Problem Set 6 - Waze Shiny Dashboard

Yuting Meng

2024-11-14

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: YM
2. "I have uploaded the names of anyone I worked with on the problem set **here**" YM (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. Submit your `ps6.qmd`, `ps6.pdf`, `requirements.txt`, and all created folders (we will create three Shiny apps so you will have at least three additional folders) to the gradescope repo assignment (5 points).
7. Submit `ps6.pdf` and also link your Github repo via Gradescope (5 points)
8. Tag your submission in Gradescope. For the Code Style part (10 points) please tag the whole correspondingsection for the code style rubric.

Notes: see the Quarto documentation (link) for directions on inserting images into your knitted document.

IMPORTANT: For the App portion of the PS, in case you can not arrive to the expected functional dashboard we will need to take a look at your `app.py` file. You can use the following *code chunk template to "import" and print the content of that file. Please, don't forget to also tag the corresponding code chunk as part of your submission!*

Background

Data Download and Exploration (20 points)

1.

```
sample_df = pd.read_csv('waze_data_sample.csv')

print(sample_df.columns)
```

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

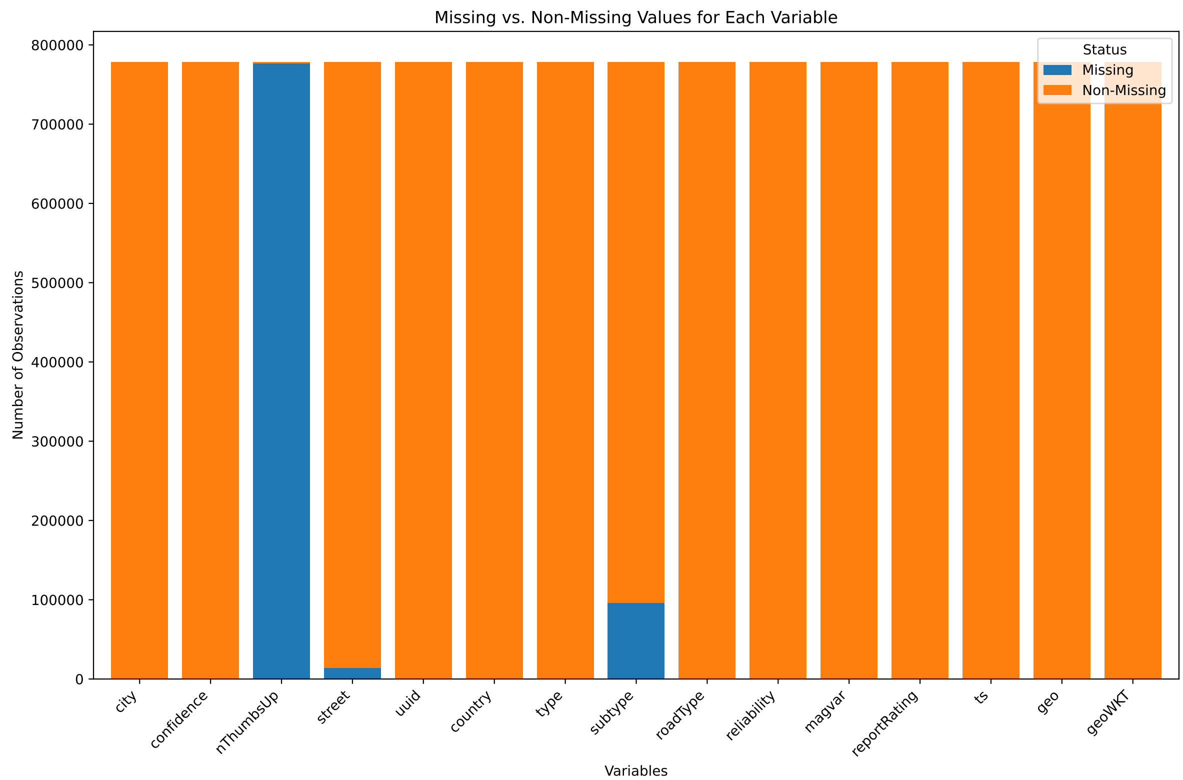
```
variable_types = sample_df.dtypes.to_dict()

altair_types = {
    'int64': 'Quantitative',
    'float64': 'Quantitative',
    'object': 'Nominal'
}

altair_variable_types = {col: altair_types[str(dtype)] for col, dtype in
↪  variable_types.items() if col not in ['ts', 'geo', 'geoWKT']}

altair_variable_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.

```python
import matplotlib.pyplot as plt
data_df = pd.read_csv('waze_data.csv')

null_counts = data_df.isnull().sum()
non_null_counts = data_df.notnull().sum()

null_data = pd.DataFrame({
    'Variable': data_df.columns,
    'Missing': null_counts,
    'Non-Missing': non_null_counts
})

fig, ax = plt.subplots(figsize=(12, 8))

null_data.set_index('Variable')[['Missing', 'Non-Missing']].plot(
    kind='bar', stacked=True, ax=ax, width=0.8
)


ax.set_title('Missing vs. Non-Missing Values for Each Variable')
ax.set_xlabel('Variables')
ax.set_ylabel('Number of Observations')
ax.legend(title='Status')

plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

missing_vars = null_data[null_data['Missing'] > 0]['Variable'].tolist()
highest_missing_var = null_counts.idxmax()

print("Variables with missing values:", missing_vars)
print("Variable with the highest share of missing observations:",
↪ highest_missing_var)
```




```
Variables with missing values: ['nThumbsUp', 'street', 'subtype']
Variable with the highest share of missing observations: nThumbsUp
```

3.

```
data_df = pd.read_csv('waze_data.csv')

type_unique = data_df['type'].unique()
subtype_unique = data_df['subtype'].unique()

type_mapping = {
    original: f"Clean_{original}" for original in type_unique
}

subtype_mapping = {
    original: f"Clean_{original}" for original in subtype_unique
}

type_crosswalk_df = pd.DataFrame({
    'Original Type': list(type_mapping.keys()),
```

```python
    'Cleaned Type': list(type_mapping.values())
})

subtype_crosswalk_df = pd.DataFrame({
    'Original Subtype': list(subtype_mapping.keys()),
    'Cleaned Subtype': list(subtype_mapping.values())
})

type_crosswalk_df, subtype_crosswalk_df
```

```
(  Original Type       Cleaned Type
 0           JAM          Clean_JAM
 1      ACCIDENT     Clean_ACCIDENT
 2   ROAD_CLOSED  Clean_ROAD_CLOSED
 3        HAZARD       Clean_HAZARD,
                       Original Subtype  \
 0                                  NaN
 1                       ACCIDENT_MAJOR
 2                       ACCIDENT_MINOR
 3                       HAZARD_ON_ROAD
 4           HAZARD_ON_ROAD_CAR_STOPPED
 5          HAZARD_ON_ROAD_CONSTRUCTION
 6     HAZARD_ON_ROAD_EMERGENCY_VEHICLE
 7                   HAZARD_ON_ROAD_ICE
 8                HAZARD_ON_ROAD_OBJECT
 9              HAZARD_ON_ROAD_POT_HOLE
 10  HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT
 11                  HAZARD_ON_SHOULDER
 12      HAZARD_ON_SHOULDER_CAR_STOPPED
 13                      HAZARD_WEATHER
 14                HAZARD_WEATHER_FLOOD
 15                   JAM_HEAVY_TRAFFIC
 16                JAM_MODERATE_TRAFFIC
 17             JAM_STAND_STILL_TRAFFIC
 18                   ROAD_CLOSED_EVENT
 19           HAZARD_ON_ROAD_LANE_CLOSED
 20                  HAZARD_WEATHER_FOG
 21            ROAD_CLOSED_CONSTRUCTION
 22              HAZARD_ON_ROAD_ROAD_KILL
 23          HAZARD_ON_SHOULDER_ANIMALS
 24     HAZARD_ON_SHOULDER_MISSING_SIGN
 25                   JAM_LIGHT_TRAFFIC
```

```
26          HAZARD_WEATHER_HEAVY_SNOW
27                 ROAD_CLOSED_HAZARD
28                HAZARD_WEATHER_HAIL

                              Cleaned Subtype
0                                   Clean_nan
1                       Clean_ACCIDENT_MAJOR
2                       Clean_ACCIDENT_MINOR
3                       Clean_HAZARD_ON_ROAD
4           Clean_HAZARD_ON_ROAD_CAR_STOPPED
5          Clean_HAZARD_ON_ROAD_CONSTRUCTION
6     Clean_HAZARD_ON_ROAD_EMERGENCY_VEHICLE
7                   Clean_HAZARD_ON_ROAD_ICE
8                Clean_HAZARD_ON_ROAD_OBJECT
9              Clean_HAZARD_ON_ROAD_POT_HOLE
10  Clean_HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT
11                  Clean_HAZARD_ON_SHOULDER
12      Clean_HAZARD_ON_SHOULDER_CAR_STOPPED
13                      Clean_HAZARD_WEATHER
14                Clean_HAZARD_WEATHER_FLOOD
15                   Clean_JAM_HEAVY_TRAFFIC
16                Clean_JAM_MODERATE_TRAFFIC
17             Clean_JAM_STAND_STILL_TRAFFIC
18                   Clean_ROAD_CLOSED_EVENT
19          Clean_HAZARD_ON_ROAD_LANE_CLOSED
20                  Clean_HAZARD_WEATHER_FOG
21            Clean_ROAD_CLOSED_CONSTRUCTION
22            Clean_HAZARD_ON_ROAD_ROAD_KILL
23          Clean_HAZARD_ON_SHOULDER_ANIMALS
24     Clean_HAZARD_ON_SHOULDER_MISSING_SIGN
25                   Clean_JAM_LIGHT_TRAFFIC
26          Clean_HAZARD_WEATHER_HEAVY_SNOW
27                  Clean_ROAD_CLOSED_HAZARD
28                 Clean_HAZARD_WEATHER_HAIL  )
```

```python
unique_types = data_df['type'].unique()
unique_subtypes = data_df['subtype'].unique()

print("Unique values in 'type':", unique_types)
print("Unique values in 'subtype':", unique_subtypes)

# Step 2: Count the number of types with NA subtypes
```

```python
na_subtypes_count = data_df[data_df['subtype'].isna()]['type'].nunique()
print("Number of unique 'type' values with NA subtypes:", na_subtypes_count)
```

```
Unique values in 'type': ['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']
Unique values in 'subtype': [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 unique 'type' values with NA subtypes: 4
```

There are 4 unique type values (JAM, ACCIDENT, ROAD_CLOSED, HAZARD) that have NA values in subtype.

Based on the type and subtype combinations, HAZARD and JAM appear to have enough detailed information that they could have hierarchical sub-subtypes. For example: HAZARD has various conditions like ON_ROAD, ON_SHOULDER, WEATHER, each with further details. JAM has HEAVY_TRAFFIC, MODERATE_TRAFFIC, and STAND_STILL_TRAFFIC subtypes, which could represent traffic severity.

Accident

Major Minor

Traffic Jam

Heavy Traffic, Moderate Traffic, Standstill Traffic, Light Traffic,

Hazard

1. On Road

Car Stopped, Construction, Emergency Vehicle, Ice, Object, Pothole, Traffic Light Fault, Lane Closed, Road Kill

2. On Shoulder

Car Stopped, Animals, Missing Sign

Weather

Flood, Fog, Heavy Snow, Hail

Road Closed

Due to Event, Due to Construction, Due to Hazard

Yes, we should keep the NA subtypes, but they should be recoded as "Unclassified" to avoid any ambiguity. NA subtypes might indicate a general classification that doesn't fit into more specific categories, but they still provide valuable data. Coding them as "Unclassified" preserves this information while clarifying its meaning for end-users.

4.

a.

