Problem Set 6 - Waze Shiny Dashboard

Peter Ganong, Maggie Shi, and Andre Oviedo

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

```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("```")

print_file_contents("./top_alerts_map_byhour/app.py") # Change accordingly
```

Background

Data Download and Exploration (20 points)

1.

```python
# Read the sample data
df_sample = pd.read_csv("/Users/kohanchen/Documents/Fall
↪  2024/student30538/problem_sets/ps6/waze_data/waze_data_sample.csv")

# Get column names and types, excluding specified columns
cols_to_skip = ['ts', 'geo', 'geoWKT']
col_info = [(col, str(df_sample[col].dtype)) for col in df_sample.columns if
↪  col not in cols_to_skip]

print("Column names and their Altair data types:")
for col, dtype in col_info:
    # Map pandas dtypes to Altair types
```

```
    if dtype.startswith('int') or dtype.startswith('float'):
        alt_type = "Quantitative"
    elif dtype == 'object' or dtype == 'category':
        alt_type = "Nominal"
    elif dtype.startswith('datetime'):
        alt_type = "Temporal"
    else:
        alt_type = "Unknown"

    print(f"{col}: {alt_type}")
```

```
Column names and their Altair data types:
Unnamed: 0: Quantitative
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.

```
df_full = pd.read_csv("/Users/kohanchen/Documents/Fall
 ↪  2024/student30538/problem_sets/ps6/waze_data/waze_data.csv")

null_counts = pd.DataFrame({
    'Variable': df_full.columns,
    'Missing': df_full.isnull().sum(),
    'Not Missing': df_full.notnull().sum()
}).melt(id_vars=['Variable'], var_name='Status', value_name='Count')

# Create stacked bar chart
chart = alt.Chart(null_counts).mark_bar().encode(
    x=alt.X('Variable:N', title='Variables'),
    y=alt.Y('Count:Q',
            title='Number of Observations',
            stack='normalize'),  # stack parameter goes inside Y encoding
    color=alt.Color('Status:N', scale=alt.Scale(scheme='set2')),
    tooltip=[
        alt.Tooltip('Variable:N'),
        alt.Tooltip('Status:N'),
        alt.Tooltip('Count:Q', format=','),
        alt.Tooltip('Count:Q', aggregate='sum', title='Total Rows',
 ↪  format=',')
    ]
).properties(
    title='Distribution of Missing vs Non-Missing Values by Variable'
)
```

nThumbsUp, Street, and subtypes have missing values. And nThumbsUp has the highest share of missing values. 3.

```
print("Unique values in 'type':")
print(df_full['type'].unique())

print("All unique values in 'subtype':")
# Filter out NaN values before sorting
unique_subtypes = df_full['subtype'].dropna().unique()
print(sorted(unique_subtypes))

# Create crosswalk DataFrames
type_crosswalk = pd.DataFrame({
    'original': ['ROAD_CLOSED', 'JAM', 'ACCIDENT', 'HAZARD'],
    'clean_name': ['Road Closure', 'Traffic Jam', 'Accident', 'Hazard'] })

subtype_crosswalk = pd.DataFrame({
    'original': [
        # Accidents
        'ACCIDENT_MAJOR',
        'ACCIDENT_MINOR',

        # Jams
        'JAM_HEAVY_TRAFFIC',
        'JAM_LIGHT_TRAFFIC',
        'JAM_MODERATE_TRAFFIC',
        'JAM_STAND_STILL_TRAFFIC',
        # Road Closures
        'ROAD_CLOSED_CONSTRUCTION',
        'ROAD_CLOSED_EVENT',
        'ROAD_CLOSED_HAZARD',

        # Road Hazards
        'HAZARD_ON_ROAD',
        'HAZARD_ON_ROAD_CAR_STOPPED',
        'HAZARD_ON_ROAD_CONSTRUCTION',
        'HAZARD_ON_ROAD_EMERGENCY_VEHICLE',
        'HAZARD_ON_ROAD_ICE',
        'HAZARD_ON_ROAD_LANE_CLOSED',
        'HAZARD_ON_ROAD_OBJECT',
        'HAZARD_ON_ROAD_POT_HOLE',
        'HAZARD_ON_ROAD_ROAD_KILL',
        'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT',

        # Shoulder Hazards
        'HAZARD_ON_SHOULDER',
        'HAZARD_ON_SHOULDER_ANIMALS',
        'HAZARD_ON_SHOULDER_CAR_STOPPED',
        'HAZARD_ON_SHOULDER_MISSING_SIGN',

                # Weather Hazards
        'HAZARD_WEATHER',
        'HAZARD_WEATHER_FLOOD',
        'HAZARD_WEATHER_FOG',
        'HAZARD_WEATHER_HAIL',
        'HAZARD_WEATHER_HEAVY_SNOW'
    ],
    'clean_name': [
        # Accidents
        'Major Accident',
        'Minor Accident',

        # Jams
        'Heavy Traffic',
        'Light Traffic',
        'Moderate Traffic',
        'Standstill Traffic',

        # Road Closures
        'Construction Closure',
        'Event Closure',
        'Hazard Closure',

        # Road Hazards
        'Road Hazard',
        'Stopped Vehicle on Road',
        'Construction on Road',
        'Emergency Vehicle',
        'Ice on Road',
        'Lane Closure',
        'Object on Road',
        'Pothole',
        'Road Kill',
        'Traffic Light Issue',

        # Shoulder Hazards
        'Shoulder Hazard',
        'Animals on Shoulder',
        'Stopped Vehicle on Shoulder',
        'Missing Sign',

        # Weather Hazards
        'Weather Hazard',
        'Flooding',
        'Fog',
        'Hail',
        'Heavy Snow'
    ]
})
# Create mapping dictionaries
type_mapping = dict(zip(type_crosswalk['original'],
 ↪  type_crosswalk['clean_name']))
subtype_mapping = dict(zip(subtype_crosswalk['original'],
 ↪  subtype_crosswalk['clean_name']))

# Apply mappings while preserving NaN values
df_full['type_clean'] = df_full['type'].map(type_mapping)
df_full['subtype_clean'] = df_full['subtype'].map(subtype_mapping)

# Verify the results
print("Sample of cleaned data (including some rows with non-null subtypes):")
print(df_full[df_full['subtype'].notna()][['type', 'type_clean', 'subtype',
 ↪ 'subtype_clean']].head())
```

```
Unique values in 'type':
['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']
All unique values in 'subtype':
['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_LANE_CLOSED', 'HAZARD_ON_ROAD_OBJECT',
'HAZARD_ON_ROAD_POT_HOLE', 'HAZARD_ON_ROAD_ROAD_KILL',
'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT', 'HAZARD_ON_SHOULDER',
'HAZARD_ON_SHOULDER_ANIMALS', 'HAZARD_ON_SHOULDER_CAR_STOPPED',
'HAZARD_ON_SHOULDER_MISSING_SIGN', 'HAZARD_WEATHER', 'HAZARD_WEATHER_FLOOD',
'HAZARD_WEATHER_FOG', 'HAZARD_WEATHER_HAIL', 'HAZARD_WEATHER_HEAVY_SNOW',
'JAM_HEAVY_TRAFFIC', 'JAM_LIGHT_TRAFFIC', 'JAM_MODERATE_TRAFFIC',
'JAM_STAND_STILL_TRAFFIC', 'ROAD_CLOSED_CONSTRUCTION', 'ROAD_CLOSED_EVENT',
'ROAD_CLOSED_HAZARD']
Sample of cleaned data (including some rows with non-null subtypes):
         type type_clean         subtype   subtype_clean
122  ACCIDENT   Accident  ACCIDENT_MAJOR  Major Accident
123  ACCIDENT   Accident  ACCIDENT_MAJOR  Major Accident
124  ACCIDENT   Accident  ACCIDENT_MAJOR  Major Accident
125  ACCIDENT   Accident  ACCIDENT_MAJOR  Major Accident
126  ACCIDENT   Accident  ACCIDENT_MAJOR  Major Accident
```

```
# Print original values
print("Original unique values:")
print("\nTypes:")
print(sorted(df_full['type'].unique()))
print("\nSubtypes:")
print(sorted(df_full['subtype'].dropna().unique()))

# Print cleaned values
print("\nCleaned unique values:")
print("\nTypes:")
print(sorted(df_full['type_clean'].unique()))
print("\nSubtypes:")
print(sorted(df_full['subtype_clean'].dropna().unique()))
```

```
Original unique values:
```

```
Types:
['ACCIDENT', 'HAZARD', 'JAM', 'ROAD_CLOSED']

Subtypes:
['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_LANE_CLOSED', 'HAZARD_ON_ROAD_OBJECT',
'HAZARD_ON_ROAD_POT_HOLE', 'HAZARD_ON_ROAD_ROAD_KILL',
'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT', 'HAZARD_ON_SHOULDER',
'HAZARD_ON_SHOULDER_ANIMALS', 'HAZARD_ON_SHOULDER_CAR_STOPPED',
'HAZARD_ON_SHOULDER_MISSING_SIGN', 'HAZARD_WEATHER', 'HAZARD_WEATHER_FLOOD',
'HAZARD_WEATHER_FOG', 'HAZARD_WEATHER_HAIL', 'HAZARD_WEATHER_HEAVY_SNOW',
'JAM_HEAVY_TRAFFIC', 'JAM_LIGHT_TRAFFIC', 'JAM_MODERATE_TRAFFIC',
'JAM_STAND_STILL_TRAFFIC', 'ROAD_CLOSED_CONSTRUCTION', 'ROAD_CLOSED_EVENT',
'ROAD_CLOSED_HAZARD']

Cleaned unique values:

Types:
['Accident', 'Hazard', 'Road Closure', 'Traffic Jam']

Subtypes:
['Animals on Shoulder', 'Construction Closure', 'Construction on Road',
'Emergency Vehicle', 'Event Closure', 'Flooding', 'Fog', 'Hail', 'Hazard
Closure', 'Heavy Snow', 'Heavy Traffic', 'Ice on Road', 'Lane Closure',
'Light Traffic', 'Major Accident', 'Minor Accident', 'Missing Sign',
'Moderate Traffic', 'Object on Road', 'Pothole', 'Road Hazard', 'Road Kill',
'Shoulder Hazard', 'Standstill Traffic', 'Stopped Vehicle on Road', 'Stopped
Vehicle on Shoulder', 'Traffic Light Issue', 'Weather Hazard']
```

```python
# Count NAs by type
print("\nCount of NA subtypes by oringinal type:")
na_by_type = df_full[df_full['subtype'].isna()].groupby('type').size()
print(na_by_type)

