Problem Set 6 - Waze Shiny Dashboard

Clarice Tee

2024-11-23

Statement of integrity: I used the slides on shiny app creation, chatgpt, and https://shiny.posit.co/py/docs/overv to guide me and troubleshoot/debug.

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: * CT *
2. "I have uploaded the names of anyone I worked with on the problem set **here**" (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.

Background

Data Download and Exploration (20 points)

1.

```
import zipfile
```

```
zip_file_path = r'C:\Users\clari\OneDrive\Documents\Python II\problem set
↪  6\waze_data.zip'

# Extract the ZIP file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(r'C:\Users\clari\OneDrive\Documents\Python II\problem
↪  set 6')  # Ensure the path is valid

# Read the extracted sample CSV file
sample_csv_path = r'C:\Users\clari\OneDrive\Documents\Python II\problem set
↪  6\waze_data_sample.csv'
waze_sample_df = pd.read_csv(sample_csv_path)

# Inspect the DataFrame
print(waze_sample_df.head())
```

```
   Unnamed: 0         city  confidence  nThumbsUp        street  \
0      584358  Chicago, IL           0        NaN           NaN
1      472915  Chicago, IL           0        NaN        I-90 E
2      550891  Chicago, IL           0        NaN        I-90 W
3      770659  Chicago, IL           0        NaN           NaN
4      381054  Chicago, IL           0        NaN  N Pulaski Rd

                                   uuid country         type  \
0  c9b88a12-79e8-44cb-aadd-a75855fc4bcb      US          JAM
1  7c634c0a-099c-4262-b57f-e893bdebce73      US  ROAD_CLOSED
2  7aa3c61a-f8dc-4fe8-bbb0-db6b9e0dc53b      US       HAZARD
3  3b95dd2f-647c-46de-b4e1-8ebc73aa9221      US       HAZARD
4  13a5e230-a28a-4bf4-b928-bc1dd38850e0      US          JAM

                        subtype  roadType  reliability  magvar  \
0                           NaN        17            5     116
1             ROAD_CLOSED_EVENT         3            6     173
```

```
2  HAZARD_ON_SHOULDER_CAR_STOPPED         3            5     308
3                   HAZARD_ON_ROAD        20            5     155
4               JAM_HEAVY_TRAFFIC         7            5     178

   reportRating                       ts                          geo  \
0             5  2024-07-02 18:27:40 UTC   POINT(-87.64577 41.892743)
1             0  2024-06-16 10:13:19 UTC  POINT(-87.646359 41.886295)
2             5  2024-05-02 19:01:47 UTC   POINT(-87.695982 41.93272)
3             2  2024-03-25 18:53:24 UTC  POINT(-87.669253 41.904497)
4             2  2024-06-03 21:17:33 UTC  POINT(-87.728322 41.978769)

                       geoWKT
0   Point(-87.64577 41.892743)
1  Point(-87.646359 41.886295)
2   Point(-87.695982 41.93272)
3  Point(-87.669253 41.904497)
4  Point(-87.728322 41.978769)
```

These are the data types: - Unnamed: 0: Nominal - city: Nominal - confidence: Quantitative - nThumbsUp: Quantitative - street: Nominal - uuid: Nominal - country: Nominal - type: Nominal - subtype: Nominal - roadType: Quantitative - reliability: Quantitative - magvar: Quantitative - reportRating: Quantitative

2.

```
waze_data_path = r'C:\Users\clari\OneDrive\Documents\Python II\problem set
 ↪  6\waze_data.csv'

waze_data_df = pd.read_csv(waze_data_path)

# Count missing and non-missing values for each column
missing_data_summary = pd.DataFrame({
    'Variable': waze_data_df.columns,
    'NULL': waze_data_df.isnull().sum(),
    'Not NULL': waze_data_df.notnull().sum()
})

# Transform to long format for Altair visualization
long_data = missing_data_summary.melt(
    id_vars='Variable',
    value_vars=['NULL', 'Not NULL'],
    var_name='Status',
    value_name='Count'
```

```
)

# Create the stacked bar chart
chart = alt.Chart(long_data).mark_bar().encode(
    x=alt.X('Variable', sort=None, title='Variables'),
    y=alt.Y('Count', title='Number of Observations'),
    color=alt.Color('Status', scale=alt.Scale(scheme='category10'),
 ↪  title='Observation Status'),
    tooltip=['Variable', 'Status', 'Count']
).properties(
    title='Missing vs Non-Missing Observations by Variable',
    width=800,
    height=400
)

chart
```

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

I can observe NULL values for nThumbsUp, some in street, and some in subtype. The variable with the highest share of NULLs is nThumbsUp, which consists almost entirely of NULLs.

3. Taking a look at the values

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

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

# Count types with NA subtypes
na_subtypes_count =
 ↪  waze_data_df[waze_data_df['subtype'].isna()]['type'].nunique()
print(f"\nNumber of types with NA subtypes: {na_subtypes_count}")

# Identify types with potential sub-subtypes
subtypes_for_types =
 ↪  waze_data_df[waze_data_df['subtype'].notna()].groupby('type')['subtype'].unique()
```

```
print("\nSubtypes for each type:")
print(subtypes_for_types)
```

```
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
```

a. All four main types have at least one instance of NA subtypes. There are multiple examples of 'sub-sub-types'. For example, the type Hazard specifies On Road, which can then further specify into what the exact hazard was.

Hierarcy * Jam * Traffic * Standstill * Heavy * Moderate * Light * Unclasified * Road Closed * Construction * Event * Accident * Unclasified * Accident * Major * Minor * Reported * Hazard * On Road * Specific HAzards * On Shoulder * Specific Hazars * Weather * Weather conditinos * Note on Map * General * Point of Interest

I think we should keep the NA subtypes and code them as "Unclassified". This would mean that we keep all the information on reported events, which we could use in the future. Right now, we could find patterns or gaps in the reporting system, and maintains transparency about data

limitations. NAs could, afterall, be due to incomplete reports from users in a hurry, ambiguous that don't fit neatly into existing categories, or technical issues in data processing.

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

4.
5.

```
# 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': ''
})
```

2.

```
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
```

3.

```
# Merge the crosswalk with the original data
merged_df = r'C:\Users\clari\OneDrive\Documents\Python II\problem set
 ↪  6\merged_df.csv'
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
```

There are zero rows for which accident is unclassified.

4.

```
# Check if all type and subtype combinations in the merged dataset exist in
 ↪  the crosswalk
merged_combinations = merged_df[['type', 'subtype']].drop_duplicates()
crosswalk_combinations = crosswalk_df[['type', 'subtype']]

are_equal = merged_combinations.equals(crosswalk_combinations)

print(f"Crosswalk and merged dataset have the same type and subtype
 ↪  combinations: {are_equal}")

if not are_equal:
    print("Differences:")

     ↪  print(merged_combinations[~merged_combinations.isin(crosswalk_combinations)].dropna()
```

```
Crosswalk and merged dataset have the same type and subtype combinations:
False
Differences:
Empty DataFrame
Columns: [type, subtype]
Index: []
```

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

1. ChatGPT's response

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')

# 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\clari\OneDrive\Documents\Python II\problem set
 ↪  6\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)

# 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            -87.92            1
672            41.96            -87.93            1
673            41.67            -87.66            1
674            42.00            -87.91            1
675            41.96            -87.80            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\clari\AppData\Local\Temp\ipykernel_23332\1671116365.py:15:
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

C:\Users\clari\AppData\Local\Temp\ipykernel_23332\1671116365.py:16:
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
```

We observe 647 unique long-lat bins. We are aggregating by Jam and Traffic.

