E34240
18	new_abboip	1213083017	37.77407	-122.43730	0	0101000020E61000002C66A032F5D5EC0A0680B014F342...

Total rows: 1000 of 5957058    Query complete 00:00:12.086    Ln 99, Col 1

This screenshot shows the dataset after removing duplicate or near-duplicate records where consecutive timestamps for the same taxi were recorded less than 60 seconds apart. This cleaning step ensures that only relevant and necessary data is kept for further analysis.

### Q5: Speed Calculation and Removing Outliers

We first added a new column `speed_kmh` to the `Positions` table to store the calculated speed for each taxi. Then, we used a common table expression (CTE) to calculate the speed between consecutive GPS points for each taxi by finding the distance between locations and the time difference between timestamps. The speed was calculated in meters per second and then converted to kilometers per hour. Finally, we updated the `Positions` table with the calculated speeds and deleted any records where the calculated speed was unrealistically high, specifically greater than 120 km/h.

#### SQL Query:

```
ALTER TABLE Positions ADD COLUMN speed_kmh FLOAT;
```

```
WITH position_data AS (
```

```
  SELECT
```

```
    taxi_id,
```

```
    timestamp AS timestamp1,
```



```

LAG(location) OVER (PARTITION BY taxi_id ORDER BY timestamp) AS prev_location,

LAG(timestamp) OVER (PARTITION BY taxi_id ORDER BY timestamp) AS prev_timestamp,

ST_Distance(location::geography, LAG(location) OVER (PARTITION BY taxi_id ORDER BY
timestamp)::geography) AS distance_meters,

(timestamp - LAG(timestamp) OVER (PARTITION BY taxi_id ORDER BY timestamp)) AS
time_seconds

FROM Positions

)

UPDATE Positions

SET speed_kmh = (position_data.distance_meters / position_data.time_seconds) * 3.6 -- Convert m/s
to km/h

FROM position_data

WHERE Positions.taxi_id = position_data.taxi_id

AND Positions.timestamp = position_data.timestamp1

AND position_data.time_seconds > 0;

DELETE FROM Positions

WHERE speed_kmh > 120;

```

## Results Screenshot:

	taxi_id [PK] character varying (50)	timestamp [PK] bigint	latitude double precision	longitude double precision	occupancy smallint	location geometry	speed_kmh double precision
537	new_icdultha	1211943168	37.75127	-122.40309	1	0101000020E6100000FD6FF39CC995EC0A33B889D29E04240	119.98832296039696
538	new_oncixpi	1211267983	37.74392	-122.39589	1	0101000020E6100000FB4024356995EC0906B43C538DF4240	119.92334742720001
539	new_omluaj	1212469860	37.73611	-122.40727	0	0101000020E610000015A930B6109A5EC018213CDA38DE4240	119.89926346342858
540	new_egreosko	1211047067	37.76024	-122.39232	0	0101000020E6100000456458C51B995EC0378E588B4FE14240	119.89612521148234
541	new_idiholv	1211219564	37.77091	-122.39778	1	0101000020E61000002FC03E3A75995EC06614CB2DADE24240	119.88491312449182
542	new_isvayd	1211607545	37.7631	-122.40518	1	0101000020E6100000923F1878EE995EC0EFC9C342ADE14240	119.87801451470456
543	new_ujtrud	1212434687	37.76392	-122.39264	0	0101000020E61000007EC6850321995EC052616C21C8E14240	119.86419433890909
544	new_egwicjuv	1212992003	37.72276	-122.40098	0	0101000020E6100000689604A8A9995EC0B16D516683DC4240	119.8625512836
545	new_eddreba	1211334966	37.7991	-122.43511	1	0101000020E6100000670A9DD7D89B5EC0E71DA7E848E64240	119.85848878904999
546	new_efghakdi	1212692268	37.75649	-122.39225	1	0101000020E6100000C976BE9F1A995EC022E010AAD4E04240	119.84236449330338
547	new_easnsvu	1211476392	37.74655	-122.39399	0	0101000020E6100000DE3CD52137995EC0166A4DF38EDF4240	119.8340771347742
548	new_eliswcky	1211074404	37.73903	-122.4005	1	0101000020E61000001283C0CAA1995EC0A661F88898DE4240	119.833478514
549	new_ewufri	1211431770	37.78349	-122.36279	0	0101000020E610000037548CF337975EC01C5F7B6649E44240	119.82662761834285
550	new_eefayb	1212220236	37.72822	-122.40318	0	0101000020E6100000AF997CB3CD995EC085251E5036DD4240	119.8251854648276
551	new_ifanfadd	1211669774	37.79157	-122.43607	1	0101000020E610000012312592E89B5EC0533F6F2A52E54240	119.8201291048696
552	new_edglevi	1212612319	37.74799	-122.39291	1	0101000020E61000005E11FC6F25995EC016DEE522BEDF4240	119.81369694441177
553	new_itbadpi	1212458337	37.71029	-122.39546	0	0101000020E610000013B875374F995EC0825660C8EADA4240	119.78644302240001
554	new_ejagm	1211000116	37.73658	-122.40325	1	0101000020E6100000DD8A470D0005F00036C2D645DD4240	119.781155406

Here, a new column speed\_kmh has been added to the taxi positions table, displaying the calculated speed for each taxi in kilometers per hour. These values are based on the distance between consecutive GPS points and the time difference between their timestamps. Taxis exceeding 120 km/h were identified and removed.