# Look at subtype patterns to identify potential sub-subtypes
print("\nSubtype patterns (showing first few for each type):")
for type_val in sorted(df_full['type'].unique()):
    subtypes = df_full[df_full['type'] ==
 ↪  type_val]['subtype'].dropna().unique()
    print(f"\n{type_val}:")
    print(sorted(subtypes)[:5])
```

```
Count of NA subtypes by oringinal type:
type
ACCIDENT       24359
HAZARD          3212
JAM            55041
ROAD_CLOSED    13474
dtype: int64

Subtype patterns (showing first few for each type):

ACCIDENT:
['ACCIDENT_MAJOR', 'ACCIDENT_MINOR']

HAZARD:
['HAZARD_ON_ROAD', 'HAZARD_ON_ROAD_CAR_STOPPED',
'HAZARD_ON_ROAD_CONSTRUCTION', 'HAZARD_ON_ROAD_EMERGENCY_VEHICLE',
'HAZARD_ON_ROAD_ICE']

JAM:
['JAM_HEAVY_TRAFFIC', 'JAM_LIGHT_TRAFFIC', 'JAM_MODERATE_TRAFFIC',
'JAM_STAND_STILL_TRAFFIC']

ROAD_CLOSED:
['ROAD_CLOSED_CONSTRUCTION', 'ROAD_CLOSED_EVENT', 'ROAD_CLOSED_HAZARD']
```

Based on the output, HAZARD clearly shows a three-layer hierarchical structure.

Waze Incident Types Hierarchy

- Accident
    - Major
    - Minor
- Traffic Jam
    - Heavy
    - Light
    - Moderate
    - Standstill

- Road Closure
  - Construction
  - Event
  - Hazard
- Hazard
  - Road
    - General
    - Stopped Vehicle
    - Construction
    - Emergency Vehicle
    - Ice
    - Lane Closure
    - Object
    - Pothole
    - Road Kill
    - Traffic Light Issue
  - Shoulder
    - General
    - Stopped Vehicle
    - Animals
    - Missing Sign
  - Weather
    - General
    - Flooding
    - Fog
    - Hail
    - Heavy Snow

Yes we should keep the NA subtypes.

```
df_full['subtype_clean'] = df_full['subtype_clean'].fillna('Unclassified')

print("Distribution of subtypes including Unclassified:")
print(df_full.groupby(['type_clean',
 ↪  'subtype_clean']).size().reset_index(name='count'))
```

```
Distribution of subtypes including Unclassified:
      type_clean                subtype_clean   count
0       Accident               Major Accident    6669
1       Accident               Minor Accident    2509
```

```
2         Accident                 Unclassified   24359
3           Hazard          Animals on Shoulder     115
4           Hazard         Construction on Road   32094
5           Hazard            Emergency Vehicle    8360
6           Hazard                     Flooding    2844
7           Hazard                          Fog     697
8           Hazard                         Hail       7
9           Hazard                   Heavy Snow     138
10          Hazard                  Ice on Road     234
11          Hazard                 Lane Closure     541
12          Hazard                 Missing Sign      76
13          Hazard               Object on Road   16050
14          Hazard                      Pothole   28268
15          Hazard                  Road Hazard   34069
16          Hazard                    Road Kill      65
17          Hazard              Shoulder Hazard      40
18          Hazard      Stopped Vehicle on Road    5482
19          Hazard  Stopped Vehicle on Shoulder  176751
20          Hazard          Traffic Light Issue    4874
21          Hazard                 Unclassified    3212
22          Hazard               Weather Hazard    2146
23    Road Closure         Construction Closure     129
24    Road Closure                Event Closure   42393
25    Road Closure               Hazard Closure      13
26    Road Closure                 Unclassified   13474
27     Traffic Jam                Heavy Traffic  170442
28     Traffic Jam                Light Traffic       5
29     Traffic Jam             Moderate Traffic    4617
30     Traffic Jam           Standstill Traffic  142380
31     Traffic Jam                 Unclassified   55041
```

4.
5.

```
crosswalk = pd.DataFrame(columns=[
    'type',
    'subtype',
    'updated_type',
    'updated_subtype',
    'updated_subsubtype'
])
```

```
print("Crosswalk structure:")
print(crosswalk.columns.tolist())
```

```
Crosswalk structure:
['type', 'subtype', 'updated_type', 'updated_subtype', 'updated_subsubtype']
```

2.

```
# Create the crosswalk DataFrame
crosswalk = pd.DataFrame([
    # Accident
    {'type': 'ACCIDENT', 'subtype': 'ACCIDENT_MAJOR',
     'updated_type': 'Accident', 'updated_subtype': 'Major',
      ↪  'updated_subsubtype': None},
    {'type': 'ACCIDENT', 'subtype': 'ACCIDENT_MINOR',
     'updated_type': 'Accident', 'updated_subtype': 'Minor',
      ↪  'updated_subsubtype': None},

    # Traffic Jam
    {'type': 'JAM', 'subtype': 'JAM_HEAVY_TRAFFIC',
     'updated_type': 'Traffic Jam', 'updated_subtype': 'Heavy',
      ↪  'updated_subsubtype': None},
    {'type': 'JAM', 'subtype': 'JAM_LIGHT_TRAFFIC',
     'updated_type': 'Traffic Jam', 'updated_subtype': 'Light',
      ↪  'updated_subsubtype': None},
    {'type': 'JAM', 'subtype': 'JAM_MODERATE_TRAFFIC',
     'updated_type': 'Traffic Jam', 'updated_subtype': 'Moderate',
      ↪  'updated_subsubtype': None},
    {'type': 'JAM', 'subtype': 'JAM_STAND_STILL_TRAFFIC',
     'updated_type': 'Traffic Jam', 'updated_subtype': 'Standstill',
      ↪  'updated_subsubtype': None},

    # Road Closure
    {'type': 'ROAD_CLOSED', 'subtype': 'ROAD_CLOSED_CONSTRUCTION',
     'updated_type': 'Road Closure', 'updated_subtype': 'Construction',
      ↪  'updated_subsubtype': None},
    {'type': 'ROAD_CLOSED', 'subtype': 'ROAD_CLOSED_EVENT',
     'updated_type': 'Road Closure', 'updated_subtype': 'Event',
      ↪  'updated_subsubtype': None},
    {'type': 'ROAD_CLOSED', 'subtype': 'ROAD_CLOSED_HAZARD',
     'updated_type': 'Road Closure', 'updated_subtype': 'Hazard',
      ↪  'updated_subsubtype': None},
    # Hazard - Road
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD',
     'updated_type': 'Hazard', 'updated_subtype': 'Road',
      ↪  'updated_subsubtype': 'General'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_CAR_STOPPED',
     'updated_type': 'Hazard', 'updated_subtype': 'Road',
      ↪  'updated_subsubtype': 'Stopped Vehicle'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_CONSTRUCTION',
     'updated_type': 'Hazard', 'updated_subtype': 'Road',
      ↪  'updated_subsubtype': 'Construction'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_EMERGENCY_VEHICLE',
     'updated_type': 'Hazard', 'updated_subtype': 'Road',
      ↪  'updated_subsubtype': 'Emergency Vehicle'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_ICE',
     'updated_type': 'Hazard', 'updated_subtype': 'Road',
      ↪  'updated_subsubtype': 'Ice'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_LANE_CLOSED',
     'updated_type': 'Hazard', 'updated_subtype': 'Road',
      ↪  'updated_subsubtype': 'Lane Closure'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_OBJECT',
     'updated_type': 'Hazard', 'updated_subtype': 'Road',
      ↪  'updated_subsubtype': 'Object'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_POT_HOLE',
     'updated_type': 'Hazard', 'updated_subtype': 'Road',
      ↪  'updated_subsubtype': 'Pothole'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_ROAD_KILL',
     'updated_type': 'Hazard', 'updated_subtype': 'Road',
      ↪  'updated_subsubtype': 'Road Kill'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT',
     'updated_type': 'Hazard', 'updated_subtype': 'Road',
      ↪  'updated_subsubtype': 'Traffic Light Issue'},

    # Hazard - Shoulder
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_SHOULDER',
     'updated_type': 'Hazard', 'updated_subtype': 'Shoulder',
      ↪  'updated_subsubtype': 'General'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_SHOULDER_ANIMALS',
     'updated_type': 'Hazard', 'updated_subtype': 'Shoulder',
      ↪  'updated_subsubtype': 'Animals'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_SHOULDER_CAR_STOPPED',
     'updated_type': 'Hazard', 'updated_subtype': 'Shoulder',
      ↪  'updated_subsubtype': 'Stopped Vehicle'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_SHOULDER_MISSING_SIGN',
     'updated_type': 'Hazard', 'updated_subtype': 'Shoulder',
      ↪  'updated_subsubtype': 'Missing Sign'},

    # Hazard - Weather
    {'type': 'HAZARD', 'subtype': 'HAZARD_WEATHER',
     'updated_type': 'Hazard', 'updated_subtype': 'Weather',
      ↪  'updated_subsubtype': 'General'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_WEATHER_FLOOD',
     'updated_type': 'Hazard', 'updated_subtype': 'Weather',
      ↪  'updated_subsubtype': 'Flooding'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_WEATHER_FOG',
     'updated_type': 'Hazard', 'updated_subtype': 'Weather',
      ↪  'updated_subsubtype': 'Fog'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_WEATHER_HAIL',
     'updated_type': 'Hazard', 'updated_subtype': 'Weather',
      ↪  'updated_subsubtype': 'Hail'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_WEATHER_HEAVY_SNOW',
     'updated_type': 'Hazard', 'updated_subtype': 'Weather',
      ↪  'updated_subsubtype': 'Heavy Snow'}
])

# Add row for NA values
na_rows = [
    {'type': 'ACCIDENT', 'subtype': None,
     'updated_type': 'Accident', 'updated_subtype': 'Unclassified',
      ↪  'updated_subsubtype': None},
    {'type': 'JAM', 'subtype': None,
     'updated_type': 'Traffic Jam', 'updated_subtype': 'Unclassified',
      ↪  'updated_subsubtype': None},
    {'type': 'ROAD_CLOSED', 'subtype': None,
     'updated_type': 'Road Closure', 'updated_subtype': 'Unclassified',
      ↪  'updated_subsubtype': None},
    {'type': 'HAZARD', 'subtype': None,
     'updated_type': 'Hazard', 'updated_subtype': 'Unclassified',
      ↪  'updated_subsubtype': None}
]

crosswalk = pd.concat([crosswalk, pd.DataFrame(na_rows)], ignore_index=True)

print(f"\nTotal rows in crosswalk: {len(crosswalk)}")  #
```

```python
# Display the first few rows to verify
print(crosswalk.head(10))
```

```
Total rows in crosswalk: 32
          type                   subtype  updated_type updated_subtype  \
0     ACCIDENT            ACCIDENT_MAJOR      Accident           Major
1     ACCIDENT            ACCIDENT_MINOR      Accident           Minor
2          JAM         JAM_HEAVY_TRAFFIC   Traffic Jam           Heavy
3          JAM         JAM_LIGHT_TRAFFIC   Traffic Jam           Light
4          JAM      JAM_MODERATE_TRAFFIC   Traffic Jam        Moderate
5          JAM   JAM_STAND_STILL_TRAFFIC   Traffic Jam      Standstill
6  ROAD_CLOSED  ROAD_CLOSED_CONSTRUCTION  Road Closure    Construction
7  ROAD_CLOSED         ROAD_CLOSED_EVENT  Road Closure           Event
8  ROAD_CLOSED        ROAD_CLOSED_HAZARD  Road Closure          Hazard
9       HAZARD            HAZARD_ON_ROAD        Hazard            Road

  updated_subsubtype
0                None
1                None
2                None
3                None
4                None
5                None
6                None
7                None
8                None
9             General
```

3.

