Problem Set 6 - Waze Shiny Dashboard

AUTHOR
Peter Ganong, Maggie Shi, and Andre Oviedo

PUBLISHED
November 19, 2024

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: **P.C.**
2. "I have uploaded the names of anyone I worked with on the problem set **here**"
   I did not work with anyone on this problem set.
3. Late coins used this pset: **0**
   Late coins left after submission: **4**
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 corresponding section for the code style rubric.

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

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

```
def print_file_contents(file_path):
    """Print contents of a file."""
    try:
        with open(file_path, 'r') as f:
            content = f.read()
            print("```python")
            print(content)
            print("```")
```

```
    except FileNotFoundError:
        print("```python")
        print(f"Error: File '{file_path}' not found")
        print("```")
    except Exception as e:
        print("```python")
        print(f"Error reading file: {e}")
        print("```")

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

Background

Data Download and Exploration (20 points)

1.

```
import pandas as pd

file_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\waze_data\waze_data_sample.csv"

# Read CSV with the first column as the index
waze_data_sample = pd.read_csv(file_path, index_col=0)

# List the variable names and data types (ignoring ts, geo, and geoWKT columns)
excluded_columns = ['ts', 'geo', 'geoWKT']
variables = {col: waze_data_sample[col].dtype for col in waze_data_sample.columns if col not i

# Map pandas dtypes to Altair data types, with a special case for roadType
altair_types = {
    'int64': 'Quantitative',
    'float64': 'Quantitative',
    'object': 'Nominal',
    'datetime64[ns]': 'Temporal',
    'bool': 'Nominal'
}

# Adjust Altair classification for specific variables
# Assuming roadType is an integer but should be classified as Nominal
adjusted_types = {
    'roadType': 'Nominal',  # Override to classify roadType as Nominal
}

# Convert data types to Altair types, applying adjustments where needed
variables_with_altair_types = {
    col: adjusted_types.get(col, altair_types.get(str(dtype), 'Unknown'))
    for col, dtype in variables.items()
}

# Display the result
```

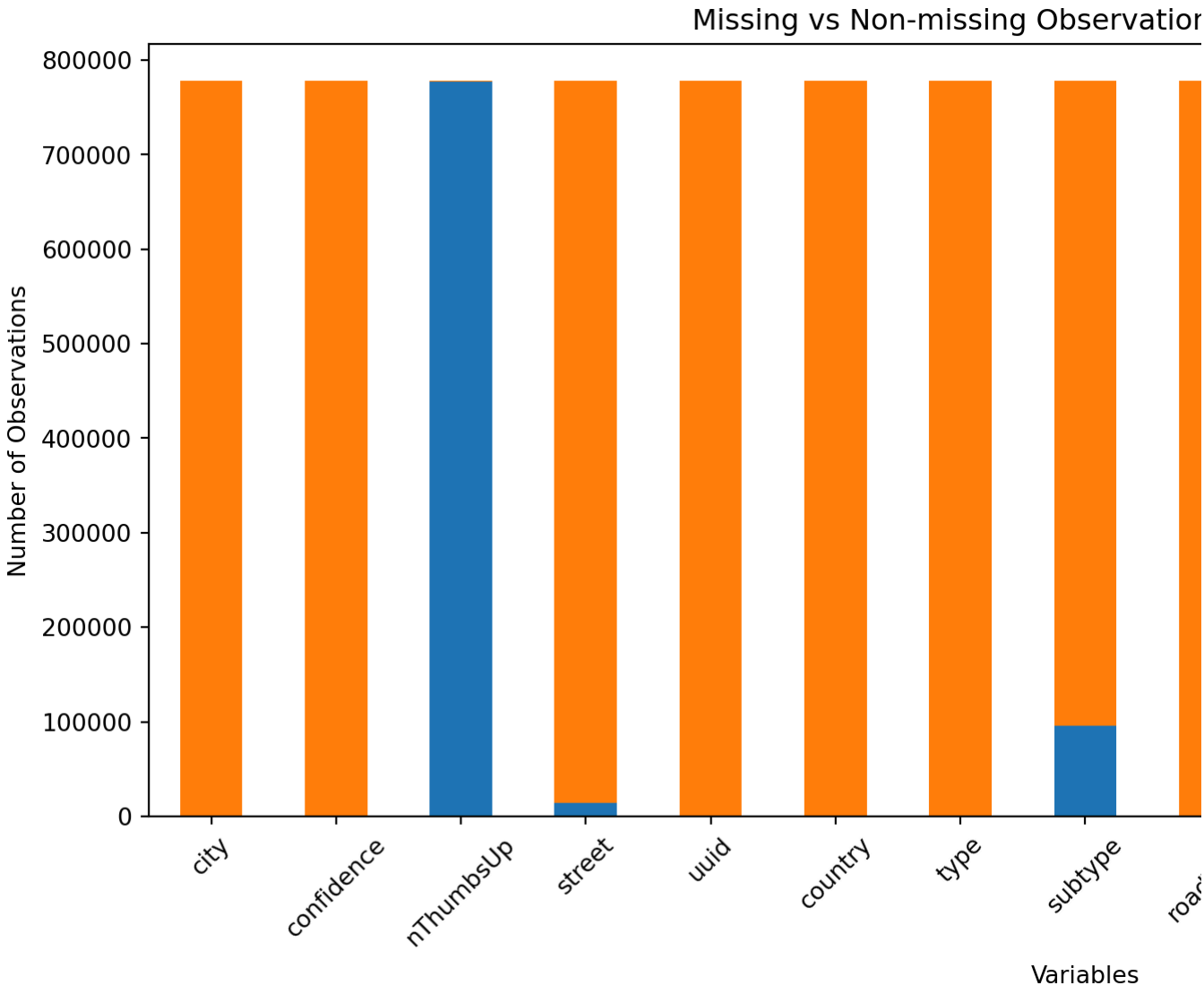
```
for col, dtype in variables_with_altair_types.items():
    print(f"{col}: {dtype}")
```

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

2.

```
import pandas as pd
import matplotlib.pyplot as plt

# Load the CSV file
file_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\waze_data\waze_data.csv"
waze_data = pd.read_csv(file_path)

# Calculate missing and non-missing counts for each column
missing_counts = waze_data.isnull().sum()
non_missing_counts = waze_data.notnull().sum()

# Create a DataFrame for plotting
missing_data = pd.DataFrame({
    'Missing': missing_counts,
    'Non-missing': non_missing_counts
})

# Plot a stacked bar chart
missing_data.plot(kind='bar', stacked=True, figsize=(12, 6))
plt.title("Missing vs Non-missing Observations for Each Variable")
plt.xlabel("Variables")
plt.ylabel("Number of Observations")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Identify variables with missing values and the variable with the highest share of missing va
missing_vars = missing_counts[missing_counts > 0]
most_missing_var = missing_vars.idxmax()
highest_missing_share = missing_vars.max()

print("Variables with NULL values:")
```

```
print(missing_vars)
print(f"\nVariable with the highest share of missing observations: {most_missing_var} with {hi
```




```
Variables with NULL values:
nThumbsUp    776723
street        14073
subtype       96086
dtype: int64

Variable with the highest share of missing observations: nThumbsUp with 776723 missing values.
```

3.

```
# 1. Print unique values for `type` and `subtype`, and count types with `NA` subtypes
import pandas as pd

# Load the CSV file
file_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\waze_data\waze_data.csv"
waze_data = pd.read_csv(file_path)
```

```
# Define readable names for `type` and `subtype`
type_clean_map = {
    'JAM': 'Traffic Jam',
    'ACCIDENT': 'Accident',
    'ROAD_CLOSED': 'Road Closure',
    'HAZARD': 'Hazard'
}

subtype_clean_map = {
    'ACCIDENT_MAJOR': 'Major Accident',
    'ACCIDENT_MINOR': 'Minor Accident',
    'HAZARD_ON_ROAD': 'Road Hazard',
    'HAZARD_ON_ROAD_CAR_STOPPED': 'Stopped Car on Road',
    'HAZARD_ON_ROAD_CONSTRUCTION': 'Construction Hazard',
    'HAZARD_ON_ROAD_EMERGENCY_VEHICLE': 'Emergency Vehicle Hazard',
    'HAZARD_ON_ROAD_ICE': 'Icy Road Hazard',
    'HAZARD_ON_ROAD_OBJECT': 'Object on Road',
    'HAZARD_ON_ROAD_POT_HOLE': 'Pothole Hazard',
    'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT': 'Traffic Light Fault',
    'HAZARD_ON_SHOULDER': 'Shoulder Hazard',
    'HAZARD_ON_SHOULDER_CAR_STOPPED': 'Stopped Car on Shoulder',
    'HAZARD_WEATHER': 'Weather Hazard',
    'HAZARD_WEATHER_FLOOD': 'Flood Hazard',
    'JAM_HEAVY_TRAFFIC': 'Heavy Traffic',
    'JAM_MODERATE_TRAFFIC': 'Moderate Traffic',
    'JAM_STAND_STILL_TRAFFIC': 'Standstill Traffic',
    'ROAD_CLOSED_EVENT': 'Event Closure',
    'HAZARD_ON_ROAD_LANE_CLOSED': 'Lane Closure Hazard',
    'HAZARD_WEATHER_FOG': 'Fog Hazard',
    'ROAD_CLOSED_CONSTRUCTION': 'Construction Closure',
    'HAZARD_ON_ROAD_ROAD_KILL': 'Animal on Road',
    'HAZARD_ON_SHOULDER_ANIMALS': 'Animals on Shoulder',
    'HAZARD_ON_SHOULDER_MISSING_SIGN': 'Missing Sign on Shoulder',
    'JAM_LIGHT_TRAFFIC': 'Light Traffic',
    'HAZARD_WEATHER_HEAVY_SNOW': 'Heavy Snow Hazard',
    'ROAD_CLOSED_HAZARD': 'Hazard Closure',
    'HAZARD_WEATHER_HAIL': 'Hail Hazard'
}

# 1. Print unique values for `type` and `subtype`, and count types with `NA` subtypes
unique_types = waze_data['type'].unique()
unique_subtypes = waze_data['subtype'].unique()
type_with_na_subtype = waze_data[waze_data['subtype'].isna()]['type'].unique()
num_type_with_na_subtype = len(type_with_na_subtype)

print("Unique values in 'type':")
for t in unique_types:
    print(f"- {type_clean_map.get(t, t)}")

print("\nUnique values in 'subtype':")
for s in unique_subtypes:
    cleaned_subtype = subtype_clean_map.get(s, "Unclassified" if pd.isna(s) else s.replace("_"
    print(f"- {cleaned_subtype}")
```

```
print(f"\nNumber of types with a NA subtype: {num_type_with_na_subtype}")
print("Types with NA subtypes:")
for t in type_with_na_subtype:
    print(f"- {type_clean_map.get(t, t)}")


# 2. Generate a readable hierarchical bullet list
print("\nHierarchical Bullet List:")
for t in unique_types:
    # Get readable `type` name
    cleaned_type = type_clean_map.get(t, t)
    print(f"- {cleaned_type}")

    # Get and print readable subtypes for each type
    subtypes = waze_data[waze_data['type'] == t]['subtype'].unique()
    for s in subtypes:
        cleaned_subtype = subtype_clean_map.get(s, "Unclassified" if pd.isna(s) else s.replace
        print(f"  - {cleaned_subtype}")

# 3. Discuss handling of NA subtypes
print("\nHandling of NA subtypes:")
print("If we decide to keep NA subtypes, we will label them as 'Unclassified' to indicate unsp
```

```
Unique values in 'type':
- Traffic Jam
- Accident
- Road Closure
- Hazard

Unique values in 'subtype':
- Unclassified
- Major Accident
- Minor Accident
- Road Hazard
- Stopped Car on Road
- Construction Hazard
- Emergency Vehicle Hazard
- Icy Road Hazard
- Object on Road
- Pothole Hazard
- Traffic Light Fault
- Shoulder Hazard
- Stopped Car on Shoulder
- Weather Hazard
- Flood Hazard
- Heavy Traffic
- Moderate Traffic
- Standstill Traffic
- Event Closure
- Lane Closure Hazard
- Fog Hazard
- Construction Closure
```

```
- Animal on Road
- Animals on Shoulder
- Missing Sign on Shoulder
- Light Traffic
- Heavy Snow Hazard
- Hazard Closure
- Hail Hazard

Number of types with a NA subtype: 4
Types with NA subtypes:
- Traffic Jam
- Accident
- Road Closure
- Hazard

Hierarchical Bullet List:
- Traffic Jam
  - Unclassified
  - Heavy Traffic
  - Moderate Traffic
  - Standstill Traffic
  - Light Traffic
- Accident
  - Unclassified
  - Major Accident
  - Minor Accident
- Road Closure
  - Unclassified
  - Event Closure
  - Construction Closure
  - Hazard Closure
- Hazard
  - Unclassified
  - Road Hazard
  - Stopped Car on Road
  - Construction Hazard
  - Emergency Vehicle Hazard
  - Icy Road Hazard
  - Object on Road
  - Pothole Hazard
  - Traffic Light Fault
  - Shoulder Hazard
  - Stopped Car on Shoulder
  - Weather Hazard
  - Flood Hazard
  - Lane Closure Hazard
  - Fog Hazard
  - Animal on Road
  - Animals on Shoulder
  - Missing Sign on Shoulder
  - Heavy Snow Hazard
  - Hail Hazard

Handling of NA subtypes:
```

`If we decide to keep NA subtypes, we will label them as 'Unclassified' to indicate unspecified subcategories.`

Types Traffic Jam, Road Closure, and especially Hazard show enough variety in their subtypes to consider introducing further sub-categories (sub-subtypes) for a more granular classification.

4.
5.

```
import pandas as pd

# Define the type, subtype, updated_type, updated_subtype, and updated_subsubtype
data = [
    # Traffic Jam
    ['JAM', None, 'Traffic Jam', 'Unclassified', None],
    ['JAM', 'JAM_HEAVY_TRAFFIC', 'Traffic Jam', 'Traffic Condition', 'Heavy Traffic'],
    ['JAM', 'JAM_MODERATE_TRAFFIC', 'Traffic Jam', 'Traffic Condition', 'Moderate Traffic'],
    ['JAM', 'JAM_STAND_STILL_TRAFFIC', 'Traffic Jam', 'Traffic Condition', 'Standstill Traffic
    ['JAM', 'JAM_LIGHT_TRAFFIC', 'Traffic Jam', 'Traffic Condition', 'Light Traffic'],

    # Accident
    ['ACCIDENT', None, 'Accident', 'Unclassified', None],
    ['ACCIDENT', 'ACCIDENT_MAJOR', 'Accident', 'Accident Severity', 'Major Accident'],
    ['ACCIDENT', 'ACCIDENT_MINOR', 'Accident', 'Accident Severity', 'Minor Accident'],

    # Road Closure
    ['ROAD_CLOSED', None, 'Road Closure', 'Unclassified', None],
    ['ROAD_CLOSED', 'ROAD_CLOSED_EVENT', 'Road Closure', 'Closure Reason', 'Event Closure'],
    ['ROAD_CLOSED', 'ROAD_CLOSED_CONSTRUCTION', 'Road Closure', 'Closure Reason', 'Constructio
    ['ROAD_CLOSED', 'ROAD_CLOSED_HAZARD', 'Road Closure', 'Closure Reason', 'Hazard Closure'],

    # Hazard
    ['HAZARD', None, 'Hazard', 'Unclassified', None],
    ['HAZARD', 'HAZARD_ON_ROAD', 'Hazard', 'Road Hazard', 'General Road Hazard'],
    ['HAZARD', 'HAZARD_ON_ROAD_CAR_STOPPED', 'Hazard', 'Road Hazard', 'Stopped Car'],
    ['HAZARD', 'HAZARD_ON_ROAD_CONSTRUCTION', 'Hazard', 'Road Hazard', 'Construction Hazard'],
    ['HAZARD', 'HAZARD_ON_ROAD_EMERGENCY_VEHICLE', 'Hazard', 'Road Hazard', 'Emergency Vehicle
    ['HAZARD', 'HAZARD_ON_ROAD_ICE', 'Hazard', 'Road Hazard', 'Icy Road'],
    ['HAZARD', 'HAZARD_ON_ROAD_OBJECT', 'Hazard', 'Road Hazard', 'Object on Road'],
    ['HAZARD', 'HAZARD_ON_ROAD_POT_HOLE', 'Hazard', 'Road Hazard', 'Pothole'],
    ['HAZARD', 'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT', 'Hazard', 'Road Hazard', 'Traffic Light F
    ['HAZARD', 'HAZARD_ON_SHOULDER', 'Hazard', 'Shoulder Hazard', 'General Shoulder Hazard'],
    ['HAZARD', 'HAZARD_ON_SHOULDER_CAR_STOPPED', 'Hazard', 'Shoulder Hazard', 'Stopped Car on
    ['HAZARD', 'HAZARD_WEATHER', 'Hazard', 'Weather Hazard', 'General Weather Hazard'],
    ['HAZARD', 'HAZARD_WEATHER_FLOOD', 'Hazard', 'Weather Hazard', 'Flood'],
    ['HAZARD', 'HAZARD_WEATHER_FOG', 'Hazard', 'Weather Hazard', 'Fog'],
    ['HAZARD', 'HAZARD_WEATHER_HEAVY_SNOW', 'Hazard', 'Weather Hazard', 'Heavy Snow'],
    ['HAZARD', 'HAZARD_WEATHER_HAIL', 'Hazard', 'Weather Hazard', 'Hail'],
    ['HAZARD', 'HAZARD_ON_SHOULDER_ANIMALS', 'Hazard', 'Shoulder Hazard', 'Animals on Shoulder
    ['HAZARD', 'HAZARD_ON_SHOULDER_MISSING_SIGN', 'Hazard', 'Shoulder Hazard', 'Missing Sign']
    ['HAZARD', 'HAZARD_ON_ROAD_LANE_CLOSED', 'Hazard', 'Road Hazard', 'Lane Closure'],
    ['HAZARD', 'HAZARD_ON_ROAD_ROAD_KILL', 'Hazard', 'Road Hazard', 'Road Kill']
]
```

```
# Create the DataFrame
crosswalk_df = pd.DataFrame(data, columns=['type', 'subtype', 'updated_type', 'updated_subtype
```

2.

