Metr-LA

In [1]:
```
import numpy as np
import pandas as pd
```

Sensor Data

In [2]:
```
# Import the HDFS class
from pandas import DataFrame, HDFStore
# Create a storage file where data is to be stored

speed_readings = pd.read_hdf(r"D:\Master\Thesis\Code\traffic\metr-la\metr-la.h5")
speed_readings
```

Out[2]:

	773869	767541	767542	717447	717446	717445	773062	767620	737529	717816	...	772167	769372	774204	76980
2012-03-01 00:00:00	64.375000	67.625000	67.125000	61.500000	66.875000	68.750000	65.125000	67.125000	59.625000	62.750000	...	45.625000	65.500000	64.500000	66.42857
2012-03-01 00:05:00	62.666667	68.555556	65.444444	62.444444	64.444444	68.111111	65.000000	65.000000	57.444444	63.333333	...	50.666667	69.875000	66.666667	58.55555
2012-03-01 00:10:00	64.000000	63.750000	60.000000	59.000000	66.500000	66.250000	64.500000	64.250000	63.875000	65.375000	...	44.125000	69.000000	56.500000	59.25000
2012-03-01 00:15:00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.00000
2012-03-01 00:20:00	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.00000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2012-06-27 23:35:00	65.000000	65.888889	68.555556	61.666667	0.000000	54.555556	62.444444	63.333333	59.222222	65.333333	...	52.888889	69.000000	65.111111	55.66666
2012-06-27 23:40:00	61.375000	65.625000	66.500000	62.750000	0.000000	50.500000	62.000000	67.000000	65.250000	67.125000	...	54.000000	69.250000	60.125000	60.50000
2012-06-27 23:45:00	67.000000	59.666667	69.555556	61.000000	0.000000	44.777778	64.222222	63.777778	59.777778	57.666667	...	51.333333	67.888889	64.333333	57.00000
2012-06-27 23:50:00	66.750000	62.250000	66.000000	59.625000	0.000000	53.000000	64.285714	64.125000	60.875000	66.250000	...	51.125000	69.375000	61.625000	60.50000
2012-06-27 23:55:00	65.111111	66.888889	66.777778	61.222222	0.000000	49.555556	65.777778	65.111111	63.000000	61.666667	...	56.000000	67.444444	64.888889	60.88888

34272 rows × 207 columns

Final CSV with sensor speed readings

In [4]:
```
# Unpivoting the data to have one observation per row
# Reset the index to turn the timestamp into a column
speed_readings_noindex = speed_readings.reset_index()

# Now, the timestamp is a column, and we can rename it if needed
speed_readings_noindex.rename(columns={'index': 'Timestamp'}, inplace=True)

# Unpivot the data using melt()
speed_long = speed_readings_noindex.melt(id_vars=["Timestamp"], var_name="sensor_id", value_name="measurement")
speed_long
```

Out[4]:

	Timestamp	sensor_id	measurement
0	2012-03-01 00:00:00	773869	64.375000
1	2012-03-01 00:05:00	773869	62.666667
2	2012-03-01 00:10:00	773869	64.000000
3	2012-03-01 00:15:00	773869	0.000000
4	2012-03-01 00:20:00	773869	0.000000
...	...	...	...
7094299	2012-06-27 23:35:00	769373	62.333333
7094300	2012-06-27 23:40:00	769373	62.000000
7094301	2012-06-27 23:45:00	769373	61.222222
7094302	2012-06-27 23:50:00	769373	63.500000
7094303	2012-06-27 23:55:00	769373	61.777778