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='red').encode(
    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=12,  # Axis label font size
    titleFontSize=14,  # Axis title font size
    titleFontWeight='bold',  # Bold axis titles
    labelPadding=10  # Space between axis labels and the axis
).configure_title(
    fontSize=16,  # Font size for chart title
    fontWeight='bold'  # Bold chart title
).configure_legend(
```

```
    titleFontSize=14,  # Font size for legend title
    labelFontSize=12,  # Font size for legend labels
    symbolSize=100  # Adjust size of the legend symbols
)

# Display the chart
top_10_chart
```

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

3.

a.

```
# Specify the directory and file path
directory = r"C:\Users\clari\OneDrive\Documents\Python II\problem set 6"
file_path = os.path.join(directory, "chicago-boundaries.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()

# Create the directory if it doesn't exist
os.makedirs(directory, exist_ok=True)

# 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\clari\OneDrive\Documents\Python II\problem set
6\chicago-boundaries.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 map
base = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).properties(
    width=600,
    height=400
).project(
    type='equirectangular',
)

# If the map shows up correctly, continue with the points layer
top_10_chart = alt.Chart(top_10).mark_circle(color='red').encode(
    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 Jam - Heavy Traffic Alerts',
    width=600,
    height=400
)

# Apply configurations to the combined chart (outside of alt.layer)
jam_chart = jam_chart.configure_view(
    strokeWidth=0  # Remove border around the chart
).configure_axis(
    grid=True,  # Add grid lines
    labelFontSize=12,
    titleFontSize=14,  # Axis title font size
    titleFontWeight='bold',  # Bold axis titles
    labelPadding=10  # Padding for axis labels
).configure_title(
    fontSize=16,  # Font size for chart title
    fontWeight='bold'  # Bold chart title
).configure_legend(
    titleFontSize=14,  # Font size for legend title