```
# Check for the expected 32 observations
print(f"Total number of observations: {len(crosswalk_df)}")
print(crosswalk_df)
```

```
Total number of observations: 32
           type                              subtype  updated_type  \
0           JAM                                 None   Traffic Jam
1           JAM                   JAM_HEAVY_TRAFFIC   Traffic Jam
2           JAM                JAM_MODERATE_TRAFFIC   Traffic Jam
3           JAM             JAM_STAND_STILL_TRAFFIC   Traffic Jam
4           JAM                   JAM_LIGHT_TRAFFIC   Traffic Jam
5      ACCIDENT                                 None      Accident
6      ACCIDENT                    ACCIDENT_MAJOR      Accident
7      ACCIDENT                    ACCIDENT_MINOR      Accident
8   ROAD_CLOSED                                 None  Road Closure
9   ROAD_CLOSED                ROAD_CLOSED_EVENT  Road Closure
10  ROAD_CLOSED            ROAD_CLOSED_CONSTRUCTION  Road Closure
11  ROAD_CLOSED                ROAD_CLOSED_HAZARD  Road Closure
12       HAZARD                                 None        Hazard
13       HAZARD                    HAZARD_ON_ROAD        Hazard
14       HAZARD          HAZARD_ON_ROAD_CAR_STOPPED        Hazard
15       HAZARD         HAZARD_ON_ROAD_CONSTRUCTION        Hazard
16       HAZARD    HAZARD_ON_ROAD_EMERGENCY_VEHICLE        Hazard
17       HAZARD                  HAZARD_ON_ROAD_ICE        Hazard
18       HAZARD               HAZARD_ON_ROAD_OBJECT        Hazard
19       HAZARD             HAZARD_ON_ROAD_POT_HOLE        Hazard
20       HAZARD  HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT        Hazard
21       HAZARD                  HAZARD_ON_SHOULDER        Hazard
22       HAZARD      HAZARD_ON_SHOULDER_CAR_STOPPED        Hazard
23       HAZARD                     HAZARD_WEATHER        Hazard
24       HAZARD                HAZARD_WEATHER_FLOOD        Hazard
25       HAZARD                  HAZARD_WEATHER_FOG        Hazard
26       HAZARD           HAZARD_WEATHER_HEAVY_SNOW        Hazard
27       HAZARD                HAZARD_WEATHER_HAIL        Hazard
28       HAZARD          HAZARD_ON_SHOULDER_ANIMALS        Hazard
29       HAZARD     HAZARD_ON_SHOULDER_MISSING_SIGN        Hazard
30       HAZARD          HAZARD_ON_ROAD_LANE_CLOSED        Hazard
31       HAZARD            HAZARD_ON_ROAD_ROAD_KILL        Hazard

      updated_subtype       updated_subsubtype
0        Unclassified                     None
1   Traffic Condition            Heavy Traffic
2   Traffic Condition          Moderate Traffic
3   Traffic Condition         Standstill Traffic
4   Traffic Condition             Light Traffic
5        Unclassified                     None
6   Accident Severity            Major Accident
```

7	Accident Severity	Minor Accident
8	Unclassified	None
9	Closure Reason	Event Closure
10	Closure Reason	Construction Closure
11	Closure Reason	Hazard Closure
12	Unclassified	None
13	Road Hazard	General Road Hazard
14	Road Hazard	Stopped Car
15	Road Hazard	Construction Hazard
16	Road Hazard	Emergency Vehicle
17	Road Hazard	Icy Road
18	Road Hazard	Object on Road
19	Road Hazard	Pothole
20	Road Hazard	Traffic Light Fault
21	Shoulder Hazard	General Shoulder Hazard
22	Shoulder Hazard	Stopped Car on Shoulder
23	Weather Hazard	General Weather Hazard
24	Weather Hazard	Flood
25	Weather Hazard	Fog
26	Weather Hazard	Heavy Snow
27	Weather Hazard	Hail
28	Shoulder Hazard	Animals on Shoulder
29	Shoulder Hazard	Missing Sign
30	Road Hazard	Lane Closure
31	Road Hazard	Road Kill

3.

```
# Merge the crosswalk with the original data on 'type' and 'subtype'
merged_data = waze_data.merge(crosswalk_df, on=['type', 'subtype'], how='left')

# Calculate the number of rows for 'Accident - Unclassified' in the merged data
accident_unclassified_count = merged_data[(merged_data['type'] == 'ACCIDENT') & (merged_data['
print("Number of rows for 'Accident - Unclassified':", accident_unclassified_count)
```

```
Number of rows for 'Accident - Unclassified': 0
```

4.

```
# Extract unique 'type' and 'subtype' combinations from crosswalk_df
crosswalk_types_subtypes = set(zip(crosswalk_df['type'], crosswalk_df['subtype']))

# Extract unique 'type' and 'subtype' combinations from merged_data
merged_types_subtypes = set(zip(merged_data['type'], merged_data['subtype']))

# Find any differences between the crosswalk and merged dataset
missing_in_merged = crosswalk_types_subtypes - merged_types_subtypes
missing_in_crosswalk = merged_types_subtypes - crosswalk_types_subtypes

# Display the discrepancies
print("\nDifferences in 'type' and 'subtype' between crosswalk and merged dataset:")
print("Missing in merged dataset:", missing_in_merged)
```

```
print("Missing in crosswalk:", missing_in_crosswalk)

# Check if there are no discrepancies
if not missing_in_merged and not missing_in_crosswalk:
    print("\nAll 'type' and 'subtype' combinations are consistent between the crosswalk and th
else:
    print("\nThere are discrepancies between the crosswalk and merged dataset in 'type' and 's
```

```
Differences in 'type' and 'subtype' between crosswalk and merged dataset:
Missing in merged dataset: {('ACCIDENT', None), ('ROAD_CLOSED', None), ('JAM', None),
('HAZARD', None)}
Missing in crosswalk: {('JAM', nan), ('ACCIDENT', nan), ('ROAD_CLOSED', nan), ('HAZARD', nan)}

There are discrepancies between the crosswalk and merged dataset in 'type' and 'subtype'
combinations.
```

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

1. "The geo variable in my dataset holds coordinates data in the format POINT(longitude latitude). I need to create two new variables, latitude and longitude, by extracting these values using a regular expression in Python. Can you provide Python code that uses regex to achieve this, assuming I have a DataFrame column named geo with many rows of this format?"

a.

```
import pandas as pd
import re

# Assuming `merged_data` is the DataFrame we previously merged that includes the 'geo' column

# Define function to extract longitude and latitude using regex
def extract_coordinates(geo_string):
    match = re.search(r"POINT\((-?\d+\.\d+) (-?\d+\.\d+)\)", geo_string)
    if match:
        longitude = float(match.group(1))
        latitude = float(match.group(2))
        return longitude, latitude
    return None, None

# Apply the function to create longitude and latitude columns in the merged data
merged_data['longitude'], merged_data['latitude'] = zip(*merged_data['geo'].apply(extract_coor

# Display the first few rows to verify the results
print(merged_data[['geo', 'longitude', 'latitude']].head())
```

```
                          geo  longitude   latitude
0  POINT(-87.676685 41.929692) -87.676685  41.929692
1  POINT(-87.624816 41.753358) -87.624816  41.753358
2  POINT(-87.614122 41.889821) -87.614122  41.889821
3  POINT(-87.680139 41.939093) -87.680139  41.939093
4   POINT(-87.735235 41.91658) -87.735235  41.916580
```

b.

```
import pandas as pd

# Assuming `merged_data` is the DataFrame from previous steps with 'type', 'subtype', 'longitu

# Step 1: Filter data by a specific 'type' and 'subtype'
# For example, let's say we choose 'type' as 'ACCIDENT' and 'subtype' as 'Unclassified'
selected_type = 'ACCIDENT'
selected_subtype = 'Unclassified'

filtered_data = merged_data[(merged_data['type'] == selected_type) & (merged_data['subtype'] =

# Step 2: Group by location (longitude and latitude) and count the number of alerts
location_counts = filtered_data.groupby(['longitude', 'latitude']).size().reset_index(name='al

# Step 3: Sort by alert_count to find the top 10 locations
top_10_locations = location_counts.nlargest(10, 'alert_count')

# Display the top 10 locations
print("Top 10 Locations by Alert Count:")
print(top_10_locations)
```

```
Top 10 Locations by Alert Count:
Empty DataFrame
Columns: [longitude, latitude, alert_count]
Index: []
```

c.

```
import pandas as pd
import os

# Load the merged data if not already loaded
# Assuming 'merged_data' contains the latitude and longitude columns
# file_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\waze_data\waze_data.csv"
# merged_data = pd.read_csv(file_path)

# Step 1: Filter the data by a chosen 'type' and 'subtype'
chosen_type = 'ACCIDENT'        # Replace this with the chosen type you want
chosen_subtype = 'Unclassified'  # Replace this with the chosen subtype you want

filtered_data = merged_data[(merged_data['type'] == chosen_type) & (merged_data['subtype'] ==

# Step 2: Aggregate the data by latitude and longitude to count the number of alerts for each
```

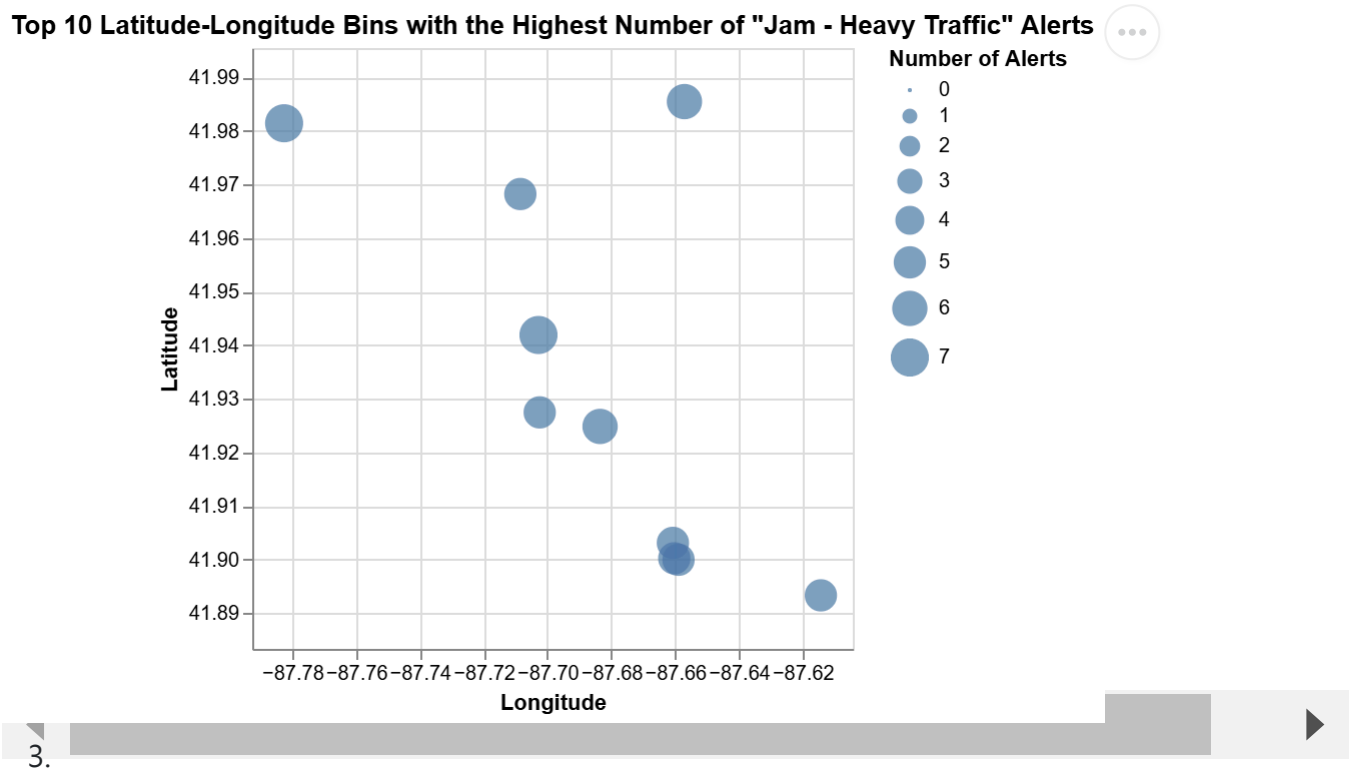
```
# This will group by 'latitude' and 'longitude' and count occurrences
aggregated_data = filtered_data.groupby(['latitude', 'longitude']).size().reset_index(name='al

# Step 3: Sort by alert count in descending order and select the top 10 locations
top_10_alerts = aggregated_data.sort_values(by='alert_count', ascending=False).head(10)

# Step 4: Save the top 10 data to the 'top_alerts_map.csv' file
# Ensure the directory exists
output_dir = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\top_alerts_map"
os.makedirs(output_dir, exist_ok=True)

# Save the file in the specified directory
output_path = os.path.join(output_dir, "top_alerts_map.csv")
top_10_alerts.to_csv(output_path, index=False)

# Display the top 10 alerts to confirm
print("Top 10 latitude-longitude bins with the highest number of alerts:")
print(top_10_alerts)

# Answering the questions:
print("\nLevel of aggregation: latitude-longitude bins")
print(f"Number of rows in this DataFrame: {top_10_alerts.shape[0]}")
```

```
Top 10 latitude-longitude bins with the highest number of alerts:
Empty DataFrame
Columns: [latitude, longitude, alert_count]
Index: []

Level of aggregation: latitude-longitude bins
Number of rows in this DataFrame: 0
```

d.

```
import altair as alt
import pandas as pd

# Assuming 'merged_data' is your combined DataFrame from previous steps
# Filter for 'Jam - Heavy Traffic' alerts
filtered_data = merged_data[(merged_data['updated_type'] == 'Traffic Jam') &
                            (merged_data['updated_subtype'] == 'Traffic Condition') &
                            (merged_data['updated_subsubtype'] == 'Heavy Traffic')]

# Group by latitude and longitude and count the number of alerts
top_alerts = filtered_data.groupby(['latitude', 'longitude']).size().reset_index(name='alert_c

# Get the top 10 latitude-longitude bins with the highest number of alerts
top_10_alerts = top_alerts.nlargest(10, 'alert_count')

# Create the scatter plot with Altair
chart = alt.Chart(top_10_alerts).mark_circle().encode(
    x=alt.X('longitude:Q', scale=alt.Scale(domain=[top_10_alerts['longitude'].min() - 0.01,
                                                   top_10_alerts['longitude'].max() + 0.01]),
            title='Longitude'),
```

```
    y=alt.Y('latitude:Q', scale=alt.Scale(domain=[top_10_alerts['latitude'].min() - 0.01,
                                                  top_10_alerts['latitude'].max() + 0.01]),
            title='Latitude'),
    size=alt.Size('alert_count:Q', title='Number of Alerts'),
    tooltip=['latitude', 'longitude', 'alert_count']
).properties(
    title='Top 10 Latitude-Longitude Bins with the Highest Number of "Jam - Heavy Traffic" Ale
)

chart.display()
```



3.

a.