7094304 rows × 3 columns

In [ ]:
```
# Number of sensors is 207
speed_long["sensor_id"].nunique()
```

Out[ ]:
```
207
```

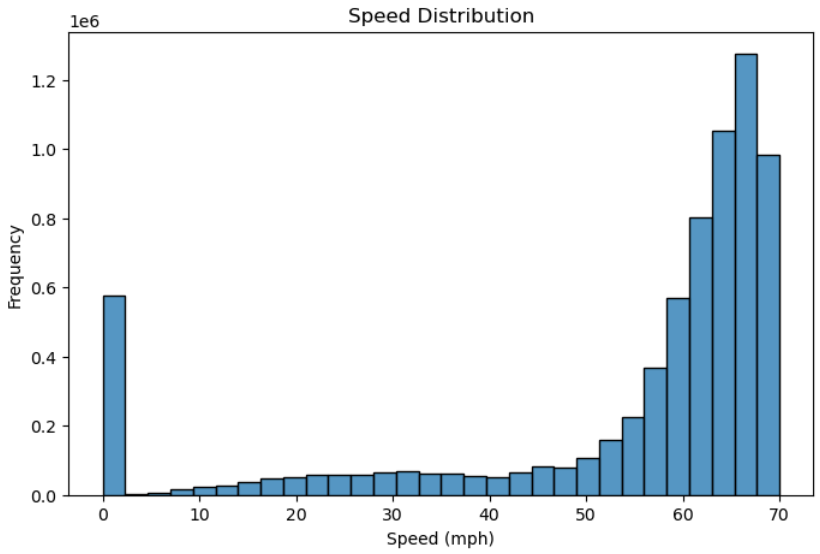
In [ ]:
```
# Basic statistics regarding speed measurements
speed_long['measurement'].describe()
```

Out[ ]:
```
count    7.094304e+06
mean     5.371902e+01
std      2.026143e+01
min      0.000000e+00
25%      5.312500e+01
50%      6.244444e+01
75%      6.625000e+01
max      7.000000e+01
Name: measurement, dtype: float64
```

In [ ]:
```
# No null values
speed_long.isnull().sum()
```

Out[ ]:
```
Timestamp      0
sensor_id      0
measurement    0
dtype: int64
```

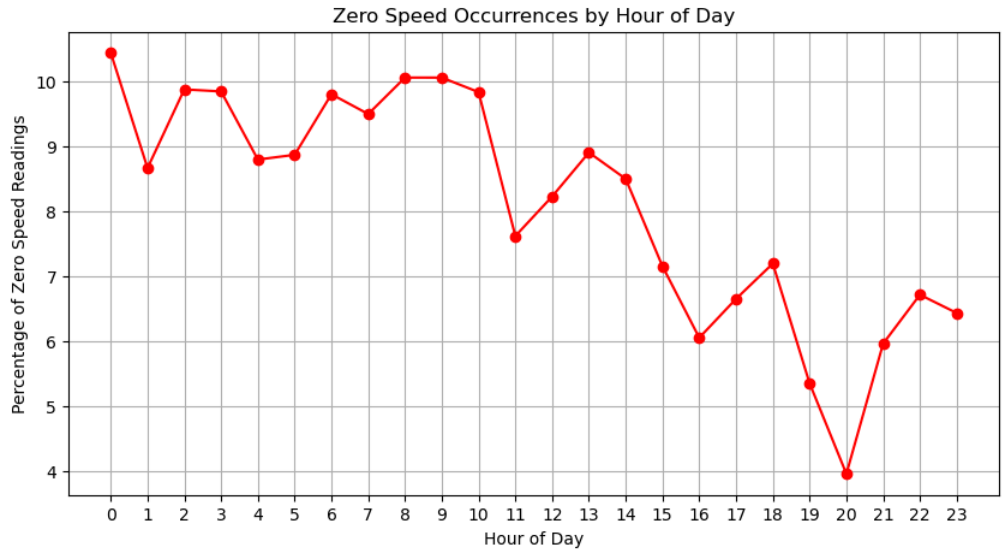
In [ ]:
```
# Speed distribution among all sensor recordings
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8,5))
sns.histplot(speed_long["measurement"], bins=30)
plt.title("Speed Distribution")
plt.xlabel("Speed (mph)")
plt.ylabel("Frequency")
plt.show()
```



