Problem Set 6 - Waze Shiny Dashboard

Xinyi Chen

2024-11-22

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: XC
2. "I have uploaded the names of anyone I worked with on the problem set **here**" XC (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. Submit 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 the gradescope repo assignment (5 points).
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 corresponding-section 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!*

```
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
```

Background

Data Download and Exploration (20 points)

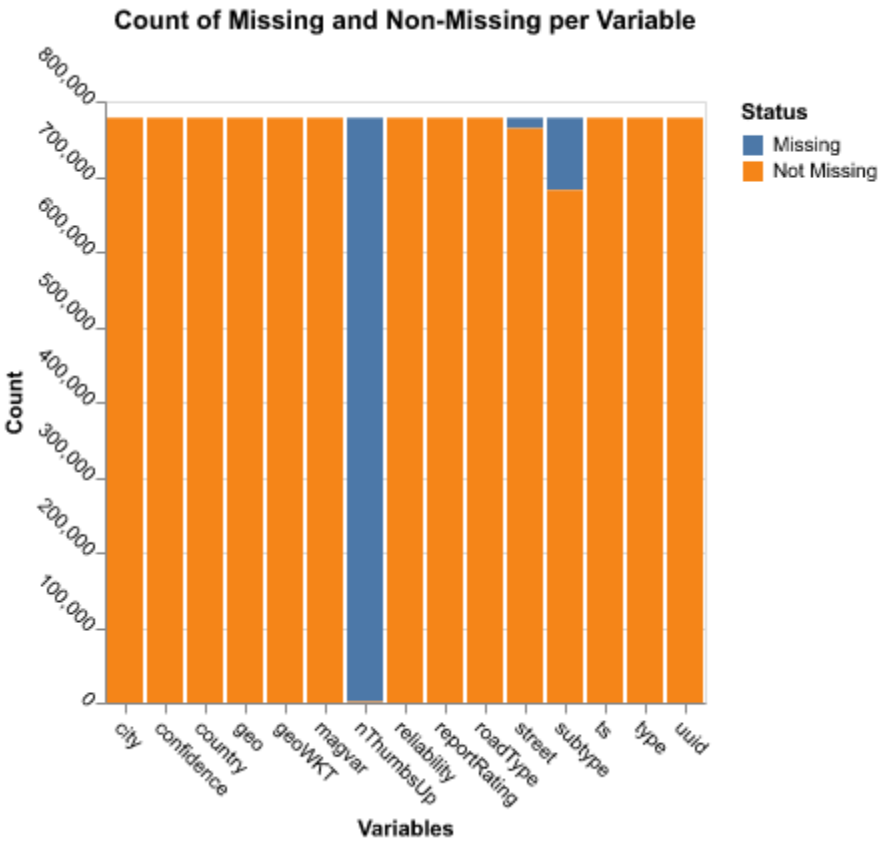
1.

```
import zipfile

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

sample_data = pd.read_csv("waze_data_sample.csv")
```

Variable Name	Altair Data Type(s)
city	Nominal (N)
confidence	Quantitative (Q)
nThumbsUp	Quantitative (Q)
street	Nominal (N)
uuid	Nominal (N)
country	Nominal (N)
type	Nominal (N)
subtype	Nominal (N)
roadType	Quantitative (Q)
reliability	Quantitative (Q)
magvar	Quantitative (Q)
reportRating	Quantitative (Q)

2.

```
waze_data = pd.read_csv("waze_data.csv")

missing_data_long = waze_data.isnull().melt(
    var_name='Variable', value_name='is_missing')
missing_data_long['Status'] = missing_data_long['is_missing'].map(
    {True: 'Missing', False: 'Not Missing'})
missing_data_long = missing_data_long.groupby(
    ['Variable', 'Status']).size().reset_index(name='Count')

alt.Chart(missing_data_long).mark_bar().encode(
    x=alt.X('Variable', sort=None, title='Variables'),
    y=alt.Y('Count', title='Count'),
    color=alt.Color('Status')).properties(
    title="Count of Missing and Non-Missing per Variable",
    width=300,
    height=300).configure_axis(
    labelAngle=45)
```



3.

```
# 3-1: Print unique values for 'type' and 'subtype'
unique_types = waze_data['type'].unique()
unique_subtypes = waze_data['subtype'].unique()
print("Unique values in 'type':", unique_types)
print("Unique values in 'subtype':", unique_subtypes)
```

```
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']
```

```
# 3-2: Count types with NA in 'subtype'
na_subtype = waze_data[waze_data['subtype'].isna()]['type'].nunique()
print(f"Number of unique 'type' values with NA in 'subtype': {na_subtype}")
```

```
Number of unique 'type' values with NA in 'subtype': 4
```

```
# 3-3: Identify types with complex subtypes that could have sub-subtypes
combinations = waze_data[['type', 'subtype']].drop_duplicates()
print("Unique combinations of 'type' and 'subtype':")
print(combinations)
```

```
Unique combinations of 'type' and 'subtype':
               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
```

3-4:

ChatGpt, "What is bulleted listed means? show me a qmd file example"

- **ACCIDENT**
  - Major
  - Minor
  - 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
- **JAM**
    - Heavy Traffic
    - Moderate Traffic
    - Standstill Traffic
    - Light Traffic
    - Unclassified
- **ROAD CLOSED**
    - Event
    - Construction
    - Hazard
    - Unclassified

3-5:

Yes, I think we should keep NA subtypes here. Even if subtypes are missing, they may still provide meaningful information when comparing across types, such as when examining total counts. Removing these missing subtypes could lead to incomplete data and potentially introduce bias into the final results.

4.

```
# 4-a:
import pandas as pd

columns = ['type', 'subtype', 'updated_type',
           'updated_subtype', 'updated_subsubtype']

crosswalk_df = pd.DataFrame(columns=columns)
```

2.

```
# 4-b
type_subtype_data = [
    ('ACCIDENT', 'ACCIDENT_MAJOR', 'Accident', 'Major', None),
    ('ACCIDENT', 'ACCIDENT_MINOR', 'Accident', 'Minor', None),
    ('ACCIDENT', None, 'Accident', 'Unclassified', None),

    ('HAZARD', 'HAZARD_ON_ROAD_CAR_STOPPED', 'Hazard', 'On Road', 'Car Stopped'),
    ('HAZARD', 'HAZARD_ON_ROAD_CONSTRUCTION', 'Hazard', 'On Road', 'Construction'),
    ('HAZARD', 'HAZARD_ON_ROAD_EMERGENCY_VEHICLE',
     'Hazard', 'On Road', 'Emergency Vehicle'),
    ('HAZARD', 'HAZARD_ON_ROAD_ICE', 'Hazard', 'On Road', 'Ice'),
```

```
    ('HAZARD', 'HAZARD_ON_ROAD_OBJECT', 'Hazard', 'On Road', 'Object'),
    ('HAZARD', 'HAZARD_ON_ROAD_POT_HOLE', 'Hazard', 'On Road', 'Pothole'),
    ('HAZARD', 'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT',
     'Hazard', 'On Road', 'Traffic Light Fault'),
    ('HAZARD', 'HAZARD_ON_ROAD_LANE_CLOSED', 'Hazard', 'On Road', 'Lane Closed'),
    ('HAZARD', 'HAZARD_ON_ROAD_ROAD_KILL', 'Hazard', 'On Road', 'Roadkill'),
    ('HAZARD', 'HAZARD_ON_ROAD', 'Hazard', 'On Road', None),

    ('HAZARD', 'HAZARD_ON_SHOULDER_CAR_STOPPED',
     'Hazard', 'On Shoulder', 'Car Stopped'),
    ('HAZARD', 'HAZARD_ON_SHOULDER_ANIMALS', 'Hazard', 'On Shoulder', 'Animals'),
    ('HAZARD', 'HAZARD_ON_SHOULDER_MISSING_SIGN',
     'Hazard', 'On Shoulder', 'Missing Sign'),
    ('HAZARD', 'HAZARD_ON_SHOULDER', 'Hazard', 'On Shoulder', None),
    ('HAZARD', 'HAZARD_WEATHER_FLOOD', 'Hazard', 'Weather', 'Flood'),
    ('HAZARD', 'HAZARD_WEATHER_FOG', 'Hazard', 'Weather', 'Fog'),
    ('HAZARD', 'HAZARD_WEATHER_HEAVY_SNOW', 'Hazard', 'Weather', 'Heavy Snow'),
    ('HAZARD', 'HAZARD_WEATHER_HAIL', 'Hazard', 'Weather', 'Hail'),
    ('HAZARD', 'HAZARD_WEATHER', 'Hazard', 'Weather', None),
    ('HAZARD', None, 'Hazard', 'Unclassified', None),

    ('JAM', 'JAM_HEAVY_TRAFFIC', 'Jam', 'Heavy Traffic', None),
    ('JAM', 'JAM_MODERATE_TRAFFIC', 'Jam', 'Moderate Traffic', None),
    ('JAM', 'JAM_STAND_STILL_TRAFFIC', 'Jam', 'Standstill Traffic', None),
    ('JAM', 'JAM_LIGHT_TRAFFIC', 'Jam', 'Light Traffic', None),
    ('JAM', None, 'Jam', 'Unclassified', None),

    ('ROAD_CLOSED', 'ROAD_CLOSED_EVENT', 'Road Closed', 'Event', None),
    ('ROAD_CLOSED', 'ROAD_CLOSED_CONSTRUCTION',
     'Road Closed', 'Construction', None),
    ('ROAD_CLOSED', 'ROAD_CLOSED_HAZARD', 'Road Closed', 'Hazard', None),
    ('ROAD_CLOSED', None, 'Road Closed', 'Unclassified', None)]

type_subtype_data_df = pd.DataFrame(type_subtype_data, columns=columns)
crosswalk_df = pd.concat(
    [crosswalk_df, type_subtype_data_df], ignore_index=True)

# Chatgpt, "help me debug my code, I want to my each () to be a row in a dataframe"
print(len(crosswalk_df))
```

