Problem Set 6 - Waze Shiny Dashboard

Luis Señires

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: **LS**
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **LS** (2 point)
3. Late coins used this pset: **00** Late coins left after submission: **01**
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!

```
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.

```
# Extract data from zip file
base = r"C:\Users\LUIS\Documents\GitHub\problem-set-6"
folder = "waze_data"
path = os.path.join(base, folder, "waze_data_sample.csv")

with ZipFile("waze_data.zip", "r") as f:
    f.extractall(folder)

# Load sample data
df_waze_sample = pd.read_csv(path)
```

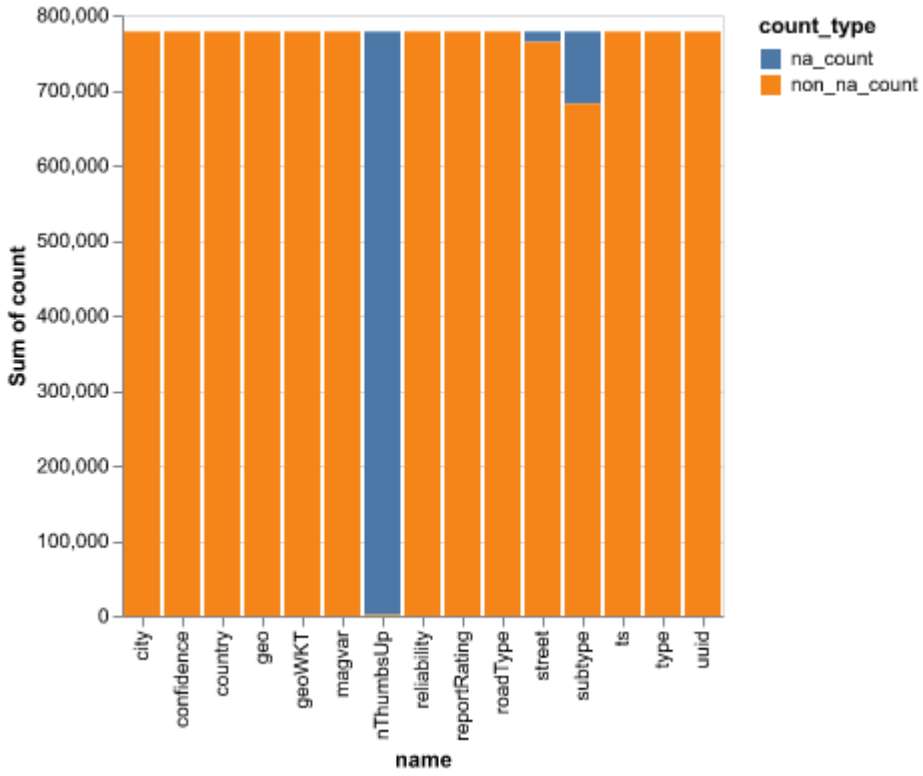
```
# Get variable names and data types
df_cols = pd.DataFrame()
df_cols["name"] = df_waze_sample.columns.tolist()
df_cols["dtype"] = df_waze_sample.dtypes.tolist()

# Assign data types using Altair syntax
df_cols["altair_dtype"] = ["O", "N", "Q", "Q", "N", "N",
                           "N", "N", "N", "O", "Q", "Q", "O", "-", "-", "-"]

# Print table
print(df_cols.to_markdown())
```

```
|    | name          | dtype   | altair_dtype   |
|---:|:--------------|:--------|:---------------|
|  0 | Unnamed: 0    | int64   | O              |
|  1 | city          | object  | N              |
|  2 | confidence    | int64   | Q              |
|  3 | nThumbsUp     | float64 | Q              |
|  4 | street        | object  | N              |
|  5 | uuid          | object  | N              |
|  6 | country       | object  | N              |
|  7 | type          | object  | N              |
|  8 | subtype       | object  | N              |
|  9 | roadType      | int64   | O              |
| 10 | reliability   | int64   | Q              |
| 11 | magvar        | int64   | Q              |
| 12 | reportRating  | int64   | O              |
| 13 | ts            | object  | -              |
| 14 | geo           | object  | -              |
| 15 | geoWKT        | object  | -              |
```

2.

```
# Load full data
folder = "waze_data"
path = os.path.join(base, folder, "waze_data.csv")

df_waze = pd.read_csv(path)

# Count number of missing and non-missing values
na_count = df_waze.isna().sum()
non_na_count = df_waze.notna().sum()
```

```
# Create new dataframe with counts
df_count = pd.DataFrame({
    "name": na_count.index,
    "na_count": na_count.values,
    "non_na_count": non_na_count.values
})

# Melt data for stacking
df_count_long = df_count.melt(id_vars="name", value_vars=[
                              "na_count", "non_na_count"],
                               ↪  var_name="count_type", value_name="count")

# Create stacked bar chart
chart_stacked_count = alt.Chart(df_count_long).mark_bar().encode(
    alt.X("name:N"),
    alt.Y("sum(count):Q"),
    alt.Color("count_type")
)

chart_stacked_count
```



The variables with missing values are nThumbsUp, street, and subtype, with nThumbsUp having the highest share of missing values.

3.

```
# Combine data by "type" and "subtype"
df_type = df_waze.groupby(
    ["type", "subtype"], dropna=False).size().reset_index()

# Get unique values
unique_type = df_type["type"].unique()
print(unique_type)

unique_subtype = df_type["subtype"].unique()
print(unique_subtype)
```

```
['ACCIDENT' 'HAZARD' 'JAM' 'ROAD_CLOSED']
['ACCIDENT_MAJOR' 'ACCIDENT_MINOR' nan 'HAZARD_ON_ROAD'
 'HAZARD_ON_ROAD_CAR_STOPPED' 'HAZARD_ON_ROAD_CONSTRUCTION'
 'HAZARD_ON_ROAD_EMERGENCY_VEHICLE' 'HAZARD_ON_ROAD_ICE'
```