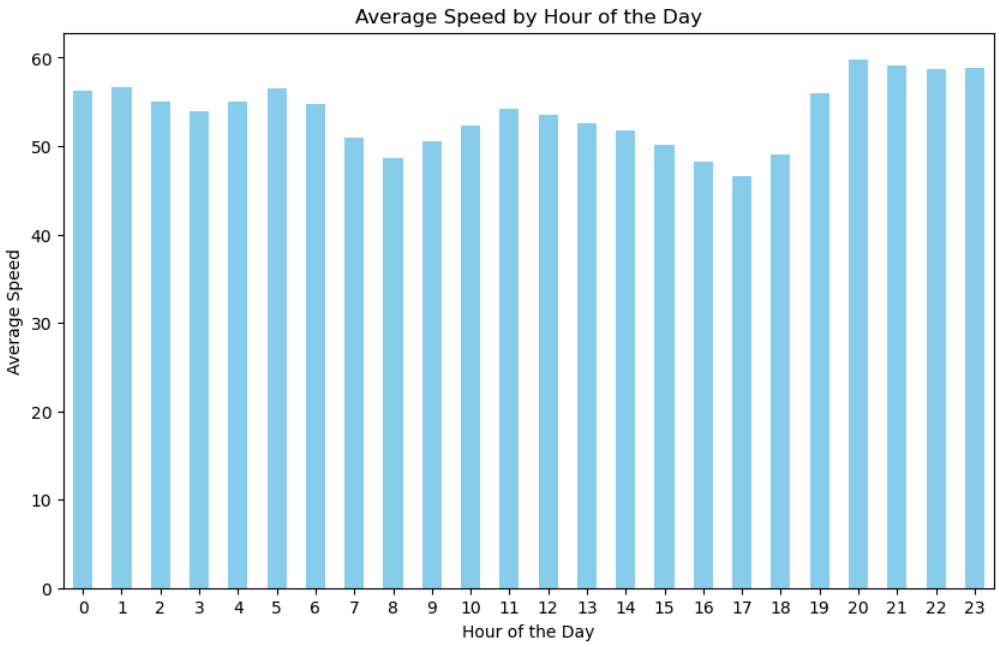
In [ ]:
```
# We observe how 575302 out of 7.09M measurements are 0 (no traffic, stopped traffic)
speed_long.eq(0).sum()
```

Out[ ]:
```
Timestamp           0
sensor_id           0
measurement    575302
dtype: int64
```

In [ ]:
```
# Exploring the occurrences of zero speed along hours of the day
speed_long_hour = speed_long.copy()
speed_long_hour["hour"] = speed_long_hour["Timestamp"].dt.hour

# Compute percentage of zero speed readings per hour
zero_speed_by_hour = speed_long_hour[speed_long_hour["measurement"] == 0].groupby("hour")["measurement"].count() / speed_long_hour.groupby("hour")["me

# Plot results
plt.figure(figsize=(10, 5))
plt.plot(zero_speed_by_hour.index, zero_speed_by_hour.values, marker="o", linestyle="-", color="red")
plt.xlabel("Hour of Day")
plt.ylabel("Percentage of Zero Speed Readings")
plt.title("Zero Speed Occurrences by Hour of Day")
plt.xticks(range(24))
plt.grid()
plt.show()
```



In [ ]:
```
# Average speed recorded by hour of the day
avg_speed_by_hour = speed_long_hour.groupby('hour')['measurement'].mean()

# Plot the average speed by hour
plt.figure(figsize=(10, 6))
avg_speed_by_hour.plot(kind='bar', color='skyblue')
plt.title('Average Speed by Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Average Speed')
plt.xticks(rotation=0)
plt.show()
```



Semantics

In [12]:
```
# Collect sensor locations
sensor_locations = pd.read_csv(r"D:\Master\Thesis\Code\traffic\metr-la\graph_sensor_locations.csv")
sensor_locations.head()
```

Out[12]:

	index	sensor_id	latitude	longitude
0	0	773869	34.15497	-118.31829
1	1	767541	34.11621	-118.23799
2	2	767542	34.11641	-118.23819
3	3	717447	34.07248	-118.26772
4	4	717446	34.07142	-118.26572

In [13]:
```
import geopandas as gpd
from shapely.geometry import Point

# Convert sensor locations to GeoDataFrame
sensor_locations["geometry"] = sensor_locations.apply(lambda row: Point(row["longitude"], row["latitude"]), axis=1)
gdf_sensors = gpd.GeoDataFrame(sensor_locations, geometry="geometry", crs="EPSG:4326")
gdf_sensors
```

Out[13]:

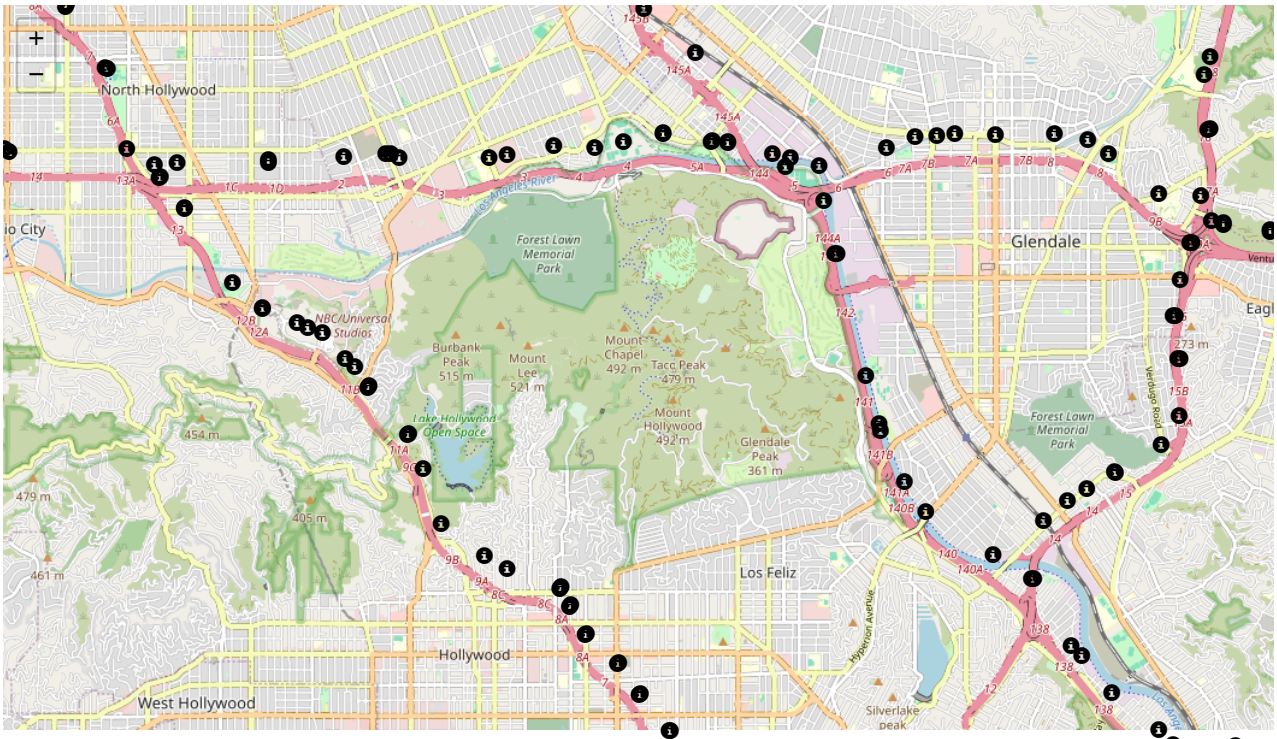
	index	sensor_id	latitude	longitude	geometry
0	0	773869	34.15497	-118.31829	POINT (-118.31829 34.15497)
1	1	767541	34.11621	-118.23799	POINT (-118.23799 34.11621)
2	2	767542	34.11641	-118.23819	POINT (-118.23819 34.11641)
3	3	717447	34.07248	-118.26772	POINT (-118.26772 34.07248)
4	4	717446	34.07142	-118.26572	POINT (-118.26572 34.07142)
...	...	...	...	...	...
202	202	717592	34.14604	-118.22430	POINT (-118.2243 34.14604)
203	203	717595	34.14163	-118.18290	POINT (-118.1829 34.14163)
204	204	772168	34.16542	-118.47985	POINT (-118.47985 34.16542)
205	205	718141	34.15133	-118.37456	POINT (-118.37456 34.15133)
206	206	769373	34.10262	-118.31747	POINT (-118.31747 34.10262)