32

3.

```
# 4-c:
merged_data = waze_data.merge(crosswalk_df, on=['type', 'subtype'], how='left')

accident_unclassified_count = merged_data.query(
    "updated_type == 'Accident' and updated_subtype == 'Unclassified'").shape[0]

# reference of .query():
 ↪  https://stackoverflow.com/questions/44610766/select-columns-using-pandas-dataframe-query
# ChatGpt, 'Why my code with .query() do not show the count?", Using .shape[0] on the result
 ↪  gives the count of rows that match the criteria
```

```
print(accident_unclassified_count)
```

```
24359
```

24359 rows are there for Accident - Unclassified.

4.

```
# 4-d:

# Perform an outer merge
comparison = crosswalk_df[['type', 'subtype']].merge(
    merged_data[['type', 'subtype']].drop_duplicates(),
    on=['type', 'subtype'],
    how='outer',
    indicator=True)

only_in_crosswalk = comparison[comparison['_merge'] == 'left_only']
only_in_merged = comparison[comparison['_merge'] == 'right_only']

# ChatGpt, 'how can I filter merge result column to compare if I do not have left_only and
 ↪  right_only', to get help with comparison[]

if only_in_crosswalk.empty and only_in_merged.empty:
    print("The crosswalk and merged dataset have the same values in type and subtype.")
else:
    print("The crosswalk and merged dataset do not have the same values in type and subtype.")
```

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

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

1.

a.

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

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

# ChatGpt, Can you help me write a a regular expression that extracts the coordinates of
 ↪  longitude and latitude. And Chatgpt tell me to use .str.extract(r'POINT\(([-\d.]+)
 ↪  [-\d.]+\)')
```

b.

```
# round longitude and latitude into step of 0.01
merged_data['binned_longitude'] = merged_data['longitude'].apply(
    lambda x: round(x, 2))
merged_data['binned_latitude'] = merged_data['latitude'].apply(
    lambda x: round(x, 2))

# Group by the binned and count
```

```
merged_data['binned_combination'] = list(
    zip(merged_data['binned_latitude'], merged_data['binned_longitude']))

grouped = merged_data['binned_combination'].value_counts()

most_frequent_combination = grouped.idxmax()
most_frequent_count = grouped.max()

print(
    f"The most frequent binned latitude-longitude combination is: {most_frequent_combination}")

print(f"Number of observations: {most_frequent_count}")

# referece of using zip() and list():
# https://stackoverflow.com/questions/29139350/difference-between-ziplist-and-ziplist
# https://realpython.com/python-zip-function/
```

```
The most frequent binned latitude-longitude combination is: (41.88, -87.65)
Number of observations: 21325
```

c.

```
aggregated_data = (merged_data
    .groupby(['binned_latitude', 'binned_longitude', 'updated_type', 'updated_subtype'])
    .size()
    .reset_index(name='alert_count')
)

# Sort by the highest alert counts
top_alerts = aggregated_data.sort_values(by='alert_count', ascending=False)

output_file = os.path.join('top_alerts_map', "top_alerts_map.csv")
top_alerts.to_csv(output_file, index=False)

# number of rows in DataFrame/CSV
print(len(top_alerts))
```

```
6675
```

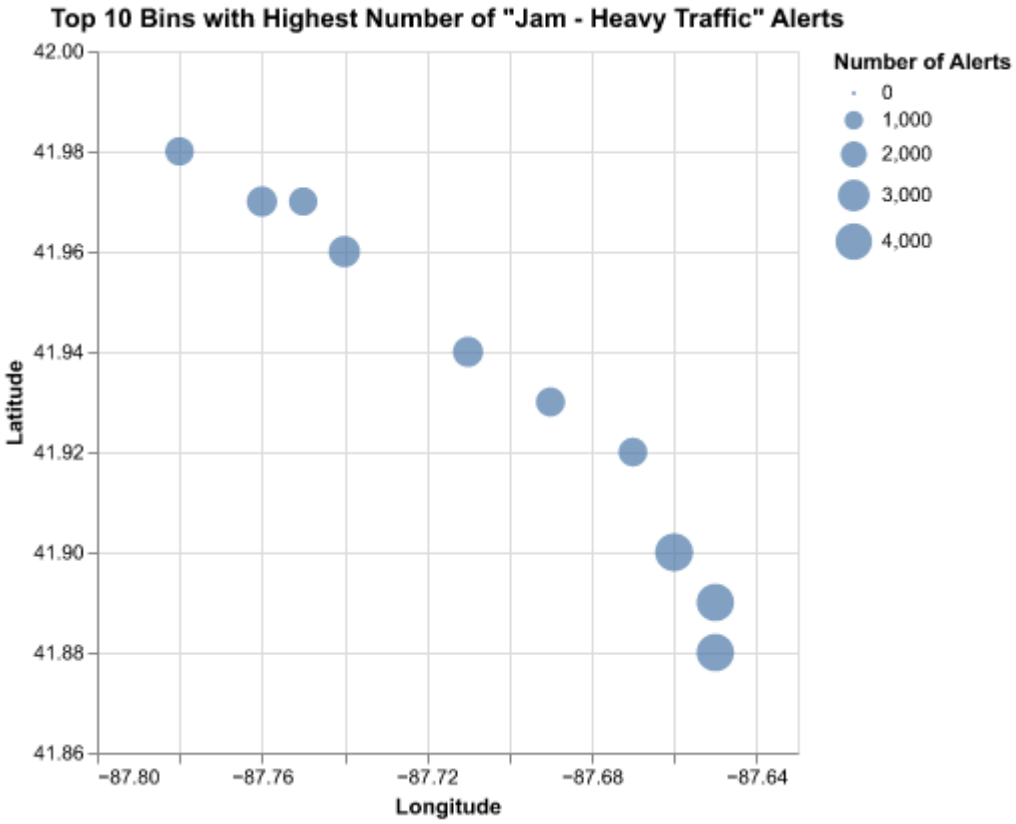
The aggregation data is groupby [‘binned_latitude’, ‘binned_longitude’, ‘type’, ‘subtype’ ‘count’].

And the number of rows in the DataFrame is: 6675.

2.