## Q6: Filtering Data by Time

In this part, we focused on filtering a large dataset of taxi trips to only look at data from May 17, 2008, between 8:00 PM and 9:00 PM. The key idea was to select the time period we were interested in by converting the times into Unix timestamps and using these to filter the rows. After narrowing down the data, we also looked at which vehicles were stationary (with a speed of 0) during this time. This gave us an understanding of how many taxis weren't moving within this time window.

Python Code:

```
import pandas as pd
```

```
file_path = 'C:/Users/Feuer_Frei/Desktop/DataGeospatial_After_Q5.csv'
```

```
taxi_data = pd.read_csv(file_path)
```

```
start_time = 1211044800 # 8:00 PM
```

```
end_time = 1211048400    # 9:00 PM
```

```
may_17_evening = taxi_data[(taxi_data['timestamp'] >= start_time) & (taxi_data['timestamp'] <=
end_time)]
```

```
output_filtered_file = 'C:/Users/Feuer_Frei/Desktop/Filtered_Data_17May_20_21.csv'
```

```
may_17_evening.to_csv(output_filtered_file, index=False)
```

```
stationary_cars = may_17_evening[may_17_evening['speed_kmh'] == 0]
```

```
num_stationary_cars = stationary_cars['taxi_id'].nunique()
```

stationary\_cars

### Results Screenshot:

Stationary vehicle IDs: ['new\_abbop', 'new\_abcolj', 'new\_abdremlu', 'new\_abgibo', 'new\_abjoolau', 'new\_abmyawu', 'new\_abnlar', 'new\_abronkak', 'new\_abtyff', 'new\_abcncij', 'new\_abyakif', 'new\_acdlyito', 'new\_acduuu', 'new\_acgerl', 'new\_acitva', 'new\_ackgrica', 'new\_acpegho', 'new\_

This screenshot shows the number of unique stationary vehicles during the selected time period, with a total of 536 taxis identified as stationary. The list of taxi\_ids is also displayed, indicating which vehicles were not in motion.

## Q7: Identifying Close Encounters Between Moving Vehicles

In this task, we worked with the filtered data from the previous step to find pairs of taxis that were moving and were very close to each other in space and time. First, we converted the location data into geometrical points so that we could use spatial analysis techniques. Then, we built a KD-Tree, which is a structure that lets us efficiently search for taxis that were within 5 meters of each other. We ensured the two vehicles were close not just in space but also in time by making sure the time difference between their positions was 60 seconds or less.

The KD-Tree is ideal for this task because it efficiently handles multidimensional data, like geographical coordinates, allowing us to quickly find nearby points without comparing every pair. In large datasets, this is much faster than brute-force methods, which would require checking every combination of points. Unlike other approaches, such as grid-based methods or brute-force searches, the KD-Tree organizes data in a hierarchical structure, making proximity queries more efficient. While methods like R-trees are also used for spatial searches, KD-Trees are particularly well-suited for this kind of fixed-radius search in Euclidean space.

### Python Code:

```
import pandas as pd

import geopandas as gpd

from shapely.wkb import loads

from scipy.spatial import cKDTree

import numpy as np

filtered_file = 'C:/Users/Feuer_Frei/Desktop/Filtered_Data_17May_20_21.csv'

evening_data = pd.read_csv(filtered_file)

evening_data['geometry'] = evening_data['location'].apply(lambda loc: loads(bytes.fromhex(loc)))

geo_evening_data = gpd.GeoDataFrame(evening_data, geometry='geometry', crs="EPSG:4326")

moving_cars = geo_evening_data[geo_evening_data['speed_kmh'] > 0]

moving_cars_meters = moving_cars.to_crs(epsg=3857)

car_coords = np.array([(point.x, point.y) for point in moving_cars_meters['geometry']])

car_tree = cKDTree(car_coords)

close_car_pairs = car_tree.query_pairs(r=5)

nearby_cars = []

for i, j in close_car_pairs:

    time_diff = abs(moving_cars_meters.iloc[i]['timestamp'] - moving_cars_meters.iloc[j]['timestamp'])
```

```

if time_diff <= 60:
    nearby_cars.append({
        'car_1': moving_cars_meters.iloc[i]['taxi_id'],
        'car_2': moving_cars_meters.iloc[j]['taxi_id'],
        'distance_meters': 5, # Defined proximity threshold
        'time_difference_seconds': time_diff
    })

```

```
close_cars_df = pd.DataFrame(nearby_cars)
```

```
close_cars_df
```

Results Screenshot:

	vehicle_1	vehicle_2	distance_meters	time_difference_seconds
0	new_itpivoa	new_itpivoa	5	60
1	new_ewbglo	new_ewbglo	5	60
2	new_utwoab	new_edodblea	5	3
3	new_avdyab	new_avdyab	5	60
4	new_avpavi	new_avpavi	5	60
...	...	...	...	...
163	new_iorjtwav	new_ochtin	5	25
164	new_orocdu	new_orocdu	5	60
165	new_ucbiyaym	new_ucbiyaym	5	60
166	new_arcurbig	new_utwoab	5	47
167	new_orocdu	new_orocdu	5	60

168 rows × 4 columns

This screenshot presents the pairs of taxis that were found to be within 5 meters of each other and had a time difference of 60 seconds or less. Each pair of vehicles is listed along with the calculated distance between them and the time difference in seconds.