```python
df_merged = df_full.merge(crosswalk, on=['type', 'subtype'], how='left')

# Count Accident - Unclassified rows
accident_unclassified = df_merged[
    (df_merged['updated_type'] == 'Accident') &
    (df_merged['updated_subtype'] == 'Unclassified')
].shape[0]

print("Number of Accident - Unclassified rows:", accident_unclassified)
```

```
Number of Accident - Unclassified rows: 24359
```

4.

```
from pathlib import Path
# Get unique combinations from original merged data
merged_combos = df_merged[['type', 'subtype',
                          'updated_type', 'updated_subtype',
                          'updated_subsubtype']].drop_duplicates()

# Get unique combinations from crosswalk
crosswalk_combos = crosswalk[['type', 'subtype',
                             'updated_type', 'updated_subtype',
                             'updated_subsubtype']].drop_duplicates()

# Compare the number of unique combinations
print("\nNumber of unique combinations:")
print(f"Merged dataset: {len(merged_combos)}")
print(f"Crosswalk: {len(crosswalk_combos)}")

# Check if any combinations in merged data are not in crosswalk
missing_combos = merged_combos.merge(
    crosswalk_combos,
    on=['type', 'subtype', 'updated_type', 'updated_subtype',
 ↪  'updated_subsubtype'],
    how='outer',
    indicator=True
)

if len(missing_combos[missing_combos['_merge'] != 'both']) > 0:
    print("\nFound mismatched combinations:")
    print(missing_combos[missing_combos['_merge'] != 'both'])
else:
    print("\nAll combinations match between crosswalk and merged dataset.")

output_path = Path("./data/processed_waze_data.csv")
output_path.parent.mkdir(exist_ok=True)
df_merged.to_csv(output_path, index=False)
print(f"Saved processed dataset to {output_path}")
```

```

Number of unique combinations:
Merged dataset: 32
Crosswalk: 32
```

```
All combinations match between crosswalk and merged dataset.
Saved processed dataset to data/processed_waze_data.csv
```

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

1.

a.latitude_bin:41.88, longitude_bin:-87.65 is the most frequent location.

```
import pandas as pd
import re
from pathlib import Path
import numpy as np
import altair as alt

#a
df_merged = pd.read_csv("/Users/kohanchen/Documents/Fall
 ↪  2024/student30538/problem_sets/ps6/data/processed_waze_data.csv")
print(f"Loaded {len(df_merged)} rows of data")

# Extract coordinates from WKT format
def extract_coordinates(geo_string):
    pattern = r'POINT\(([-\d.]+) ([-\d.]+)\)'
    match = re.search(pattern, str(geo_string))
    if match:
        return float(match.group(2)), float(match.group(1))
    return None, None

# Create latitude and longitude columns
df_merged[['latitude', 'longitude']] = df_merged['geo'].apply(
    lambda x: pd.Series(extract_coordinates(x))
)

# Fill NaN values in updated_subsubtype with "None"
df_merged['updated_subsubtype'] =
 ↪  df_merged['updated_subsubtype'].fillna("None")

# Verify coordinate extraction
print("\nVerifying coordinate extraction:")
print(df_merged[['geo', 'latitude', 'longitude']].head())
```

```
# Verify alert type columns
print("\nVerifying alert type columns:")
print(df_merged[['updated_type', 'updated_subtype',
 ↪  'updated_subsubtype']].head())

# Check for any null values
print("\nChecking for null values in key columns:")
print(df_merged[['latitude', 'longitude', 'updated_type', 'updated_subtype',
 ↪  'updated_subsubtype']].isnull().sum())
```

```
Loaded 778094 rows of data

Verifying coordinate extraction:
                           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

Verifying alert type columns:
   updated_type updated_subtype updated_subsubtype
0   Traffic Jam    Unclassified               None
1      Accident    Unclassified               None
2  Road Closure    Unclassified               None
3   Traffic Jam    Unclassified               None
4   Traffic Jam    Unclassified               None

Checking for null values in key columns:
latitude              0
longitude             0
updated_type          0
updated_subtype       0
updated_subsubtype    0
dtype: int64
```

b.

```
#b
# Bin coordinates (round to 2 decimal places)
df_merged['latitude_bin'] = np.round(df_merged['latitude'], decimals=2)
```

```python
df_merged['longitude_bin'] = np.round(df_merged['longitude'], decimals=2)

# Group by binned coordinates to find most frequent location
location_counts = df_merged.groupby(['latitude_bin',
 ↪  'longitude_bin']).size().reset_index(name='count')
most_frequent = location_counts.sort_values('count', ascending=False).head()

print("\nTop 5 most frequent locations (binned):")
print(most_frequent.to_string(index=False))
```

```
Top 5 most frequent locations (binned):
 latitude_bin  longitude_bin  count
        41.88         -87.65  21325
        41.89         -87.65  19996
        41.96         -87.75  16309
        41.97         -87.75  14570
        41.90         -87.66  14197
```

Answer for b.latitude_bin:41.88, longitude_bin:-87.65 is the most frequent location.

c.

```python
#c.
def create_top_alerts_map_data(df):
    agg_df = df.groupby(
        ['latitude_bin', 'longitude_bin', 'updated_type', 'updated_subtype']
    ).size().reset_index(name='alert_count')

    # Save to CSV
    output_path = Path("/Users/kohanchen/Documents/Fall
 ↪  2024/student30538/problem_sets/ps6/top_alerts_map/top_alerts_map.csv")
    agg_df.to_csv(output_path, index=False)

    print("\nAggregation level: binned latitude-longitude + alert type +
     ↪  alert subtype")
    print(f"Number of rows in aggregated data: {len(agg_df)}")

    print("\nSample of aggregated data:")
    print(agg_df.head())

    return agg_df
```

```
agg_df = create_top_alerts_map_data(df_merged)
```

```
Aggregation level: binned latitude-longitude + alert type + alert subtype
Number of rows in aggregated data: 6675

Sample of aggregated data:
   latitude_bin  longitude_bin  updated_type updated_subtype  alert_count
0         41.64         -87.61        Hazard            Road            3
1         41.65         -87.62        Hazard            Road           16
2         41.65         -87.62  Road Closure    Unclassified            1
3         41.65         -87.62    Traffic Jam    Unclassified            1
4         41.65         -87.61        Hazard            Road           12
```

Aggregation level: binned latitude-longitude + alert type + alert subtype Number of rows in aggregated data: 6675

2.
3.

a.

```
import requests
from pathlib import Path

def download_chicago_boundaries():
    # URL for Chicago Neighborhoods GeoJSON
    url =
↪   "https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON"

    try:
        # Send GET request
        response = requests.get(url)
        response.raise_for_status()

        # Create directory if it doesn't exist
        output_path = Path("/Users/kohanchen/Documents/Fall
↪   2024/student30538/problem_sets/ps6/top_alerts_map/chicago-boundaries.geojson")
        output_path.parent.mkdir(exist_ok=True)

        # Save the file
        with open(output_path, "w") as f:
            f.write(response.text)

        print(f"Successfully downloaded Chicago boundaries to {output_path}")

    except requests.RequestException as e:
        print(f"Error downloading file: {e}")
        return None

# Download the boundaries
download_chicago_boundaries()
```

```
Successfully downloaded Chicago boundaries to /Users/kohanchen/Documents/Fall
2024/student30538/problem_sets/ps6/top_alerts_map/chicago-boundaries.geojson
```

b.

```
import json
import altair as alt

# MODIFY ACCORDINGLY
file_path = "/Users/kohanchen/Documents/Fall
↪   2024/student30538/problem_sets/ps6/top_alerts_map/chicago-boundaries.geojson"
#----

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

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

4.
5.