```
columns = ['type', 'subtype', 'updated_type', 'updated_subtype',
↪  'updated_subsubtype']

crosswalk_df = pd.DataFrame(columns=columns)

crosswalk_df
```

type	subtype	updated_type	updated_subtype	updated_subsubtype

b.

```
crosswalk_data = [
    # Accident type
    {'type': 'ACCIDENT', 'subtype': None, 'updated_type': 'Accident',
↪   'updated_subtype': 'Unclassified', 'updated_subsubtype': None},
    {'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
    {'type': 'JAM', 'subtype': None, 'updated_type': 'Traffic Jam',
↪   'updated_subtype': 'Unclassified', 'updated_subsubtype': None},
    {'type': 'JAM', 'subtype': 'JAM_HEAVY_TRAFFIC', 'updated_type': 'Traffic
↪   Jam', 'updated_subtype': 'Heavy Traffic', 'updated_subsubtype': None},
    {'type': 'JAM', 'subtype': 'JAM_MODERATE_TRAFFIC', 'updated_type':
↪   'Traffic Jam', 'updated_subtype': 'Moderate Traffic',
↪   'updated_subsubtype': None},
    {'type': 'JAM', 'subtype': 'JAM_STAND_STILL_TRAFFIC', 'updated_type':
↪   'Traffic Jam', 'updated_subtype': 'Standstill Traffic',
↪   'updated_subsubtype': None},
    {'type': 'JAM', 'subtype': 'JAM_LIGHT_TRAFFIC', 'updated_type': 'Traffic
↪   Jam', 'updated_subtype': 'Light Traffic', 'updated_subsubtype': None},

    # Hazard type
    {'type': 'HAZARD', 'subtype': None, 'updated_type': 'Hazard',
↪   'updated_subtype': 'Unclassified', 'updated_subsubtype': None},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD', 'updated_type': 'Hazard',
↪   'updated_subtype': 'On Road', 'updated_subsubtype': None},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_CAR_STOPPED',
↪   'updated_type': 'Hazard', 'updated_subtype': 'On Road',
↪   'updated_subsubtype': 'Car Stopped'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_CONSTRUCTION',
↪   'updated_type': 'Hazard', 'updated_subtype': 'On Road',
↪   'updated_subsubtype': 'Construction'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_EMERGENCY_VEHICLE',
↪   'updated_type': 'Hazard', 'updated_subtype': 'On Road',
↪   'updated_subsubtype': 'Emergency Vehicle'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_ICE', 'updated_type':
↪   'Hazard', 'updated_subtype': 'On Road', 'updated_subsubtype': 'Ice'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_OBJECT', 'updated_type':
↪   'Hazard', 'updated_subtype': 'On Road', 'updated_subsubtype': 'Object'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_POT_HOLE', 'updated_type':
↪   'Hazard', 'updated_subtype': 'On Road', 'updated_subsubtype': 'Pothole'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT',
↪   'updated_type': 'Hazard', 'updated_subtype': 'On Road',
↪   'updated_subsubtype': 'Traffic Light Fault'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_LANE_CLOSED',
↪   'updated_type': 'Hazard', 'updated_subtype': 'On Road',
↪   'updated_subsubtype': 'Lane Closed'},
```

```
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_ROAD_ROAD_KILL', 'updated_type':
↪   'Hazard', 'updated_subtype': 'On Road', 'updated_subsubtype': 'Road
↪   Kill'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_SHOULDER', 'updated_type':
↪   'Hazard', 'updated_subtype': 'On Shoulder', 'updated_subsubtype': None},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_SHOULDER_CAR_STOPPED',
↪   'updated_type': 'Hazard', 'updated_subtype': 'On Shoulder',
↪   'updated_subsubtype': 'Car Stopped'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_SHOULDER_ANIMALS',
↪   'updated_type': 'Hazard', 'updated_subtype': 'On Shoulder',
↪   'updated_subsubtype': 'Animals'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_ON_SHOULDER_MISSING_SIGN',
↪   'updated_type': 'Hazard', 'updated_subtype': 'On Shoulder',
↪   'updated_subsubtype': 'Missing Sign'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_WEATHER', 'updated_type': 'Hazard',
↪   'updated_subtype': 'Weather', 'updated_subsubtype': None},
    {'type': 'HAZARD', 'subtype': 'HAZARD_WEATHER_FLOOD', 'updated_type':
↪   'Hazard', 'updated_subtype': 'Weather', 'updated_subsubtype': 'Flood'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_WEATHER_FOG', 'updated_type':
↪   'Hazard', 'updated_subtype': 'Weather', 'updated_subsubtype': 'Fog'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_WEATHER_HEAVY_SNOW',
↪   'updated_type': 'Hazard', 'updated_subtype': 'Weather',
↪   'updated_subsubtype': 'Heavy Snow'},
    {'type': 'HAZARD', 'subtype': 'HAZARD_WEATHER_HAIL', 'updated_type':
↪   'Hazard', 'updated_subtype': 'Weather', 'updated_subsubtype': 'Hail'},

    # Road Closed type
    {'type': 'ROAD_CLOSED', 'subtype': None, 'updated_type': 'Road Closed',
↪   'updated_subtype': 'Event', 'updated_subsubtype': None},
    {'type': 'ROAD_CLOSED', 'subtype': 'ROAD_CLOSED_EVENT', 'updated_type':
↪   'Road Closed', 'updated_subtype': 'Event', 'updated_subsubtype': None},
    {'type': 'ROAD_CLOSED', 'subtype': 'ROAD_CLOSED_CONSTRUCTION',
↪   'updated_type': 'Road Closed', 'updated_subtype': 'Construction',
↪   'updated_subsubtype': None},
    {'type': 'ROAD_CLOSED', 'subtype': 'ROAD_CLOSED_HAZARD', 'updated_type':
↪   'Road Closed', 'updated_subtype': 'Hazard', 'updated_subsubtype': None},
]

crosswalk_df = pd.DataFrame(crosswalk_data)

print("Number of observations in the crosswalk:", crosswalk_df.shape[0])
```

```
crosswalk_df.head(32)
```

```
Number of observations in the crosswalk: 32
```

	type	subtype	updated_type	updated_su
0	ACCIDENT	None	Accident	Unclassified
1	ACCIDENT	ACCIDENT_MAJOR	Accident	Major
2	ACCIDENT	ACCIDENT_MINOR	Accident	Minor
3	JAM	None	Traffic Jam	Unclassified
4	JAM	JAM_HEAVY_TRAFFIC	Traffic Jam	Heavy Traff
5	JAM	JAM_MODERATE_TRAFFIC	Traffic Jam	Moderate T
6	JAM	JAM_STAND_STILL_TRAFFIC	Traffic Jam	Standstill T
7	JAM	JAM_LIGHT_TRAFFIC	Traffic Jam	Light Traffic
8	HAZARD	None	Hazard	Unclassified
9	HAZARD	HAZARD_ON_ROAD	Hazard	On Road
10	HAZARD	HAZARD_ON_ROAD_CAR_STOPPED	Hazard	On Road
11	HAZARD	HAZARD_ON_ROAD_CONSTRUCTION	Hazard	On Road
12	HAZARD	HAZARD_ON_ROAD_EMERGENCY_VEHICLE	Hazard	On Road
13	HAZARD	HAZARD_ON_ROAD_ICE	Hazard	On Road
14	HAZARD	HAZARD_ON_ROAD_OBJECT	Hazard	On Road
15	HAZARD	HAZARD_ON_ROAD_POT_HOLE	Hazard	On Road
16	HAZARD	HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT	Hazard	On Road
17	HAZARD	HAZARD_ON_ROAD_LANE_CLOSED	Hazard	On Road
18	HAZARD	HAZARD_ON_ROAD_ROAD_KILL	Hazard	On Road
19	HAZARD	HAZARD_ON_SHOULDER	Hazard	On Shoulder
20	HAZARD	HAZARD_ON_SHOULDER_CAR_STOPPED	Hazard	On Shoulder
21	HAZARD	HAZARD_ON_SHOULDER_ANIMALS	Hazard	On Shoulder
22	HAZARD	HAZARD_ON_SHOULDER_MISSING_SIGN	Hazard	On Shoulder
23	HAZARD	HAZARD_WEATHER	Hazard	Weather
24	HAZARD	HAZARD_WEATHER_FLOOD	Hazard	Weather
25	HAZARD	HAZARD_WEATHER_FOG	Hazard	Weather
26	HAZARD	HAZARD_WEATHER_HEAVY_SNOW	Hazard	Weather
27	HAZARD	HAZARD_WEATHER_HAIL	Hazard	Weather
28	ROAD_CLOSED	None	Road Closed	Event
29	ROAD_CLOSED	ROAD_CLOSED_EVENT	Road Closed	Event
30	ROAD_CLOSED	ROAD_CLOSED_CONSTRUCTION	Road Closed	Constructio
31	ROAD_CLOSED	ROAD_CLOSED_HAZARD	Road Closed	Hazard

c.

```python
merged_df = data_df.merge(crosswalk_df, on=['type', 'subtype'], how='left')

accident_unclassified_count = merged_df[(merged_df['updated_type'] ==
↪ 'Accident') &
                                        (merged_df['updated_subtype'] ==
↪ 'Unclassified')].shape[0]

print("Number of rows for Accident - Unclassified:",
↪ accident_unclassified_count)
```

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

d.