```
jam_heavy = merged_data[
    (merged_data['updated_type'] == 'Jam') & (merged_data['updated_subtype'] == 'Heavy
 ↪  Traffic')]

# Aggregate again by binned_latitude and binned_longtitude
aggregated_jam_heavy = (
    jam_heavy.groupby(['binned_latitude', 'binned_longitude'])
    .size()
    .reset_index(name='alert_count'))

top_jam_bins = aggregated_jam_heavy.sort_values(
```

```
    by='alert_count', ascending=False).head(10)

lat_min, lat_max = top_jam_bins['binned_latitude'].min(
) - 0.02, top_jam_bins['binned_latitude'].max() + 0.02
long_min, long_max = top_jam_bins['binned_longitude'].min(
) - 0.02, top_jam_bins['binned_longitude'].max() + 0.02
# ChatGpt, can you write a example of define a axis domains for better visualization that I can
 ↪  set between some minimum and maximum values for X and Y.

scatter_plot = (
    alt.Chart(top_jam_bins)
    .mark_circle()
    .encode(
        x=alt.X('binned_longitude:Q', title='Longitude',
                scale=alt.Scale(domain=[long_min, long_max])),
        y=alt.Y('binned_latitude:Q', title='Latitude',
                scale=alt.Scale(domain=[lat_min, lat_max])),
        size=alt.Size('alert_count:Q', title='Number of Alerts'),
        tooltip=['binned_latitude', 'binned_longitude', 'alert_count']
    )
    .properties(
        title='Top 10 Bins with Highest Number of "Jam - Heavy Traffic" Alerts',
        width=350,
        height=350
    ))

scatter_plot.show()
```


Top 10 Bins with Highest Number of "Jam - Heavy Traffic" Alerts
Number of Alerts
0
1,000
2,000
3,000
4,000
Latitude
42.00
41.98
41.96
41.94
41.92
41.90
41.88
41.86
−87.80
−87.76
−87.72
−87.68
−87.64
Longitude

3.

a.

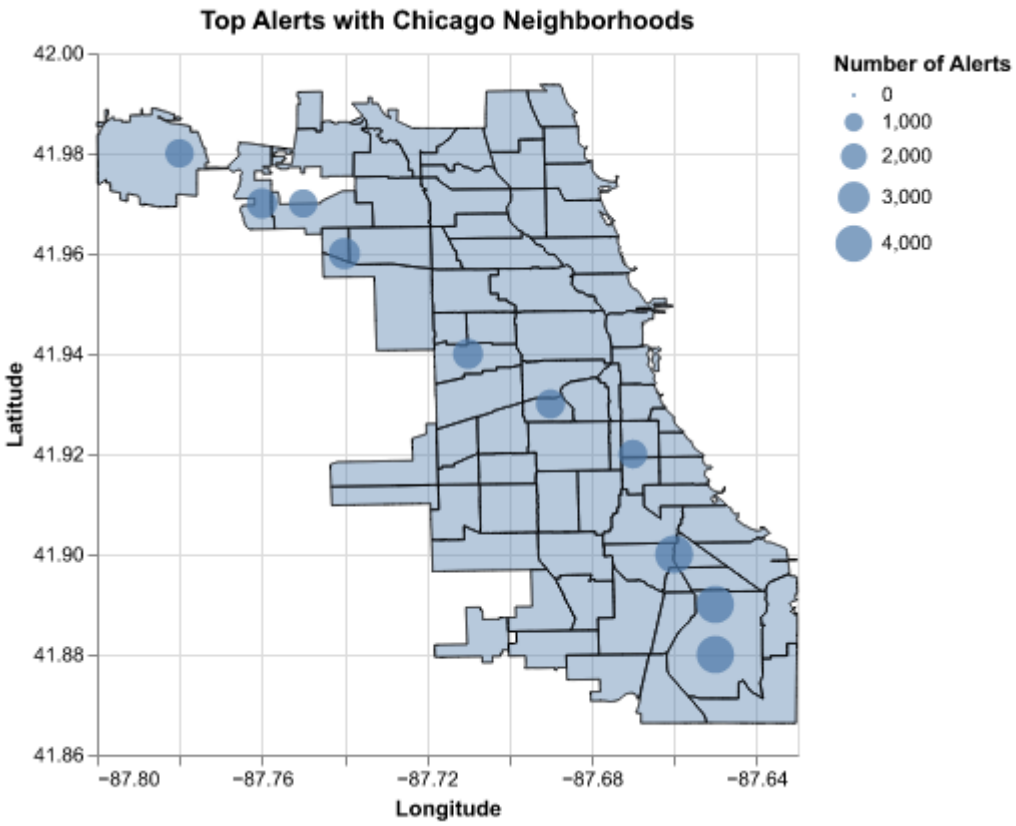
```
import requests

url = "https://data.cityofchicago.org/resource/igwz-8jzy.geojson"

path = "Boundaries_Neighborhoods.geojson"

with open(path, "wb") as file:
    file.write(requests.get(url).content)
```

b.

```
from shapely.geometry import shape

file_path =
 ↪  "/Users/sara/Desktop/Python/problem_sets/ps6/PS6_Xinyi/Boundaries_Neighborhoods.geojson"

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

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

4.

```
chicago_map = alt.Chart(geo_data).mark_geoshape(
    fillOpacity=0.4,
    stroke='black'
).project(
    type="identity", reflectY=True
).encode(
    tooltip=["properties.neighborhood:N"]
)
# ChatGpt, can you help me adjust transparency for my map, like change it to 0.4.

final_chart = chicago_map + scatter_plot
final_chart.properties(
    title="Top Alerts with Chicago Neighborhoods",
    height=350,
    width=350
)
```



5.

a.

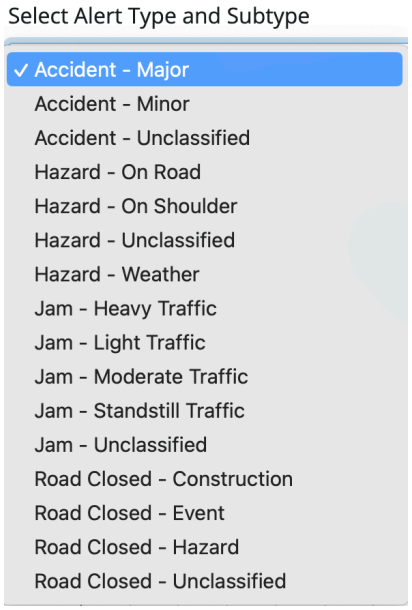


Figure 1: dropdown_menu

There are 16 combinations of type and subtypes in the Shiny App.

b.

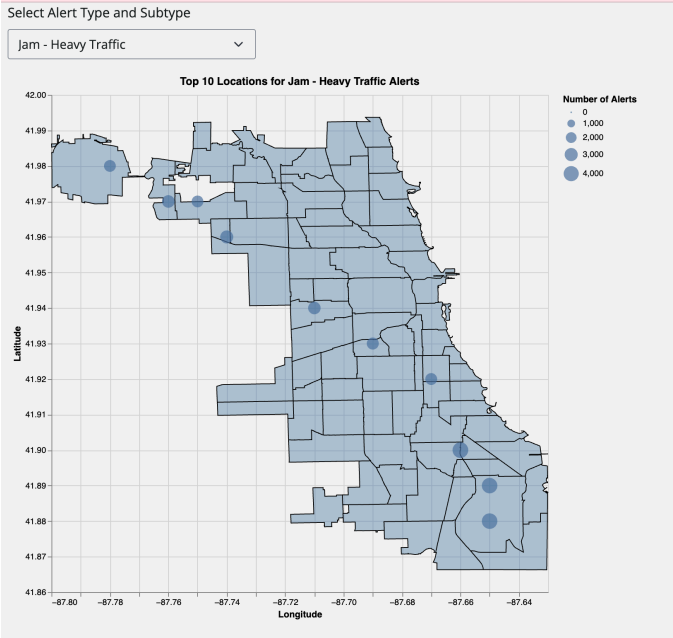


Figure 2: jam_heavy traffic

c.