```
'HAZARD_ON_ROAD_LANE_CLOSED' 'HAZARD_ON_ROAD_OBJECT'
'HAZARD_ON_ROAD_POT_HOLE' 'HAZARD_ON_ROAD_ROAD_KILL'
'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT' 'HAZARD_ON_SHOULDER'
'HAZARD_ON_SHOULDER_ANIMALS' 'HAZARD_ON_SHOULDER_CAR_STOPPED'
'HAZARD_ON_SHOULDER_MISSING_SIGN' 'HAZARD_WEATHER' 'HAZARD_WEATHER_FLOOD'
'HAZARD_WEATHER_FOG' 'HAZARD_WEATHER_HAIL' 'HAZARD_WEATHER_HEAVY_SNOW'
'JAM_HEAVY_TRAFFIC' 'JAM_LIGHT_TRAFFIC' 'JAM_MODERATE_TRAFFIC'
'JAM_STAND_STILL_TRAFFIC' 'ROAD_CLOSED_CONSTRUCTION' 'ROAD_CLOSED_EVENT'
'ROAD_CLOSED_HAZARD']
```

There are four "types" present in the dataset (i.e. accident, hazard, jam, and road closed), and all four have a missing value "subtype." The Hazard type has enough information to consider that it could have sub-subtypes.

- Accident
  - Major
  - Minor
- Hazard
  - On road
    - Car stopped
    - Construction
    - Emergency vehicle
    - Ice
    - Lane closed
    - Object
    - Pot hole
    - Road kill
    - Traffic light fault
  - On shoulder
    - Animals
    - Car stopped
    - Missing sign
  - Weather
    - Flood
    - Fog
    - Hail
    - Heavy snow
- Jam
  - Heavy traffic
  - Light traffic

  - Moderate traffic
  - Stand still traffic

- Road Closed
  - Construction
  - Event
  - Hazard

I think we should keep the NA subtype because there is a significant number of entries in the dataset with this subtype, and removing them could change the composition of our data and affect the results of our analysis/dashboard.

```
# Reclassify missing values
df_type["subtype"] = df_type["subtype"].fillna("Unclassified")
```

4.

a.

```
# Create new df with initial columns
df_crosswalk = pd.DataFrame()
df_crosswalk["type"] = df_type["type"]
df_crosswalk["subtype"] = df_type["subtype"]
df_crosswalk["updated_type"] = ""
df_crosswalk["updated_subtype"] = ""
df_crosswalk["updated_subsubtype"] = ""
```

b.

```
# Fill in updated type
df_crosswalk["updated_type"] = df_crosswalk["type"].str.capitalize()

# Define a function to update subtype


def update_subtype(row):
    # Check if unclassified
    if row == "Unclassified":
        return "Unclassified"

    # Check if hazard
    if "HAZARD" in row:
```

```
        # Create list of Hazard subtypes
        hazard_subtypes = ["ON_ROAD", "ON_SHOULDER", "WEATHER"]

        # Iterate over list
        for index in range(0, 3):
            if hazard_subtypes[index] in row:
                return hazard_subtypes[index]

    # Create list of remaining alert types
    alert_types = ["ACCIDENT", "JAM", "ROAD_CLOSED"]

    # Iterate over list
    for index in range(0, 3):
        length = len(alert_types[index])

        if alert_types[index] in row:
            return row[(length + 1):]


# Fill in updated subtype
df_crosswalk["updated_subtype"] =
 ↪  df_crosswalk["subtype"].apply(update_subtype)
df_crosswalk["updated_subtype"] =
 ↪  df_crosswalk["updated_subtype"].str.capitalize()

# Define a function to update subsubtype


def update_subsubtype(row):
    # Check if hazard
    if "HAZARD" in row:

        # Create list of Hazard subtypes
        hazard_subtypes = ["ON_ROAD_", "ON_SHOULDER_", "WEATHER_"]

        # Iterate over list
        for index in range(0, 3):
            length = len(hazard_subtypes[index])

            if hazard_subtypes[index] in row:
                return row[(length + 7):]
```

```
    else:
        return np.nan


# Fill in updated subsubtype
df_crosswalk["updated_subsubtype"] = df_crosswalk["subtype"].apply(
    update_subsubtype)
df_crosswalk["updated_subsubtype"] =
↪  df_crosswalk["updated_subsubtype"].str.capitalize()
```

c.

```
# Reclassify missing subtypes in original dataset
df_waze["subtype"] = df_waze["subtype"].fillna("Unclassified")

# Merge crosswalk with original df
df_merged = df_waze.merge(df_crosswalk, on=["type", "subtype"])

# Count number of rows for Accident - Unclassified
updated_type_subtype_count = df_merged.value_counts(
    ["updated_type", "updated_subtype"])
updated_type_subtype_count.get(("Accident", "Unclassified"))
```

```
np.int64(24359)
```

There are 24,359 rows tagged as Accident - Unclassified.

d.

```
# Compare "type" values for crosswalk and merged dfs
crosswalk_types = set(df_crosswalk["type"].unique())
merged_types = set(df_merged["type"].unique())

crosswalk_types == merged_types
```

```
True
```

```
# Compare "subtype" values for crosswalk and merged dfs
crosswalk_subtypes = set(df_crosswalk["subtype"].unique())
merged_subtypes = set(df_merged["subtype"].unique())

crosswalk_subtypes == merged_subtypes
```

```
True
```

Since both checks return "True" as shown above, this confirms that the crosswalk and merged datasets have the same values in type and subtype.

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

1.

a.

```
# Extract coordinates from geo data
coordinate_pattern = r"([-+]?\d+\.\d+)\s+([-+]?\d+\.\d+)"
df_merged[["longitude", "latitude"]
          ] = df_merged["geo"].str.extract(coordinate_pattern)

# Convert coordinates to float
df_merged["latitude"] = df_merged["latitude"].astype(float)
df_merged["longitude"] = df_merged["longitude"].astype(float)
```

ChatGPT prompt: in python, create a regular expression that extracts coordinates from a geo variable in a dataframe that holds coordinates data in the form POINT(-87.676685 41.929692)

b.

