Problem Set 6 - Waze Shiny Dashboard

Eddie Andujar

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: *EA*
2. "I have uploaded the names of anyone I worked with on the problem set **here**" *EA** (2 point)
3. Late coins used this pset: *1* Late coins left after submission: *0*
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!

For debugging, Stack Overflow and ChatGPT were used, along with collaboration with my peers. After coding, ChatGPT was then used for additional clean-up and further debugging, as sometimes the app would run successfully one day, then crash the next.

Background

Data Download and Exploration (20 points)

1.

```
file_path =r'C:\Users\eddie\OneDrive\Documents\ps6'

waze_zip_path =
 ↪  r'C:\Users\eddie\OneDrive\Documents\GitHub\student30538\problem_sets\ps6\waze_data\waze_

df = pd.read_csv(waze_zip_path)

# Create a list of column names to more easily visualize

column_names = df.columns.tolist()
column_names
```

```
['city',
 'confidence',
 'nThumbsUp',
 'street',
 'uuid',
 'country',
 'type',
 'subtype',
 'roadType',
 'reliability',
 'magvar',
 'reportRating',
 'ts',
 'geo',
 'geoWKT']
```

city = Nominal confidence = Quantitative nThumbsUp = Quantitative street = Nominal uuid = Nominal country = Nominal type = Nominal subtype = Nominal roadType = Nominal reliability = Quantitative magvar = Quantitative reportRating = Quantitative

2.

```
# Calculate missing and non-missing counts
missing_counts = df.isnull().sum()
non_missing_counts = df.notnull().sum()

# Create a DataFrame for Altair
missing_data = pd.DataFrame({
    'Variable': df.columns,
    'Missing': missing_counts.values,
    'Not Missing': non_missing_counts.values
})

# Melt the DataFrame to long format for Altair
missing_data_long = missing_data.melt(
    id_vars='Variable',
    value_vars=['Missing', 'Not Missing'],
    var_name='Status',
    value_name='Count'
)

# Create the stacked bar chart using Altair
chart = alt.Chart(missing_data_long).mark_bar().encode(
    x=alt.X('Variable:N', title='Variables', sort=None),
    y=alt.Y('Count:Q', title='Count'),
    color=alt.Color(
        'Status:N',
        scale=alt.Scale(domain=['Missing', 'Not Missing'], range=['red',
 ↪  'green']),
        legend=alt.Legend(title="Data Status")
    ),
    tooltip=['Variable:N', 'Status:N', 'Count:Q']
).properties(
    title='Stacked Bar Chart of Missing and Non-Missing Values by Variable',
    width=800,
    height=400
).configure_axis(
    labelAngle=45
)
```

```
chart.show()

# Analyze the variables with missing values
variables_with_missing = missing_data[missing_data['Missing'] >
 ↪  0].sort_values(by='Missing', ascending=False)
print("\nVariables with missing values:")
print(variables_with_missing)

# Identify the variable with the highest share of missing values
highest_missing_share = (missing_counts / len(df)).idxmax()
print(f"\nVariable with the highest share of missing values:
 ↪  {highest_missing_share}")
```

```
alt.Chart(...)


Variables with missing values:
    Variable  Missing  Not Missing
2  nThumbsUp   776723         1371
7    subtype    96086       682008
3     street    14073       764021

Variable with the highest share of missing values: nThumbsUp
```

3.

```
# Extract the type and subtype columns

waze_data_path = r'C:\Users\eddie\OneDrive\Documents\ps6\waze_data.csv'

waze_data_df = pd.read_csv(waze_data_path)

types = waze_data_df['type']
subtypes = waze_data_df['subtype']

# Print unique values for type and subtype
print("Unique types:")
print(types.unique())
print("\nUnique subtypes:")
print(subtypes.unique())
```

```python
# Count the number of types with NA subtypes
na_subtypes_count = types[subtypes.isna()].nunique()
print(f"\nNumber of types with NA subtypes: {na_subtypes_count}")

# Identify types with potential sub-subtypes by grouping non-NA subtypes
subtypes_for_types = (
    waze_data_df.loc[subtypes.notna()]
    .groupby('type')['subtype']
    .unique()
    .sort_index()  # Optional, ensures consistent order
)
print("\nSubtypes for each type:")
print(subtypes_for_types)

# Fill NA subtypes with "Unclassified"
waze_data_df['subtype'] = subtypes.fillna('Unclassified')
```

```
Unique types:
['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']

Unique subtypes:
[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']

Number of types with NA subtypes: 4

Subtypes for each type:
type
ACCIDENT                          [ACCIDENT_MAJOR, ACCIDENT_MINOR]
HAZARD         [HAZARD_ON_ROAD, HAZARD_ON_ROAD_CAR_STOPPED, H...
JAM            [JAM_HEAVY_TRAFFIC, JAM_MODERATE_TRAFFIC, JAM_...
ROAD_CLOSED    [ROAD_CLOSED_EVENT, ROAD_CLOSED_CONSTRUCTION, ...
```

```
Name: subtype, dtype: object
```

Hierarchy (4 types)

Jam Traffic Heavy Moderate Light Unclassified Hazard On Road Car Stopped Pot Hole Object Unclassified On Shoulder Car Stopped Unclassified Weather Flood Fog Hail Accident Major Minor Unclassified Road Closed Event Unclassified Hazard

We may as well include the NA subtypes, since it's possible these correspond to important situations that either fall under multiple sub-categories or there wasn't enough information.

```
waze_data_df['subtype'] = waze_data_df['subtype'].fillna('Unclassified')
```

4.

a.

```
# Get unique combinations of type and subtype
unique_combinations = waze_data_df[['type', 'subtype']].drop_duplicates()

# Create the crosswalk DataFrame
crosswalk_df = pd.DataFrame({
    'type': unique_combinations['type'],
    'subtype': unique_combinations['subtype'].fillna('Unclassified'),
    'updated_type': '',
    'updated_subtype': '',
    'updated_subsubtype': ''
})
```

```
# Save merged_df as a CSV file in the specified directory

crosswalk_df.to_csv(r'C:\Users\eddie\OneDrive\Documents\ps6\crosswalk_data.csv',
 ↪  index=False)
```

b.

```
def clean_name(name):
    """Cleans and formats a string by replacing underscores with spaces and
     ↪  capitalizing each word."""
    return ' '.join(word.capitalize() for word in name.replace('_', '
     ↪  ').split())

def create_user_friendly_label(row):
```

```
    """Create a user-friendly label for the last column."""
    label_parts = [row['updated_type']]
    if row['updated_subsubtype'] != 'Unclassified':
        label_parts.append(row['updated_subsubtype'])
    if row['updated_subtype'] != 'Unclassified':
        label_parts.append(row['updated_subtype'])
    return ' - '.join(label_parts)

def map_to_hierarchy(row):
    # Extract original fields
    original_type = row['type']
    original_subtype = row['subtype']

    # Initialize new fields
    row['updated_type'] = ''
    row['updated_subtype'] = ''
    row['updated_subsubtype'] = ''

    # Map types and subtypes to the hierarchy
    if 'JAM' in original_type:
        row['updated_type'] = 'Jam'
        row['updated_subtype'] = 'Traffic'
        if 'STAND_STILL' in original_subtype:
            row['updated_subsubtype'] = 'Stand Still'
        elif 'HEAVY' in original_subtype:
            row['updated_subsubtype'] = 'Heavy'
        elif 'MODERATE' in original_subtype:
            row['updated_subsubtype'] = 'Moderate'
        elif 'LIGHT' in original_subtype:
            row['updated_subsubtype'] = 'Light'
        else:
            row['updated_subsubtype'] = 'Unclassified'
    elif 'ROAD_CLOSED' in original_type:
        row['updated_type'] = 'Road Closed'
        if 'CONSTRUCTION' in original_subtype:
            row['updated_subtype'] = 'Construction'
        elif 'EVENT' in original_subtype:
            row['updated_subtype'] = 'Event'
        elif 'HAZARD' in original_subtype:
            row['updated_subtype'] = 'Hazard'
        else:
            row['updated_subtype'] = 'Unclassified'
```

```
    elif 'ACCIDENT' in original_type:
        row['updated_type'] = 'Accident'
        if 'MAJOR' in original_subtype:
            row['updated_subtype'] = 'Major'
        elif 'MINOR' in original_subtype:
            row['updated_subtype'] = 'Minor'
        else:
            row['updated_subtype'] = 'Reported'
    elif 'HAZARD' in original_type:
        row['updated_type'] = 'Hazard'
        if 'ON_ROAD' in original_subtype:
            row['updated_subtype'] = 'On Road'
            if original_subtype == 'HAZARD_ON_ROAD':
                row['updated_subsubtype'] = 'Unclassified'
            else:
                row['updated_subsubtype'] =
↪   clean_name(original_subtype.replace('HAZARD_ON_ROAD_', ''))
        elif 'ON_SHOULDER' in original_subtype:
            row['updated_subtype'] = 'On Shoulder'
            if original_subtype == 'HAZARD_ON_SHOULDER':
                row['updated_subsubtype'] = 'Unclassified'
            else:
                row['updated_subsubtype'] =
↪   clean_name(original_subtype.replace('HAZARD_ON_SHOULDER_', ''))
        elif 'WEATHER' in original_subtype:
            row['updated_subtype'] = 'Weather'
            row['updated_subsubtype'] =
↪   clean_name(original_subtype.replace('HAZARD_WEATHER_', ''))
        else:
            row['updated_subtype'] = 'Unclassified'

    # Fallback for unclassified rows
    if not row['updated_type']:
        row['updated_type'] = clean_name(original_type)
    if not row['updated_subtype']:
        row['updated_subtype'] = 'Unclassified'
    if not row['updated_subsubtype']:
        row['updated_subsubtype'] = 'Unclassified'

    # Create user-friendly label
    row['user_friendly_label'] = create_user_friendly_label(row)
    return row
```

```
# Apply the mapping function to each row
crosswalk_df = crosswalk_df.apply(map_to_hierarchy, axis=1)

# Rearrange columns: keep original type/subtype, then new columns, then the
 ↪  user-friendly label
columns_order = ['type', 'subtype', 'updated_type', 'updated_subtype',
                 'updated_subsubtype', 'user_friendly_label']
crosswalk_df = crosswalk_df[columns_order]

# Sort the DataFrame alphabetically by the user-friendly label
crosswalk_df = crosswalk_df.sort_values(['user_friendly_label'])

# Display the final DataFrame
print(crosswalk_df)
```

```
              type                              subtype updated_type  \
122        ACCIDENT                 ACCIDENT_MAJOR     Accident
131        ACCIDENT                 ACCIDENT_MINOR     Accident
1          ACCIDENT                   Unclassified     Accident
26           HAZARD                   Unclassified       Hazard
21447        HAZARD          HAZARD_ON_SHOULDER_ANIMALS       Hazard
190          HAZARD          HAZARD_ON_ROAD_CAR_STOPPED       Hazard
485          HAZARD      HAZARD_ON_SHOULDER_CAR_STOPPED       Hazard
240          HAZARD         HAZARD_ON_ROAD_CONSTRUCTION       Hazard
276          HAZARD    HAZARD_ON_ROAD_EMERGENCY_VEHICLE       Hazard
857          HAZARD                HAZARD_WEATHER_FLOOD       Hazard
5557         HAZARD                  HAZARD_WEATHER_FOG       Hazard
229005       HAZARD                 HAZARD_WEATHER_HAIL       Hazard
854          HAZARD                      HAZARD_WEATHER       Hazard
44216        HAZARD           HAZARD_WEATHER_HEAVY_SNOW       Hazard
302          HAZARD                  HAZARD_ON_ROAD_ICE       Hazard
1905         HAZARD          HAZARD_ON_ROAD_LANE_CLOSED       Hazard
21940        HAZARD     HAZARD_ON_SHOULDER_MISSING_SIGN       Hazard
303          HAZARD               HAZARD_ON_ROAD_OBJECT       Hazard
148          HAZARD                     HAZARD_ON_ROAD       Hazard
483          HAZARD                 HAZARD_ON_SHOULDER       Hazard
355          HAZARD             HAZARD_ON_ROAD_POT_HOLE       Hazard
21443        HAZARD            HAZARD_ON_ROAD_ROAD_KILL       Hazard
478          HAZARD  HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT       Hazard
858             JAM                  JAM_HEAVY_TRAFFIC          Jam
38546           JAM                  JAM_LIGHT_TRAFFIC          Jam
```

```
1122            JAM                JAM_MODERATE_TRAFFIC           Jam
1184            JAM             JAM_STAND_STILL_TRAFFIC           Jam
0               JAM                         Unclassified          Jam
2       ROAD_CLOSED                         Unclassified  Road Closed
7331    ROAD_CLOSED            ROAD_CLOSED_CONSTRUCTION  Road Closed
1335    ROAD_CLOSED                   ROAD_CLOSED_EVENT  Road Closed
54556   ROAD_CLOSED                 ROAD_CLOSED_HAZARD  Road Closed

       updated_subtype   updated_subsubtype  \
122              Major         Unclassified
131              Minor         Unclassified
1             Reported         Unclassified
26         Unclassified         Unclassified
21447       On Shoulder              Animals
190             On Road          Car Stopped
485         On Shoulder          Car Stopped
240             On Road         Construction
276             On Road    Emergency Vehicle
857             Weather                Flood
5557            Weather                  Fog
229005          Weather                 Hail
854             Weather       Hazard Weather
44216           Weather           Heavy Snow
302             On Road                  Ice
1905            On Road          Lane Closed
21940       On Shoulder         Missing Sign
303             On Road               Object
148             On Road         Unclassified
483         On Shoulder         Unclassified
355             On Road             Pot Hole
21443           On Road            Road Kill
478             On Road  Traffic Light Fault
858             Traffic                Heavy
38546           Traffic                Light
1122            Traffic             Moderate
1184            Traffic          Stand Still
0               Traffic         Unclassified
2          Unclassified         Unclassified
7331       Construction         Unclassified
1335              Event         Unclassified
54556            Hazard         Unclassified

                          user_friendly_label
```

```
122                           Accident - Major
131                           Accident - Minor
1                           Accident - Reported
26                                    Hazard
21447           Hazard - Animals - On Shoulder
190             Hazard - Car Stopped - On Road
485          Hazard - Car Stopped - On Shoulder
240             Hazard - Construction - On Road
276        Hazard - Emergency Vehicle - On Road
857                    Hazard - Flood - Weather
5557                     Hazard - Fog - Weather
229005                  Hazard - Hail - Weather
854           Hazard - Hazard Weather - Weather
44216             Hazard - Heavy Snow - Weather
302                      Hazard - Ice - On Road
1905            Hazard - Lane Closed - On Road
21940       Hazard - Missing Sign - On Shoulder
303                   Hazard - Object - On Road
148                           Hazard - On Road
483                        Hazard - On Shoulder
355                 Hazard - Pot Hole - On Road
21443              Hazard - Road Kill - On Road
478     Hazard - Traffic Light Fault - On Road
858                       Jam - Heavy - Traffic
38546                     Jam - Light - Traffic
1122                   Jam - Moderate - Traffic
1184                Jam - Stand Still - Traffic
0                                Jam - Traffic
2                                  Road Closed
7331                Road Closed - Construction
1335                      Road Closed - Event
54556                     Road Closed - Hazard
```