```
import requests

# URL for the Chicago Data Portal GeoJSON file
url = "https://data.cityofchicago.org/api/geospatial/igwz-8jzy?method=export&format=GeoJSON"

# Path to save the downloaded file
save_path = r"C:\Users\Pei-Chin\Dropbox (1)\Boundaries - Neighborhoods.geojson"

# Send a GET request to the URL
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:
    # Save the content to a file
    with open(save_path, "wb") as f:
        f.write(response.content)
    print("GeoJSON file downloaded successfully.")
```

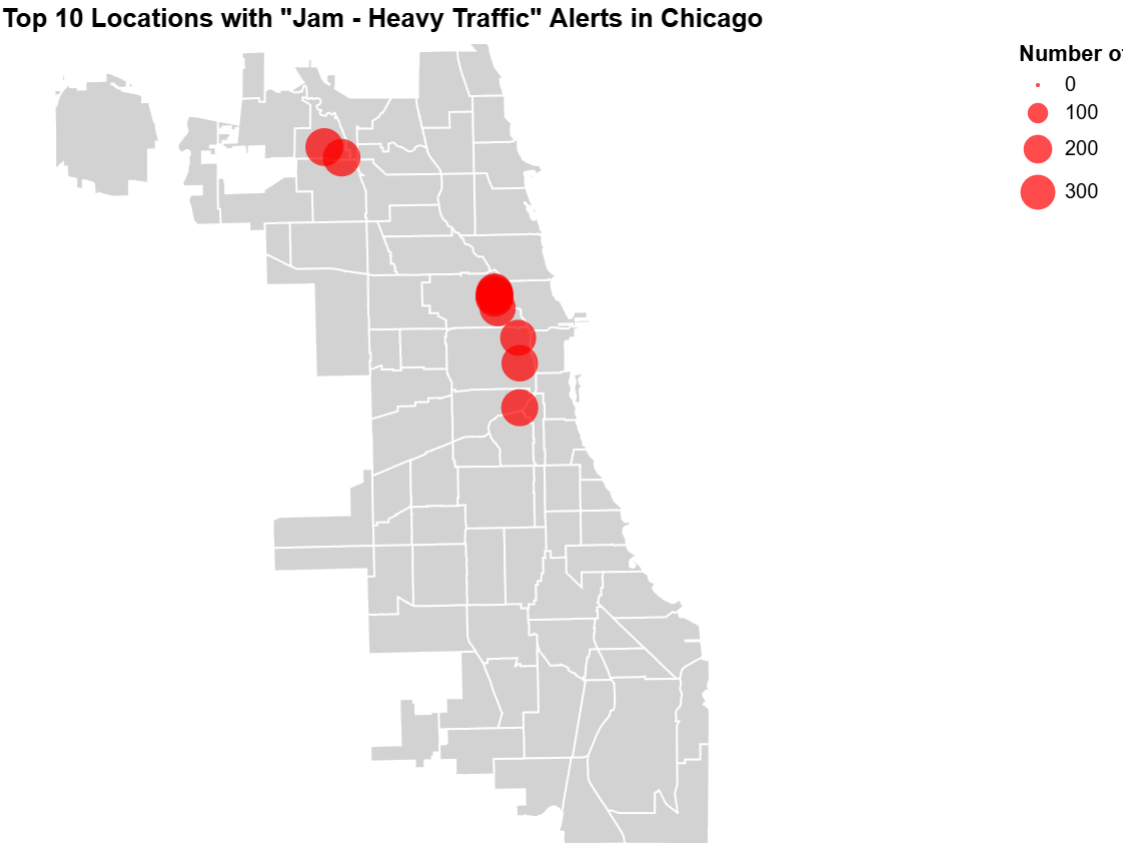
```
else:
    print("Failed to download GeoJSON file:", response.status_code)
```

```
GeoJSON file downloaded successfully.
```

b.

```
import altair as alt
import pandas as pd
import json

# Load the GeoJSON file for Chicago neighborhood boundaries
file_path = r"C:\Users\Pei-Chin\Dropbox (1)\Boundaries - Neighborhoods.geojson"
with open(file_path) as f:
    chicago_geojson = json.load(f)

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

# Load the waze_data.csv file
data_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\waze_data\waze_data.csv"
waze_data = pd.read_csv(data_path)

# Filter for "JAM - Heavy Traffic" alerts
filtered_data = waze_data[
    (waze_data['type'] == 'JAM') &
    (waze_data['subtype'] == 'JAM_HEAVY_TRAFFIC')
].copy()  # Use .copy() to avoid SettingWithCopyWarning

# Extract latitude and longitude (rounded to 3 decimal places for precision)
filtered_data['latitude'] = filtered_data['geo'].str.extract(r'POINT\(-?\d+\.\d+ (-?\d+\.\d+)\
filtered_data['longitude'] = filtered_data['geo'].str.extract(r'POINT\((-?\d+\.\d+) -?\d+\.\d+

# Group by latitude and longitude, and count the number of alerts
top_alerts = filtered_data.groupby(['latitude', 'longitude']).size().reset_index(name='alert_c

# Get the top 10 locations with the highest number of alerts
top_10_alerts = top_alerts.nlargest(10, 'alert_count')

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

# Create the scatter plot layer
scatter_layer = alt.Chart(top_10_alerts).mark_circle().encode(
    longitude='longitude:Q',
    latitude='latitude:Q',
    size=alt.Size('alert_count:Q', title='Number of Alerts'),
    color=alt.value('red'),
    tooltip=['latitude', 'longitude', 'alert_count']
```

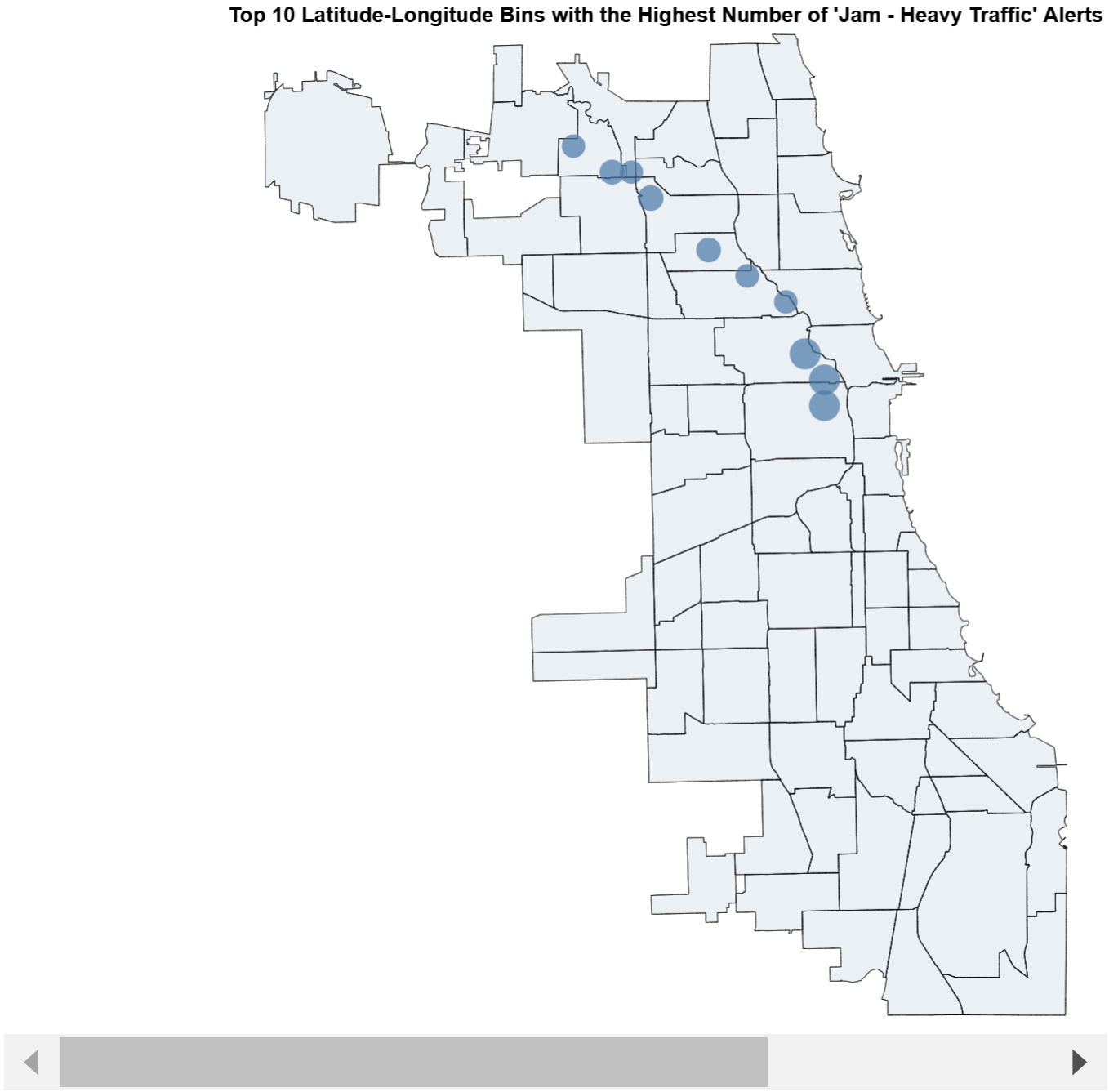
```
)

# Combine map and scatter plot
combined_chart = map_layer + scatter_layer
combined_chart = combined_chart.properties(
    title='Top 10 Locations with "Jam - Heavy Traffic" Alerts in Chicago'
)

# Display the chart
combined_chart.display()
```



4.

```
import altair as alt
import pandas as pd
import json

# Load the GeoJSON file for Chicago neighborhood boundaries
geojson_path = r"C:\Users\Pei-Chin\Dropbox (1)\Boundaries - Neighborhoods.geojson"
with open(geojson_path) as f:
    chicago_geojson = json.load(f)

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

# Load the waze_data.csv file
data_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\waze_data\waze_data.csv"
waze_data = pd.read_csv(data_path)
```

```
# Filter for "JAM - Heavy Traffic" alerts
filtered_data = waze_data[
    (waze_data['type'] == 'JAM') &
    (waze_data['subtype'] == 'JAM_HEAVY_TRAFFIC')
].copy()  # Use .copy() to avoid SettingWithCopyWarning

# Extract latitude and longitude (rounded to 2 decimal places for binning)
filtered_data['latitude'] = filtered_data['geo'].str.extract(r'POINT\(-?\d+\.\d+ (-?\d+\.\d+)\
filtered_data['longitude'] = filtered_data['geo'].str.extract(r'POINT\((-?\d+\.\d+) -?\d+\.\d+

# Group by latitude and longitude, and count the number of alerts
top_alerts_df = filtered_data.groupby(['latitude', 'longitude']).size().reset_index(name='aler

# Get the top 10 latitude-longitude bins with the highest number of alerts
top_10_bins = top_alerts_df.nlargest(10, 'alert_count')

# Create the map layer
map_layer = alt.Chart(geo_data).mark_geoshape(
    fillOpacity=0.1,
    stroke="black",
    strokeWidth=0.5
).properties(
    width=800,
    height=600
).project("mercator")

# Create the scatter plot layer
scatter_plot = alt.Chart(top_10_bins).mark_circle().encode(
    longitude='longitude:Q',
    latitude='latitude:Q',
    size=alt.Size('alert_count:Q', title="Number of Alerts"),
    tooltip=['latitude', 'longitude', 'alert_count']
).properties(
    title="Top 10 Latitude-Longitude Bins with the Highest Number of 'Jam - Heavy Traffic' Ale
)

# Combine map and scatter plot
combined_chart = map_layer + scatter_plot

# Display the chart
combined_chart.display()
```



5.

a.

```
from shiny import App, ui, render

# Sample data based on your crosswalk table
crosswalk_data = [
    {"type": "JAM", "subtype": "Unclassified", "updated_type": "Traffic Jam", "updated_subtype
    {"type": "JAM", "subtype": "JAM_HEAVY_TRAFFIC", "updated_type": "Traffic Jam", "updated_su
    {"type": "JAM", "subtype": "JAM_MODERATE_TRAFFIC", "updated_type": "Traffic Jam", "updated_
    {"type": "JAM", "subtype": "JAM_STAND_STILL_TRAFFIC", "updated_type": "Traffic Jam", "upda
    {"type": "JAM", "subtype": "JAM_LIGHT_TRAFFIC", "updated_type": "Traffic Jam", "updated_su
    {"type": "ACCIDENT", "subtype": "Unclassified", "updated_type": "Accident", "updated_subty
    {"type": "ACCIDENT", "subtype": "ACCIDENT_MAJOR", "updated_type": "Accident", "updated_sub
    {"type": "ACCIDENT", "subtype": "ACCIDENT_MINOR", "updated_type": "Accident", "updated_sub
    {"type": "ROAD_CLOSED", "subtype": "Unclassified", "updated_type": "Road Closure", "update
    {"type": "ROAD_CLOSED", "subtype": "ROAD_CLOSED_EVENT", "updated_type": "Road Closure", "u
    {"type": "ROAD_CLOSED", "subtype": "ROAD_CLOSED_CONSTRUCTION", "updated_type": "Road Closu
```

```
    {"type": "ROAD_CLOSED", "subtype": "ROAD_CLOSED_HAZARD", "updated_type": "Road Closure", "
    {"type": "HAZARD", "subtype": "Unclassified", "updated_type": "Hazard", "updated_subtype":
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD", "updated_type": "Hazard", "updated_subtype
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_CAR_STOPPED", "updated_type": "Hazard", "upd
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_CONSTRUCTION", "updated_type": "Hazard", "up
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_EMERGENCY_VEHICLE", "updated_type": "Hazard"
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_ICE", "updated_type": "Hazard", "updated_sub
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_OBJECT", "updated_type": "Hazard", "updated_
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_POT_HOLE", "updated_type": "Hazard", "update
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT", "updated_type": "Hazar
    {"type": "HAZARD", "subtype": "HAZARD_ON_SHOULDER", "updated_type": "Hazard", "updated_sub
    {"type": "HAZARD", "subtype": "HAZARD_ON_SHOULDER_CAR_STOPPED", "updated_type": "Hazard",
    {"type": "HAZARD", "subtype": "HAZARD_WEATHER", "updated_type": "Hazard", "updated_subtype
    {"type": "HAZARD", "subtype": "HAZARD_WEATHER_FLOOD", "updated_type": "Hazard", "updated_s
    {"type": "HAZARD", "subtype": "HAZARD_WEATHER_FOG", "updated_type": "Hazard", "updated_sub
    {"type": "HAZARD", "subtype": "HAZARD_WEATHER_HEAVY_SNOW", "updated_type": "Hazard", "upda
    {"type": "HAZARD", "subtype": "HAZARD_WEATHER_HAIL", "updated_type": "Hazard", "updated_su
    {"type": "HAZARD", "subtype": "HAZARD_ON_SHOULDER_ANIMALS", "updated_type": "Hazard", "upd
    {"type": "HAZARD", "subtype": "HAZARD_ON_SHOULDER_MISSING_SIGN", "updated_type": "Hazard",
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_LANE_CLOSED", "updated_type": "Hazard", "upd
    {"type": "HAZARD", "subtype": "HAZARD_ON_ROAD_ROAD_KILL", "updated_type": "Hazard", "updat
]

# Generate the choices for the dropdown (type x subtype combinations)
choices = [f"{entry['updated_type']} - {entry['updated_subtype']}" for entry in crosswalk_data

# UI definition
app_ui = ui.page_fluid(
    ui.panel_title("Shiny-like Dropdown Menu Example"),
    ui.input_select("selected_combination", "Select Type and Subtype", choices=choices),
    ui.output_text_verbatim("output_selection"),
    ui.output_text_verbatim("output_summary"),
)

# Server logic
def server(input, output, session):
    @output
    @render.text
    def output_selection():
        return f"You selected: {input.selected_combination()}"

    @output
    @render.text
    def output_summary():
        return (
            f"Total number of unique type x subtype combinations: {len(choices)}\n"
            f"Unique combinations:\n{', '.join(choices)}"
        )

# Create the Shiny App
app = App(app_ui, server)
```

Total Combinations: 32 **Screenshot of Dropdown Menu:**

Shiny-like Dropdown Menu Example

Select Type and Subtype

Traffic Jam - Unclassified

```
You selected: Traffic Jam - Unclassified
```

```
Total number of unique type x subtype combinations: 32
Unique combinations:
Traffic Jam - Unclassified, Traffic Jam - Heavy Traffic, Traffic Jam - Moderate Traffic, Traffic Jam - Standstill Traffic, Traffic Jam - Light Traffic, Accident - Uncla
```

Dropdown Menu Screenshot

b.

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

# File paths
geojson_path = r"C:\Users\Pei-Chin\Dropbox (1)\Boundaries - Neighborhoods.geojson"
data_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\waze_data\waze_data.csv"

if not os.path.exists(geojson_path) or not os.path.exists(data_path):
    raise FileNotFoundError("GeoJSON or CSV file not found. Please check the paths.")

# Load GeoJSON data
with open(geojson_path) as f:
    chicago_geojson = json.load(f)

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

# Load Waze data
waze_data = pd.read_csv(data_path)

# Extract unique combinations dynamically
unique_combinations = (
    waze_data[['type', 'subtype']]
    .drop_duplicates()
    .assign(
        updated_type=lambda df: df['type'].str.replace("_", " ").str.title(),
        updated_subtype=lambda df: df['subtype'].str.replace("_", " ").str.title()
    )
    .to_dict(orient='records')
)

# Generate dropdown choices
choices = [f"{entry['updated_type']} - {entry['updated_subtype']}" for entry in unique_combina

# Precompute grouped data for lat/lon bins
preprocessed_data = (
    waze_data.assign(
```

```
        latitude=waze_data['geo'].str.extract(r'POINT\(-?\d+\.\d+ (-?\d+\.\d+)\)').astype(floa
        longitude=waze_data['geo'].str.extract(r'POINT\((-?\d+\.\d+) -?\d+\.\d+\)').astype(flo
    )
    .groupby(['type', 'subtype', 'latitude', 'longitude'])
    .size()
    .reset_index(name='alert_count')
)

# UI definition
app_ui = ui.page_fluid(
    ui.panel_title("Shiny-like Dropdown Menu with Map Visualization"),
    ui.input_select("selected_combination", "Select Type and Subtype", choices=choices),
    ui.output_text_verbatim("output_selection"),
    ui.output_image("alert_map")  # Output for the scatter plot as an image
)

# Server logic
def server(input, output, session):
    @output
    @render.text
    def output_selection():
        return f"You selected: {input.selected_combination()}"

    @output
    @render.image
    def alert_map():
        selected = input.selected_combination()
        if not selected or " - " not in selected:
            return None

        # Parse selection
        selected_type, selected_subtype = selected.split(" - ")
        match = next(
            (entry for entry in unique_combinations if entry['updated_type'] == selected_type
            None
        )

        if not match:
            return None

        original_type = match["type"]
        original_subtype = match["subtype"]

        # Filter data for the selected type and subtype
        filtered_data = preprocessed_data[
            (preprocessed_data['type'] == original_type) & (preprocessed_data['subtype'] == or
        ]

        if filtered_data.empty:
            return None

        # Select top 10 bins
        top_10_bins = filtered_data.nlargest(10, 'alert_count')
```