    labelFontSize=12,  # Font size for legend labels
    symbolSize=100  # Adjust size of the legend symbols
)
```

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

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

a. Total number of type-subtype combinations: 11

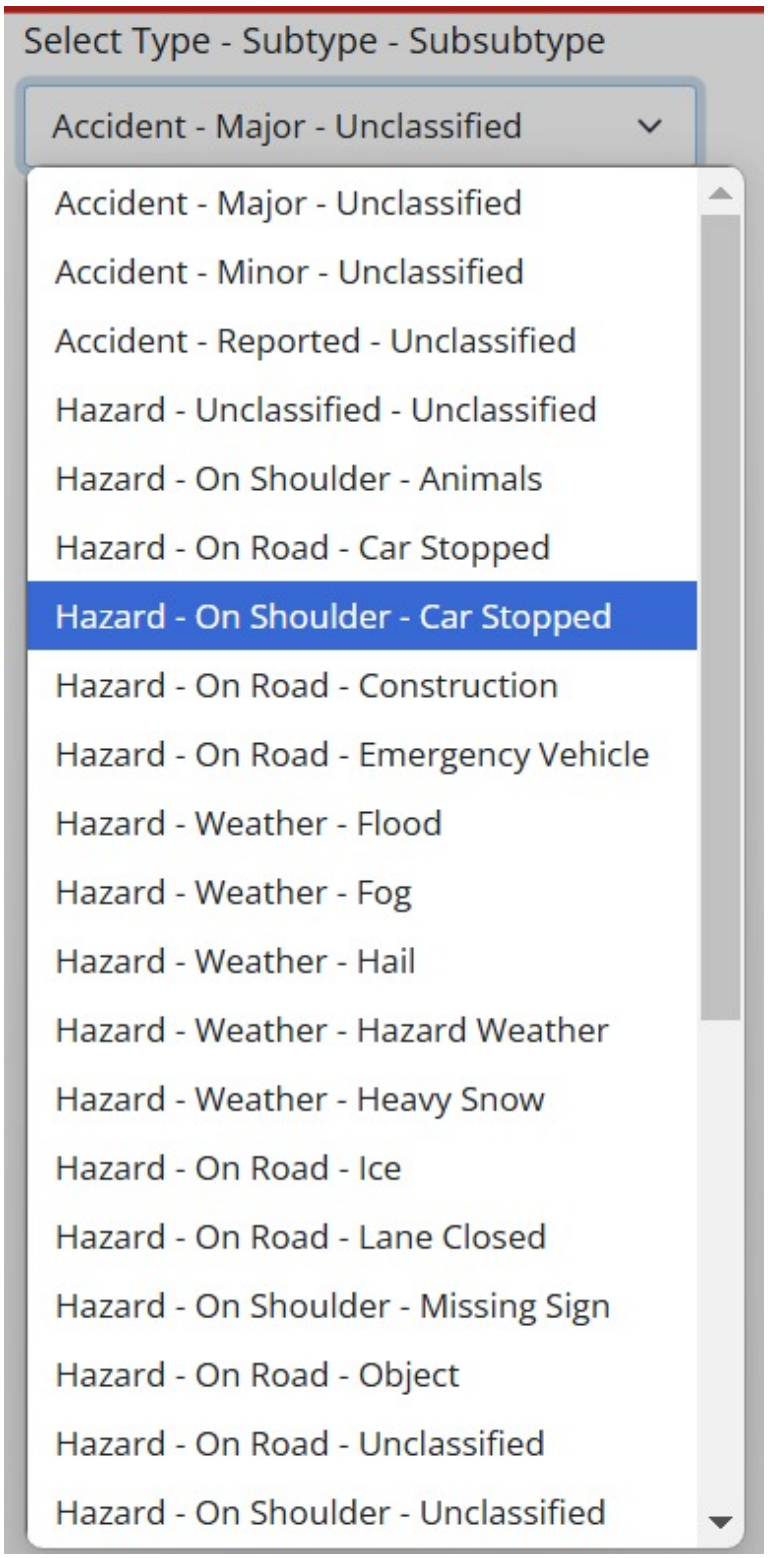


Figure 1: Dropdown menu

b.

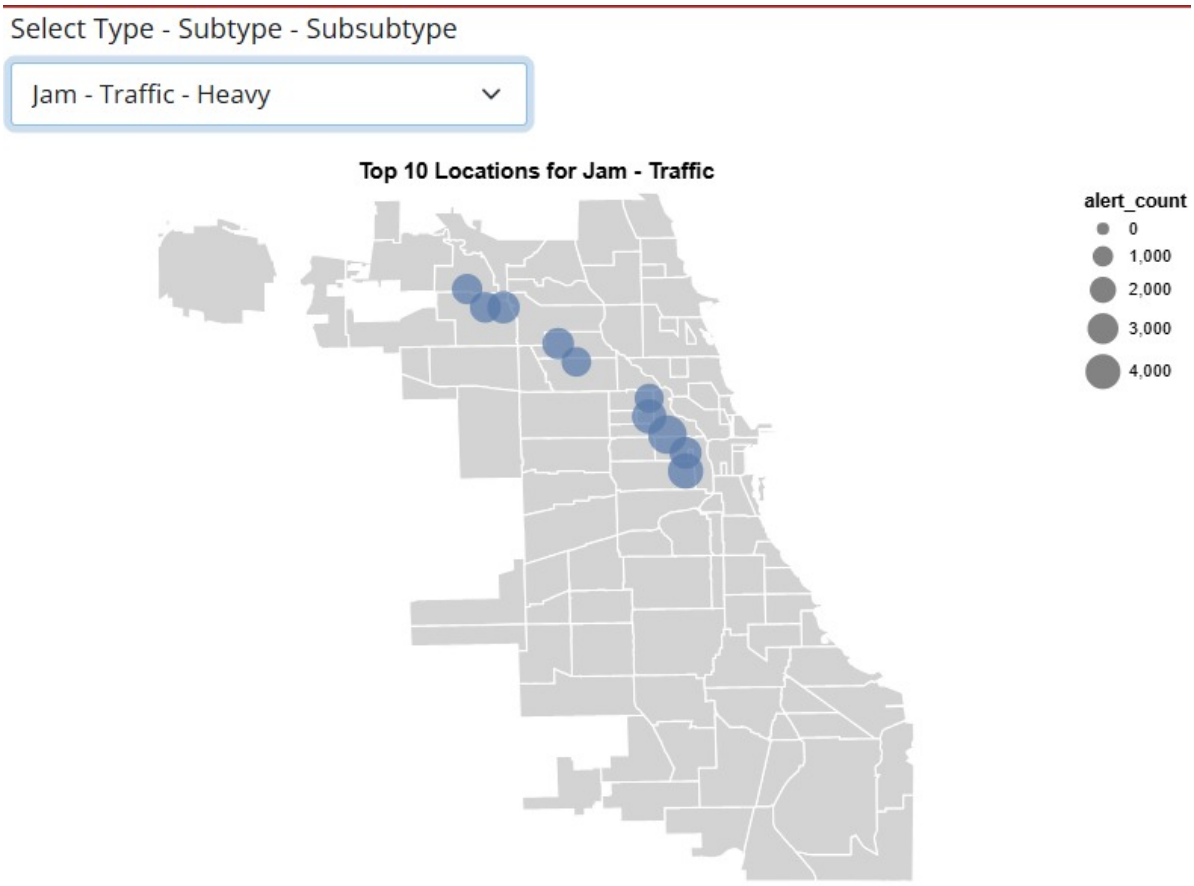


Figure 2: Jam- Heavy Traffic

c. Road closures due to an event seem to be most common at the North-West side of Chicago. Unfortunately, I don't have the long-lat because it won't load properly if I add it in, so I can't give the location.

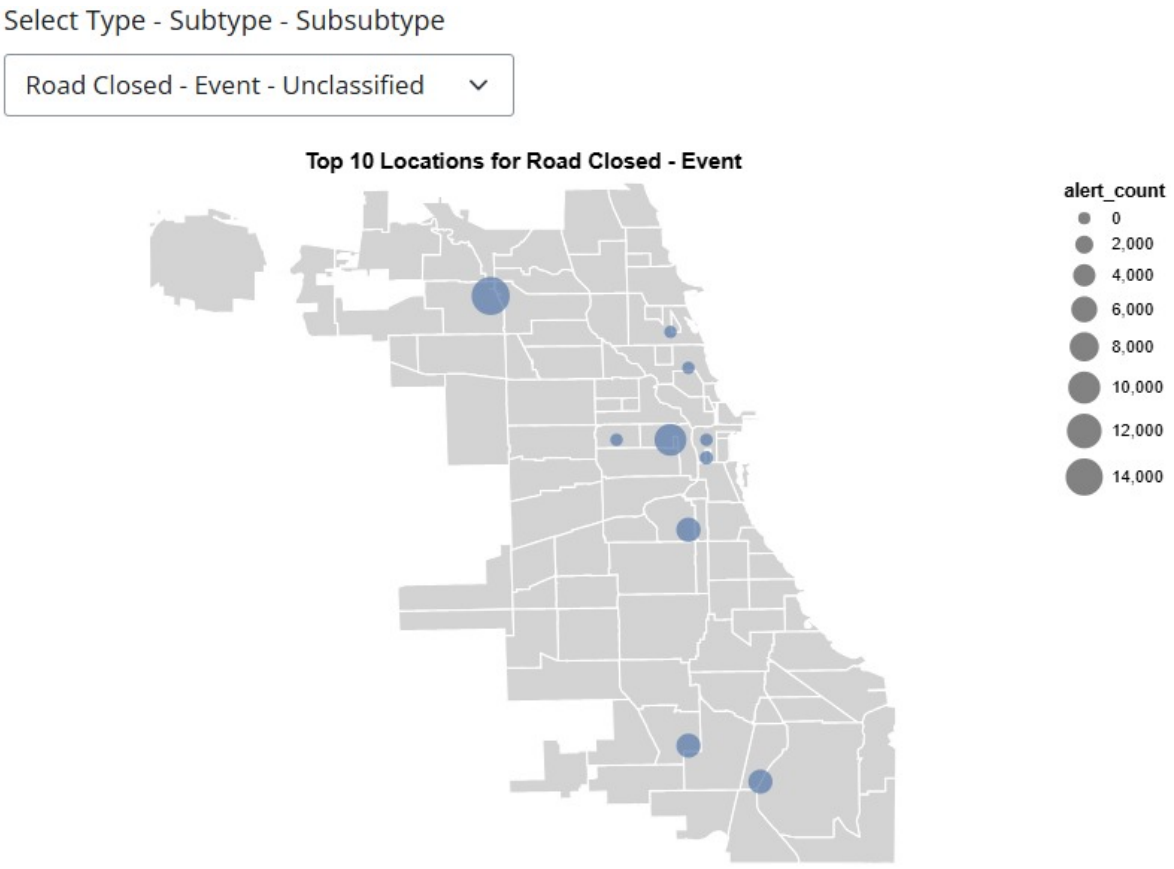


Figure 3: Road CLosure- Events

d.Where can we expect there to be the most number of major traffic accidents?

We can expect most of the major traffic accidents to be around the upper- middle partof Chicago. Unfortunately, I don't have the long-lat because it won't load properly if I add it in, so I can't give the location.

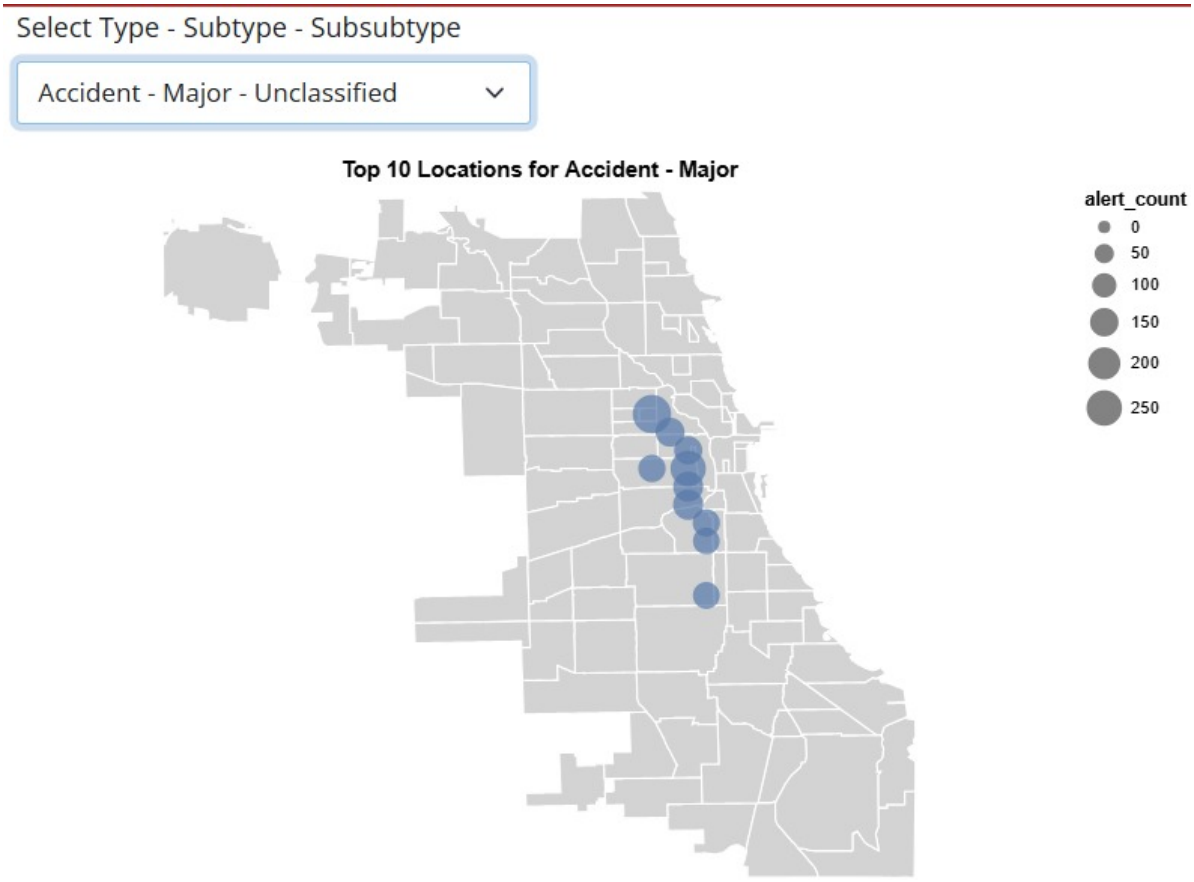


Figure 4: Major Traffic Accidents

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

The dashboard could benefit from more information about the time of day in which these accidents occur, along with more specific information as to where they occur. Currently, our dashboard displays this data in coordinates relative to a city map, which is helpful if you're already familiar with the area. For an outsider, more details about specific streets and highways would be quite helpful.

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

1.

a. Yes, I think it's a good idea, especially if we aim to analyze alerts by hour of the day (as suggested in the query), collapsing the data by ts into hourly bins makes sense. This

would allow you to identify patterns or trends in alerts based on the time of day. Reducing Data Size: If the dataset is very large, collapsing by ts can reduce its size, especially if you're aggregating alerts into hourly or daily bins. Time-Based Analysis: Collapsing by ts enables temporal analysis, such as understanding peak traffic hours, accident-prone times, or hazard trends.

b.