c.

```
# Merge the crosswalk with the original data
merged_df = waze_data_df.merge(crosswalk_df, on=['type', 'subtype'],
↪  how='left')

# Count rows for Accident - Unclassified
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: 0
```

d. (EXTRA CREDIT ATTEMPT)

```
# Step 1: Get unique combinations of 'type' and 'subtype' in both DataFrames
crosswalk_combinations = crosswalk_df[['type', 'subtype']].drop_duplicates()
merged_combinations = merged_df[['type', 'subtype']].drop_duplicates()

# Step 2: Check for mismatches or missing values
# Find combinations in the crosswalk that are missing from the merged dataset
missing_in_merged = crosswalk_combinations.merge(
    merged_combinations, on=['type', 'subtype'], how='left', indicator=True
).query("_merge == 'left_only'")

# Find combinations in the merged dataset that are missing from the crosswalk
missing_in_crosswalk = merged_combinations.merge(
    crosswalk_combinations, on=['type', 'subtype'], how='left',
 ↪  indicator=True
).query("_merge == 'left_only'")

# Step 3: Display results
if missing_in_merged.empty and missing_in_crosswalk.empty:
    print("All combinations of type and subtype match between the crosswalk
     ↪  and the merged dataset!")
else:
    print("Mismatch detected:")
    if not missing_in_merged.empty:
        print("\nCombinations in crosswalk but not in merged dataset:")
        print(missing_in_merged[['type', 'subtype']])
    if not missing_in_crosswalk.empty:
        print("\nCombinations in merged dataset but not in crosswalk:")
        print(missing_in_crosswalk[['type', 'subtype']])
```

```
All combinations of type and subtype match between the crosswalk and the
merged dataset!
```

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

1.

ChatGPT was used for assistance here. I asked it how to extract coordinates, specifically isolating latitude and longitude. I also provided it with the specific question about Waze ad for some added context.

a.

```
import re

pattern = r'POINT\((-?\d+\.\d+)\s(-?\d+\.\d+)\)'

# Extract latitude and longitude using the updated regex
waze_data_df[['longitude', 'latitude']] =
 ↪  waze_data_df['geo'].str.extract(pattern)

# Convert the extracted values to float
waze_data_df[['latitude', 'longitude']] = waze_data_df[['latitude',
 ↪  'longitude']].astype(float)

# Display the updated DataFrame with latitude and longitude
print(waze_data_df[['geo', 'latitude', 'longitude']])
```

```
                              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
4        POINT(-87.735235 41.91658)  41.916580 -87.735235
...                             ...        ...        ...
778089  POINT(-87.615862 41.887432)  41.887432 -87.615862
778090  POINT(-87.615882 41.887442)  41.887442 -87.615882
778091  POINT(-87.645584 41.884419)  41.884419 -87.645584
778092  POINT(-87.598843 41.692532)  41.692532 -87.598843
778093  POINT(-87.598843 41.692532)  41.692532 -87.598843

[778094 rows x 3 columns]
```

b.

```
# Bin latitude and longitude into bins with step size 0.01
waze_data_df['binned_latitude'] = (waze_data_df['latitude'] // 0.01) * 0.01
waze_data_df['binned_longitude'] = (waze_data_df['longitude'] // 0.01) * 0.01
```

```
# Count the number of observations for each binned latitude-longitude
 ↪  combination
binned_counts = waze_data_df.groupby(['binned_latitude',
 ↪  'binned_longitude']).size().reset_index(name='count')
```

```
# Identify the binned latitude-longitude combination with the greatest number
 ↪  of observations
max_binned = binned_counts.loc[binned_counts['count'].idxmax()]

result = f"({max_binned['binned_latitude']:.2f},
 ↪  {max_binned['binned_longitude']:.2f})"

print(f"The binned latitude-longitude combination with the greatest number of
 ↪  observations is: {result}")
print(f"Number of observations: {max_binned['count']}")
```

```
The binned latitude-longitude combination with the greatest number of
observations is: (41.96, -87.75)
Number of observations: 26537.0
```

c.

```
import os as os
```

```
chosen_type = 'Jam'
chosen_subtype = 'Traffic'

# Merge waze_data_df with crosswalk_df to get updated types and subtypes
merged_df = waze_data_df.merge(crosswalk_df, on=['type', 'subtype'],
 ↪  how='left')
directory = r'C:\Users\eddie\OneDrive\Documents\ps6'
output_path = os.path.join(directory, "merged_df.csv")

# Save merged_df as CSV
merged_df.to_csv(output_path, index=False)
# Filter data for chosen updated type and subtype
```

```
filtered_data = merged_df[
    (merged_df['updated_type'] == chosen_type) &
    (merged_df['updated_subtype'] == chosen_subtype)
]

# Bin latitude and longitude into bins with step size 0.01
filtered_data['binned_latitude'] = (filtered_data['latitude'] // 0.01) * 0.01
filtered_data['binned_longitude'] = (filtered_data['longitude'] // 0.01) *
 ↪  0.01

# Aggregate the data to count the number of alerts per binned latitude and
 ↪  longitude
aggregated_data = filtered_data.groupby(['binned_latitude',
 ↪  'binned_longitude']).size().reset_index(name='alert_count')

# Sort the aggregated data by alert_count in descending order
sorted_data = aggregated_data.sort_values(by='alert_count', ascending=False)

# Ensure the directory exists
output_dir =
 ↪  r'C:\Users\eddie\OneDrive\Documents\GitHub\student30538\problem_sets\ps6\top_alerts_map'
os.makedirs(output_dir, exist_ok=True)

# Save the resulting DataFrame as 'top_alerts_map.csv' in the specified
 ↪  folder
output_path = os.path.join(output_dir, 'top_alerts_map.csv')
sorted_data.to_csv(output_path, index=False)

# Load the saved DataFrame to count the number of rows
saved_data = pd.read_csv(output_path)

# Count the number of rows in the saved DataFrame
num_rows = saved_data.shape[0]

# Level of aggregation
level_of_aggregation = "Binned latitude and longitude for chosen updated type
 ↪  and subtype"

print(f"Level of aggregation: {level_of_aggregation}")
print(f"Number of rows in the DataFrame: {num_rows}")
print(saved_data)
```

```python
# Additional information
print(f"\nChosen type: {chosen_type}")
print(f"Chosen subtype: {chosen_subtype}")
print(f"\nTotal alerts for {chosen_type} - {chosen_subtype}:
 ↪  {filtered_data.shape[0]}")
print(f"Number of unique latitude-longitude bins:
 ↪  {aggregated_data.shape[0]}")
```

```
Level of aggregation: Binned latitude and longitude for chosen updated type
and subtype
Number of rows in the DataFrame: 676
     binned_latitude  binned_longitude  alert_count
0              41.89            -87.66        10866
1              41.87            -87.65         9034
2              41.88            -87.65         7950
3              41.90            -87.67         7072
4              41.96            -87.75         6750
..               ...               ...          ...
671            41.96            -87.80            1
672            41.95            -87.83            1
673            41.67            -87.69            1
674            41.67            -87.66            1
675            42.02            -87.68            1

[676 rows x 3 columns]

Chosen type: Jam
Chosen subtype: Traffic

Total alerts for Jam - Traffic: 372485
Number of unique latitude-longitude bins: 676

C:\Users\eddie\AppData\Local\Temp\ipykernel_15272\712055582.py:18:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  filtered_data['binned_latitude'] = (filtered_data['latitude'] // 0.01) *
  0.01
```

```
C:\Users\eddie\AppData\Local\Temp\ipykernel_15272\712055582.py:19:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  filtered_data['binned_longitude'] = (filtered_data['longitude'] // 0.01) *
  0.01
```

The level of aggregation is based on latitude-longitude bins (0.01 step) and our chosen type and subtype: Jam and Traffic. There are 676 unique bins here.