207 rows × 5 columns

In [14]:
```
import folium

# Create a folium map centered at the average of all sensor locations
map_center = [sensor_locations['latitude'].mean(), sensor_locations['longitude'].mean()]
mymap = folium.Map(location=map_center, zoom_start=13)

# Add markers for each sensor
for index, row in sensor_locations.iterrows():
```

```
    folium.Marker(
        location=[row['latitude'], row['longitude']],
        popup=f"Sensor ID: {row['sensor_id']}",
        icon=folium.Icon(color='blue', icon='info-sign')
    ).add_to(mymap)

mymap
```

Out[14]:



Enrichment per road sensor

In [ ]:

```
# Get all road segments in the LA area
import requests

query = """
[out:json];
area["name"="Los Angeles"]->.searchArea;
(
  way["highway"](area.searchArea);
);
out body;
"""

# Send the query to Overpass API
overpass_url = "http://overpass-api.de/api/interpreter"
response = requests.get(overpass_url, params={'data': query})
road_segments = response.json()['elements']

# Convert to dataframe
road_segments_df = pd.DataFrame(road_segments)
road_segments_df
```

Out[ ]:

	type	id	nodes	tags
0	way	2417713	[297523835, 364042999]	{'alt_name': 'Ritchie Valens Memorial Highway'...
1	way	3124334	[27363713, 371965757, 371965758, 8939644274, 1...	{'highway': 'tertiary', 'name': 'Marmion Way'}
2	way	3126730	[4011845729, 2258860790, 2258860802, 2258860933]	{'highway': 'primary', 'lanes': '4', 'maxspeed...
3	way	3126775	[3645297874, 7298369813, 6753599042, 729836981...	{'highway': 'primary', 'lanes': '4', 'maxspeed...
4	way	3129874	[72402804, 4032187012]	{'bridge': 'yes', 'highway': 'secondary', 'lan...
...	...	...	...	...
193876	way	1370147296	[12689367366, 12689367379, 12689367378, 126893...	{'crossing': 'traffic_signals', 'crossing:isla...
193877	way	1370147297	[12689367368, 12689367385, 12689367384, 126893...	{'crossing': 'traffic_signals', 'crossing:isla...
193878	way	1370147298	[12689367370, 12689367383, 12689367382, 126893...	{'crossing': 'traffic_signals', 'crossing:isla...
193879	way	1370147301	[12689367367, 12689367374, 12689367375, 126893...	{'footway': 'sidewalk', 'highway': 'footway', ...
193880	way	1370147302	[5547032465, 8749200177]	{'access': 'customers', 'highway': 'service', ...

193881 rows × 4 columns

Skip this cell and directly load data from local

In [ ]:
```
"""# Overpass query to get all nodes in a region (e.g., Los Angeles) SKIP, DO CELL THAT LOADS JSON
query = """
[out:json];
area["name"="Los Angeles"]->.searchArea;
node(area.searchArea);
out body;
"""

# Send the request to Overpass API to fetch all nodes in the area
overpass_url = "http://overpass-api.de/api/interpreter"
response = requests.get(overpass_url, params={'data': query})

# Parse the response
data = response.json()

# Extract the latitudes and longitudes of all nodes
nodes_coordinates = []
for element in data['elements']:
    node_id = element['id']
    node_lat = element['lat']
    node_lon = element['lon']
    nodes_coordinates.append({'node_id': node_id, 'latitude': node_lat, 'longitude': node_lon})"""
```

In [ ]:
```
"""import json

def save_nodes_to_file(nodes_coordinates, filename="LA_nodes_coordinates.json"):
    with open(filename, 'w') as f:
        json.dump(nodes_coordinates, f)

# Call the function to save nodes to a file
save_nodes_to_file(nodes_coordinates)"""
```

In [ ]:
```
# Load location of all nodes from Overpass API in the LA area (to later create polyline for each road segment)
import json

# Function to load the dictionary from the JSON file
def load_nodes_from_file(filename=r"D:\Master\Thesis\Code\traffic\metr-la\LA_nodes_coordinates.json"):
    with open(filename, 'r') as f:
        return json.load(f)
nodes_coordinates = load_nodes_from_file()

nodes_coordinates_df = pd.DataFrame(nodes_coordinates)
nodes_coordinates_df
```

