Problem Set 6 - Waze Shiny Dashboard

Nasser Alshaya

2024-11-20

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: `NA`
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: 2
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 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)

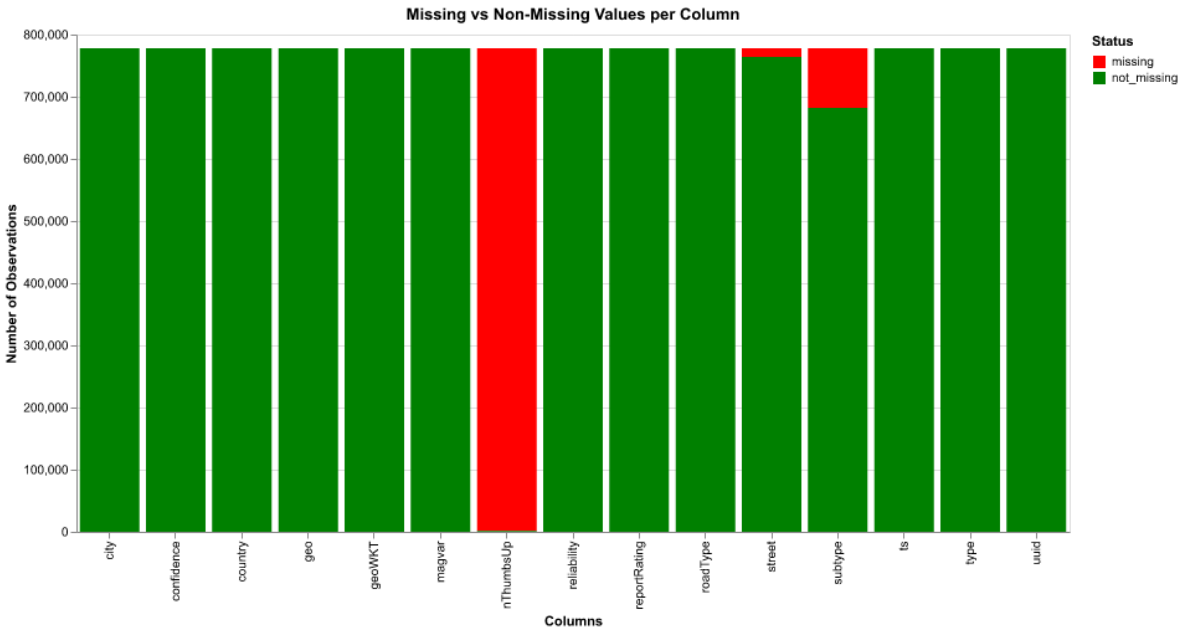
1. Unnamed: 0: Ordinal 'index from original df', city: Nominal, confidence: Ordinal, nThumbsUp: Quantitative, street: Nominal, uuid: Nominal, country: Nominal, type: Nominal, subtype: Nominal, roadType: Nominal, reliability: Ordinal, magvar: Ordinal, reportRating: Ordinal

```
import os
base_path =
 ↪  r"/Users/nasser.alshaya/Desktop/Fall-2024/PPHA-30538/PS6/waze_data"
path_data = os.path.join(base_path,"waze_data_sample.csv")

waze_sample = pd.read_csv(path_data)
waze_sample.columns
```

```
Index(['Unnamed: 0', 'city', 'confidence', 'nThumbsUp', 'street', 'uuid',
       'country', 'type', 'subtype', 'roadType', 'reliability', 'magvar',
       'reportRating', 'ts', 'geo', 'geoWKT'],
      dtype='object')
```

2. nThumbsUp has only 1371 inputs with 776723 NULL values. street has 14073 NULL values. subtype has 96086 NULL values. The remaining variables have no NULL values.

```
path_data = os.path.join(base_path,"waze_data.csv")
df_waze = pd.read_csv(path_data)

# Create a dataframe to count nulls:
nulls_count = pd.DataFrame({
    'variable': df_waze.columns,
    'missing': df_waze.isnull().sum(),
    'not_missing': df_waze.notnull().sum()
})

# Transform to long to add status column to plot:
nulls_count_long = nulls_count.melt(
    id_vars='variable',value_vars=['missing', 'not_missing'],
    var_name='status', value_name='count')

import warnings
warnings.filterwarnings('ignore')
alt.renderers.enable("png")

chart = alt.Chart(nulls_count_long).mark_bar().encode(
    x=alt.X('variable:N', title='Columns'),
    y=alt.Y('count:Q', title='Number of Observations'),
    color=alt.Color('status:N', scale=alt.Scale(domain=['missing',
 ↪  'not_missing'],range=['red', 'green']),title='Status'),
).properties(
    width=800,
    height=400,
    title='Missing vs Non-Missing Values per Column'
)

chart
```



3. All types have missing subtypes, after slicing the dataframe based on missing values in subtypes, the lowest type is Accident with `9178` observations. I believe all hazards have enough information to consider that they could have sub-subtypes without the need to keep NAs.

```
df_waze['type'].unique()
```

```
array(['JAM', 'ACCIDENT', 'ROAD_CLOSED', 'HAZARD'], dtype=object)
```

```
df_waze['subtype'].unique()
```

```
array([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'], dtype=object)
```

```
# Finding types with null subtypes:
missing_subtype = df_waze[df_waze['subtype'].notnull()]
missing_subtype['type'].unique()
```

```
array(['ACCIDENT', 'HAZARD', 'JAM', 'ROAD_CLOSED'], dtype=object)
```

```
# Identify which types should have subtypes:
types_cols = ['JAM', 'ACCIDENT', 'ROAD_CLOSED', 'HAZARD']
types = pd.DataFrame(
    {'count': [len(missing_subtype[missing_subtype['type']=='JAM']),
               len(missing_subtype[missing_subtype['type']=='ACCIDENT']),
               len(missing_subtype[missing_subtype['type']=='ROAD_CLOSED']),
               len(missing_subtype[missing_subtype['type']=='HAZARD'])]},
    index = types_cols
)
types
```

	count
JAM	317444
ACCIDENT	9178
ROAD_CLOSED	42535
HAZARD	312851

`bulleted liste`: 1 Accident: 1.1 Major 1.2 Minor 2 Hazard: 2.1 Road Hazards 2.1.1 Object on Road 2.1.2 Pothole 2.1.3 Traffic Light Fault 2.1.4 Road Kill 2.1.5 Lane Closed 2.1.6 Emergency Vehicle 2.1.7 Ice 2.2 Shoulder Hazards 2.2.1 Car Stopped 2.2.2 Animals 2.2.3 Missing Sign 2.3 Weather Hazards 2.3.1 Fog 2.3.2 Heavy Snow 2.3.3 Hail 2.3.4 Flood 2.3.5 Ice 3 Traffic Jams 3.1 Heavy Traffic 3.2 Moderate Traffic 3.3 Light Traffic 3.3 Stand Still Traffic 4 Road Closures 4.1 Construction 4.1 Hazard-Related Closure 4.3 Event-Related Closure

4.

a. The new dataframe is shown below and will be completed in part b

```
# Unique combinations of type and subtype
unique_combinations = df_waze[['type', 'subtype']].drop_duplicates()

df_crosswalk = pd.DataFrame({
    'type': unique_combinations['type'],
    'subtype': unique_combinations['subtype'],
    'updated_type': None,
    'updated_subtype': None,
    'updated_subsubtype': None
})

df_crosswalk.reset_index(drop=True, inplace=True)

df_crosswalk.head()
```

	type	subtype	updated_type	updated_subtype	updated_subsubtype
0	JAM	NaN	None	None	None
1	ACCIDENT	NaN	None	None	None
2	ROAD_CLOSED	NaN	None	None	None
3	HAZARD	NaN	None	None	None
4	ACCIDENT	ACCIDENT_MAJOR	None	None	None