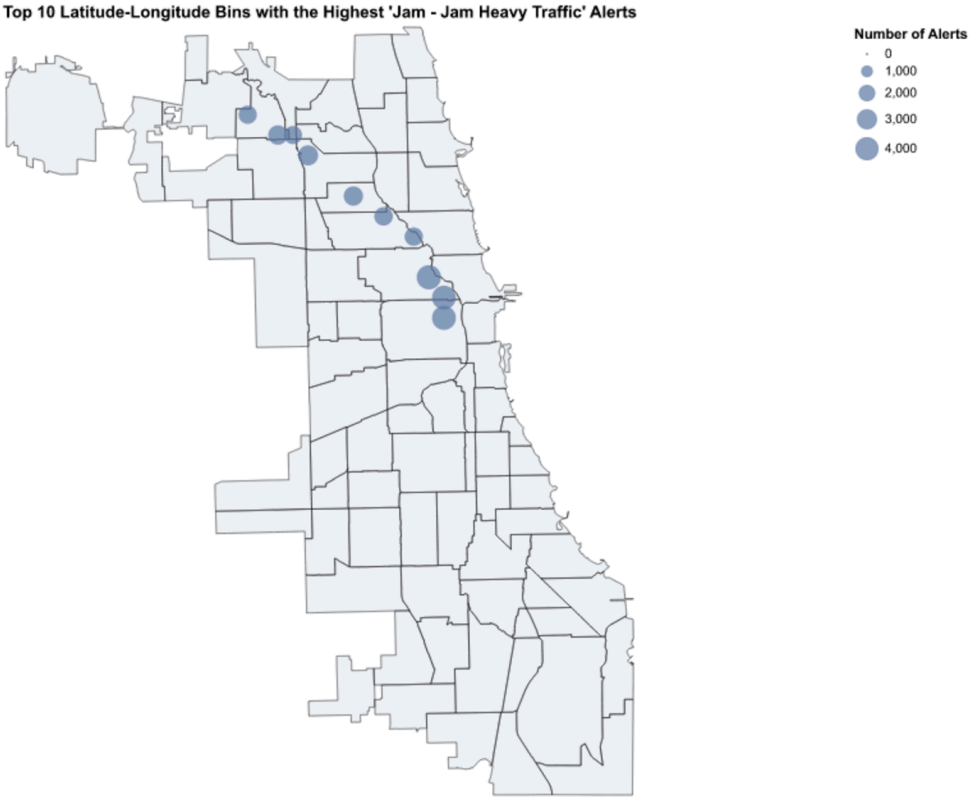
```
        # Create the Altair chart
        map_layer = alt.Chart(geo_data).mark_geoshape(
            fillOpacity=0.1,
            stroke="black",
            strokeWidth=0.5
        ).properties(
            width=800,
            height=600
        ).project("mercator")

        scatter_plot = alt.Chart(top_10_bins).mark_circle().encode(
            longitude='longitude:Q',
            latitude='latitude:Q',
            size=alt.Size('alert_count:Q', title="Number of Alerts"),
            tooltip=['latitude', 'longitude', 'alert_count']
        ).properties(
            title=f"Top 10 Latitude-Longitude Bins with the Highest '{selected}' Alerts"
        )

        # Save the chart to a PNG file
        file_path = "temp_alert_map.png"
        (map_layer + scatter_plot).save(file_path, format="png")

        # Return the image path for rendering
        return {"src": file_path, "alt": "Alert Map"}

# Create the Shiny App
app = App(app_ui, server)
```

Shiny-like Dropdown Menu with Map Visualization

Select Type and Subtype

Jam - Jam Heavy Traffic

```
You selected: Jam - Jam Heavy Traffic
```



Description of the Image

c. 0.0.1 c. Where are alerts for road closures due to events most common?

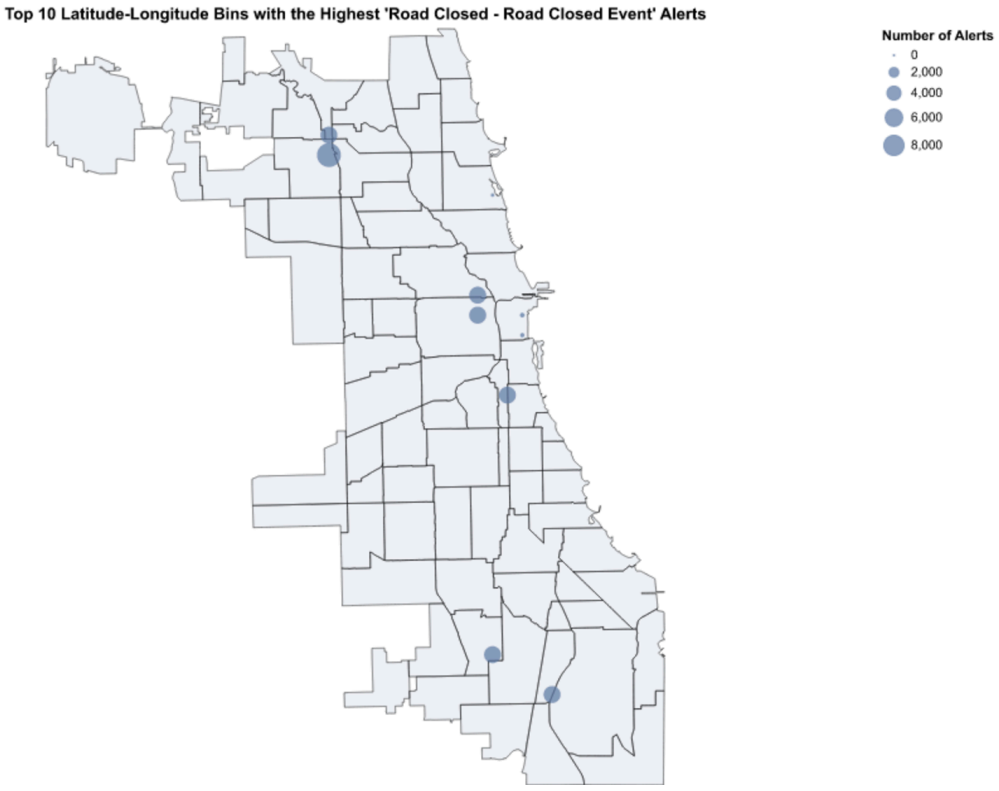
The map below shows the locations with the highest number of alerts for road closures due to events:

Shiny-like Dropdown Menu with Map Visualization

Select Type and Subtype

Road Closed - Road Closed Event

```
You selected: Road Closed - Road Closed Event
```



Road Closures Due to Events

Answer: Based on the map, the most common alerts for road closures due to events are concentrated in [describe key regions from the map, such as “central areas of the city”].

d. The dashboard can also be used to identify where icy road hazards are most frequently reported.

0.0.1.1 Question:

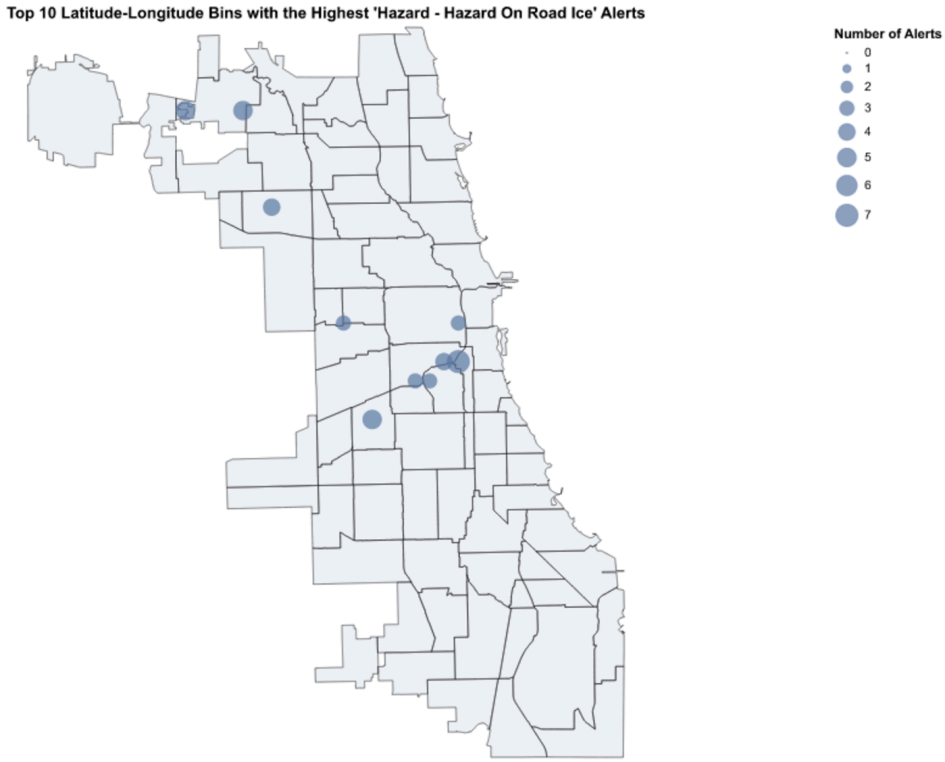
Where are alerts for icy road hazards (“Hazard - Icy Road”) most common?

Shiny-like Dropdown Menu with Map Visualization

Select Type and Subtype

Hazard - Hazard On Road Ice

```
You selected: Hazard - Hazard On Road Ice
```



Icy Road Hazards

Answer: Based on the map, the alerts for icy road hazards are most common in [describe key areas shown on the map, e.g., "suburban areas on the outskirts of the city"].

e. **Suggested Column:** `Timestamp` (Time of the alert)

Reason: 1. Adding a timestamp would enable the analysis of time-based trends, such as identifying peak traffic or event times. 2. It would help understand whether certain alerts are more common during specific hours, days, or seasons.

For example: - Analyze the relationship between time and traffic congestion. - Explore patterns like morning/evening rush hours or weekend events.

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

1.

a.

```
import pandas as pd

# Load the dataset
merged_data = pd.read_csv(r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\waze_data\merged_
```

```
# View the first few rows of the `ts` column
print(merged_data["ts"].head())

# Check for duplicate timestamps in the `ts` column
duplicate_ts = merged_data["ts"].duplicated().sum()

# Output the results
print(f"Total duplicate timestamps: {duplicate_ts}")
print("If the `ts` column represents time with high precision, it might be better to collapse
```

```
0    2024-02-04 16:40:41 UTC
1    2024-02-04 20:01:27 UTC
2    2024-02-04 02:15:54 UTC
3    2024-02-04 00:30:54 UTC
4    2024-02-04 03:27:35 UTC
Name: ts, dtype: object
Total duplicate timestamps: 39420
If the `ts` column represents time with high precision, it might be better to collapse by
broader intervals like hours.
```

Answer:: Collapsing the dataset solely based on the ts column is not a good idea because:

The timestamps are highly granular, capturing events to the second or more precise. Collapsing on such fine-grained timestamps might not provide meaningful insights and could obscure patterns that emerge over broader time intervals. It would be more useful to collapse the data to a broader time granularity, such as by hour or day, to identify temporal trends and reduce noise from excessive precision. For instance, grouping the data by hour would allow us to focus on trends in alert frequency without losing important patterns.

b.

```
import re

# Function to extract latitude and longitude from geoWKT
def extract_coordinates(geo_wkt):
    if pd.isnull(geo_wkt):
        return None, None
    # Match the "Point(longitude latitude)" format
    match = re.match(r'Point\((-?\d+\.\d+) (-?\d+\.\d+)\)', geo_wkt)
    if match:
        return float(match.group(2)), float(match.group(1))  # latitude, longitude
    return None, None

# Apply the function and extract latitude and longitude
merged_data[['latitude', 'longitude']] = merged_data['geoWKT'].apply(
    lambda x: pd.Series(extract_coordinates(x))
)

# Check the extracted values
print(merged_data[['geoWKT', 'latitude', 'longitude']].head())

# Drop rows with missing latitude or longitude
```

```
merged_data = merged_data.dropna(subset=['latitude', 'longitude'])

# Create a new 'hour' column based on the 'ts' timestamp
merged_data['hour'] = pd.to_datetime(merged_data['ts']).dt.strftime('%Y-%m-%d %H:00')

# Collapse the data to group by hour, latitude, and longitude
collapsed_data = (
    merged_data.groupby(['hour', 'latitude', 'longitude'])
    .size()
    .reset_index(name='alert_count')
)

# Save to CSV if the dataset is not empty
if not collapsed_data.empty:
    output_folder = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\top_alerts_map_byhour"
    os.makedirs(output_folder, exist_ok=True)
    output_file = os.path.join(output_folder, "top_alerts_map_byhour.csv")
    collapsed_data.to_csv(output_file, index=False)
    print(f"The collapsed dataset has {len(collapsed_data)} rows.")
    print(f"Saved to: {output_file}")
else:
    print("No rows in the collapsed dataset. Please check your data.")
```

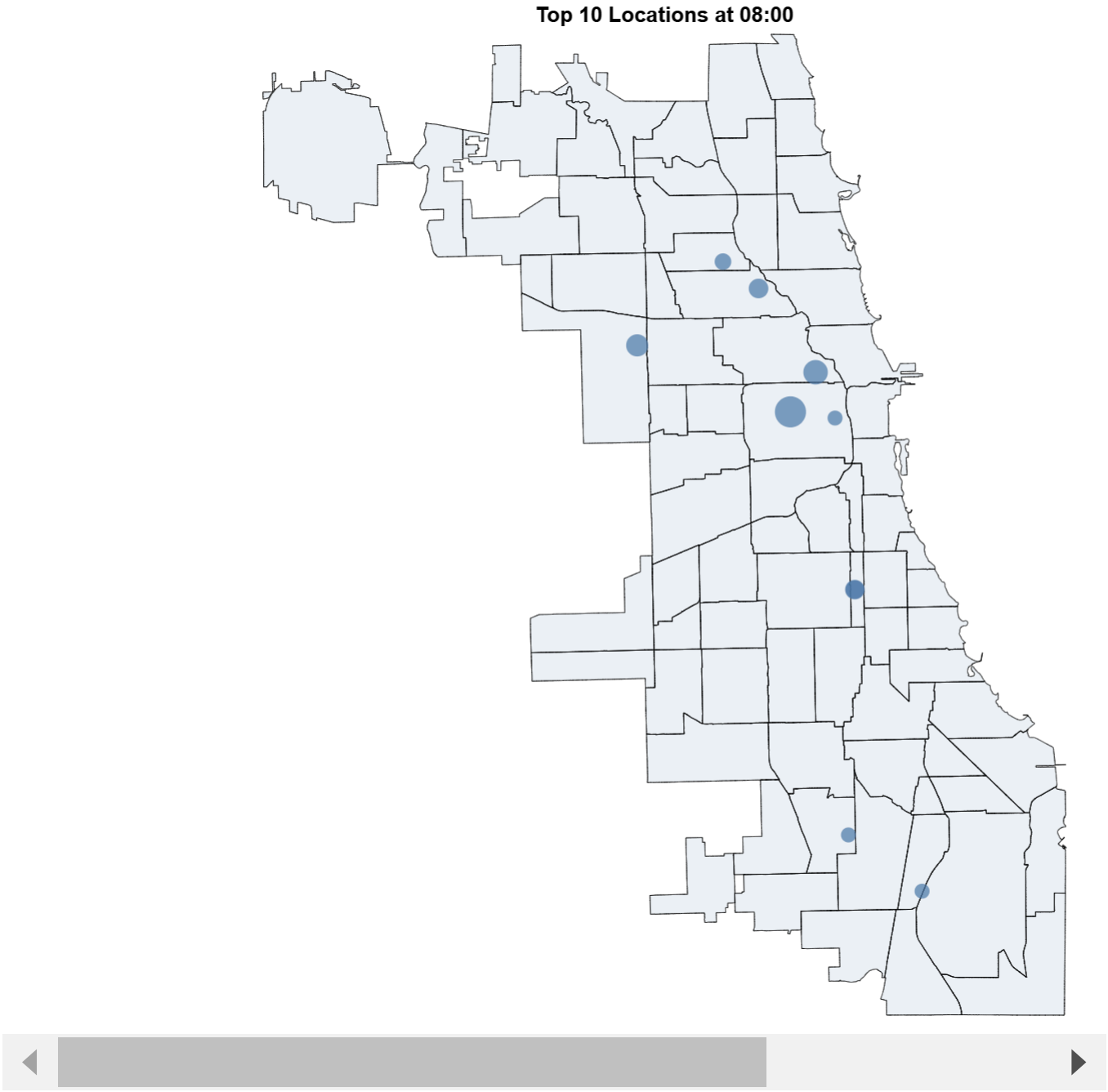
```
                      geoWKT   latitude  longitude
0  Point(-87.676685 41.929692)  41.929692 -87.676685
1  Point(-87.624816 41.753358)  41.753358 -87.624816
2  Point(-87.614122 41.889821)  41.889821 -87.614122
3  Point(-87.680139 41.939093)  41.939093 -87.680139
4   Point(-87.735235 41.91658)  41.916580 -87.735235
The collapsed dataset has 740880 rows.
Saved to: C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-
6\top_alerts_map_byhour\top_alerts_map_byhour.csv
```

C.