Out[ ]:

	node_id	latitude	longitude
0	653688	34.027025	-118.429548
1	653689	34.028639	-118.425473
2	653690	34.030125	-118.421841
3	653691	34.031147	-118.419979
4	653692	34.031978	-118.418457
...	...	...	...
13161929	12668805939	33.931177	-118.383531
13161930	12668805940	33.931177	-118.383597
13161931	12668805941	33.931168	-118.383627
13161932	12668805944	33.931170	-118.387934
13161933	12668805945	33.931170	-118.387968

13161934 rows × 3 columns

In [17]:
```
# Convert nodes into dictionary for easy access
nodes_dict = {node['node_id']: (node['latitude'], node['longitude']) for node in nodes_coordinates}
```

In [18]:
```
# Following code creates a linestring based on the nodes that compose each road segment, to later link it with each sensor
from shapely.geometry import LineString

# Function to get coordinates of nodes from node IDs
def get_segment_coordinates(node_ids, nodes_dict):

    coordinates = []
    for node_id in node_ids:
        # Lookup node coordinates directly in the dictionary
        node_data = nodes_dict.get(node_id)  # This is O(1) lookup time

        if node_data:  # If the node was found
            coordinates.append(node_data)

    return coordinates


# Add LineString column to road_segments_df
```

```
def add_linestring_column(road_segments_df, nodes_dict):
    linestrings = []

    for index, row in road_segments_df.iterrows():
        # Get the coordinates of the nodes forming this segment
        segment_coordinates = get_segment_coordinates(row['nodes'], nodes_dict)

        # Create LineString from the coordinates (if there are at least two coordinates)
        if len(segment_coordinates) >= 2:
            linestring = LineString(segment_coordinates)
        else:
            linestring = None  # Handle edge case for segments with fewer than two nodes

        linestrings.append(linestring)

    # Add the LineString as a new column
    road_segments_df['geometry'] = linestrings
    return road_segments_df

# Apply the function to add the LineString geometry to your road segments DataFrame
road_segments_linestring_df = add_linestring_column(road_segments_df, nodes_dict)

# Convert to geodf
gdf_road_segments = gpd.GeoDataFrame(road_segments_linestring_df, geometry="geometry", crs="EPSG:4326")

gdf_road_segments["geometry"] = gdf_road_segments["geometry"].apply(
    lambda geom: LineString([(lon, lat) for lat, lon in geom.coords]) if geom else None
)

gdf_road_segments
```

Out[18]:

	type	id	nodes	tags	geometry
0	way	2417713	[297523835, 364042999]	{'alt_name': 'Ritchie Valens Memorial Highway'...	LINESTRING (-118.4387 34.25992, -118.43532 34....
1	way	3124334	[27363713, 371965757, 371965758, 8939644274, 1...	{'highway': 'tertiary', 'name': 'Marmion Way'}	LINESTRING (-118.17999 34.11078, -118.17989 34...
2	way	3126730	[4011845729, 2258860790, 2258860802, 2258860933]	{'highway': 'primary', 'lanes': '4', 'maxspeed...	LINESTRING (-118.19845 34.0739, -118.19824 34....
3	way	3126775	[3645297874, 7298369813, 6753599042, 729836981...	{'highway': 'primary', 'lanes': '4', 'maxspeed...	LINESTRING (-118.21562 34.07307, -118.21562 34...
4	way	3129874	[72402804, 4032187012]	{'bridge': 'yes', 'highway': 'secondary', 'lan...	LINESTRING (-118.21962 34.0749, -118.21938 34....
...	...	...	...	...	...
193876	way	1370147296	[12689367366, 12689367379, 12689367378, 126893...	{'crossing': 'traffic_signals', 'crossing:isla...	None
193877	way	1370147297	[12689367368, 12689367385, 12689367384, 126893...	{'crossing': 'traffic_signals', 'crossing:isla...	None
193878	way	1370147298	[12689367370, 12689367383, 12689367382, 126893...	{'crossing': 'traffic_signals', 'crossing:isla...	None
193879	way	1370147301	[12689367367, 12689367374, 12689367375, 126893...	{'footway': 'sidewalk', 'highway': 'footway', ...	None
193880	way	1370147302	[5547032465, 8749200177]	{'access': 'customers', 'highway': 'service', ...	LINESTRING (-118.30818 33.81711, -118.30832 33...

