Problem Set 6 - Waze Shiny Dashboard

Boya Lin

2024-11-19

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: **BL**
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **BL** (2 point)
3. Late coins used this pset: **1** Late coins left after submission: **0**
4. Before starting the problem set, make sure to read and agree to the terms of data usage for the Waze data here.
5. Knit your `ps6.qmd` as a pdf document and name it `ps6.pdf`.
6. 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!*

```
/Users/boyalin/Documents/GitHub/ppha30538_ps/problem-set-6-boyalin/.venv/lib/python3.9/site-p
NotOpenSSLWarning:

urllib3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' module is
compiled with 'LibreSSL 2.8.3'. See:
https://github.com/urllib3/urllib3/issues/3020
```

Background

Data Download and Exploration (20 points)

1.

```
# unzip the file
with zipfile.ZipFile('waze_data.zip', 'r') as zip_ref:
    zip_ref.extractall('waze_data')

# load the sample dataset and report in general way
waze_data_sample = pd.read_csv('waze_data/waze_data_sample.csv', index_col=0)
print(f'Variable names and general
↪  dtypes:\n{waze_data_sample.dtypes.head()}')

def map_to_altair_dtype(dtype):
    """function to change Pandas dtypes to Altair syntax"""
    if dtype in ['int64', 'float64']:
        return 'Quantitative'
    elif dtype == 'object':
        return 'Nominal'
    else:
        return 'Other'

altair_dtypes = waze_data_sample.dtypes.apply(map_to_altair_dtype).to_dict()

# exclude specified columns
excluded_columns = ['ts', 'geo', 'geoWKT']
filtered_variables = {var: dtype for var,
                      dtype in altair_dtypes.items() if var not in
↪  excluded_columns}
```

```
# create dataframe
altair_dtype_df = pd.DataFrame(list(filtered_variables.items()), columns=[
    'Variable', 'Altair Data Type'])
# generate markdown table & display
markdown_table = altair_dtype_df.to_markdown(index=False)
print(markdown_table)
# ChatGPT reference for "upzip file directly in Python"
# ChatGPT reference for "exclude specific columns when reporting dtypes"
# ChatGPT reference for "make markdown table for the results"
```

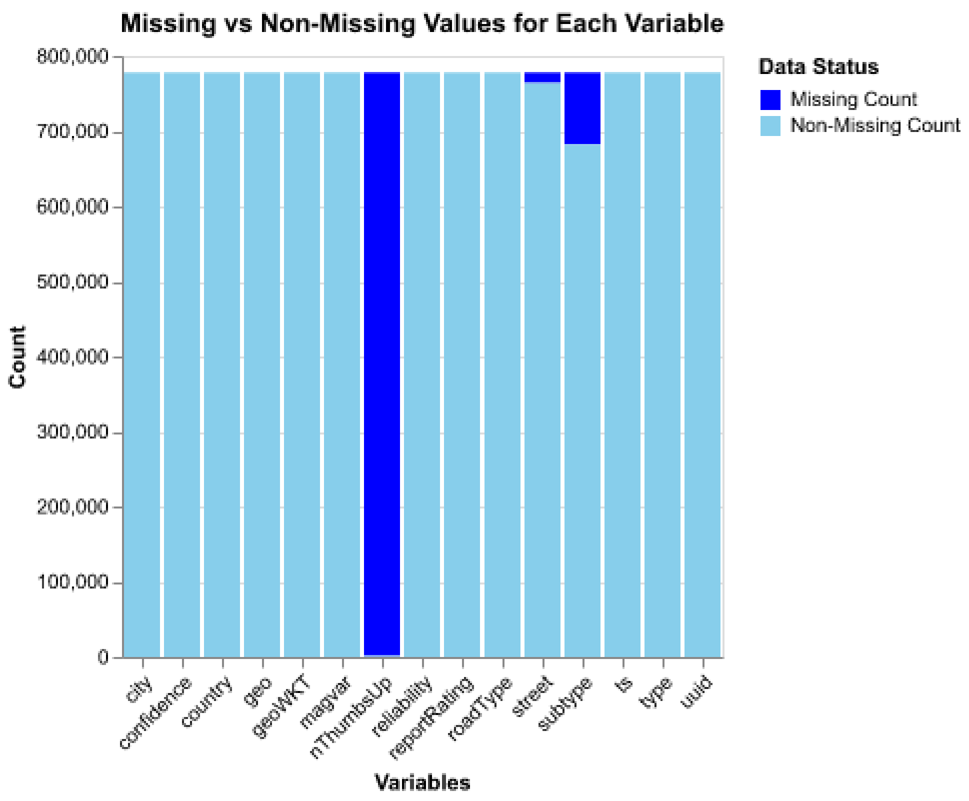
```
Variable names and general dtypes:
city           object
confidence      int64
nThumbsUp     float64
street         object
uuid           object
dtype: object
| Variable     | Altair Data Type   |
|:-------------|:-------------------|
| city         | Nominal            |
| confidence   | Quantitative       |
| nThumbsUp    | Quantitative       |
| street       | Nominal            |
| uuid         | Nominal            |
| country      | Nominal            |
| type         | Nominal            |
| subtype      | Nominal            |
| roadType     | Quantitative       |
| reliability  | Quantitative       |
| magvar       | Quantitative       |
| reportRating | Quantitative       |
```

