Problem Set 6 - Waze Shiny Dashboard

Peter Ganong, Maggie Shi, and Andre Oviedo

2024-11-23

1. **ps6:** Due Sat 23rd at 5:00PM Central. Worth 100 points (80 points from questions, 10 points for correct submission and 10 points for code style) + 10 extra credit.

We use (*) to indicate a problem that we think might be time consuming.

Steps to submit (10 points on PS6)

1. "This submission is my work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: BZ
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **___** (2 point)
3. Late coins used this pset: 0 Late coins left after submission: 4
4. Before starting the problem set, make sure to read and agree to the terms of data usage for the Waze data here.
5. Knit your `ps6.qmd` as a pdf document and name it `ps6.pdf`.
6. Push your `ps6.qmd`, `ps6.pdf`, `requirements.txt`, and all created folders (we will create three Shiny apps so you will have at least three additional folders) to your Github repo (5 points). It is fine to use Github Desktop.
7. Submit `ps6.pdf` and also link your Github repo via Gradescope (5 points)
8. Tag your submission in Gradescope. For the Code Style part (10 points) please tag the whole correspondingsection for the code style rubric.

Notes: see the Quarto documentation (link) for directions on inserting images into your knitted document.

IMPORTANT: For the App portion of the PS, in case you can not arrive to the expected functional dashboard we will need to take a look at your `app.py` file. You can use the following *code chunk template to "import" and print the content of that file. Please, don't forget to also tag the corresponding code chunk as part of your submission!*

```python
def print_file_contents(file_path):
    """Print contents of a file."""
    try:
        with open(file_path, 'r') as f:
            content = f.read()
            print("```python")
            print(content)
            print("```")
    except FileNotFoundError:
        print("```python")
        print(f"Error: File '{file_path}' not found")
        print("```")
    except Exception as e:
        print("```python")
        print(f"Error reading file: {e}")
        print("```")

print_file_contents("./top_alerts_map_byhour/app.py") # Change accordingly
```

```
RendererRegistry.enable('png')
```

Background

Data Download and Exploration (20 points)

1.

```python
import zipfile

with zipfile.ZipFile('waze_data.zip', 'r') as zip_ref:
    zip_ref.extractall()

waze_sample_df = pd.read_csv('waze_data_sample.csv')

# Specified data types
data_types = {
    'city': 'Nominal',
    'confidence': 'Quantitative',
```

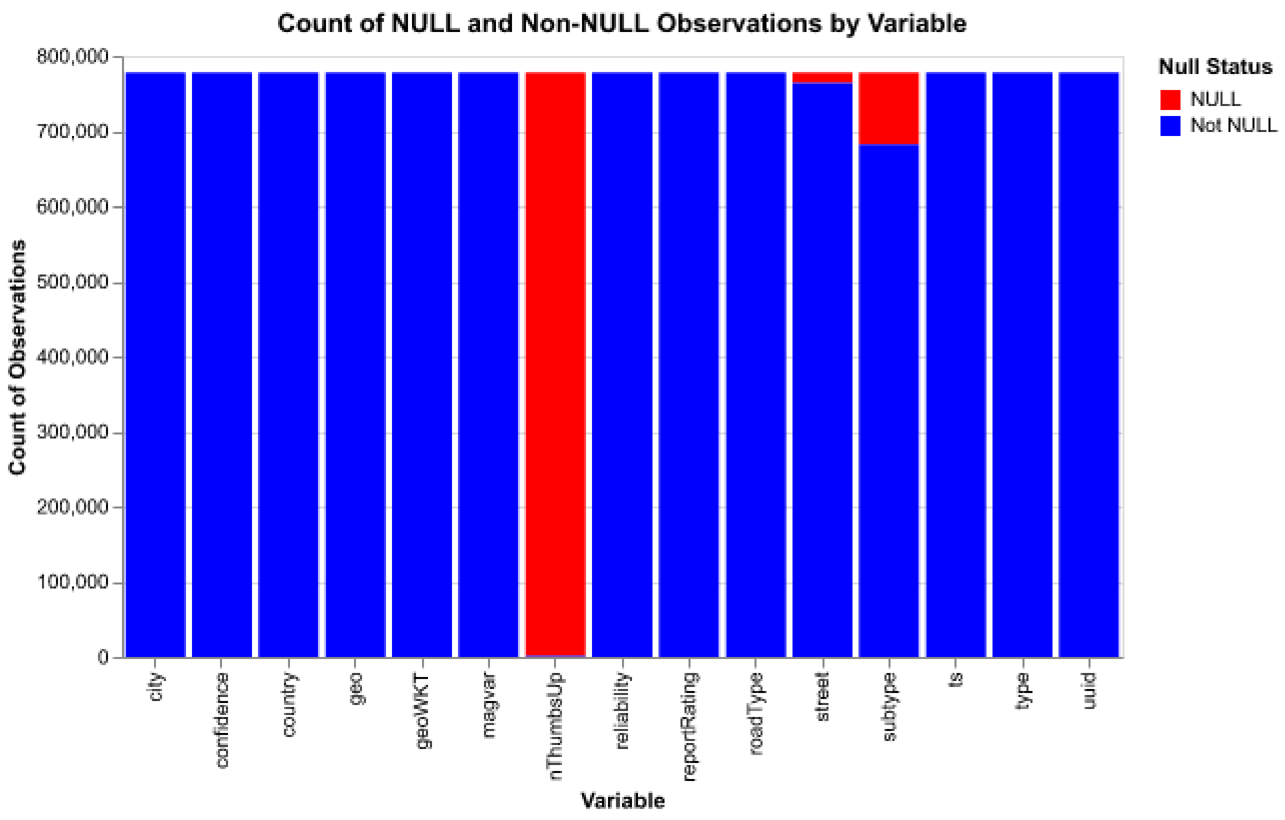
```python
    'nThumbsUp': 'Quantitative',
    'street': 'Nominal',
    'uuid': 'Nominal',
    'country': 'Nominal',
    'type': 'Nominal',
    'subtype': 'Nominal',
    'roadType': 'Quantitative',
    'reliability': 'Quantitative',
    'magvar': 'Quantitative',
    'reportRating': 'Quantitative'
}

markdown_table = "| Variable Name                 | Altair Data Type
↪  |\n"
markdown_table +=
↪  "|-----------------------------|-------------------------|\n"
for variable, dtype in data_types.items():
    markdown_table += f"| {variable:<29} | {dtype:<23} |\n"

print(markdown_table)
#Reference: I don't know how to make the markdown table, chatGPT gives me the
↪  format.
```

```
| Variable Name                 | Altair Data Type        |
|-----------------------------|-------------------------|
| city                          | Nominal                 |
| confidence                    | Quantitative            |
| nThumbsUp                     | Quantitative            |
| street                        | Nominal                 |
| uuid                          | Nominal                 |
| country                       | Nominal                 |
| type                          | Nominal                 |
| subtype                       | Nominal                 |
| roadType                      | Quantitative            |
| reliability                   | Quantitative            |
| magvar                        | Quantitative            |
| reportRating                  | Quantitative            |
```

2.

```python
waze_df = pd.read_csv('waze_data.csv')
```

```python
null_counts = waze_df.isnull().sum()
non_null_counts = waze_df.notnull().sum()

null_data = pd.DataFrame({
    'Variable': null_counts.index,
    'NULL': null_counts.values,
    'Not NULL': non_null_counts.values
})

null_data_melted = null_data.melt(id_vars='Variable',
                                  value_vars=['NULL', 'Not NULL'],
                                  var_name='Null Status',
                                  value_name='Count')

chart = alt.Chart(null_data_melted).mark_bar().encode(
    x=alt.X('Variable:N', title='Variable'),
    y=alt.Y('Count:Q', title='Count of Observations'),
    color=alt.Color('Null Status:N', scale=alt.Scale(domain=['NULL', 'Not
 ↪  NULL'], range=['red', 'blue']),
                    title='Null Status')
).properties(
    width=500,
    height=300,
    title='Count of NULL and Non-NULL Observations by Variable'
)

chart.display()
```



nThumbsUp, street and substype have NULL values, nThumbsUp has the highest share of observations that are missing.

3.

```python
unique_types = waze_df['type'].unique()
unique_subtypes = waze_df['subtype'].unique()

print("Unique values in 'type':", unique_types)
print("Unique values in 'subtype':", unique_subtypes)

type_subtype_combinations = waze_df[['type', 'subtype']].drop_duplicates()
print(type_subtype_combinations)

na_subtypes_count =
↪  type_subtype_combinations[type_subtype_combinations['subtype'].isna()]['type'].nunique()
print(f"Number of types with NA subtypes: {na_subtypes_count}")
```

```
Unique values in 'type': ['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']
Unique values in 'subtype': [nan 'ACCIDENT_MAJOR' 'ACCIDENT_MINOR'
'HAZARD_ON_ROAD'
 'HAZARD_ON_ROAD_CAR_STOPPED' 'HAZARD_ON_ROAD_CONSTRUCTION'
 'HAZARD_ON_ROAD_EMERGENCY_VEHICLE' 'HAZARD_ON_ROAD_ICE'
 'HAZARD_ON_ROAD_OBJECT' 'HAZARD_ON_ROAD_POT_HOLE'
 'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT' 'HAZARD_ON_SHOULDER'
 'HAZARD_ON_SHOULDER_CAR_STOPPED' 'HAZARD_WEATHER' 'HAZARD_WEATHER_FLOOD'
 'JAM_HEAVY_TRAFFIC' 'JAM_MODERATE_TRAFFIC' 'JAM_STAND_STILL_TRAFFIC'
 'ROAD_CLOSED_EVENT' 'HAZARD_ON_ROAD_LANE_CLOSED' 'HAZARD_WEATHER_FOG'
 'ROAD_CLOSED_CONSTRUCTION' 'HAZARD_ON_ROAD_ROAD_KILL'
 'HAZARD_ON_SHOULDER_ANIMALS' 'HAZARD_ON_SHOULDER_MISSING_SIGN'
 'JAM_LIGHT_TRAFFIC' 'HAZARD_WEATHER_HEAVY_SNOW' 'ROAD_CLOSED_HAZARD'
 'HAZARD_WEATHER_HAIL']
               type                             subtype
0               JAM                                 NaN
1          ACCIDENT                                 NaN
2       ROAD_CLOSED                                 NaN
26           HAZARD                                 NaN
122        ACCIDENT                      ACCIDENT_MAJOR
131        ACCIDENT                      ACCIDENT_MINOR
148          HAZARD                      HAZARD_ON_ROAD
190          HAZARD          HAZARD_ON_ROAD_CAR_STOPPED
240          HAZARD         HAZARD_ON_ROAD_CONSTRUCTION
276          HAZARD    HAZARD_ON_ROAD_EMERGENCY_VEHICLE
302          HAZARD                  HAZARD_ON_ROAD_ICE
303          HAZARD               HAZARD_ON_ROAD_OBJECT
355          HAZARD             HAZARD_ON_ROAD_POT_HOLE
478          HAZARD  HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT
483          HAZARD                  HAZARD_ON_SHOULDER
485          HAZARD      HAZARD_ON_SHOULDER_CAR_STOPPED
854          HAZARD                      HAZARD_WEATHER
857          HAZARD                HAZARD_WEATHER_FLOOD
858             JAM                   JAM_HEAVY_TRAFFIC
1122            JAM                JAM_MODERATE_TRAFFIC
1184            JAM             JAM_STAND_STILL_TRAFFIC
1335    ROAD_CLOSED                   ROAD_CLOSED_EVENT
1905         HAZARD          HAZARD_ON_ROAD_LANE_CLOSED
5557         HAZARD                  HAZARD_WEATHER_FOG
7331    ROAD_CLOSED            ROAD_CLOSED_CONSTRUCTION
21443        HAZARD            HAZARD_ON_ROAD_ROAD_KILL
21447        HAZARD          HAZARD_ON_SHOULDER_ANIMALS
21940        HAZARD     HAZARD_ON_SHOULDER_MISSING_SIGN
```

```
38546          JAM                  JAM_LIGHT_TRAFFIC
44216       HAZARD          HAZARD_WEATHER_HEAVY_SNOW
54556   ROAD_CLOSED                 ROAD_CLOSED_HAZARD
229005      HAZARD                 HAZARD_WEATHER_HAIL
Number of types with NA subtypes: 4
```

Hierarchical Structure: (I asked chatGPT how to put the table in a qmd file)

- **JAM**
  - Heavy Traffic
  - Moderate Traffic
  - Standstill Traffic
  - Light Traffic
  - Unclassified
- **ACCIDENT**
  - Major
  - Minor
  - Unclassified
- **ROAD CLOSED**
  - Event
  - Construction
  - Hazard
  - Unclassified
- **HAZARD**
  - On Road
    * Car Stopped
    * Construction
    * Emergency Vehicle
    * Ice
    * Object
    * Pothole
    * Traffic Light Fault
    * Lane Closed
    * Roadkill
  - On Shoulder
    * Car Stopped
    * Animals
    * Missing Sign
  - Weather

        * Flood
        * Fog
        * Heavy Snow
        * Hail
    - Unclassified

I choose to keep NA as it may be beneficial to keep the NA subtypes, as they provide an "Unclassified" option that can still convey useful information even when details are unavailable. It also keep the completeness of the whole data for our future work.