## Q8: Calculating Taxi Behavior Statistics

In this task, we aimed to calculate some statistics for each taxi during the filtered time window. Specifically, we wanted to know how often the taxis sent location updates (measured by time gaps between consecutive records) and how fast they were going on average, as well as their fastest and slowest speeds. We grouped the data by taxi ID, sorted each group by time, and calculated the statistics for each taxi.

### Python Code:

```
import pandas as pd

# Group the filtered data by taxi ID
car_groups = may_17_evening.groupby('taxi_id')

# List to store stats for each car
car_stats = []

# Loop over each taxi group to calculate stats
for taxi_id, group in car_groups:
    # Sort the group by timestamp to analyze time intervals
    group_sorted = group.sort_values(by='timestamp')

    # Calculate time differences between consecutive records
    time_gaps = group_sorted['timestamp'].diff().dropna()

    min_gap = time_gaps.min()
    avg_gap = time_gaps.mean()
    max_gap = time_gaps.max()

    min_speed = group_sorted['speed_kmh'].min()
    avg_speed = group_sorted['speed_kmh'].mean()
    max_speed = group_sorted['speed_kmh'].max()

    car_stats.append({
        'taxi_id': taxi_id,
```

```

'min_time_gap': min_gap,
'avg_time_gap': avg_gap,
'max_time_gap': max_gap,
'min_speed': min_speed,
'avg_speed': avg_speed,
'max_speed': max_speed
})

```

```
car_stats_df = pd.DataFrame(car_stats)
```

```
car_stats_df
```

Results Screenshot:

	taxi_id	min_time_diff	mean_time_diff	max_time_diff	min_speed	mean_speed	max_speed
0	new_abboip	60.0	107.633333	250.0	0.251388	13.841708	77.978755
1	new_abdremlu	60.0	98.909091	289.0	2.741534	25.189745	89.312042
2	new_abgibo	60.0	97.440000	204.0	0.445693	22.331954	118.743719
3	new_abjoolaw	60.0	108.588235	234.0	0.116904	16.085924	53.891579
4	new_abniar	60.0	102.555556	168.0	1.124450	17.495361	80.519351
...	...	...	...	...	...	...	...
380	new_utwoab	60.0	100.028571	228.0	0.000000	11.972821	32.412613
381	new_uvburki	60.0	114.766667	199.0	0.240702	16.472410	37.251650
382	new_uvigcho	60.0	113.866667	296.0	0.000000	31.544517	76.894023
383	new_uvjeahot	60.0	101.074074	757.0	2.169475	31.091530	101.786370
384	new_uvreoipy	60.0	98.958333	149.0	0.140475	20.146435	86.727696

385 rows × 7 columns

In this table, the time gap and speed statistics for each taxi are shown. The min\_time\_diff, mean\_time\_diff, and max\_time\_diff columns provide insights into the frequency of GPS updates for each taxi, while the min\_speed, mean\_speed, and max\_speed columns show the speed behavior over the recorded period.

## Q9: Creating Trips and SpeedLimits Tables

We created two new tables: Trips and SpeedLimits. The Trips table stores information about individual taxi trips, including the taxi ID, departure and arrival times (in Unix timestamp format), starting and ending locations (as geographical points), and the occupancy status (whether the taxi was occupied or free). We used a CTE to calculate when a taxi's occupancy status changed, marking the beginning and

end of a trip. The SpeedLimits table stores information about speed limits for different road segments, including the street name, type, speed limit, and geographical representation of the road as a MultiLineString.

We also created the SpeedLimits table to store information about speed limits for different road segments. Each record includes details such as the road's name, type, start and end points, speed limits (including special limits for school zones), and the geographical representation of the road as a MultiLineString. Finally, we used the COPY command to load data from a CSV file into the SpeedLimits table, ensuring we have accurate speed limit data for analysis.

#### SQL Query:

```
CREATE TABLE Trips (  
    trip_id SERIAL PRIMARY KEY,  
    taxi_id VARCHAR(255),  
    depart_time BIGINT, -- Χρονική στιγμή σε Unix format (BIGINT)  
    arrival_time BIGINT, -- Χρονική στιγμή σε Unix format (BIGINT)  
    depart_location GEOMETRY(Point, 4326),  
    arrival_location GEOMETRY(Point, 4326),  
    occupancy SMALLINT -- 0 = ελεύθερο, 1 = κατειλημμένο  
);  
  
WITH Trip_Segments AS (  
    SELECT  
        taxi_id, -- Μετατροπή σε INTEGER  
        timestamp AS event_time_unix,  
        location AS event_location,  
        occupancy,  
        LAG(occupancy) OVER (PARTITION BY taxi_id ORDER BY timestamp) AS previous_occupancy,  
        LAG(timestamp) OVER (PARTITION BY taxi_id ORDER BY timestamp) AS previous_time_unix,  
        LAG(location) OVER (PARTITION BY taxi_id ORDER BY timestamp) AS previous_location  
    FROM Positions  
)  
INSERT INTO Trips (taxi_id, depart_time, arrival_time, depart_location, arrival_location, occupancy)  
SELECT
```

```
taxi_id,  
previous_time_unix AS depart_time,  
event_time_unix AS arrival_time,  
previous_location AS depart_location,  
event_location AS arrival_location,  
previous_occupancy AS occupancy  
FROM Trip_Segments  
WHERE previous_occupancy IS NOT NULL  
AND occupancy != previous_occupancy;
```

```
CREATE TABLE SpeedLimits (  
    objectid SERIAL PRIMARY KEY,  
    cnn VARCHAR(50),  
    street VARCHAR(100),  
    st_type VARCHAR(50),  
    from_st VARCHAR(100),  
    to_st VARCHAR(100),  
    speedlimit INTEGER,  
    schoolzone BOOLEAN,  
    schoolzone_limit INTEGER,  
    analysis_neighborhood VARCHAR(100),  
    geometry GEOMETRY(MultiLineString, 4326)  
;
```

```
COPY SpeedLimits (objectid, cnn, street, st_type, from_st, to_st, speedlimit, schoolzone,  
schoolzone_limit, analysis_neighborhood, geometry)  
FROM 'E:/ModifiedDownloads/Filtered_Speed_Limits.csv' -- Update with the correct path  
WITH (FORMAT csv, HEADER true, DELIMITER ',', NULL '', QUOTE '');
```