2.

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

# count missing and non-missing values for each variable
missing_count = waze_data.isnull().sum()
non_missing_count = waze_data.notnull().sum()

# create a DataFrame to display the counts
```

```
count_df = pd.DataFrame({
    'Variable': missing_count.index,
    'Missing Count': missing_count,
    'Non-Missing Count': non_missing_count
}).reset_index(drop=True)
print(f'Counts for missing and non-missing: \n{count_df.head()}')

# melt the dataframe into a long format for Altair
count_df_long = count_df.melt(id_vars=['Variable'], value_vars=[
    'Missing Count', 'Non-Missing Count'], var_name='Data Status',
     ↪  value_name='Count')

# create the Altair stacked bar chart
stacked_bar_chart = alt.Chart(count_df_long).mark_bar().encode(
    x=alt.X('Variable:N', axis=alt.Axis(title='Variables', labelAngle=315)),
    y=alt.Y('Count:Q', axis=alt.Axis(title='Count')),
    color=alt.Color('Data Status:N', scale=alt.Scale(domain=['Missing Count',
 ↪  'Non-Missing Count'], range=['blue', 'skyblue']),
 ↪  legend=alt.Legend(title='Data Status'))
).properties(
    title='Missing vs Non-Missing Values for Each Variable'
)

stacked_bar_chart.display()

# calculate share of observations that are missing & sort
share_of_null = count_df[count_df['Missing Count'] > 0].copy()
share_of_null['share'] = share_of_null['Missing Count'] /
 ↪  (share_of_null['Missing Count'] + share_of_null['Non-Missing Count'])
share_of_null_sorted = share_of_null.sort_values(by='share', ascending=False)
print(f'Share of missing: \n{share_of_null_sorted}')
# Google for "count missing and non-missing in python"
# ChatGPT reference for "revising the dataset format for Altair stacked bar
 ↪  chart plotting"
```

```
Counts for missing and non-missing:
     Variable  Missing Count  Non-Missing Count
0        city              0             778094
1  confidence              0             778094
2   nThumbsUp         776723               1371
3      street          14073             764021
4        uuid              0             778094
```



```
Share of missing:
    Variable  Missing Count  Non-Missing Count     share
2  nThumbsUp         776723               1371  0.998238
7    subtype          96086             682008  0.123489
3     street          14073             764021  0.018087
```

The variables nThumbsUp, street, and subtype contain NULL values, with nThumbsUp having the highest proportion of missing observations (approximately 99.82%).

3.

a.

```
# print unique values for columns type and subtype
unique_types = waze_data['type'].unique()
unique_subtypes = waze_data['subtype'].unique()
print(f'Unique values in type: {unique_types}')
print(f'Unique values in subtype: {unique_subtypes}')

# find number of types having a subtype that is NA
```

```python
na_subtype_count = waze_data[waze_data['subtype'].isna()]['type'].nunique()
print(f'Number of unique type values where subtype is NA:
 ↪  {na_subtype_count}')

# groupby and rename
type_subtype = waze_data.groupby('type')['subtype'].unique().reset_index()
type_subtype = type_subtype.rename(columns={'subtype': 'Unique Subtypes'})

# count the number of unique subtypes for each type
type_subtype['Num of Unique Subtypes'] = type_subtype['Unique
 ↪  Subtypes'].apply(len)
print(f'Type-Subtype combinations with unique subtypes:\n{type_subtype}')
# ChatGPT reference for "rename columns" and "count number of unique subtype
 ↪  values for each btype"
```

```
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 where subtype is NA: 4
Type-Subtype combinations with unique subtypes:
          type                                  Unique Subtypes  \
0     ACCIDENT               [nan, ACCIDENT_MAJOR, ACCIDENT_MINOR]
1       HAZARD  [nan, HAZARD_ON_ROAD, HAZARD_ON_ROAD_CAR_STOPP...
2          JAM  [nan, JAM_HEAVY_TRAFFIC, JAM_MODERATE_TRAFFIC,...
3  ROAD_CLOSED  [nan, ROAD_CLOSED_EVENT, ROAD_CLOSED_CONSTRUCT...