b. Crosswalk dataframe with replacing all NaN with unclassified is shown below

```
def crosswalk_builder(df):

    for index, row in df.iterrows():
        type = row['type'].replace('_', ' ').title()
        subtype = row['subtype']

        if pd.isna(subtype) or not isinstance(subtype, str):
            df.at[index, 'updated_type'] = type
            df.at[index, 'subtype'] = 'Unclassified'
            df.at[index, 'updated_subtype'] = 'Unclassified'
            df.at[index, 'updated_subsubtype'] = 'Unclassified'
            continue

        if 'ACCIDENT_MAJOR' in subtype:
            df.at[index, 'updated_type'] = 'Accident'
            df.at[index, 'updated_subtype'] = 'Major'
```

```
    if 'ACCIDENT_MINOR' in subtype:
        df.at[index, 'updated_type'] = 'Accident'
        df.at[index, 'updated_subtype'] = 'Minor'

    if 'HAZARD_ON_ROAD' in subtype:
        df.at[index, 'updated_type'] = 'Hazard'
        df.at[index, 'updated_subtype'] = 'Road Hazards'
        df.at[index, 'updated_subsubtype'] = 'Unclassified'

    if 'HAZARD_ON_ROAD_ICE' in subtype:
        df.at[index, 'updated_type'] = 'Hazard'
        df.at[index, 'updated_subtype'] = 'Road Hazards'
        df.at[index, 'updated_subsubtype'] = 'Ice'

    if 'HAZARD_ON_ROAD_OBJECT' in subtype:
        df.at[index, 'updated_type'] = 'Hazard'
        df.at[index, 'updated_subtype'] = 'Road Hazards'
        df.at[index, 'updated_subsubtype'] = 'Object on Road'

    if 'HAZARD_ON_ROAD_POT_HOLE' in subtype:
        df.at[index, 'updated_type'] = 'Hazard'
        df.at[index, 'updated_subtype'] = 'Road Hazards'
        df.at[index, 'updated_subsubtype'] = 'Pothole'

    if 'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT' in subtype:
        df.at[index, 'updated_type'] = 'Hazard'
        df.at[index, 'updated_subtype'] = 'Road Hazards'
        df.at[index, 'updated_subsubtype'] = 'Traffic Light Fault'

    if 'HAZARD_ON_ROAD_ROAD_KILL' in subtype:
        df.at[index, 'updated_type'] = 'Hazard'
        df.at[index, 'updated_subtype'] = 'Road Hazards'
        df.at[index, 'updated_subsubtype'] = 'Road Kill'

    if 'HAZARD_ON_ROAD_LANE_CLOSED' in subtype:
        df.at[index, 'updated_type'] = 'Hazard'
        df.at[index, 'updated_subtype'] = 'Road Hazards'
        df.at[index, 'updated_subsubtype'] = 'Lane Closed'

    if 'HAZARD_ON_ROAD_EMERGENCY_VEHICLE' in subtype:
        df.at[index, 'updated_type'] = 'Hazard'
```

```
            df.at[index, 'updated_subtype'] = 'Road Hazards'
            df.at[index, 'updated_subsubtype'] = 'Emergency Vehicle'

        if 'HAZARD_ON_SHOULDER' in subtype:
            df.at[index, 'updated_type'] = 'Hazard'
            df.at[index, 'updated_subtype'] = 'Shoulder Hazards'
            df.at[index, 'updated_subsubtype'] = 'Unclassified'

        if 'HAZARD_ON_SHOULDER_CAR_STOPPED' in subtype:
            df.at[index, 'updated_type'] = 'Hazard'
            df.at[index, 'updated_subtype'] = 'Shoulder Hazards'
            df.at[index, 'updated_subsubtype'] = 'Car Stopped'

        if 'HAZARD_ON_ROAD_CAR_STOPPED' in subtype:
            df.at[index, 'updated_type'] = 'Hazard'
            df.at[index, 'updated_subtype'] = 'Road Hazards'
            df.at[index, 'updated_subsubtype'] = 'Car Stopped'

        if 'HAZARD_ON_SHOULDER_ANIMALS' in subtype:
            df.at[index, 'updated_type'] = 'Hazard'
            df.at[index, 'updated_subtype'] = 'Shoulder Hazards'
            df.at[index, 'updated_subsubtype'] = 'Animals'

        if 'HAZARD_ON_SHOULDER_MISSING_SIGN' in subtype:
            df.at[index, 'updated_type'] = 'Hazard'
            df.at[index, 'updated_subtype'] = 'Shoulder Hazards'
            df.at[index, 'updated_subsubtype'] = 'Missing Sign'

        if 'HAZARD_WEATHER' in subtype:
            df.at[index, 'updated_type'] = 'Hazard'
            df.at[index, 'updated_subtype'] = 'Weather Hazards'
            df.at[index, 'updated_subsubtype'] = 'Unclassified'

        if 'HAZARD_WEATHER_FOG' in subtype:
            df.at[index, 'updated_type'] = 'Hazard'
            df.at[index, 'updated_subtype'] = 'Weather Hazards'
            df.at[index, 'updated_subsubtype'] = 'Fog'

        if 'HAZARD_WEATHER_HEAVY_SNOW' in subtype:
            df.at[index, 'updated_type'] = 'Hazard'
            df.at[index, 'updated_subtype'] = 'Weather Hazards'
            df.at[index, 'updated_subsubtype'] = 'Heavy Snow'
```

```
    if 'HAZARD_WEATHER_HAIL' in subtype:
        df.at[index, 'updated_type'] = 'Hazard'
        df.at[index, 'updated_subtype'] = 'Weather Hazards'
        df.at[index, 'updated_subsubtype'] = 'Hail'

    if 'HAZARD_WEATHER_FLOOD' in subtype:
        df.at[index, 'updated_type'] = 'Hazard'
        df.at[index, 'updated_subtype'] = 'Weather Hazards'
        df.at[index, 'updated_subsubtype'] = 'Flood'

    if 'HAZARD_WEATHER_ICE' in subtype:
        df.at[index, 'updated_type'] = 'Hazard'
        df.at[index, 'updated_subtype'] = 'Weather Hazards'
        df.at[index, 'updated_subsubtype'] = 'Ice'

    if 'JAM_HEAVY_TRAFFIC' in subtype:
        df.at[index, 'updated_type'] = 'Jam'
        df.at[index, 'updated_subtype'] = 'Heavy Traffic'

    if 'JAM_MODERATE_TRAFFIC' in subtype:
        df.at[index, 'updated_type'] = 'Jam'
        df.at[index, 'updated_subtype'] = 'Moderate Traffic'

    if 'JAM_LIGHT_TRAFFIC' in subtype:
        df.at[index, 'updated_type'] = 'Jam'
        df.at[index, 'updated_subtype'] = 'Light Traffic'

    if 'JAM_STAND_STILL_TRAFFIC' in subtype:
        df.at[index, 'updated_type'] = 'Jam'
        df.at[index, 'updated_subtype'] = 'Stand Still Traffic'

    if 'ROAD_CLOSED_CONSTRUCTION' in subtype:
        df.at[index, 'updated_type'] = 'Road Closures'
        df.at[index, 'updated_subtype'] = 'Construction'

    if 'ROAD_CLOSED_HAZARD' in subtype:
        df.at[index, 'updated_type'] = 'Road Closures'
        df.at[index, 'updated_subtype'] = 'Hazard-Related Closure'

    if 'ROAD_CLOSED_EVENT' in subtype:
        df.at[index, 'updated_type'] = 'Road Closures'
```

```
            df.at[index, 'updated_subtype'] = 'Event-Related Closure'
    return df

df_crosswalk = crosswalk_builder(df_crosswalk)

# Filling any remaining NaN with 'Unclassified'
df_crosswalk[['updated_type', 'updated_subtype',
 'updated_subsubtype']] = df_crosswalk[['updated_type',
  'updated_subtype', 'updated_subsubtype']].fillna('Unclassified')

df_crosswalk['updated_type'] = df_crosswalk['updated_type'].replace('Road
 ↪  Closed', 'Road Closures')

df_crosswalk.head()
```

	type	subtype	updated_type	updated_subtype	updated_subsubtype
0	JAM	Unclassified	Jam	Unclassified	Unclassified
1	ACCIDENT	Unclassified	Accident	Unclassified	Unclassified
2	ROAD_CLOSED	Unclassified	Road Closures	Unclassified	Unclassified
3	HAZARD	Unclassified	Hazard	Unclassified	Unclassified
4	ACCIDENT	ACCIDENT_MAJOR	Accident	Major	Unclassified

c. There are 24359 Accidents with unclassified subtype.

```
# To properly merge change NaN values to Unclassified
df_waze['subtype'] = df_waze['subtype'].fillna('Unclassified')
df_merged = df_waze.merge(
    df_crosswalk, on = ['type','subtype'],how = 'left')

len(df_merged[(
    df_merged['updated_type'] == 'Accident') &
    (df_merged['updated_subtype'] == 'Unclassified')])
```

```
24359
```

d. The values match between the two dataframes

```
set(df_merged['updated_type'].unique()) ==
 ↪  set(df_crosswalk['updated_type'].unique())
```

```
True
```

```
set(df_merged['updated_subtype'].unique()) ==
 ↪  set(df_crosswalk['updated_subtype'].unique())
```

```
True
```

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

1.