```python
crosswalk_unique = crosswalk_df[['type', 'subtype']].drop_duplicates()

merged_unique = merged_df[['type', 'subtype']].drop_duplicates()

same_values = set(map(tuple, crosswalk_unique.values)) == set(map(tuple,
↪ merged_unique.values))

print("Do crosswalk and merged dataset have the same values in type and
↪ subtype?", same_values)

if not same_values:
    missing_in_merged = crosswalk_unique.merge(merged_unique, on=['type',
↪ 'subtype'], how='left', indicator=True)
    missing_in_merged = missing_in_merged[missing_in_merged['_merge'] ==
↪ 'left_only'][['type', 'subtype']]

    missing_in_crosswalk = merged_unique.merge(crosswalk_unique, on=['type',
↪ 'subtype'], how='left', indicator=True)
    missing_in_crosswalk =
↪ missing_in_crosswalk[missing_in_crosswalk['_merge'] ==
↪ 'left_only'][['type', 'subtype']]

    print("Combinations in crosswalk but not in merged dataset:\n",
     ↪ missing_in_merged)
    print("Combinations in merged dataset but not in crosswalk:\n",
     ↪ missing_in_crosswalk)
```

```
Do crosswalk and merged dataset have the same values in type and subtype?
False
Combinations in crosswalk but not in merged dataset:
 Empty DataFrame
Columns: [type, subtype]
Index: []
Combinations in merged dataset but not in crosswalk:
 Empty DataFrame
Columns: [type, subtype]
Index: []
```

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

1.

a.

```python
import re

pattern = r"POINT\s*\(\s*([-]?\d+\.\d+)\s+([-]?\d+\.\d+)\s*\)"

def extract_coordinates(geo_string):
    if pd.notna(geo_string):
        geo_string = geo_string.strip()
        match = re.match(pattern, geo_string)
        if match:
            longitude = float(match.group(1))
            latitude = float(match.group(2))
            return pd.Series([latitude, longitude])
    return pd.Series([None, None])

merged_df[['latitude', 'longitude']] =
↪  merged_df['geo'].apply(extract_coordinates)

merged_df.head()
```

	city	confidence	nThumbsUp	street	uuid	country	type
0	Chicago, IL	0	NaN	NaN	004025a4-5f14-4cb7-9da6-2615daafbf37	US	JAM
1	Chicago, IL	1	NaN	NaN	ad7761f8-d3cb-4623-951d-dafb419a3ec3	US	ACCI
2	Chicago, IL	0	NaN	NaN	0e5f14ae-7251-46af-a7f1-53a5272cd37d	US	ROAI

	city	confidence	nThumbsUp	street	uuid	country	type
3	Chicago, IL	0	NaN	Alley	654870a4-a71a-450b-9f22-bc52ae4f69a5	US	JAM
4	Chicago, IL	0	NaN	Alley	926ff228-7db9-4e0d-b6cf-6739211ffc8b	US	JAM

b.

```
merged_df['binned_latitude'] = merged_df['latitude'].round(2)
merged_df['binned_longitude'] = merged_df['longitude'].round(2)

merged_df.head()
```

	city	confidence	nThumbsUp	street	uuid	country	type
0	Chicago, IL	0	NaN	NaN	004025a4-5f14-4cb7-9da6-2615daafbf37	US	JAM
1	Chicago, IL	1	NaN	NaN	ad7761f8-d3cb-4623-951d-dafb419a3ec3	US	ACC
2	Chicago, IL	0	NaN	NaN	0e5f14ae-7251-46af-a7f1-53a5272cd37d	US	ROA
3	Chicago, IL	0	NaN	Alley	654870a4-a71a-450b-9f22-bc52ae4f69a5	US	JAM
4	Chicago, IL	0	NaN	Alley	926ff228-7db9-4e0d-b6cf-6739211ffc8b	US	JAM

```
binned_counts = merged_df.groupby(['binned_latitude',
↪  'binned_longitude']).size().reset_index(name='count')

max_binned_location = binned_counts.loc[binned_counts['count'].idxmax()]

print("Binned latitude-longitude combination with the greatest number of
↪  observations:")
print(max_binned_location)
```

```
Binned latitude-longitude combination with the greatest number of
observations:
binned_latitude        41.88
binned_longitude      -87.65
count              21325.00
Name: 396, dtype: float64
```

c.

```python
top_alerts_df = (
    merged_df.groupby(['binned_latitude', 'binned_longitude', 'type',
 ↪  'subtype'])
    .size()
    .reset_index(name='alert_count')
)

top_10_alerts_df = top_alerts_df.sort_values(by='alert_count',
 ↪  ascending=False).head(10)

top_alerts_map_folder = 'top_alerts_map'
top_alerts_map_filepath = f'{top_alerts_map_folder}/top_alerts_map.csv'

import os
os.makedirs(top_alerts_map_folder, exist_ok=True)

top_10_alerts_df.to_csv(top_alerts_map_filepath, index=False)

aggregation_level = ['binned_latitude', 'binned_longitude', 'type',
 ↪  'subtype']
num_rows = top_10_alerts_df.shape[0]

print("Level of aggregation:", aggregation_level)
print("Number of rows in the DataFrame:", num_rows)
```

```
Level of aggregation: ['binned_latitude', 'binned_longitude', 'type',
'subtype']
Number of rows in the DataFrame: 10
```

2.

```python
import altair as alt

# Load data
top_10_alerts_df = pd.read_csv('top_alerts_map/top_alerts_map.csv')

# Define latitude and longitude bounds
lat_min, lat_max = top_10_alerts_df['binned_latitude'].min() - 0.01,
 ↪  top_10_alerts_df['binned_latitude'].max() + 0.01
lon_min, lon_max = top_10_alerts_df['binned_longitude'].min() - 0.01,
 ↪  top_10_alerts_df['binned_longitude'].max() + 0.01
```

```python
# Create scatter plot with updated axes
scatter_plot = alt.Chart(top_10_alerts_df).mark_circle().encode(
    x=alt.X('binned_latitude:Q', title='Latitude',
 ↪  scale=alt.Scale(domain=[lat_min, lat_max])),
    y=alt.Y('binned_longitude:Q', title='Longitude',
 ↪  scale=alt.Scale(domain=[lon_min, lon_max])),
    size=alt.Size('alert_count:Q', title='Number of Alerts',
 ↪  scale=alt.Scale(range=[100, 1000])),
    tooltip=['binned_latitude', 'binned_longitude', 'alert_count']
).properties(
    title='Top 10 Locations with Highest Number of Alerts',
    width=400,
    height=300
)

scatter_plot
```

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

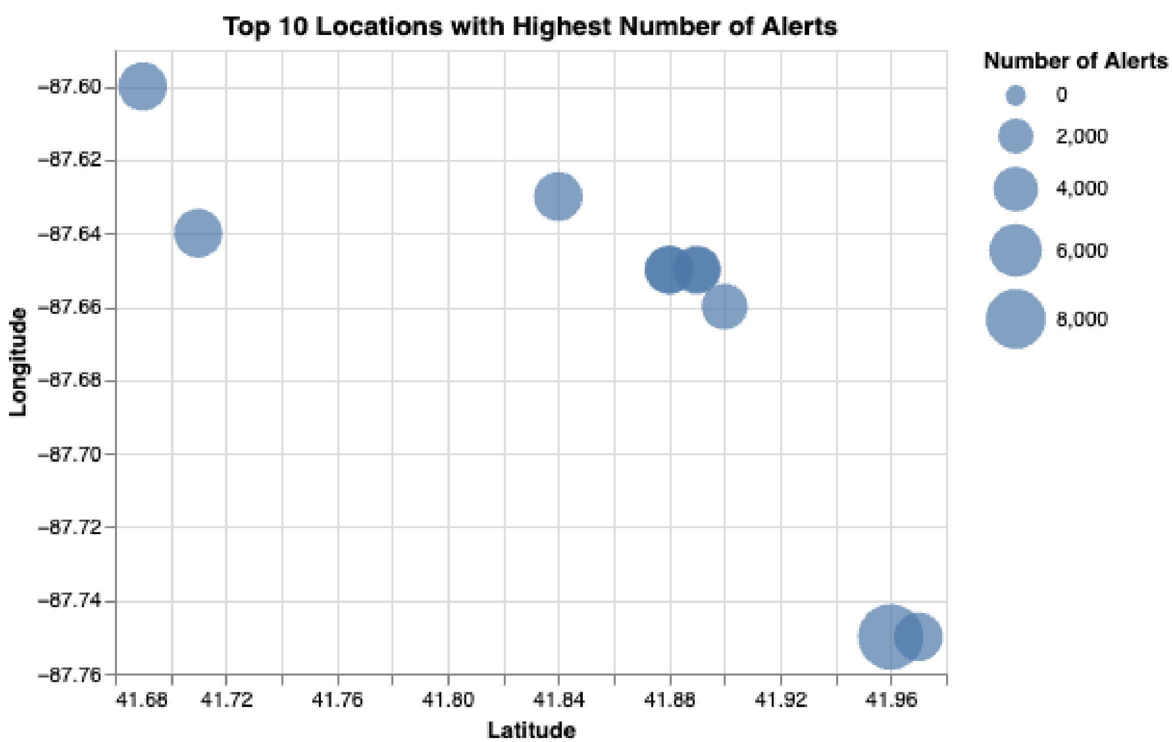


Figure 1: Top 10 Locations

3.

a.

```
import requests

url = "https://data.cityofchicago.org/resource/igwz-8jzy.geojson"

geojson_filepath = "chicago_neighborhoods.geojson"

response = requests.get(url)
if response.status_code == 200:
    with open(geojson_filepath, 'wb') as file:
        file.write(response.content)
    print("GeoJSON file downloaded and saved as:", geojson_filepath)
else:
    print("Failed to download GeoJSON file. Status code:",
     ↪  response.status_code)
```

```
GeoJSON file downloaded and saved as: chicago_neighborhoods.geojson
```

b.