4.
5.

```python
crosswalk_df = pd.DataFrame(columns=['type', 'subtype', 'updated_type',
↪  'updated_subtype', 'updated_subsubtype'])
```

2.

```python
# I asked chatGPT to help me fill in as typing workload is too heavy
crosswalk_data = [
    # JAM hierarchy
    {"type": "JAM", "subtype": "JAM_HEAVY_TRAFFIC", "updated_type": "Jam",
↪   "updated_subtype": "Heavy Traffic", "updated_subsubtype": None},
    {"type": "JAM", "subtype": "JAM_MODERATE_TRAFFIC", "updated_type": "Jam",
↪   "updated_subtype": "Moderate Traffic", "updated_subsubtype": None},
    {"type": "JAM", "subtype": "JAM_STAND_STILL_TRAFFIC", "updated_type":
↪   "Jam", "updated_subtype": "Standstill Traffic", "updated_subsubtype":
↪   None},
    {"type": "JAM", "subtype": "JAM_LIGHT_TRAFFIC", "updated_type": "Jam",
↪   "updated_subtype": "Light Traffic", "updated_subsubtype": None},
    {"type": "JAM", "subtype": None, "updated_type": "Jam",
↪   "updated_subtype": "Unclassified", "updated_subsubtype": None},

    # ACCIDENT hierarchy
    {"type": "ACCIDENT", "subtype": "ACCIDENT_MAJOR", "updated_type":
↪   "Accident", "updated_subtype": "Major", "updated_subsubtype": None},
    {"type": "ACCIDENT", "subtype": "ACCIDENT_MINOR", "updated_type":
↪   "Accident", "updated_subtype": "Minor", "updated_subsubtype": None},
    {"type": "ACCIDENT", "subtype": None, "updated_type": "Accident",
↪   "updated_subtype": "Unclassified", "updated_subsubtype": None},

    # ROAD CLOSED hierarchy
    {"type": "ROAD_CLOSED", "subtype": "ROAD_CLOSED_EVENT", "updated_type":
↪   "Road Closed", "updated_subtype": "Event", "updated_subsubtype": None},
    {"type": "ROAD_CLOSED", "subtype": "ROAD_CLOSED_CONSTRUCTION",
↪   "updated_type": "Road Closed", "updated_subtype": "Construction",
↪   "updated_subsubtype": None},
    {"type": "ROAD_CLOSED", "subtype": "ROAD_CLOSED_HAZARD", "updated_type":
↪   "Road Closed", "updated_subtype": "Hazard", "updated_subsubtype": None},
    {"type": "ROAD_CLOSED", "subtype": None, "updated_type": "Road Closed",
↪   "updated_subtype": "Unclassified", "updated_subsubtype": None},

    # HAZARD hierarchy
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD", "updated_type": "Hazard",
↪   "updated_subtype": "On Road", "updated_subsubtype": None},
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_CAR_STOPPED",
↪   "updated_type": "Hazard", "updated_subtype": "On Road",
↪   "updated_subsubtype": "Car Stopped"},
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_CONSTRUCTION",
↪   "updated_type": "Hazard", "updated_subtype": "On Road",
↪   "updated_subsubtype": "Construction"},
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_EMERGENCY_VEHICLE",
↪   "updated_type": "Hazard", "updated_subtype": "On Road",
↪   "updated_subsubtype": "Emergency Vehicle"},
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_ICE", "updated_type":
↪   "Hazard", "updated_subtype": "On Road", "updated_subsubtype": "Ice"},
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_OBJECT", "updated_type":
↪   "Hazard", "updated_subtype": "On Road", "updated_subsubtype": "Object"},
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_POT_HOLE", "updated_type":
↪   "Hazard", "updated_subtype": "On Road", "updated_subsubtype": "Pothole"},
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT",
↪   "updated_type": "Hazard", "updated_subtype": "On Road",
↪   "updated_subsubtype": "Traffic Light Fault"},
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_LANE_CLOSED",
↪   "updated_type": "Hazard", "updated_subtype": "On Road",
↪   "updated_subsubtype": "Lane Closed"},
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_ROAD_KILL", "updated_type":
↪   "Hazard", "updated_subtype": "On Road", "updated_subsubtype":
↪   "Roadkill"},
    {"type": "HAZARD", "subtype": "HAZARD_ON_SHOULDER", "updated_type":
↪   "Hazard", "updated_subtype": "On Shoulder", "updated_subsubtype": None},
    {"type": "HAZARD", "subtype": "HAZARD_ON_SHOULDER_CAR_STOPPED",
↪   "updated_type": "Hazard", "updated_subtype": "On Shoulder",
↪   "updated_subsubtype": "Car Stopped"},
    {"type": "HAZARD", "subtype": "HAZARD_ON_SHOULDER_ANIMALS",
 ↪  "updated_type": "Hazard", "updated_subtype": "On Shoulder",
 ↪  "updated_subsubtype": "Animals"},
    {"type": "HAZARD", "subtype": "HAZARD_ON_SHOULDER_MISSING_SIGN",
 ↪  "updated_type": "Hazard", "updated_subtype": "On Shoulder",
 ↪  "updated_subsubtype": "Missing Sign"},
    {"type": "HAZARD", "subtype": "HAZARD_WEATHER", "updated_type": "Hazard",
 ↪  "updated_subtype": "Weather", "updated_subsubtype": None},
    {"type": "HAZARD", "subtype": "HAZARD_WEATHER_FLOOD", "updated_type":
 ↪  "Hazard", "updated_subtype": "Weather", "updated_subsubtype": "Flood"},
    {"type": "HAZARD", "subtype": "HAZARD_WEATHER_FOG", "updated_type":
 ↪  "Hazard", "updated_subtype": "Weather", "updated_subsubtype": "Fog"},
    {"type": "HAZARD", "subtype": "HAZARD_WEATHER_HEAVY_SNOW",
 ↪  "updated_type": "Hazard", "updated_subtype": "Weather",
 ↪  "updated_subsubtype": "Heavy Snow"},
    {"type": "HAZARD", "subtype": "HAZARD_WEATHER_HAIL", "updated_type":
 ↪  "Hazard", "updated_subtype": "Weather", "updated_subsubtype": "Hail"},
    {"type": "HAZARD", "subtype": None, "updated_type": "Hazard",
 ↪  "updated_subtype": "Unclassified", "updated_subsubtype": None}
]

crosswalk_df = pd.DataFrame(crosswalk_data)
```

3.

```
merged_df = waze_df.merge(crosswalk_df, on=['type', 'subtype'], how='left')

accident_unclassified_count = merged_df[(merged_df['updated_type'] ==
 ↪  'Accident') &
                                        (merged_df['updated_subtype'] ==
 ↪  'Unclassified')].shape[0]

print(f"Number of rows for Accident - Unclassified:
 ↪  {accident_unclassified_count}")
```

```
Number of rows for Accident - Unclassified: 24359
```

4.

```python
crosswalk_types_subtypes = crosswalk_df[['type',
 ↪  'subtype']].drop_duplicates()
merged_types_subtypes = merged_df[['type', 'subtype']].drop_duplicates()

mismatch_rows = pd.merge(crosswalk_types_subtypes, merged_types_subtypes,
 ↪  how='outer', indicator=True).query('_merge != "both"')

print(f"Number of mismatched rows: {len(mismatch_rows)}")
```

```
Number of mismatched rows: 0
```

The crosswalk and the new merged dataset have the same values in type and subtype.

App #1: Top Location by Alert Type Dashboard (30 points)

1.

a.

```python
# Used chatGPT

# Extract longitude and latitude using regex
merged_df['longitude'] = merged_df['geo'].str.extract(r'POINT\(([-\d.]+)
 ↪  [-\d.]+\)').astype(float)

merged_df['latitude'] = merged_df['geo'].str.extract(r'POINT\([-\d.]+
 ↪  ([-\d.]+)\)').astype(float)
```

b.

```python
merged_df['latitude_bin'] = merged_df['latitude'].round(2)
merged_df['longitude_bin'] = merged_df['longitude'].round(2)

bin_counts = merged_df.groupby(['latitude_bin',
 ↪  'longitude_bin']).size().reset_index(name='count')

top_bin = bin_counts.loc[bin_counts['count'].idxmax()]

print(f"The binned latitude-longitude combination with the greatest number of
 ↪  observations is: {top_bin[['latitude_bin', 'longitude_bin']].values} with
 ↪  {top_bin['count']} observations.")
```

```
The binned latitude-longitude combination with the greatest number of
observations is: [ 41.88 -87.65] with 21325.0 observations.
```

c.

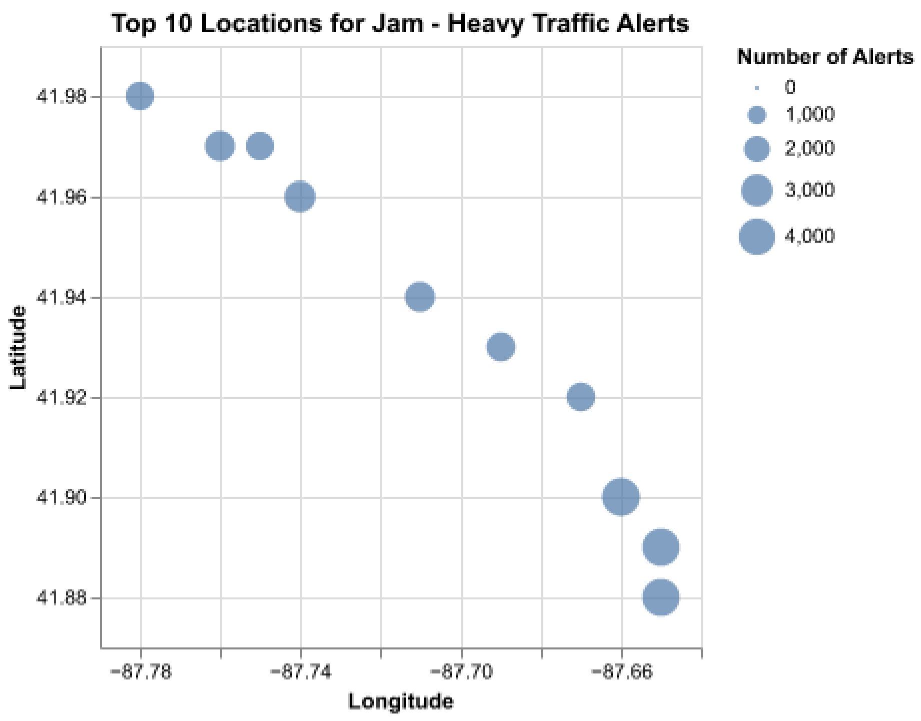
```
collapsed_df = (
    merged_df.groupby(['latitude_bin', 'longitude_bin',
                      'updated_type', 'updated_subtype'])
    .size()
    .reset_index(name='alert_count')
)

top_alerts = (
    collapsed_df.sort_values('alert_count', ascending=False)
)

output_folder = "top_alerts_map"
output_path = os.path.join(output_folder, "top_alerts_map.csv")
top_alerts.to_csv(output_path, index=False)

print(f"Data saved to {output_path}")
print(f"Number of rows in the aggregated DataFrame: {len(top_alerts)}")
```

```
Data saved to top_alerts_map/top_alerts_map.csv
Number of rows in the aggregated DataFrame: 6675
```

The level of aggregation is latitude-longitude bins, grouped by type and subtype. Each unique combination of binned latitude, longitude, type, and subtype represents one aggregated row.

2.

```
jam_heavy_df = merged_df[
    (merged_df['updated_type'] == 'Jam') & (merged_df['updated_subtype'] ==
 ↪  'Heavy Traffic')
]

collapsed_jam_heavy_df = (
    jam_heavy_df.groupby(['latitude_bin', 'longitude_bin'])
    .size()
    .reset_index(name='alert_count')
)

top_10_jam_heavy = (
    collapsed_jam_heavy_df.sort_values('alert_count', ascending=False)
    .head(10)
)

x_min, x_max = top_10_jam_heavy['longitude_bin'].min() - 0.01,
↪   top_10_jam_heavy['longitude_bin'].max() + 0.01
y_min, y_max = top_10_jam_heavy['latitude_bin'].min() - 0.01,
↪   top_10_jam_heavy['latitude_bin'].max() + 0.01

scatter_plot = (
    alt.Chart(top_10_jam_heavy)
    .mark_circle()
    .encode(
        x=alt.X('longitude_bin:Q', title='Longitude',
↪   scale=alt.Scale(domain=[x_min, x_max])),
        y=alt.Y('latitude_bin:Q', title='Latitude',
↪   scale=alt.Scale(domain=[y_min, y_max])),
        size=alt.Size('alert_count:Q', title='Number of Alerts'),
        tooltip=['latitude_bin', 'longitude_bin', 'alert_count']
    )
    .properties(
        title='Top 10 Locations for Jam - Heavy Traffic Alerts'
    )
)

scatter_plot
```



3.

a.