```
# Bin coordinates by rounding
df_merged["latitude"] = df_merged["latitude"].round(2)
df_merged["longitude"] = df_merged["longitude"].round(2)

# Count observations per coordinate pair
df_coord_count = df_merged.groupby(
    ["latitude", "longitude"]).size().reset_index()
df_coord_count.sort_values(by=0, ascending=False, inplace=True)
df_coord_count.head(1)
```

	latitude	longitude	0
396	41.88	-87.65	21325

The binned coordinates (-87.65, 41.88) has the greatest number of observations in the dataset with 21,325.

c.

```
# Aggregate binned counts based on type, subtype, and coordinates
df_coord_count = df_merged.groupby(
    ["updated_type", "updated_subtype", "latitude",
 ↪  "longitude"]).size().reset_index()
df_coord_count = df_coord_count.rename(columns={0: "count"})

# Aggregate by type and subtype and get top 10 largest rows by count
df_coord_count_top10 = df_coord_count.groupby(
    ["updated_type", "updated_subtype"], group_keys=False).apply(lambda x:
 ↪  x.nlargest(10, "count"))

# Save to csv
folder = "top_alerts_map"
filepath = os.path.join(base, folder, "top_alerts_map.csv")
df_coord_count_top10.to_csv(filepath, index=False)

# Get number of rows
len(df_coord_count_top10)
```

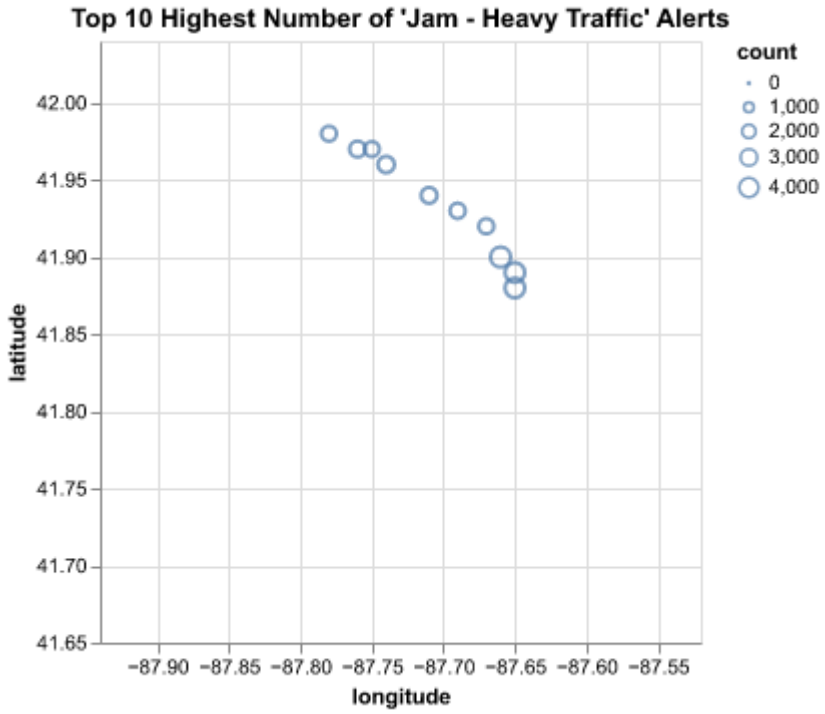
```
C:\Users\LUIS\AppData\Local\Temp\ipykernel_41552\4116047037.py:8:
DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns.
This behavior is deprecated, and in a future version of pandas the grouping
columns will be excluded from the operation. Either pass
`include_groups=False` to exclude the groupings or explicitly select the
grouping columns after groupby to silence this warning.
  ["updated_type", "updated_subtype"], group_keys=False).apply(lambda x:
  x.nlargest(10, "count"))
```

```
155
```

The dataframe has 155 rows. Aggregation was first done based on type, subtype, latitude, and longitude to get the number of observations for each unique combination. Then we further aggregate the data to get the top 10 rows for each unique combination of type and subtype based on count.

2.

```
# Set max and min values
# Constants manually set to correspond with graph produced in app
x_min = min(df_coord_count_top10["longitude"]) - 0.1
x_max = max(df_coord_count_top10["longitude"]) + 0.06
y_min = min(df_coord_count_top10["latitude"])
y_max = max(df_coord_count_top10["latitude"]) + 0.05

# Filter data for "Jam - Heavy Traffic" alerts
df_jam_heavy = df_coord_count_top10.loc[
    (df_coord_count_top10["updated_type"] == "Jam") &
 ↪  (df_coord_count_top10["updated_subtype"] == "Heavy_traffic")]

# Create scatter plot
scatter_plot = alt.Chart(df_jam_heavy).mark_point().encode(
    x=alt.X("longitude:Q", scale=alt.Scale(domain=[x_min, x_max])),
    y=alt.Y("latitude:Q", scale=alt.Scale(domain=[y_min, y_max])),
    size=alt.Size("count:Q", scale=alt.Scale(range=[1, 100]))
).properties(
  title="Top 10 Highest Number of 'Jam - Heavy Traffic' Alerts"
)

scatter_plot
```



3.

a.

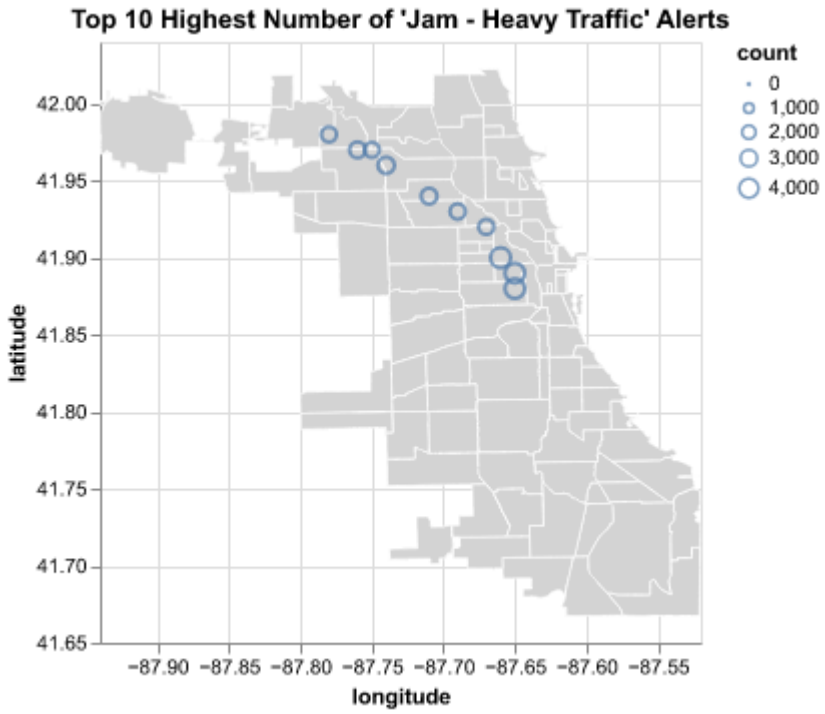
```
# Send a get request
url =
 ↪  "https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON"
response = requests.get(url)