```
file_path = "./chicago_neighborhoods.geojson"

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

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

4.

```
chicago_map = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).project(
    'mercator'
).properties(
    width=600,
    height=400
)
```

```
scatter_plot = alt.Chart(top_10_alerts_df).mark_circle().encode(
    longitude='binned_longitude:Q',
    latitude='binned_latitude:Q',
    size=alt.Size('alert_count:Q', title='Number of Alerts',
 ↪  scale=alt.Scale(range=[100, 800])),
    color=alt.value("red"),
    tooltip=['binned_latitude', 'binned_longitude', 'alert_count']
)

combined_chart = chicago_map + scatter_plot

combined_chart
```

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

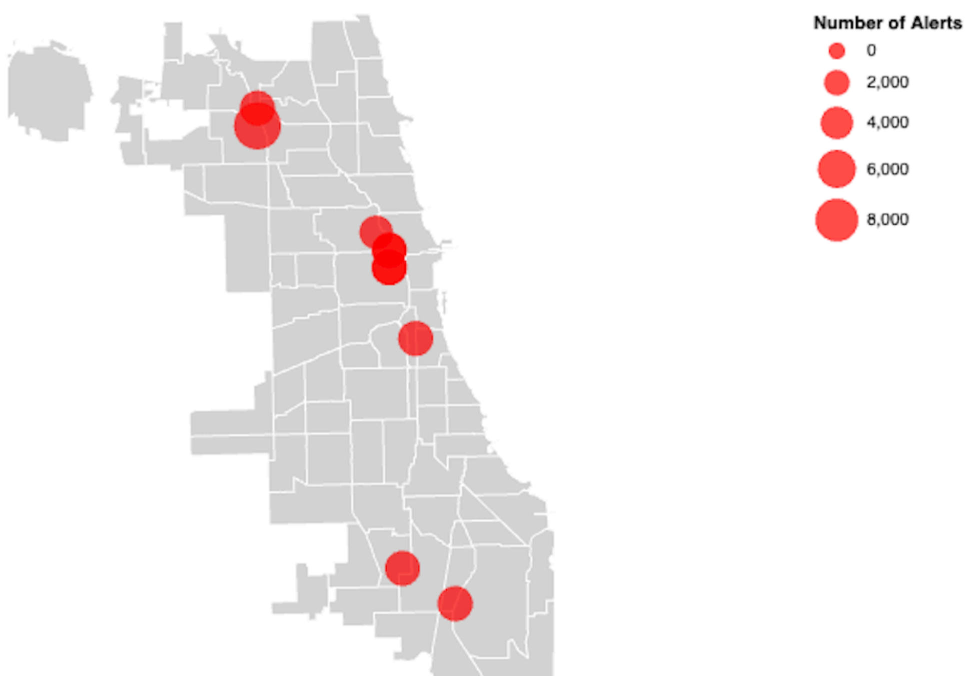


Figure 2: Combined Chart

5.

a.

The total number of unique type x subtype combinations is 3.

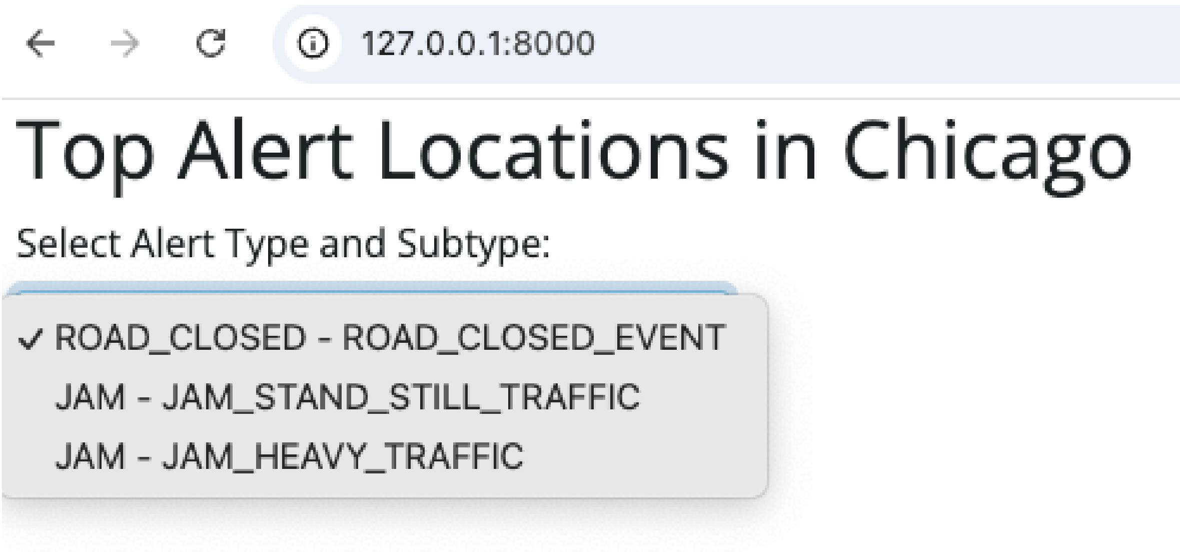
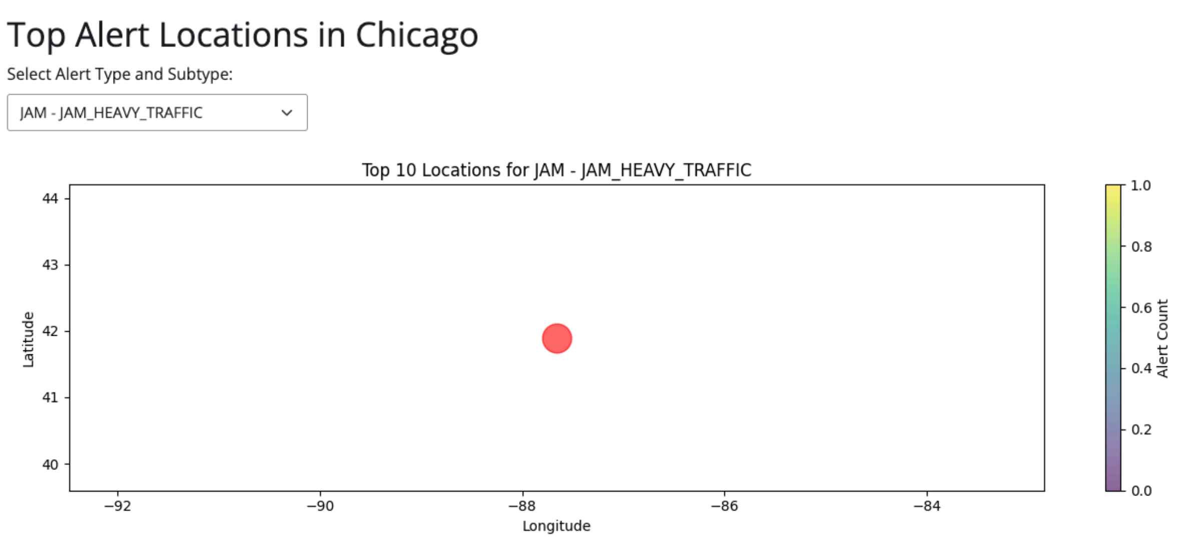


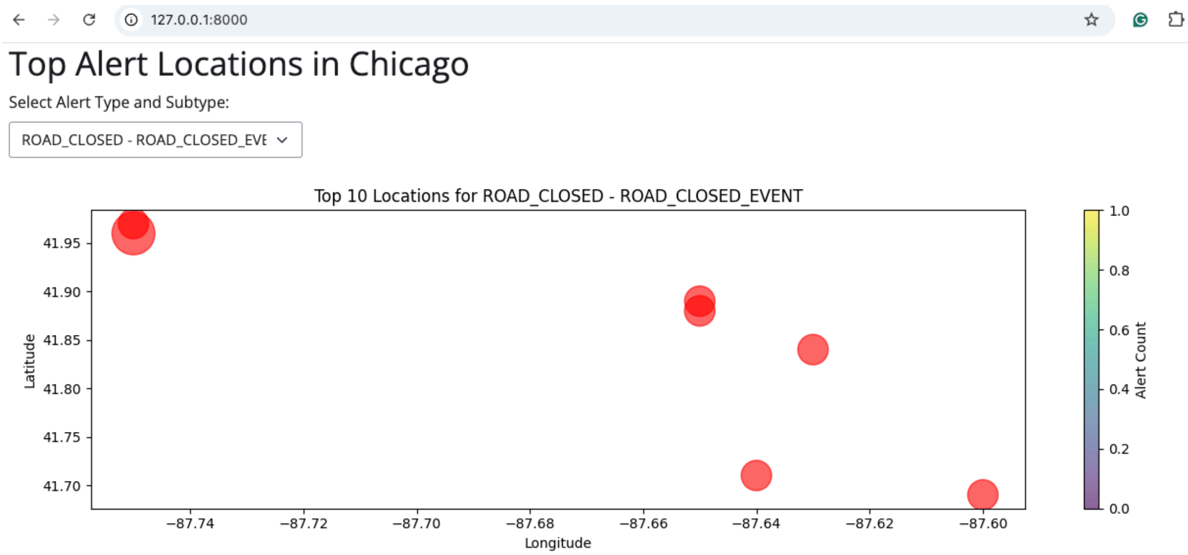
Figure 3: Dropdown menu screenshot

b.



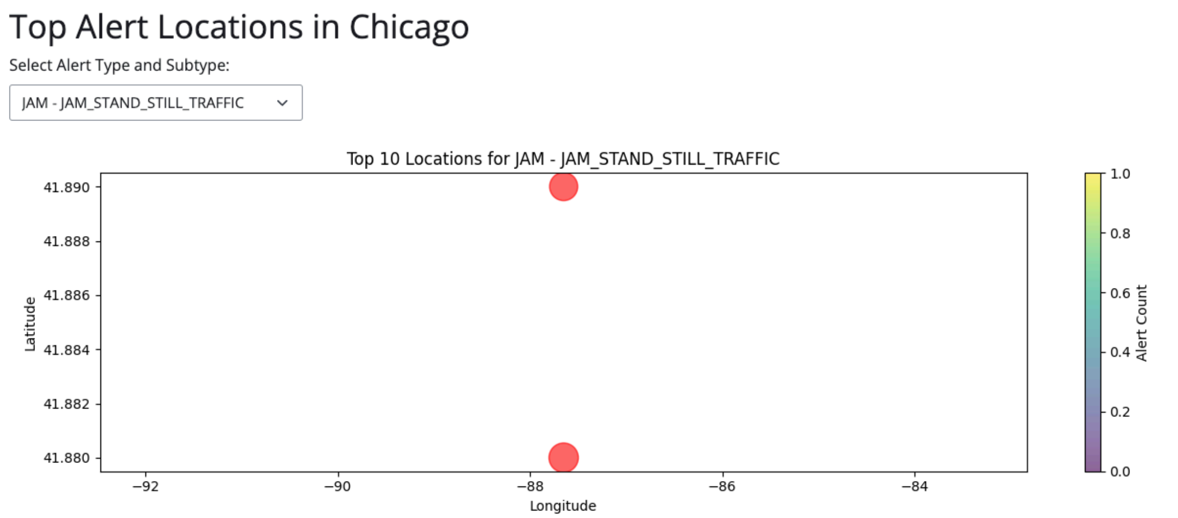
c.

According to the graph, alerts for road closures due to event is most common around the location of (-87.76, 41.97).



d.