```
merged_df_path = r'C:\Users\clari\OneDrive\Documents\Python II\problem set
 ↪  6\merged_df.csv'
merged_df = pd.read_csv(merged_df_path)

# Check if 'geo' column exists
if 'geo' in merged_df.columns:
    # Extract latitude and longitude from the 'geo' column using regex
    pattern = r'POINT\((-?\d+\.\d+)\s(-?\d+\.\d+)\)'
    merged_df[['longitude', 'latitude']] = merged_df['geo'].str.extract(
        pattern).astype(float)
else:
    raise ValueError("The dataset does not contain a 'geo' column.")
```

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

# Check how many rows the dataset has
row_count = merged_df.shape[0]
print(f"The dataset contains {row_count} rows.")
```

```
The dataset contains 778094 rows.
```

```
if merged_df['ts'].dtype != 'datetime64[ns]':
    # Convert 'ts' to datetime format
    merged_df['ts'] = pd.to_datetime(
        merged_df['ts'].str.replace("UTC", ""), errors='coerce')

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

```
C:\Users\clari\AppData\Local\Temp\ipykernel_23332\1751780735.py:7:
FutureWarning:

'H' is deprecated and will be removed in a future version, please use 'h'
instead.
```

```
# Group and aggregate by hour, binned latitude, and longitude
aggregated_data = (
    merged_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\clari\OneDrive\Documents\Python II\problem set
 ↪  6\top_alerts_map_byhour'
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
```

```
data_path = r'C:\Users\clari\OneDrive\Documents\Python II\problem set
 ↪  6\top_alerts_map_byhour\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
# Modify this list to the specific hours you want
specific_hours = ['08:00', '12:00', '18:00']

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

# Load GeoJSON data for Chicago boundaries
geojson_path = r'C:\Users\clari\OneDrive\Documents\Python II\problem set
 ↪  6\chicago-boundaries.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)
    base_map = alt.Chart(geo_data).mark_geoshape(
        fill='lightgray',
        stroke='white'
    ).properties(
        width=600,
        height=400
    )
```

```
    # Add points layer for top locations
    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
    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}"
    )

    # Append chart to list of charts
    jam_hour_charts.append((jam_hour_chart, hour))
```

```
Selected specific hours: ['08:00', '12:00', '18:00']

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
36281 2024-06-09 08:00:00            41.96            -87.75           13
38431 2024-06-18 08:00:00            41.96            -87.75           13
31015 2024-05-18 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
33404 2024-05-28 08:00:00            41.96            -87.75           12
29819 2024-05-13 08:00:00            41.96            -87.75           12
32929 2024-05-26 08:00:00            41.96            -87.75           12

Data for hour 12:00:
                     hour  binned_latitude  binned_longitude  alert_count
52675 2024-08-17 12:00:00            41.88            -87.68           53
52672 2024-08-17 12:00:00            41.87            -87.68           38
28661 2024-05-08 12:00:00            41.96            -87.75           29
32251 2024-05-23 12:00:00            41.96            -87.75           23
```

```
34879 2024-06-03 12:00:00            41.96            -87.75           21
33920 2024-05-30 12:00:00            41.96            -87.75           20
31772 2024-05-21 12:00:00            41.96            -87.75           20
27223 2024-05-02 12:00:00            41.96            -87.75           20
40150 2024-06-25 12:00:00            41.96            -87.75           20
35362 2024-06-05 12:00:00            41.96            -87.75           19

Data for hour 18:00:
                     hour  binned_latitude  binned_longitude  alert_count
29199 2024-05-10 18:00:00            41.96            -87.75           21
30872 2024-05-17 18:00:00            41.88            -87.65           20
35898 2024-06-07 18:00:00            41.88            -87.65           19
30637 2024-05-16 18:00:00            41.88            -87.65           19
29438 2024-05-11 18:00:00            41.96            -87.75           19
32068 2024-05-22 18:00:00            41.88            -87.65           18
37329 2024-06-13 18:00:00            41.88            -87.65           18
31595 2024-05-20 18:00:00            41.96            -87.75           18
28958 2024-05-09 18:00:00            41.96            -87.75           17
37811 2024-06-15 18:00:00            41.96            -87.75           17
```

```
# Save each chart as an HTML file if needed
output_folder = r'C:\Users\clari\OneDrive\Documents\Python II\problem set 6'
os.makedirs(output_folder, exist_ok=True)

for i, (chart, hour) in enumerate(jam_hour_charts):
    output_path = os.path.join(
        output_folder, f'jam_hour_chart_{hour.replace(":", "-")}.html')
    chart.save(output_path)
    print(f"Chart saved to: {output_path}")
```

```
Chart saved to: C:\Users\clari\OneDrive\Documents\Python II\problem set
6\jam_hour_chart_08-00.html
Chart saved to: C:\Users\clari\OneDrive\Documents\Python II\problem set
6\jam_hour_chart_12-00.html
Chart saved to: C:\Users\clari\OneDrive\Documents\Python II\problem set
6\jam_hour_chart_18-00.html
```

```
jam_hour_chart
```

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

2.

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


# Printing the contents of your app2.py file
print_file_contents(
    r"C:\Users\clari\OneDrive\Documents\Python II\problem set 6\app2.py")
```