193881 rows × 5 columns

In [ ]:
```
# Number of road segments whose geometry could not be extrapolated
gdf_road_segments['geometry'].isna().sum()
```

Out[ ]: 1375

In [ ]:
```
# Example of tags a road segment can incldue
gdf_road_segments["tags"][0]
```

Out[ ]:
```
{'alt_name': 'Ritchie Valens Memorial Highway',
 'bicycle': 'no',
 'hgv': 'designated',
 'highway': 'motorway',
 'hov:lanes': 'designated||||||',
 'lanes': '7',
 'maxspeed': '65 mph',
 'maxspeed:hgv': '55 mph',
 'maxspeed:trailer': '55 mph',
 'name': 'Golden State Freeway',
 'old_ref': 'US 6;US 99',
 'oneway': 'yes',
 'ref': 'I 5',
 'surface': 'paved'}
```

In [ ]:
```
# Associate, for all sensors, a specific road segment and link the attributes (semantics) to the sensor
import warnings

# Suppress all warnings
warnings.filterwarnings('ignore', category=UserWarning)
```

```
# Ensure CRS consistency
gdf_road_segments = gdf_road_segments.set_crs("EPSG:4326")  # WGS84
gdf_sensors = gdf_sensors.set_crs("EPSG:4326")

# Create spatial index for road segments
road_index = gdf_road_segments.sindex

# Function to find the nearest road segment for a sensor
def find_nearest_road(sensor_point, roads_gdf, road_index, buffer_size = 0.0005, max_buffer_steps = 5):
    # Create a bounding box around the sensor point (tiny buffer)
    for step in range(max_buffer_steps):
        # Create a bounding box around the sensor point (buffer)
        sensor_bbox = sensor_point.buffer(buffer_size * (step + 1))  # Increase buffer size step-by-step

        # Get possible road segment matches using spatial index
        possible_matches_idx = list(road_index.intersection(sensor_bbox.bounds))

        # If matches are found, process and return the nearest road
        if possible_matches_idx:
            possible_matches = roads_gdf.iloc[possible_matches_idx]

            # Compute exact distances and find the nearest segment
            nearest_segment_idx = possible_matches.geometry.distance(sensor_point).idxmin()
            nearest_segment = roads_gdf.loc[nearest_segment_idx]

            return nearest_segment

# Match each sensor to the nearest road segment
sensor_matches = []

for idx, sensor_row in gdf_sensors.iterrows():
    sensor_point = sensor_row.geometry

    # Find the closest road segment
    nearest_road = find_nearest_road(sensor_point, gdf_road_segments, road_index)

        # If no nearest road was found, skip this sensor or add a placeholder
    if nearest_road is None:
        #print(f"No road found for sensor {sensor_row['sensor_id']}")
        continue  # Skip to the next sensor


    # Store the result
    sensor_matches.append({
        "sensor_id": sensor_row["sensor_id"],
        "matched_road_id": nearest_road["id"],
        "matched_road_name": nearest_road.get("tags", {}).get("name", "Unknown"),
        "matched_road_type": nearest_road.get("tags", {}).get("highway", "Unknown"),
        "max_speed": nearest_road.get("tags", {}).get("maxspeed", "Unknown"),
        "lanes": nearest_road.get("tags", {}).get("lanes", "Unknown"),
        "distance_to_road": sensor_point.distance(nearest_road.geometry)
    })

# Convert results into a DataFrame
sensor_matches_df = pd.DataFrame(sensor_matches)
sensor_matches_df
```

Out[ ]:

	sensor_id	matched_road_id	matched_road_name	matched_road_type	max_speed	lanes	distance_to_road
0	773869	156415224	Ventura Freeway	motorway	65 mph	5	0.000019
1	767541	863801551	Glendale Freeway	motorway	65 mph	4	0.000100
2	767542	863796619	Glendale Freeway	motorway	65 mph	4	0.000016
3	717447	148252631	Hollywood Freeway	motorway	55 mph	4	0.000018
4	717446	607788519	Hollywood Freeway	motorway	55 mph	4	0.000015
...	...	...	...	...	...	...	...
175	717592	607802964	Ventura Freeway	motorway	65 mph	5	0.000005
176	717595	119257908	Ventura Freeway	motorway	65 mph	5	0.000033
177	772168	864611298	Ventura Freeway	motorway	65 mph	6	0.000023
178	718141	55904299	Unknown	motorway	55 mph	3	0.000078
179	769373	864558247	Hollywood Freeway	motorway	55 mph	4	0.000045