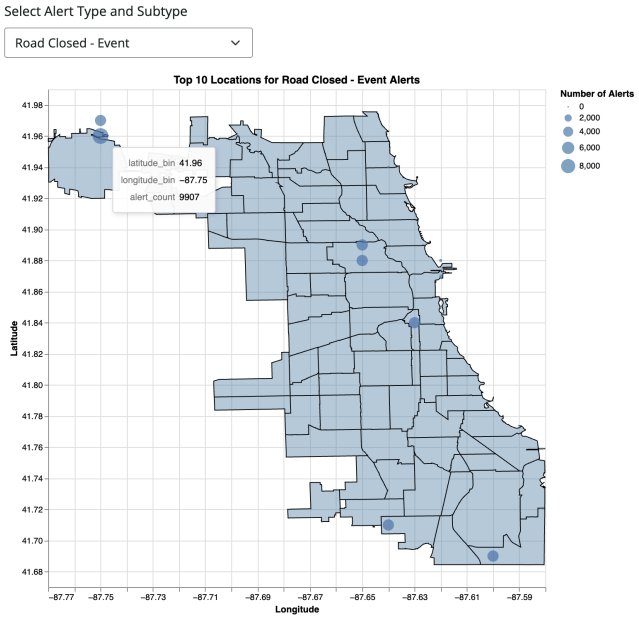


Figure 3: Road_close event

d.

Question: How does the distribution of “Road Closed - Construction” alerts compare to “Road Closed - Hazard” alerts in Chicago?

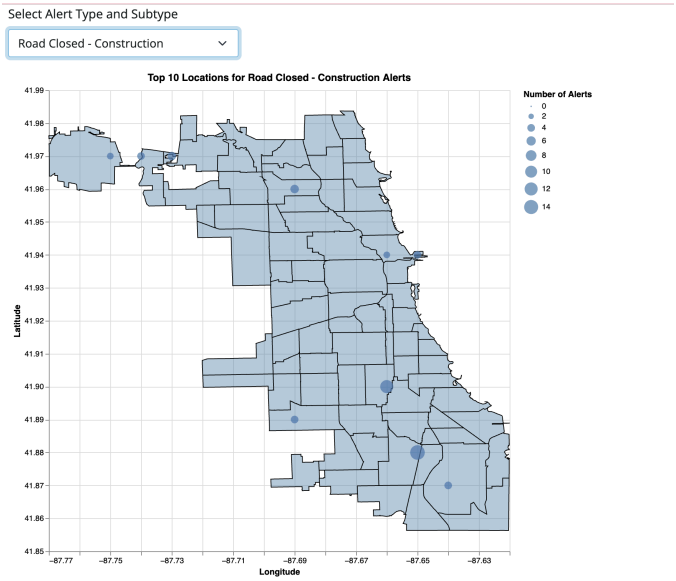


Figure 4: road close construction

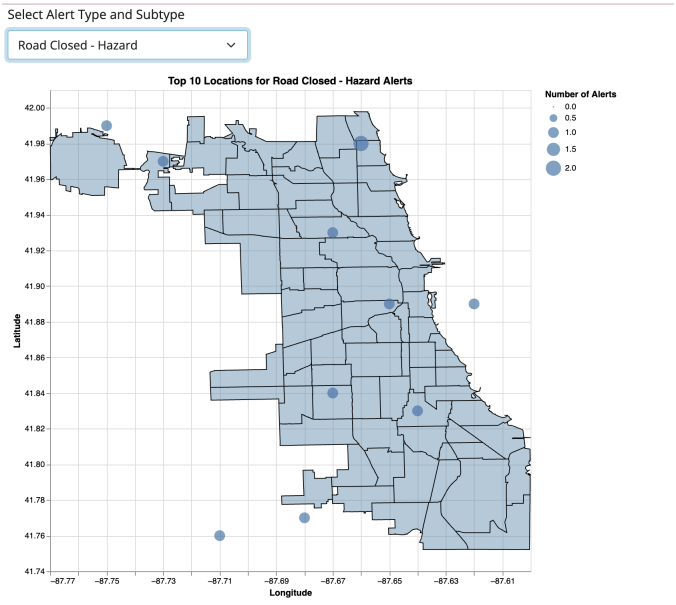


Figure 5: road close hazard

Answer: “Road Closed - Construction” alerts have higher densities in specific areas, while “Road Closed - Hazard” alerts are sparsely distributed. And that might becasue construction alerts likely follow scheduled urban development plans, whereas hazard alerts appear due to temporary or unplanned issues.

e.

By adding a "Time of Alert" column, which captures the timestamp of each alert and "Severity Level" column which categorizes the intensity or importance of each alert. For example, "Accident - Major" is high, "Jam - Light Traffic" is low. And for time column, on dashboard we can choose a time range like, Show alerts between 8 AM and 6 PM" for more specific queries.

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

1.

a.

Looking at the provided dataset, collapsing it by the ts (timestamp) column might not be a good idea in most cases. Each row corresponds to a unique event with its associated attributes (e.g., updated_type, updated_subtype, location). Collapsing by ts would aggregate events occurring at the same time, potentially losing the granularity and specificity of individual events. Instead, consider creating additional features or aggregating over broader time intervals while preserving the granularity of other columns for more comprehensive analysis.

b.

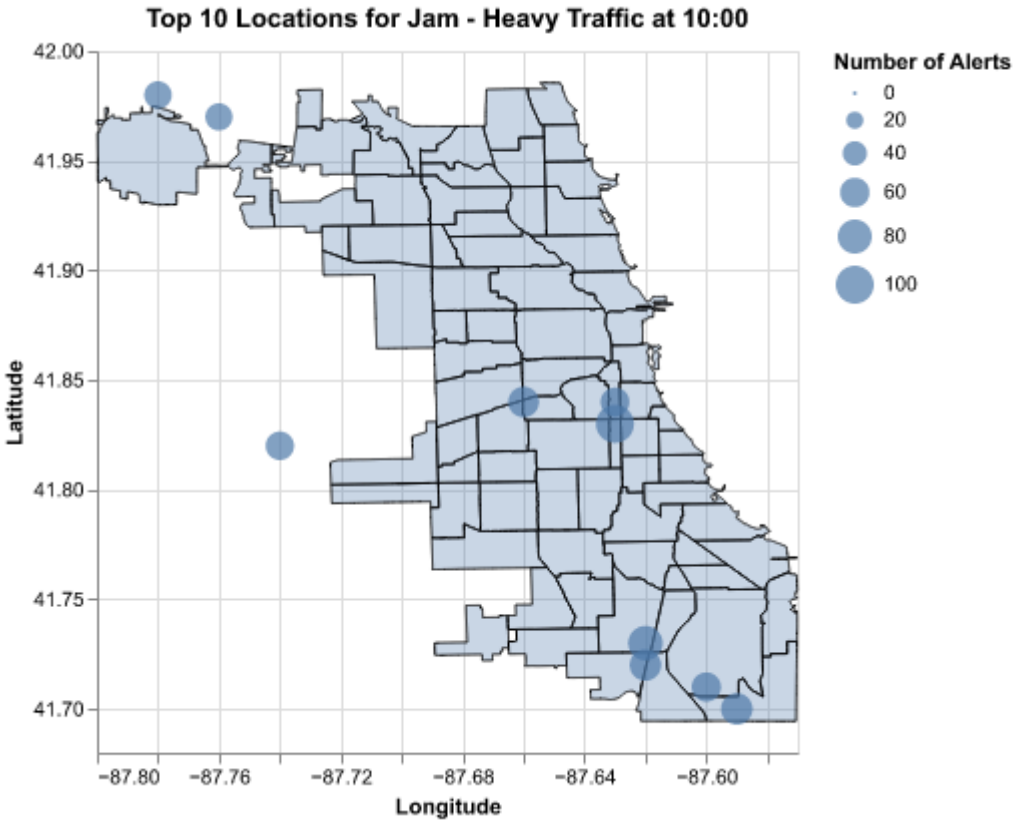
```
merged_data = pd.read_csv("merged_data.csv")

# Convert 'ts' to extract the hour
merged_data['hour'] = pd.to_datetime(
    merged_data['ts']).dt.hour.astype(str) + ":00"

merged_data['latitude_bin'] = merged_data['latitude'].round(2)
merged_data['longitude_bin'] = merged_data['longitude'].round(2)

collapsed_by_hour = (
    merged_data
    .groupby(['hour', 'updated_type', 'updated_subtype', 'latitude_bin', 'longitude_bin'])
    .size()
    .reset_index(name='alert_count'))

output_folder = "top_alerts_map_byhour"
output_file = f"{output_folder}/top_alerts_map_byhour.csv"
collapsed_by_hour.to_csv(output_file, index=False)

# ChatGpt, 'How can I extract hour from my data of time, suppose I already change the format to
 ↪  datatime'

print(f"Number of rows in the dataset: {len(collapsed_by_hour)}")
```