a.

```
import re

# Function to extract latitude and longitude
def extract_coordinates(geo_string):
    # Use regex to extract numbers from the POINT string
    match = re.search(r"POINT\s*\(\s*([\-\d\.]+)\s+([\-\d\.]+)\s*\)",
 ↪  geo_string)
    if match:
        longitude = float(match.group(1))
        latitude = float(match.group(2))
        return latitude, longitude
    else:
        return None, None

# Apply the function to the geo column
df_merged['latitude'], df_merged['longitude'] = zip(
    *df_merged['geo'].apply(extract_coordinates))

# Check if extraction worked
df_merged[['geo', 'latitude', 'longitude']].head()
```

	geo	latitude	longitude
0	POINT(-87.676685 41.929692)	41.929692	-87.676685
1	POINT(-87.624816 41.753358)	41.753358	-87.624816
2	POINT(-87.614122 41.889821)	41.889821	-87.614122
3	POINT(-87.680139 41.939093)	41.939093	-87.680139

	geo	latitude	longitude
4	POINT(-87.735235 41.91658)	41.916580	-87.735235

Attribution: propmted Chatgbt to extract latitude and longitude from my dataframe: 'using regrex my geo column values look like this :Point(-87.647848 41.967935), create two variables latitude and longitude after extracting the latitude and longitude from the string'

b. bin(41.88, -87.65) has the maximum number of observations with 21325 counts.

```
# Bin latitude and longitude to 2 decimal places
df_merged['latitude'] = df_merged['latitude'].round(2)
df_merged['longitude'] = df_merged['longitude'].round(2)

# Finding binned combination with max observations
bin_counts = df_merged.groupby(['latitude',
 ↪  'longitude']).size().reset_index(name='count')

bin_counts.loc[bin_counts['count'].idxmax()]
```

```
latitude        41.88
longitude      -87.65
count        21325.00
Name: 396, dtype: float64
```

c. The level of aggregation is based on the top 10 ranked types and subtypes based on the count of their incidents with respect to their longitude and latitude. I ended up with a DataFrame with 155 rows.

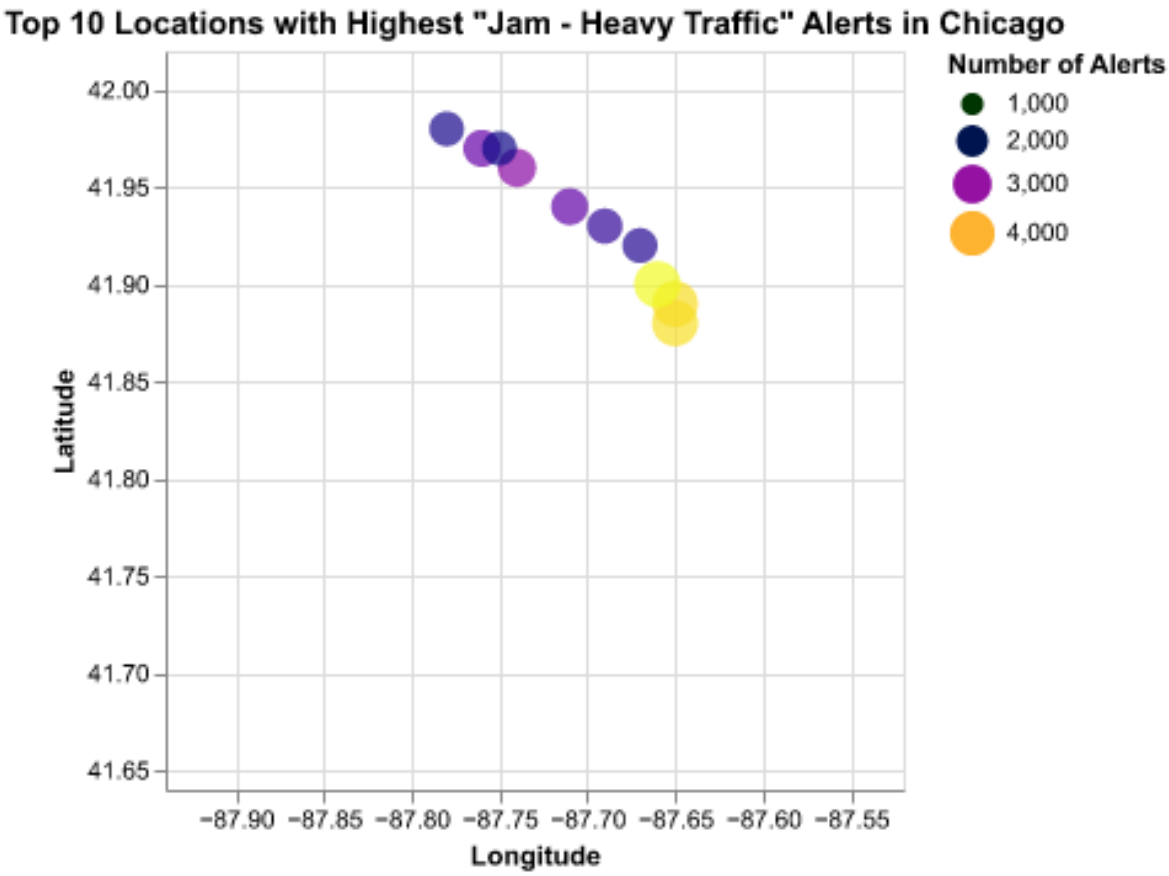
```
top_alerts = (df_merged.groupby(['latitude',
     'longitude', 'updated_type', 'updated_subtype']).size()
    .reset_index(name ='count'))

# Ranking top types and subtypes
top_alerts['rank'] = top_alerts.groupby(
    ['updated_type', 'updated_subtype'])['count'].rank(
    method ='first', ascending = False)

# Keep only top 10 ranks
top_alerts_map = top_alerts[
    top_alerts['rank'] <= 10].drop(columns ='rank').reset_index(drop = True)
```

```
top_alerts_map.to_csv('top_alerts_map.csv', index = True)

len(top_alerts_map)
```

```
155
```

d.

```
scatter_plot = alt.Chart(top_alerts_map).mark_circle().transform_filter(
    (alt.datum.updated_type == 'Jam') &
    (alt.datum.updated_subtype == 'Heavy Traffic')).encode(
    x=alt.X('longitude:Q', title='Longitude', scale = alt.Scale(domain =
↪   [-87.94, -87.52])),
    y = alt.Y('latitude:Q', title='Latitude', scale = alt.Scale(domain =
↪   [41.64, 42.02])),
    size = alt.Size('count:Q', title = 'Number of Alerts', legend =
↪   alt.Legend(orient='right')),
    color = alt.Color('count:Q', scale=alt.Scale(scheme ='plasma')),
    tooltip = ['latitude:Q', 'longitude:Q', 'count:Q', 'updated_type:N',
↪   'updated_subtype:N']).properties(
    width = 300,
    height = 300,
    title='Top 10 Locations with Highest "Jam - Heavy Traffic" Alerts in
↪   Chicago')

scatter_plot
```



3.

a.

```
import requests

url =
 ↪  "https://data.cityofchicago.org/api/views/y6yq-dbs2/rows.json?accessType=DOWNLOAD"

response = requests.get(url)

if response.status_code == 200:
    data = response.json()
    with open('chicago_neighborhoods.geojson', 'w') as f:
        json.dump(data, f)
    print("GeoJSON file downloaded successfully!")
```

```
else:
    print("Request failed with status code:", response.status_code)
```

```
GeoJSON file downloaded successfully!
```

b.

```
# MODIFY ACCORDINGLY
file_path =
 ↪  "/Users/nasser.alshaya/Desktop/Fall-2024/PPHA-30538/PS6/top_alerts_map/chicago-boundaries
#----

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

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