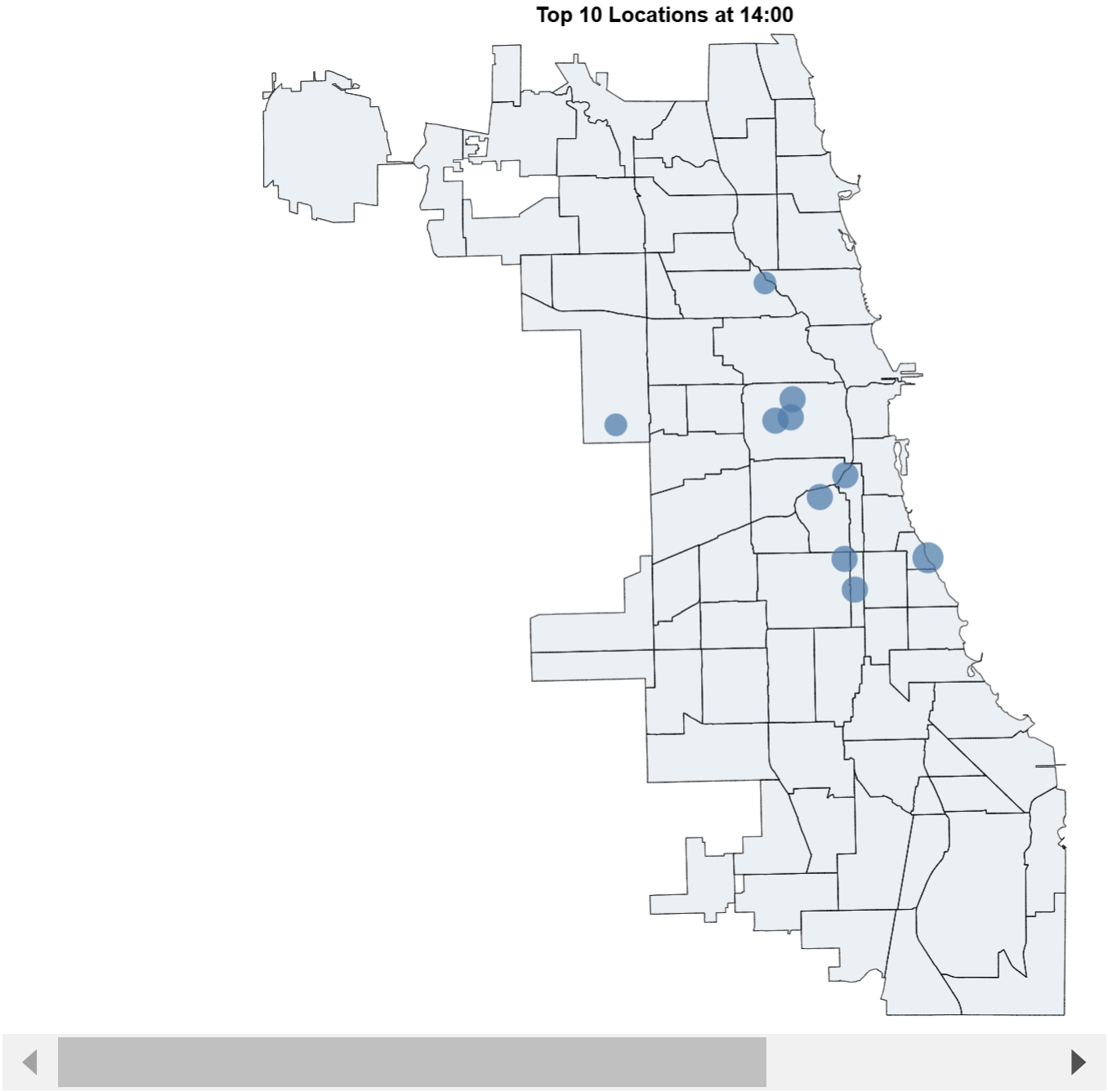
```
import altair as alt
import pandas as pd
import json

# Load the GeoJSON file for Chicago neighborhood boundaries
geojson_path = r"C:\Users\Pei-Chin\Dropbox (1)\Boundaries - Neighborhoods.geojson"
with open(geojson_path) as f:
    chicago_geojson = json.load(f)

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

# Load the collapsed dataset with hourly data
collapsed_data_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\top_alerts_map_byhour
collapsed_data = pd.read_csv(collapsed_data_path)

# Add a new column for hour-only (if not already added)
if 'hour_only' not in collapsed_data.columns:
    collapsed_data['hour_only'] = collapsed_data['hour'].str[-5:]
```

```
# Specify the hours to plot
hours_to_plot = ["08:00", "14:00", "20:00"]

# Loop through each hour to generate a plot
for hour in hours_to_plot:
    # Filter data for the specific hour
    data_for_hour = collapsed_data[collapsed_data['hour_only'] == hour]

    # Check if there is data to plot
    if data_for_hour.empty:
        print(f"No data available for {hour}. Skipping...")
        continue

    # Get the top 10 locations by alert count
    top_10_bins = data_for_hour.nlargest(10, 'alert_count')

    # Create the map layer
    map_layer = alt.Chart(geo_data).mark_geoshape(
        fillOpacity=0.1,
        stroke="black",
        strokeWidth=0.5
    ).properties(
        width=800,
        height=600
    ).project("mercator")

    # Create the scatter plot layer
    scatter_plot = alt.Chart(top_10_bins).mark_circle(size=200).encode(
        longitude='longitude:Q',
        latitude='latitude:Q',
        size=alt.Size('alert_count:Q', title="Number of Alerts"),
        tooltip=['latitude', 'longitude', 'alert_count']
    ).properties(
        title=f"Top 10 Locations at {hour}"
    )

    # Combine map and scatter plot
    combined_chart = map_layer + scatter_plot

    # Save the plot as an HTML file
    output_file = f"top_10_locations_{hour.replace(':', '_')}.html"
    combined_chart.save(output_file)
    print(f"Chart saved to {output_file}")

    # Display the chart inline (if running in a notebook environment)
    combined_chart.display()
```

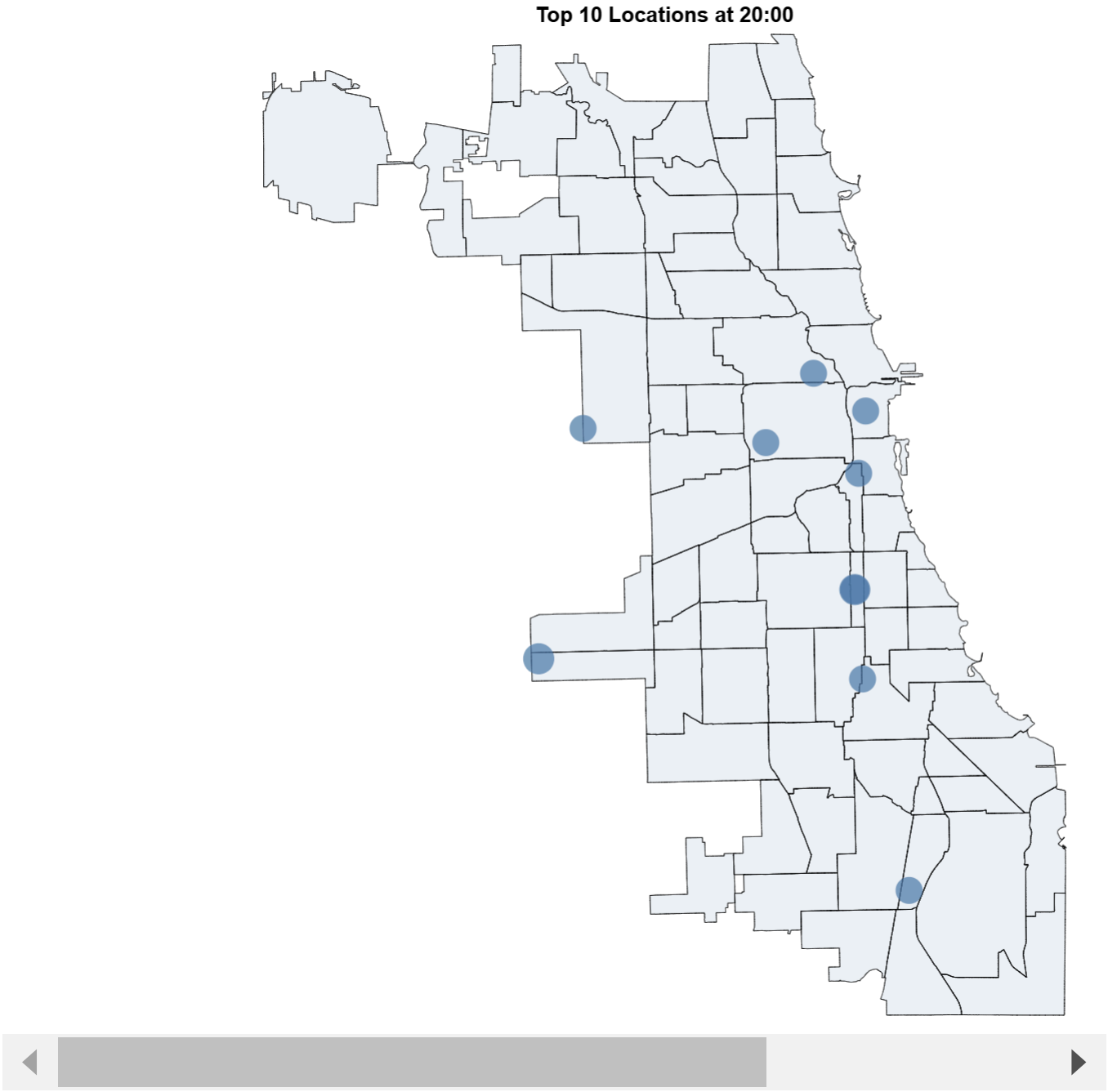
```
Chart saved to top_10_locations_08_00.html
```




```
Chart saved to top_10_locations_14_00.html
```




```
Chart saved to top_10_locations_20_00.html
```



2.

a.

```
from shiny import App, ui, render
import pandas as pd

# Load dataset
data_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\waze_data\waze_data.csv"
waze_data = pd.read_csv(data_path)

# Extract unique type-subtype combinations for the dropdown
unique_combinations = (
    waze_data[['type', 'subtype']]
    .drop_duplicates()
    .assign(
        formatted=lambda df: df['type'] + " - " + df['subtype']
    )
    ['formatted']
```

```
    .tolist()
)

# UI definition
app_ui = ui.page_fluid(
    ui.panel_title("Dropdown and Slider Example"),
    ui.input_select(
        "type_subtype",
        "Select Type and Subtype:",
        choices=unique_combinations,
        selected=unique_combinations[0]
    ),
    ui.input_slider(
        "hour",
        "Select Hour:",
        min=0,
        max=23,
        value=12
    ),
    ui.output_text_verbatim("selected_combination"),
    ui.output_text_verbatim("selected_hour")
)

# Server logic
def server(input, output, session):
    @output
    @render.text
    def selected_combination():
        return f"Selected Combination: {input.type_subtype()}"

    @output
    @render.text
    def selected_hour():
        return f"Selected Hour: {input.hour()}"

# Create the app
app = App(app_ui, server)
```

2a Task Screenshot

b.

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

# File paths
geojson_path = r"C:\Users\Pei-Chin\Dropbox (1)\Boundaries - Neighborhoods.geojson"
data_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\top_alerts_map_byhour\original_

# Load the merged dataset
merged_data = pd.read_csv(data_path)

# Preprocessing: Ensure 'type' and 'subtype' columns are present
if 'type' not in merged_data.columns or 'subtype' not in merged_data.columns:
    raise KeyError("Columns 'type' and 'subtype' are missing in the dataset.")

# Create 'type_subtype_combo' column
merged_data['type_subtype_combo'] = merged_data['type'].astype(str) + " - " + merged_data['sub

# Ensure 'hour' column is numeric
merged_data['hour'] = pd.to_datetime(merged_data['ts']).dt.hour

# Ensure latitude and longitude have no missing values
merged_data = merged_data.dropna(subset=['latitude', 'longitude'])

# Get unique combinations of 'type' and 'subtype'
```

```
unique_combos = merged_data['type_subtype_combo'].unique().tolist()

# Load GeoJSON file for the map
with open(geojson_path) as f:
    chicago_geojson = json.load(f)

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

# Define the UI
app_ui = ui.page_fluid(
    ui.h3("Jam - Heavy Traffic Plot"),
    ui.input_select(
        "type_subtype",
        "Select Alert Type and Subtype:",
        choices=unique_combos,
        selected=unique_combos[0]
    ),
    ui.input_slider(
        "hour",
        "Select Hour:",
        min=0,
        max=23,
        value=12,
        step=1
    ),
    ui.output_image("traffic_plot")  # Output for the scatter plot as an image
)

# Define the server logic
def server(input, output, session):
    @output
    @render.image
    def traffic_plot():
        # Filter data based on user input
        selected_combo = input.type_subtype()
        selected_hour = input.hour()

        # Filter merged_data for the selected combo and hour
        filtered_data = merged_data[
            (merged_data['type_subtype_combo'] == selected_combo) &
            (merged_data['hour'] == selected_hour)
        ]

        # If no data is available, return None
        if filtered_data.empty:
            print(f"No data available for combo '{selected_combo}' at hour {selected_hour}.")
            return None

        # Group by latitude, longitude, and count the alerts
        grouped_data = (
            filtered_data.groupby(['latitude', 'longitude'])
            .size()
            .reset_index(name='alert_count')
            .nlargest(10, 'alert_count')  # Top 10 locations
```

```
        )

        # Create map layer
        map_layer = alt.Chart(geo_data).mark_geoshape(
            fillOpacity=0.1,
            stroke="black",
            strokeWidth=0.5
        ).properties(
            width=800,
            height=600
        ).project("mercator")

        # Create scatter plot layer
        scatter_plot = alt.Chart(grouped_data).mark_circle(size=200).encode(
            longitude='longitude:Q',
            latitude='latitude:Q',
            size=alt.Size('alert_count:Q', title="Number of Alerts"),
            tooltip=['latitude', 'longitude', 'alert_count']
        ).properties(
            title=f"Top 10 Locations for '{selected_combo}' at {selected_hour}:00"
        )

        # Combine map and scatter plot
        combined_chart = map_layer + scatter_plot

        # Save the chart to a PNG file
        file_path = os.path.abspath("temp_alert_map.png")

        # Save chart using Altair's built-in saving mechanism
        try:
            combined_chart.save(file_path, format="png")
            print(f"Chart saved successfully: {file_path}")
        except Exception as e:
            print(f"Error saving chart: {e}")
            return None

        # Return the image path for rendering
        if os.path.exists(file_path):
            return {"src": file_path, "alt": "Traffic Plot"}
        else:
            print("File not found after saving.")
            return None