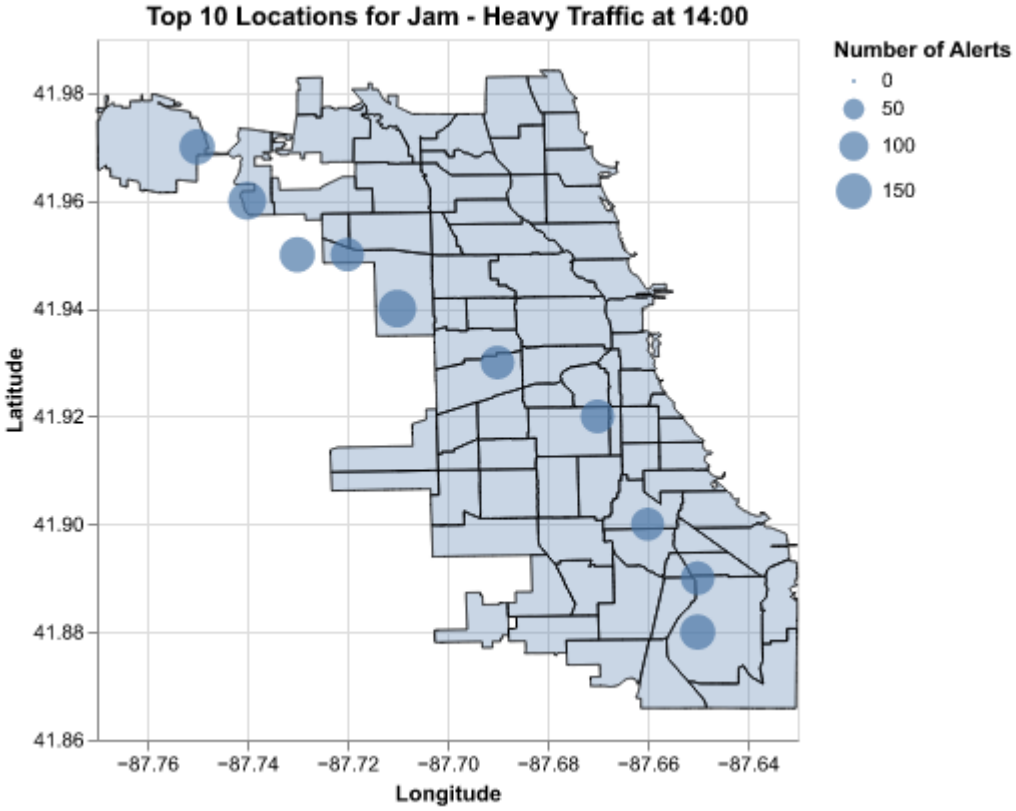
```
Number of rows in the dataset: 62825
```

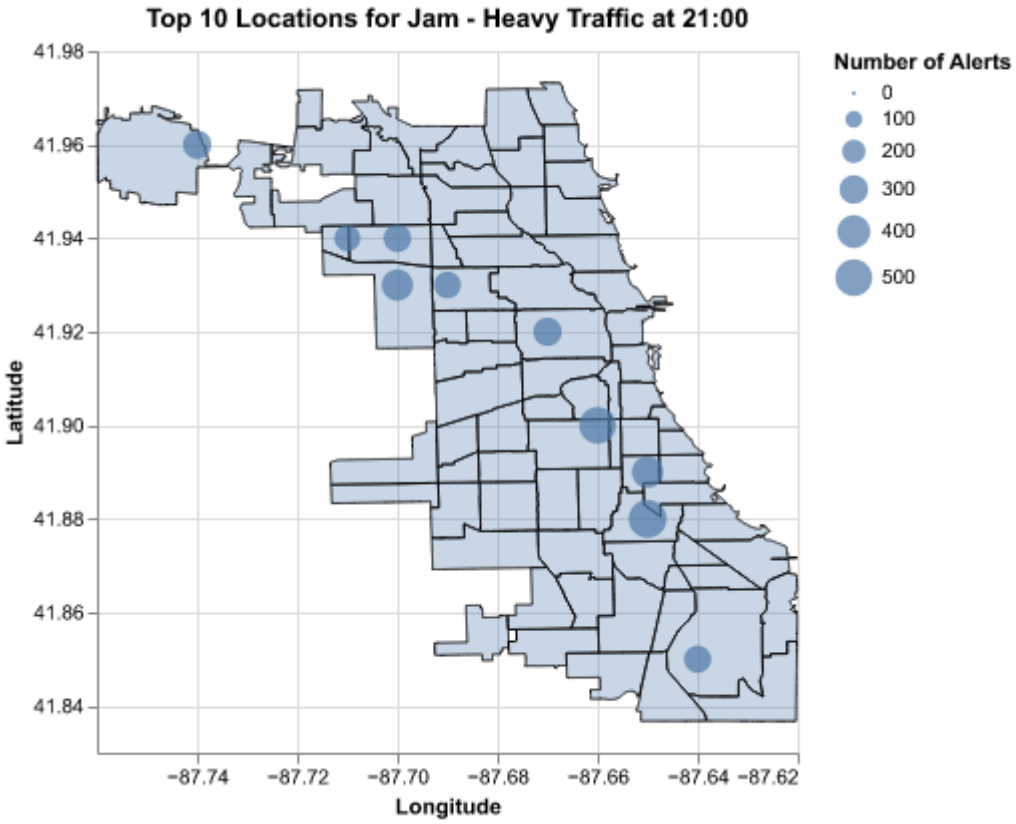
c.

```
# same step to extract time to hour
merged_data['hour'] = pd.to_datetime(
    merged_data['ts']).dt.hour.astype(str) + ":00"

jam_heavy = merged_data[
    (merged_data['updated_type'] == 'Jam') & (
        merged_data['updated_subtype'] == 'Heavy Traffic')
]
```

```
jam_heavy['binned_latitude'] = jam_heavy['latitude'].round(2)
jam_heavy['binned_longitude'] = jam_heavy['longitude'].round(2)

# Select three hours I want
hours_to_plot = ['10:00', '14:00', '21:00']

plots = []
for hour in hours_to_plot:
    jam_heavy_hour = jam_heavy[jam_heavy['hour'] == hour]
    aggregated_jam_heavy_hour = (
        jam_heavy_hour.groupby(['binned_latitude', 'binned_longitude'])
        .size()
        .reset_index(name='alert_count'))

    top_jam_bins = aggregated_jam_heavy_hour.sort_values(
        by='alert_count', ascending=False).head(10)

    # set domain of x and y to ensure all point insde
    lat_min, lat_max = top_jam_bins['binned_latitude'].min(
    ) - 0.02, top_jam_bins['binned_latitude'].max() + 0.02
    long_min, long_max = top_jam_bins['binned_longitude'].min(
    ) - 0.02, top_jam_bins['binned_longitude'].max() + 0.02

    scatter_plot = alt.Chart(top_jam_bins).mark_circle().encode(
        x=alt.X('binned_longitude:Q', title='Longitude',
                scale=alt.Scale(domain=[long_min, long_max])),
        y=alt.Y('binned_latitude:Q', title='Latitude',
                scale=alt.Scale(domain=[lat_min, lat_max])),
        size=alt.Size('alert_count:Q', title='Number of Alerts')
    ).properties(
        title=f'Top 10 Locations for Jam - Heavy Traffic at {hour}',
        width=350,
        height=350)

    map_layer = alt.Chart(geo_data).mark_geoshape(
        fillOpacity=0.3,
        stroke='black').encode(
        tooltip=["properties.neighborhood:N"]
    ).project(
        type="identity", reflectY=True).properties(
        width=350,
        height=350)

    combined_plot = map_layer + scatter_plot
    plots.append(combined_plot)

# ChatGpt, "helo debug my code that I want to apply three hours I choose and apply to the map,
 ↪  and append map with scatter plot togather with time"

for plot in plots:
    plot.display()
```