2.

```
# Filter for heavy traffic jams: type = 'Jam', subtype = 'Traffic', and
 ↪  subsubtype = 'Heavy'
jam_heavy = merged_df[
    (merged_df['updated_type'] == 'Jam') &
    (merged_df['updated_subtype'] == 'Traffic') &
    (merged_df['updated_subsubtype'] == 'Heavy')
]

# Aggregate data by latitude-longitude bins
aggregated = jam_heavy.groupby(
    ['binned_latitude',
 ↪  'binned_longitude']).size().reset_index(name='alert_count')

# Filter the top 10 locations with the highest alert count
top_10 = aggregated.nlargest(10, 'alert_count')


# Create the chart with the top 10 locations
# Calculate min and max for latitude and longitude to adjust axis range
min_lat, max_lat = top_10['binned_latitude'].min(),
 ↪  top_10['binned_latitude'].max()
min_lon, max_lon = top_10['binned_longitude'].min(),
 ↪  top_10['binned_longitude'].max()

# Add padding to the range for better visibility
lat_padding = (max_lat - min_lat) * 0.1
lon_padding = (max_lon - min_lon) * 0.1
```

```
top_10_chart = alt.Chart(top_10).mark_circle(color='blue').encode(  # Changed
 ↪  color to blue
    x=alt.X('binned_longitude:Q', title='Longitude',
 ↪  scale=alt.Scale(domain=[min_lon - lon_padding, max_lon + lon_padding])),
    y=alt.Y('binned_latitude:Q', title='Latitude',
 ↪  scale=alt.Scale(domain=[min_lat - lat_padding, max_lat + lat_padding])),
    size=alt.Size('alert_count:Q', scale=alt.Scale(range=[50, 750]),
 ↪  title='Alert Count'),
    tooltip=[
        alt.Tooltip('binned_latitude:Q', title='Latitude'),
        alt.Tooltip('binned_longitude:Q', title='Longitude'),
        alt.Tooltip('alert_count:Q', title='Alert Count')
    ]
).properties(
    title='Top 10 Locations for Jam - Heavy Traffic Alerts',
    width=600,
    height=400
).configure_axis(
    grid=True,  # Disable grid lines
    labelFontSize=14,  # Increased axis label font size
    titleFontSize=16,  # Increased axis title font size
    titleFontWeight='bold',  # Bold axis titles
    labelPadding=10  # Space between axis labels and the axis
).configure_title(
    fontSize=18,  # Increased font size for chart title
    fontWeight='bold'  # Bold chart title
).configure_legend(
    titleFontSize=16,  # Increased font size for legend title
    labelFontSize=14,  # Increased font size for legend labels
    symbolSize=120  # Adjusted size of the legend symbols
)

# Display the chart
top_10_chart
```

```
alt.Chart(...)
```

3.

a.

```
# Specify the directory and file path
file_path = r"C:\Users\eddie\OneDrive\Documents\GitHub\ps6\Boundaries -
 ↪  Neighborhoods.geojson"
with open(file_path) as f:
    chicago_geojson = json.load(f)

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

b.

```
import requests
import json
```

```
# URL of the Chicago neighborhood boundaries GeoJSON
url =
 ↪  "https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON"

# Download the file
response = requests.get(url)
chicago_geojson = response.json()

# Save the file locally
with open(file_path, "w") as f:
    json.dump(chicago_geojson, f)

print(f"File saved to: {file_path}")
geo_data = alt.Data(values=chicago_geojson["features"])
```

```
File saved to: C:\Users\eddie\OneDrive\Documents\GitHub\ps6\Boundaries -
Neighborhoods.geojson
```

```
if "features" in chicago_geojson and chicago_geojson["features"]:
    geo_data = alt.Data(values=chicago_geojson["features"])
else:
    print("GeoJSON 'features' key is missing or empty")
```

4.

```
# Apply equirectangular projection to the map
base = alt.Chart(geo_data).mark_geoshape(
    fill='lightblue',  # Changed fill color for better contrast
    stroke='white'
).properties(
    width=600,
    height=400
).project(
    type='equirectangular'
)

# Add points layer for top 10 jam-heavy traffic locations
top_10_chart = alt.Chart(top_10).mark_circle(color='darkorange').encode(  #
 ↪  Changed point color to dark orange
    x=alt.X(
        'binned_longitude:Q',
        title='Longitude',
        scale=alt.Scale(domain=[min_lon - lon_padding, max_lon +
 ↪  lon_padding])
    ),
    y=alt.Y(
        'binned_latitude:Q',
        title='Latitude',
        scale=alt.Scale(domain=[min_lat - lat_padding, max_lat +
 ↪  lat_padding])
    ),
    size=alt.Size(
        'alert_count:Q',
        scale=alt.Scale(range=[50, 750]),
        title='Alert Count'
    ),
    tooltip=[
        alt.Tooltip('binned_latitude:Q', title='Latitude'),
        alt.Tooltip('binned_longitude:Q', title='Longitude'),
        alt.Tooltip('alert_count:Q', title='Alert Count')
    ]
)

# Combine the base map and points layer using alt.layer
jam_chart = alt.layer(base, top_10_chart).properties(
    title='Top 10 Locations for Heavy Traffic Jam Alerts',
    width=600,
```

```
    height=400
)

# Apply configurations to the combined chart
jam_chart = jam_chart.configure_view(
    strokeWidth=0  # Remove border around the chart
).configure_axis(
    grid=True,  # Add grid lines
    labelFontSize=14,  # Increased axis label font size for better
 ↪  readability
    titleFontSize=16,  # Increased axis title font size for better
 ↪  readability
    titleFontWeight='bold',  # Bold axis titles
    labelPadding=10  # Padding for axis labels
).configure_title(
    fontSize=18,  # Increased font size for chart title
    fontWeight='bold'  # Bold chart title
).configure_legend(
    titleFontSize=16,  # Increased font size for legend title
    labelFontSize=14,  # Increased font size for legend labels
    symbolSize=120  # Adjusted size of the legend symbols
)