```
import geopandas as gpd
geo_data = gpd.read_file(file_path)
geo_data.total_bounds
```

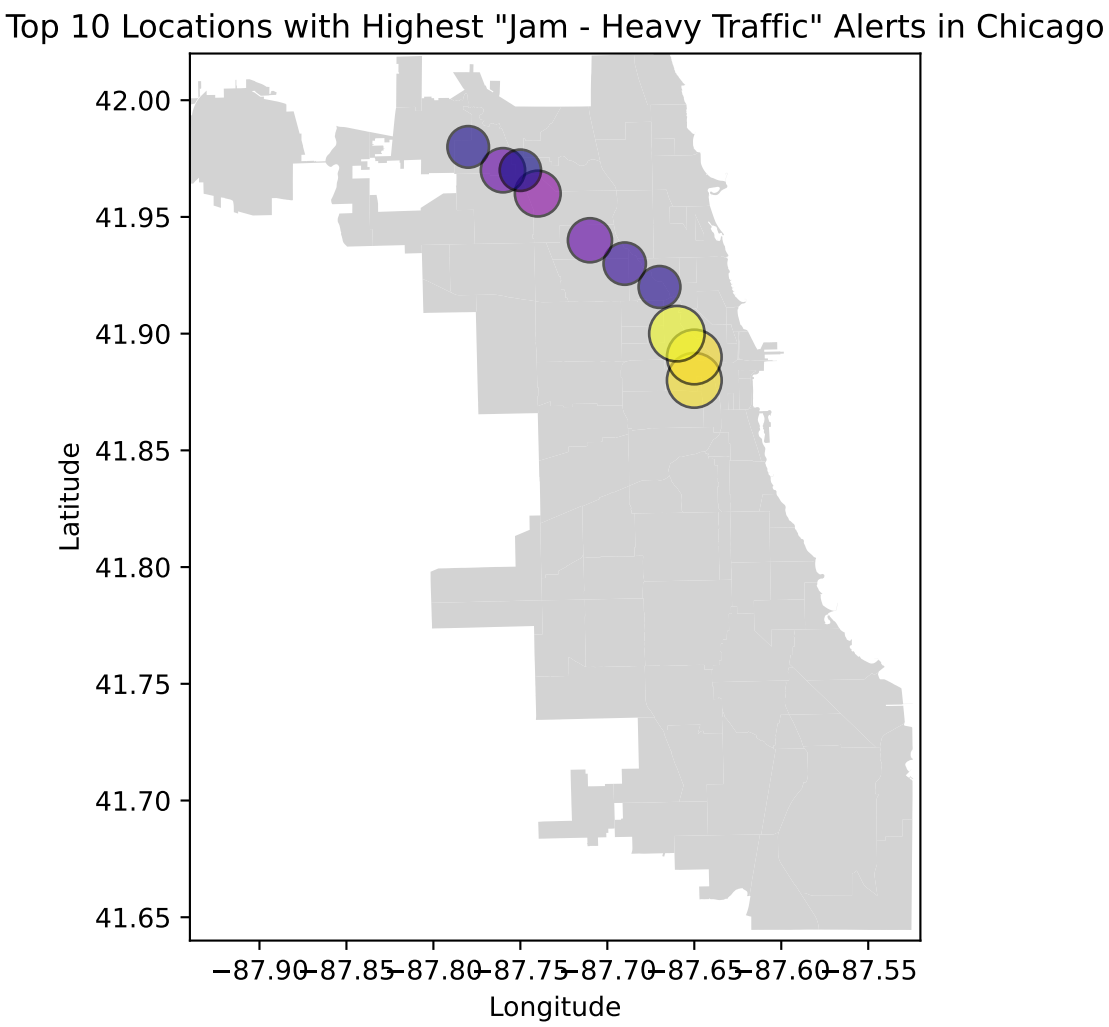
```
array([-87.94011408,  41.64454312, -87.5241371 ,  42.02303859])
```

4.

```
map = alt.Chart(geo_data).mark_geoshape().encode(
    tooltip=['properties.name:N']
).properties(
    width = 300,
    height = 300
)

map + scatter_plot
```

```
# Note: due to errors in rendering altair, the app is rendering matplotlib
 ↪  instead
filtered_data = top_alerts_map[
    (top_alerts_map['updated_type'] == 'Jam') &
    (top_alerts_map['updated_subtype'] == 'Heavy Traffic')
]
```

```
fig, ax = plt.subplots(figsize=(6, 6))

geo_data.plot(ax=ax, color='lightgray')

scatter = ax.scatter(
    filtered_data['longitude'],
    filtered_data['latitude'],
    s=filtered_data['count'] * 0.1,
    c=filtered_data['count'],
    cmap='plasma',
    alpha=0.6,
    edgecolors='k'
)

ax.set_xlim(-87.94, -87.52)
ax.set_ylim(41.64, 42.02)

ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
ax.set_title('Top 10 Locations with Highest "Jam - Heavy Traffic" Alerts in
 ↪  Chicago')

plt.show()
```



5.

a. There are 16 type subtype combination

```
# Create a combined column for dropdown selection
top_alerts_map['type_subtype'] = (top_alerts_map['updated_type'] +
" - " + top_alerts_map['updated_subtype'])
type_combinations = top_alerts_map['type_subtype'].unique()
len(type_combinations)
```

```
16
```

```
app_ui = ui.page_fluid(
    ui.input_select(
        id = "type_subtype",
        label = "Choose a combination:",
        choices = []),
    ui.output_plot("ts"),
)
```

Figure 1: APP1_UI

b.

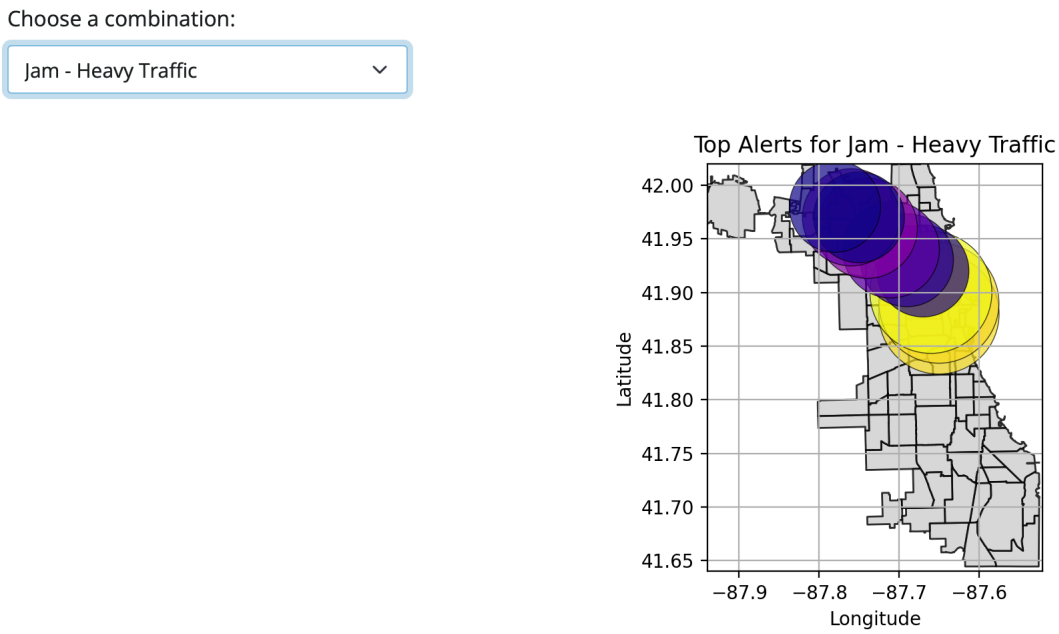


Figure 2: Jam - Heavy Traffic plot

c. Where do construction works result in road closures the most? It seems there are more road closures up north.

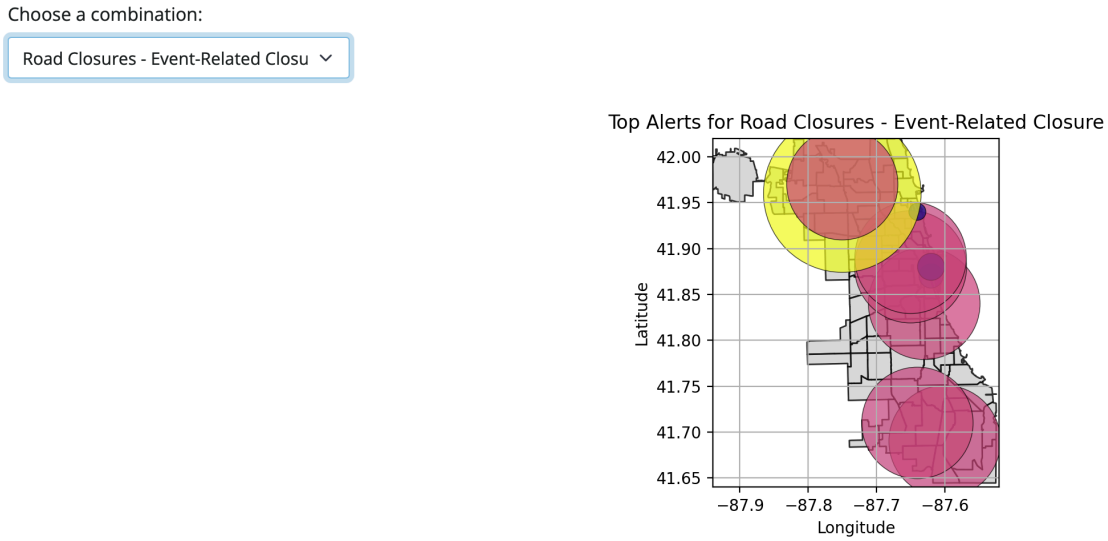
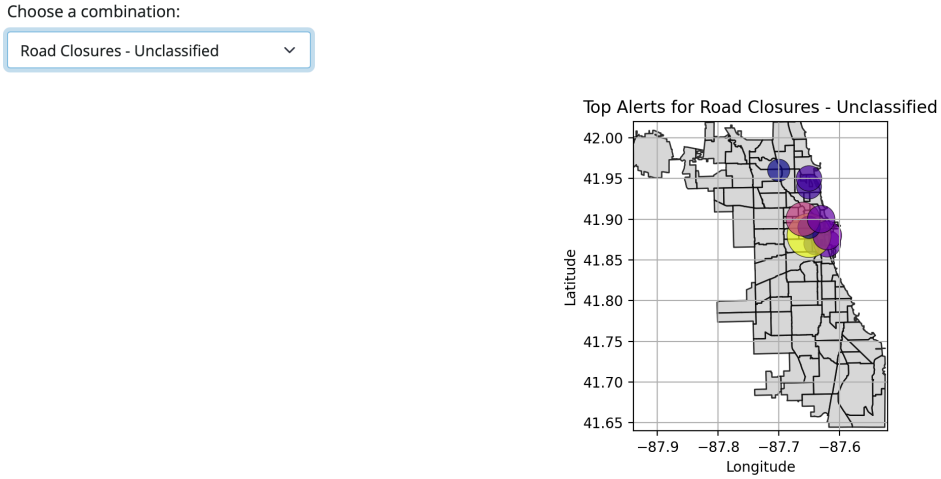


Figure 3: Road_Closure_Events

d. Where do unclassified road closures occur the most? West loop and downtwon



e. Adding a time column to the columns will enhance the information about the incidents.

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

1.

a. Yes, I believe adding the time domain to our analysis will provide a more accurate indication of the status of road traffic in the city.

b. There are 3202 rows in this dataframe.

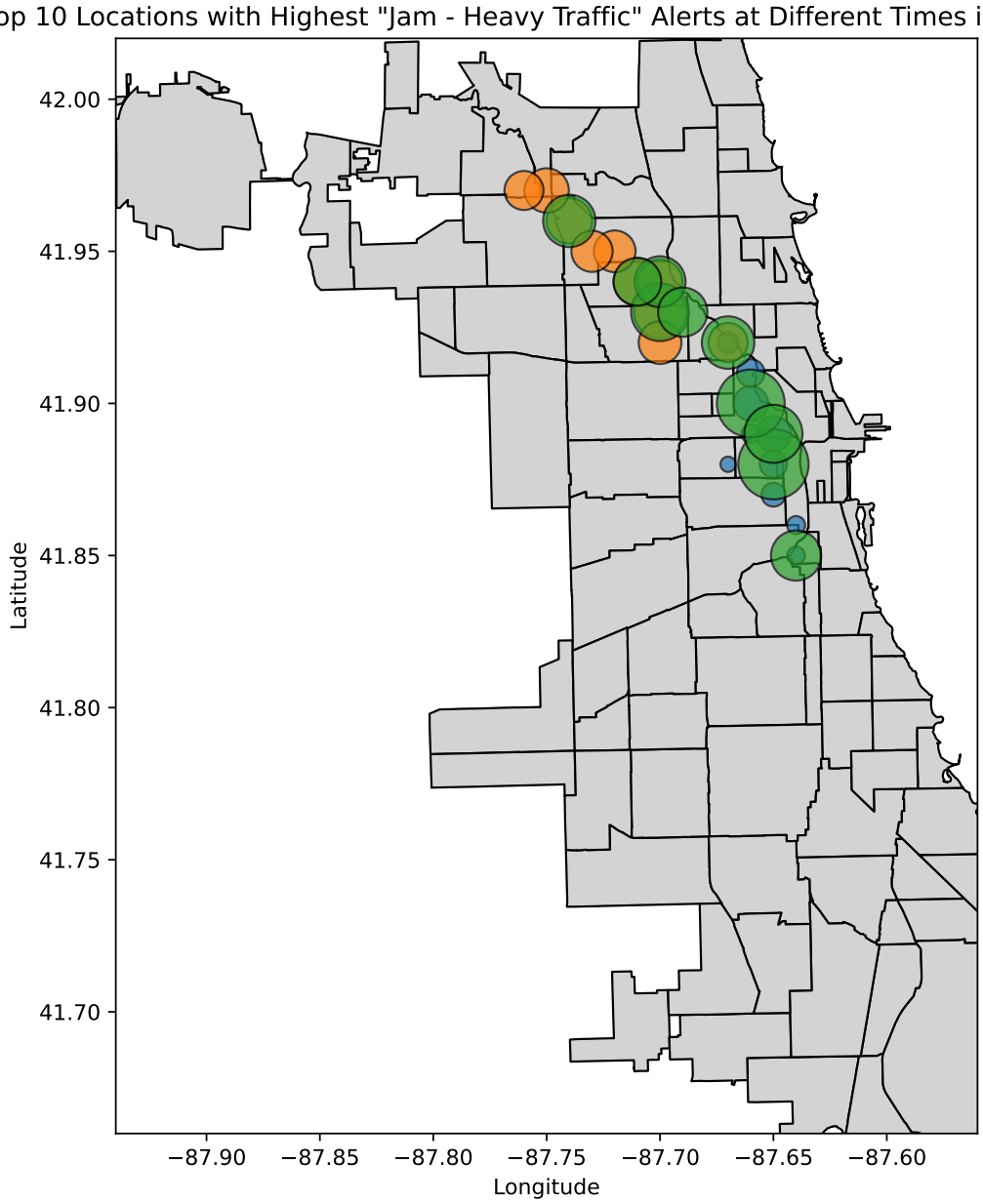
```
df_merged['ts'] = pd.to_datetime(df_merged['ts'])
df_merged['hour'] = df_merged['ts'].dt.strftime('%H:00')