Example question: how many locations in top 10 locations are due to jam stand still traffic? Answer would be two locations according to the graph.



e.

I can consider adding the time frame fo example "day of the week" column that could filter alerts based on when they occured, which would be helpful to understand the traffic patterns.

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

1.

a.

Given that the ts column represents a timestamp of the reported alert, it would not necessarily be ideal to collapse the dataset solely by this column, depending on the goal of the analysis.

The reason is that collapsing by timestamp would aggregate all alerts that occurred at exactly the same moment in time, which might not provide meaningful insights if the aim is to analyze patterns by other dimensions, such as location or type of alert. Timestamps in traffic or alert data are often very granular, down to the second or millisecond, meaning there could be many unique timestamps.

Instead, it may be more beneficial to aggregate the data by broader time intervals, such as by hour or day, in combination with other variables like location or alert type, to observe patterns or trends over time.

b.

```
import pandas as pd
import os

data_df = pd.read_csv('waze_data.csv', parse_dates=['ts'])

if 'geo' in data_df.columns:
    import re
    def extract_coordinates(geo_string):
        if pd.notna(geo_string):
            match = re.match(r"POINT\s*\(([-\d.]+)\s+([-\d.]+)\)",
↪   geo_string.strip())
            if match:
                return pd.Series([float(match.group(2)),
                 ↪   float(match.group(1))])
        return pd.Series([None, None])

    data_df[['latitude', 'longitude']] =
↪   data_df['geo'].apply(extract_coordinates)

data_df['binned_latitude'] = data_df['latitude'].round(2)
data_df['binned_longitude'] = data_df['longitude'].round(2)
```

```
data_df['hour'] = data_df['ts'].dt.floor('H')

collapsed_df = (
    data_df.groupby(['hour', 'binned_latitude', 'binned_longitude', 'type',
 ↪  'subtype'])
    .size()
    .reset_index(name='alert_count')
)

output_folder = 'top_alerts_map_byhour'
os.makedirs(output_folder, exist_ok=True)
collapsed_file_path = os.path.join(output_folder,
 ↪  'top_alerts_map_byhour.csv')

collapsed_df.to_csv(collapsed_file_path, index=False)

print(f"The collapsed dataset has {collapsed_df.shape[0]} rows and is saved
 ↪  to {collapsed_file_path}.")
```

```
The collapsed dataset has 510302 rows and is saved to
top_alerts_map_byhour/top_alerts_map_byhour.csv.
```

c.

```
collapsed_file_path = './top_alerts_map_byhour/top_alerts_map_byhour.csv'
geojson_filepath = './chicago_neighborhoods.geojson'

collapsed_df = pd.read_csv(collapsed_file_path)

collapsed_df['hour'] =
 ↪  pd.to_datetime(collapsed_df['hour']).dt.tz_localize(None).dt.floor('H')

jam_heavy_traffic = collapsed_df[
    (collapsed_df['type'] == 'JAM') & (collapsed_df['subtype'] ==
 ↪  'JAM_HEAVY_TRAFFIC')
]

print(jam_heavy_traffic['hour'].unique())

if jam_heavy_traffic.empty:
    print("No data found for 'Jam - Heavy Traffic'. Please verify the
     ↪  dataset.")
```

```
else:
    print(f"Filtered data contains {len(jam_heavy_traffic)} rows.")
```

```
<DatetimeArray>
['2024-01-02 10:00:00', '2024-01-02 11:00:00', '2024-01-02 12:00:00',
 '2024-01-02 13:00:00', '2024-01-02 14:00:00', '2024-01-02 15:00:00',
 '2024-01-02 16:00:00', '2024-01-02 17:00:00', '2024-01-02 18:00:00',
 '2024-01-02 19:00:00',
 ...
 '2024-10-12 14:00:00', '2024-10-12 15:00:00', '2024-10-12 16:00:00',
 '2024-10-12 17:00:00', '2024-10-12 18:00:00', '2024-10-12 19:00:00',
 '2024-10-12 20:00:00', '2024-10-12 21:00:00', '2024-10-12 22:00:00',
 '2024-10-12 23:00:00']
Length: 5273, dtype: datetime64[ns]
Filtered data contains 130579 rows.
```

```
import altair as alt

selected_hours = [
    '2024-01-02 10:00:00',
    '2024-01-02 14:00:00',
    '2024-01-02 18:00:00'
]
selected_hours_dt = pd.to_datetime(selected_hours)

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

geo_data = alt.Data(values=chicago_geojson["features"])
chicago_map = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).properties(
    width=600,
    height=400
).project('mercator')

plots = []
for hour in selected_hours_dt:
    filtered_df = jam_heavy_traffic[jam_heavy_traffic['hour'] == hour]

    if filtered_df.empty:
```

```
        print(f"No data available for 'Jam - Heavy Traffic' at {hour}.
         ↪  Skipping this hour.")
        continue
    else:
        print(f"Data for {hour}: {len(filtered_df)} rows found.")

    top_10_df = filtered_df.nlargest(10, 'alert_count')

    scatter_plot = alt.Chart(top_10_df).mark_circle().encode(
        longitude='binned_longitude:Q',
        latitude='binned_latitude:Q',
        size=alt.Size('alert_count:Q', title='Number of Alerts',
 ↪  scale=alt.Scale(range=[100, 1000])),
        color=alt.value("red"),
        tooltip=['binned_latitude', 'binned_longitude', 'alert_count']
    ).properties(
        title=f"Top 10 Locations for 'Jam - Heavy Traffic' at
 ↪  {hour.strftime('%Y-%m-%d %H:%M')}",
        width=600,
        height=400
    )

    plots.append(chicago_map + scatter_plot)


if plots:
    final_chart = alt.vconcat(*plots)

    final_chart = final_chart.configure_view(
        strokeWidth=0
    ).configure_title(
        fontSize=16
    )

final_chart
```

```
Data for 2024-01-02 10:00:00: 2 rows found.
Data for 2024-01-02 14:00:00: 10 rows found.
Data for 2024-01-02 18:00:00: 2 rows found.
```

```
alt.VConcatChart(...)
```

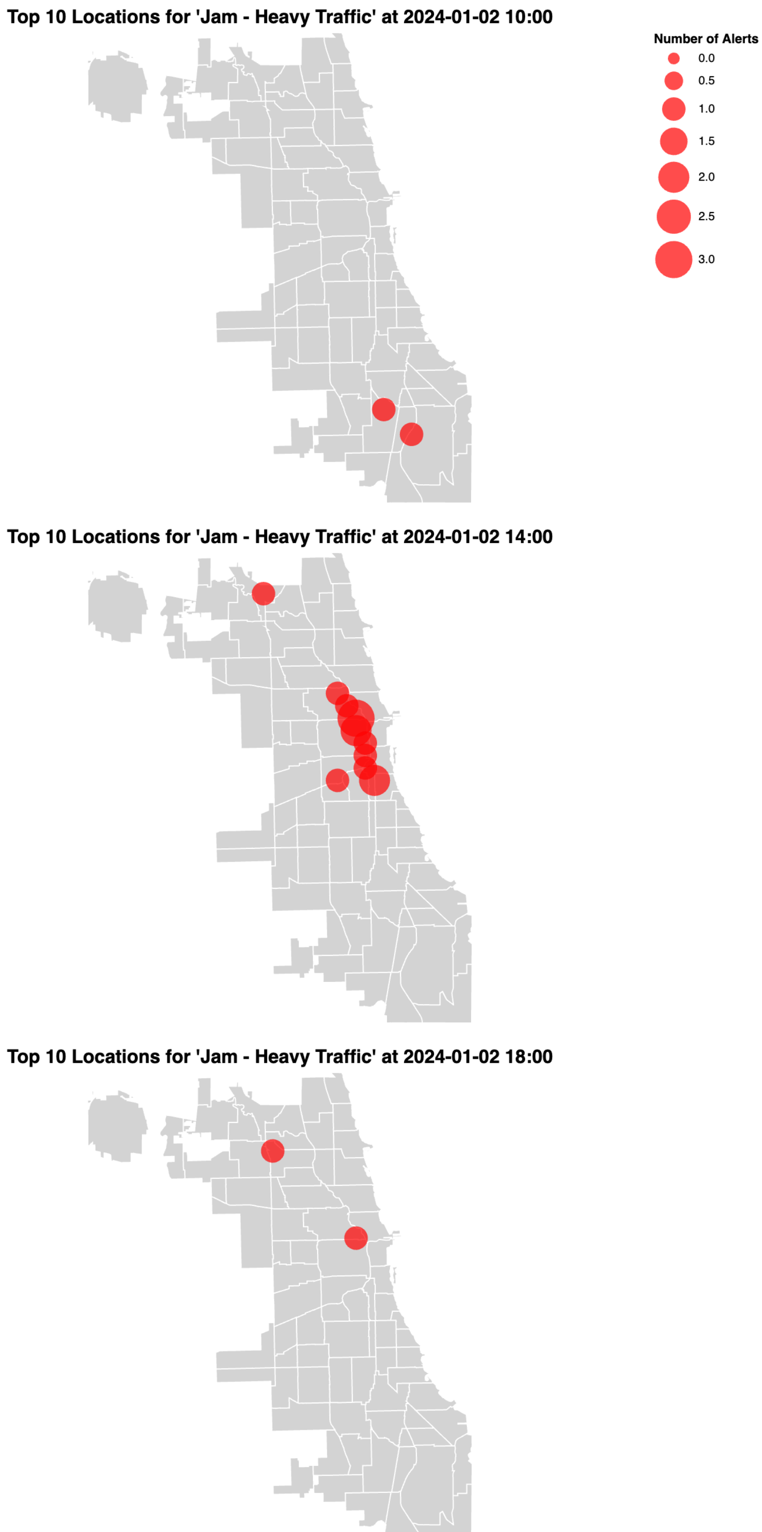


Figure 4: Top 10 locations by three times

2.

a.

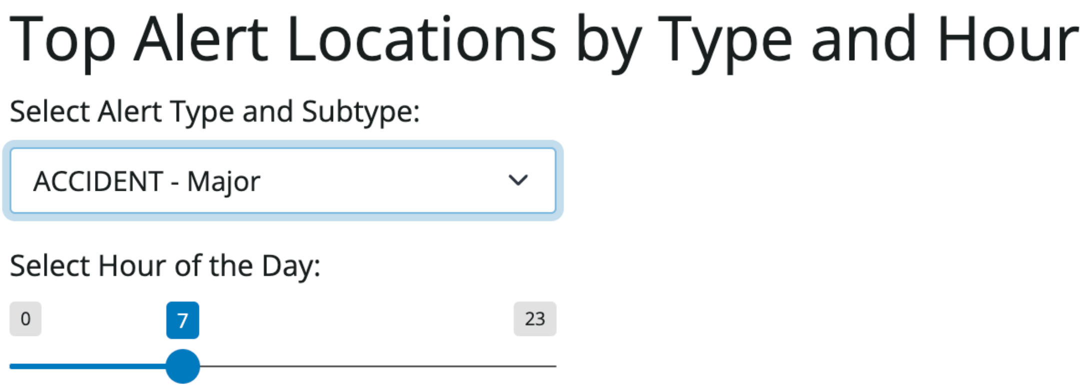


Figure 5: Dropdown and Slider

b.