# Save the chart as an interactive HTML file
jam_chart.save('jam_chart.html')

# Display the chart
jam_chart
```

```
alt.LayerChart(...)
```

5.

There are 11 subtype to type combinations according to our current analysis.

b.

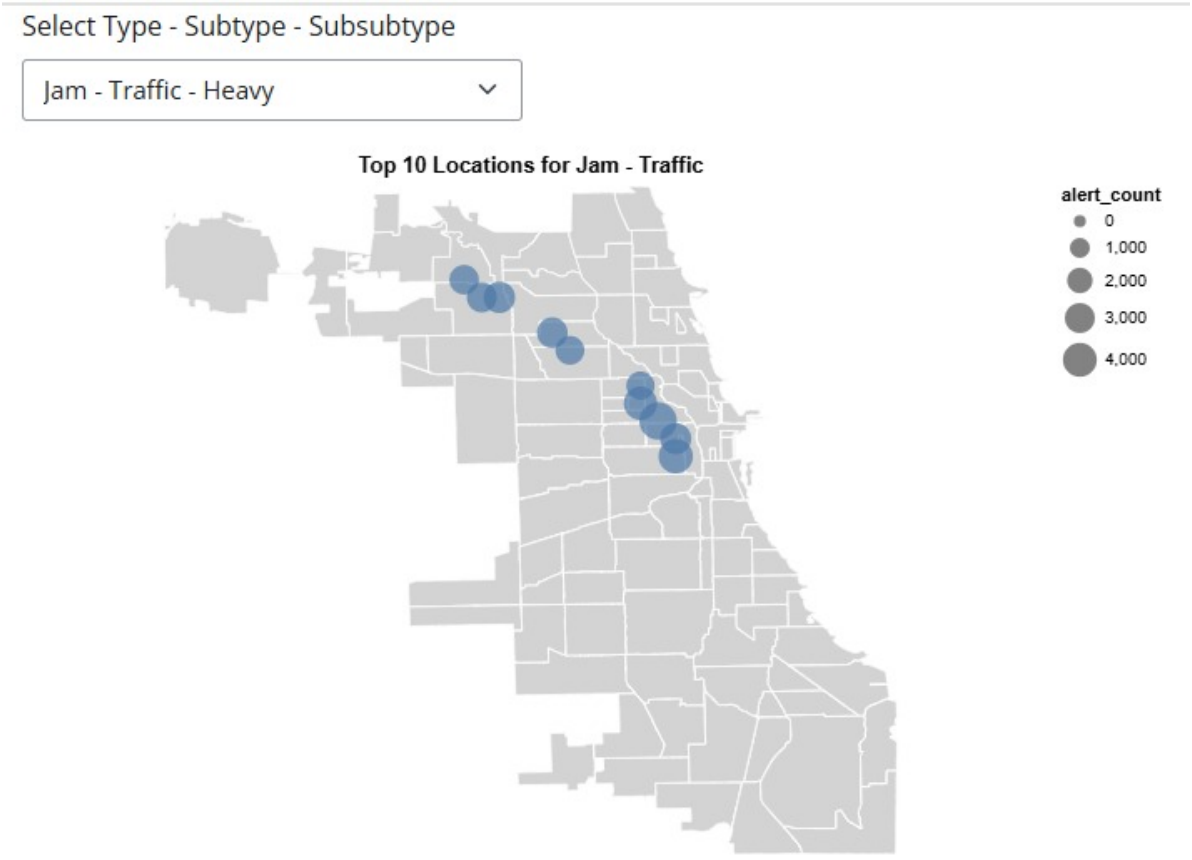

Figure 1: Dropdown

c. Use your dashboard to answer the following question: where are alerts for road closures due to events most common? Insert a screenshot as your answer below.

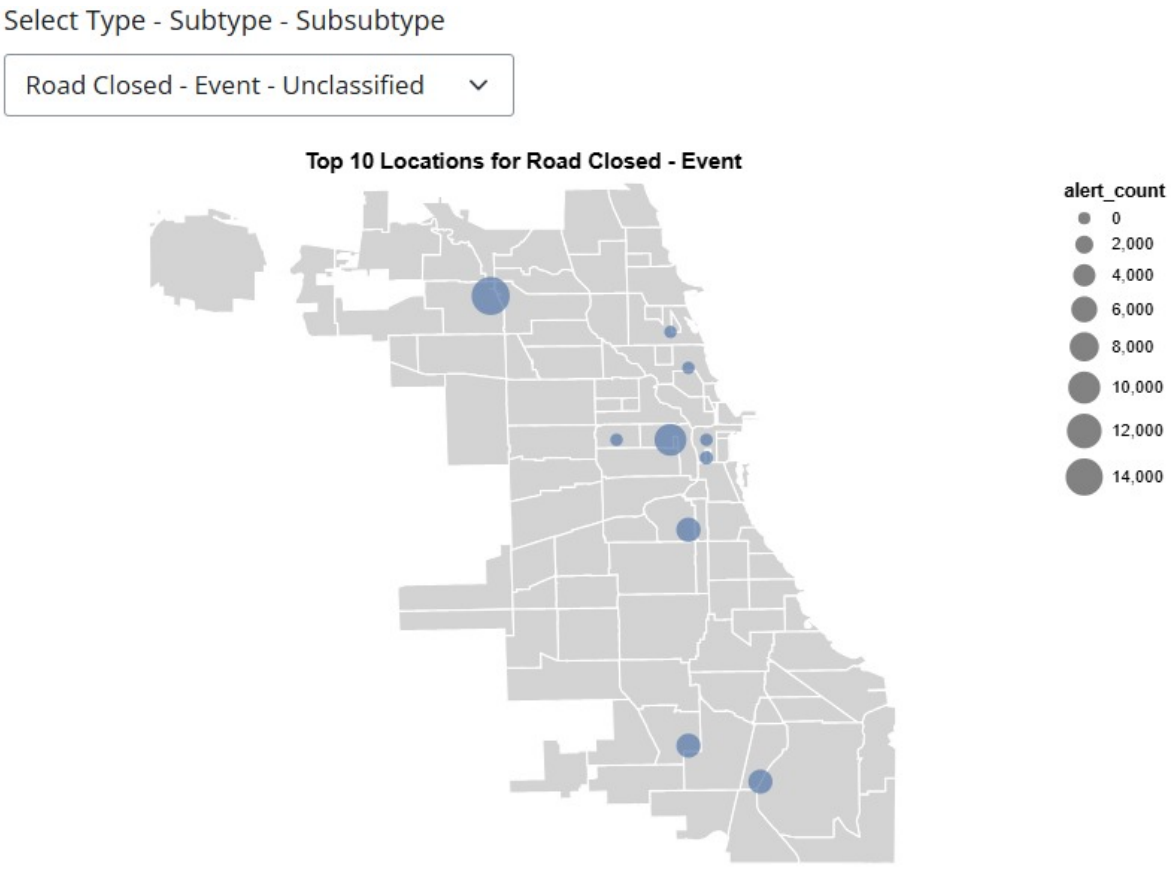

Figure 2: Closures

d. What is the most hazardous area in Chicago according to fogginess?

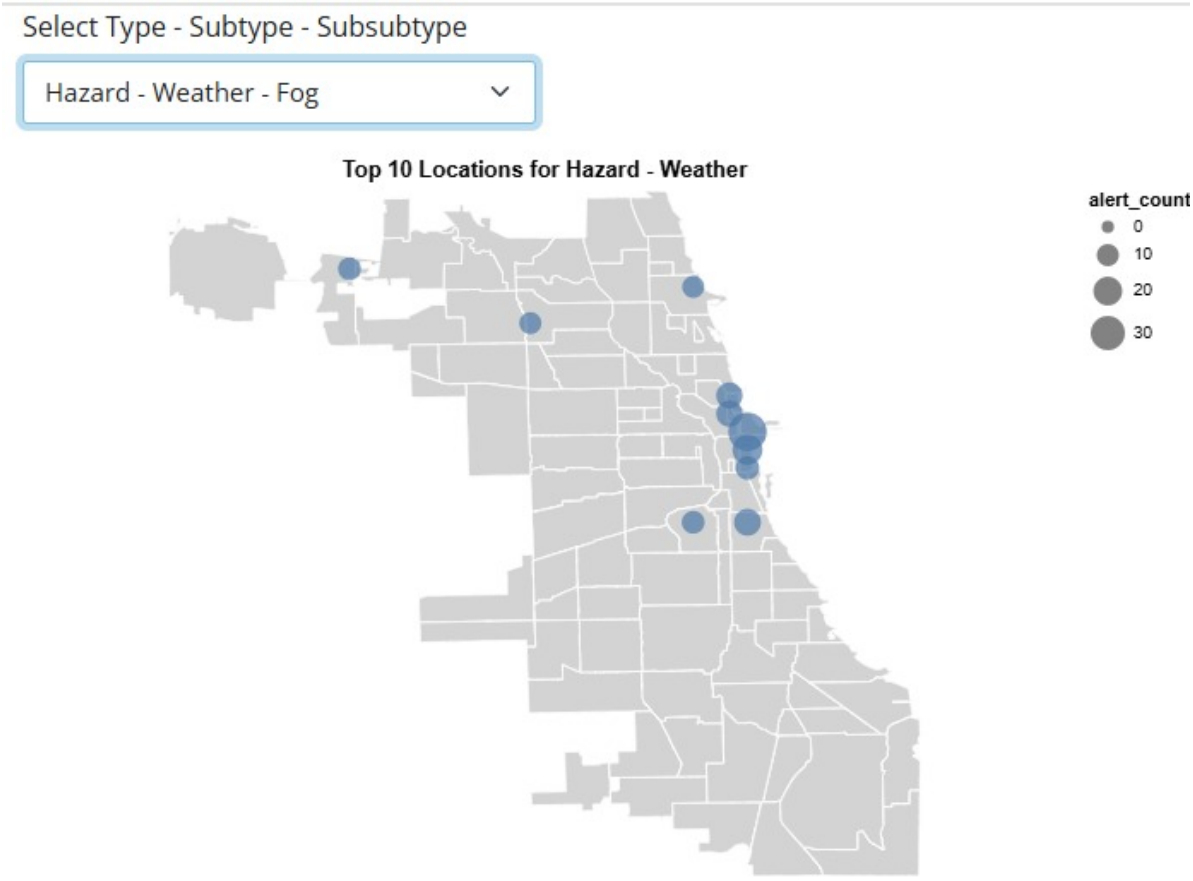

Figure 3: Fog

e. Can you suggest adding another column to the dashboard to enhance our analysis?

It would be great to have a column that also indicates the time of day when accidents / jams / closures are most frequent Another thing would be weather conditions that contributed to the accidents / jams / closures (even if there is a separate hazard category that tracks weather, it is likely also a causal factor for accidents and jams).

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

1.

Collapsing the data by ts (being the timestamp of the incident) would allow us to streamline a lot of the information and group it up into more digestible categories. Although we lose

out on the exact seconds and minutes by collapsing by the hour, hourly analysis on its own already gives us a great idea of these accident / jam trends and makes the process far more computationally accessible for shiny. As such, as long as we're ok sacrificing some precision, we can collapse to increase readability and reduce size of data.

b.

```
# Load the dataset from the specified path
waze_data_path = r'C:\Users\eddie\OneDrive\Documents\ps6\waze_data.csv'
waze_data_df = pd.read_csv(waze_data_path)

# Check if the dataset contains a 'geo' column
if 'geo' in waze_data_df.columns:
    # Extract longitude and latitude from the 'geo' column formatted as
     ↪ POINT(longitude latitude)
    # Use regex to extract numerical values for longitude and latitude
    pattern = r'POINT\((-?\d+\.\d+)\s(-?\d+\.\d+)\)'
    waze_data_df[['longitude', 'latitude']] =
 ↪  waze_data_df['geo'].str.extract(pattern).astype(float)
else:
    # Raise an error if the 'geo' column is missing
    raise ValueError("The dataset does not contain a 'geo' column.")


# Bin latitude and longitude into bins with step size 0.01
waze_data_df['binned_latitude'] = (waze_data_df['latitude'] // 0.01) * 0.01
waze_data_df['binned_longitude'] = (waze_data_df['longitude'] // 0.01) * 0.01
```

c.

```
# Convert 'ts' column to datetime and remove timezone info if needed
if waze_data_df['ts'].dtype != 'datetime64[ns]':
    waze_data_df['ts'] = pd.to_datetime(waze_data_df['ts'].str.replace("UTC",
 ↪  ""), errors='coerce')

# Extract the hour (rounded down to the start of the hour) from 'ts'
waze_data_df['hour'] = waze_data_df['ts'].dt.floor('H')
```

```
C:\Users\eddie\AppData\Local\Temp\ipykernel_15272\3166143818.py:6:
FutureWarning: 'H' is deprecated and will be removed in a future version,
please use 'h' instead.
  waze_data_df['hour'] = waze_data_df['ts'].dt.floor('H')
```

```
# Group and aggregate by hour, binned latitude, and longitude
aggregated_data = (
    waze_data_df.groupby(['hour', 'binned_latitude', 'binned_longitude'])
    .size()
    .reset_index(name='alert_count')
)

# Rank and filter top 10 alerts per hour
aggregated_data['rank'] =
 ↪  aggregated_data.groupby('hour')['alert_count'].rank(ascending=False,
 ↪  method='first')
top_10_per_hour = aggregated_data[aggregated_data['rank'] <=
 ↪  10].drop(columns='rank')

# Save the collapsed dataset
output_folder = r'C:\Users\eddie\OneDrive\Documents\ps6'
os.makedirs(output_folder, exist_ok=True)
output_path = os.path.join(output_folder, 'top_alerts_map_byhour.csv')
top_10_per_hour.to_csv(output_path, index=False)

print(f"Number of rows in the dataset: {len(top_10_per_hour)}")
```

```
Number of rows in the dataset: 65826
```

c.

```
import random
```

```
# Load the dataset
data_path =
 ↪  r'C:\Users\eddie\OneDrive\Documents\ps6\top_alerts_map_byhour.csv'
top_alerts_df = pd.read_csv(data_path)

# Convert 'hour' column to datetime and remove timezone info
top_alerts_df['hour'] =
 ↪  pd.to_datetime(top_alerts_df['hour']).dt.tz_localize(None)

# Select 3 random unique hours
unique_hours = top_alerts_df['hour'].unique()
random_hours = random.sample(list(unique_hours), 3)
print(f"Randomly selected hours: {random_hours}")
```

```
# Load GeoJSON data for Chicago boundaries
geojson_path = r'C:\Users\eddie\OneDrive\Documents\ps6\Boundaries -
 ↪  Neighborhoods.geojson'
with open(geojson_path) as f:
    chicago_geojson = json.load(f)
geo_data = alt.Data(values=chicago_geojson["features"])

# Create charts for each selected hour
jam_hour_charts = []
for hour in random_hours:
    hourly_data = top_alerts_df[top_alerts_df['hour'] == hour]
    print(f"\nData for hour {hour}: {hourly_data}")  # Debugging: Check
     ↪  filtered data

    if hourly_data.empty:
        print(f"No data available for hour {hour}. Skipping...")
        continue

    # Base map layer
    base_map = alt.Chart(geo_data).mark_geoshape(
        fill='lightgray',
        stroke='white'
    ).properties(
        width=600,
        height=400
    )

    # Points layer for top alert locations
    points_layer = alt.Chart(hourly_data).mark_circle().encode(
        longitude='binned_longitude:Q',
        latitude='binned_latitude:Q',
        size=alt.Size('alert_count:Q', scale=alt.Scale(range=[100, 1000])),
        color=alt.value('red'),
        tooltip=['binned_longitude', 'binned_latitude', 'alert_count']
    )

    # Combine map and points layer
    jam_hour_chart = alt.layer(base_map, points_layer).project(
        type='mercator',
        scale=50000,
        center=[-87.65, 41.88]  # Center of Chicago
```

```
    ).properties(
        title=f"Top Locations for Alerts at {hour}"
    )

    jam_hour_charts.append((jam_hour_chart, hour))

# Save charts as HTML files
output_folder = r'C:\Users\eddie\OneDrive\Documents\ps6'
os.makedirs(output_folder, exist_ok=True)

for chart, hour in jam_hour_charts:
    output_path = os.path.join(output_folder,
 ↪  f'jam_hour_chart_{hour.strftime("%Y-%m-%d_%H-%M-%S")}.html')
    chart.save(output_path)
    print(f"Chart saved to: {output_path}")

jam_hour_chart.show()
```

```
Randomly selected hours: [Timestamp('2024-06-03 10:00:00'),
Timestamp('2024-01-06 22:00:00'), Timestamp('2024-07-28 20:00:00')]