## Results Screenshot:

	trip_id [PK] integer	taxi_id character varying (255)	depart_time bigint	arrival_time bigint	depart_location geometry	arrival_location geometry	occupancy smallint
1	1	new_abboip	1211035796	1211035931	0101000020E61000002C4833164D995EC04E085EF415E04240	0101000020E6100000957D5704FF995EC08E01D9EBDDDF42	0
2	2	new_abboip	1211036158	1211036230	0101000020E61000007407B133859A5EC0A9BC1DE1B4E042	0101000020E61000007B4963B48E9A5EC0C45A7C0A80E142	1
3	3	new_abboip	1211037475	1211037705	0101000020E6100000740CC85EEF9A5EC05C55F65D11E042	0101000020E6100000EF8FF7AA959B5EC0F12900C633E04240	0
4	4	new_abboip	1211037705	1211037776	0101000020E6100000EF8FF7AA959B5EC0F12900C633E04240	0101000020E61000006FF39CC979B5EC040FB912232E042	1
5	5	new_abboip	1211038995	1211039091	0101000020E6100000187D0569C69A5EC0E65C8AABCAE242	0101000020E6100000F4A62215C69A5EC06D73637AC2E242	0
6	6	new_abboip	1211039461	1211040647	0101000020E61000002D026F5F079A5EC0D942908312DE4240	0101000020E6100000DAC9E02879995EC024B4E55C8ADB42	1
7	7	new_abboip	1211041117	1211041181	0101000020E610000082FFAD64C79A5EC07FBC57AD4CE042	0101000020E6100000C286A757CA9A5EC0D44334BA83E042	0
8	8	new_abboip	1211042024	1211042173	0101000020E6100000BF9A0304739A5EC0378E588B4FE54240	0101000020E61000009F8EC70C549A5EC0D93D7958A8E542	1
9	9	new_abboip	1211043006	1211043164	0101000020E61000005070B1A2069B5EC097FF907EFBE64240	0101000020E6100000B0AC3429059B5EC0DEEB0C3923E642	0
10	10	new_abboip	1211043996	1211045137	0101000020E6100000C87BD5CAB4995EC0CFF753E3A5DB42	0101000020E6100000BEA4315A47995EC0986E1283C0DA42	1
11	11	new_abboip	1211045137	1211045324	0101000020E6100000BEA4315A47995EC0986E1283C0DA42	0101000020E6100000F27B9BFEEC995EC0014D840D4FDF42	0
12	12	new_abboip	1211045324	1211045438	0101000020E6100000F27B9BFEEC995EC0014D840D4FDF42	0101000020E6100000F27B9BFEEC995EC007EBFF1CE6DF42	1
13	13	new_abboip	1211045910	1211046091	0101000020E610000043CA4FAA7D9A5EC0876D8B321BE042	0101000020E610000078280AF4899A5EC03E22A64412E14240	0
14	14	new_abboip	1211046331	1211046393	0101000020E61000008907944DB99A5EC020EF552B13E24240	0101000020E6100000FFEC478AC89A5EC02098A3C7FEF42	1
15	15	new_abboip	1211046808	1211046965	0101000020E61000005A0D897B2C9B5EC05951836918E24240	0101000020E61000008D7F9F71E19A5EC07407B13385E24240	0
16	16	new_abboip	1211048883	1211049056	0101000020E6100000164D627839D5EC0FB22A12DE7E24240	0101000020E6100000849ECDAC9FD95EC0172B6A300DE342	1
17	17	new_abboip	1211049745	1211049849	0101000020E61000003A92CB7F489D5EC0486DE2E47EE342	0101000020E6100000E9D495CFF29C5EC04FC7C3194E342	0
18	18	new_abboip	1211051630	1211051760	0101000020E6100000F0A4C1743005F080F5C010073E342	0101000020E6100000E5FA364D8085EC06F0C10073E342	1

This table displays the generated Trips table, which stores the departure and arrival times, as well as the locations for each taxi trip. The occupancy field indicates whether the taxi was occupied during the trip, allowing for further analysis of taxi behavior.