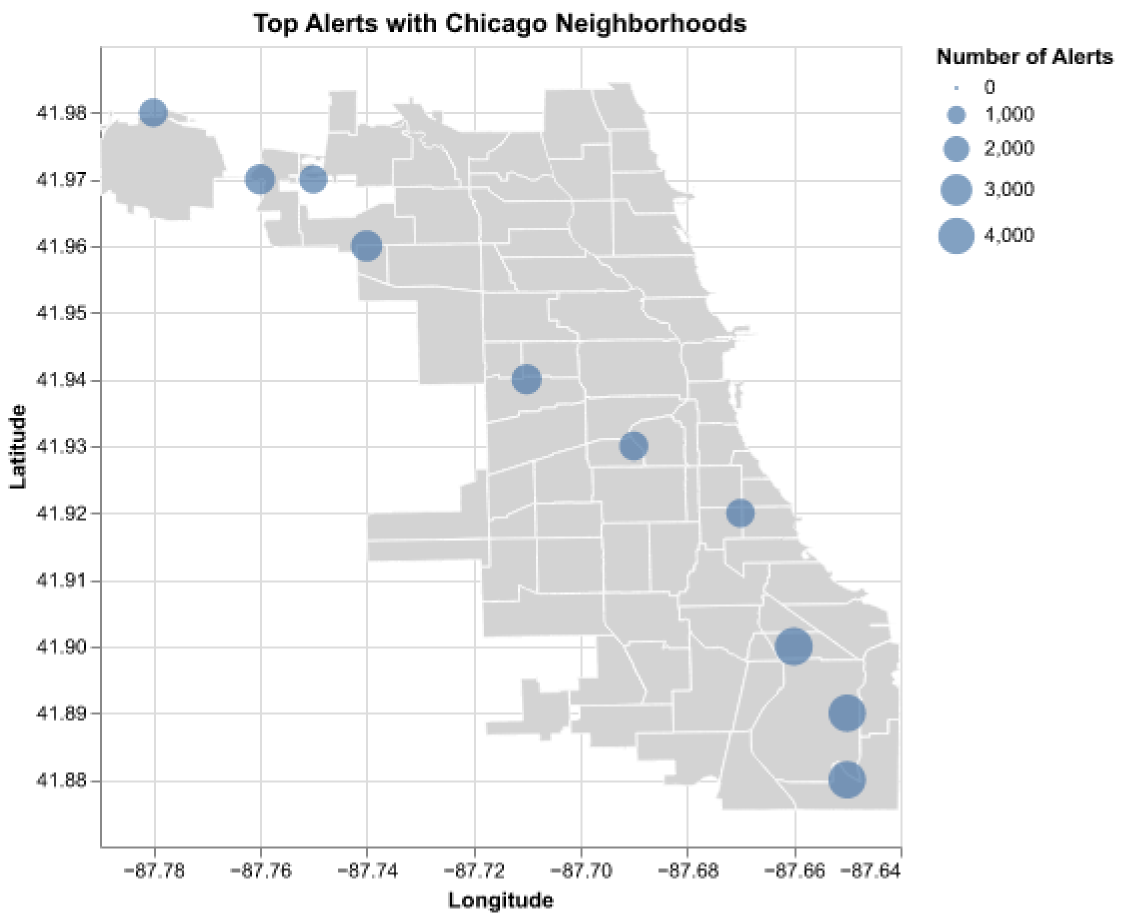
```
import requests

url =
 ↪  'https://data.cityofchicago.org/api/geospatial/igwz-8jzy?method=export&format=GeoJSON'
response = requests.get(url)

with open('chicago_neighborhoods.geojson', 'wb') as file:
    file.write(response.content)
```

b.

```
from shapely.geometry import shape

file_path =
 ↪  "/Users/benzhang/Documents/GitHub/PS6_Ben/chicago_neighborhoods.geojson"

with open(file_path) as f:
    chicago_geojson = json.load(f)

geo_data = alt.Data(values=chicago_geojson["features"])
```

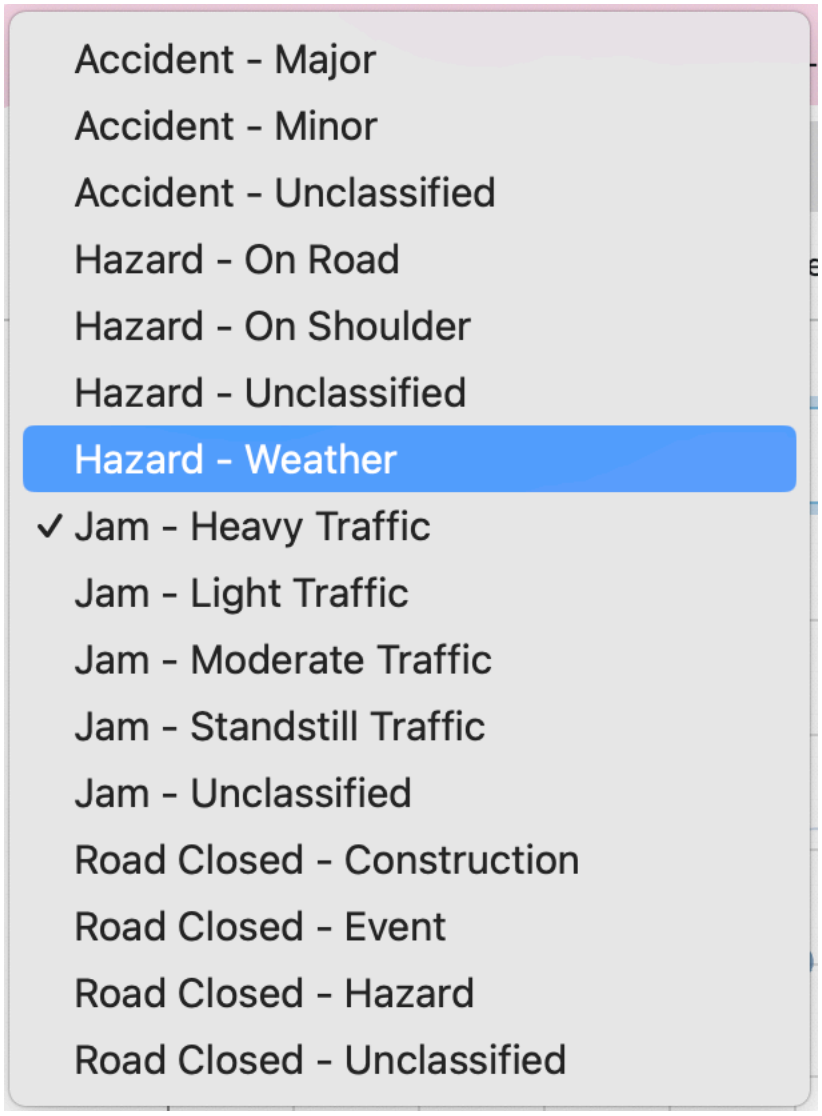
4.

```
map_layer = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).encode(
    tooltip=[
        alt.Tooltip('properties.neighborhood:N', title='Neighborhood')
    ]
).project(type='identity', reflectY=True) # got from Professor Gannong

combined_plot = map_layer + scatter_plot

combined_plot.properties(
    title="Top Alerts with Chicago Neighborhoods",
    height=400,
    width=400
)
```


Top Alerts with Chicago Neighborhoods
Number of Alerts
0
1,000
2,000
3,000
4,000
Latitude
41.98
41.97
41.96
41.95
41.94
41.93
41.92
41.91
41.90
41.89
41.88
Longitude
-87.78
-87.76
-87.74
-87.72
-87.70
-87.68
-87.66
-87.64

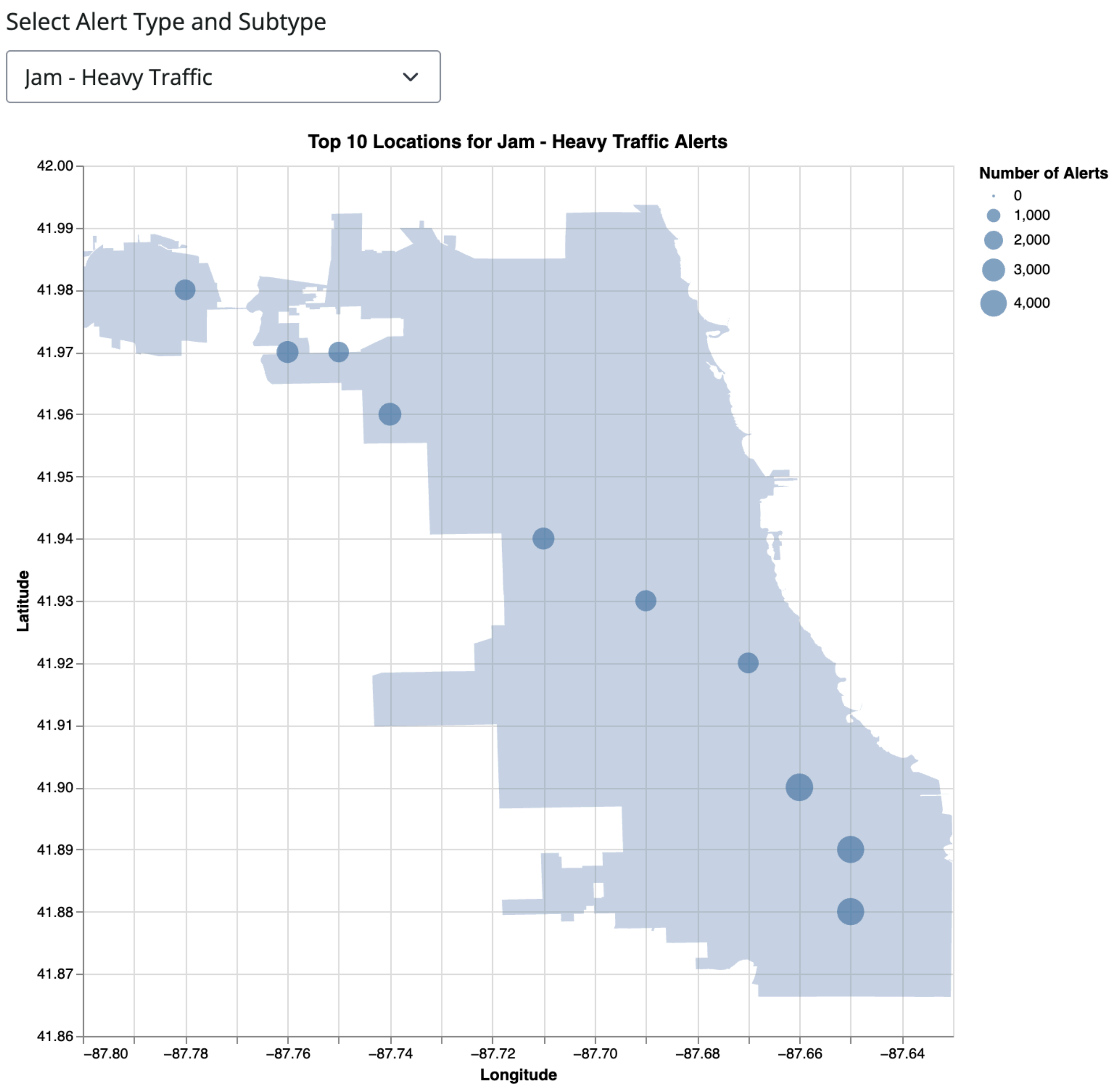
5.

a.



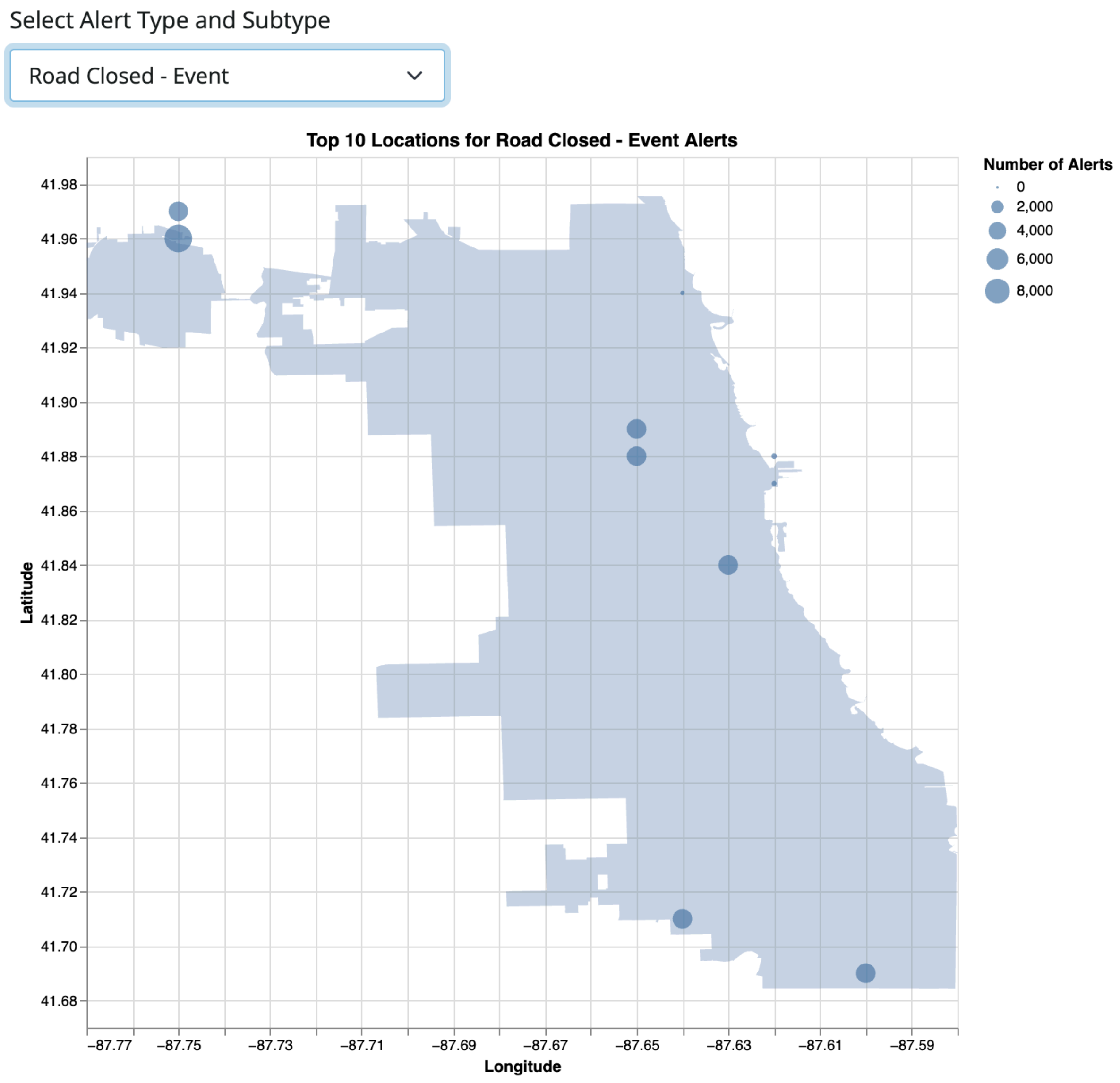
There are 16 type x subtype combinations in my dropdown menu.

b.