# Set filepath
folder = "top_alerts_map"
filepath = os.path.join(base, folder, "chicago-boundaries.geojson")

# Save file to local folder
if response.status_code == 200:
    with open(filepath, "wb") as file:
        file.write(response.content)
```

b.

```
# Load geojson file
folder = "top_alerts_map"
filepath = os.path.join(base, folder, "chicago-boundaries.geojson")

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

geo_data = alt.Data(values=chicago_geojson,
                    format=alt.DataFormat(property="features"))
```

4.

```
# Create base map of Chicago
base_chicago = alt.Chart(geo_data).mark_geoshape(
  fill="lightgray",
  stroke="white"
).project(
  type="equirectangular"
)

base_chicago + scatter_plot
```



5.

a.

Figure 1: Dropdown menu

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

b.

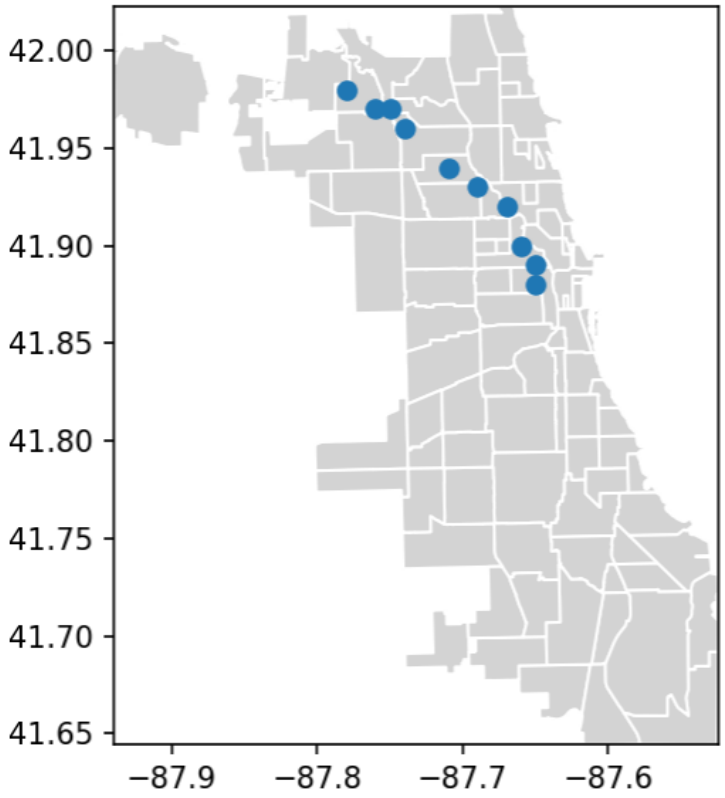


Figure 2: Jam - Heavy Traffic

c.

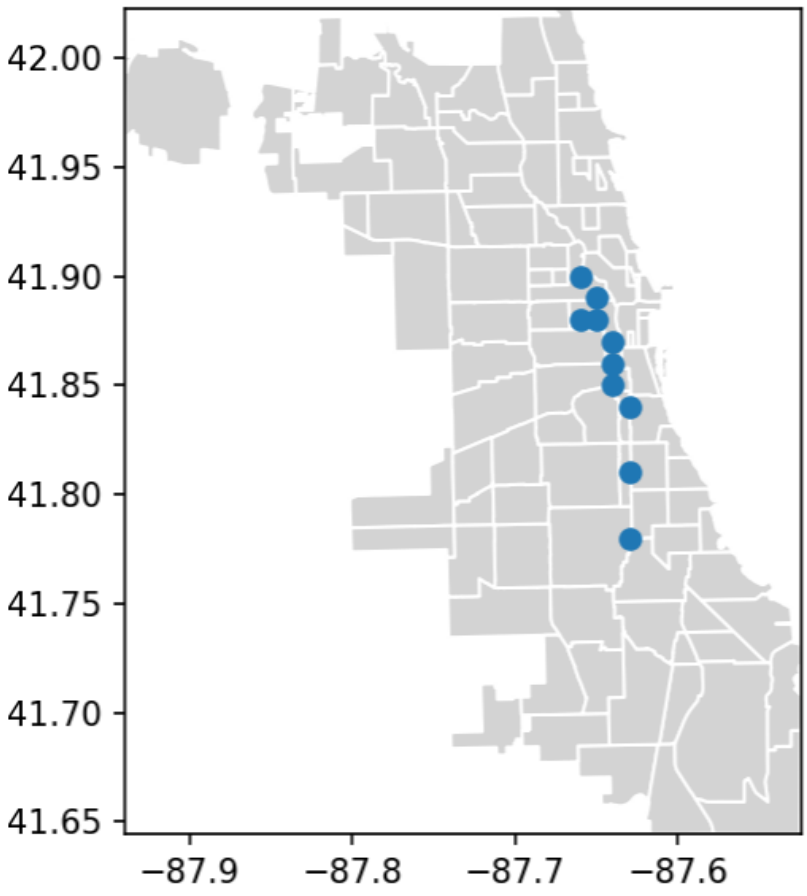


Figure 3: Road Closed - Event

d.

Where are alerts for major accidents the most common?

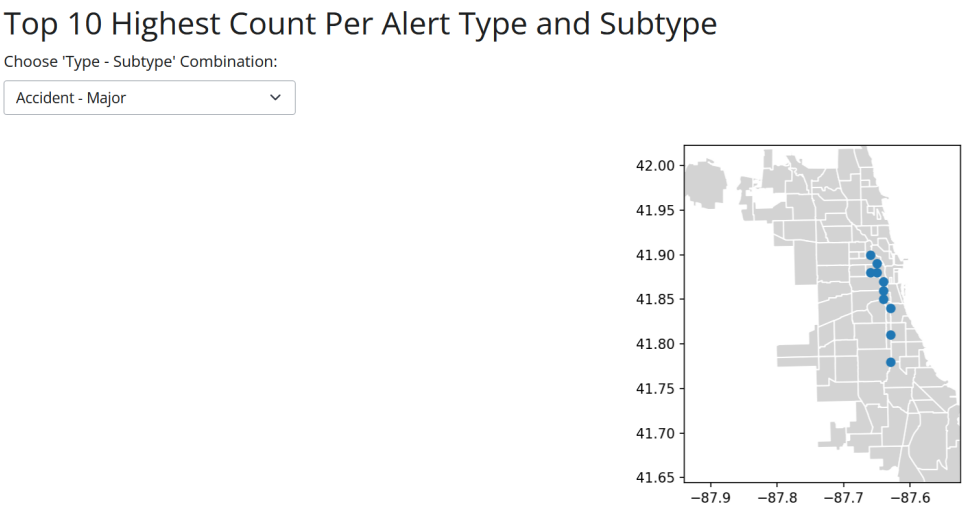


Figure 4: Accident - Major

e.

We can add another column that will display a table containing relevant information such as number of alerts and names of the neighborhoods where the most common alerts occur.

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

1.

a.

No, I think collapsing by the ts column would not be a good idea because there are too many unique values to account for since the column contains combinations of date, hour, minute, and second data.

b.