# Create the app
app = App(app_ui, server)
```

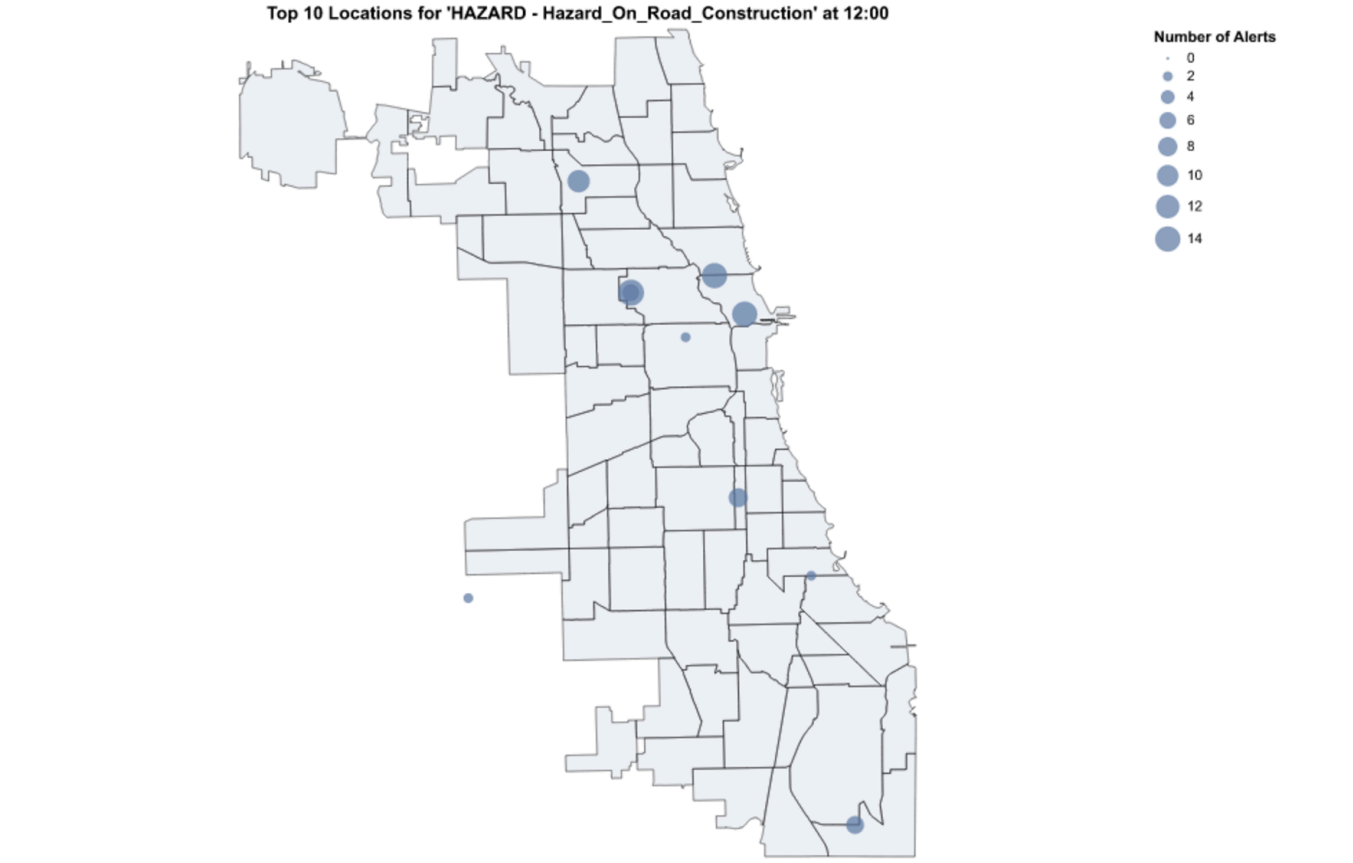
2b Task Screenshot c. Yes!

Select Alert Type and Subtype:

HAZARD - Hazard_On_Road_Constru

Select Hour:

0 12 23

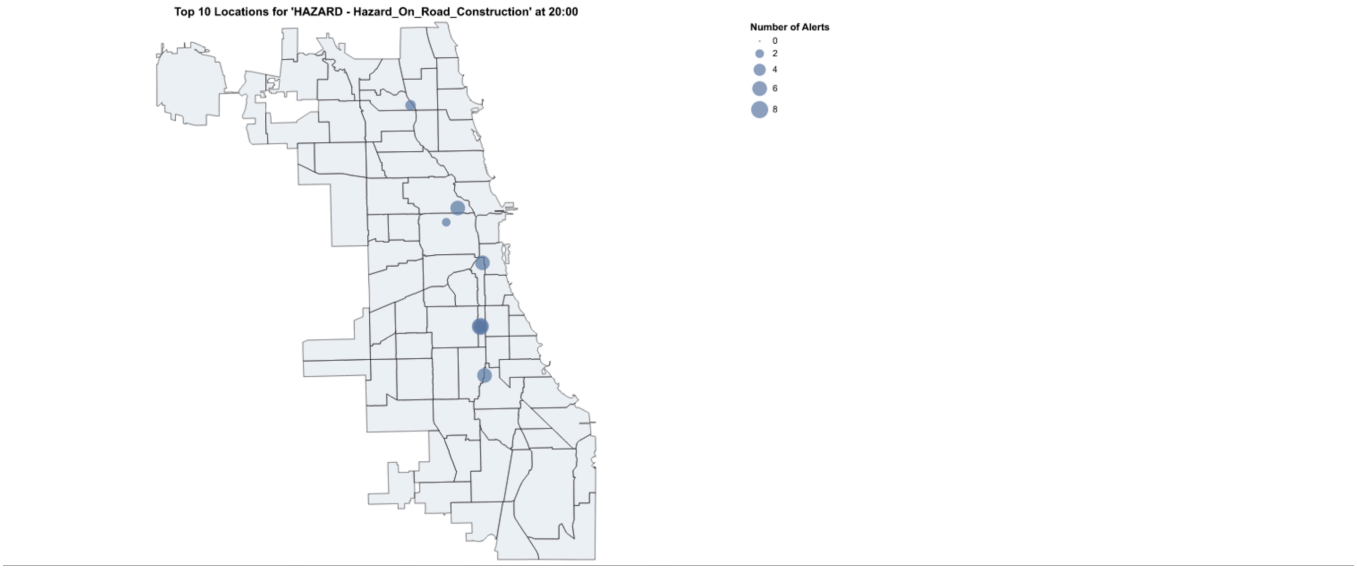


Select Alert Type and Subtype:

HAZARD - Hazard_On_Road_Constru

Select Hour:

0 20 23



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

1.

a. In this case, collapsing the dataset by range of hours is a reasonable choice, as it reduces computational overhead, improves performance, and matches the requirements of the application. However, retaining the original dataset for flexibility in future use cases is also recommended.

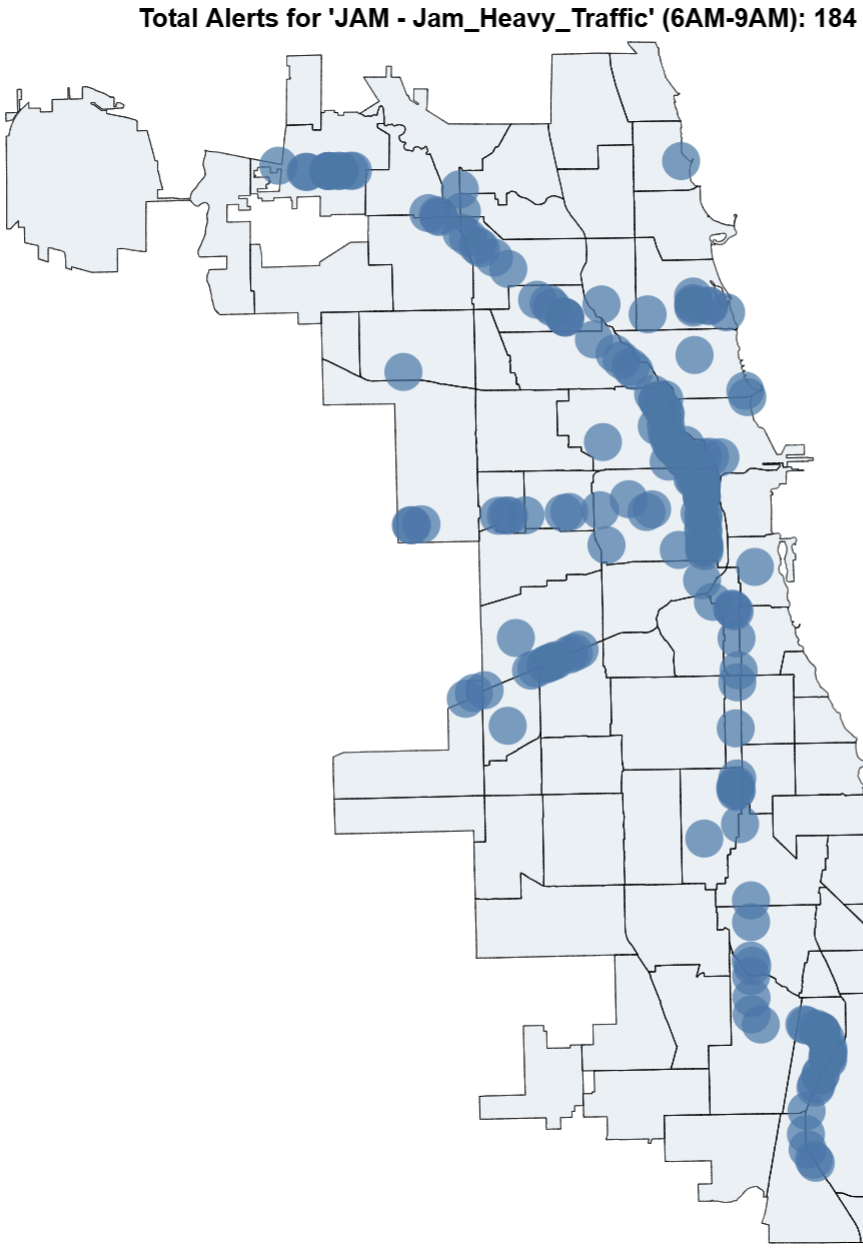
b.

```
import altair as alt
import pandas as pd
import json

# Load the GeoJSON file for Chicago neighborhood boundaries
geojson_path = r"C:\Users\Pei-Chin\Dropbox (1)\Boundaries - Neighborhoods.geojson"
with open(geojson_path) as f:
    chicago_geojson = json.load(f)

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

# Load the dataset
collapsed_data_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\top_alerts_map_byhour
merged_data = pd.read_csv(collapsed_data_path)

# Ensure 'hour' column is properly extracted
merged_data['hour'] = pd.to_datetime(merged_data['ts']).dt.hour

# Create 'type_subtype_combo' column by combining 'type' and 'subtype'
merged_data['type_subtype_combo'] = merged_data['type'] + " - " + merged_data['subtype']

# Correctly filter for 'JAM - Jam_Heavy_Traffic' instead of 'Jam - Heavy Traffic'
combo_filtered_data = merged_data[
    (merged_data['type_subtype_combo'] == 'JAM - Jam_Heavy_Traffic') &
    (merged_data['hour'] >= 6) &
    (merged_data['hour'] < 9)
]

print("Data for 'JAM - Jam_Heavy_Traffic' in 6AM to 9AM range:")
print(combo_filtered_data.head())

# Group data by location and calculate total alerts across all locations
grouped_data = (
    combo_filtered_data.groupby(['latitude', 'longitude'])
    .size()
    .reset_index(name='alert_count')
)

# Calculate the total number of alerts in the filtered data
total_alerts = grouped_data['alert_count'].sum()

# Debug grouped data
print("Grouped Data:")
print(grouped_data)
```

```
# Create the map layer
map_layer = alt.Chart(geo_data).mark_geoshape(
    fillOpacity=0.1,
    stroke="black",
    strokeWidth=0.5
).properties(
    width=800,
    height=600
).project("mercator")

# Create the scatter plot layer to display all locations with alert counts
scatter_plot = alt.Chart(grouped_data).mark_circle(size=200).encode(
    longitude='longitude:Q',
    latitude='latitude:Q',
    size=alt.Size('alert_count:Q', title="Number of Alerts"),
    tooltip=['latitude', 'longitude', 'alert_count']
).properties(
    title=f"Total Alerts for 'JAM - Jam_Heavy_Traffic' (6AM-9AM): {total_alerts}"
)

# Combine map and scatter plot
combined_chart = map_layer + scatter_plot

# Save the chart as a static file
output_file = "total_alerts_JAM_Jam_Heavy_Traffic_6AM_9AM.html"
combined_chart.save(output_file)
print(f"Chart saved to {output_file}")

# Displaying combined_chart (if in an interactive environment)
combined_chart.display()
```

```
Data for 'JAM - Jam_Heavy_Traffic' in 6AM to 9AM range:
            city  confidence  nThumbsUp         street  \
1026  Chicago, IL           0        NaN  N Ashland Ave
2753  Chicago, IL           0        NaN        I-290 W
2758  Chicago, IL           0        NaN        I-290 W
2963  Chicago, IL           0        NaN         I-90 W
3182  Chicago, IL           0        NaN  W Chicago Ave

                                   uuid country type            subtype  \
1026  98f3126b-cc46-4e0d-bf62-fb0bfda3c245      US  JAM  Jam_Heavy_Traffic
2753  1e87df19-a540-426c-b0c9-5eba881b3d86      US  JAM  Jam_Heavy_Traffic
2758  ee4d1029-2bc7-412c-8519-66eace57654f      US  JAM  Jam_Heavy_Traffic
2963  042fe299-b20e-4a48-95fa-40ded265d065      US  JAM  Jam_Heavy_Traffic
3182  7666ac53-8963-4575-b201-da2d5515294a      US  JAM  Jam_Heavy_Traffic

      roadType  reliability  ...                       ts  \
1026         7            5  ...  2024-02-04 07:02:44 UTC
2753         3            5  ...  2024-09-14 06:21:58 UTC
2758         3            5  ...  2024-09-14 06:10:39 UTC
2963         3            5  ...  2024-09-14 07:50:11 UTC
3182         7            5  ...  2024-09-14 06:31:04 UTC
```

```
                              geo                       geoWKT updated_type  \
1026  POINT(-87.668577 41.937412)  Point(-87.668577 41.937412)  Traffic Jam
2753  POINT(-87.688705 41.875751)  Point(-87.688705 41.875751)  Traffic Jam
2758  POINT(-87.704087 41.874886)  Point(-87.704087 41.874886)  Traffic Jam
2963   POINT(-87.64612 41.886456)   Point(-87.64612 41.886456)  Traffic Jam
3182   POINT(-87.652595 41.89631)   Point(-87.652595 41.89631)  Traffic Jam

        updated_subtype updated_subsubtype   latitude  longitude  hour  \
1026  Traffic Condition      Heavy Traffic  41.937412 -87.668577     7
2753  Traffic Condition      Heavy Traffic  41.875751 -87.688705     6
2758  Traffic Condition      Heavy Traffic  41.874886 -87.704087     6
2963  Traffic Condition      Heavy Traffic  41.886456 -87.646120     7
3182  Traffic Condition      Heavy Traffic  41.896310 -87.652595     6

           type_subtype_combo
1026  JAM - Jam_Heavy_Traffic
2753  JAM - Jam_Heavy_Traffic
2758  JAM - Jam_Heavy_Traffic
2963  JAM - Jam_Heavy_Traffic
3182  JAM - Jam_Heavy_Traffic

[5 rows x 22 columns]
Grouped Data:
      latitude  longitude  alert_count
0    41.669883 -87.597434            1
1    41.670847 -87.598263            1
2    41.674129 -87.600886            1
3    41.678927 -87.601592            1
4    41.686259 -87.601339            1
..         ...        ...          ...
179  41.982526 -87.798815            1
180  41.982559 -87.794625            1
181  41.982572 -87.793265            1
182  41.984152 -87.824820            1
183  41.985618 -87.654416            1

[184 rows x 3 columns]
Chart saved to total_alerts_JAM_Jam_Heavy_Traffic_6AM_9AM.html
```



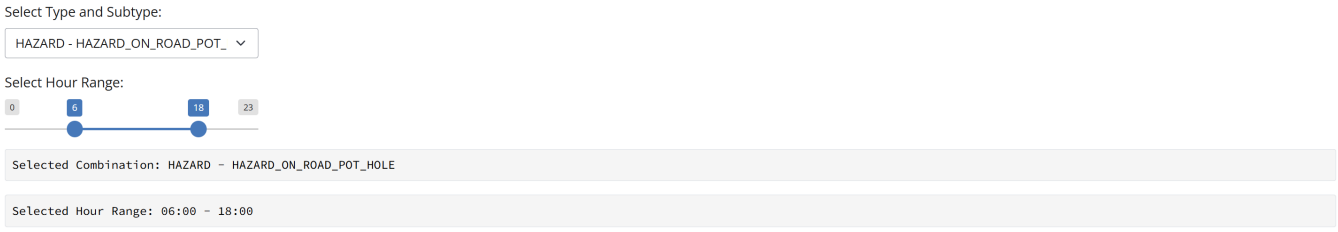
2.

a.

```
from shiny import App, ui, render
import pandas as pd

# Load dataset
data_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\waze_data\waze_data.csv"
waze_data = pd.read_csv(data_path)

# Create a new column combining 'type' and 'subtype'
waze_data['type_subtype_combo'] = waze_data['type'] + " - " + waze_data['subtype']

# Extract unique type-subtype combinations for the dropdown
unique_combinations = waze_data['type_subtype_combo'].drop_duplicates().tolist()

# UI definition
app_ui = ui.page_fluid(
```

```
    ui.panel_title("Dropdown and Hour Range Slider"),
    ui.input_select(
        "type_subtype",
        "Select Type and Subtype:",
        choices=unique_combinations,
        selected=unique_combinations[0]
    ),
    ui.input_slider(
        "hour_range",
        "Select Hour Range:",
        min=0,
        max=23,
        value=(6, 9)  # Default range
    ),
    ui.output_text_verbatim("selected_combination"),
    ui.output_text_verbatim("selected_hour_range")
)

# Server logic
def server(input, output, session):
    @output
    @render.text
    def selected_combination():
        # Display the selected type and subtype combination
        return f"Selected Combination: {input.type_subtype()}"

    @output
    @render.text
    def selected_hour_range():
        # Display the selected hour range
        hour_min, hour_max = input.hour_range()
        return f"Selected Hour Range: {hour_min:02}:00 - {hour_max:02}:00"

# Create the app
app = App(app_ui, server)
```

Dropdown and Hour Range Slider



3a Task Screenshot

b.

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

```
import os

# File paths
geojson_path = r"C:\Users\Pei-Chin\Dropbox (1)\Boundaries - Neighborhoods.geojson"
data_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\top_alerts_map_byhour\original_

# Load the merged dataset
merged_data = pd.read_csv(data_path)

# Preprocessing: Ensure 'type' and 'subtype' columns are present
if 'type' not in merged_data.columns or 'subtype' not in merged_data.columns:
    raise KeyError("Columns 'type' and 'subtype' are missing in the dataset.")