Select Alert Type and Subtype

Jam - Heavy Traffic

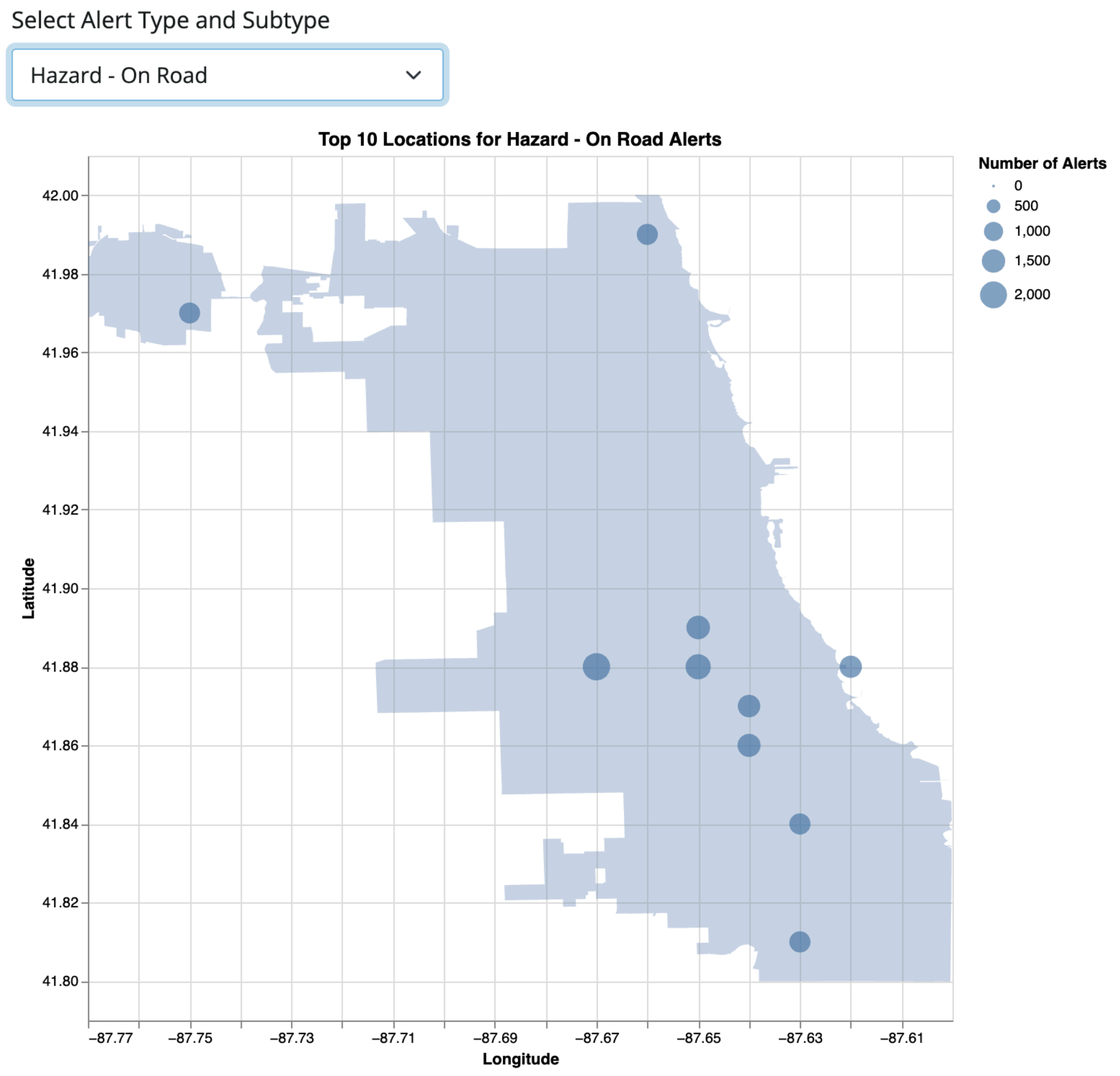
c.


Select Alert Type and Subtype

Road Closed - Event

d.

Where are "Hazard - On Road" alerts most common in Chicago?


Select Alert Type and Subtype

Hazard - On Road

e. We could add the column Time_of_Day Adding a column indicating the time of day (e.g., Morning, Afternoon, Evening, Night) for each alert could enhance the analysis by allowing users to identify temporal patterns in alerts. For example, we can see when road closures are more common during specific times of the day, such as rush hours or late evenings. This information would enable better decision-making, such as scheduling travel.

App #2: Top Location by Alert Type and Hour Dashboard (20 points)

1.

a. Collapsing the dataset by the ts column directly is generally not a good idea in this case. Timestamps in the ts column have a very high level of precise, down to seconds. If we group or collapse by the ts column, each unique timestamp could become a separate group, leading to an unnecessarily large and fragmented dataset. The purpose of this analysis is to identify trends, such as the most common alert locations by hour. Grouping by ts directly would not allow for meaningful aggregations across time periods like hours or other intervals. By collapsing the data by broader time intervals, like hourly, we still retain sufficient detail to answer relevant questions, such as identifying the top locations for alerts at specific times of day.

b.

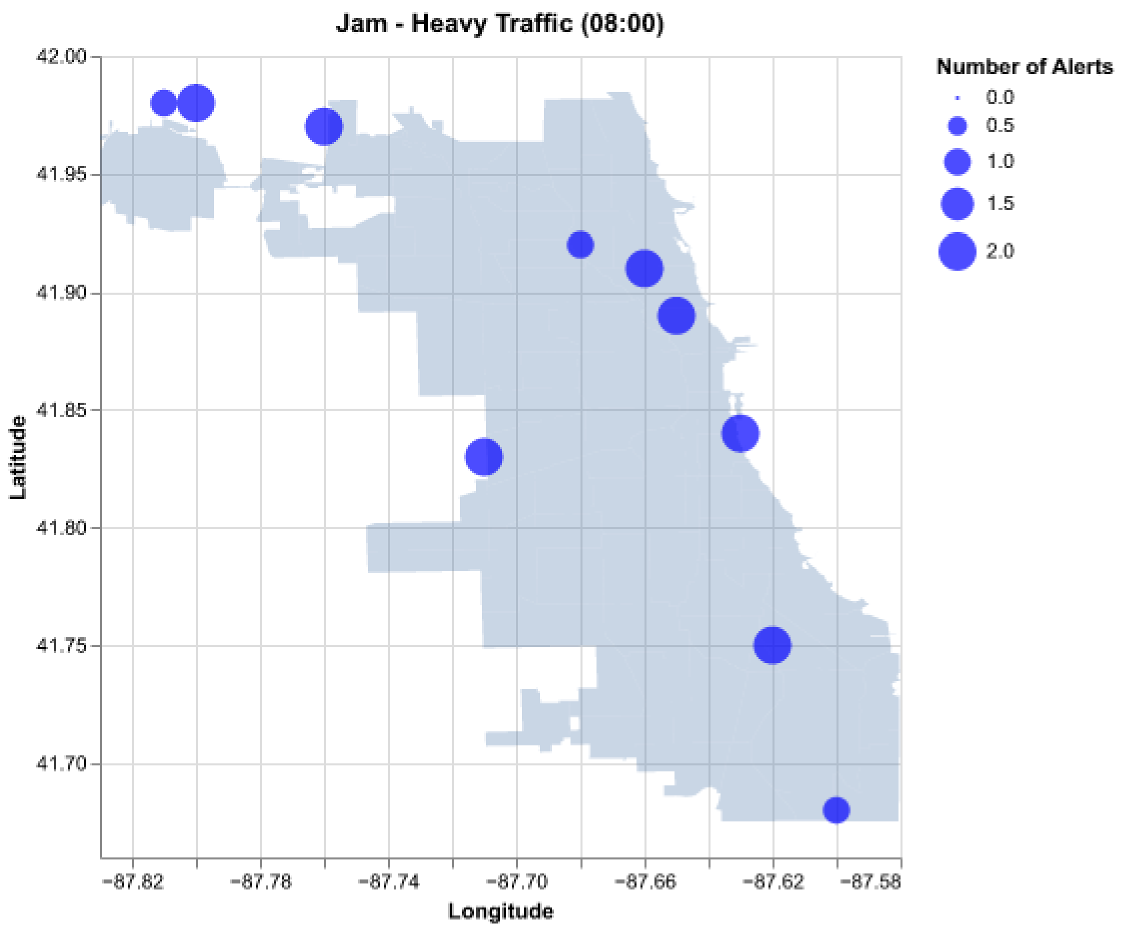
```
merged_df['ts'] = pd.to_datetime(merged_df['ts'], utc=True)

merged_df['hour'] = merged_df['ts'].dt.strftime('%H:00')
#chatGPT shows me how to change the hour format
collapsed_df_hour = (
    merged_df.groupby(
        ['hour', 'updated_type', 'updated_subtype', 'latitude_bin',
 ↪  'longitude_bin'])
    .size()
    .reset_index(name='alert_count')
)

output_path = "top_alerts_map_byhour/top_alerts_map_byhour.csv"
collapsed_df_hour.to_csv(output_path, index=False)

print(f"The collapsed dataset has {collapsed_df_hour.shape[0]} rows.")
```

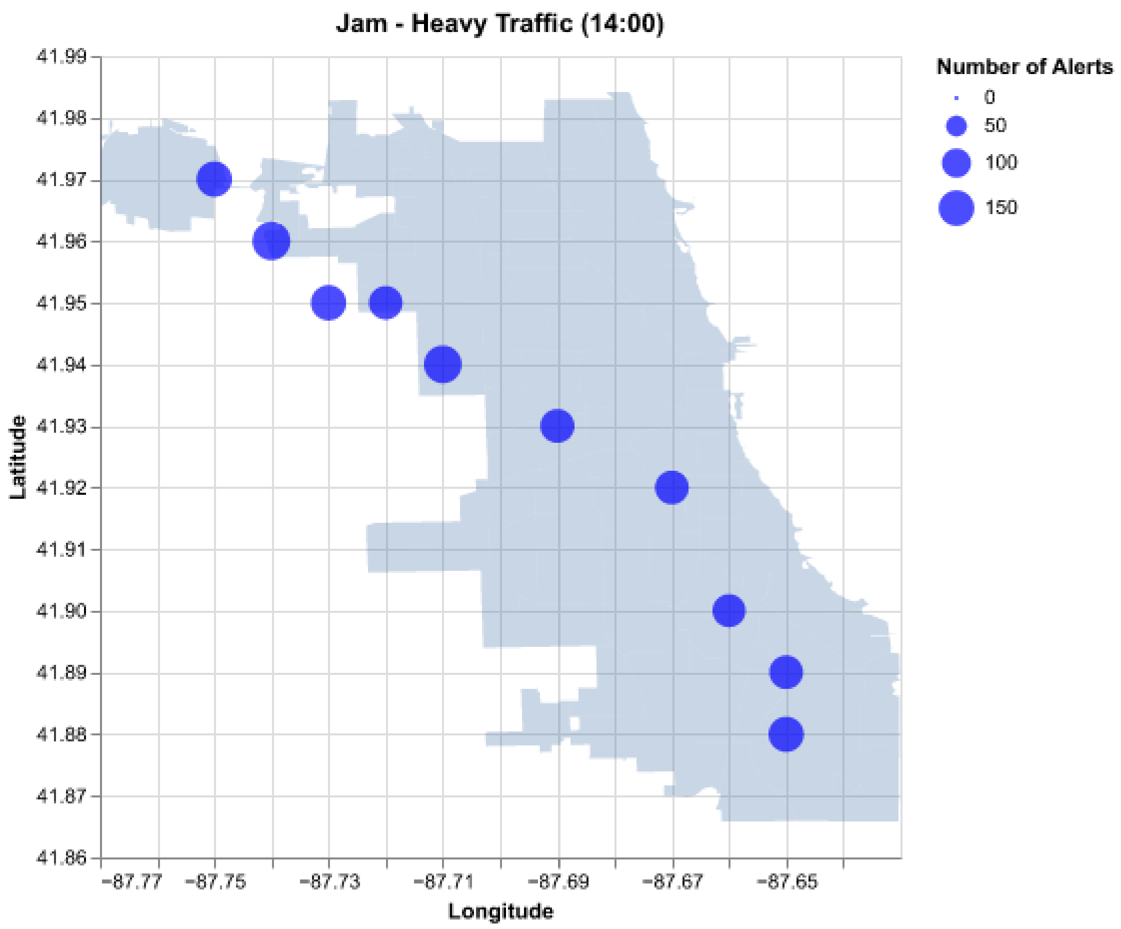
```
The collapsed dataset has 62825 rows.
```