a. Total number of type-subtype combinations: 16

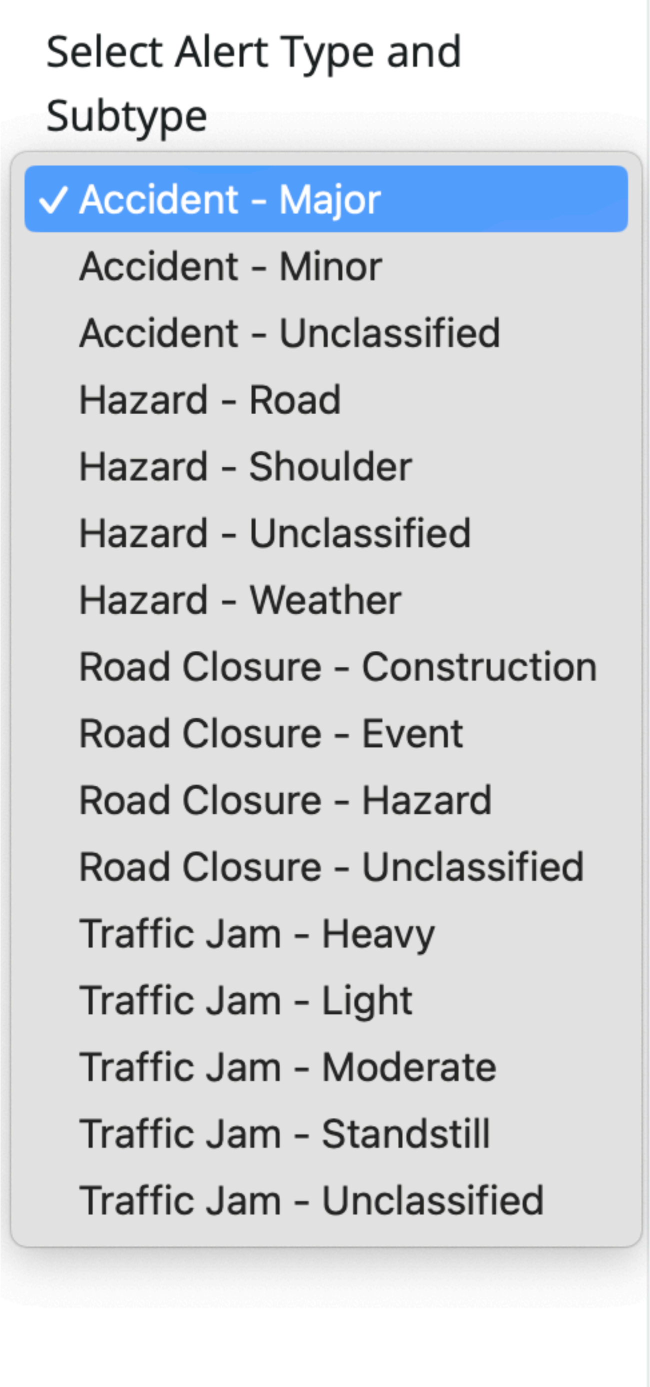
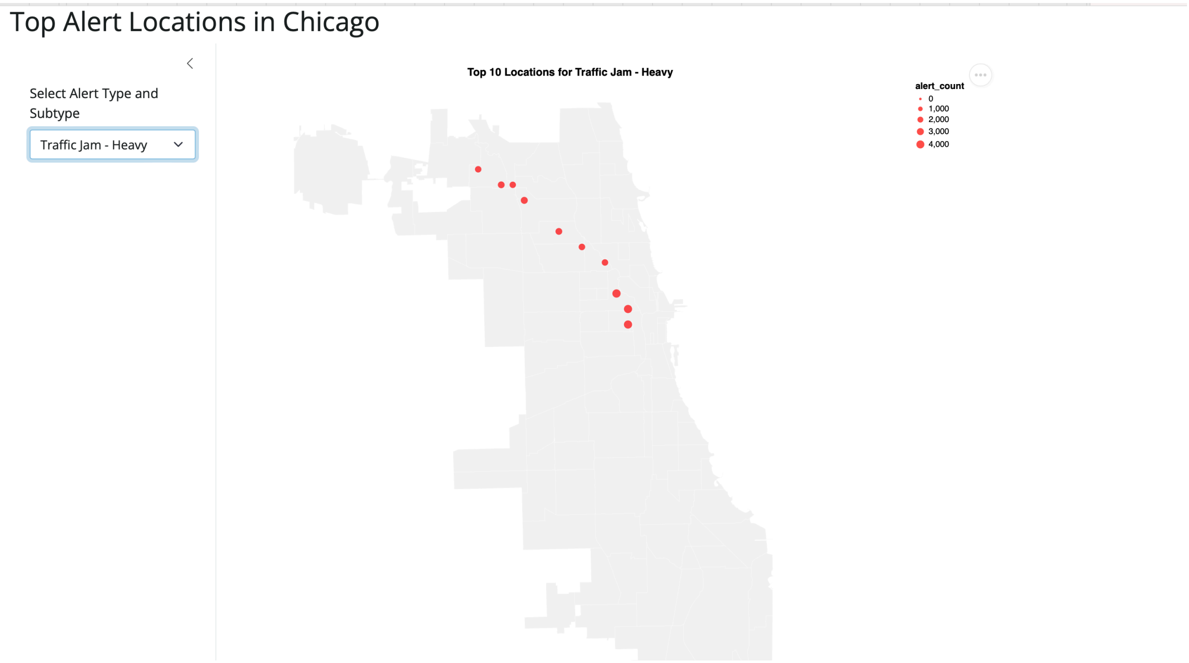


Figure 1: Dropdown menu showing type-subtype combinations

b.



Top Alert Locations in Chicago

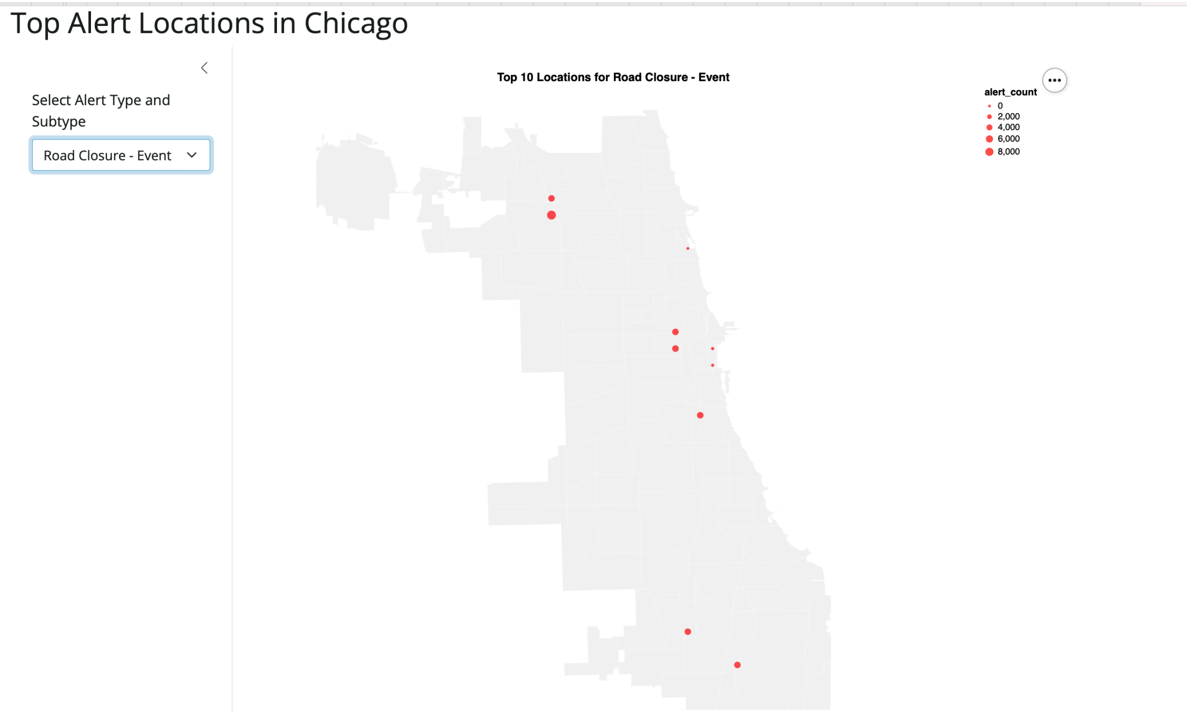
Select Alert Type and Subtype

Traffic Jam - Heavy

Figure 2: Map showing top 10 locations for “Traffic Jam - Heavy”

The map above shows the top 10 locations where heavy traffic jams were reported, with larger circles indicating more alerts at that location.

c.



Top Alert Locations in Chicago

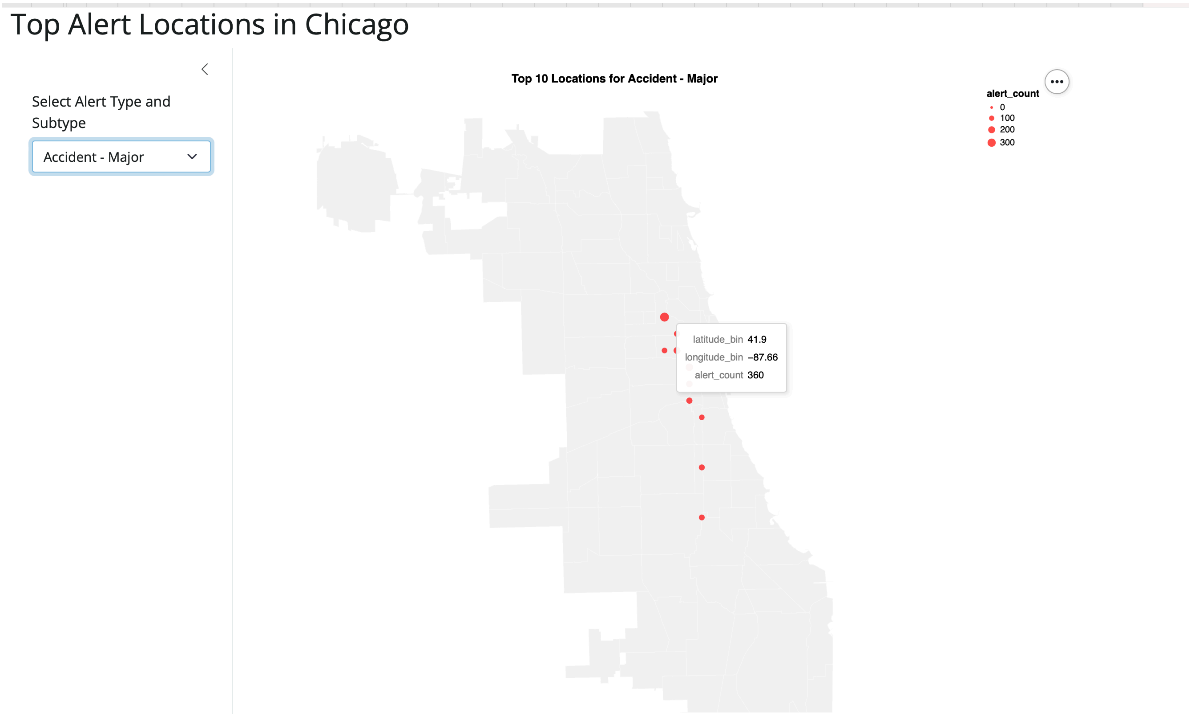
Select Alert Type and Subtype

Road Closure - Event

Figure 3: Map showing most common location for road closure due to event

latitude_bin:41.96, longitude_bin:-87.75 is the most common location for road closure due to event.

d. What is the most common location for major accidents? latitude_bin:41.9, longitude_bin:-87.66 is the most common location for major accidents.



Top Alert Locations in Chicago

Select Alert Type and Subtype

Accident - Major

Figure 4: Map showing most common location for major accidents

e. I suggest adding a reliability score column to enhance the dashboard. Since our dataset includes both `reliability` and `confidence` metrics, this addition would help users filter alerts based on their trustworthiness. For example, when analyzing traffic jams, users could focus on high-reliability reports (e.g., those with multiple thumbs-up and high confidence scores), making their route planning more dependable. This feature would be particularly useful during rush hours when accurate traffic information is crucial. The reliability score could be displayed using color intensity on the map, making it visually intuitive to identify the most reliable alerts.

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

1.

a. Given the timestamp format in our dataset (e.g., “2024-02-04 16:40:41 UTC”), it would not be a good idea to collapse the dataset directly by the ts column. The timestamps contain very detailed information including year, month, day, hours, minutes, and seconds in UTC format. Since our goal is to analyze traffic patterns by hour of the day, collapsing by the exact timestamp would create unnecessary fragmentation of the data, where similar events occurring at slightly different times (like 16:40:41 and 16:41:00) would be treated as separate groups. Instead, it would be more meaningful to extract just the hour component from these timestamps, allowing us to aggregate traffic patterns into 24 hourly slots that better represent daily traffic trends.