# Create 'type_subtype_combo' column
merged_data['type_subtype_combo'] = merged_data['type'].astype(str) + " - " + merged_data['sub

# Ensure 'hour' column is numeric
merged_data['hour'] = pd.to_datetime(merged_data['ts']).dt.hour

# Ensure latitude and longitude have no missing values
merged_data = merged_data.dropna(subset=['latitude', 'longitude'])

# Get unique combinations of 'type' and 'subtype'
unique_combos = merged_data['type_subtype_combo'].unique().tolist()

# Load GeoJSON file for the map
with open(geojson_path) as f:
    chicago_geojson = json.load(f)

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

# Define the UI
app_ui = ui.page_fluid(
    ui.h3("Dynamic Traffic Plot"),
    ui.input_select(
        "type_subtype",
        "Select Alert Type and Subtype:",
        choices=unique_combos,
        selected=unique_combos[0]
    ),
    ui.input_slider(
        "hour_range",
        "Select Hour Range:",
        min=0,
        max=23,
        value=(6, 9),  # Default range: 6AM to 9AM
        step=1
    ),
    ui.output_image("traffic_plot")  # Output for the scatter plot as an image
)

# Define the server logic
def server(input, output, session):
    @output
```

```
    @render.image
    def traffic_plot():
        try:
            # Get user inputs
            selected_combo = input.type_subtype()
            hour_min, hour_max = input.hour_range()

            # Filter data for the selected combination and hour range
            filtered_data = merged_data[
                (merged_data['type_subtype_combo'] == selected_combo) &
                (merged_data['hour'] >= hour_min) &
                (merged_data['hour'] <= hour_max)
            ]

            # If no data is available, return None
            if filtered_data.empty:
                print(f"No data available for combo '{selected_combo}' in range {hour_min}-{ho
                return None

            # Group by latitude, longitude, and count the alerts
            grouped_data = (
                filtered_data.groupby(['latitude', 'longitude'], as_index=False)
                .agg(alert_count=('type_subtype_combo', 'count'))  # Count occurrences of type
            )

            # Ensure alert_count is an integer
            grouped_data['alert_count'] = grouped_data['alert_count'].astype(int)

            # Calculate total alerts for the selected period
            total_alerts = grouped_data['alert_count'].sum()

            # Debug grouped data
            print(f"Grouped Data for {selected_combo} ({hour_min}-{hour_max}):")
            print(grouped_data)

            # Create the map layer
            map_layer = alt.Chart(geo_data).mark_geoshape(
                fillOpacity=0.1,
                stroke="black",
                strokeWidth=0.5
            ).properties(
                width=800,
                height=600
            ).project("mercator")

            # Create scatter plot layer
            scatter_plot = alt.Chart(grouped_data).mark_circle().encode(
                longitude='longitude:Q',
                latitude='latitude:Q',
                size=alt.Size('alert_count:Q', title="Number of Alerts",
                              scale=alt.Scale(domain=[1, grouped_data['alert_count'].max()], r
                tooltip=['latitude', 'longitude', 'alert_count']
            ).properties(
                title=f"Total Alerts for '{selected_combo}' ({hour_min}-{hour_max}): {total_al
```

```
            )

            # Combine map and scatter plot
            combined_chart = map_layer + scatter_plot

            # Generate a unique file name based on selected combo and time range
            file_name = f"temp_alert_map_{selected_combo.replace(' ', '_')}_{hour_min}_{hour_m
            file_path = os.path.abspath(file_name)

            # Save the chart to a PNG file
            combined_chart.save(file_path, format="png")

            # Return the image path for rendering
            if os.path.exists(file_path):
                return {"src": file_path, "alt": f"Traffic Plot for {selected_combo} ({hour_mi
            else:
                print("File not found after saving.")
                return None

        except Exception as e:
            print(f"Error generating plot: {e}")
            return None

# Create the app
app = App(app_ui, server)
```

3.

a.

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

# File paths
geojson_path = r"C:\Users\Pei-Chin\Dropbox (1)\Boundaries - Neighborhoods.geojson"
data_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\top_alerts_map_byhour\original_

# Load the merged dataset
merged_data = pd.read_csv(data_path)

# Preprocessing: Ensure 'type' and 'subtype' columns are present
if 'type' not in merged_data.columns or 'subtype' not in merged_data.columns:
    raise KeyError("Columns 'type' and 'subtype' are missing in the dataset.")

# Create 'type_subtype_combo' column
merged_data['type_subtype_combo'] = merged_data['type'].astype(str) + " - " + merged_data['sub

# Ensure 'hour' column is numeric
merged_data['hour'] = pd.to_datetime(merged_data['ts']).dt.hour
```

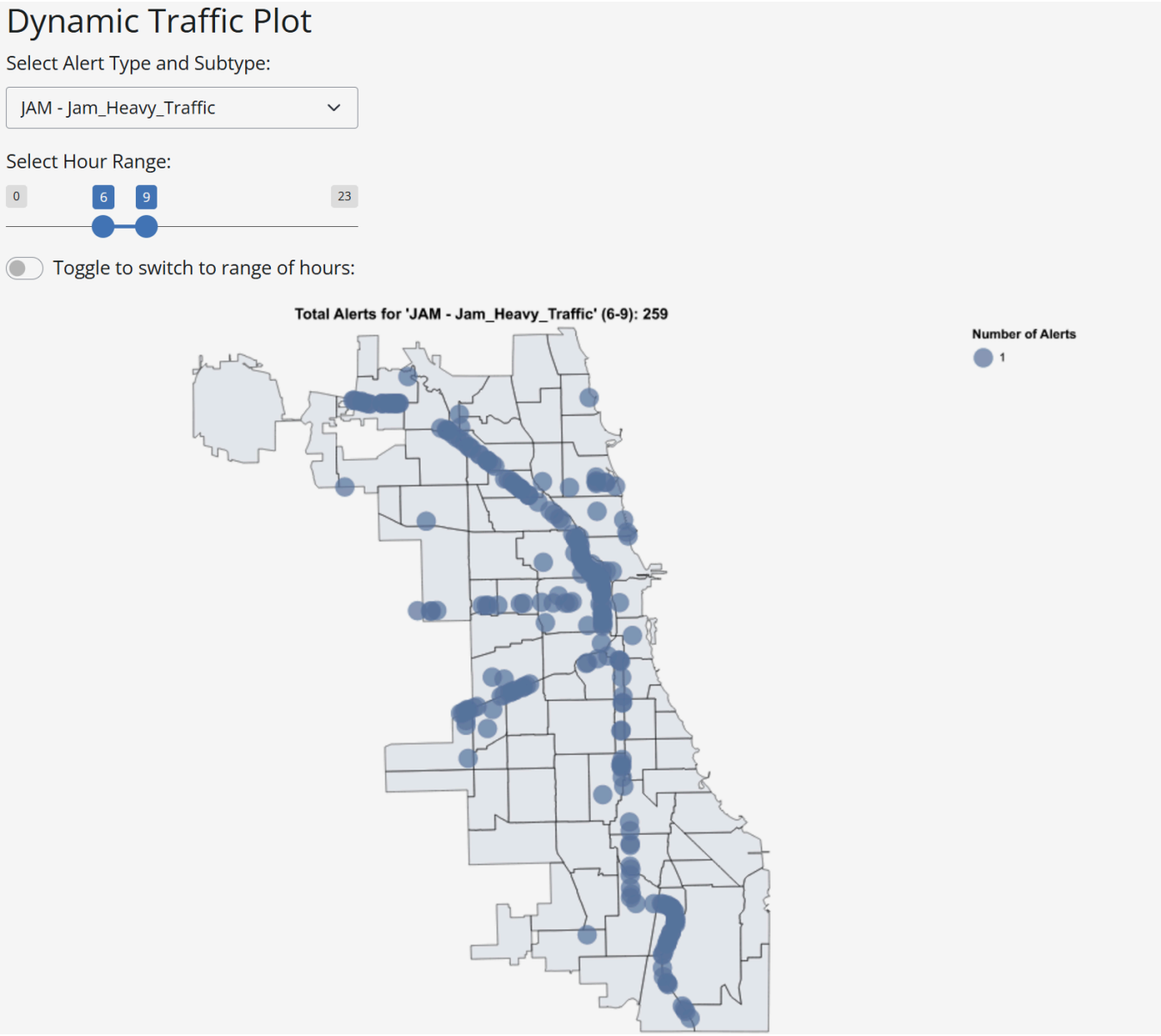
```
# Ensure latitude and longitude have no missing values
merged_data = merged_data.dropna(subset=['latitude', 'longitude'])

# Get unique combinations of 'type' and 'subtype'
unique_combos = merged_data['type_subtype_combo'].unique().tolist()

# Load GeoJSON file for the map
with open(geojson_path) as f:
    chicago_geojson = json.load(f)

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

# Define the UI
app_ui = ui.page_fluid(
    ui.h3("Dynamic Traffic Plot"),
    ui.input_select(
        "type_subtype",
        "Select Alert Type and Subtype:",
        choices=unique_combos,
        selected=unique_combos[0]
    ),
    ui.input_slider(
        "hour_range",
        "Select Hour Range:",
        min=0,
        max=23,
        value=(6, 9),  # Default range: 6AM to 9AM
        step=1
    ),
    ui.input_switch(
        "switch_button",
        "Toggle to switch to range of hours:",
        value=False  # Default is False (off)
    ),
    ui.output_image("traffic_plot")  # Output for the scatter plot as an image
)

# Define the server logic
def server(input, output, session):
    @output
    @render.image
    def traffic_plot():
        try:
            # Get user inputs
            selected_combo = input.type_subtype()
            hour_min, hour_max = input.hour_range()
            switch_state = input.switch_button()  # Retrieve the switch state (True/False)

            # Debugging switch state
            print(f"Switch state: {switch_state}")

            # Filter data for the selected combination and hour range
            filtered_data = merged_data[
                (merged_data['type_subtype_combo'] == selected_combo) &
```

```
        (merged_data['hour'] >= hour_min) &
        (merged_data['hour'] <= hour_max)
    ]

    # If no data is available, return None
    if filtered_data.empty:
        print(f"No data available for combo '{selected_combo}' in range {hour_min}-{ho
        return None

    # Group by latitude, longitude, and count the alerts
    grouped_data = (
        filtered_data.groupby(['latitude', 'longitude'], as_index=False)
        .agg(alert_count=('type_subtype_combo', 'count'))  # Count occurrences of type
    )

    # Ensure alert_count is an integer
    grouped_data['alert_count'] = grouped_data['alert_count'].astype(int)

    # Calculate total alerts for the selected period
    total_alerts = grouped_data['alert_count'].sum()

    # Create the map layer
    map_layer = alt.Chart(geo_data).mark_geoshape(
        fillOpacity=0.1,
        stroke="black",
        strokeWidth=0.5
    ).properties(
        width=800,
        height=600
    ).project("mercator")

    # Create scatter plot layer
    scatter_plot = alt.Chart(grouped_data).mark_circle().encode(
        longitude='longitude:Q',
        latitude='latitude:Q',
        size=alt.Size('alert_count:Q', title="Number of Alerts",
                      scale=alt.Scale(domain=[1, grouped_data['alert_count'].max()], r
        tooltip=['latitude', 'longitude', 'alert_count']
    ).properties(
        title=f"Total Alerts for '{selected_combo}' ({hour_min}-{hour_max}): {total_al
    )

    # Combine map and scatter plot
    combined_chart = map_layer + scatter_plot

    # Generate a unique file name based on selected combo and time range
    file_name = f"temp_alert_map_{selected_combo.replace(' ', '_')}_{hour_min}_{hour_m
    file_path = os.path.abspath(file_name)

    # Save the chart to a PNG file
    combined_chart.save(file_path, format="png")

    # Return the image path for rendering
    if os.path.exists(file_path):
```

```
                return {"src": file_path, "alt": f"Traffic Plot for {selected_combo} ({hour_mi
            else:
                print("File not found after saving.")
                return None

        except Exception as e:
            print(f"Error generating plot: {e}")
            return None

# Create the app
app = App(app_ui, server)
```

1 Answering the questions:

The switch_button has two possible values: True when the switch is toggled on, and False when it is toggled off (default). In the user interface, the ui.input_switch() function is used to add this toggle button. On the server side, the state of the switch can be accessed via input.switch_button(), allowing the application to respond dynamically to the user's selection. The state of the switch represents a logical value, where True indicates the switch is active, and False indicates it is inactive.



b.

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

# File paths
geojson_path = r"C:\Users\Pei-Chin\Dropbox (1)\Boundaries - Neighborhoods.geojson"
data_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\top_alerts_map_byhour\original_

# Load the merged dataset
merged_data = pd.read_csv(data_path)

# Preprocessing: Ensure 'type' and 'subtype' columns are present
if 'type' not in merged_data.columns or 'subtype' not in merged_data.columns:
    raise KeyError("Columns 'type' and 'subtype' are missing in the dataset.")

# Create 'type_subtype_combo' column
merged_data['type_subtype_combo'] = merged_data['type'].astype(str) + " - " + merged_data['sub

# Ensure 'hour' column is numeric
merged_data['hour'] = pd.to_datetime(merged_data['ts']).dt.hour

# Ensure latitude and longitude have no missing values
merged_data = merged_data.dropna(subset=['latitude', 'longitude'])

# Get unique combinations of 'type' and 'subtype'
unique_combos = merged_data['type_subtype_combo'].unique().tolist()

# Load GeoJSON file for the map
with open(geojson_path) as f:
    chicago_geojson = json.load(f)

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

# Define the UI
app_ui = ui.page_fluid(
    ui.h3("Dynamic Traffic Plot"),
    ui.input_select(
        "type_subtype",
        "Select Alert Type and Subtype:",
        choices=unique_combos,
        selected=unique_combos[0]
    ),
    ui.input_switch(
        "switch_button",
        "Toggle to switch to range of hours:",
        value=False  # Default is False (off)
    ),
    ui.output_ui("hour_slider"),  # Output placeholder for dynamic slider
    ui.output_image("traffic_plot")  # Output for the scatter plot as an image
)
```

```
# Define the server logic
def server(input, output, session):
    @output
    @render.ui
    def hour_slider():
        """Render a single hour slider or a range slider based on switch state."""
        if input.switch_button():
            # Show single-hour slider when switch is ON
            return ui.input_slider(
                "single_hour",
                "Select Single Hour:",
                min=0,
                max=23,
                value=6,  # Default hour
                step=1
            )
        else:
            # Show range slider when switch is OFF
            return ui.input_slider(
                "hour_range",
                "Select Hour Range:",
                min=0,
                max=23,
                value=(6, 9),  # Default range
                step=1
            )

    @output
    @render.image
    def traffic_plot():
        try:
            # Get user inputs
            selected_combo = input.type_subtype()
            switch_state = input.switch_button()  # Retrieve the switch state (True/False)

            # Determine the selected hours based on switch state
            if switch_state:
                hour_min = hour_max = input.single_hour()  # Single hour
            else:
                hour_min, hour_max = input.hour_range()  # Hour range

            # Filter data for the selected combination and hour range
            filtered_data = merged_data[
                (merged_data['type_subtype_combo'] == selected_combo) &
                (merged_data['hour'] >= hour_min) &
                (merged_data['hour'] <= hour_max)
            ]

            # If no data is available, return None
            if filtered_data.empty:
                print(f"No data available for combo '{selected_combo}' in range {hour_min}-{ho
                return None
```

```
            # Group by latitude, longitude, and count the alerts
            grouped_data = (
                filtered_data.groupby(['latitude', 'longitude'], as_index=False)
                .agg(alert_count=('type_subtype_combo', 'count'))  # Count occurrences of type
            )

            # Ensure alert_count is an integer
            grouped_data['alert_count'] = grouped_data['alert_count'].astype(int)

            # Calculate total alerts for the selected period
            total_alerts = grouped_data['alert_count'].sum()

            # Create the map layer
            map_layer = alt.Chart(geo_data).mark_geoshape(
                fillOpacity=0.1,
                stroke="black",
                strokeWidth=0.5
            ).properties(
                width=800,
                height=600
            ).project("mercator")

            # Create scatter plot layer
            scatter_plot = alt.Chart(grouped_data).mark_circle().encode(
                longitude='longitude:Q',
                latitude='latitude:Q',
                size=alt.Size('alert_count:Q', title="Number of Alerts",
                              scale=alt.Scale(domain=[1, grouped_data['alert_count'].max()], r
                tooltip=['latitude', 'longitude', 'alert_count']
            ).properties(
                title=f"Total Alerts for '{selected_combo}' ({hour_min}-{hour_max}): {total_al
            )

            # Combine map and scatter plot
            combined_chart = map_layer + scatter_plot

            # Generate a unique file name based on selected combo and time range
            file_name = f"temp_alert_map_{selected_combo.replace(' ', '_')}_{hour_min}_{hour_m
            file_path = os.path.abspath(file_name)

            # Save the chart to a PNG file
            combined_chart.save(file_path, format="png")

            # Return the image path for rendering
            if os.path.exists(file_path):
                return {"src": file_path, "alt": f"Traffic Plot for {selected_combo} ({hour_mi
            else:
                print("File not found after saving.")
                return None

        except Exception as e:
            print(f"Error generating plot: {e}")
            return None
```