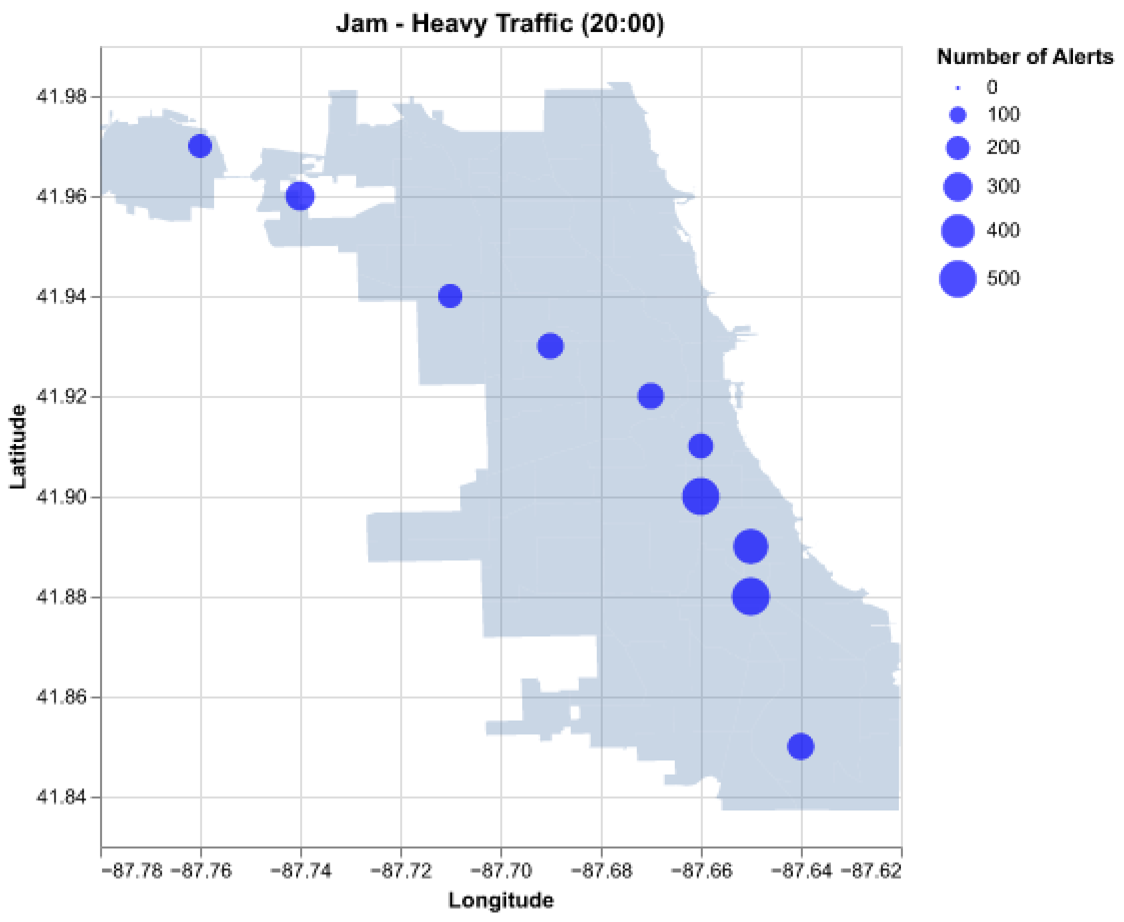
c.

```
collapsed_df_hour["type_subtype"] = collapsed_df_hour["updated_type"] + \
    " - " + collapsed_df_hour["updated_subtype"]

hours_to_plot = ['08:00', '14:00', '20:00']
```

```
selected_type = "Jam"
selected_subtype = "Heavy Traffic"

for hour in hours_to_plot:

    filtered_df_hour = collapsed_df_hour[
        (collapsed_df_hour['updated_type'] == selected_type) &
        (collapsed_df_hour['updated_subtype'] == selected_subtype) &
        (collapsed_df_hour['hour'] == hour)
    ]

    aggregated_alerts = filtered_df_hour.groupby(['latitude_bin',
 ↪  'longitude_bin'])[
        'alert_count'].sum().reset_index()
    top_alerts = aggregated_alerts.sort_values(
        'alert_count', ascending=False).head(10)

    if top_alerts.empty:
        plot = alt.Chart().mark_text(
            text=f"No data available for {hour}.",
            align='center',
            baseline='middle',
            size=15
        ).properties(width=400, height=300)
    else:

        lat_min, lat_max = top_alerts['latitude_bin'].min(
        ) - 0.02, top_alerts['latitude_bin'].max() + 0.02
        long_min, long_max = top_alerts['longitude_bin'].min(
        ) - 0.02, top_alerts['longitude_bin'].max() + 0.02

        scatter_plot = alt.Chart(top_alerts).mark_circle(size=80,
 ↪  color='blue', opacity=0.7).encode(
            x=alt.X('longitude_bin:Q', title='Longitude',
                    scale=alt.Scale(domain=[long_min, long_max])),
            y=alt.Y('latitude_bin:Q', title='Latitude',
                    scale=alt.Scale(domain=[lat_min, lat_max])),
            size=alt.Size('alert_count:Q', title='Number of Alerts'),
            tooltip=['latitude_bin', 'longitude_bin', 'alert_count']
        ).properties(
            title=f'{selected_type} - {selected_subtype} ({hour})',
            width=400,
            height=400
        )

        map_layer = alt.Chart(geo_data).mark_geoshape(
            fillOpacity=0.3,
            stroke=None
        ).encode(
            tooltip=["properties.neighborhood:N"]
        ).project(
            type="identity", reflectY=True
        ).properties(
            width=400,
            height=400
        )

        plot = map_layer + scatter_plot

    plot.show()
```

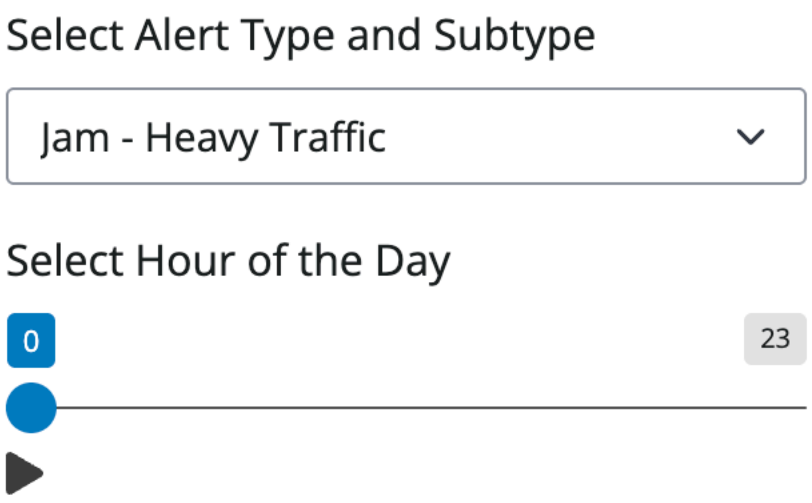

Jam - Heavy Traffic (08:00)
Number of Alerts
0.0
0.5
1.0
1.5
2.0
Latitude
42.00
41.95
41.90
41.85
41.80
41.75
41.70
Longitude
-87.82
-87.78
-87.74
-87.70
-87.66
-87.62
-87.58


Jam - Heavy Traffic (14:00)
Number of Alerts
0
50
100
150
Latitude
41.99
41.98
41.97
41.96
41.95
41.94
41.93
41.92
41.91
41.90
41.89
41.88
41.87
41.86
Longitude
−87.77
−87.75
−87.73
−87.71
−87.69
−87.67
−87.65


Jam - Heavy Traffic (20:00)
Number of Alerts
0
100
200
300
400
500
Latitude
41.98
41.96
41.94
41.92
41.90
41.88
41.86
41.84
Longitude
-87.78
-87.76
-87.74
-87.72
-87.70
-87.68
-87.66
-87.64
-87.62

2.

a.


Select Alert Type and Subtype
Jam - Heavy Traffic
Select Hour of the Day
0
23

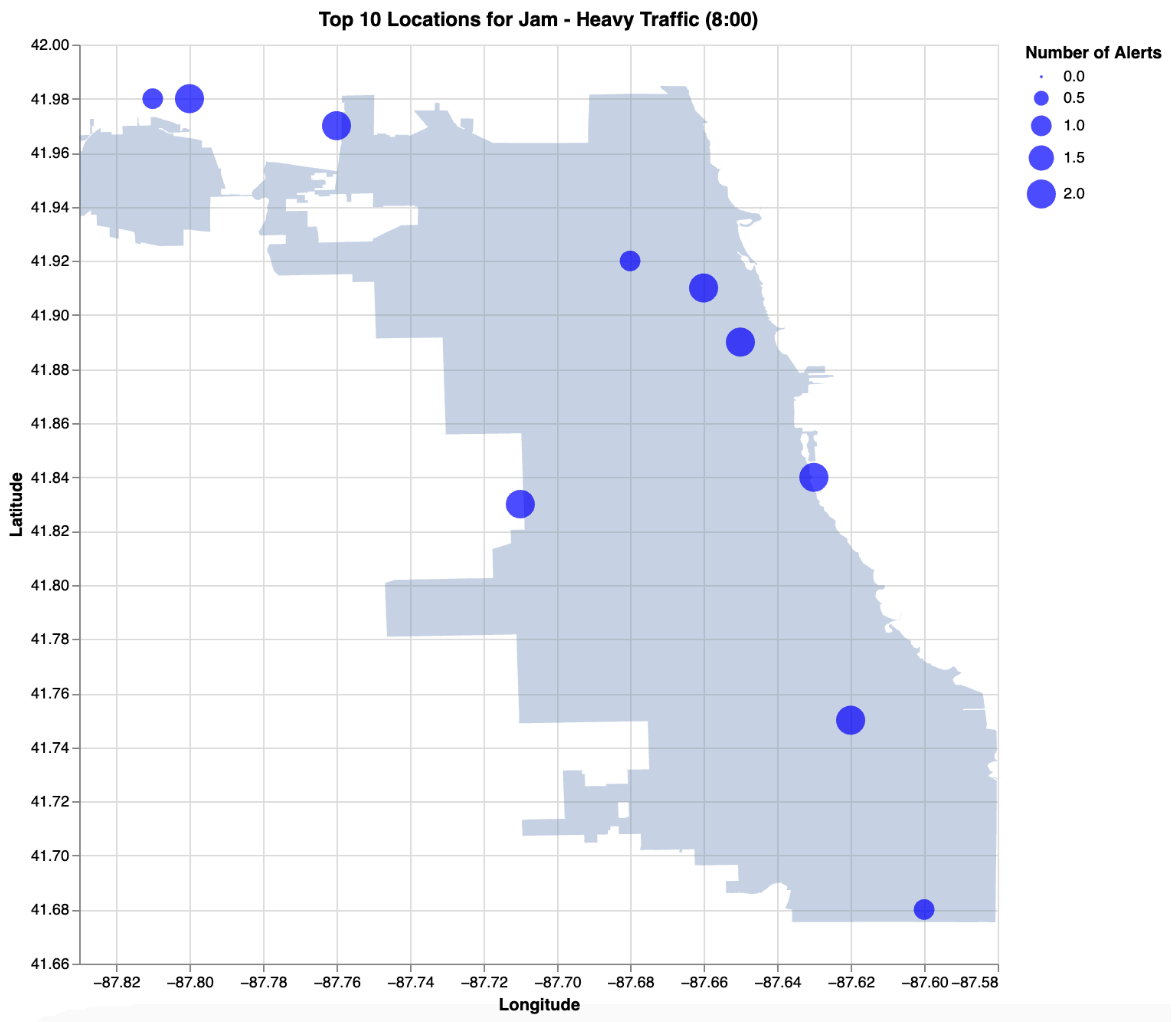
b.