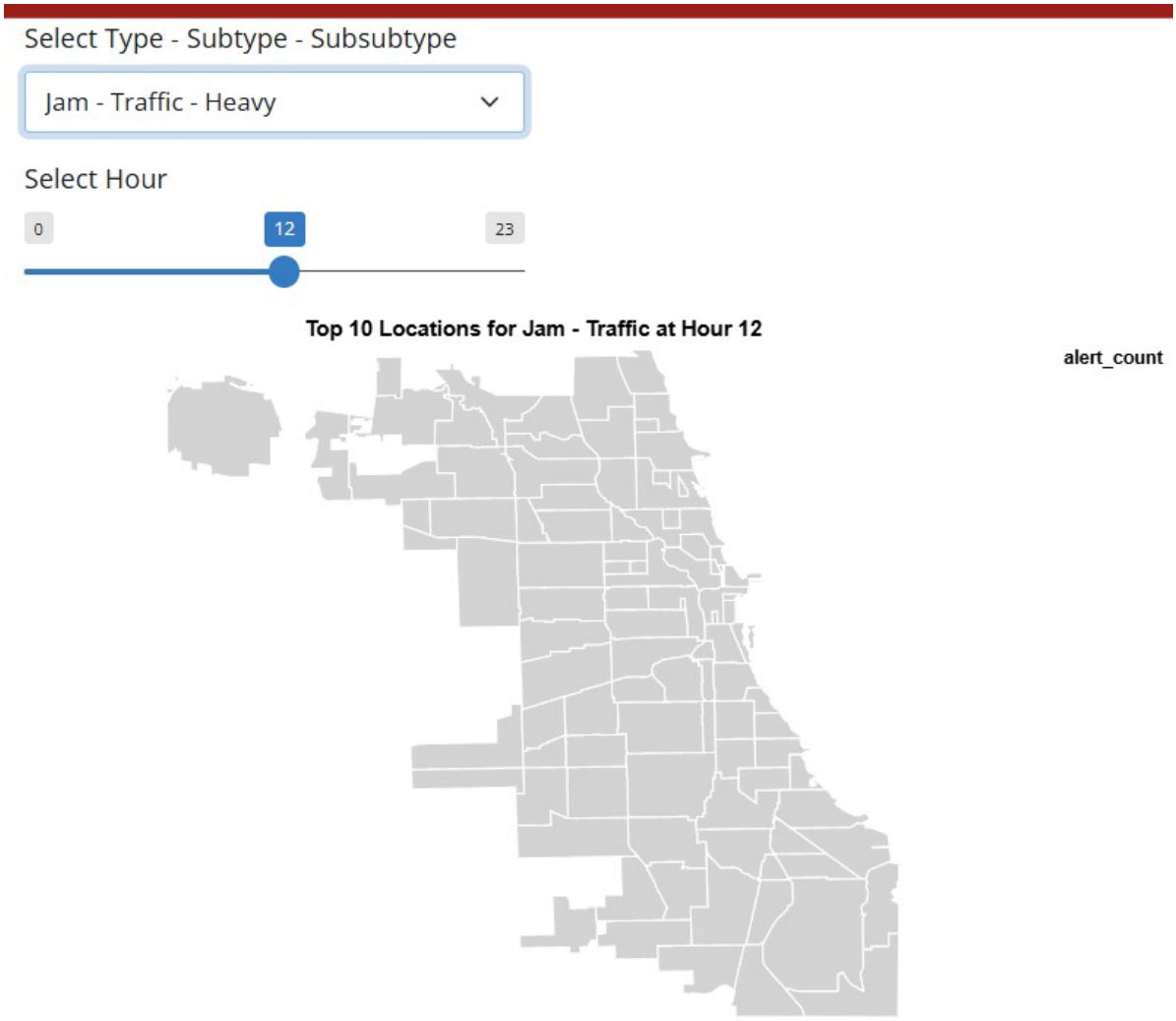


Figure 5: App2 Type-Subtype

a.

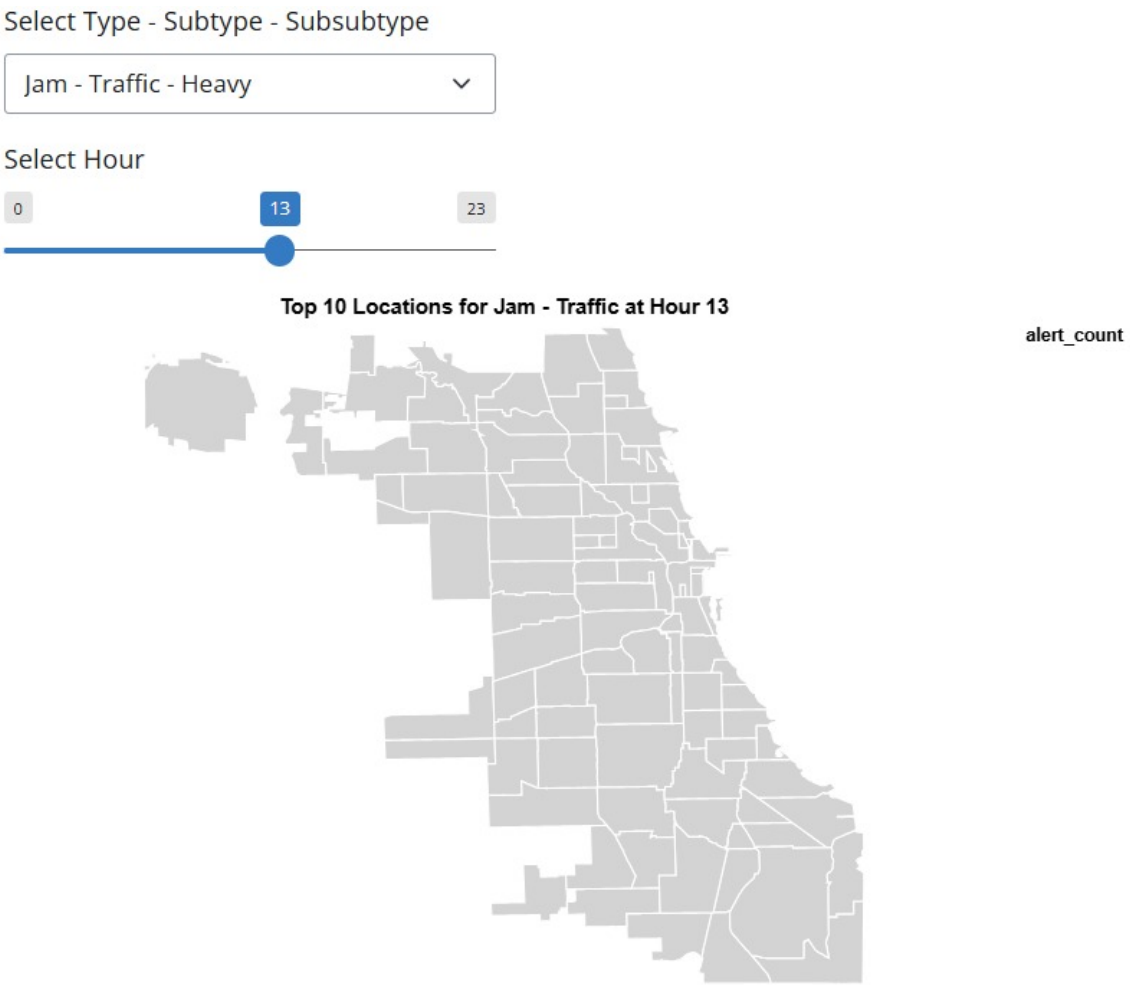


Figure 6: App2 Type-Subtype

b.

c. Unfortunately, although my app was working successfully before, after returning to it the following day, I couldn't recreate the points on the graph despite using the exact same code. This means I'm unable to provide screenshots (i.e. regardless of what time I choose, the map is empty), though based on when the app was running, I was able to gather than closures to to construction are much more common at night than they are during the morning. This makes sense, as late into the night is when they would be ideally be the least disruptive to commuters.

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

1.

a. Though collapsing the data was useful earlier and easier for shiny to handle, it presents a problem if we're allowing users to select their own range of hours. If we collapse the data to pre-arrange it into certain time ranges, we remove user's ability to choose the windows of time for themselves. By not collapsing, we allow the data to dynamically adjust to whatever hours specified.

b.

```
data_path = r'C:\Users\clari\OneDrive\Documents\Python II\problem set
 ↪  6\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)
# Modify this list to the specific hours you want
specific_hours = ['06:00', '07:00', '08:00', '09:00']

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

# Load GeoJSON data for Chicago boundaries
geojson_path = r'C:\Users\clari\OneDrive\Documents\Python II\problem set
 ↪  6\chicago-boundaries.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\clari\OneDrive\Documents\Python II\problem set 6'
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')
    # Using kaleido to save as PNG
    chart.save(output_path, renderer='kaleido', scale=2.0)
    print(f"Chart saved to: {output_path}")
```

```
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
28124 2024-05-06 06:00:00            41.96            -87.75           13
28364 2024-05-07 06:00:00            41.96            -87.75           13
30757 2024-05-17 06:00:00            41.88            -87.65           12
37691 2024-06-15 06:00:00            41.96            -87.75           12
30278 2024-05-15 06:00:00            41.96            -87.75           12
38411 2024-06-18 06:00:00            41.96            -87.75           12
39851 2024-06-24 06:00:00            41.96            -87.75           12
30038 2024-05-14 06:00:00            41.96            -87.75           12
27644 2024-05-04 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
36272 2024-06-09 07:00:00            41.96            -87.75           13
34352 2024-06-01 07:00:00            41.96            -87.75           13
34111 2024-05-31 07:00:00            41.96            -87.75           13
35312 2024-06-05 07:00:00            41.96            -87.75           12
36752 2024-06-11 07:00:00            41.96            -87.75           12
```