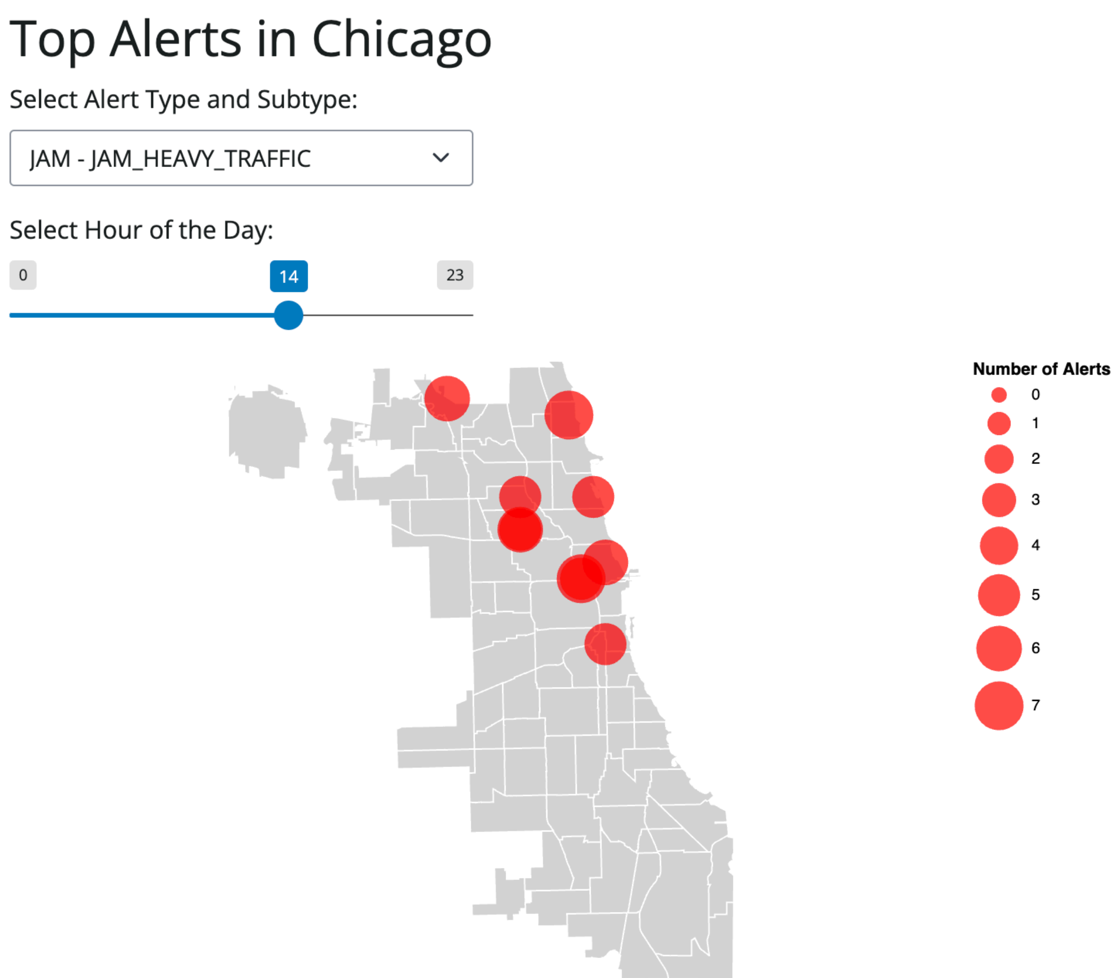
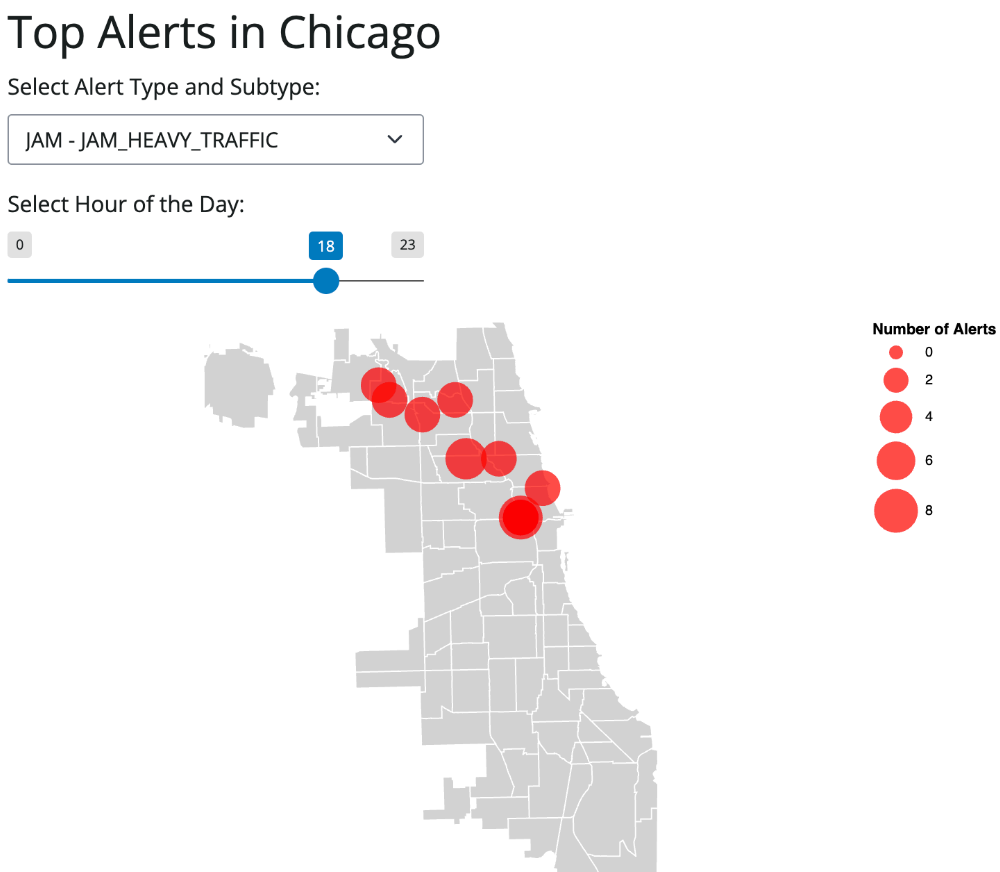


Figure 6: Plot1



c.

According to these two graphs: Morning (4 AM): There appears to be minimal road construction activity, as indicated by fewer alerts concentrated in one location.

Night (8 PM): There is significantly more activity with multiple alerts across various locations, suggesting that road construction is more prevalent during nighttime hours.

Thus, it seems that road construction is done more during nighttime hours compared to the morning.