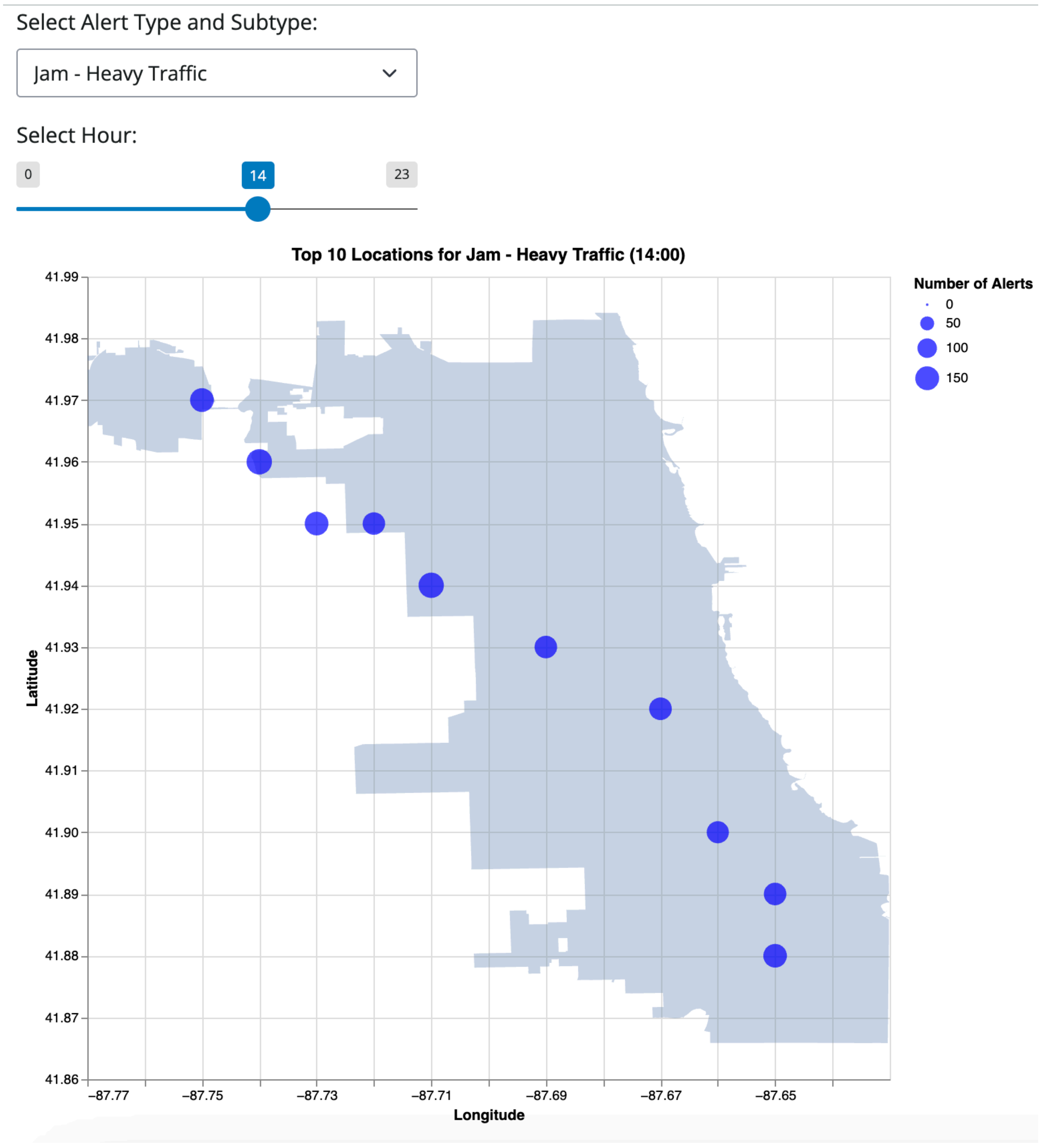
Select Alert Type and Subtype:

Jam - Heavy Traffic

Select Hour:

0 8 23



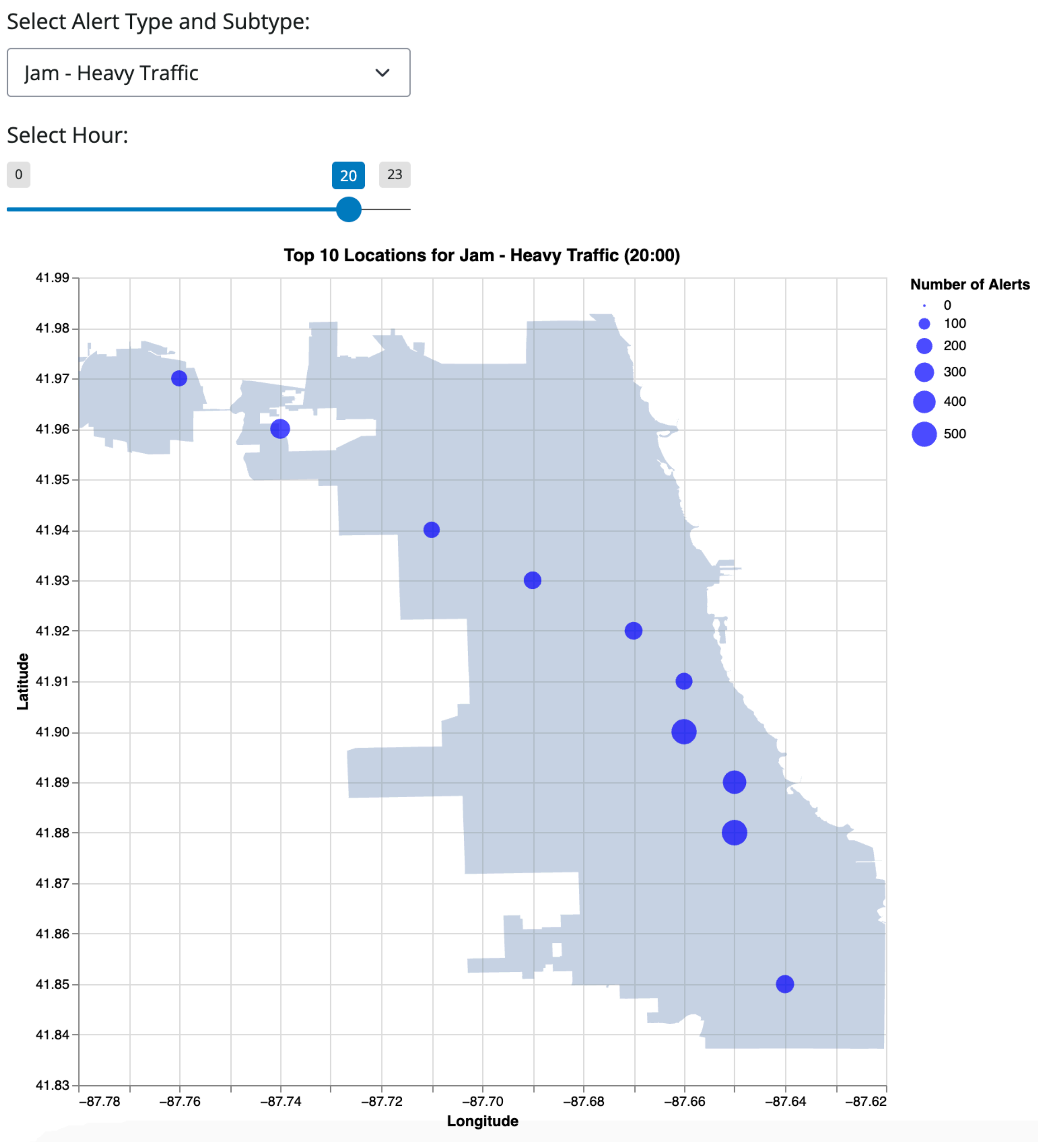


Select Alert Type and Subtype:

Jam - Heavy Traffic

Select Hour:

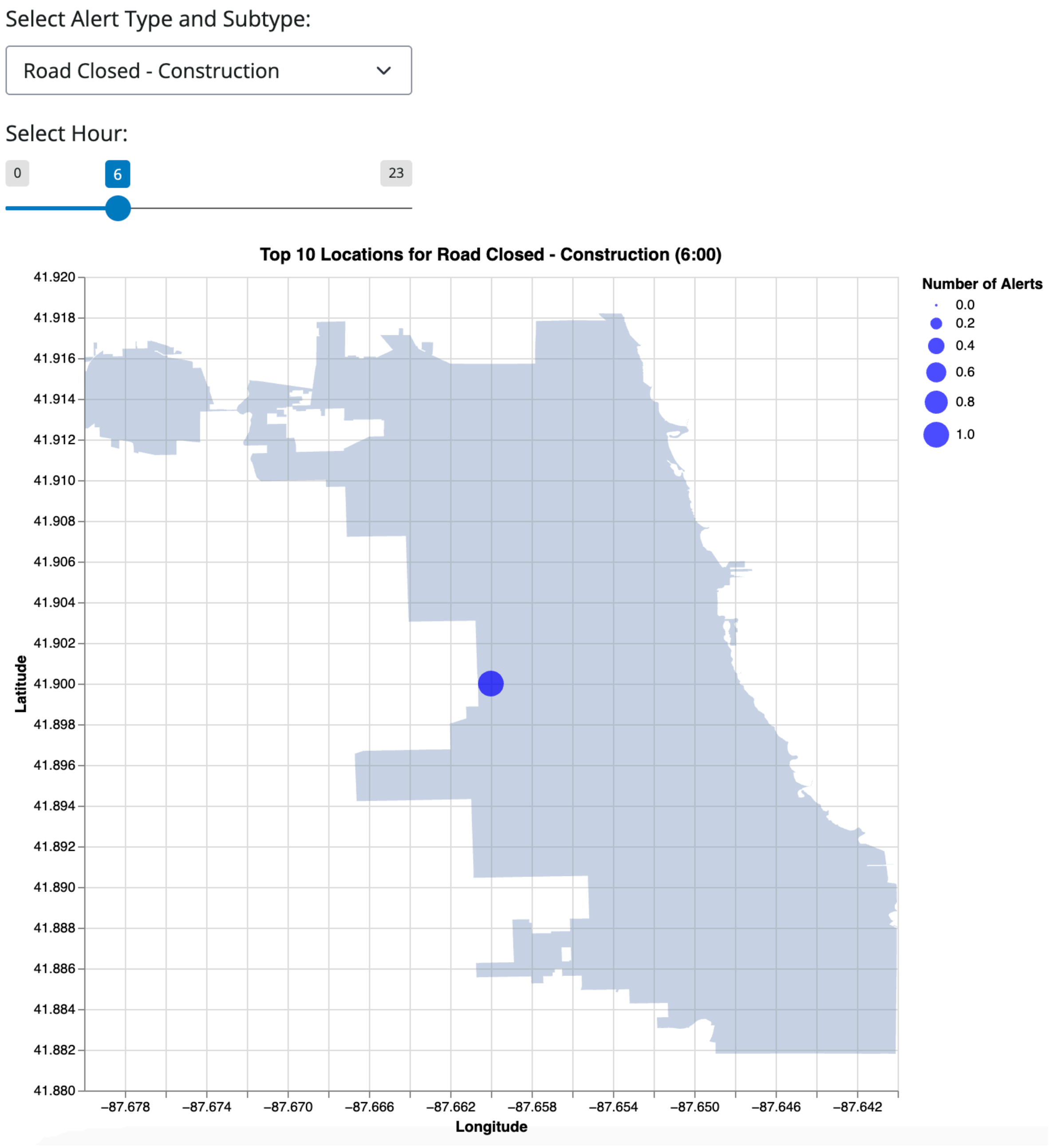
0 14 23


Select Alert Type and Subtype:

Jam - Heavy Traffic

Select Hour:

c.