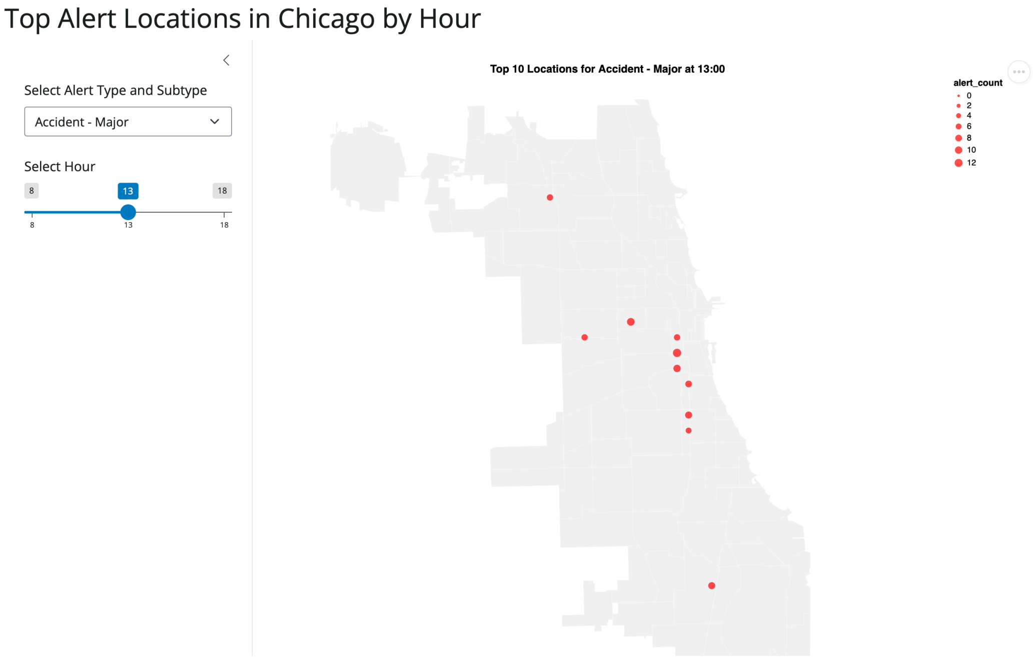
b.

Saved hourly aggregation data with 62825 rows.

c.

2.

a.



Top Alert Locations in Chicago by Hour

Select Alert Type and Subtype

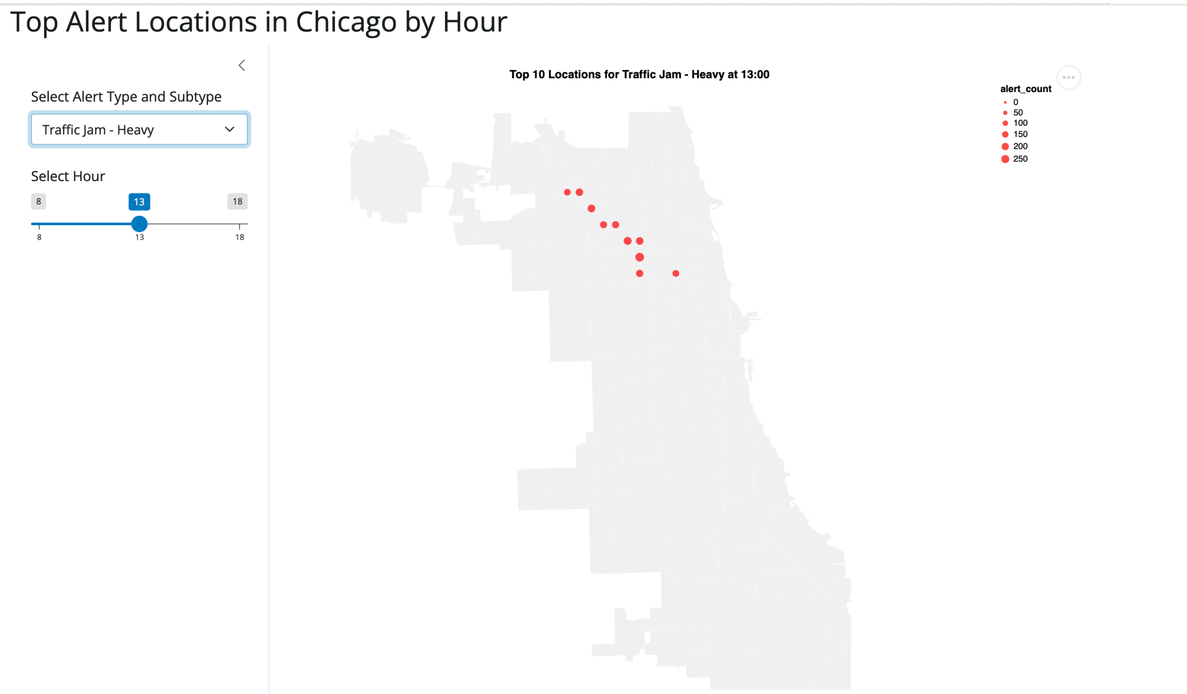
Accident - Major

Select Hour

13

Top 10 Locations for Accident - Major at 13:00

Figure 5: UI for selection with hour slider



Top Alert Locations in Chicago by Hour

Select Alert Type and Subtype

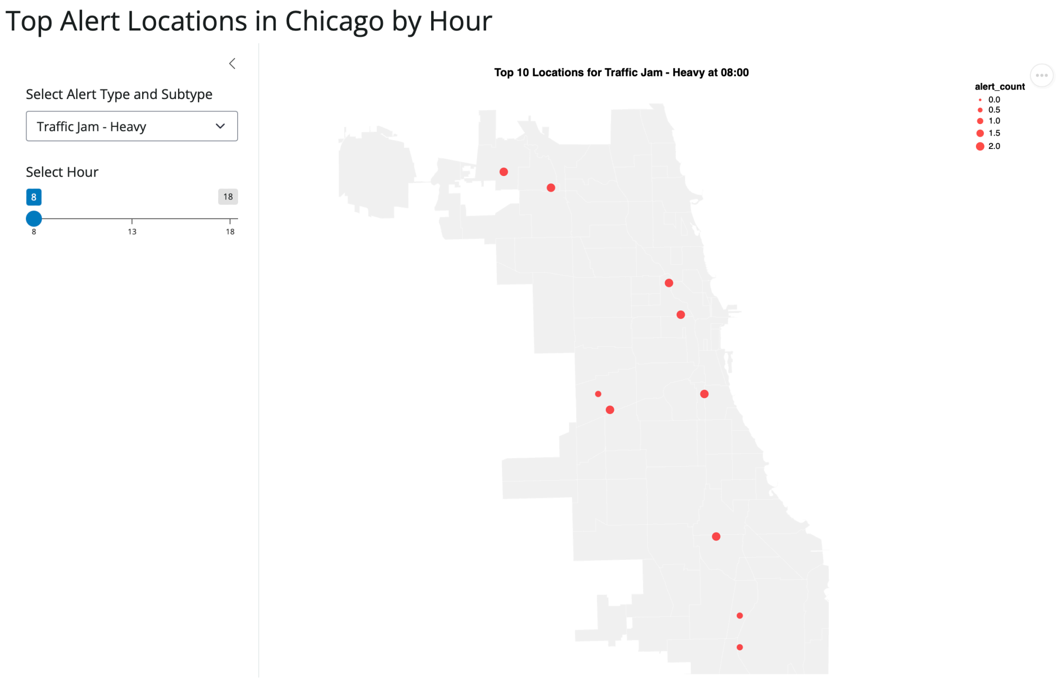
Traffic Jam - Heavy

Select Hour

13

Figure 6: UI for selection with hour slider

b.