## Results Screenshot:

	objectid [PK] integer	cnn character varying (50)	street character varying (100)	st_type character varying (50)	from_st character varying (100)	to_st character varying (100)	speedlimit integer	schoolzone boolean	schoolzone_limit integer	analysis_neighborhood character varying (100)	geometry geometry
1	27054	1796000	36TH	AVE	SPRICKELS LAKE DR	36TH AVE	0	[null]	0	Golden Gate Park	0105000020E6100000010200000050000000EC381
2	24158	4174000	CLEMENT	ST	24TH AVE	25TH AVE	20	[null]	0	Outer Richmond	0105000020E61000000102000000020000000085CE
3	24214	10270201	PARK PRESIDIO	BLVD	BALBOA ST	CABRILLO ST	35	[null]	0	Inner Richmond, Outer Richmond	0105000020E6100000010200000004000000E2C305
4	24230	524000	12TH	AVE	BALBOA ST	BALBOA ST	0	[null]	0	Inner Richmond	0105000020E610000001020000000200000000085514
5	26703	6064201	GEARY	BLVD	27TH AVE	28TH AVE	25	[null]	0	Outer Richmond	0105000020E61000000102000000040000000056409
6	19292	3403000	CABRILLO	ST	26TH AVE	27TH AVE	0	[null]	0	Outer Richmond	0105000020E610000001020000000200000000020E68
7	24587	629000	18TH	AVE	CLEMENT ST	GEARY BLVD	0	[null]	0	Outer Richmond	0105000020E61000000102000000020000000000044CF
8	25285	19771000	FORT MILEY #3	[null]	VETERANS DR	FORT MILEY #4	0	[null]	0	Lincoln Park	0105000020E61000000102000000020000000000004872
9	24506	6096101	GEARY	BLVD	39TH AVE	40TH AVE	30	[null]	0	Outer Richmond	0105000020E61000000102000000020000000000007F838
10	26033	1370000	25TH	AVE	SEACLIFF AVE	SCENIC WAY	0	[null]	0	Seacliff	0105000020E6100000010200000002000000000000007EE2
11	24922	6096101	GEARY	BLVD	41ST AVE	42ND AVE	0	[null]	0	Outer Richmond	0105000020E610000001020000000200000000000001958
12	26119	10278000	PARKER	AVE	TURK BLVD	GOLDEN GATE AVE	0	[null]	0	Lower Mountain/USF	0105000020E610000001020000000200000000000048948
13	23794	465000	10TH	AVE	GEARY BLVD	ANZA ST	0	[null]	0	Inner Richmond	0105000020E61000000102000000020000000000000A71
14	23807	496000	11TH	AVE	GEARY BLVD	ANZA ST	0	[null]	0	Inner Richmond	0105000020E610000001020000000200000000000008D04
15	25009	3605000	CALIFORNIA	ST	28TH AVE	29TH AVE	0	[null]	0	Outer Richmond, Seacliff	0105000020E610000001020000000200000000000006FA6
16	25025	2323000	ANZA	AVE	65TH AVE	67TH AVE	0	[null]	0	Inner Richmond	0105000020E610000001020000000200000000000000B01
17	25130	523000	12TH	AVE	GEARY BLVD	ANZA ST	0	[null]	0	Inner Richmond	0105000020E610000001020000000200000000000008A82

This screenshot displays the SpeedLimits table, which includes information about speed limits for various road segments. Each record specifies the street name, segment type, speed limit, and geometry of the road segment.

## Q10: Identifying Speed Violations

In this task, we identified vehicles that violated speed limits on specific road segments. First, we created spatial indexes on both vehicle positions (location) and road segments (geometry) to optimize spatial queries. Then, we used a buffer of 50 meters around each road segment to approximate matching between vehicle positions and road segments. By checking if a vehicle's position fell within this buffer and if its speed exceeded the speed limit for that road segment, we identified speed violations.

### SQL Query:

```
CREATE INDEX idx_positions_location ON Positions USING GIST (location);
```

```
CREATE INDEX idx_speedlimits_geometry ON SpeedLimits USING GIST (geometry);
```

```
WITH Buffered_SpeedLimits AS (
```

```
    SELECT
```

```
        objectid,
```

```
        speedlimit,
```

```
        ST_Buffer(geometry::geography, 50)::geometry AS buffered_geometry -- Creating a 50 meter buffer
```

```
    FROM SpeedLimits
```

```
),
```

```
Speed_Violations AS (
```

```
    SELECT
```

```
        p.taxi_id,
```

```
        p.timestamp,
```

```
        p.speed_kmh,
```

```
        s.speedlimit,
```

```
        p.speed_kmh - s.speedlimit AS speed_over_limit,
```

```
        s.objectid AS road_segment_id
```

```
    FROM Positions p
```

```
    JOIN Buffered_SpeedLimits s
```

```
    ON ST_Intersects(p.location, s.buffered_geometry) -- Match vehicle position to road segment using buffer
```

```
    WHERE p.speed_kmh > s.speedlimit -- Only include records where the vehicle exceeds the speed limit
```

```
)
```

```
SELECT
```

```
    taxi_id,
```

```
    timestamp,
```

```
    speed_kmh,
```

```
    speedlimit,
```

```
    speed_over_limit,
```

road\_segment\_id

FROM Speed\_Violations;