Data for hour 2024-06-03 10:00:00:                      hour  binned_latitude
binned_longitude  alert_count
34853 2024-06-03 10:00:00           41.69           -87.60            3
34854 2024-06-03 10:00:00           41.71           -87.64            3
34855 2024-06-03 10:00:00           41.71           -87.60            2
34856 2024-06-03 10:00:00           41.81           -87.75            2
34857 2024-06-03 10:00:00           41.81           -87.74            2
34858 2024-06-03 10:00:00           41.82           -87.71            2
34859 2024-06-03 10:00:00           41.82           -87.64            3
34860 2024-06-03 10:00:00           41.83           -87.64            3
34861 2024-06-03 10:00:00           41.88           -87.65            6
34862 2024-06-03 10:00:00           41.96           -87.75           12

Data for hour 2024-01-06 22:00:00:                      hour  binned_latitude
binned_longitude  alert_count
1105 2024-01-06 22:00:00           41.78           -87.64            3
1106 2024-01-06 22:00:00           41.84           -87.63            3
1107 2024-01-06 22:00:00           41.91           -87.68            6
1108 2024-01-06 22:00:00           41.92           -87.68            4
1109 2024-01-06 22:00:00           41.94           -87.72            4
1110 2024-01-06 22:00:00           41.94           -87.66           11
```

```
1111 2024-01-06 22:00:00            41.94            -87.65            9
1112 2024-01-06 22:00:00            41.95            -87.66            7
1113 2024-01-06 22:00:00            41.95            -87.65           11
1114 2024-01-06 22:00:00            41.96            -87.76            4

Data for hour 2024-07-28 20:00:00:                     hour  binned_latitude
binned_longitude  alert_count
48046 2024-07-28 20:00:00            41.72            -87.63            4
48047 2024-07-28 20:00:00            41.73            -87.63            5
48048 2024-07-28 20:00:00            41.78            -87.64            4
48049 2024-07-28 20:00:00            41.80            -87.64            5
48050 2024-07-28 20:00:00            41.83            -87.67            5
48051 2024-07-28 20:00:00            41.84            -87.64            5
48052 2024-07-28 20:00:00            41.86            -87.65            6
48053 2024-07-28 20:00:00            41.87            -87.66            6
48054 2024-07-28 20:00:00            41.89            -87.66            6
48055 2024-07-28 20:00:00            41.90            -87.67            7
Chart saved to:
C:\Users\eddie\OneDrive\Documents\ps6\jam_hour_chart_2024-06-03_10-00-00.html
Chart saved to:
C:\Users\eddie\OneDrive\Documents\ps6\jam_hour_chart_2024-01-06_22-00-00.html
Chart saved to:
C:\Users\eddie\OneDrive\Documents\ps6\jam_hour_chart_2024-07-28_20-00-00.html

alt.LayerChart(...)
```

2.

I encountered significant rendering issues with App 2. See below for print statement.

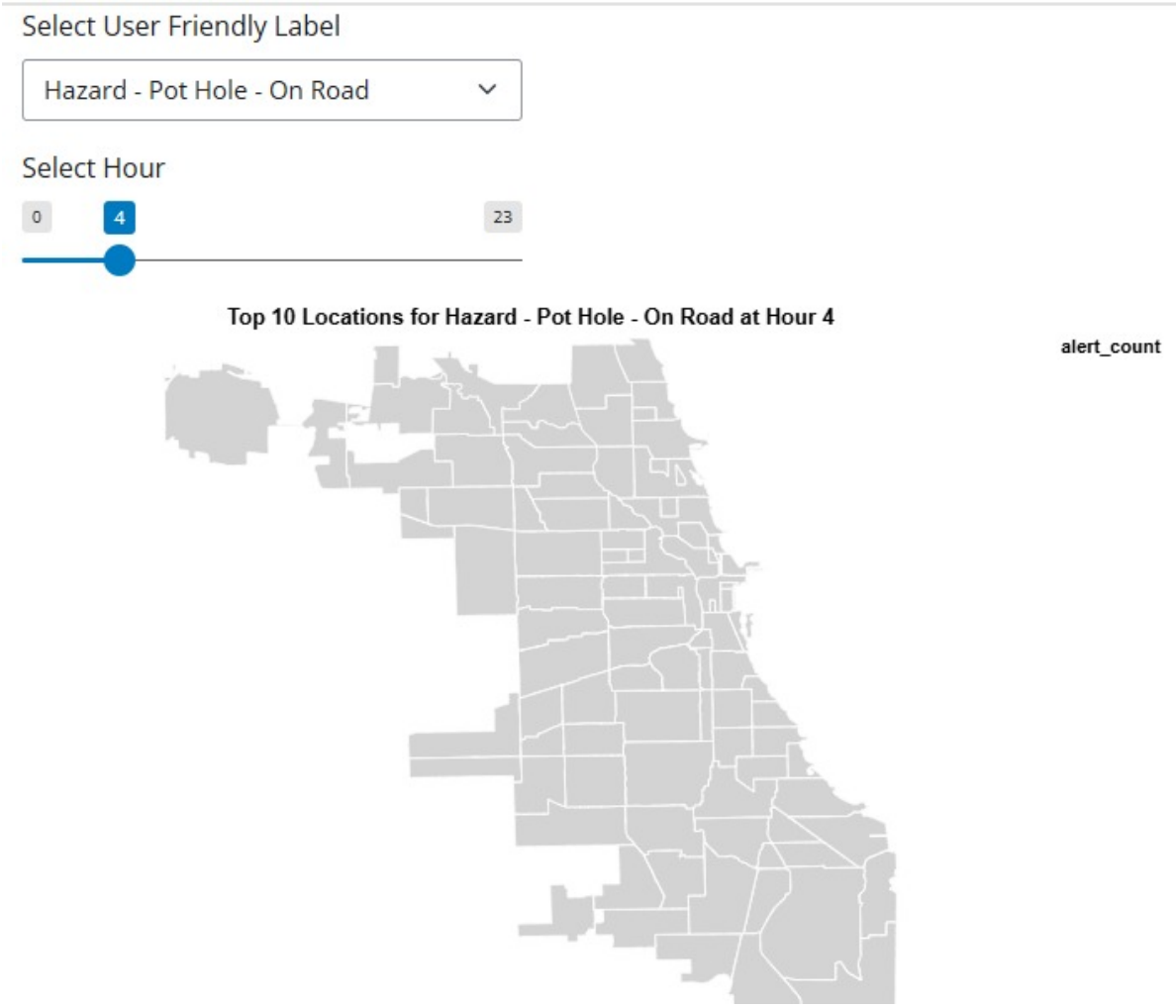

Figure 4: App 2

a.

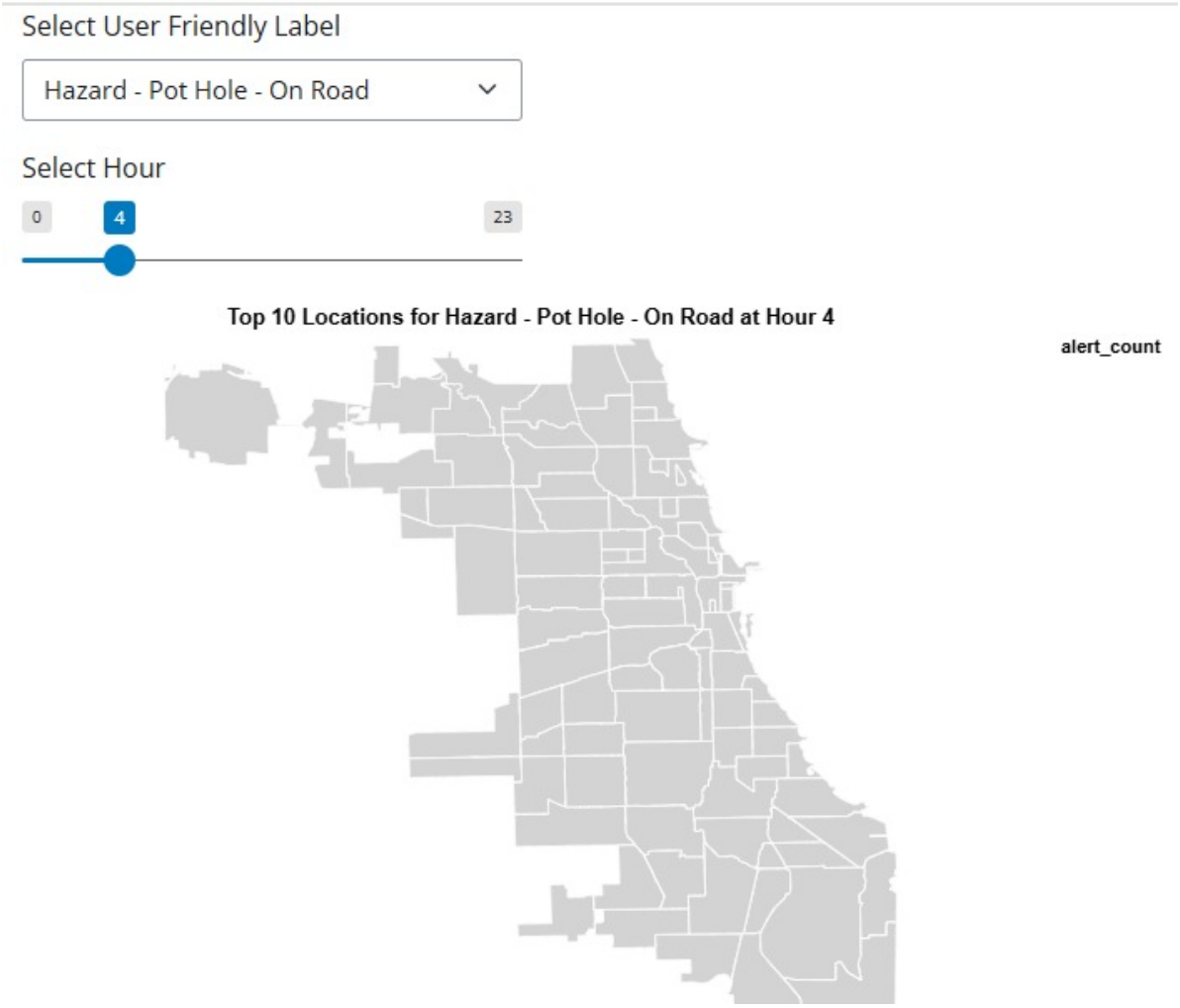

Figure 5: App 2

b.

c. Due to no points being displayed, I could only answer this question with outside research: As expected, road construction is far more prevalent at night. Anecdotally, this makes sense. We could imagine that road construction during the day would be a lot more disruptive, meaning that nighttime construction will occur more often to avoid that disturbance.

App 2 didn't load properly. Here is its contents:

from shiny import App, ui from shinywidgets import render_altair, output_widget import pandas as pd import altair as alt import os import json

Load merged_df.csv (for user_friendly_label dropdown)

merged_df_path = os.path.join(directory, “merged_df.csv”) merged_df = pd.read_csv(merged_df_path)

Load top_alerts_map_byhour.csv (for slider filtering)

top_alerts_map_byhour_path = os.path.join(directory, “top_alerts_map_byhour.csv”) top_alerts_map_byhour = pd.read_csv(top_alerts_map_byhour_path)

Debug: Print column names to verify structure

print(“merged_df columns:”, merged_df.columns) print(“top_alerts_map_byhour columns:”, top_alerts_map_byhour.columns)

Load Chicago boundaries GeoJSON

with open(os.path.join(directory, “Boundaries - Neighborhoods.geojson”)) as f: chicago_geojson = json.load(f)

geo_data = alt.Data(values=chicago_geojson[“features”])

Create a list of unique user_friendly_label values for dropdown

user_friendly_labels = merged_df[‘user_friendly_label’].unique().tolist()

Debug: Print the user-friendly label options to ensure it’s correct

print(“Dropdown options (user_friendly_label):”, user_friendly_labels)

Define the UI

app_ui = ui.page_fluid(ui.input_select(“user_friendly_label”, “Select User Friendly Label”, user_friendly_labels), ui.input_slider(“selected_hour”, “Select Hour”, min=0, max=23, value=12, step=1), output_widget(“map_plot”))

Define the server logic

def server(input, output, session): @output @render_altair def map_plot(): # Get user inputs using the correct method for Shiny selected = input‘user_friendly_label’ # Access the dropdown value print(“Selected dropdown value (user_friendly_label):”, selected) # Debugging line to see the selected value