Top Alert Locations in Chicago by Hour

Select Alert Type and Subtype

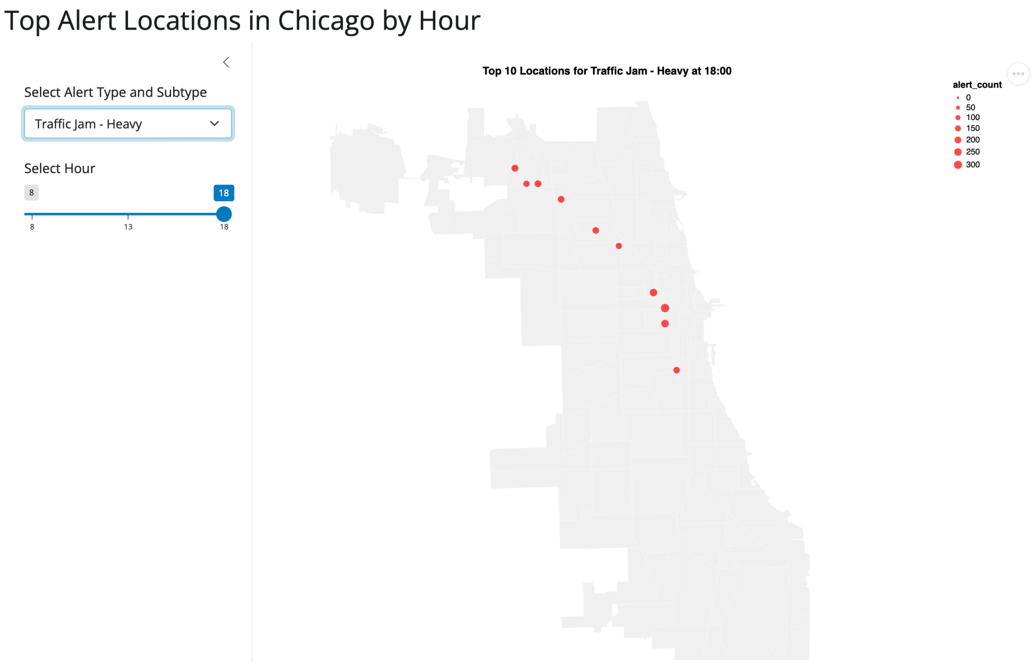
Traffic Jam - Heavy

Select Hour

8

18

Figure 7: Jam - Heavy Traffic with time slider



Top Alert Locations in Chicago by Hour

Select Alert Type and Subtype

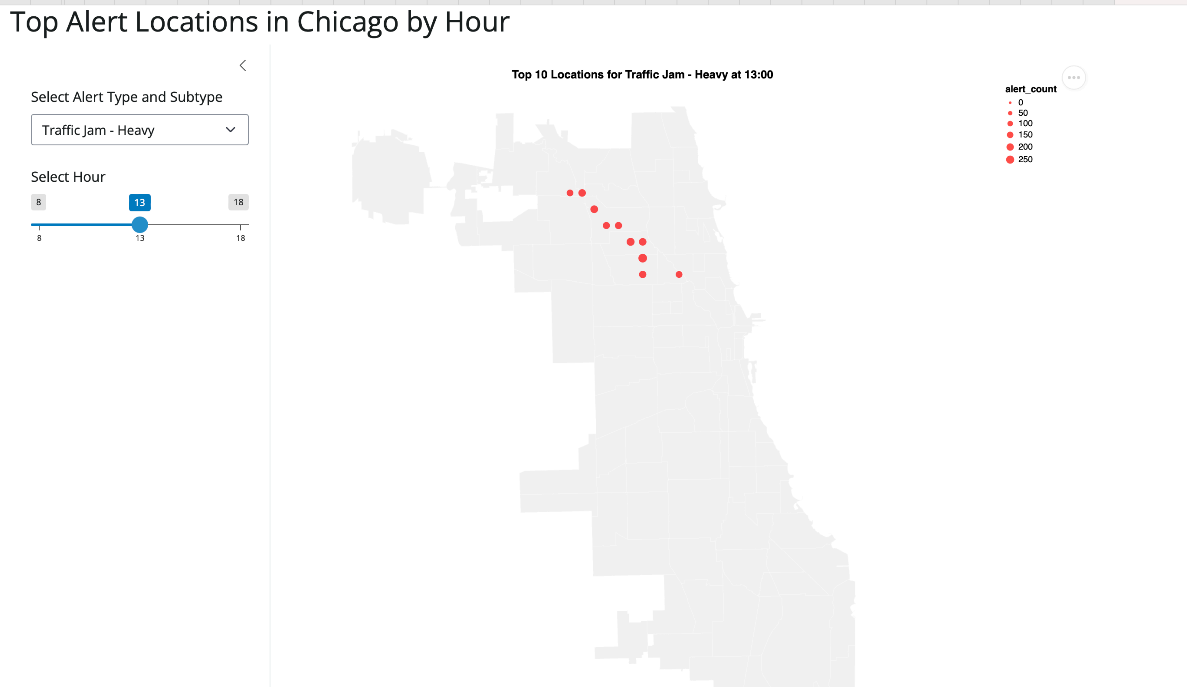
Traffic Jam - Heavy

Select Hour

8

18

Figure 8: Jam - Heavy Traffic with time slider



Top Alert Locations in Chicago by Hour

Select Alert Type and Subtype

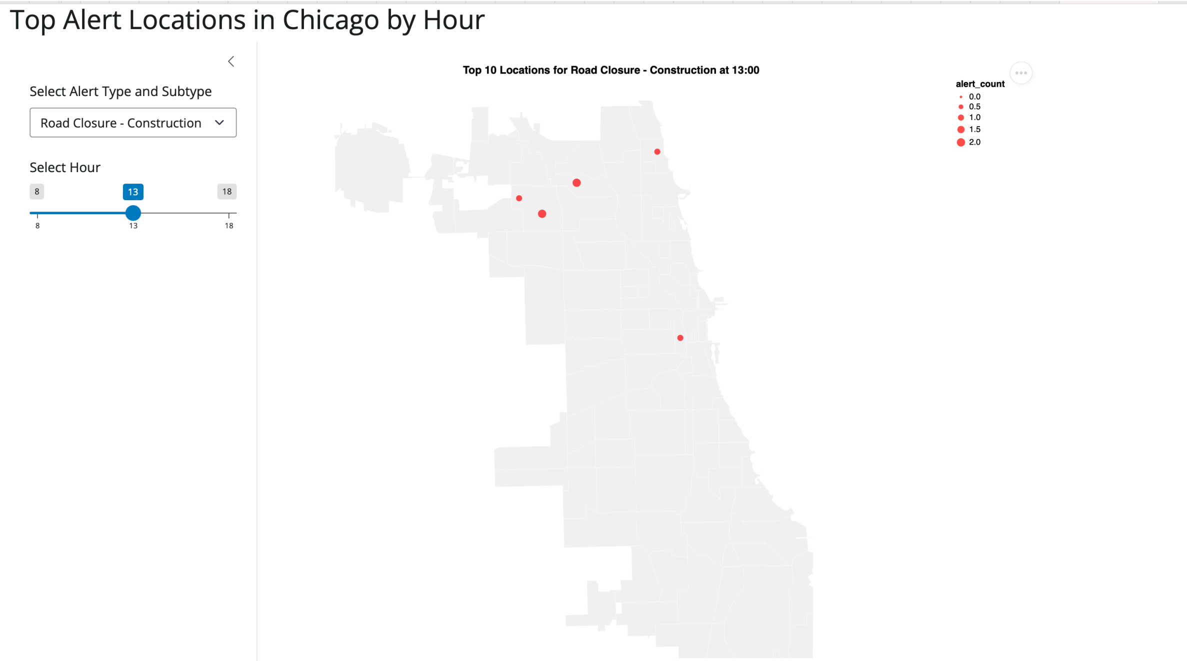
Traffic Jam - Heavy

Select Hour

13

Figure 9: Jam - Heavy Traffic with time slider

c. The road construction is done more in evening hours from the map.



Top Alert Locations in Chicago by Hour

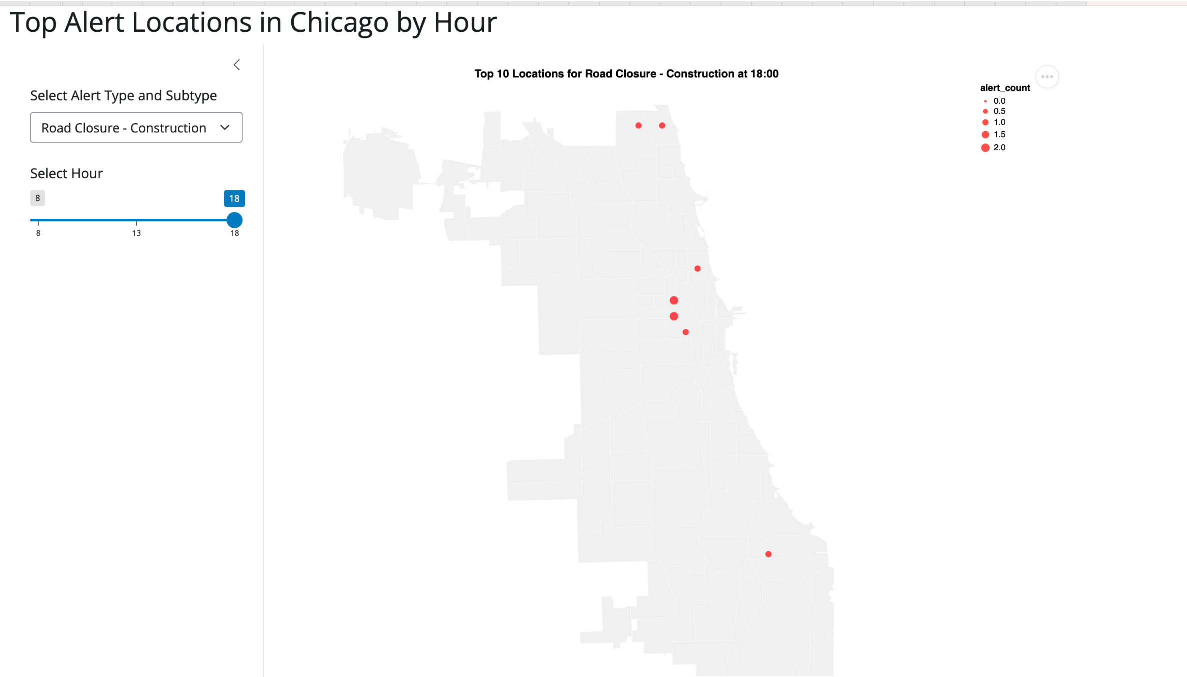
Select Alert Type and Subtype

Road Closure - Construction

Select Hour

13

Figure 10: Road Construction map



Top Alert Locations in Chicago by Hour

Select Alert Type and Subtype

Road Closure - Construction

Select Hour

18

Figure 11: Road Construction map

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

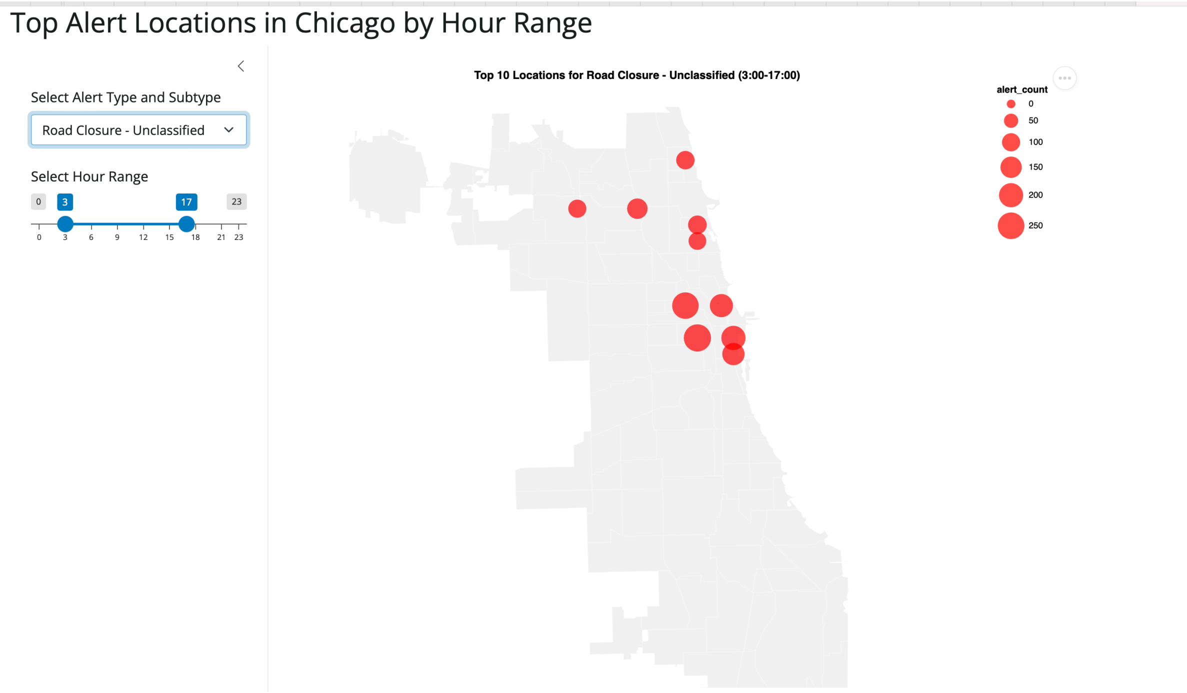
1.

a. For this app, it might be better to not collapse the dataset by range of hours initially. Instead, allow the app to dynamically aggregate data based on user-selected hour ranges. This approach provides more flexibility and allows users to explore the data in various ways.

b.