top_alerts_byhour = (df_merged.groupby(['latitude',
     'longitude', 'updated_type',
     'updated_subtype','hour']).size()
    .reset_index(name ='count'))

# Ranking top types and subtypes
top_alerts_byhour['rank'] = top_alerts_byhour.groupby(
    ['updated_type', 'updated_subtype',
    'hour'])['count'].rank(
    method ='first', ascending = False)

# Keep only top 10 ranks
top_alerts_map_byhour = top_alerts_byhour[
    top_alerts_byhour['rank'] <= 10].drop(columns ='rank').reset_index(drop =
 ↪  True)
# Reading to csv to specified path
output_path =
 ↪  r'/Users/nasser.alshaya/Desktop/Fall-2024/PPHA-30538/PS6/top_alerts_map_byhour'
output_path = os.path.join(output_path, 'top_alerts_map_byhour.csv')
top_alerts_map_byhour.to_csv(output_path, index = True)

len(top_alerts_map_byhour)
```

3202

c.

```
# Using matplotlib for the app:
hours_of_interest = [3, 13, 21]

# convert to integer
top_alerts_map_byhour['hour'] =
 ↪  top_alerts_map_byhour['hour'].str.split(':').str[0].astype(int)

filtered_data = top_alerts_map_byhour[
    (top_alerts_map_byhour['updated_type'] == 'Jam') &
    (top_alerts_map_byhour['updated_subtype'] == 'Heavy Traffic') &
    (top_alerts_map_byhour['hour'].isin(hours_of_interest))
]

filtered_data_sorted = filtered_data.sort_values(
    by='count', ascending=False)

fig, ax = plt.subplots(figsize=(8, 8))
geo_data.plot(ax=ax, color='lightgrey', edgecolor='black')

# Loop through each hour to plot the data
for hour in hours_of_interest:
    # Filter the data for the current hour
    hour_data = filtered_data_sorted[
        filtered_data_sorted['hour'] == hour].head(10)

    scatter = ax.scatter(
        hour_data['longitude'],
        hour_data['latitude'],
        s=hour_data['count'] * 2,
        alpha=0.7,
        edgecolor='k',
    )

ax.set_xlim([-87.94, -87.56])
ax.set_ylim([41.66, 42.02])
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
ax.set_title(
    'Top 10 Locations with Highest "Jam - Heavy Traffic" Alerts at Different
     ↪  Times in Chicago'
)
```

```
plt.tight_layout()
plt.show()
```



2.

a.