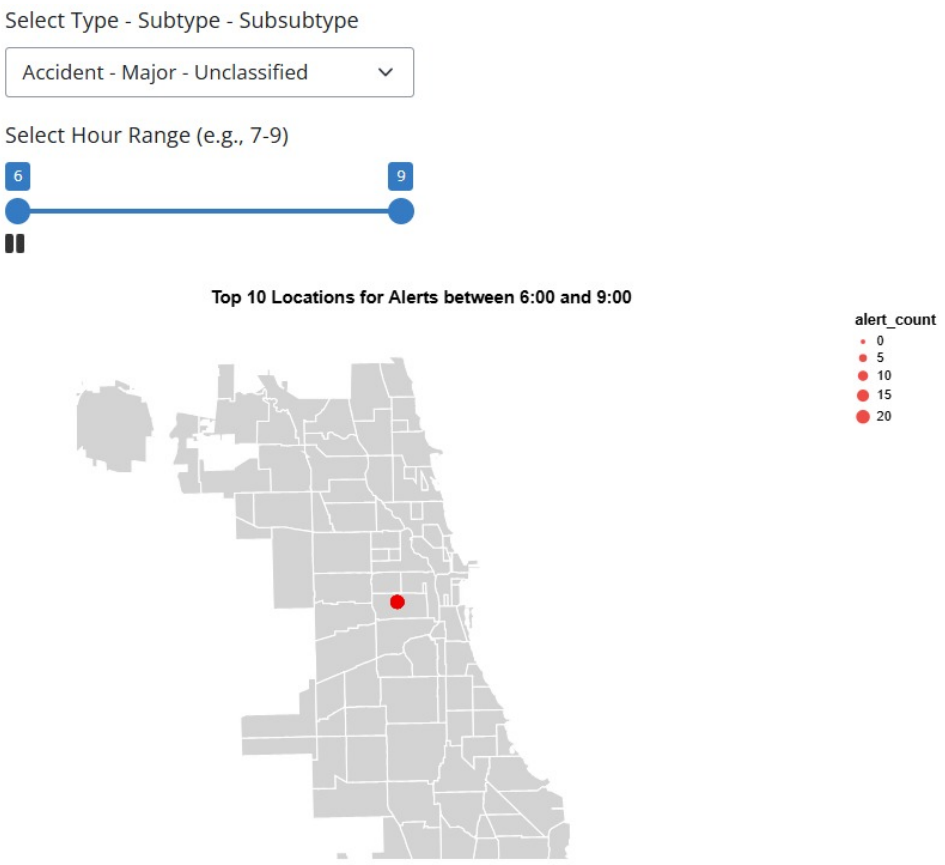
```
Data for hour 08:00:
                     hour  binned_latitude  binned_longitude  alert_count
45090 2024-07-16 08:00:00            41.87            -87.67           19
32449 2024-05-24 08:00:00            41.96            -87.75           14
36281 2024-06-09 08:00:00            41.96            -87.75           13
38431 2024-06-18 08:00:00            41.96            -87.75           13
31015 2024-05-18 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
33404 2024-05-28 08:00:00            41.96            -87.75           12
29819 2024-05-13 08:00:00            41.96            -87.75           12
32929 2024-05-26 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
34852 2024-06-03 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
38200 2024-06-17 09:00:00            41.96            -87.75           13
32223 2024-05-23 09:00:00            41.96            -87.75           13
30069 2024-05-14 09:00:00            41.96            -87.75           13
33179 2024-05-27 09:00:00            41.96            -87.75           13
35572 2024-06-06 09:00:00            41.96            -87.75           13
Chart saved to: C:\Users\clari\OneDrive\Documents\Python II\problem set
6\top_alerts_map_byhour_sliderrange_06-00.png
Chart saved to: C:\Users\clari\OneDrive\Documents\Python II\problem set
6\top_alerts_map_byhour_sliderrange_07-00.png
Chart saved to: C:\Users\clari\OneDrive\Documents\Python II\problem set
6\top_alerts_map_byhour_sliderrange_08-00.png
Chart saved to: C:\Users\clari\OneDrive\Documents\Python II\problem set
6\top_alerts_map_byhour_sliderrange_09-00.png
```

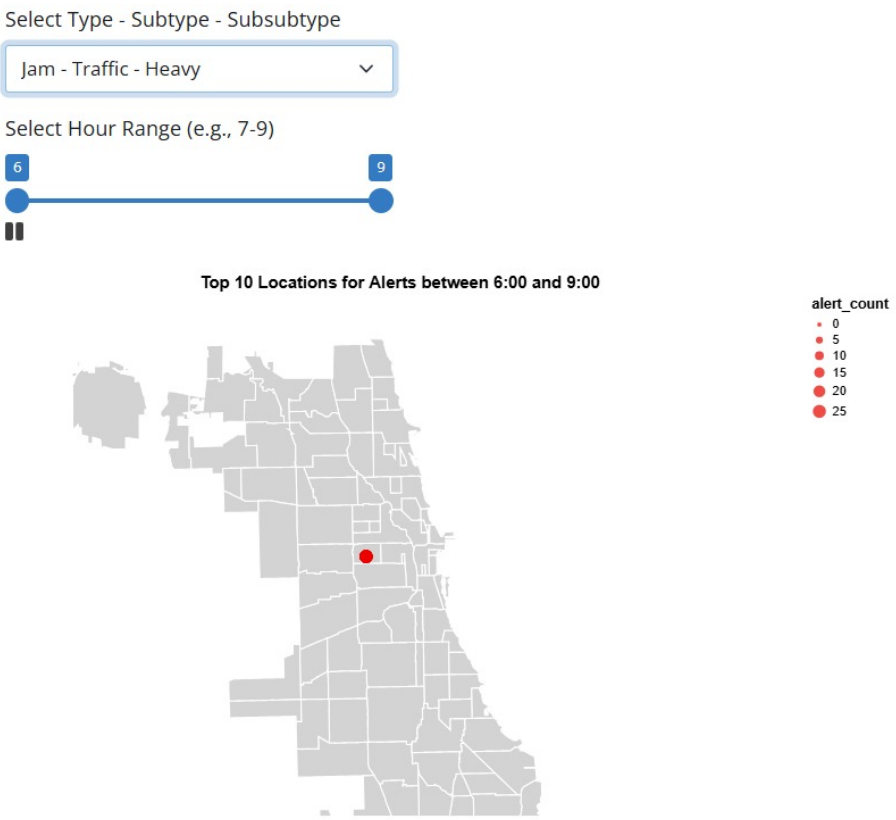
```
jam_hour_chart
```

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

2.


Select Type - Subtype - Subsubtype
Accident - Major - Unclassified
Select Hour Range (e.g., 7-9)
6
9
Top 10 Locations for Alerts between 6:00 and 9:00
alert_count
0
5
10
15
20

a. See image



b. See image

3.

a. According to the link, when using a switch button, we can assign a value of True, indicating the input.switch_button is turned on (switching to a range of hours, for example), or False, when the switch is turned off (switching to just one specific hour).