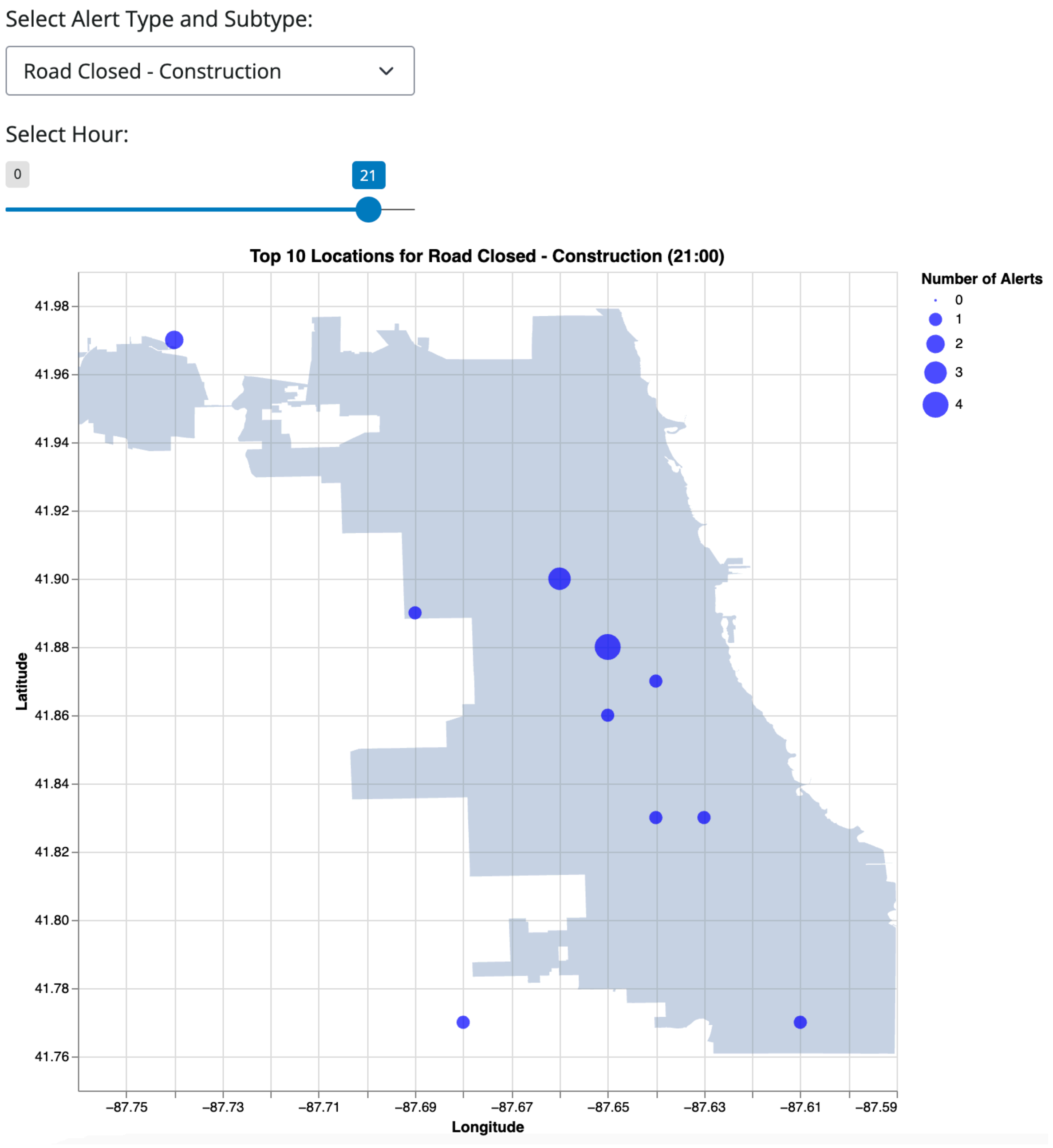


Select Alert Type and Subtype:

Road Closed - Construction

Select Hour:

0 6 23


Select Alert Type and Subtype:

Road Closed - Construction

Select Hour:

It seems like road construction is done more during night hours than morning hours. And when I select the morning times in Shiny App, lots of time has missing data.

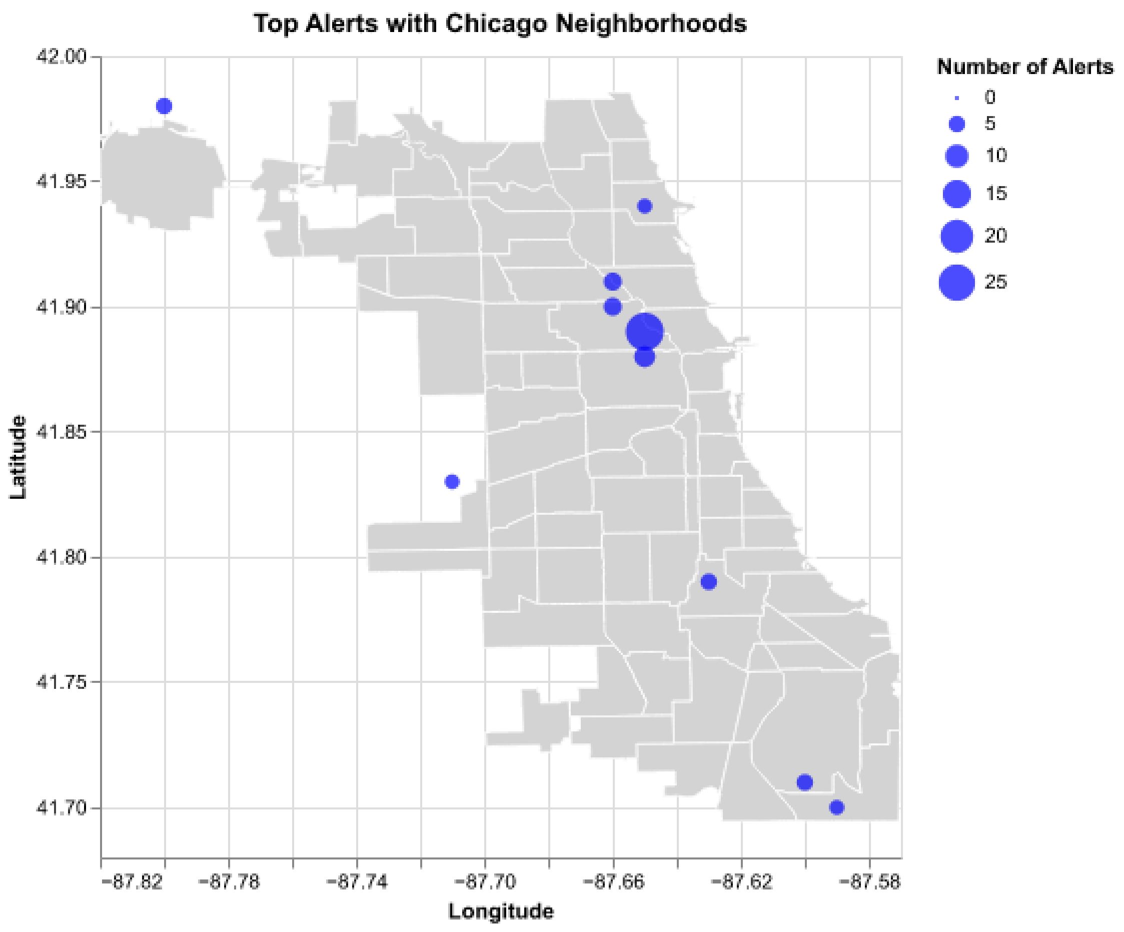
App #3: Top Location by Alert Type and Hour Dashboard (20 points)

1.

a. By aggregating the data over a specific time range, we can observe patterns that span multiple hours. For instance, during heavy traffic hours, there may be consistent high alert counts across the entire 6AM-9AM range, which would be clearer when aggregated. Rather than having to plot 24 different hours, collapsing the data into predefined ranges (like morning, afternoon, evening) could simplify the user experience and make the dashboard less overwhelming.

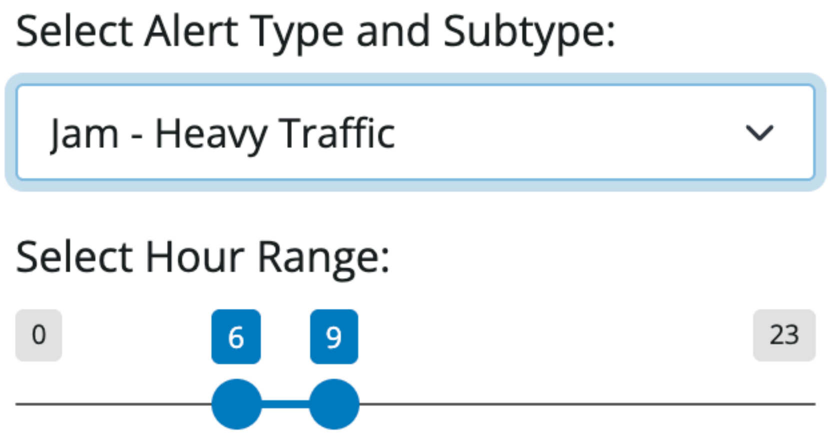
b.

```
collapsed_df_hour['hour'] = collapsed_df_hour['hour'].astype(str)

filtered_df_hour_range = collapsed_df_hour[
    (collapsed_df_hour['updated_type'] == 'Jam') &
    (collapsed_df_hour['updated_subtype'] == 'Heavy Traffic') &
    (collapsed_df_hour['hour'].apply(lambda x: 6 <= int(x.split(':')[0]) <
 ↪  9))
]

aggregated_alerts = filtered_df_hour_range.groupby(
    ['latitude_bin', 'longitude_bin'])['alert_count'].sum().reset_index()

top_alerts = aggregated_alerts.sort_values(
    'alert_count', ascending=False).head(10)

lat_min, lat_max = top_alerts['latitude_bin'].min(
) - 0.02, top_alerts['latitude_bin'].max() + 0.02
long_min, long_max = top_alerts['longitude_bin'].min(
) - 0.02, top_alerts['longitude_bin'].max() + 0.02

scatter_plot = alt.Chart(top_alerts).mark_circle(size=80, color='blue',
 ↪  opacity=0.7).encode(
    x=alt.X('longitude_bin:Q', title='Longitude',
            scale=alt.Scale(domain=[long_min, long_max])),
    y=alt.Y('latitude_bin:Q', title='Latitude',
            scale=alt.Scale(domain=[lat_min, lat_max])),
    size=alt.Size('alert_count:Q', title='Number of Alerts'),
    tooltip=['latitude_bin', 'longitude_bin', 'alert_count']
).properties(
    title="Top 10 Locations for Jam - Heavy Traffic Alerts (6:00AM - 9:00AM)"
)

map_layer = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).encode(
    tooltip=[
        alt.Tooltip('properties.neighborhood:N', title='Neighborhood')
    ]
).project(type='identity', reflectY=True)

combined_plot = map_layer + scatter_plot

combined_plot.properties(
    title="Top Alerts with Chicago Neighborhoods",
    height=400,
    width=400
)
```



2.

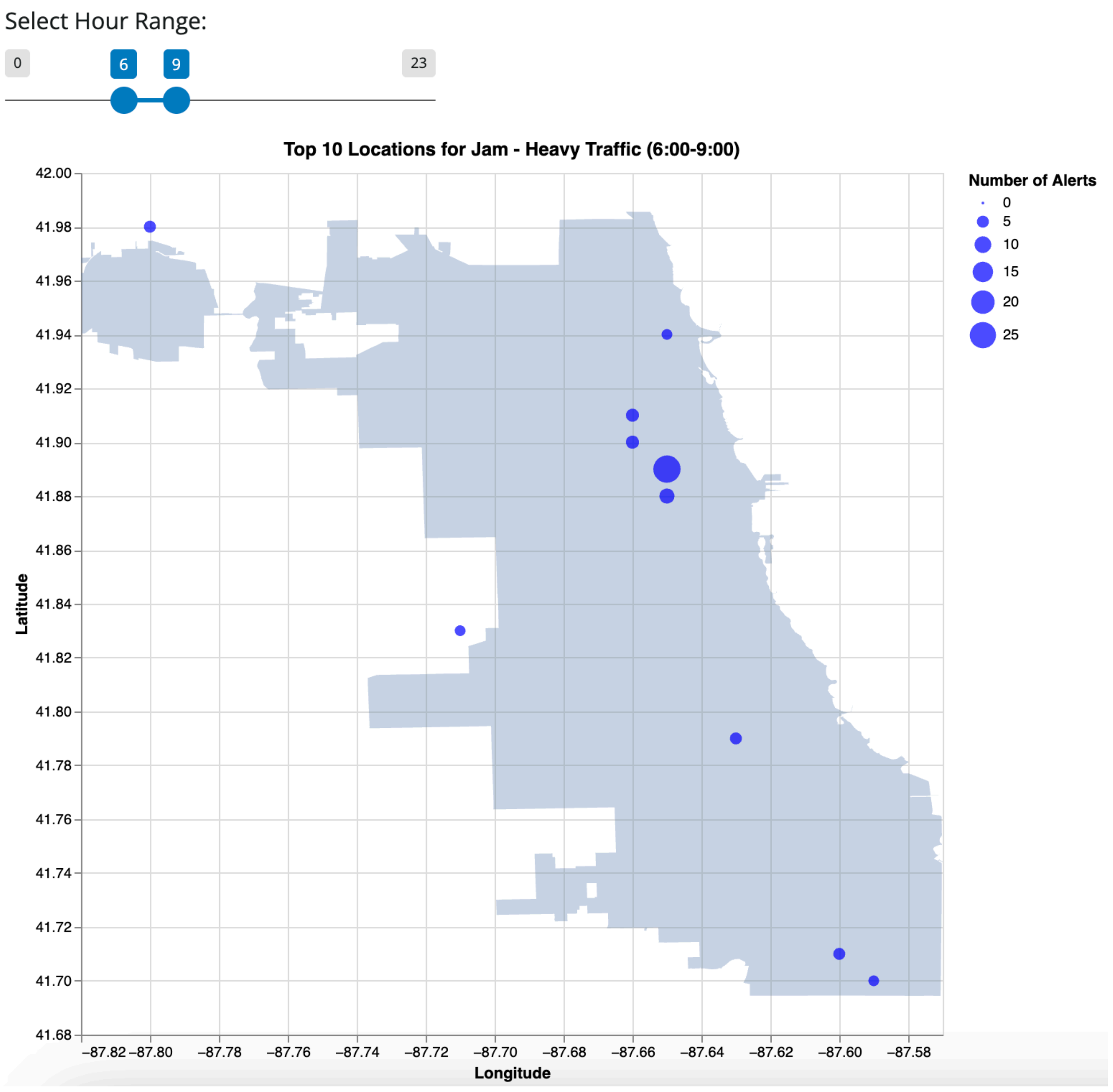
a.