Results Screenshot:

	taxi_id character varying (50)	timestamp bigint	speed_kmh double precision	speedlimit integer	speed_over_limit double precision	road_segment_id integer
1	new_uvburki	1211029229	28.741392833657148	0	28.741392833657148	27054
2	new_ackgrica	1211659511	23.224943895882355	0	23.224943895882355	27054
3	new_ideutgoa	1212755245	17.181411921000002	0	17.181411921000002	27054
4	new_ifanfadd	1211900034	42.12730941917356	0	42.12730941917356	27054
5	new_upchimy	1212865152	3.2032619734909087	0	3.2032619734909087	27054
6	new_ecforj	1211478144	36.31943673779105	0	36.31943673779105	27054
7	new_oquiat	1212372304	23.82365418918261	0	23.82365418918261	27054
8	new_ujtrud	1211625885	16.0972127934	0	16.0972127934	27054
9	new_oggluv	1211540803	15.8411273934	0	15.8411273934	27054
10	new_udveoyx	1212551047	21.65515201248555	0	21.65515201248555	27054
11	new_udveoyx	1212551108	0.051996709770491804	0	0.051996709770491804	27054
12	new_ednillo	1211385904	13.946280451764705	0	13.946280451764705	27054
13	new_eesbaj	1211923639	10.512350591855423	0	10.512350591855423	27054
14	new_ofikco	1212340901	17.200272367500002	0	17.200272367500002	27054
15	new_agivle	1212959642	3.4747018674	0	3.4747018674	27054
16	new_urfhod	1212868962	17.56522174556962	0	17.56522174556962	27054
17	new_ubjajo	1211179488	20.019809746523077	0	20.019809746523077	27054
18	new_igcurus	1212367682	14.72220000202	0	14.72220000202	27054
Total rows: 1000 of 15731935    Query complete 00:24:12.371    Ln 202, Col 1						

This table shows the result of the speed violation analysis, where each taxi's speed was compared against the road segment's speed limit. Any taxi exceeding the limit is recorded along with details such as the timestamp, speed, and the specific road segment where the violation occurred.

## Q11: Analyzing Trip Data

In this task, we calculated the percentage of trips where the taxi was occupied and analyzed the distribution of trip length and duration. We also calculated the average, minimum, and maximum trip lengths and durations for the occupied trips, providing valuable insights into the behavior of the taxi fleet.

SQL Query:

```
ALTER TABLE Trips ADD COLUMN length FLOAT;
```

```
SELECT
```

```
    COUNT(CASE WHEN occupancy = 1 THEN 1 END) AS occupancy_true_count,
```

```
    COUNT(*) AS total_count,
```

```

(COUNT(CASE WHEN occupancy = 1 THEN 1 END) * 100.0 / COUNT(*)) AS occupancy_percentage
FROM
    Trips;

```

```

UPDATE Trips SET length = ST_Distance(depart_location::geography, arrival_location::geography);

```

```

SELECT
    taxi_id,
    AVG(length) AS average_length,
    AVG(arrival_time - depart_time) AS average_duration,
    MIN(length) AS min_length,
    MAX(length) AS max_length,
    MIN(arrival_time - depart_time) AS min_duration,
    MAX(arrival_time - depart_time) AS max_duration
FROM
    Trips
WHERE
    occupancy = 1 -- Check for TRUE instead of 1
GROUP BY
    taxi_id;

```

Results Screenshot:

	occupancy_true_count bigint	total_count bigint	occupancy_percentage numeric
1	358930	717951	49.9936625201441324

This screenshot displays summary statistics of taxi occupancy, showing the total number of trips and the percentage of trips where the taxi was occupied.

## Results Screenshot:

	taxi_id character varying (255)	average_length double precision	average_duration numeric	min_length double precision	max_length double precision	min_duration bigint	max_duration bigint
1	new_abboip	521.7533794853501	194.5667107001321004	0	8508.59595663	60	3377
2	new_abcoij	603.592072787619	9353.3333333333333333	37.57585948	5773.37584877	70	131683
3	new_abdremlu	692.1927387870215	251.8340425531914894	0	10155.3349674	60	3752
4	new_abgibo	628.3678936247471	298.9154589371980676	1.76253018	8530.84337475	60	21806
5	new_abjoolaw	617.2614709180524	238.3927392739273927	7.38841357	6918.96597486	61	4059
6	new_abmuyawm	516.6561753374895	217.5937940761636107	0	6908.67554512	61	8594
7	new_abniar	601.5312820596782	232.2907444668008048	0.88132131	8136.88209257	60	13793
8	new_abnovkak	539.2421820750309	207.0932721712538226	0	8429.67003802	61	2839
9	new_abtyff	760.6424260304224	327.6204081632653061	1.41693161	9557.11421067	60	10752
10	new_abwecnij	539.1537052136316	212.5646766169154229	1.41707564	9309.75487598	60	6340
11	new_abyalwif	658.4498263179632	646.2065527065527066	2.21984586	6859.35210514	61	186263
12	new_acdiyito	560.2546701747029	1245.2868217054263566	0	7717.22408182	60	48334
13	new_acduou	486.70741484046204	197.5340179717586650	0	10620.936016	61	15663
14	new_acgerl	539.5682948930308	252.9800884955752212	0	6075.83910466	62	15320
15	new_acitva	542.2297676708745	436.0089686098654709	1.76172051	10759.13255628	61	21971
16	new_ackgrica	537.1638422865333	289.9197707736389685	2.08231134	6759.19325515	61	24726
17	new_acpeggho	597.0668835081086	253.3993963782696177	2.08227717	8607.47345373	60	16017
18	new_acubhe	551.7500775551110	218.4551055670102002	2.21084402	8625.07522206	61	2675

This table summarizes the trip length and duration statistics for each taxi. The average, minimum, and maximum lengths (in meters) and durations (in seconds) are calculated for each taxi during the analyzed period.