```
app_ui = ui.page_fluid(
    ui.input_select(
        id="type_subtype",
        label="Choose a combination:",
        choices=[],
    ),
     ui.input_slider(
         "hour",
         "Select Hour:",
         min=0,
         max=23,
         step=1,
         value=12),
    ui.output_plot("ts"),
)
```

Figure 4: APP2_UI

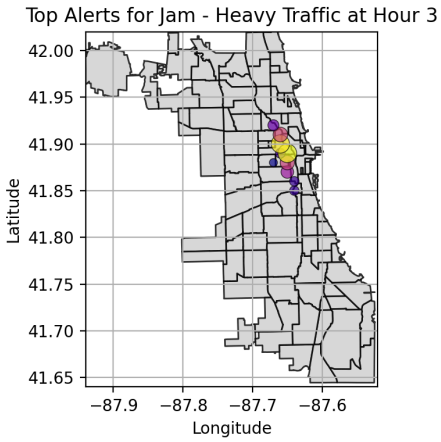
b.

Choose a combination:

Jam - Heavy Traffic

Select Hour:

0 3 23

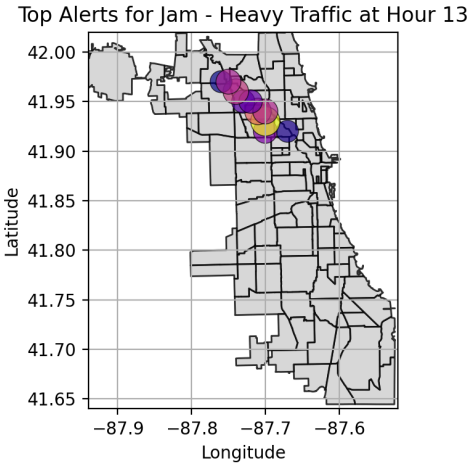


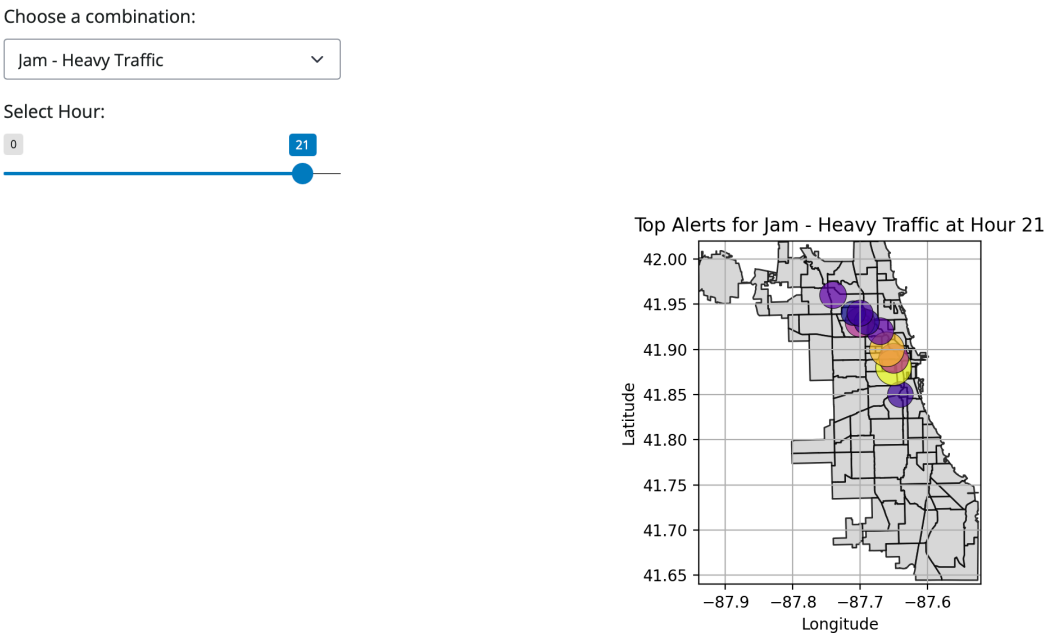
Choose a combination:

Jam - Heavy Traffic

Select Hour:

0 13 23





c. it is hard to tell but it seems like road construction is done slightly more during night hours than morning hours

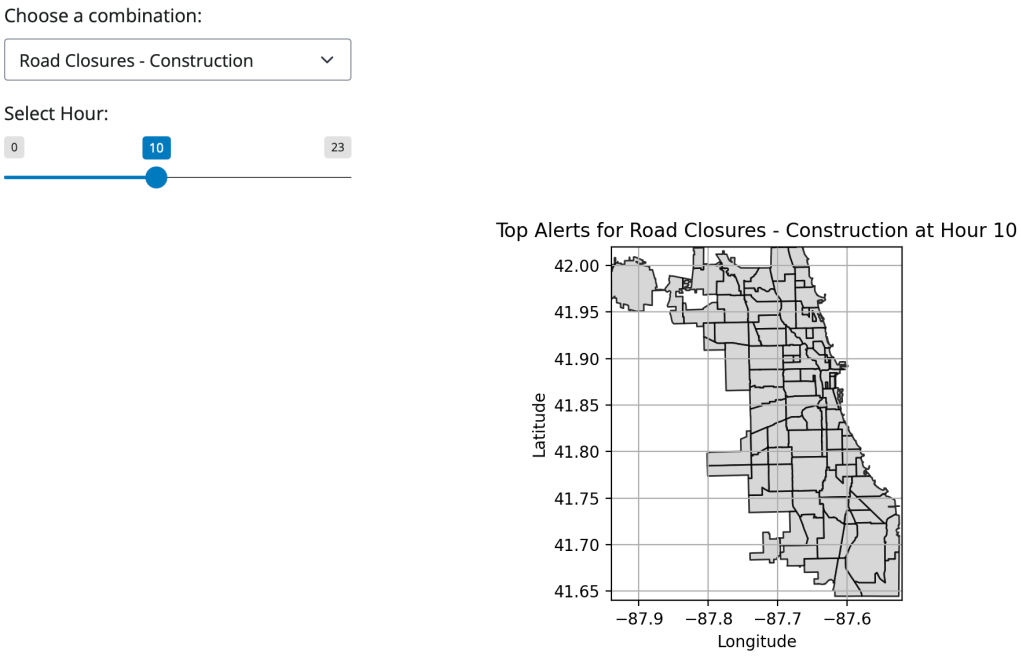


Figure 5: Road_Closure_Construction

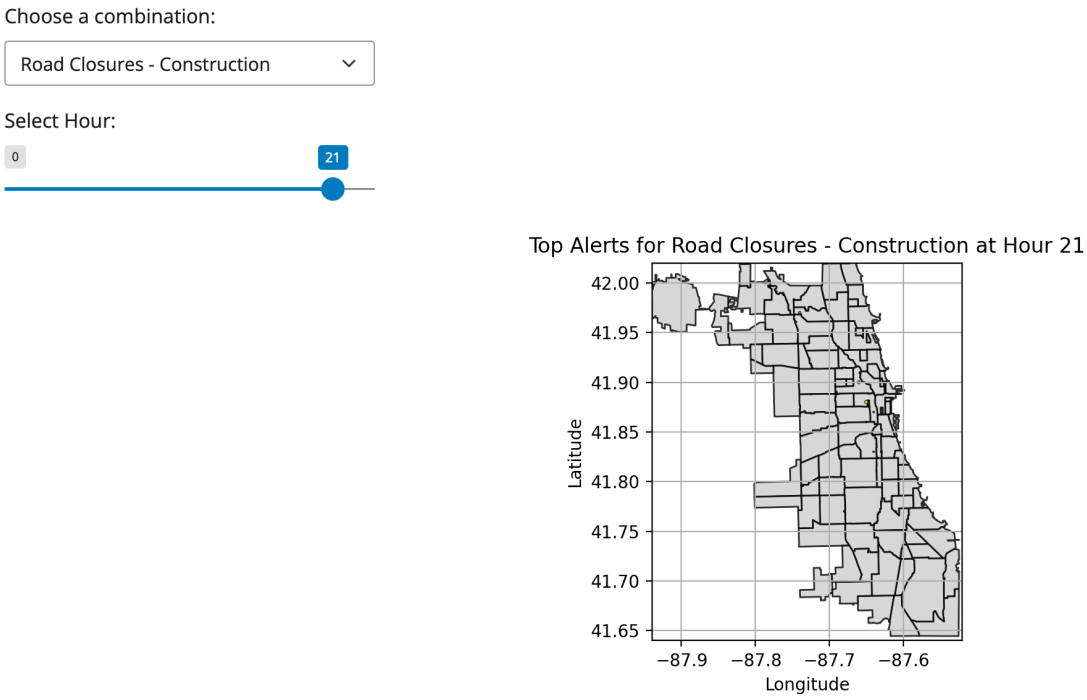


Figure 6: Road_Closure_Construction

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

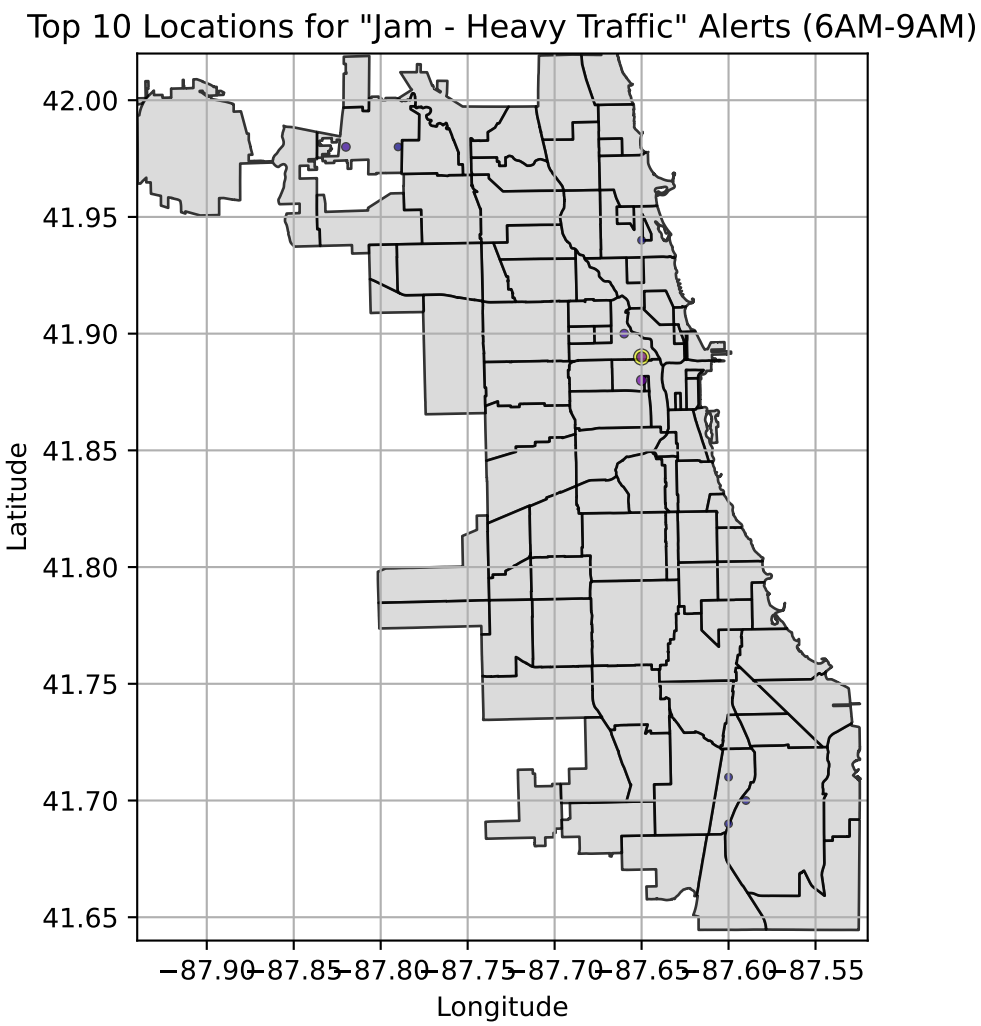
1.

a. Yes it would be a good idea since htis will allow the user to see the traffic alerts over a time span rather than a point in time which is more realistic and helpful.

b.