180 rows × 7 columns

In [ ]:
```
# 156 different road segments found for all existing road sensors
len(sensor_matches_df['matched_road_id'].unique())
```

Out[ ]: 166

In [ ]:
```
# Before 207, now 180, so 27 sensor could not be enriched with road semantics
sensor_matches_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 180 entries, 0 to 179
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sensor_id          180 non-null    int64
 1   matched_road_id    180 non-null    int64
 2   matched_road_name  180 non-null    object
 3   matched_road_type  180 non-null    object
 4   max_speed          180 non-null    object
 5   lanes              180 non-null    object
 6   distance_to_road   180 non-null    float64
dtypes: float64(1), int64(2), object(4)
memory usage: 10.0+ KB
```

In [26]:
```
sensor_matches_df.to_csv("sensor_matches.csv", index=False)
```

Adding of last semantic data, distance between each sensor

In [27]:
```
# Sensor graph data
import pickle

with open(r"D:\Master\Thesis\Code\traffic\metr-la\adj_METR-LA.pkl", 'rb') as f:
    spatial = pickle.load(f, encoding='latin1')

sensor_id = spatial[0]
sensor_nodes = spatial[1]
phyisical_net = spatial[2]
```

In [29]:
```
sensor_distances = pd.read_csv(r"D:\Master\Thesis\Code\traffic\metr-la\\distances_la_2012.csv")
sensor_distances.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 295374 entries, 0 to 295373
Data columns (total 3 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   from    295374 non-null  int64
 1   to      295374 non-null  int64
 2   cost    295374 non-null  float64
dtypes: float64(1), int64(2)
memory usage: 6.8 MB
```

In [44]:
```
# Make sure that we have the distances of all sensors in the network

sensor_ids_in_matches_df = set(sensor_matches_df['sensor_id'])
sensor_ids_in_distance_matrix = set(sensor_distances['from']).union(set(sensor_distances['to']))

# Find missing sensor IDs in the distance matrix
missing_sensors = sensor_ids_in_matches_df - sensor_ids_in_distance_matrix

if not missing_sensors:
    print("All sensor IDs from sensor_matches_df are present in the sensor_distances.")
```

```
All sensor IDs from sensor_matches_df are present in the sensor_distances.
```

In [60]:
```
# Extract the sensor IDs from sensor_matches_df
valid_sensor_ids = set(sensor_matches_df['sensor_id'])

# Filter the sensor_distances dataframe to keep only rows where 'from' and 'to' sensors are valid
filtered_sensor_distances = sensor_distances[sensor_distances['from'].isin(valid_sensor_ids) & sensor_distances['to'].isin(valid_sensor_ids)]

# Show the filtered dataframe
filtered_sensor_distances
```

Out[60]:

	from	to	cost
92905	716328	716328	0.0
92906	716328	716331	4123.8
92907	716328	716337	5179.6
92908	716328	716339	7245.5
92913	716328	716939	4785.1
...	...	...	...
235490	774067	773927	5789.0
235491	774067	773939	8269.0
235492	774067	773953	4486.5
235493	774067	773954	886.7
235494	774067	774067	0.0

9741 rows × 3 columns

In [ ]:
```
# We check distances of a specific sensor that is located far away, and find two insights:
    # distances are one-way
```

```
    # only sensors relatively close appear, (maximum distance around 10.000)
filtered_sensor_distances[(filtered_sensor_distances['from'] == 717804) | (filtered_sensor_distances['to'] == 717804)]
```

Out[ ]:

	from	to	cost
143509	717804	717499	9361.0
143510	717804	717502	8558.5
143511	717804	717804	0.0
143516	717804	717818	9074.5
143521	717804	760024	7864.6
143523	717804	765171	9642.9
143524	717804	767351	10018.3
143527	717804	769430	8554.3
143528	717804	769431	10073.7

In [66]:
```
print(filtered_sensor_distances["cost"].max())
```

```
11895.3
```