```
# Create the app
app = App(app_ui, server)
```

Dynamic Traffic Plot

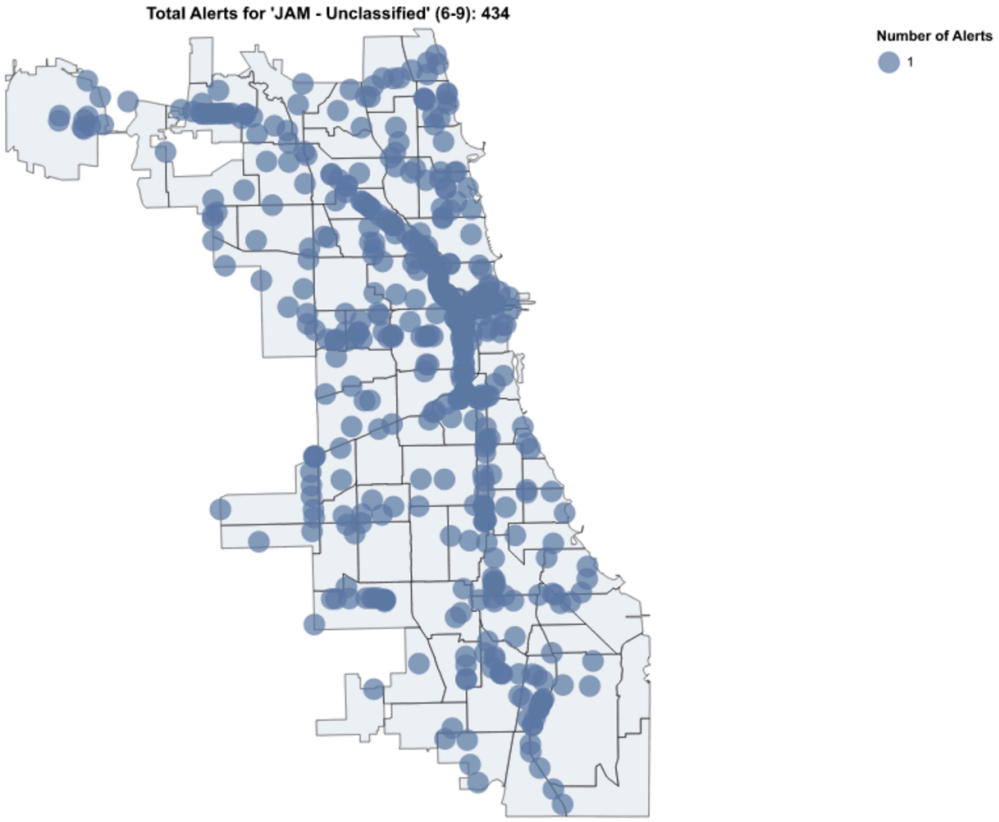
Select Alert Type and Subtype:

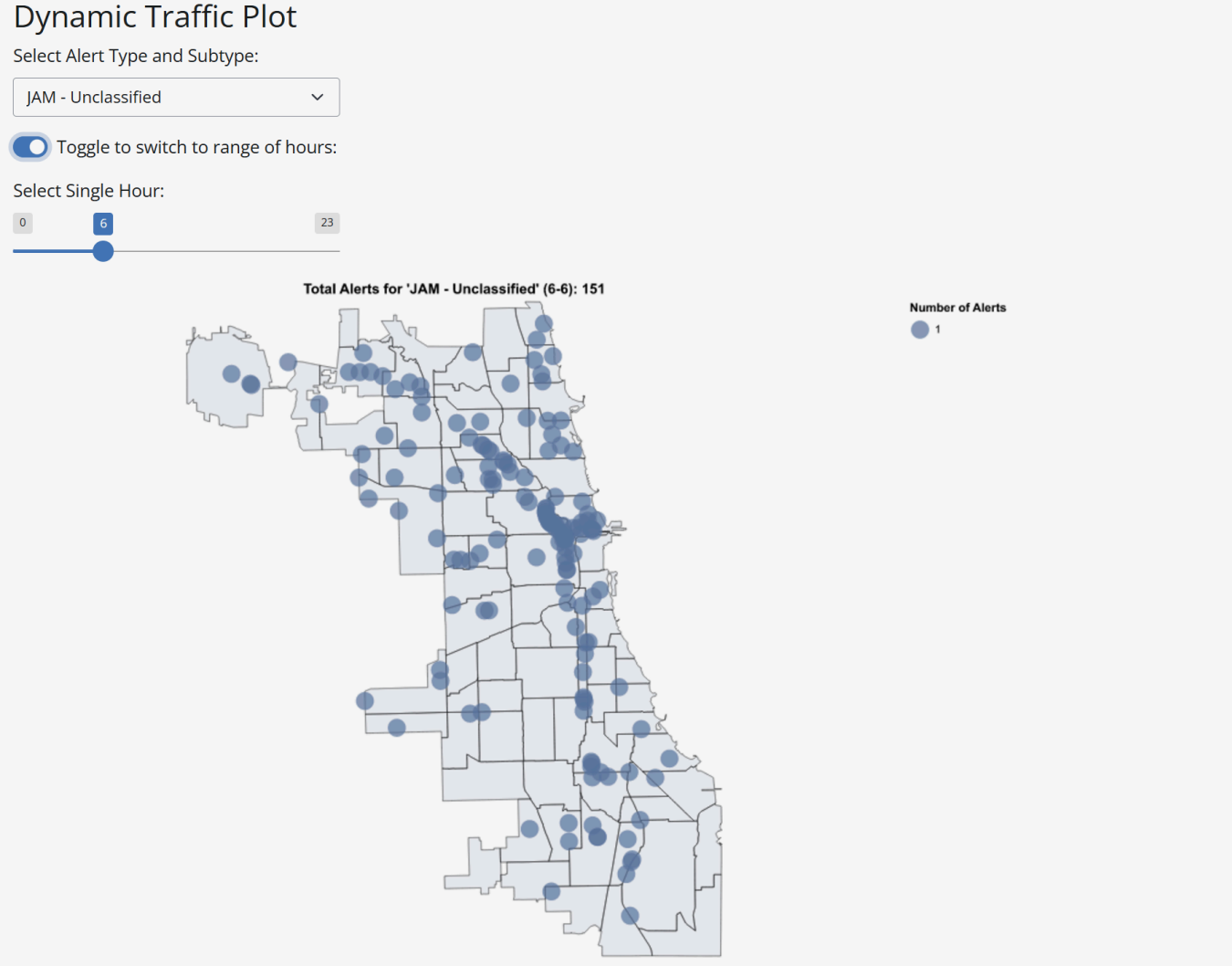
JAM - Unclassified

Toggle to switch to range of hours:

Select Hour Range:

0 6 9 23





C.

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

# File paths
geojson_path = r"C:\Users\Pei-Chin\Dropbox (1)\Boundaries - Neighborhoods.geojson"
data_path = r"C:\Users\Pei-Chin\Documents\GitHub\Problem-Set-6\top_alerts_map_byhour\original_

# Load the merged dataset
merged_data = pd.read_csv(data_path)

# Preprocessing: Ensure 'type' and 'subtype' columns are present
if 'type' not in merged_data.columns or 'subtype' not in merged_data.columns:
    raise KeyError("Columns 'type' and 'subtype' are missing in the dataset.")

# Create 'type_subtype_combo' column
merged_data['type_subtype_combo'] = merged_data['type'].astype(str) + " - " + merged_data['sub

# Ensure 'hour' column is numeric
merged_data['hour'] = pd.to_datetime(merged_data['ts']).dt.hour
```

```
# Ensure latitude and longitude have no missing values
merged_data = merged_data.dropna(subset=['latitude', 'longitude'])

# Get unique combinations of 'type' and 'subtype'
unique_combos = merged_data['type_subtype_combo'].unique().tolist()

# Load GeoJSON file for the map
with open(geojson_path) as f:
    chicago_geojson = json.load(f)

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

# Define the UI
app_ui = ui.page_fluid(
    ui.h3("Dynamic Traffic Plot"),
    ui.input_select(
        "type_subtype",
        "Select Alert Type and Subtype:",
        choices=unique_combos,
        selected=unique_combos[0]
    ),
    ui.input_switch(
        "switch_button",
        "Toggle to switch to range of hours:",
        value=False  # Default is False (off)
    ),
    ui.output_ui("hour_slider"),  # Output placeholder for dynamic slider
    ui.output_image("traffic_plot")  # Output for the scatter plot as an image
)

# Define the server logic
def server(input, output, session):
    @output
    @render.ui
    def hour_slider():
        """Render a single hour slider or a range slider based on switch state."""
        if input.switch_button():
            # Show single-hour slider when switch is ON
            return ui.input_slider(
                "single_hour",
                "Select Single Hour:",
                min=0,
                max=23,
                value=6,  # Default hour
                step=1
            )
        else:
            # Show range slider when switch is OFF
            return ui.input_slider(
                "hour_range",
                "Select Hour Range:",
                min=0,
                max=23,
                value=(6, 9),  # Default range
```

```
            step=1
        )

    @output
    @render.image
    def traffic_plot():
        try:
            # Get user inputs
            selected_combo = input.type_subtype()
            switch_state = input.switch_button()  # Retrieve the switch state (True/False)

            # Determine the selected hours based on switch state
            if switch_state:
                hour_min = hour_max = input.single_hour()  # Single hour
            else:
                hour_min, hour_max = input.hour_range()  # Hour range

            # Filter data for the selected combination and hour range
            filtered_data = merged_data[
                (merged_data['type_subtype_combo'] == selected_combo) &
                (merged_data['hour'] >= hour_min) &
                (merged_data['hour'] <= hour_max)
            ]

            # If no data is available, return None
            if filtered_data.empty:
                print(f"No data available for combo '{selected_combo}' in range {hour_min}-{ho
                return None

            # Group by latitude, longitude, and count the alerts
            grouped_data = (
                filtered_data.groupby(['latitude', 'longitude'], as_index=False)
                .agg(alert_count=('type_subtype_combo', 'count'))  # Count occurrences of type
            )

            # Ensure alert_count is an integer
            grouped_data['alert_count'] = grouped_data['alert_count'].astype(int)

            # Calculate total alerts for the selected period
            total_alerts = grouped_data['alert_count'].sum()

            # Create the map layer
            map_layer = alt.Chart(geo_data).mark_geoshape(
                fillOpacity=0.1,
                stroke="black",
                strokeWidth=0.5
            ).properties(
                width=800,
                height=600
            ).project("mercator")

            # Create scatter plot layer
            scatter_plot = alt.Chart(grouped_data).mark_circle().encode(
                longitude='longitude:Q',
```

```
                latitude='latitude:Q',
                size=alt.Size('alert_count:Q', title="Number of Alerts",
                              scale=alt.Scale(domain=[1, grouped_data['alert_count'].max()], r
                tooltip=['latitude', 'longitude', 'alert_count']
            ).properties(
                title=f"Total Alerts for '{selected_combo}' ({hour_min}-{hour_max}): {total_al
            )

            # Combine map and scatter plot
            combined_chart = map_layer + scatter_plot

            # Generate a unique file name based on selected combo and time range
            file_name = f"temp_alert_map_{selected_combo.replace(' ', '_')}_{hour_min}_{hour_m
            file_path = os.path.abspath(file_name)

            # Save the chart to a PNG file
            combined_chart.save(file_path, format="png")

            # Return the image path for rendering
            if os.path.exists(file_path):
                return {"src": file_path, "alt": f"Traffic Plot for {selected_combo} ({hour_mi
            else:
                print("File not found after saving.")
                return None

        except Exception as e:
            print(f"Error generating plot: {e}")
            return None

# Create the app
app = App(app_ui, server)
```

Dynamic Traffic Plot

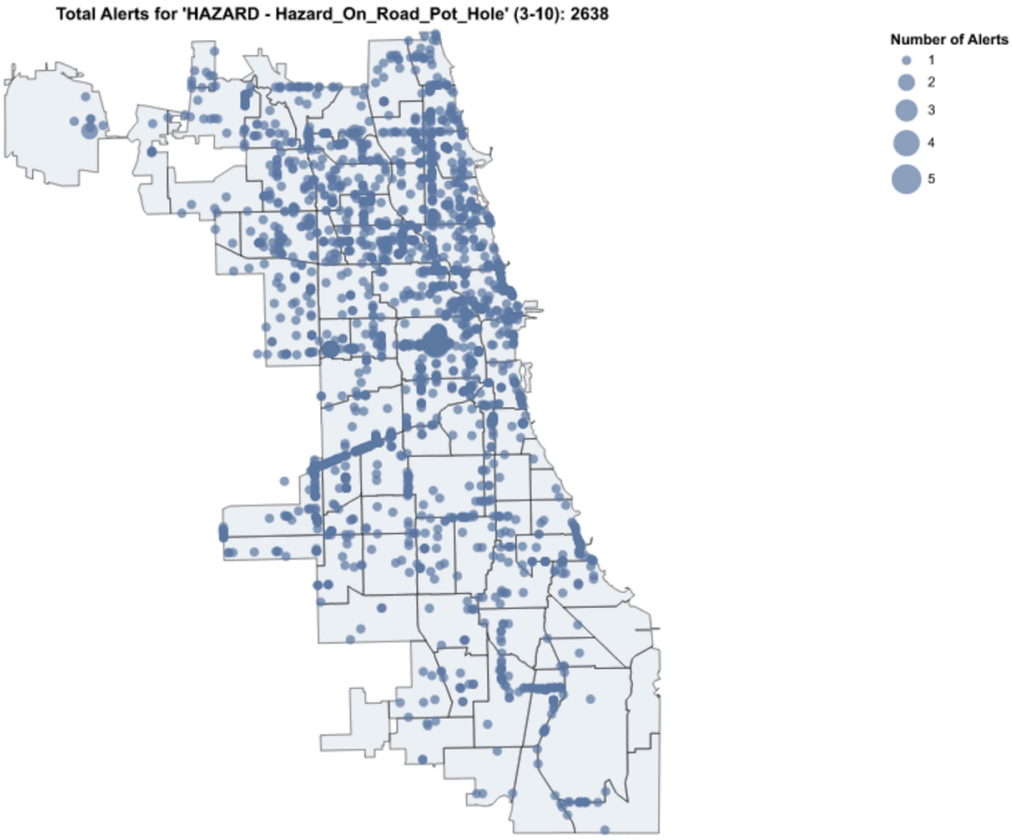
Select Alert Type and Subtype:

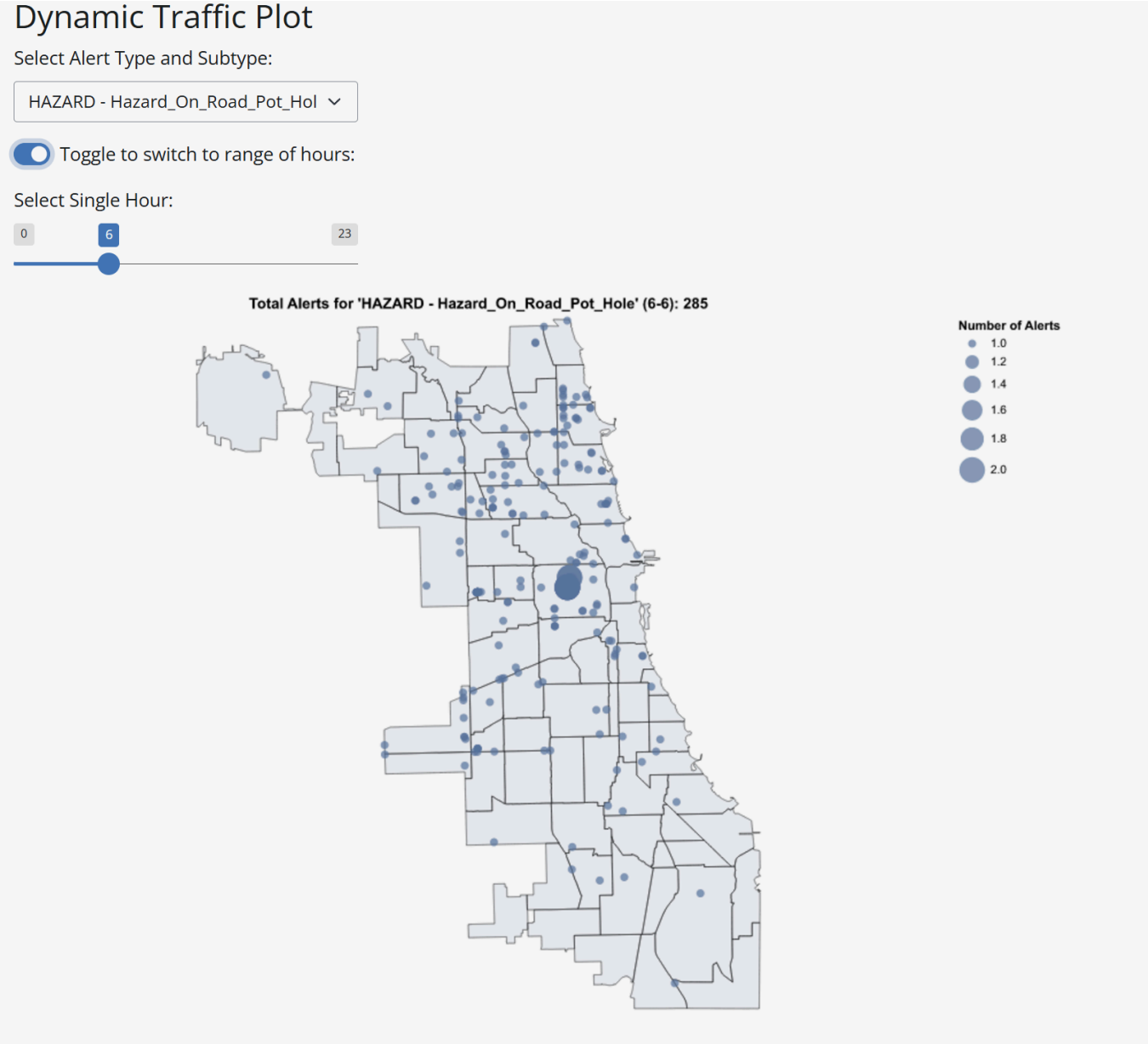
HAZARD - Hazard_On_Road_Pot_Hol

Toggle to switch to range of hours:

Select Hour Range:

0 3 10 23





d. 2 Answering the questions:

-Categorize Time: Create a time_period column for categorizing the time into meaningful periods (e.g., Morning, Afternoon). -Add UI Control: Add a new input control for selecting time periods. -Update Data Filtering: Filter data based on the selected time period(s). -Add Color Coding: Differentiate points on the map by time period using color. -Update Tooltip and Legend: Include time_period in the tooltip and add a color legend to the plot.