```
filtered_df = top_alerts_map_byhour[
    (top_alerts_map_byhour["updated_type"] == "Jam") &
    (top_alerts_map_byhour["updated_subtype"] == "Heavy Traffic") &
    (top_alerts_map_byhour["hour"] >= 6) &
    (top_alerts_map_byhour["hour"] <= 9)
]
filtered_df = filtered_df.sort_values(
```

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

fig, ax = plt.subplots(figsize=(10, 6))
geo_data.plot(ax=ax, color='lightgrey', edgecolor='black', alpha=0.8)

scatter = ax.scatter(
    filtered_df['longitude'],
    filtered_df['latitude'],
    s=filtered_df['count'] * 2,
    c=filtered_df['count'],
    cmap='plasma',
    alpha=0.7,
    edgecolor='black',
    linewidth=0.5,
)

ax.set_title('Top 10 Locations for "Jam - Heavy Traffic" Alerts (6AM-9AM)')
ax.set_xlabel('Longitude')
ax.set_ylabel('Latitude')
ax.set_xlim(-87.94, -87.52)
ax.set_ylim(41.64, 42.02)
ax.grid(True)

plt.show()
```


Top 10 Locations for "Jam - Heavy Traffic" Alerts (6AM-9AM)
Latitude
42.00
41.95
41.90
41.85
41.80
41.75
41.70
41.65
−87.90
−87.85
−87.80
−87.75
−87.70
−87.65
−87.60
−87.55
Longitude

2.

a.

```
app_ui = ui.page_fluid(
    ui.input_select(
        id="type_subtype",
        label="Choose a combination:",
        choices=[],
    ),
     ui.input_slider(
         "hour_range",
         "Select Hour:",
         min=0,
         max=23,
         step=1,
         value=(6,9)),
    ui.output_plot("ts"),
)
```

Figure 7: APP3_UI

b.

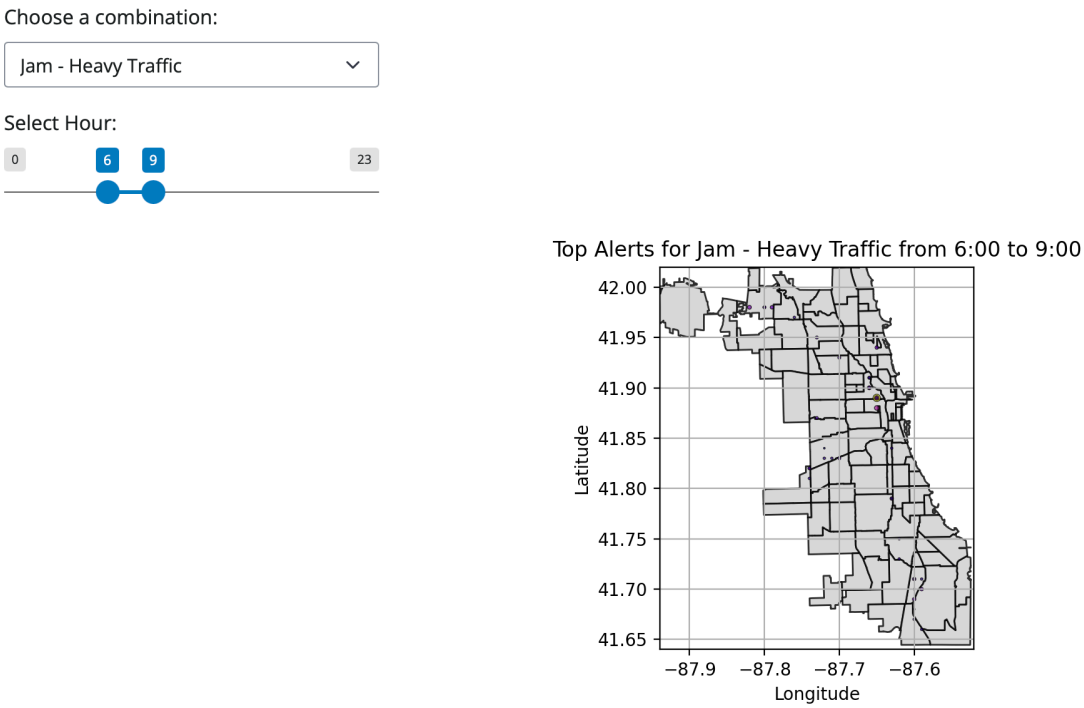


Figure 8: Jam_Heavy_Traffic_plot

3.

a. The possible vaule are true or false, since the value depends on the boolean condition of which panel to switch to.

```
app_ui = ui.page_fluid(
    ui.input_select(
        id="type_subtype",
        label="Choose a combination:",
        choices=[],
    ),
    ui.input_switch(
        id = "switch_button",
         label = "Toggle to switch to range of hours",
         value=False
    ),
    ui.panel_conditional(
         "input.switch_button == true",
    ui.input_slider(
        id =  "hour",
         label = "Select Single Hour:",
         min=0,
         max=23,
         value=12,
         step=1
    )),
    ui.panel_conditional(
         "input.switch_button == false",
    ui.input_slider(
         id = "hour_range",
         label = "Select Hour Range:",
         min=0,
         max=23,
         value=(6, 9),
         step=1)),

    ui.output_ui("dynamic_ui"),
ui.output_plot("ts"))
```

Figure 9: APP3_UI

b.