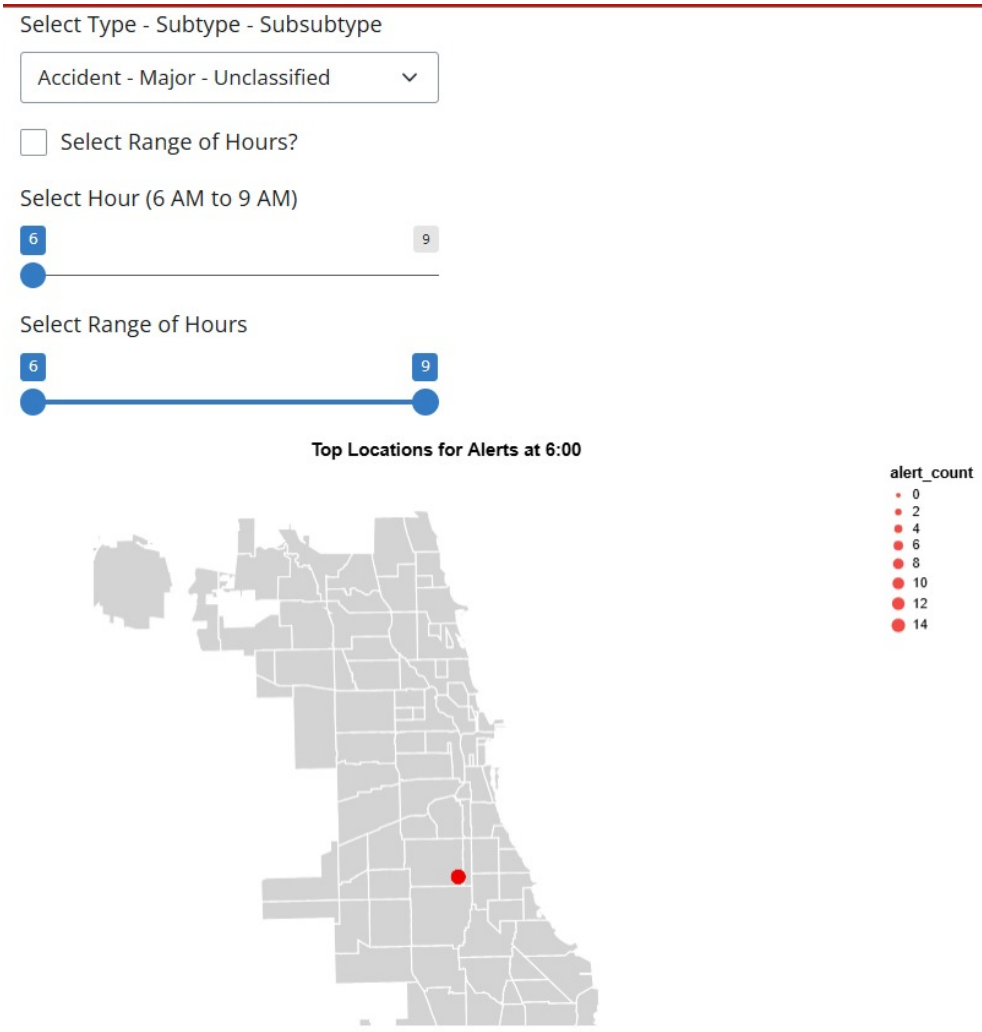
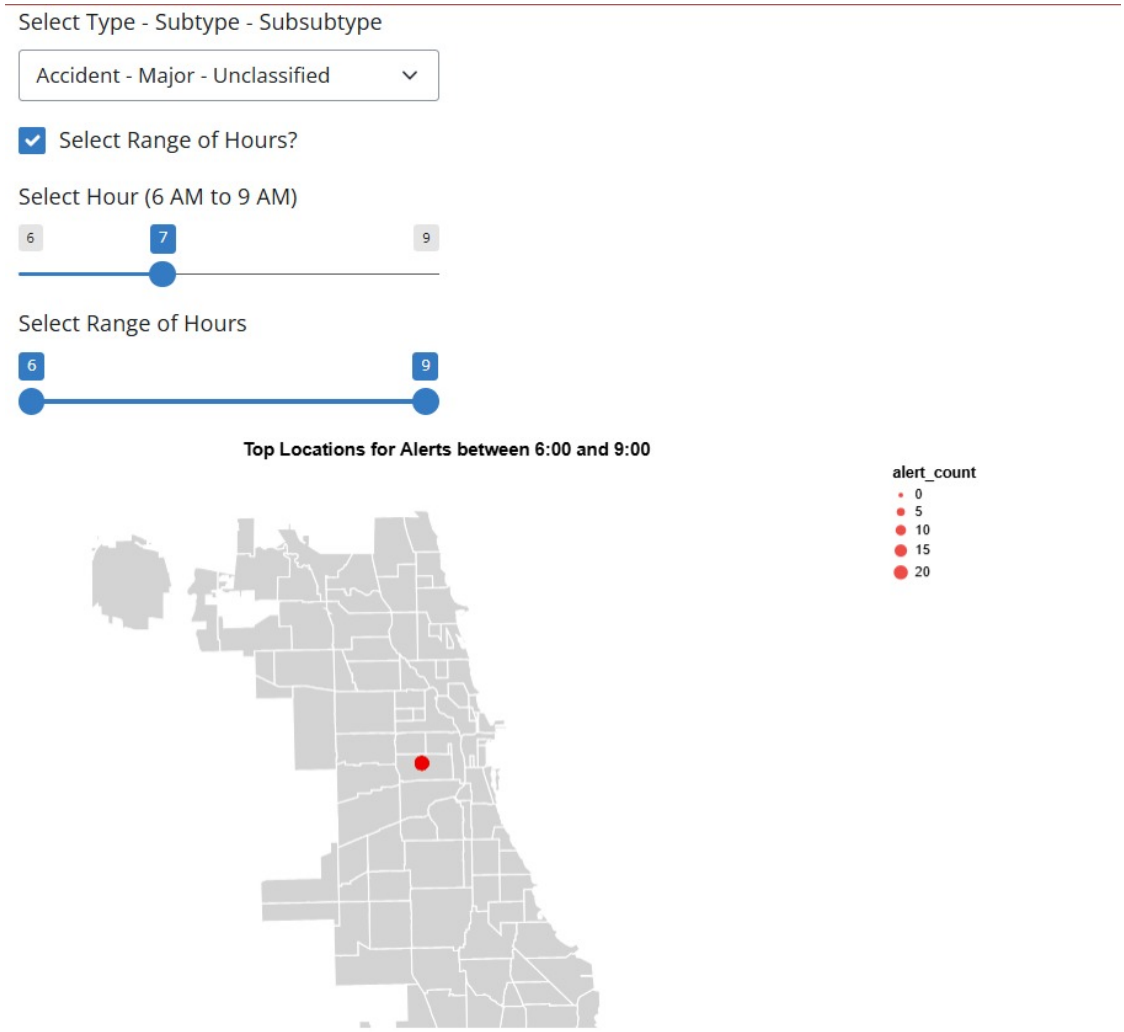


Figure 7: App3-Range


Select Type - Subtype - Subsubtype
Accident - Major - Unclassified
Select Range of Hours?
Select Hour (6 AM to 9 AM)
6
7
9
Select Range of Hours
6
9
Top Locations for Alerts between 6:00 and 9:00
alert_count
0
5
10
15
20