Top 10 Locations for Jam - Heavy Traffic at 10:00
Number of Alerts
0
20
40
60
80
100
Latitude
42.00
41.95
41.90
41.85
41.80
41.75
41.70
−87.80
−87.76
−87.72
−87.68
−87.64
−87.60
Longitude


Top 10 Locations for Jam - Heavy Traffic at 14:00
Number of Alerts
0
50
100
150
Latitude
41.98
41.96
41.94
41.92
41.90
41.88
41.86
−87.76
−87.74
−87.72
−87.70
−87.68
−87.66
−87.64
Longitude


Top 10 Locations for Jam - Heavy Traffic at 21:00
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
-87.74
-87.72
-87.70
-87.68
-87.66
-87.64
-87.62

2.

a.

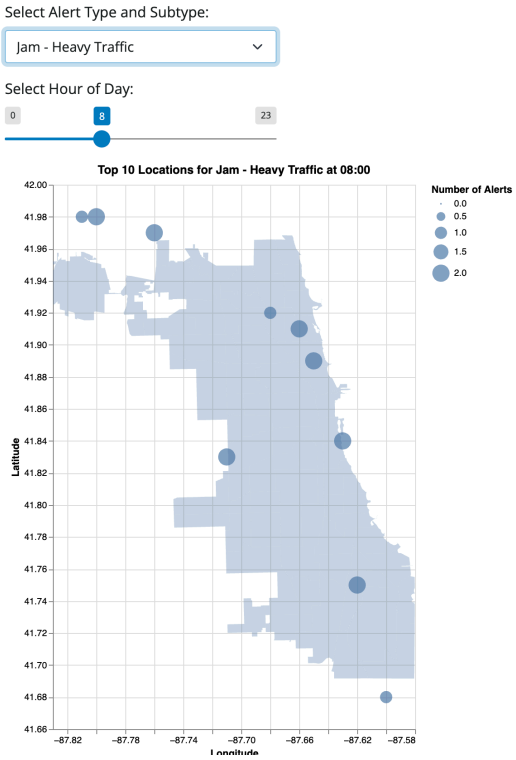


Figure 6: dropdown and slider

b.

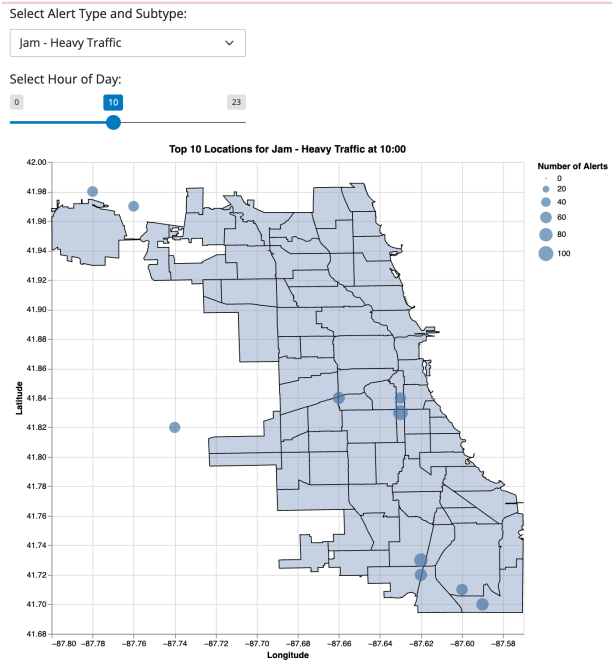


Figure 7: Top 10 Locations for Jam - Heavy Traffic at 10:00

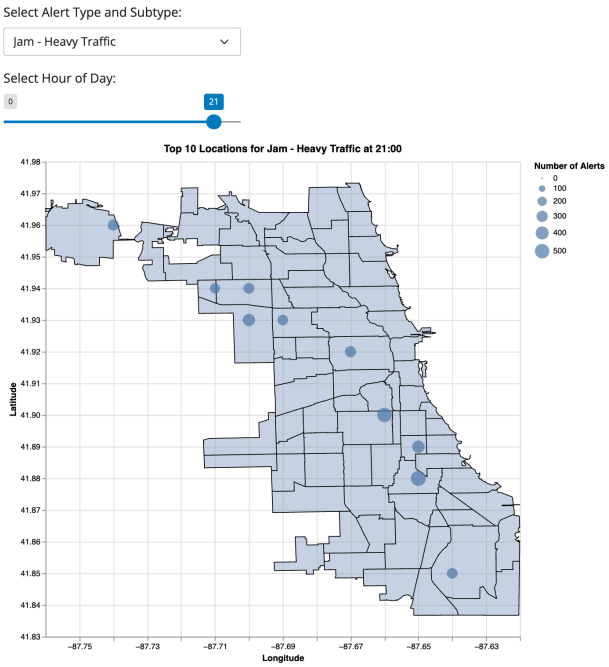


Figure 8: Top 10 Locations for Jam - Heavy Traffic at 14:00

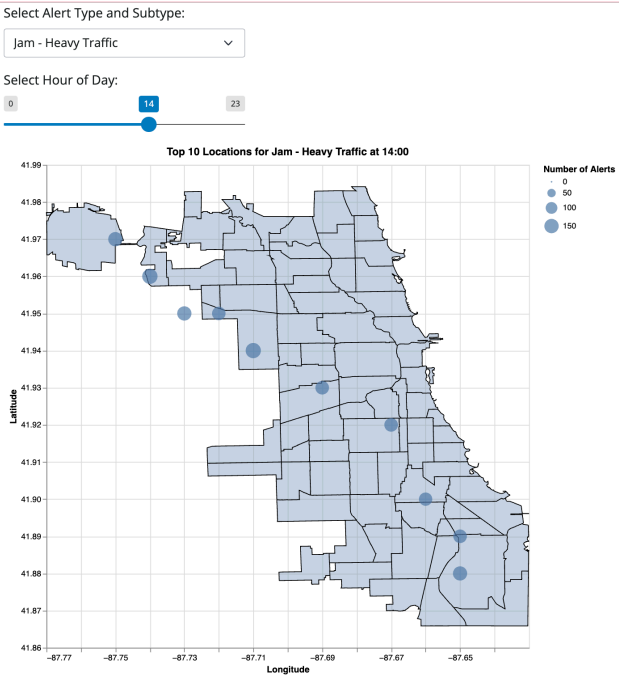


Figure 9: Top 10 Locations for Jam - Heavy Traffic at 21:00

c.