```
# Convert column values to datetime object
df_merged["hour"] = pd.to_datetime(
    df_merged["ts"], format="%Y-%m-%d %H:%M:%S UTC")

# Extract hour
df_merged["hour"] = df_merged["hour"].dt.strftime("%H:00")
```

```
# Generate new collapsed dataset
df_coord_hourly_count = df_merged.groupby(
    ["updated_type", "updated_subtype", "hour", "latitude",
 ↪  "longitude"]).size().reset_index()
df_coord_hourly_count = df_coord_hourly_count.rename(columns={
    0: "count"})

# Filter df to top 10
df_coord_hourly_count_top10 = df_coord_hourly_count.groupby(
    ["updated_type", "updated_subtype", "hour"],
 ↪  group_keys=False).apply(lambda x: x.nlargest(10, "count"))

# Save to csv
folder = "top_alerts_map_byhour"
filepath = os.path.join(base, folder, "top_alerts_map_byhour.csv")
df_coord_hourly_count_top10.to_csv(filepath, index=False)

# Count rows
len(df_coord_hourly_count_top10)
```

```
C:\Users\LUIS\AppData\Local\Temp\ipykernel_41552\3645951727.py:16:
DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns.
This behavior is deprecated, and in a future version of pandas the grouping
columns will be excluded from the operation. Either pass
`include_groups=False` to exclude the groupings or explicitly select the
grouping columns after groupby to silence this warning.
  ["updated_type", "updated_subtype", "hour"], group_keys=False).apply(lambda
  x: x.nlargest(10, "count"))
```

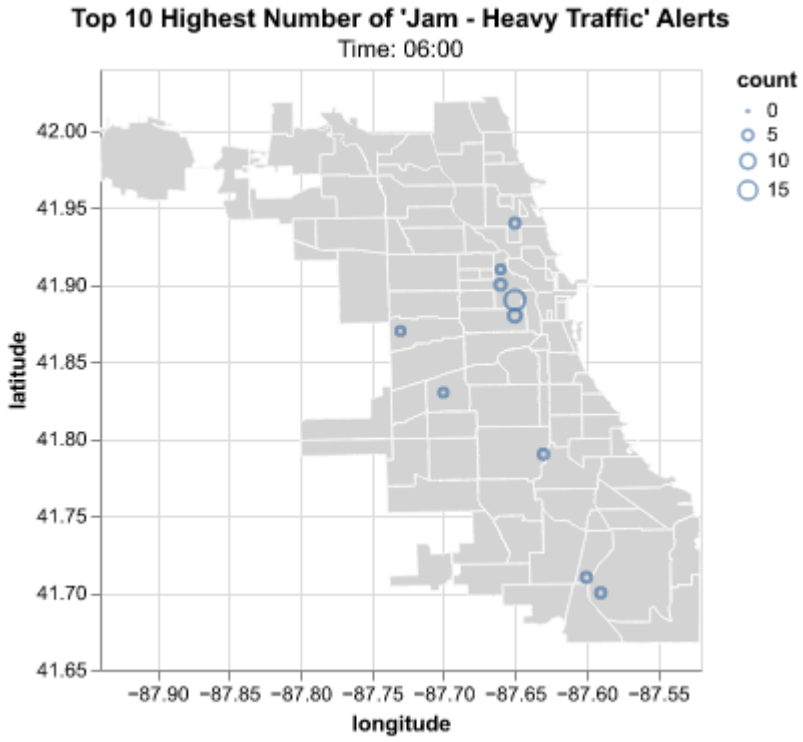
```
3202
```

The new collapsed dataset has 3202 rows.

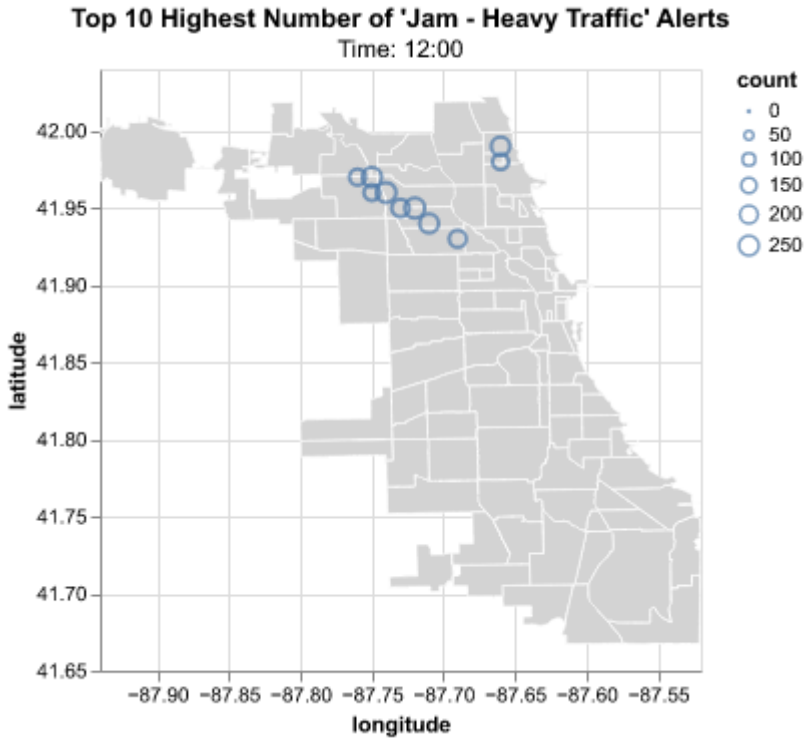
c.