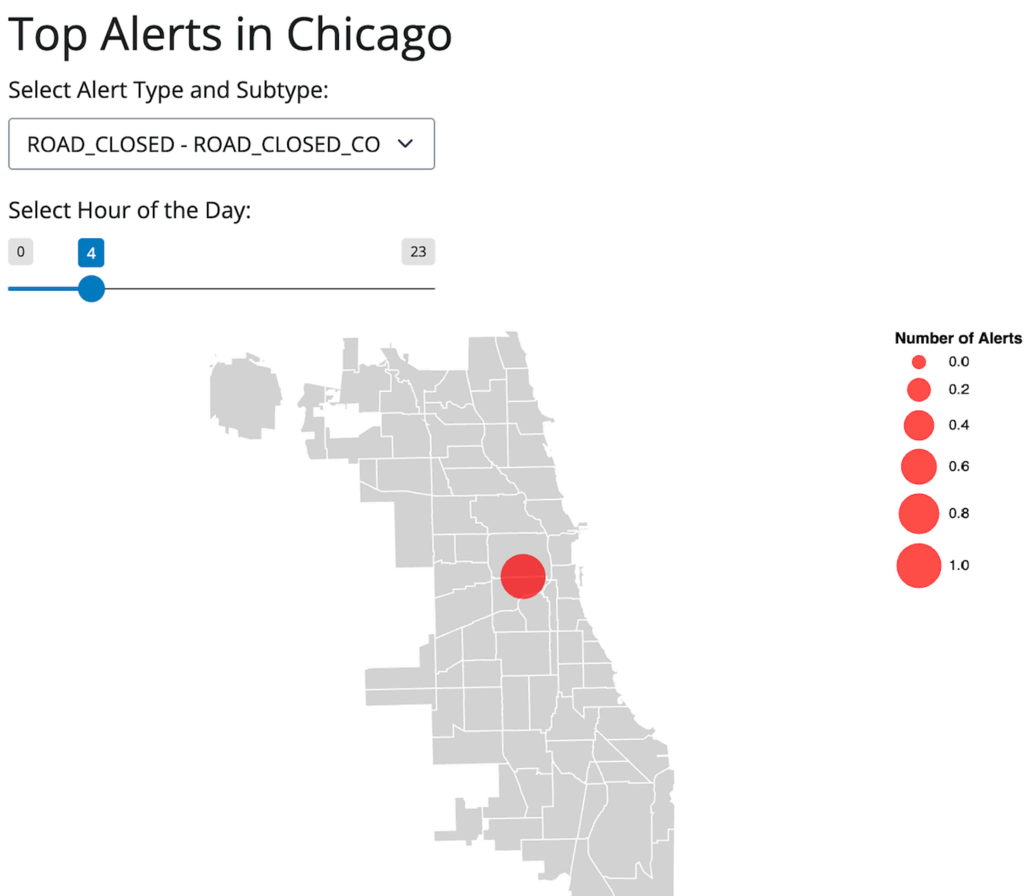


Figure 7: Morning Plot

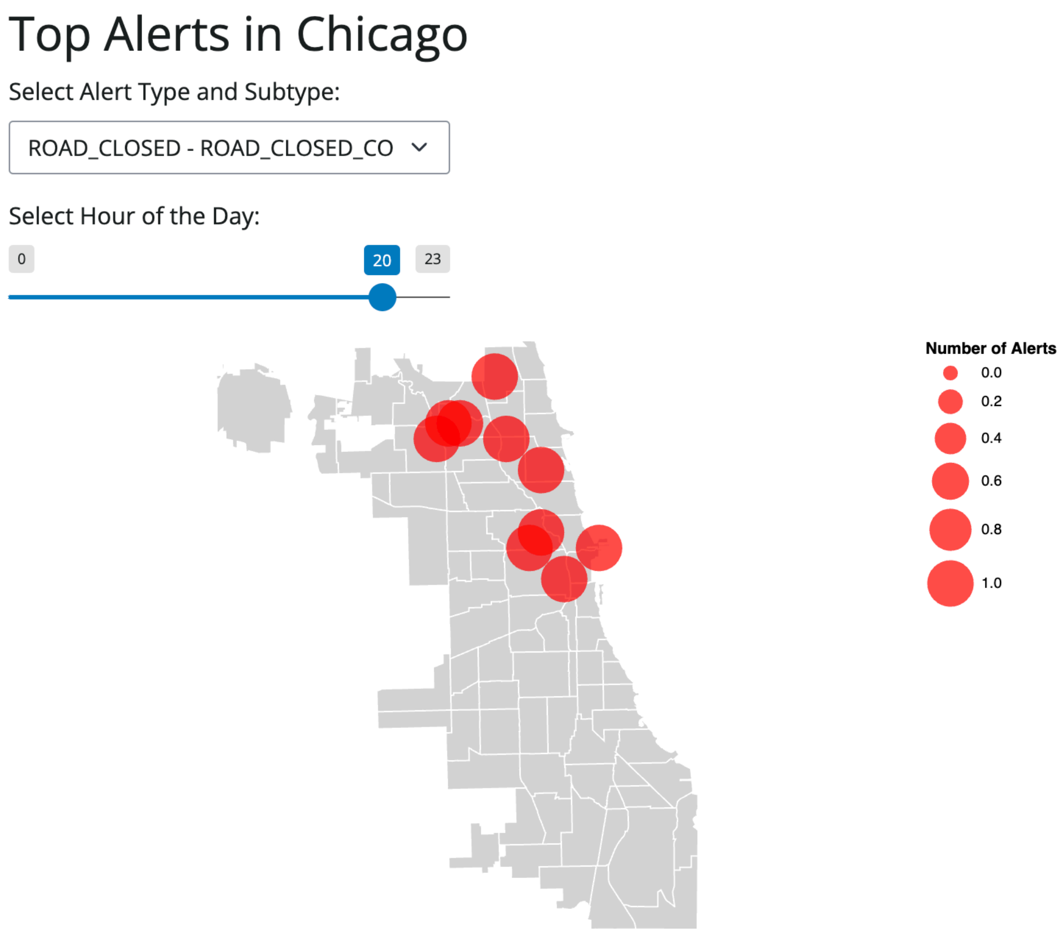


Figure 8: Evening Plot

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

1.

a.

When considering whether to collapse the dataset by a range of hours for the new Shiny app, it would not be the most efficient or practical approach. The goal of this app is to allow users to select an arbitrary range of hours dynamically (e.g., 6 AM–10 AM or 8 PM–2 AM). Pre-collapsing the dataset into predefined ranges would limit the flexibility of the app.

Users may choose ranges that were not pre-collapsed, and trying to account for every possible combination of hours would result in a massive and unwieldy dataset, making it harder to manage and query.

Additionally, collapsing the dataset ahead of time could lead to inefficiencies. Storing data pre-aggregated for all possible hour ranges would consume significant memory and computational resources. By contrast, dynamically filtering the original dataset during runtime based on the selected range of hours would allow for greater adaptability without requiring excessive pre-processing or storage. This dynamic filtering approach also avoids the need to repeatedly pre-compute new datasets for every potential range, making it both practical and resource-efficient.

Another key consideration is that pre-collapsing data by ranges of hours could lead to a loss of granularity. For example, alerts that occur at the edges of a range might not be as visible if they are averaged over a broader span of time. By keeping the data uncollapsed, the app can more accurately reflect the specific distribution and density of alerts within the chosen range, ensuring a more detailed and precise visualization.

Thus, it is better to dynamically filter the dataset based on the user-selected range of hours rather than collapsing it ahead of time. This approach maintains data granularity, provides greater flexibility for arbitrary ranges, and reduces storage requirements. The Shiny app should dynamically aggregate the data at runtime, ensuring efficiency and adaptability without sacrificing usability.

b.

```
collapsed_df = pd.read_csv(collapsed_file_path)

collapsed_df['hour'] =
↪ pd.to_datetime(collapsed_df['hour']).dt.tz_localize(None).dt.floor('H')

jam_heavy_traffic = collapsed_df[
    (collapsed_df['type'] == 'JAM') & (collapsed_df['subtype'] ==
↪ 'JAM_HEAVY_TRAFFIC')
]

if jam_heavy_traffic.empty:
    print("No data found for 'Jam - Heavy Traffic'. Please verify the
     ↪ dataset.")
else:
    print(f"Filtered data contains {len(jam_heavy_traffic)} rows.")

start_hour = pd.to_datetime('2024-01-02 06:00:00')
end_hour = pd.to_datetime('2024-01-02 09:00:00')
```

```
filtered_df = jam_heavy_traffic[
    (jam_heavy_traffic['hour'] >= start_hour) & (jam_heavy_traffic['hour'] <
 ↪  end_hour)
]

if filtered_df.empty:
    print(f"No data available for 'Jam - Heavy Traffic' between {start_hour}
     ↪  and {end_hour}.")
else:
    print(f"Filtered data contains {len(filtered_df)} rows for the specified
     ↪  range.")

top_10_df = (
    filtered_df.groupby(['binned_latitude', 'binned_longitude'])
    .agg({'alert_count': 'sum'})
    .reset_index()
    .nlargest(10, 'alert_count')
)

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

geo_data = alt.Data(values=chicago_geojson["features"])
chicago_map = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).properties(
    width=600,
    height=400
).project('mercator')

scatter_plot = alt.Chart(top_10_df).mark_circle().encode(
    longitude='binned_longitude:Q',
    latitude='binned_latitude:Q',
    size=alt.Size('alert_count:Q', title='Number of Alerts',
 ↪  scale=alt.Scale(range=[100, 1000])),
    color=alt.value("red"),
    tooltip=['binned_latitude', 'binned_longitude', 'alert_count']
).properties(
    title="Top 10 Locations for 'Jam - Heavy Traffic' (6AM - 9AM)",
    width=600,
```

```
    height=400
)

combined_chart = (chicago_map + scatter_plot).configure_view(
    strokeWidth=0
)

combined_chart
```

```
Filtered data contains 130579 rows.
No data available for 'Jam - Heavy Traffic' between 2024-01-02 06:00:00 and
2024-01-02 09:00:00.
```

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

2.

a.
```
from shiny import App, ui, render, reactive
import pandas as pd
import altair as alt
import json

collapsed_file_path =
"/Users/mengyuting/Documents/GitHub/ps6/top_alerts_map_byhour/top_alerts_map_byhour.csv
geojson_filepath =
"/Users/mengyuting/Documents/GitHub/ps6/chicago_neighborhoods.geojson"

collapsed_df = pd.read_csv(collapsed_file_path)
collapsed_df['hour'] =
pd.to_datetime(collapsed_df['hour']).dt.tz_localize(None).dt.floor('H')

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

app_ui = ui.page_sidebar(
title="Top Alerts by Range of Hours",
sidebar=ui.panel_sidebar(
ui.h2("Filter Options"),
ui.input_select(
    id="alert_type",
    label="Select Alert Type and Subtype:",
    choices={
```

```
        f"{row['type']} - {row['subtype']}":
        f"{row['type']}|{row['subtype']}"
        for _, row in collapsed_df[['type',
        'subtype']].drop_duplicates().iterrows()
    },
    selected="JAM|JAM_HEAVY_TRAFFIC"
),
ui.input_slider(
    id="hour_range",
    label="Select Hour Range:",
    min=0,
    max=23,
    value=[6, 9]
)
    ),
 main=ui.panel_main(
        ui.output_plot("alert_plot")
        )
    )


def server(input, output, session):
@reactive.Calc
def filtered_data():
selected_type, selected_subtype = input.alert_type().split("|")
filtered = collapsed_df[
    (collapsed_df['type'] == selected_type) &
    (collapsed_df['subtype'] == selected_subtype) &
    (collapsed_df['hour'].dt.hour >= input.hour_range()[0]) &
    (collapsed_df['hour'].dt.hour < input.hour_range()[1])
]
if filtered.empty:
    return pd.DataFrame(columns=['binned_latitude', 'binned_longitude',
    'alert_count'])
return (
    filtered.groupby(['binned_latitude', 'binned_longitude'])
    .agg({'alert_count': 'sum'})
    .reset_index()
    .nlargest(10, 'alert_count')
)

@output
@render.plot
```

```
def alert_plot():
top_10_df = filtered_data()
if top_10_df.empty:
    return alt.Chart(pd.DataFrame({'message': ['No data
    available']})).mark_text().encode(
        text='message'
    ).properties(
        width=600,
        height=400
    )
geo_data = alt.Data(values=chicago_geojson["features"])
chicago_map = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).properties(
    width=600,
    height=400
).project('mercator')
scatter_plot = alt.Chart(top_10_df).mark_circle().encode(
    longitude='binned_longitude:Q',
    latitude='binned_latitude:Q',
    size=alt.Size('alert_count:Q', title='Number of Alerts',
    scale=alt.Scale(range=[100, 1000])),
    color=alt.value("red"),
    tooltip=['binned_latitude', 'binned_longitude', 'alert_count']
).properties(
    title="Top 10 Locations for Selected Alerts and Time Range",
    width=600,
    height=400
)
return chicago_map + scatter_plot

app = App(app_ui, server)

if __name__ == "__main__":
        app.run()
```