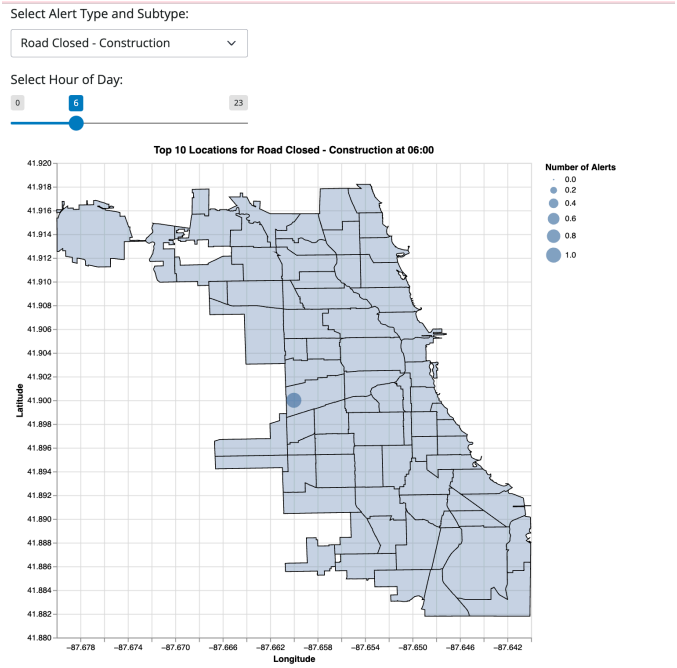


Figure 10: road construction morning

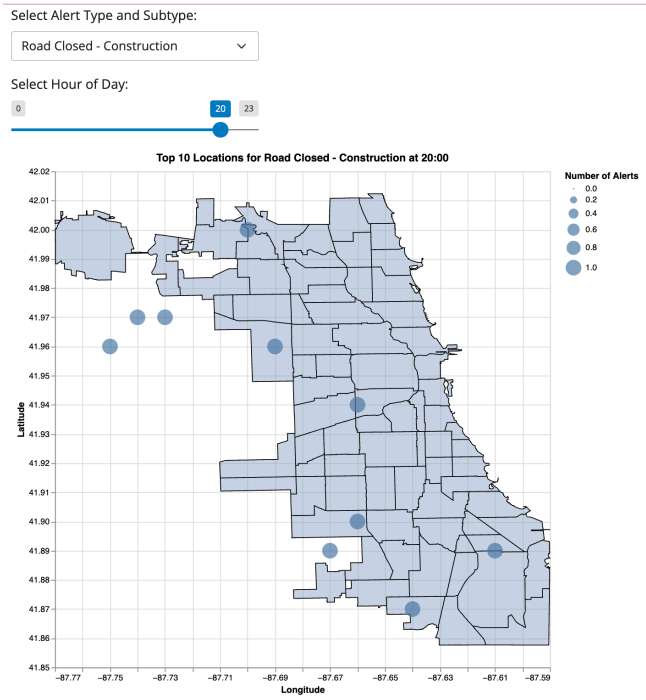


Figure 11: road construction night

road construction is done more during night hours, like 6:00 only have one alert, but 20:00 come up with more alert in more areas.

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

1.

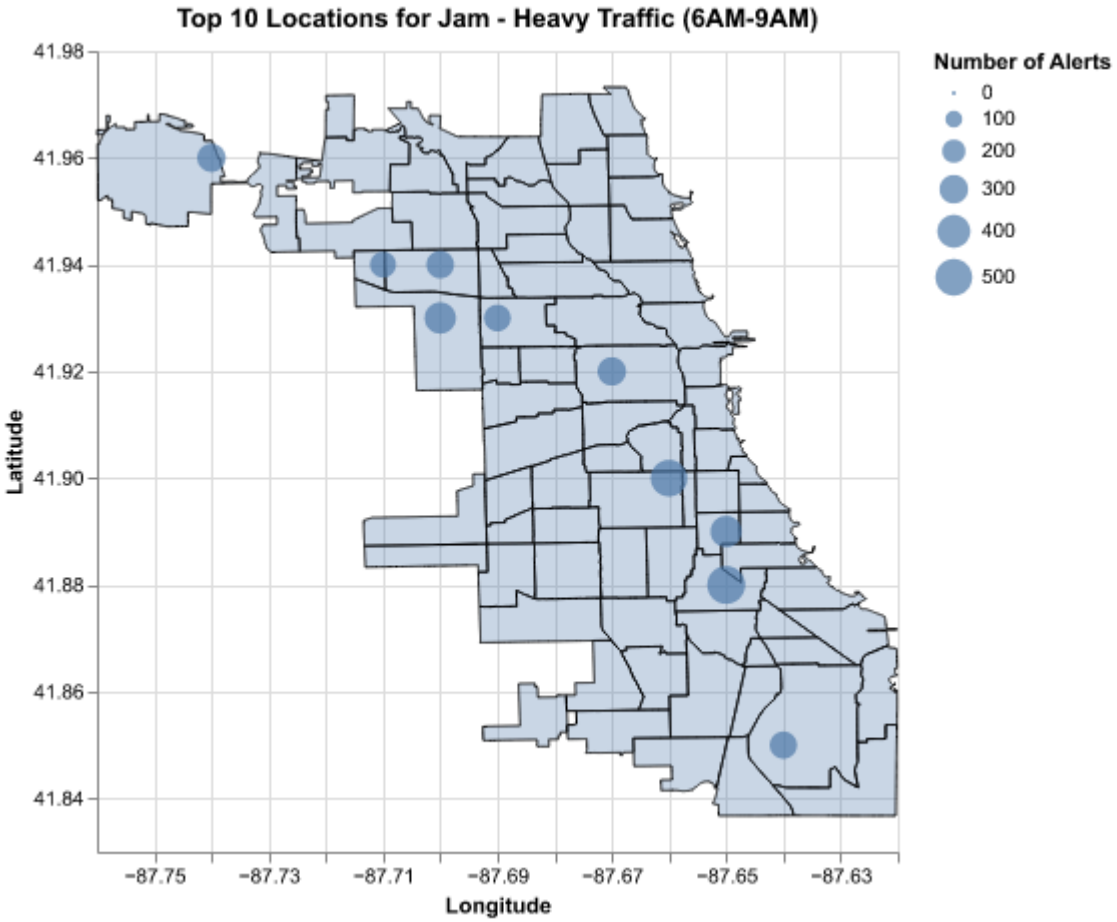
a.

Collapsing the dataset by range of hours can be a good idea because picking a range of time—such as morning, noon, afternoon, and night—is often more meaningful in reality. These broader time ranges align better with natural patterns of human activity and help users understand how time influences the top 10 alert locations in a more intuitive way.

b.

```
# Ensure 'hour' is in correct format
merged_data['ts'] = pd.to_datetime(merged_data['ts'], errors='coerce')  # Parse the datetime

merged_data['hour'] = merged_data['ts'].dt.hour

# ChatGpt, debug my code beucase there is typeerror on hour column.

# Define the default
start_hour = 6
end_hour = 9

jam_heavy['hour'] = pd.to_numeric(jam_heavy['hour'], errors='coerce')

jam_heavy_range = jam_heavy[
    (jam_heavy['hour'] >= start_hour) &
    (jam_heavy['hour'] < end_hour)
]

# Scatter plot for the top 10 locations
scatter_plot = alt.Chart(top_jam_bins).mark_circle().encode(
    x=alt.X('binned_longitude:Q', title='Longitude', scale=alt.Scale(domain=[long_min,
 ↪  long_max])),
    y=alt.Y('binned_latitude:Q', title='Latitude', scale=alt.Scale(domain=[lat_min, lat_max])),
    size=alt.Size('alert_count:Q', title='Number of Alerts'),
    tooltip=['binned_latitude', 'binned_longitude', 'alert_count']
).properties(
    title=f'Top 10 Locations for Jam - Heavy Traffic (6AM-9AM)',
)

# Map layer for Chicago
map_layer = alt.Chart(geo_data).mark_geoshape(
    fillOpacity=0.3,
    stroke='black'
).encode(
    tooltip=["properties.neighborhood:N"]
).project(
    type="identity", reflectY=True
)

combined_plot = map_layer + scatter_plot

combined_plot.properties(
    width=400,
```

```
    height=400).display()
```



2.

a.

```python
app_ui = ui.page_fluid(

    ui.input_select(
        id="type_subtype",
        label="Select Alert Type and Subtype:",
        choices=dropdown_choices,
        selected=dropdown_choices[0]
    ),

    # Slider for selecting the hour range
    ui.input_slider(
        id="hour_range",
        label="Select Hour Range:",
        min=0,  # Start
        max=23,  # End
        value=(6, 9),  # Default
        step=1  # step by 1 hour
    ),

    output_widget("top_alerts_plot")
)
```

Figure 12: ui_code

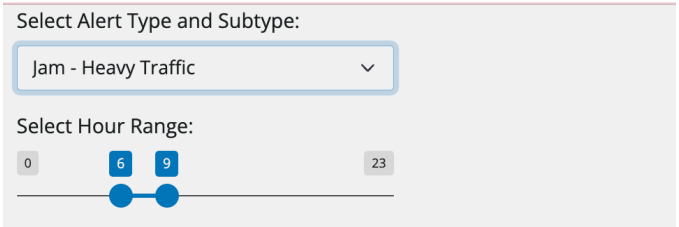


Figure 13: ui_app

b.