```
# Create list of times
times = ["06:00", "12:00", "18:00"]

# Create empty list of plots
plots = []
```

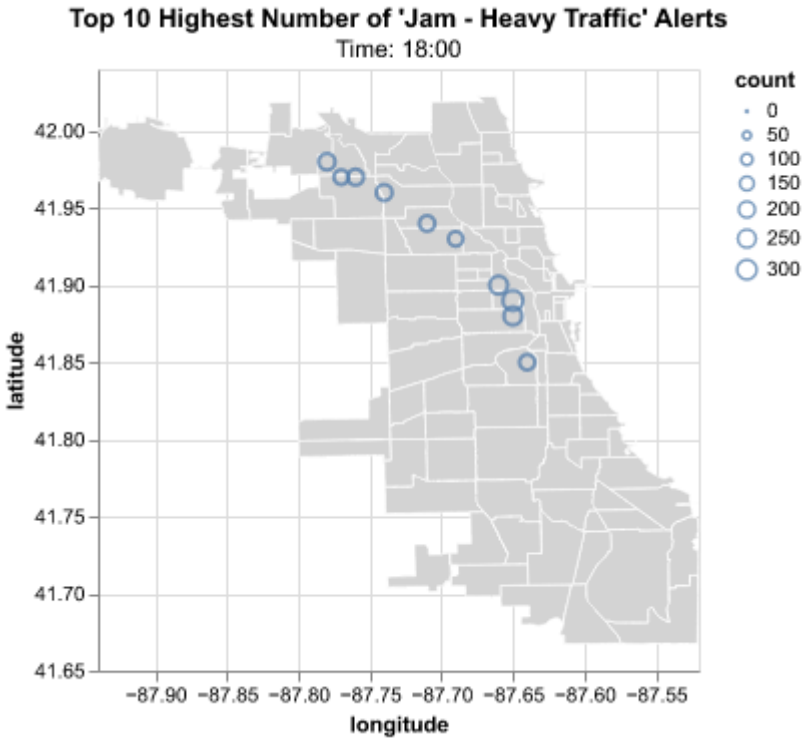
```
# Iterate over times
for time in times:

    # Filter data for "Jam - Heavy Traffic" alerts for each time
    df_jam_heavy_hourly = df_coord_hourly_count_top10.loc[
        (df_coord_hourly_count_top10["updated_type"] == "Jam") &
↪   (df_coord_hourly_count_top10["updated_subtype"] == "Heavy_traffic")]
    df_jam_heavy_hourly = df_jam_heavy_hourly[df_jam_heavy_hourly["hour"] ==
↪   time]

    # Create scatter plot
    scatter_plot_hourly = alt.Chart(df_jam_heavy_hourly).mark_point().encode(
        x=alt.X("longitude:Q", scale=alt.Scale(domain=[x_min, x_max])),
        y=alt.Y("latitude:Q", scale=alt.Scale(domain=[y_min, y_max])),
        size=alt.Size("count:Q", scale=alt.Scale(range=[1, 100]))
    ).properties(
        title={
            "text": "Top 10 Highest Number of 'Jam - Heavy Traffic' Alerts",
            "subtitle": "Time: " + time
        }
    )