```
selected_hour = input['selected_hour']()  # Access the slider value

# Filter `merged_df` for selected user_friendly_label
merged_filtered = merged_df[merged_df['user_friendly_label'] == selected]

# Filter `top_alerts_map_byhour` for selected hour
hour_filtered = top_alerts_map_byhour[top_alerts_map_byhour['hour'] ==
selected_hour]

# Merge the datasets on `binned_latitude` and `binned_longitude`
combined_data = pd.merge(
    merged_filtered,
    hour_filtered,
    on=['binned_latitude', 'binned_longitude'],  # Match locations
    how='inner'
)

# Debugging: Print merged data
print("Merged data (combined_data):")
print(combined_data.head())  # Show a few rows to check

# Aggregate and get top 10 locations
aggregated = combined_data.groupby(['binned_latitude',
'binned_longitude',
'user_friendly_label']).size().reset_index(name='alert_count')
top_10 = aggregated.nlargest(10, 'alert_count')

# Base map using identity projection and flipped Y-axis
base = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).project(
    type='identity',  # Identity projection
    reflectY=True  # Flip y-axis
).properties(
    width=600,
```

```
        height=400
    )

    # Points layer with user_friendly_label for color
    points = alt.Chart(top_10).mark_circle().encode(
        longitude='binned_longitude:Q',
        latitude='binned_latitude:Q',
        size=alt.Size('alert_count:Q', scale=alt.Scale(range=[50, 500])),
        color=alt.Color('user_friendly_label:N', legend=None),  # Color by
        user_friendly_label
        tooltip=['binned_longitude', 'binned_latitude', 'alert_count',
        'user_friendly_label']  # Show label in tooltip
    )

    # Combine the base map and points layers
    chart = alt.layer(base, points).properties(
        width=600,
        height=400,
        title=f'Top 10 Locations for {selected} at Hour {selected_hour}'
    ).configure_view(
        strokeWidth=0
    ).configure_axis(
        grid=False
    )

    return chart
```

Create the app

app = App(app_ui, server)

Run the app

if **name** == "**main**": app.run()

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

1.

a. Collapsing the data in this case is a bit trickier since the user can select any range of hourly values. This means that our timestamps need to be more flexible so they may be considered in their appropriate time slot when the range is specified. This would make things more data heavy and computationally intensive, but at least we'd ensure our time ranges are always capturing all the right entries.

b.

```
# Load your dataset
data_path =
 ↪  r'C:\Users\eddie\OneDrive\Documents\ps6\top_alerts_map_byhour.csv'
top_alerts_df = pd.read_csv(data_path)

# Remove timezone information from 'hour' column
top_alerts_df['hour'] =
 ↪  pd.to_datetime(top_alerts_df['hour']).dt.tz_localize(None)

# Specify the 3 specific hours you want to focus on (between 6AM-9AM)
specific_hours = ['06:00', '07:00', '08:00', '09:00']  # Modify this list to
 ↪  the specific hours you want

print(f"Selected specific hours: {specific_hours}")

# Load GeoJSON data for Chicago boundaries
geojson_path = r'C:\Users\eddie\OneDrive\Documents\ps6\Boundaries -
 ↪  Neighborhoods.geojson'
with open(geojson_path) as f:
    chicago_geojson = json.load(f)
geo_data = alt.Data(values=chicago_geojson["features"])

# Initialize an empty list to store charts
jam_hour_charts = []

for hour in specific_hours:
    # Filter data for the specific hour
    hourly_data = top_alerts_df[top_alerts_df['hour'].dt.strftime('%H:%M') ==
 ↪  hour]
```

```
    # Sort by alert_count and select the top 10 rows
    hourly_data_top_10 = hourly_data.sort_values(by='alert_count',
↪   ascending=False).head(10)

    # Debugging: Print filtered data
    print(f"\nData for hour {hour}:")
    print(hourly_data_top_10)

    if hourly_data_top_10.empty:
        print(f"No data available for hour {hour}. Skipping...")
        continue

    # Create map layer (base map) using the Chicago GeoJSON
    base_map = alt.Chart(geo_data).mark_geoshape(
        fill='lightgray',
        stroke='white'
    ).properties(
        width=600,
        height=400
    )

    # Add points layer for top locations (alerts)
    points_layer = alt.Chart(hourly_data_top_10).mark_circle().encode(
        longitude='binned_longitude:Q',
        latitude='binned_latitude:Q',
        size=alt.Size('alert_count:Q', scale=alt.Scale(range=[10, 100])),
        color=alt.value('red'),
        tooltip=['binned_longitude', 'binned_latitude', 'alert_count']
    )

    # Combine base map and points layer for the final chart
    jam_hour_chart = alt.layer(base_map, points_layer).project(
        type='mercator',
        scale=50000,
        center=[-87.65, 41.88]  # Approximate center of Chicago
    ).properties(
        title=f"Top 10 Locations for Alerts at {hour}",
        width=600,
        height=400
    )

    # Append the chart to the list
```

```
    jam_hour_charts.append((jam_hour_chart, hour))

# Save each chart as a PNG file using kaleido
output_folder = r'C:\Users\eddie\OneDrive\Documents\ps6'
os.makedirs(output_folder, exist_ok=True)

for i, (chart, hour) in enumerate(jam_hour_charts):
    output_path = os.path.join(output_folder,
    ↪ f'top_alerts_map_byhour_sliderrange_{hour.replace(":", "-")}.png')
    chart.save(output_path, renderer='kaleido', scale=2.0)  # Using kaleido
    ↪ to save as PNG
    print(f"Chart saved to: {output_path}")

jam_hour_chart.show()
```

```
Selected specific hours: ['06:00', '07:00', '08:00', '09:00']

Data for hour 06:00:
                      hour  binned_latitude  binned_longitude  alert_count
54942 2024-08-27 06:00:00            41.80            -87.64           15
28364 2024-05-07 06:00:00            41.96            -87.75           13
28124 2024-05-06 06:00:00            41.96            -87.75           13
28840 2024-05-09 06:00:00            41.96            -87.75           12
39371 2024-06-22 06:00:00            41.96            -87.75           12
40091 2024-06-25 06:00:00            41.96            -87.75           12
39131 2024-06-21 06:00:00            41.96            -87.75           12
38890 2024-06-20 06:00:00            41.96            -87.75           12
27884 2024-05-05 06:00:00            41.96            -87.75           12
39851 2024-06-24 06:00:00            41.96            -87.75           12

Data for hour 07:00:
                      hour  binned_latitude  binned_longitude  alert_count
53792 2024-08-22 07:00:00            41.88            -87.68           25
48400 2024-07-30 07:00:00            41.87            -87.67           20
48395 2024-07-30 07:00:00            41.80            -87.64           19
25302 2024-04-24 07:00:00            41.86            -87.65           19
29568 2024-05-12 07:00:00            41.88            -87.65           16
34352 2024-06-01 07:00:00            41.96            -87.75           13
36272 2024-06-09 07:00:00            41.96            -87.75           13
34111 2024-05-31 07:00:00            41.96            -87.75           13
31485 2024-05-20 07:00:00            41.96            -87.75           12
35312 2024-06-05 07:00:00            41.96            -87.75           12
```

```
Data for hour 08:00:
                     hour  binned_latitude  binned_longitude  alert_count
45090 2024-07-16 08:00:00            41.87            -87.67           19
32449 2024-05-24 08:00:00            41.96            -87.75           14
38431 2024-06-18 08:00:00            41.96            -87.75           13
28381 2024-05-07 08:00:00            41.96            -87.75           13
30539 2024-05-16 08:00:00            41.96            -87.75           13
31015 2024-05-18 08:00:00            41.96            -87.75           13
36281 2024-06-09 08:00:00            41.96            -87.75           13
27424 2024-05-03 08:00:00            41.96            -87.75           12
28144 2024-05-06 08:00:00            41.96            -87.75           12
37471 2024-06-14 08:00:00            41.96            -87.75           12

Data for hour 09:00:
                     hour  binned_latitude  binned_longitude  alert_count
36769 2024-06-11 09:00:00            41.82            -87.64           17
30785 2024-05-17 09:00:00            41.96            -87.75           14
38200 2024-06-17 09:00:00            41.96            -87.75           13
30069 2024-05-14 09:00:00            41.96            -87.75           13
31982 2024-05-22 09:00:00            41.96            -87.75           13
33179 2024-05-27 09:00:00            41.96            -87.75           13
32223 2024-05-23 09:00:00            41.96            -87.75           13
35572 2024-06-06 09:00:00            41.96            -87.75           13
34132 2024-05-31 09:00:00            41.96            -87.75           13
34612 2024-06-02 09:00:00            41.96            -87.75           13
Chart saved to:
C:\Users\eddie\OneDrive\Documents\ps6\top_alerts_map_byhour_sliderrange_06-00.png
Chart saved to:
C:\Users\eddie\OneDrive\Documents\ps6\top_alerts_map_byhour_sliderrange_07-00.png
Chart saved to:
C:\Users\eddie\OneDrive\Documents\ps6\top_alerts_map_byhour_sliderrange_08-00.png
Chart saved to:
C:\Users\eddie\OneDrive\Documents\ps6\top_alerts_map_byhour_sliderrange_09-00.png