## Q12: Kmeans

In this analysis, we are using the **K-Means clustering algorithm** to group vehicle locations based on their geographic coordinates (latitude and longitude) in the city of San Francisco. The goal is to identify different clusters of vehicle activity, which can help in understanding patterns of movement and where vehicles tend to congregate.

First, we use the **elbow method** to determine the optimal number of clusters. The elbow method involves running K-Means for different values of K (the number of clusters) and calculating the **Within-Cluster Sum of Squares (WCSS)** for each. This helps identify the point where increasing the number of clusters does not significantly improve the fit, indicating the optimal K.

Once cluster number is chosen (in this case, 5), we apply K-Means clustering on the vehicle locations. The result is a series of clusters that represent different regions of activity within the city. We then visualize these clusters on a map, using **GeoPandas** to plot the locations and color-code them by their cluster assignment. To provide geographic context, we overlay the data on an **OpenStreetMap** basemap, which allows us to see how the clusters align with actual city streets and landmarks.

## SQL Query:

```
import pandas as pd
```

```
import geopandas as gpd
```

```
from sklearn.cluster import KMeans
```

```
import matplotlib.pyplot as plt

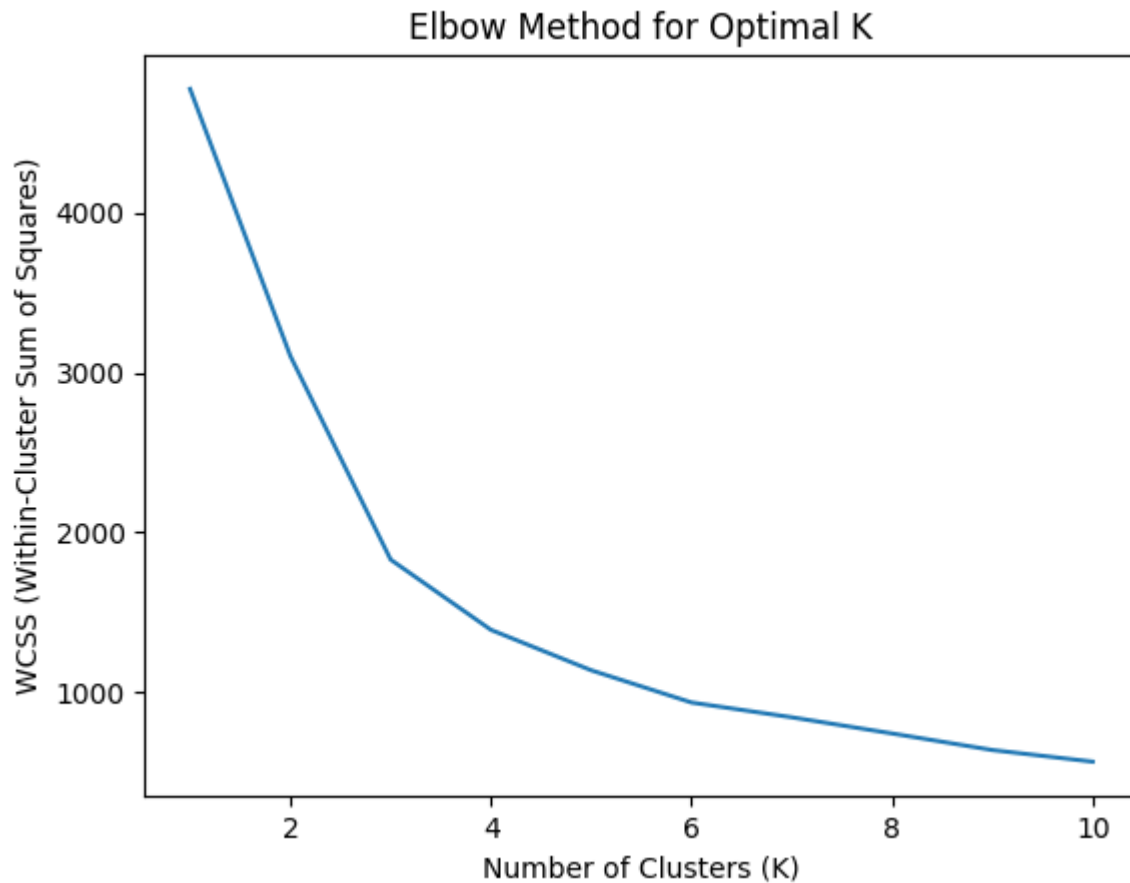
filtered_df = pd.read_csv('C:/Users/Feuer_Frei/Desktop/DataGeospatial_After_Q5.csv')

X = filtered_df[['latitude', 'longitude']]

wcss = []

for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=0).fit(X)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('WCSS (Within-Cluster Sum of Squares)')
plt.show()
```



```
import pandas as pd
import geopandas as gpd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import contextily as ctx

X = filtered_df[['latitude', 'longitude']]

kmeans = KMeans(n_clusters=5, random_state=0).fit(X)

filtered_df['cluster'] = kmeans.labels_

gdf1 = gpd.GeoDataFrame(filtered_df, geometry=gpd.points_from_xy(filtered_df['longitude'],
filtered_df['latitude']), crs="EPSG:4326")
```

```
gdf1 = gdf1.to_crs(epsg=3857)
```

```
fig, ax = plt.subplots(figsize=(10, 10))
```

```
gdf1.plot(column='cluster', ax=ax, legend=True, cmap='viridis', markersize=5)
```

```
ctx.add_basemap(ax, source=ctx.providers.OpenStreetMap.Mapnik)
```

```
ax.set_xlim(gdf1.total_bounds[[0, 2]])
```

```
ax.set_ylim(gdf1.total_bounds[[1, 3]])
```

```
plt.title('Vehicle Clusters in San Francisco')
```