    # Append to list of plots
    plots.append(scatter_plot_hourly)

# Add Chicago base map and print plot 1
base_chicago + plots[0]
```




```
# Add Chicago base map and print plot 2
base_chicago + plots[1]
```




```
# Add Chicago base map and print plot 3
base_chicago + plots[2]
```



2.

a.

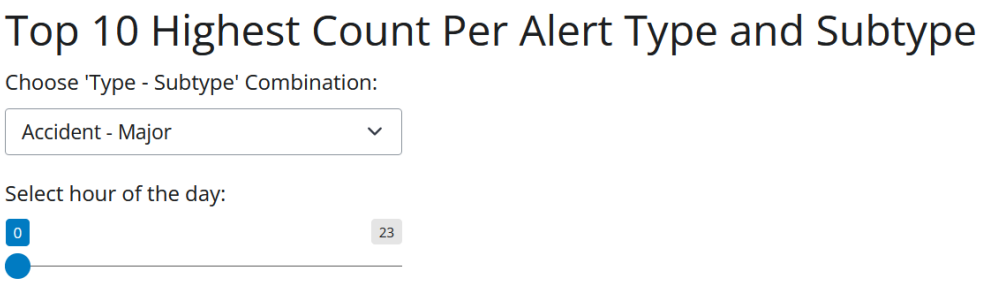


Figure 5: UI - with slider

b.

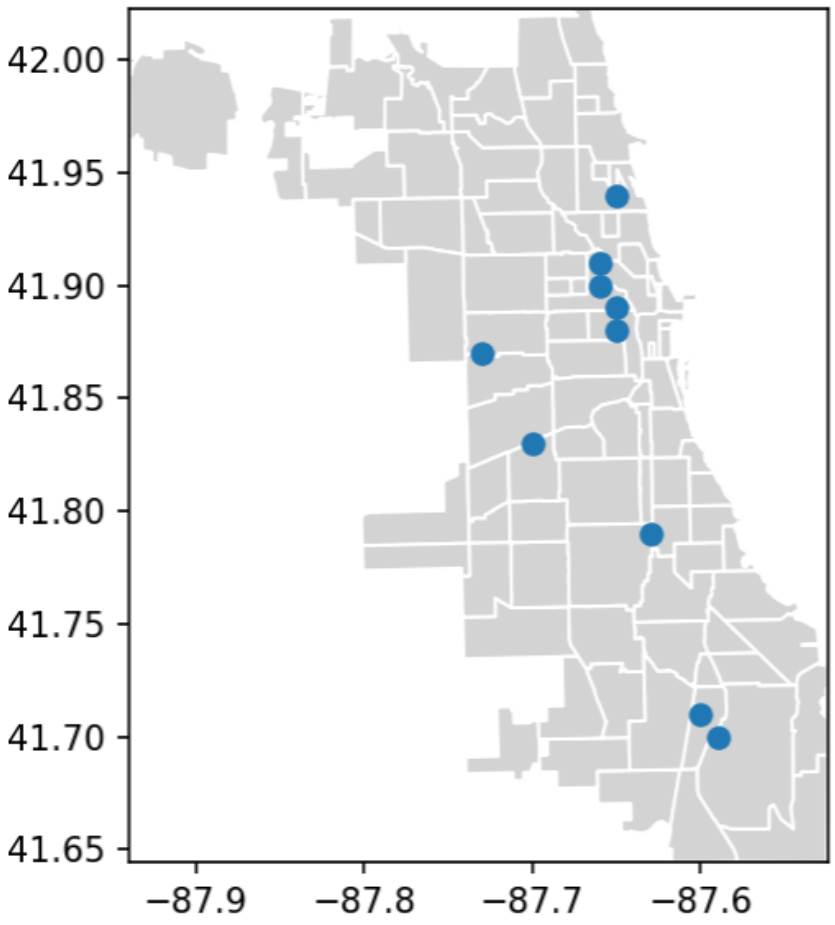


Figure 6: Jam - Heavy Traffic at 6am

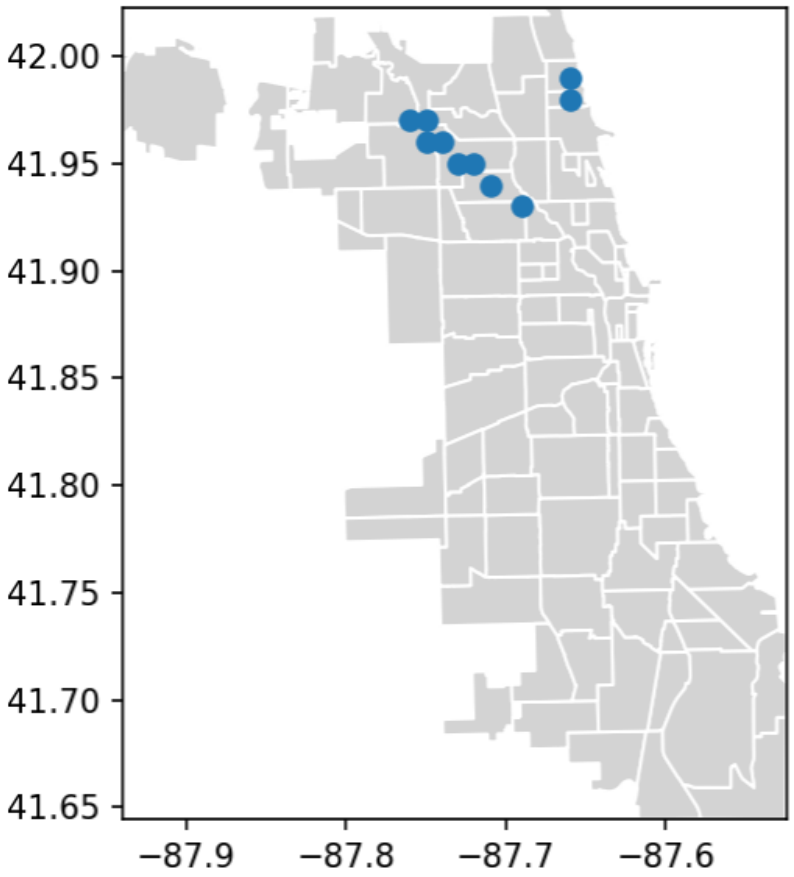


