Step 1: I activated the correct virtual environment and installed all the necessary libraries.

Step 2: Import Libraries and Read Data

Displaying the first few rows of the data helps verify that it was loaded correctly.

In [3]:
```python
# Step 2: Import Libraries
import pandas as pd  # Data manipulation
import seaborn as sns  # Data visualization
import matplotlib.pyplot as plt  # Plotting
from keplergl import KeplerGl  # Interactive maps

# Step 2.1: Load the datasets
bike_data = pd.read_csv('merged_citibike_weather.csv')  # Load Citibike data
weather_data = pd.read_csv('weather_2022.csv')  # Load weather data

# Step 2.2: Display the first few rows of both datasets for verification
print("Bike Data Sample:\n", bike_data.head())
print("Weather Data Sample:\n", weather_data.head())
```

```
C:\Users\Asus\AppData\Local\Temp\ipykernel_16488\3333922557.py:8: DtypeWarning: Columns (6,8) have mixed types. Specify dtype option on import or set low_memory=False.
  bike_data = pd.read_csv('merged_citibike_weather.csv')  # Load Citibike data
Bike Data Sample:
             ride_id  Temperature  rideable_type started_at ended_at  \
0  BFD29218AB271154         20.8  electric_bike    13:43.4  22:31.5
1  7C953F2FD7BE1302         21.7   classic_bike    30:54.2  41:43.4
2  95893ABD40CED4B8         33.1  electric_bike    52:43.1  06:35.2
3  F853B50772137378         20.2   classic_bike    35:48.2  10:50.5
4  7590ADF834797B4B         34.0   classic_bike    14:23.0  34:57.5

       start_station_name start_station_id            end_station_name  \
0  West End Ave & W 107 St          7650.05  Mt Morris Park W & W 120 St
1             4 Ave & 3 St          4028.04      Boerum Pl\t& Pacific St
2          1 Ave & E 62 St          6753.08               5 Ave & E 29 St
3          2 Ave & E 96 St          7338.02               5 Ave & E 29 St
4          6 Ave & W 34 St           6364.1               5 Ave & E 29 St

  end_station_id  start_lat  start_lng    end_lat    end_lng member_casual  \
0        7685.14  40.802117 -73.968181  40.804038 -73.945925        member
1        4488.09  40.673746 -73.985649  40.688489 -73.991160        member
2        6248.06  40.761227 -73.960940  40.745168 -73.986831        member
3        6248.06  40.783964 -73.947167  40.745168 -73.986831        member
4        6248.06  40.749640 -73.988050  40.745168 -73.986831        member

       year      STATION       DATE  PRCP  TMAX   TMIN
0  1/21/2022  USW00094728  1/21/2022   0.0 -55.0  -99.0
1  1/10/2022  USW00094728  1/10/2022   0.0  44.0  -43.0
2  1/26/2022  USW00094728  1/26/2022   0.0 -21.0  -66.0
3   1/3/2022  USW00094728   1/3/2022   0.0  28.0  -55.0
4  1/22/2022  USW00094728  1/22/2022   0.0 -16.0 -105.0
Weather Data Sample:
        STATION        DATE  PRCP  TMAX  TMIN
0  USW00094728  2022-01-01   201   133   100
1  USW00094728  2022-01-02    10   150    28
2  USW00094728  2022-01-03     0    28   -55
3  USW00094728  2022-01-04     0    11   -71
4  USW00094728  2022-01-05    58    83    -5
```

Step 3: Create a New Column and Aggregate Data

Adding a new_column makes it easier to count trips between station pairs.

Grouping by start_station_name and end_station_name calculates the total trips for each route.

In [4]:
```python
# Step 3.1: Add a new column to represent each trip
bike_data['new_column'] = 1  # This column will act as a counter for aggregation

# Step 3.2: Aggregate data by start and end stations
aggregated_df = bike_data.groupby(['start_station_name', 'end_station_name'])['new_column'].sum().reset_index()

# Step 3.3: Rename the column to something meaningful
aggregated_df.rename(columns={'new_column': 'trip_count'}, inplace=True)

# Step 3.4: Display the first few rows of the aggregated data
print("Aggregated Data Sample:\n", aggregated_df.head())
```

```
Aggregated Data Sample:
   start_station_name  end_station_name  trip_count
0   1 Ave & E 110 St  1 Ave & E 110 St          27
1   1 Ave & E 110 St   1 Ave & E 44 St           2
2   1 Ave & E 110 St   1 Ave & E 68 St           1
3   1 Ave & E 110 St   1 Ave & E 78 St           3
4   1 Ave & E 110 St   1 Ave & E 94 St          15
```

Step 4: Initialize a Kepler.gl Map

Displaying the map now lets you ensure it's rendering correctly in our notebook environment.

In [5]:
```python
# Step 4: Initialize a Kepler.gl map
map = KeplerGl()  # Create a new map instance

# Display the map (in Jupyter Notebook or similar environments)
map
```

```
User Guide: https://docs.kepler.gl/docs/keplergl-jupyter
KeplerGl()
```

Step 5: Customize the Output of Your Map

Adding data to the map makes it accessible for visualization.

Customizations (arcs, color palettes, etc.) are performed in Kepler.gl's graphical interface for simplicity.

In [6]:
```python
# Step 5.1: Add the aggregated data to the map
map.add_data(data=aggregated_df, name='trip_data')

# Note: Customize the layers, colors, and arcs directly within the Kepler.gl interface.
# Example steps (to be done in the Kepler.gl UI):
# 1. Go to "Layer Settings" and add a new layer.
# 2. Set the data source to 'trip_data'.
# 3. Add arcs between 'start_station_name' and 'end_station_name'.
# 4. Set color based on 'trip_count'.
```

Step 6: Add Filters and Interpret Output

Observations:

Add Filters:

- Filter trips where `trip_count > 10` to focus on significant routes.

Observations:

1. **Busiest Routes**:
   - The route from **1 Ave & E 110 St** to **1 Ave & E 94 St** has a trip count of 15, making it one of the busiest routes.
   - The route **1 Ave & E 110 St** to itself (a round trip) is even busier with 27 trips.
2. **Key Stations**:
   - **1 Ave & E 110 St** stands out as a primary hub with the highest trip counts across multiple routes.
   - This station's location might suggest its importance in a densely populated or high-demand area.
3. **Insights on Station Usage**:
   - Routes with higher trip counts (e.g., 1 Ave & E 110 St to 1 Ave & E 94 St) might indicate proximity to residential areas, parks, or business districts.
   - Round trips originating and ending at **1 Ave & E 110 St** suggest recreational or convenience-based usage.

Step 7: Create a Config Object and Save the Map

In [7]:
```python
# Step 7.1: Save the current configuration to a variable
config = map.config  # Capture the map's configuration
```

```python
# Step 7.2: Save the map to an HTML file
map.save_to_html(file_name='citibike_trip_map.html')  # File will be saved locally
```

Map saved to citibike_trip_map.html!

In [ ]:

```python

```