alt.LayerChart(...)
```

2.

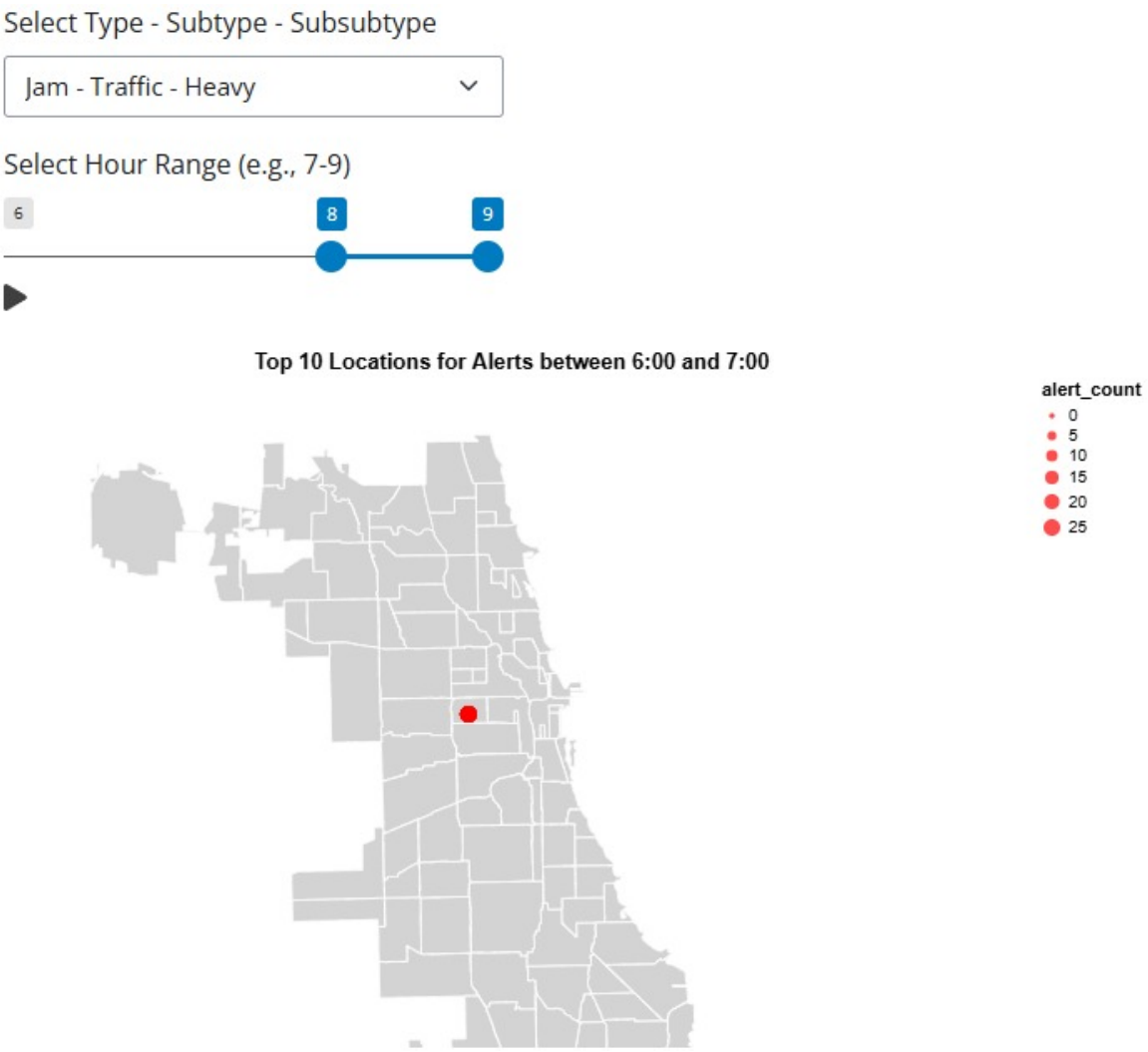

Figure 6: Jam

a.

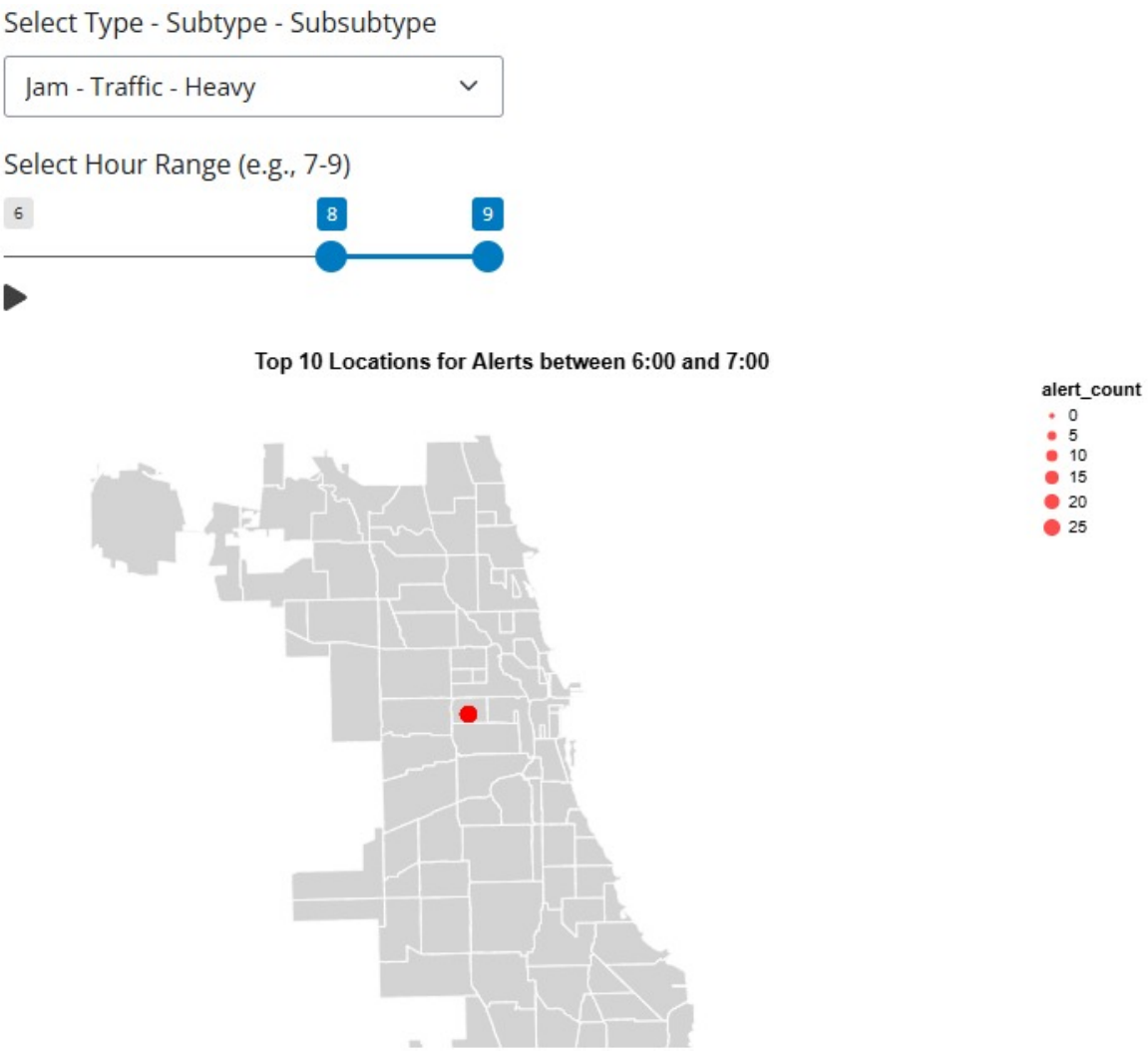

Figure 7: Jam

b.

3.

a. A switch button will allow us to alternate between either a range of hours when turned on or a single hour when turned off. This allows us to have both range or specific time on one user interface. Having the button on would correspond to True while having the button off would correspond to False.

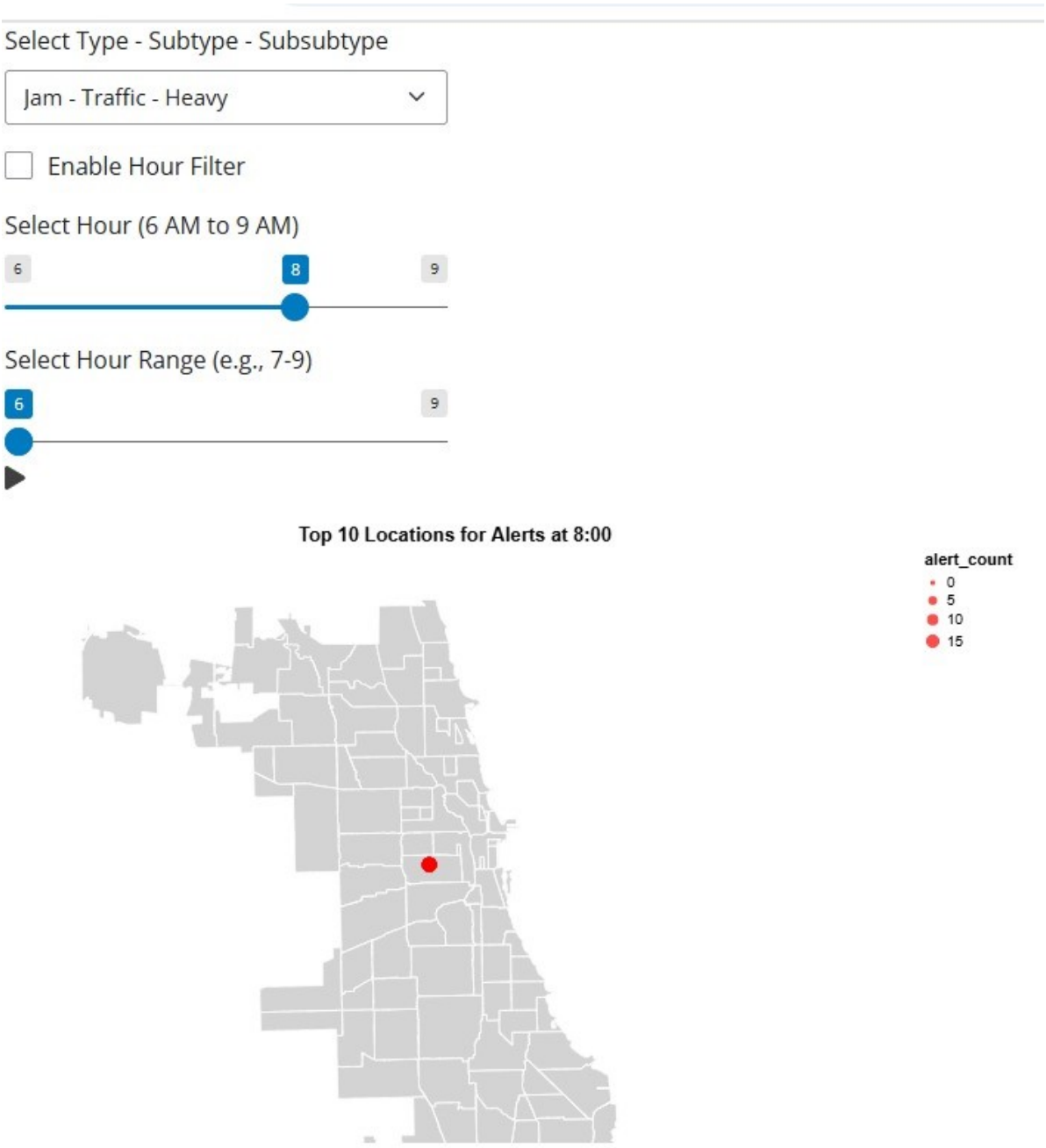

Figure 8: App 3 b

b.

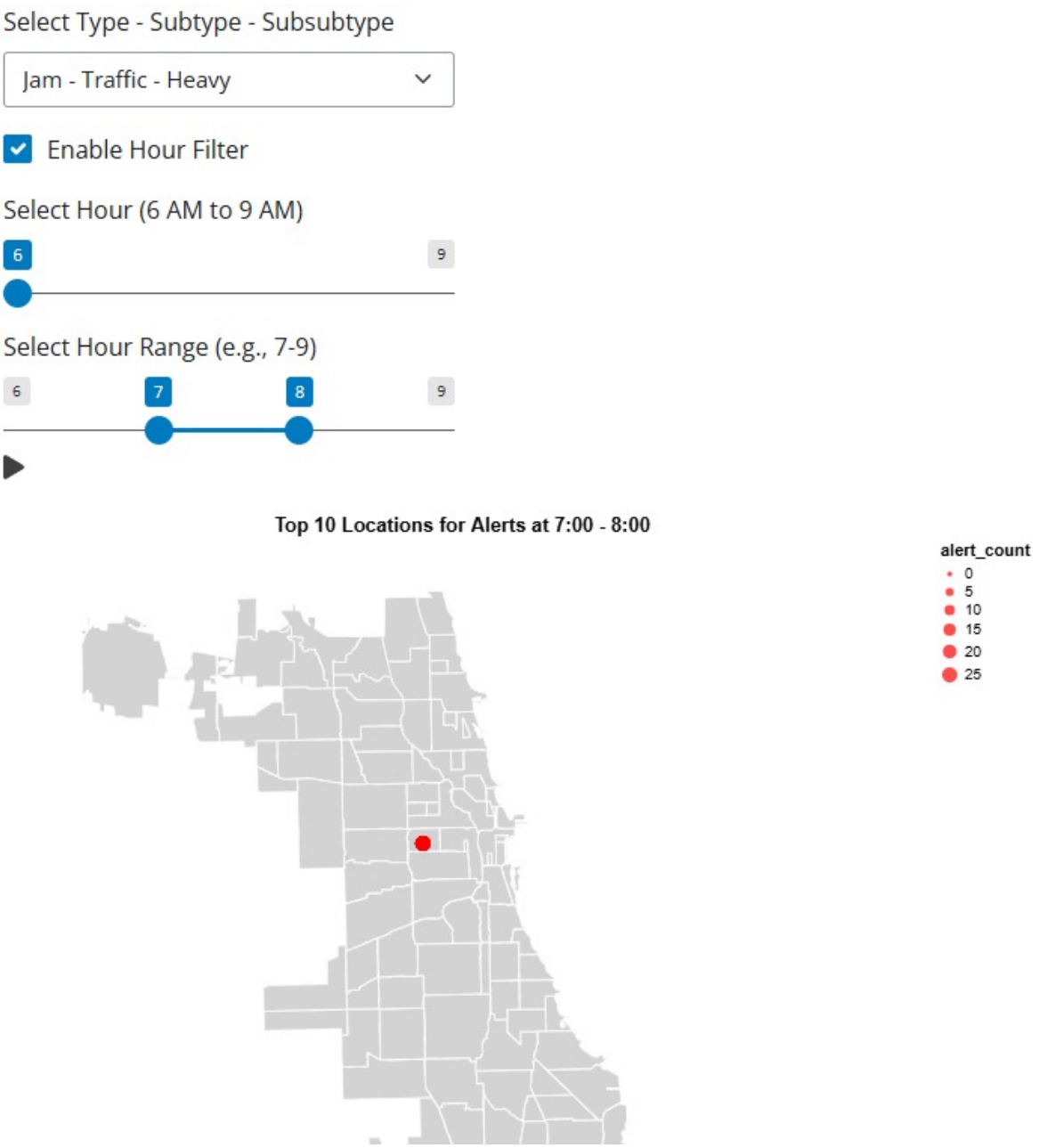

Figure 9: App 3 b Part 2

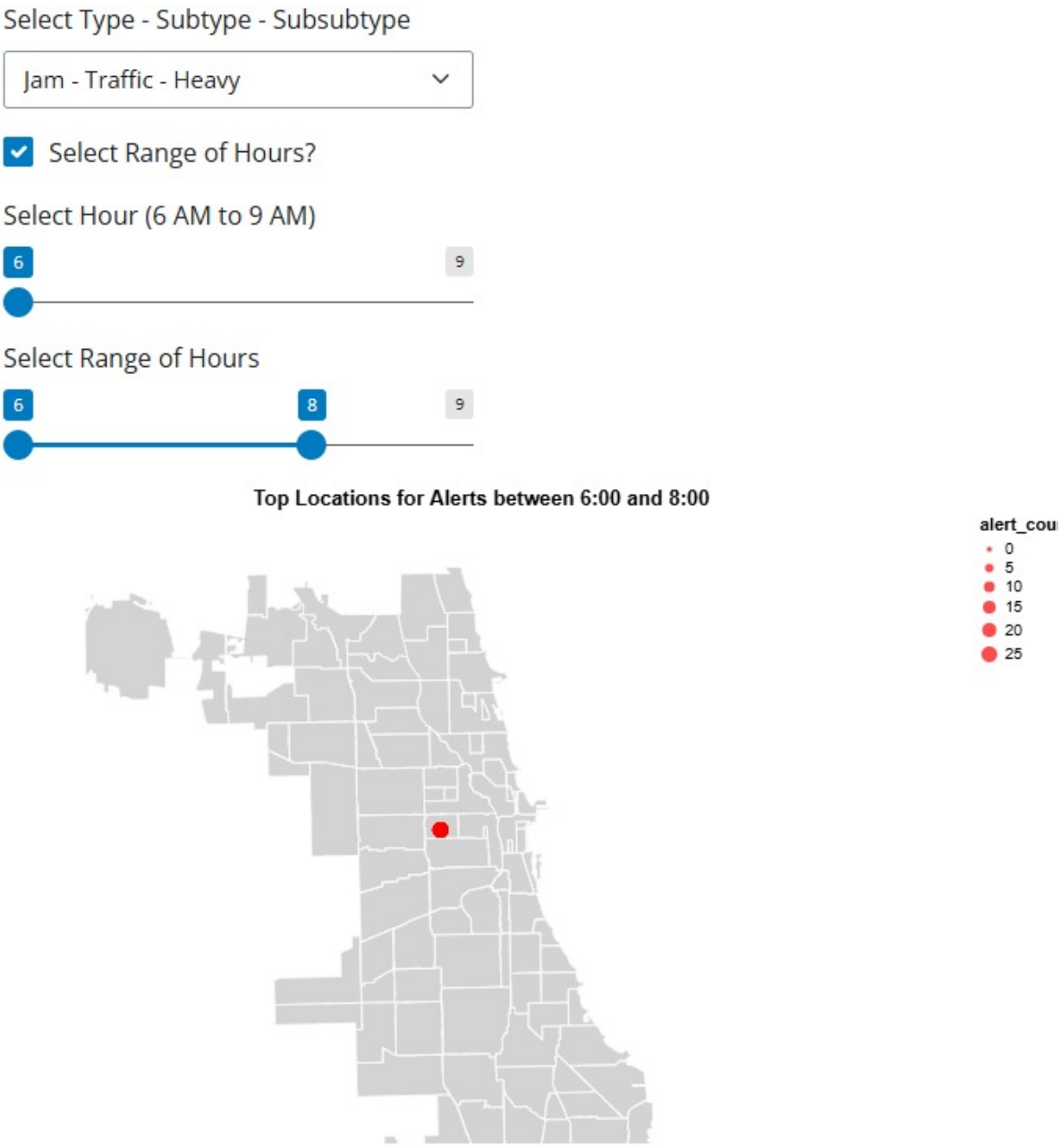

Figure 10: App 3 c 1

c.

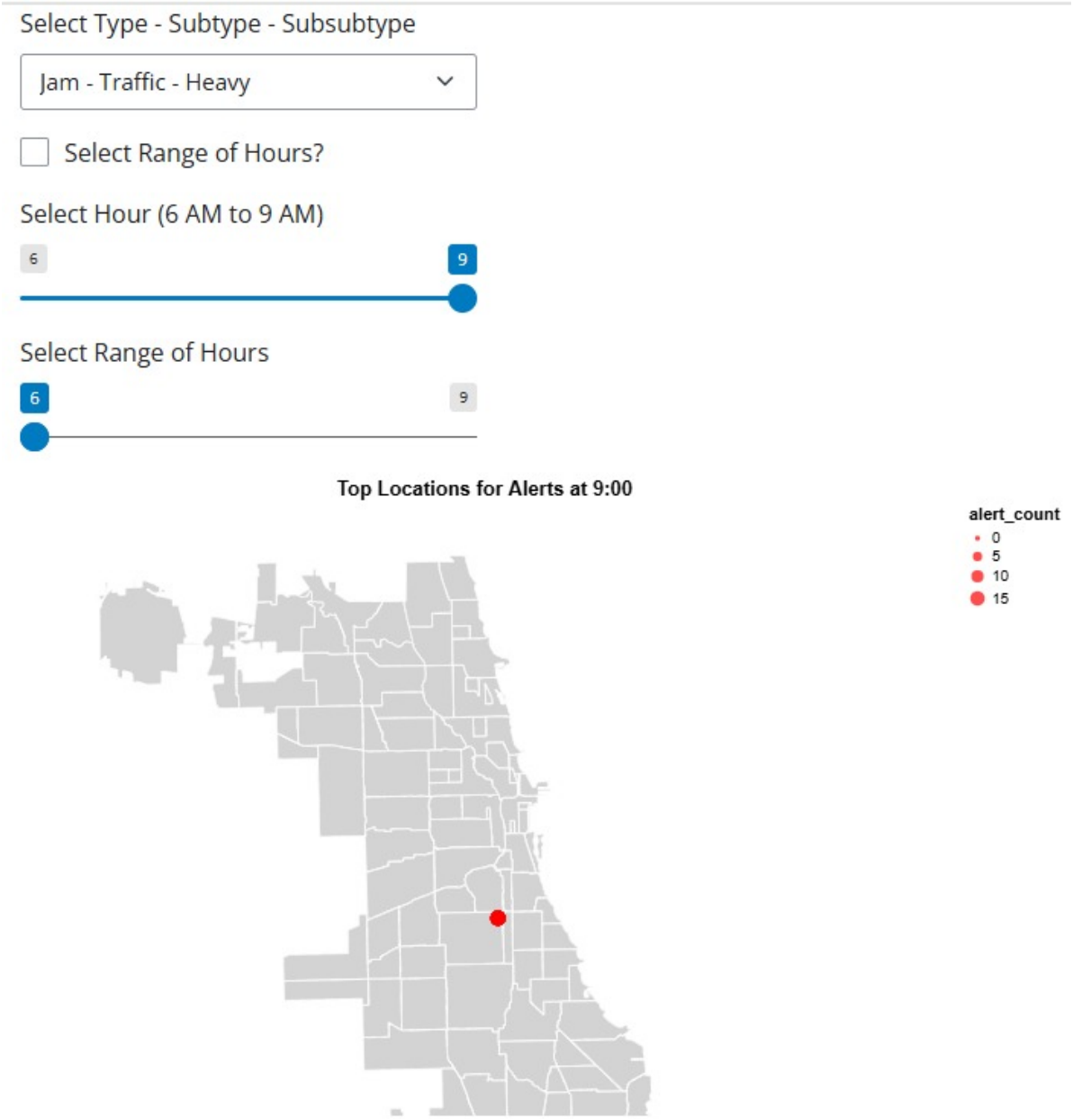

Figure 11: App 3 c 2

d. d. We'll divide the data into two subsets called morning (e.g., 12:00 AM to 12:00 PM) and afternoon (12:00 PM to 11:59 PM). We can add toggle buttons for the user to turn on/off the display of morning and afternoon points. When one is selected, the plot will show only that subset, while both on will display both sets with distinct color coding (e.g., blue for morning, red for afternoon). This way, users can focus

on either time period or view both simultaneously. If the dataset is large, we should pre-filter the data to keep things efficient. Additionally, color coding helps distinguish between morning and afternoon points, which may be beneficial for user experience.