b.
```
from shiny import App, ui, render, reactive
import pandas as pd
import altair as alt
import json
```

```
collapsed_file_path =
"/Users/mengyuting/Documents/GitHub/ps6/top_alerts_map_byhour/top_alerts_map_byhour.csv
geojson_filepath =
"/Users/mengyuting/Documents/GitHub/ps6/chicago_neighborhoods.geojson"

collapsed_df = pd.read_csv(collapsed_file_path)
collapsed_df['hour'] =
pd.to_datetime(collapsed_df['hour']).dt.tz_localize(None).dt.floor('H')

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


app_ui = ui.page_sidebar(
 title="Top Alerts by Range of Hours",
    sidebar=ui.panel_sidebar(
    ui.h2("Filter Options"),
     ui.input_select(
    id="alert_type",
    label="Select Alert Type and Subtype:",
    choices={
        f"{row['type']} - {row['subtype']}":
        f"{row['type']}|{row['subtype']}"
        for _, row in collapsed_df[['type',
        'subtype']].drop_duplicates().iterrows()
    },
    selected="JAM|JAM_HEAVY_TRAFFIC"
),
ui.input_slider(
    id="hour_range",
    label="Select Hour Range:",
    min=0,
    max=23,
    value=[6, 9]
)
),
    main=ui.panel_main(
    ui.output_plot("alert_plot")
 )
    )


def server(input, output, session):
```

```
    @reactive.Calc
def filtered_data():
selected_type, selected_subtype = input.alert_type().split("|")
filtered = collapsed_df[
    (collapsed_df['type'] == selected_type) &
    (collapsed_df['subtype'] == selected_subtype) &
    (collapsed_df['hour'].dt.hour >= input.hour_range()[0]) &
    (collapsed_df['hour'].dt.hour < input.hour_range()[1])
]
if filtered.empty:
    return pd.DataFrame(columns=['binned_latitude', 'binned_longitude',
    'alert_count'])
return (
    filtered.groupby(['binned_latitude', 'binned_longitude'])
    .agg({'alert_count': 'sum'})
    .reset_index()
    .nlargest(10, 'alert_count')
)

@output
@render.plot
def alert_plot():
top_10_df = filtered_data()
if top_10_df.empty:
    return alt.Chart(pd.DataFrame({'message': ['No data
    available']})).mark_text().encode(
        text='message'
    ).properties(
        width=600,
        height=400
    )
geo_data = alt.Data(values=chicago_geojson["features"])
chicago_map = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).properties(
    width=600,
    height=400
).project('mercator')
scatter_plot = alt.Chart(top_10_df).mark_circle().encode(
    longitude='binned_longitude:Q',
    latitude='binned_latitude:Q',
```

```
        size=alt.Size('alert_count:Q', title='Number of Alerts',
        scale=alt.Scale(range=[100, 1000])),
        color=alt.value("red"),
        tooltip=['binned_latitude', 'binned_longitude', 'alert_count']
    ).properties(
        title="Top 10 Locations for Selected Alerts and Time Range",
        width=600,
        height=400
    )
    return chicago_map + scatter_plot

    app = App(app_ui, server)

    if __name__ == "__main__":
        app.run()
```

3.

a.

```
from shiny import App, ui, render, reactive
import pandas as pd
import altair as alt
import json


collapsed_file_path =
"/Users/mengyuting/Documents/GitHub/ps6/top_alerts_map_byhour/top_alerts_map_byhour.csv
geojson_filepath =
"/Users/mengyuting/Documents/GitHub/ps6/chicago_neighborhoods.geojson"

collapsed_df = pd.read_csv(collapsed_file_path)
collapsed_df['hour'] =
pd.to_datetime(collapsed_df['hour']).dt.tz_localize(None).dt.floor('H')

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

app_ui = ui.page_sidebar(
title="Top Alerts by Hour",
sidebar=ui.panel_sidebar(
ui.h2("Filter Options"),
ui.input_switch(
    id="switch_button",
    label="Toggle to switch to range of hours",
```

```
    value=True  # Default to range slider
),
ui.input_select(
    id="alert_type",
    label="Select Alert Type and Subtype:",
    choices={
        f"{row['type']} - {row['subtype']}":
        f"{row['type']}|{row['subtype']}"
        for _, row in collapsed_df[['type',
        'subtype']].drop_duplicates().iterrows()
    },
    selected="JAM|JAM_HEAVY_TRAFFIC"
    ),
     ui.output_ui("dynamic_slider")
    ),
    main=ui.panel_main(
    ui.output_plot("alert_plot")
        )
        )

def server(input, output, session):
    # Dynamic UI for slider
    @output
    @render.ui
     def dynamic_slider():
    if input.switch_button():
         # Range slider
        return ui.input_slider(
        id="hour_range",
        label="Select Hour Range:",
        min=0,
        max=23,
        value=[6, 9]
    )
else:
    # Single hour slider
    return ui.input_slider(
        id="single_hour",
        label="Select Single Hour:",
        min=0,
        max=23,
        value=6
    )
```

```
    app = App(app_ui, server)

        if __name__ == "__main__":
            app.run()
```

b.
```
from shiny import App, ui, render, reactive
import pandas as pd
import altair as alt
import json

collapsed_file_path =
"/Users/mengyuting/Documents/GitHub/ps6/top_alerts_map_byhour/top_alerts_map_byhour.csv
geojson_filepath =
"/Users/mengyuting/Documents/GitHub/ps6/chicago_neighborhoods.geojson"

collapsed_df = pd.read_csv(collapsed_file_path)
collapsed_df['hour'] =
pd.to_datetime(collapsed_df['hour']).dt.tz_localize(None).dt.floor('H')

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

app_ui = ui.page_sidebar(
    title="Top Alerts by Hour",
     sidebar=ui.panel_sidebar(
        ui.h2("Filter Options"),
        ui.input_switch(
    id="switch_button",
    label="Toggle to switch to range of hours",
    value=True  # Default to range slider
),
ui.input_select(
    id="alert_type",
    label="Select Alert Type and Subtype:",
    choices={
        f"{row['type']} - {row['subtype']}":
        f"{row['type']}|{row['subtype']}"
        for _, row in collapsed_df[['type',
        'subtype']].drop_duplicates().iterrows()
    },
    selected="JAM|JAM_HEAVY_TRAFFIC"
),
```

```
# Conditional UI elements
ui.output_ui("dynamic_slider")
),
main=ui.panel_main(
    ui.output_plot("alert_plot")
    )
    )

def server(input, output, session):
# Dynamic UI for slider
@output
@render.ui
def dynamic_slider():
if input.switch_button():
    # Range slider when toggle is ON
    return ui.input_slider(
        id="hour_range",
        label="Select Hour Range:",
        min=0,
        max=23,
        value=[6, 9]
    )
else:
    # Single hour slider when toggle is OFF
    return ui.input_slider(
        id="single_hour",
        label="Select Single Hour:",
        min=0,
        max=23,
        value=6
    )

app = App(app_ui, server)

    if __name__ == "__main__":
        app.run()
```

c.
```
from shiny import App, ui, render, reactive
import pandas as pd
import altair as alt
import json
```

```
collapsed_file_path = "/Users/mengyuting/Documents/GitHub/ps6/
top_alerts_map_byhour/top_alerts_map_byhour.csv"
geojson_filepath =
"/Users/mengyuting/Documents/GitHub/ps6/chicago_neighborhoods.geojson"

collapsed_df = pd.read_csv(collapsed_file_path)
collapsed_df['hour'] =
pd.to_datetime(collapsed_df['hour']).dt.tz_localize(None).dt.floor('H')

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

 app_ui = ui.page_sidebar(
    title="Top Alerts by Hour",
    sidebar=ui.panel_sidebar(
    ui.h2("Filter Options"),
    ui.input_switch(
    id="switch_button",
    label="Toggle to switch to range of hours",
    value=True  # Default to range slider
),
ui.input_select(
    id="alert_type",
    label="Select Alert Type and Subtype:",
    choices={
        f"{row['type']} - {row['subtype']}":
        f"{row['type']}|{row['subtype']}"
        for _, row in collapsed_df[['type',
        'subtype']].drop_duplicates().iterrows()
    },
    selected="JAM|JAM_HEAVY_TRAFFIC"
),
# Conditional UI elements
ui.output_ui("dynamic_slider")
),
main=ui.panel_main(
ui.output_plot("alert_plot")
)
 )

def server(input, output, session):
# Dynamic UI for slider
@output
```

```
 @render.ui
def dynamic_slider():
    if input.switch_button():
    # Range slider when toggle is ON
    return ui.input_slider(
        id="hour_range",
        label="Select Hour Range:",
        min=0,
        max=23,
        value=[6, 9]
    )
else:
    # Single hour slider when toggle is OFF
    return ui.input_slider(
        id="single_hour",
        label="Select Single Hour:",
        min=0,
        max=23,
        value=6
    )

    # Filter data based on input
    @reactive.Calc
    def filtered_data():
# Extract selected type and subtype
selected_type, selected_subtype = input.alert_type().split("|")

if input.switch_button():
    # If range slider is active
    filtered = collapsed_df[
        (collapsed_df['type'] == selected_type) &
        (collapsed_df['subtype'] == selected_subtype) &
        (collapsed_df['hour'].dt.hour >= input.hour_range()[0]) &
        (collapsed_df['hour'].dt.hour < input.hour_range()[1])
    ]
else:
    # If single-hour slider is active
    filtered = collapsed_df[
        (collapsed_df['type'] == selected_type) &
        (collapsed_df['subtype'] == selected_subtype) &
        (collapsed_df['hour'].dt.hour == input.single_hour())
    ]
```

```
if filtered.empty:
    return pd.DataFrame(columns=['binned_latitude', 'binned_longitude',
    'alert_count'])

# Aggregate the data and find top 10 locations
top_10 = (
    filtered.groupby(['binned_latitude', 'binned_longitude'])
    .agg({'alert_count': 'sum'})
    .reset_index()
    .nlargest(10, 'alert_count')
)
return top_10
```

Generate the plot based on filtered data @output @render.plot def alert_plot(): top_10_df = filtered_data() if top_10_df.empty: return alt.Chart(pd.DataFrame({'message': ['No data available']})).mark_text().encode(text='message').properties(width=600, height=400)

```
# Prepare the map layer
geo_data = alt.Data(values=chicago_geojson["features"])
chicago_map = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).properties(
    width=600,
    height=400
).project('mercator')

# Create the scatter plot
scatter_plot = alt.Chart(top_10_df).mark_circle().encode(
    longitude='binned_longitude:Q',
    latitude='binned_latitude:Q',
    size=alt.Size('alert_count:Q', title='Number of Alerts',
    scale=alt.Scale(range=[100, 1000])),
    color=alt.value("red"),
    tooltip=['binned_latitude', 'binned_longitude', 'alert_count']
).properties(
    title="Top 10 Locations for Selected Alerts and Time Range",
    width=600,
    height=400
)

return chicago_map + scatter_plot
```

app = App(app_ui, server)

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

d.

Categorizing Time Periods: Add a new time_period column to the dataset (e.g., "Morning" and "Afternoon").

Updating the Visualization: Incorporate time_period as a color-encoded variable in the scatter plot to distinguish between periods.

Enhancing the Legend: Adjust the size and color legends to clearly indicate the number of alerts and time periods.

Optional Interactivity: Allow filtering by time_period for additional user control.