2.

a. This is the UI before I adjust the size of the circles, which does not have the switch button. I put it here for reference.



Top Alert Locations in Chicago by Hour Range

Select Alert Type and Subtype

Road Closure - Unclassified

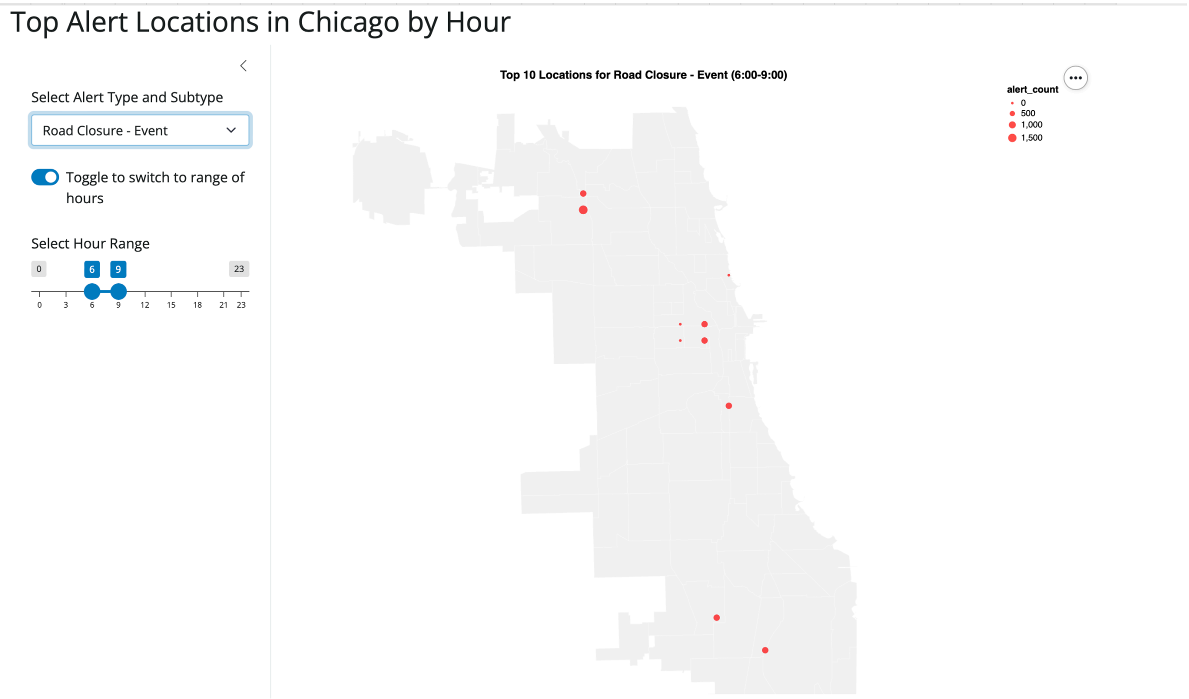
Select Hour Range

3 – 17

0
23
0 3 6 9 12 15 18 21 23
Top 10 Locations for Road Closure - Unclassified (3:00-17:00)
alert_count
0
50
100
150
200
250

Figure 12: Slider range

This one is after I adjust the size of the circles, which is why it already has the switch button.



Top Alert Locations in Chicago by Hour

Select Alert Type and Subtype

Road Closure - Event

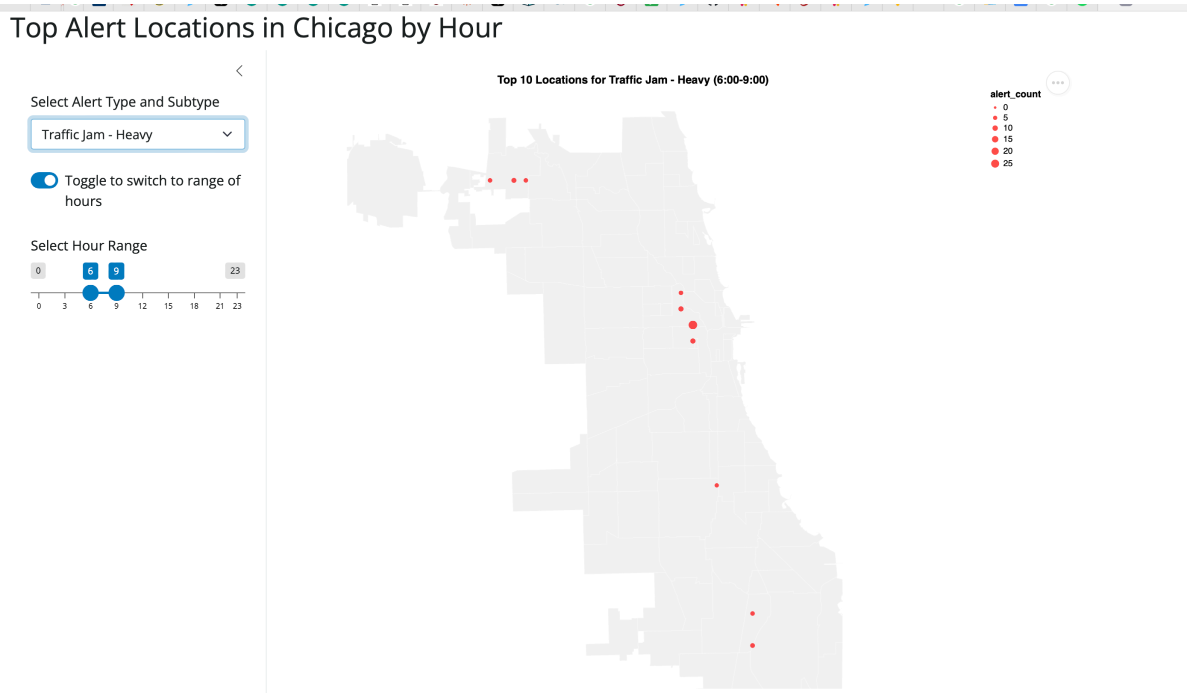
Toggle to switch to range of hours

Select Hour Range

Top 10 Locations for Road Closure - Event (6:00-9:00)

Figure 13: Slider range

b.



Top Alert Locations in Chicago by Hour

Select Alert Type and Subtype

Traffic Jam - Heavy

Toggle to switch to range of hours

Select Hour Range

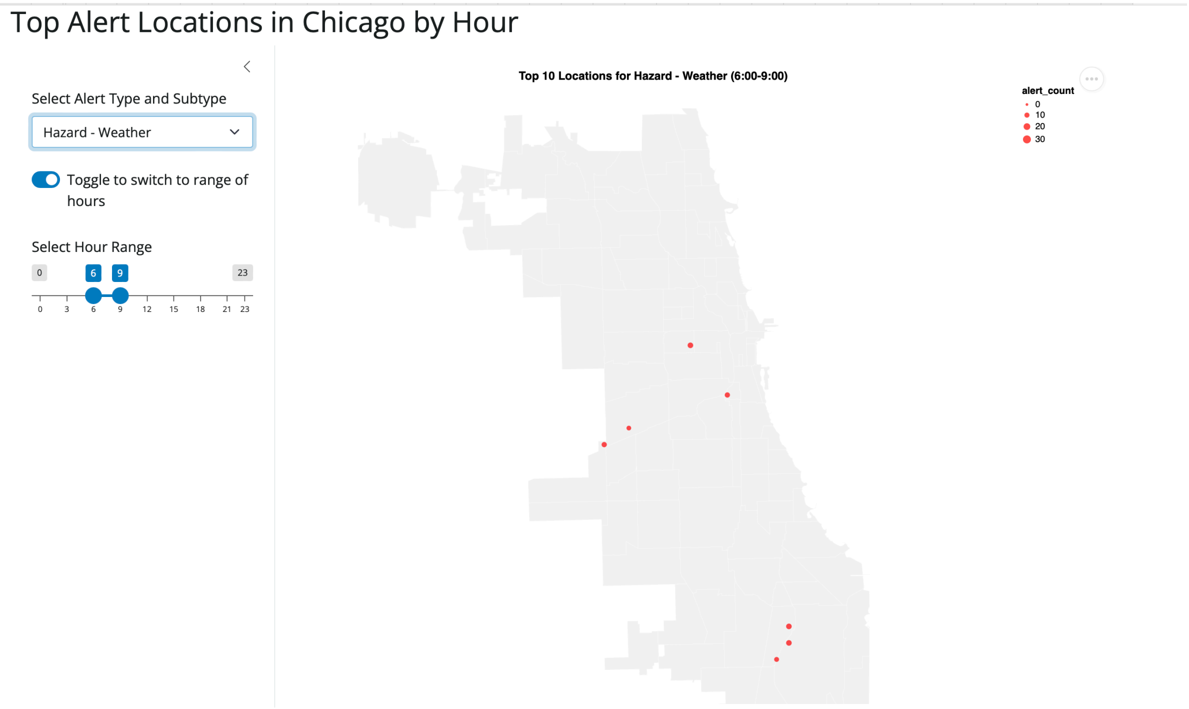
0 6 9 23

Top 10 Locations for Traffic Jam - Heavy (6:00-9:00)

Figure 14: Slider range

3.

a. The possible values are True or 1 when the switch button is on, and False or 0 when the switch button is off.