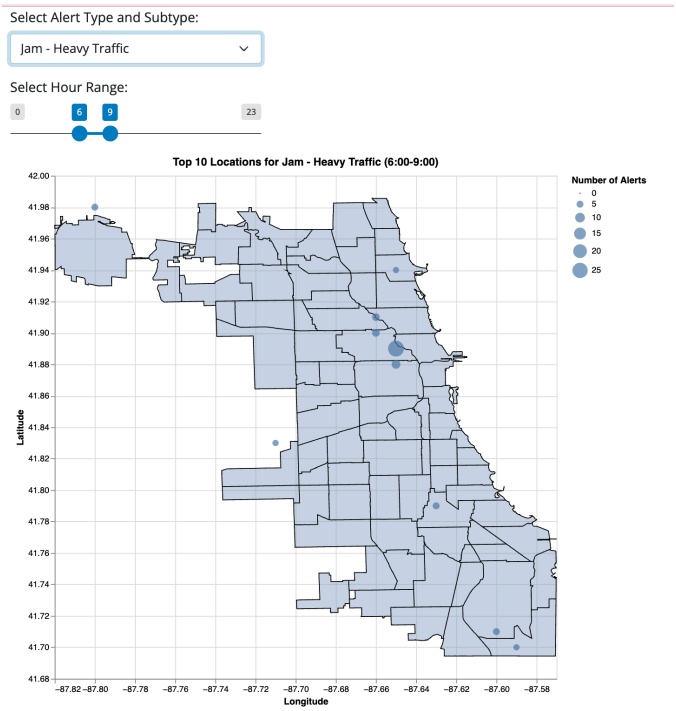


Figure 14: "Jam - Heavy Traffic with range

3.

a.

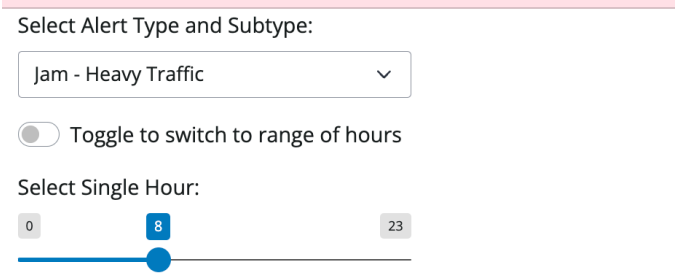


Figure 15: switch button

b.

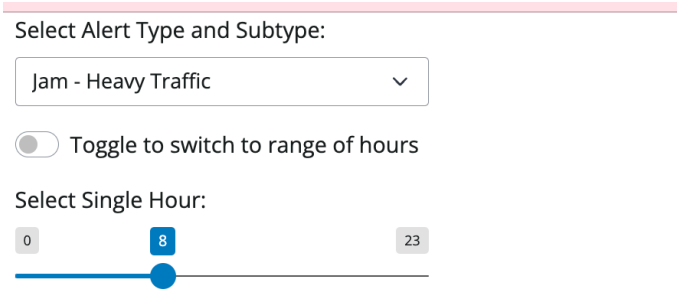


Figure 16: switch button is not toggled

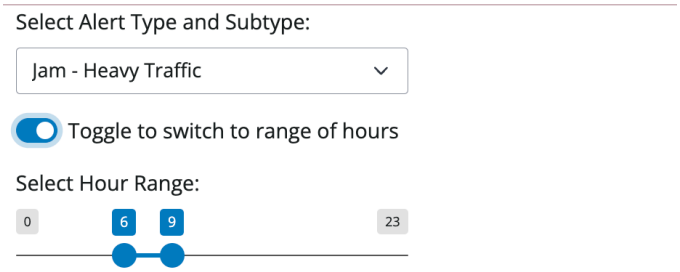


Figure 17: switch button is toggled

c.

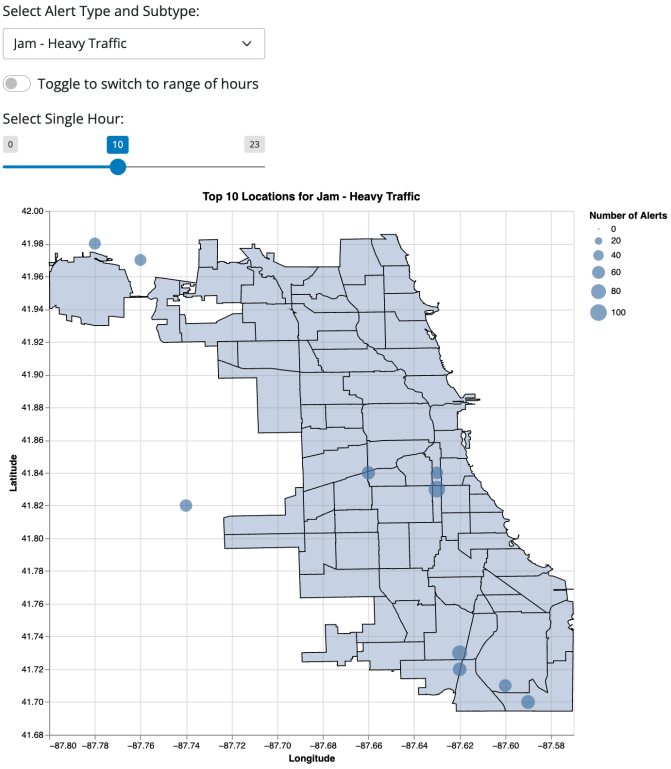


Figure 18: slider for a single hour

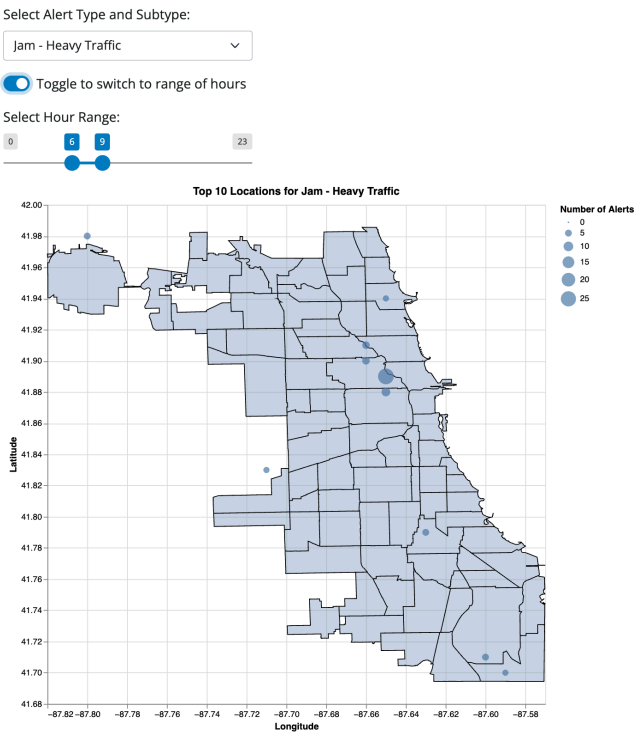


Figure 19: slider for a range of hours

d.

First, I think a new column called 'time_periods' should be added to the dataset, categorizing the data into two time periods: Morning (6 AM - 12 PM) and Afternoon (12 PM - 6 PM). Then, as described in the steps above, the data can be aggregated by latitude, longitude, and the newly created time_period variable, while also calculating the number of alerts for each group.

On the UI side, it might be useful to allow users to select either a specific time or a time period. Additionally, users could have the option to choose broader time periods, such as Morning or Afternoon, for easier interpretation and understanding.

For the graph, I suggest adding a color indicator to represent time periods (e.g., red for Morning and blue for Afternoon) and using circle size to indicate the number of alerts. This can be implemented by encoding time_period in the color channel and alert_count in the size channel of the Altair chart. This approach will visually distinguish the time periods and provide a clear understanding of the number of alerts at different times.