Figure 7: Jam - Heavy Traffic at 12pm

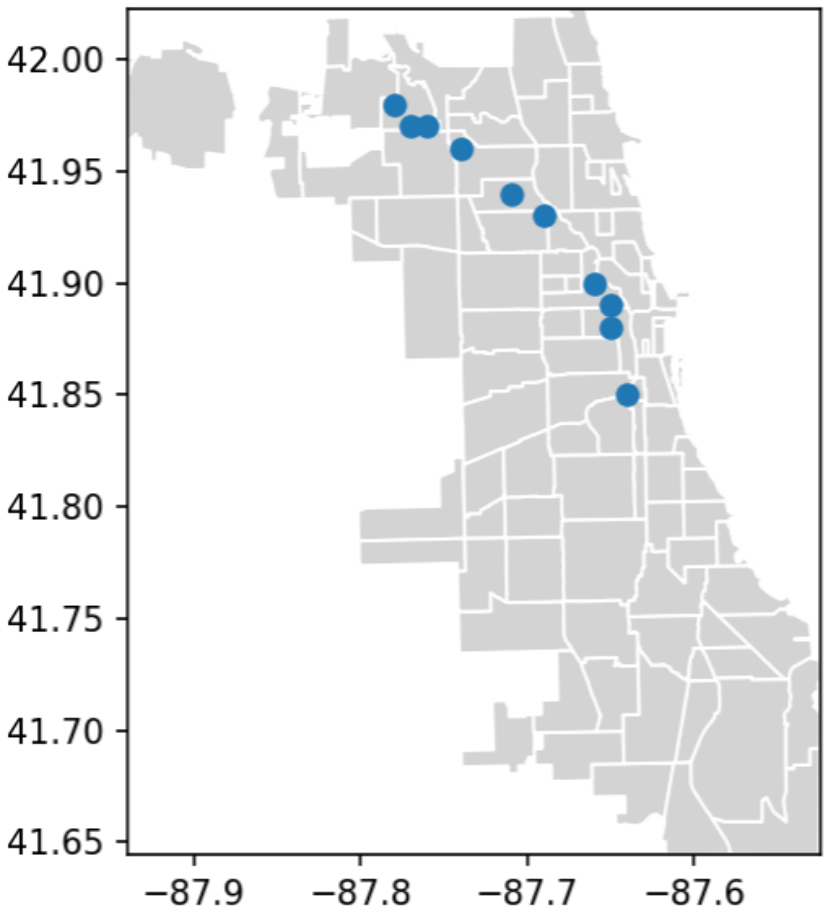


Figure 8: Jam - Heavy Traffic at 6pm

c.

It seems that road construction is done more during night hours.

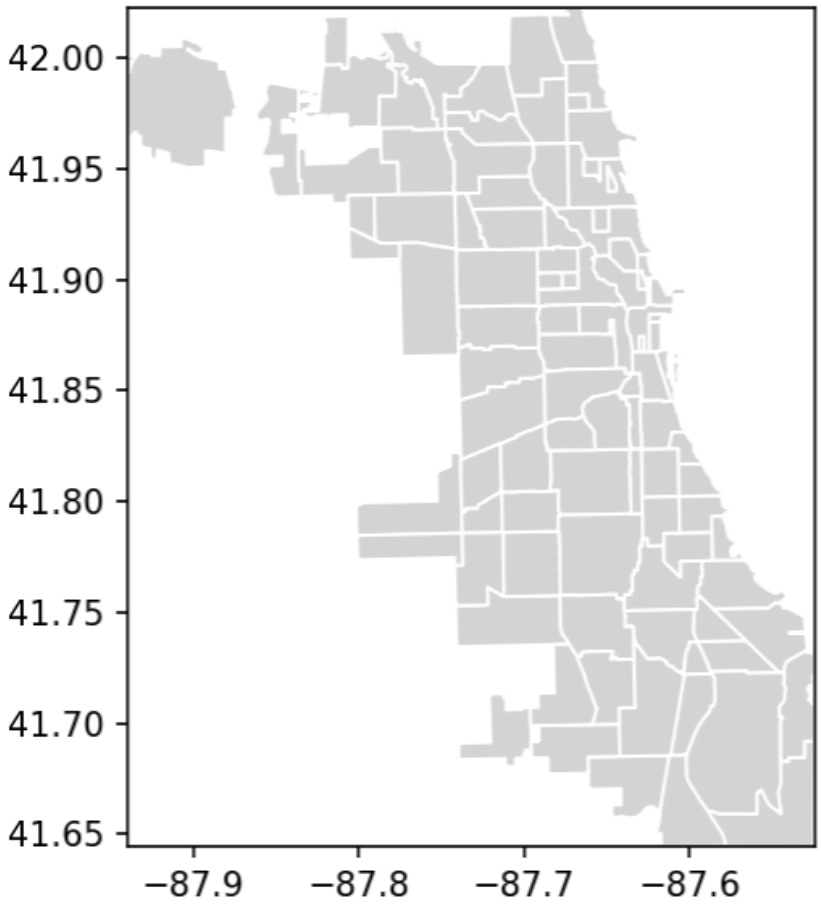


Figure 9: No alerts for road construction at 7am

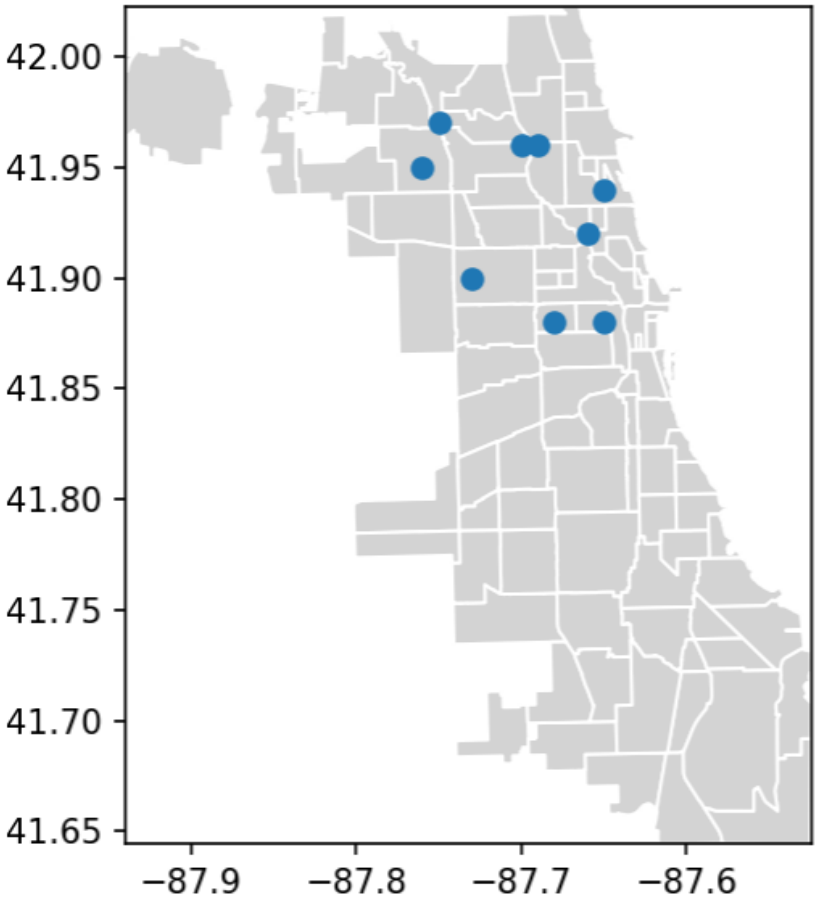


Figure 10: Multiple alerts for road construction at 7pm

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

1.

a.
b.

2.

a.
b.

3.

a.
b.
c.
d.