Top Alert Locations in Chicago by Hour

Select Alert Type and Subtype

Hazard - Weather

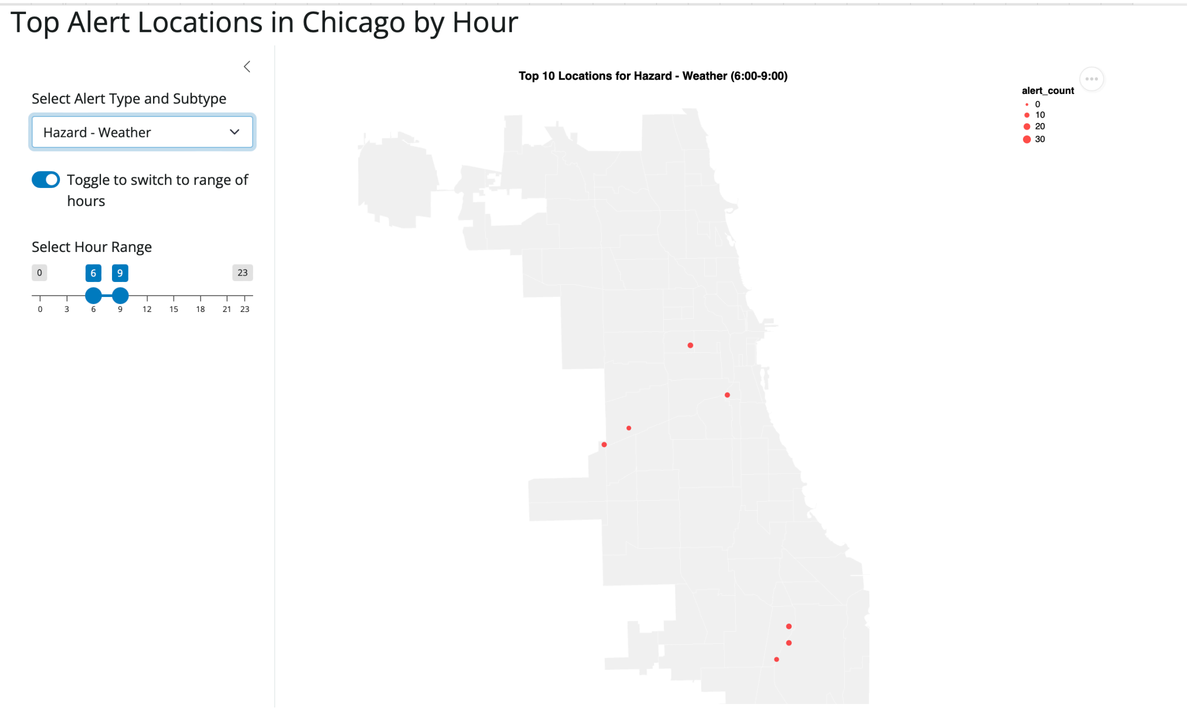
Toggle to switch to range of hours

Select Hour Range

6 9

Figure 15: Toggle button

b.



Top Alert Locations in Chicago by Hour

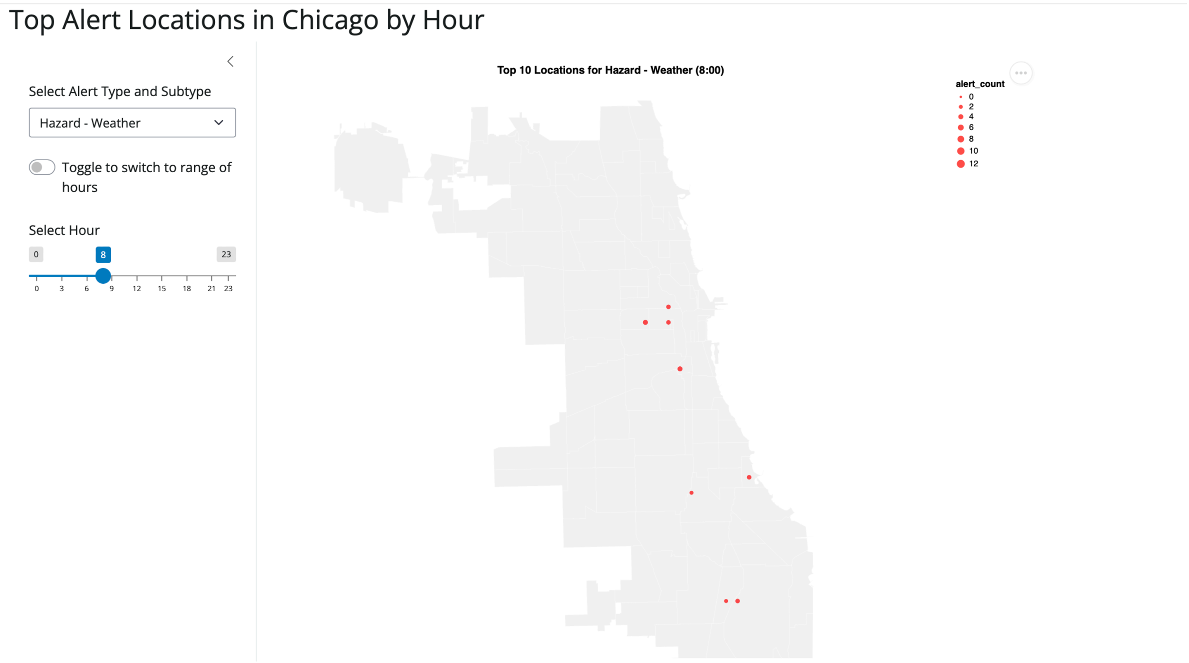
Select Alert Type and Subtype: Hazard - Weather

Toggle to switch to range of hours

Select Hour Range: 6 – 9

Top 10 Locations for Hazard - Weather (6:00-9:00)

Figure 16: Toggle button



Top Alert Locations in Chicago by Hour

Select Alert Type and Subtype: Hazard - Weather

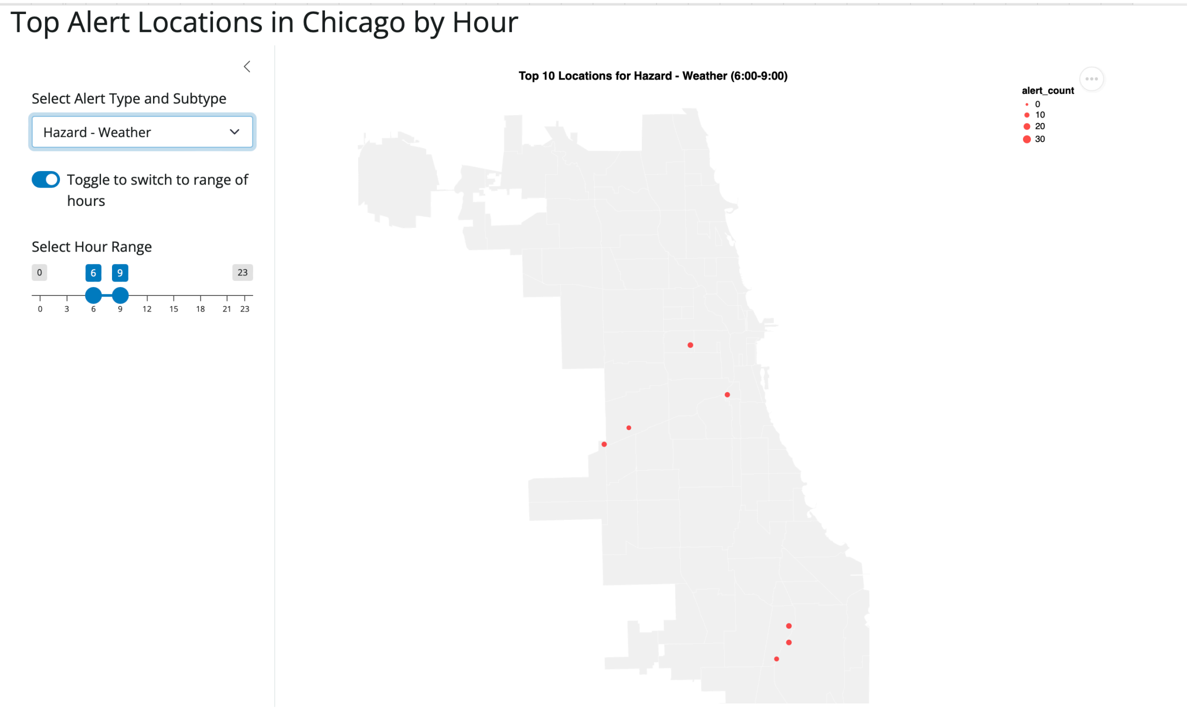
Toggle to switch to range of hours

Select Hour: 8

Top 10 Locations for Hazard - Weather (8:00)

Figure 17: Toggle button

c.



Top Alert Locations in Chicago by Hour

Select Alert Type and Subtype

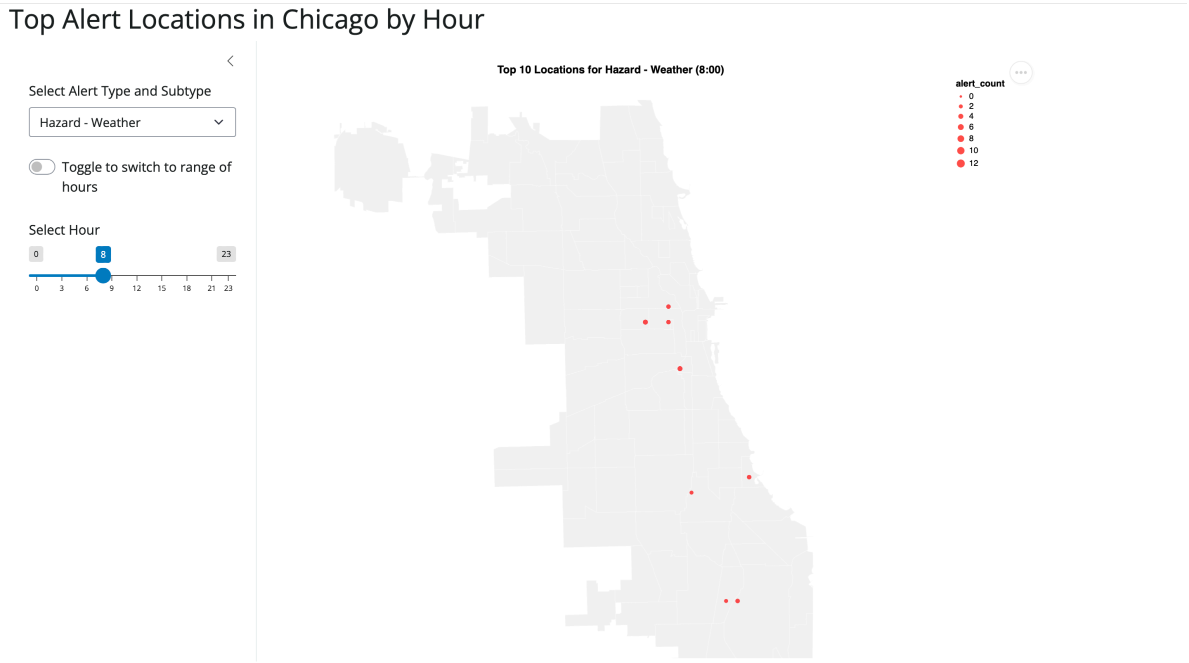
Hazard - Weather

Toggle to switch to range of hours

Select Hour Range

0 6 9 23

Figure 18: Toggle button



Top Alert Locations in Chicago by Hour

Select Alert Type and Subtype

Hazard - Weather

○ Toggle to switch to range of hours

Select Hour: 8

Figure 19: Toggle button

d. I need to add color encoding for time periods into morning and afternoon. I need to ensure data is aggredated by the defined time periods to reflect the correct number of alerts. I need to add grid lines and borders to map, adjist the opacity and fill of the map to highlight the points more effectively. I also need to adjust the size of the circles to make them better detailedly correspond to the longititute and latitude bins.