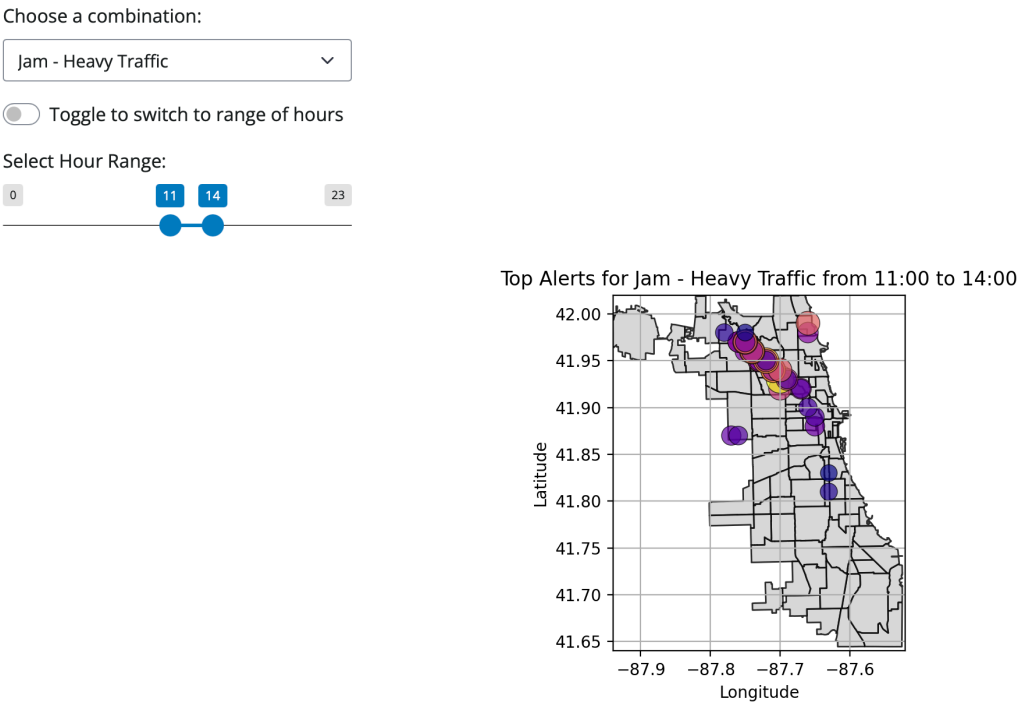


Figure 10: Jam_Heavy_Traffic_plot

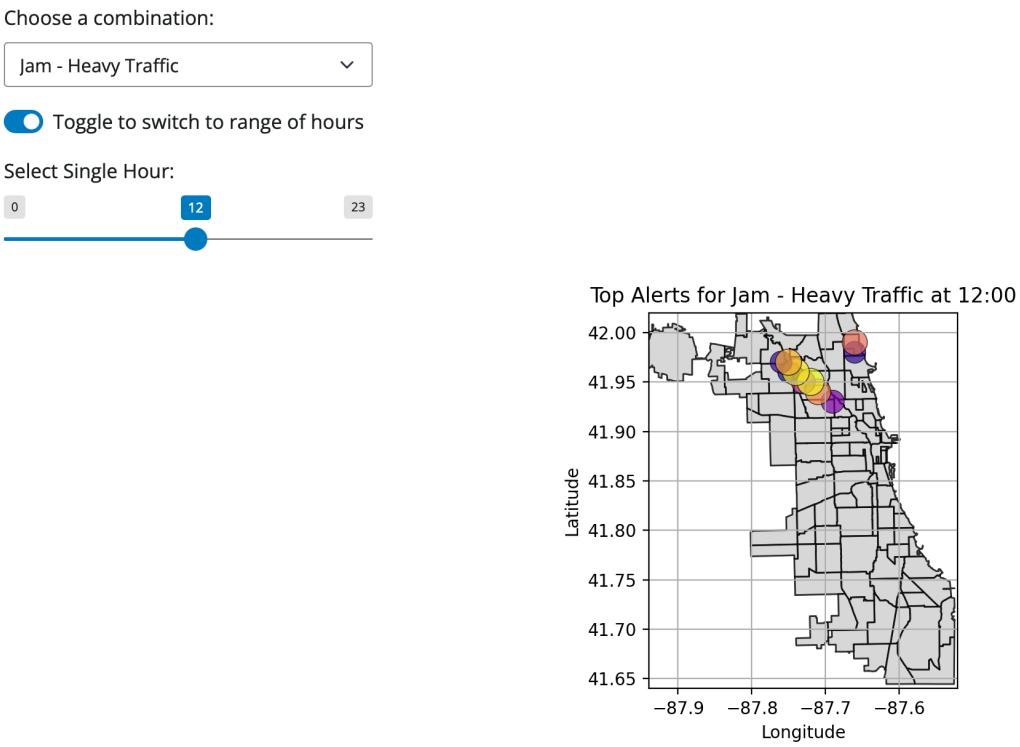


Figure 11: Jam_Heavy_Traffic_plot

c.

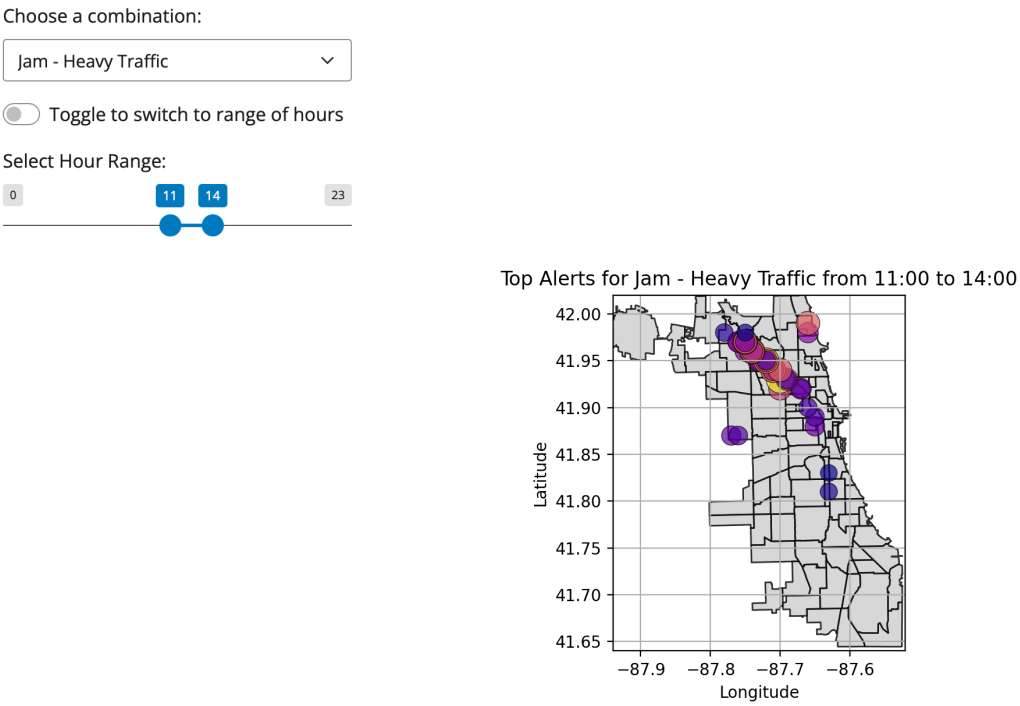


Figure 12: Jam_Heavy_Traffic_plot

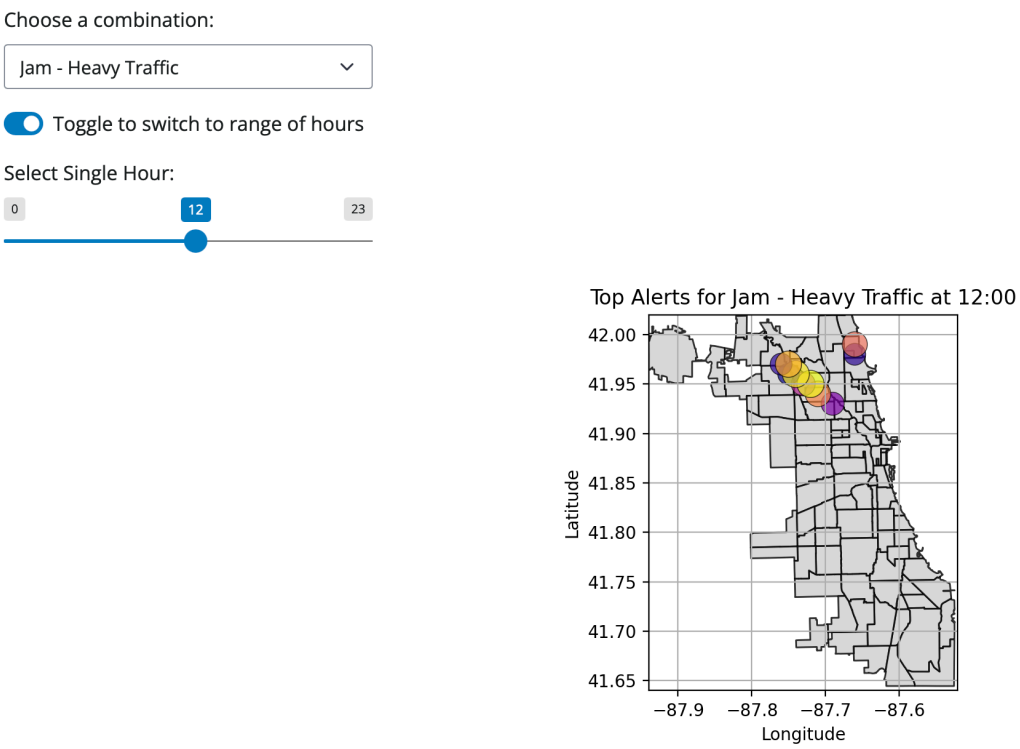


Figure 13: Jam_Heavy_Traffic_plot

d. adding a column in the dataframe for time of day based on hour to specify morning, afternoon, evening, night. For example, the values for hour between 4 - 12 morning, 13 - 15 afternoon, 16 - 19 evening, 20 - 3 night.