b. See images

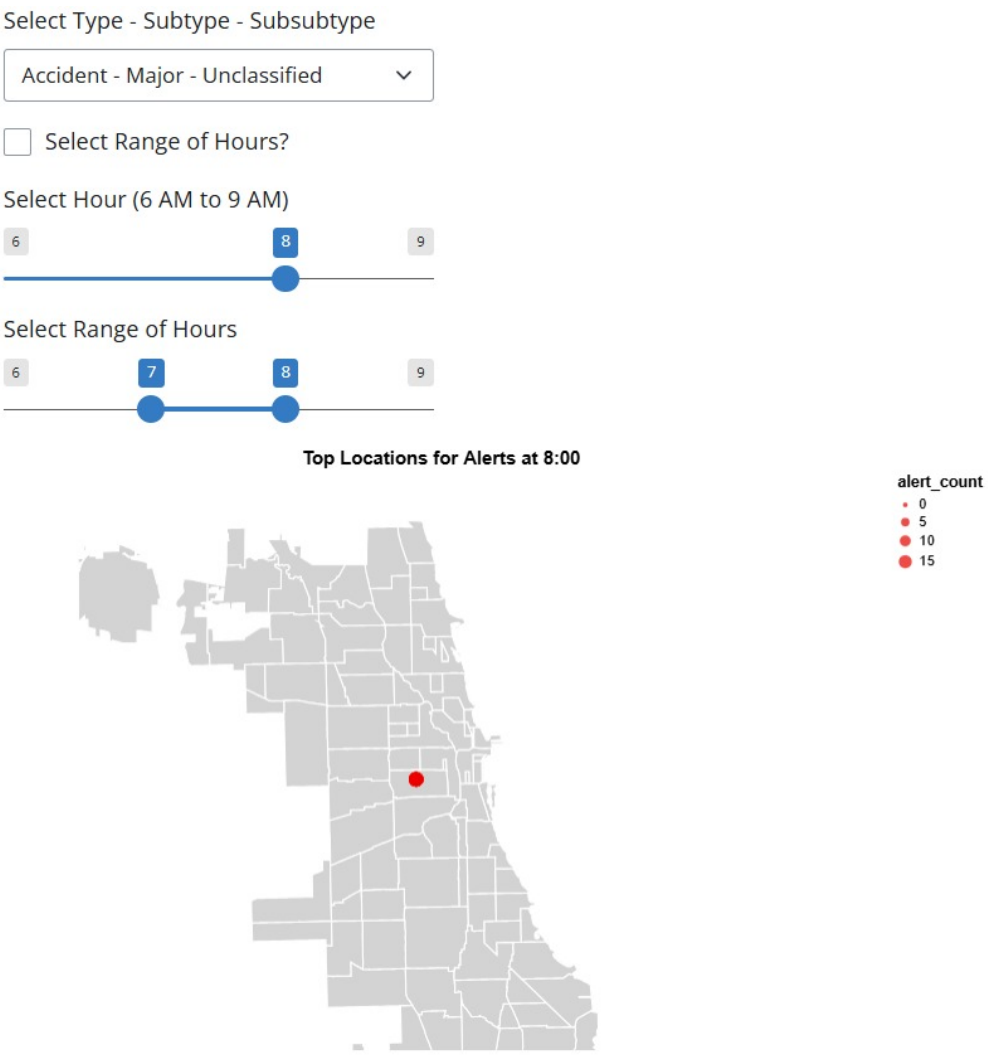


Figure 8: App3-Switch

c. See images (use the variations)

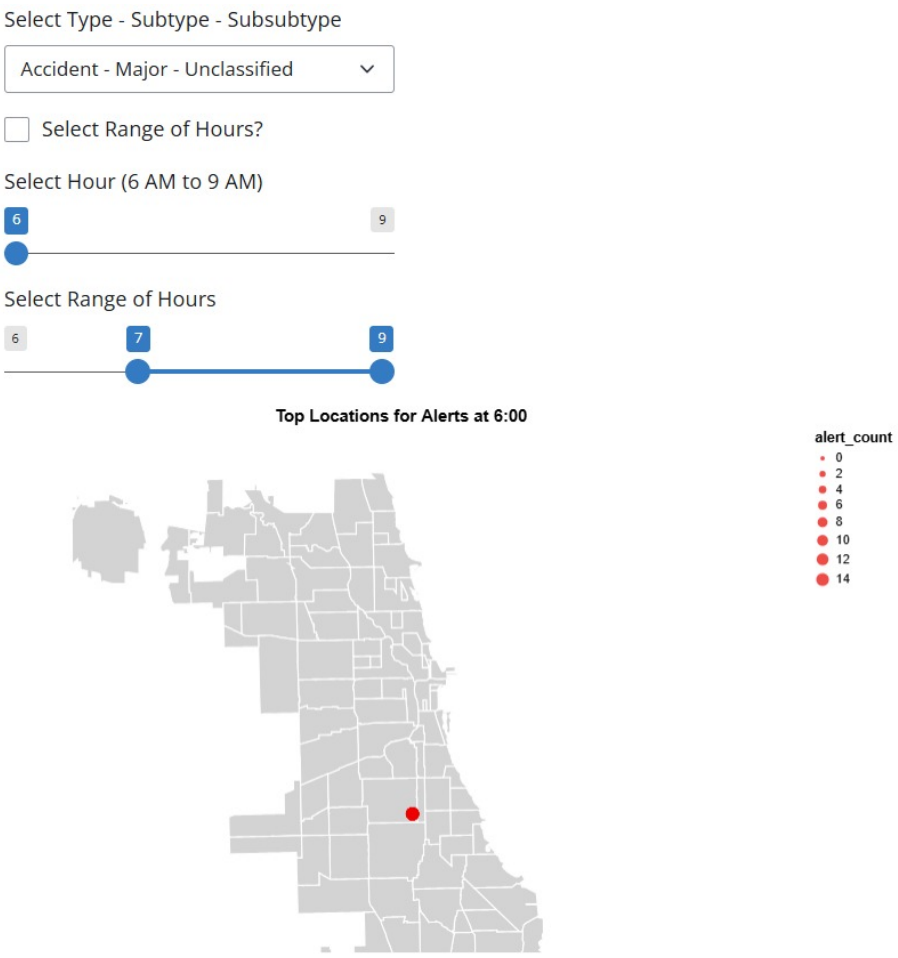


Figure 9: App3-Function

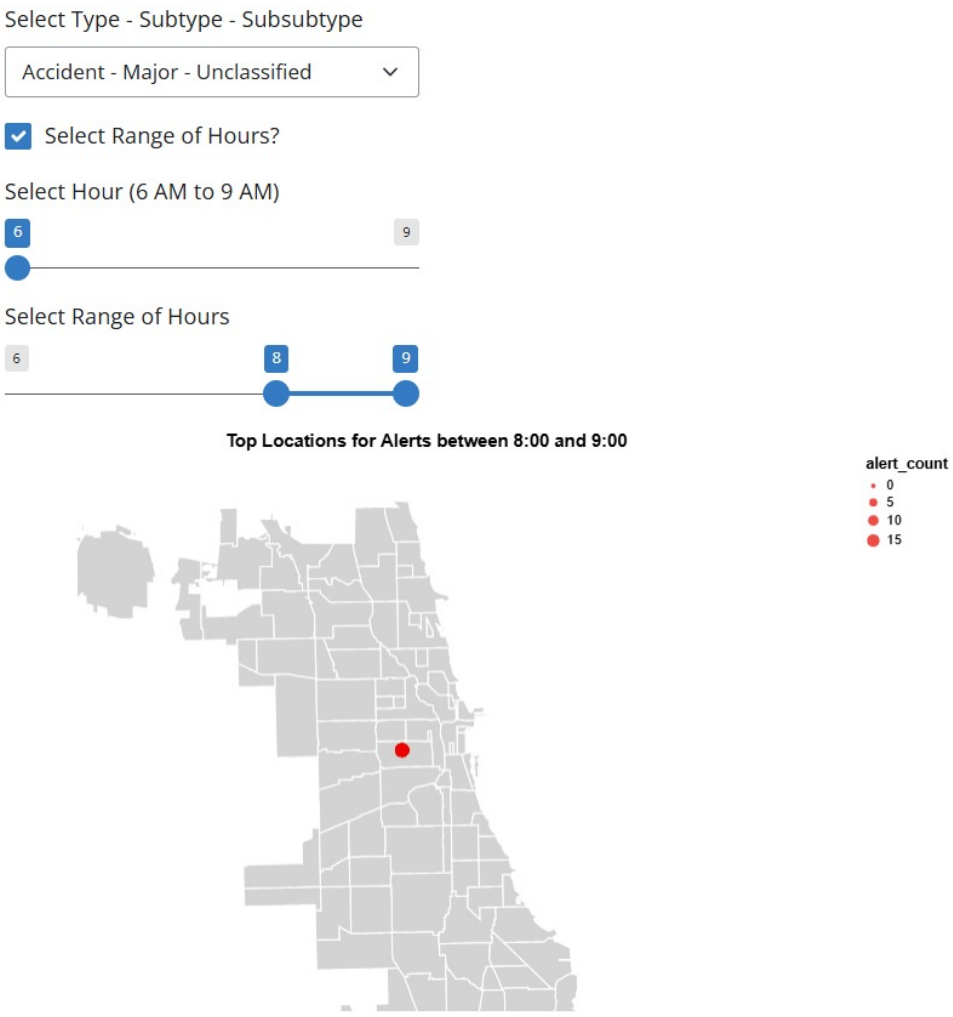


Figure 10: App3-Function

d. The plot seems to divide the data between morning and afternoon times and displays them both simultaneously. To do this, I'd assume our first step is to subset the data between the points that are in the morning (let's say all A.M. times) and another subset for the afternoon (we'll go with all P.M. times until midnight). We'd label the a.m. subset with an extra column indicating 'morning' and a similar 'afternoon' column. In the same way we had a button that toggles between one specific hour and a range, we can imagine a similar button which turns the display of morning points on or off, and another for afternoon points. When only one is on, we can focus solely on these points. If they're both on, they'll display much like what we see in this example plot.