Select Alert Type and Subtype:

Jam - Heavy Traffic

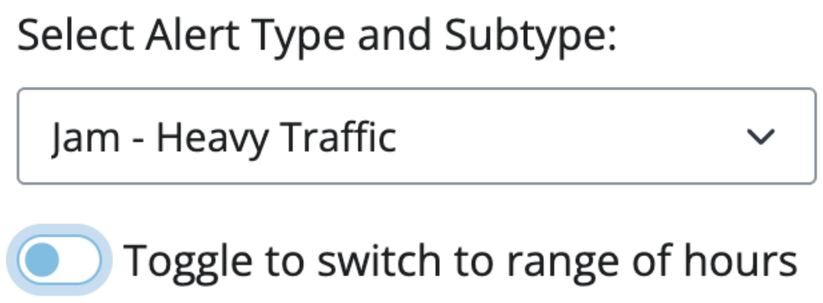
Select Hour Range:

b.


Select Hour Range:
0
6
9
23
Top 10 Locations for Jam - Heavy Traffic (6:00-9:00)
Number of Alerts
0
5
10
15
20
25
Latitude
Longitude
42.00
41.98
41.96
41.94
41.92
41.90
41.88
41.86
41.84
41.82
41.80
41.78
41.76
41.74
41.72
41.70
41.68
−87.82
−87.80
−87.78
−87.76
−87.74
−87.72
−87.70
−87.68
−87.66
−87.64
−87.62
−87.60
−87.58

3.

a.

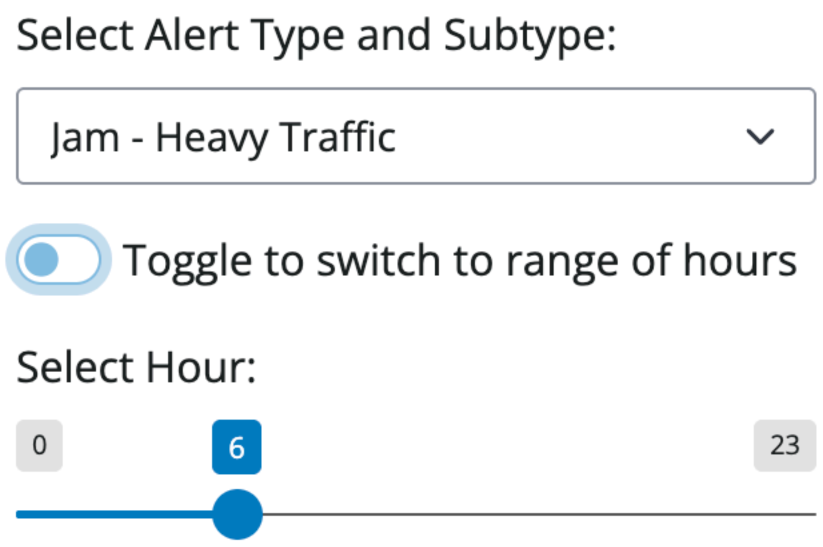


Select Alert Type and Subtype:

Jam - Heavy Traffic

Toggle to switch to range of hours

b.



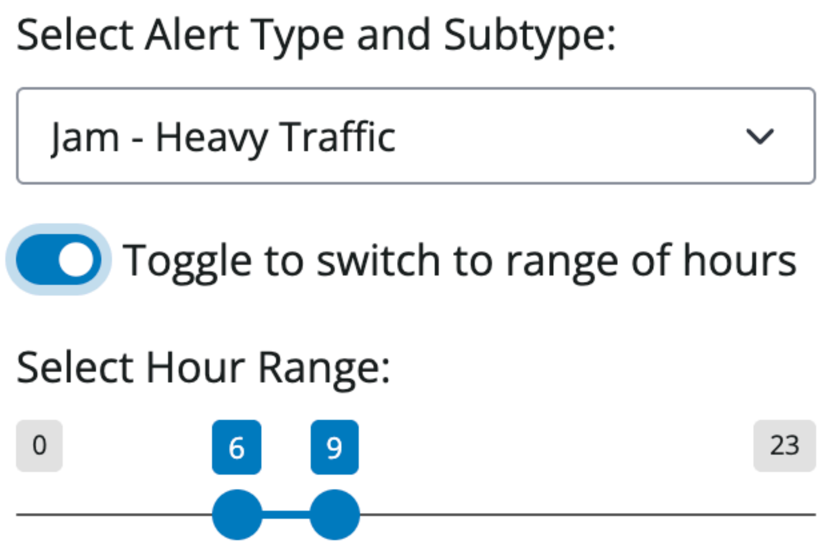
Select Alert Type and Subtype:

Jam - Heavy Traffic

Toggle to switch to range of hours

Select Hour:

0 6 23



Select Alert Type and Subtype:

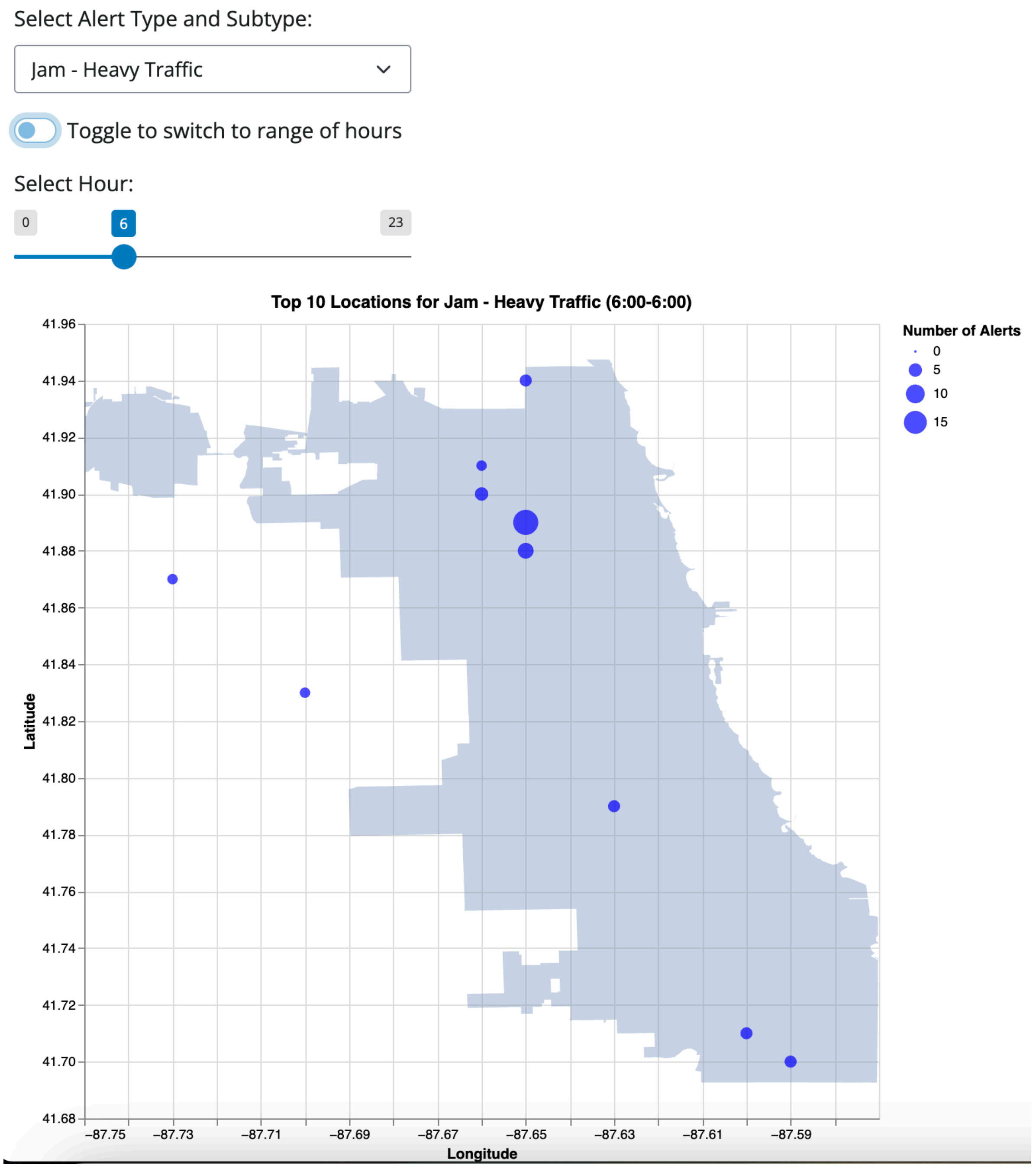
Jam - Heavy Traffic

Toggle to switch to range of hours

Select Hour Range:

0 6 9 23

c.



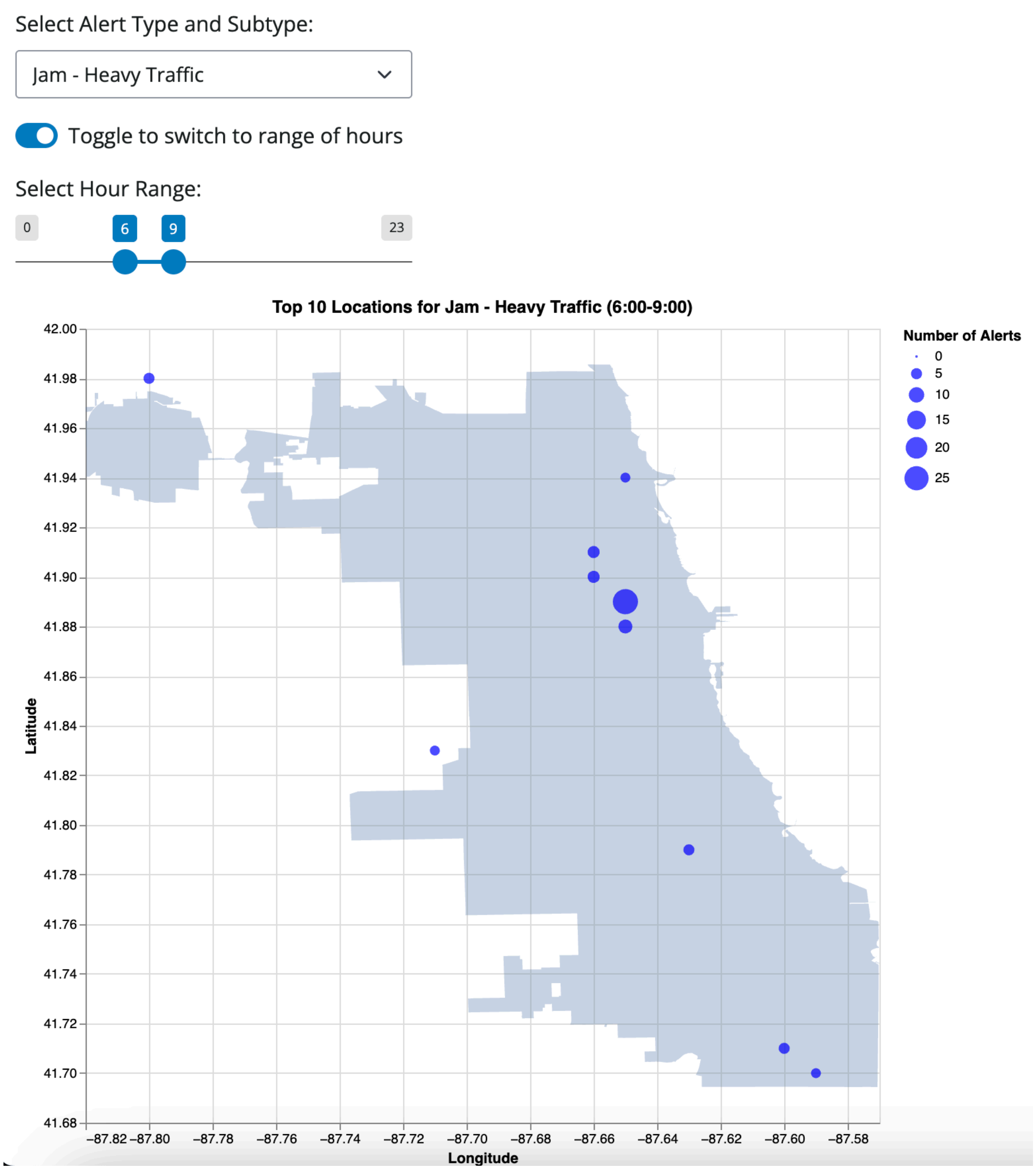
Select Alert Type and Subtype:

Jam - Heavy Traffic

Toggle to switch to range of hours

Select Hour:

0 6 23


Select Alert Type and Subtype:

Jam - Heavy Traffic

Toggle to switch to range of hours

Select Hour Range:

0 6 9 23

d. To achieve a plot similar to the one below, we need to create a new column in the dataset for time periods Morning or Afternoon. In the Altair plot, we use the new column to color the alerts, as shown in the plot (red for Morning, blue for Afternoon). To better visualization, when using mark.circle, we put fill=False to make the circle hollow in case there are overlaps.

```
pip freeze > requirements.txt
```

```
Note: you may need to restart the kernel to use updated packages.
```