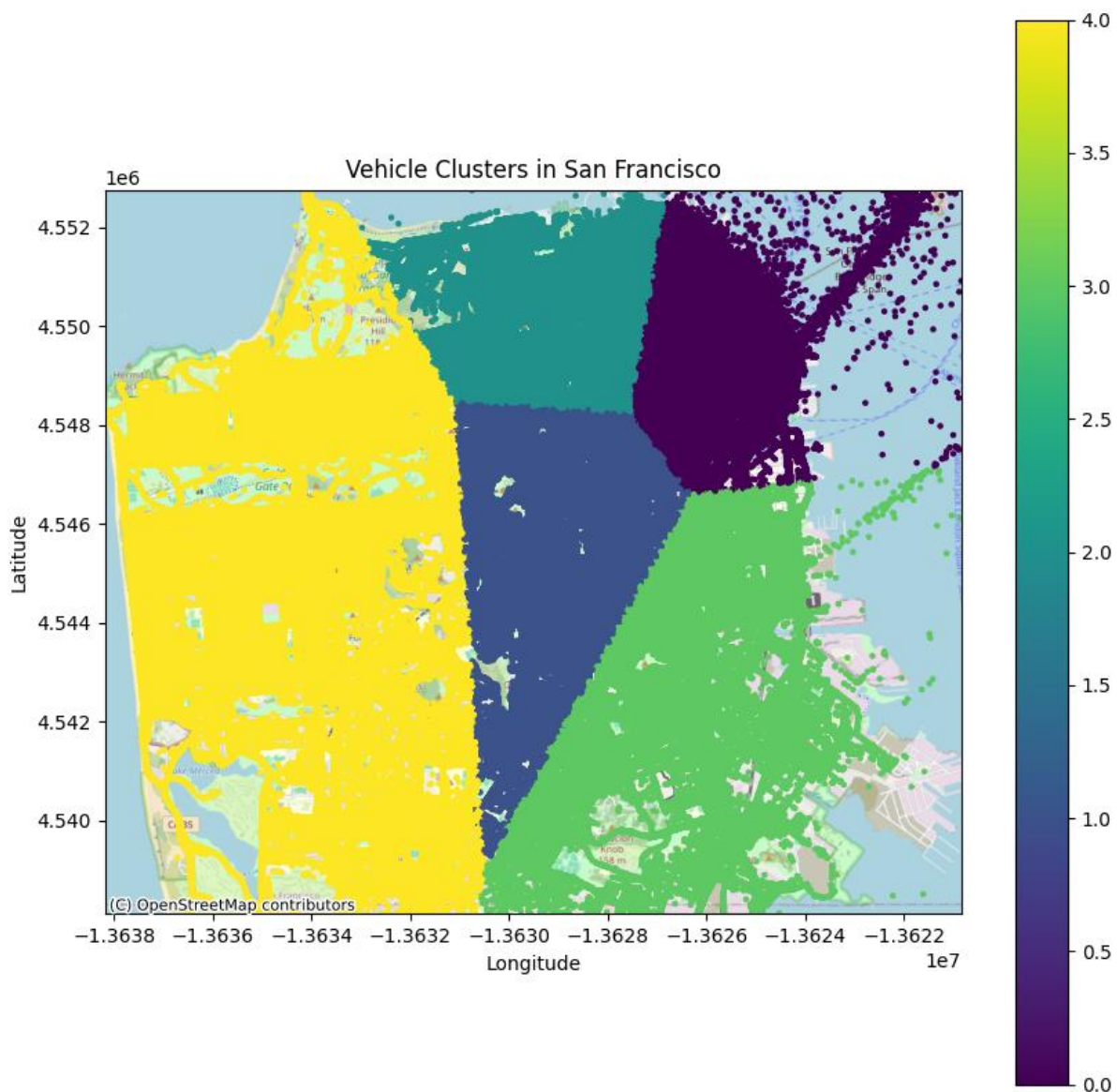
```
plt.xlabel('Longitude')
```

```
plt.ylabel('Latitude')
```

```
plt.show()
```



Results Screenshot:



The clusters in the map show how vehicle activity is distributed across various regions of San Francisco, with each color representing a different concentration of vehicles. The yellow cluster is concentrated in the western part of the city, possibly indicating a residential or less congested area where vehicles are more spread out. The green cluster covers the southeastern area, which could be a mix of residential and commercial zones with moderate traffic. The blue cluster in the central area suggests medium vehicle density, likely following major roads or city center routes, while the purple cluster in the northeastern part points to a high-traffic area, possibly a business district. The cyan cluster in the eastern region may indicate more specialized vehicle movement, perhaps near industrial areas or ports.

However, it is important to note that the clusters do not align with actual road networks, as the K-Means algorithm uses straight-line distances between points. This limitation reduces the accuracy of the results, as vehicle movement in cities is restricted by streets and not by direct, Euclidean paths. As

such, while the clusters provide insights into vehicle distribution, they may not fully reflect how vehicles travel along actual streets.