   Num of Unique Subtypes
0                       3
1                      20
2                       5
3                       4
```

Based on the names and diversity of subtypes, HAZARD (with 20 unique subtypes) seems to have enough information that could potentially have sub-subtypes.

b.

```
Hierarchical Menu Structure
1. Accident
  - Major
  - Minor
  - Unclassified
2. Hazard
  i. On Road
    - Car Stopped
    - Road Construction
    - Emergency Vehicle
    - Ice
    - Object
    - Pot Hole
    - Traffic Light Fault
    - Lane Closed
    - Road Kill
  ii. On Shoulder
    - Car Stopped
    - Animals
    - Missing Sign
  iii. Weather
    - Flood
    - Fog
    - Heavy Snow
    - Hail
  iv. Unclassified
3. Jam
  - Heavy Traffic
  - Moderate Traffic
  - Stand Still Traffic
  - Light Traffic
  - Unclassified
4. Road Closed
  - Event
  - Construction
  - Hazard
  - Unclassified
```

c. Yes, we should keep the NA subtypes because removing them could lead to data loss or distort the dataset. Keeping NA values clearly indicates missing or unclassified data, which can help identify gaps or areas needing further attention. Additionally, if more data becomes available later, these NA entries can be updated or categorized. Retaining them ensures that these records are not overlooked and remain available for future analysis or improvements.

```python
# replace NA values in the 'subtype' column with 'Unclassified'
waze_data['subtype'] = waze_data['subtype'].fillna('Unclassified')
print(waze_data[['type', 'subtype']].head())
# ChatGPT reference for replace na with "Unclassified"
```

```
          type       subtype
0          JAM  Unclassified
1     ACCIDENT  Unclassified
2  ROAD_CLOSED  Unclassified
3          JAM  Unclassified
4          JAM  Unclassified
```

4.

a.

```python
# define a DataFrame that has five columns
crosswalk_df = pd.DataFrame(columns=['type', 'subtype', 'updated_type',
↪ 'updated_subtype', 'updated_subsubtype'])
print(crosswalk_df)
```

```
Empty DataFrame
Columns: [type, subtype, updated_type, updated_subtype, updated_subsubtype]
Index: []
```

b.

```python
# define the crosswalk based on the hierarchical menu structure
crosswalk_dict = {
    'ACCIDENT': {
        'ACCIDENT_MAJOR': ['Accident', 'Major', 'Unclassified'],
        'ACCIDENT_MINOR': ['Accident', 'Minor', 'Unclassified'],
        'Unclassified': ['Accident', 'Unclassified', 'Unclassified']
    },
    'HAZARD': {
        # On Road
        'HAZARD_ON_ROAD': ['Hazard', 'On Road', 'Unclassified'],
        'HAZARD_ON_ROAD_CAR_STOPPED': ['Hazard', 'On Road', 'Car Stopped'],
        'HAZARD_ON_ROAD_CONSTRUCTION': ['Hazard', 'On Road', 'Road
         ↪  Construction'],
        'HAZARD_ON_ROAD_EMERGENCY_VEHICLE': ['Hazard', 'On Road', 'Emergency
         ↪  Vehicle'],
        'HAZARD_ON_ROAD_ICE': ['Hazard', 'On Road', 'Ice'],
        'HAZARD_ON_ROAD_OBJECT': ['Hazard', 'On Road', 'Object'],
        'HAZARD_ON_ROAD_POT_HOLE': ['Hazard', 'On Road', 'Pot Hole'],
        'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT': ['Hazard', 'On Road', 'Traffic
         ↪  Light Fault'],
        'HAZARD_ON_ROAD_LANE_CLOSED': ['Hazard', 'On Road', 'Lane Closed'],
        'HAZARD_ON_ROAD_ROAD_KILL': ['Hazard', 'On Road', 'Road Kill'],
        # On Shoulder
        'HAZARD_ON_SHOULDER': ['Hazard', 'On Shoulder', 'Unclassified'],
        'HAZARD_ON_SHOULDER_CAR_STOPPED': ['Hazard', 'On Shoulder', 'Car
         ↪  Stopped'],
        'HAZARD_ON_SHOULDER_ANIMALS': ['Hazard', 'On Shoulder', 'Animals'],
        'HAZARD_ON_SHOULDER_MISSING_SIGN': ['Hazard', 'On Shoulder', 'Missing
         ↪  Sign'],
        # Weather
        'HAZARD_WEATHER': ['Hazard', 'Weather', 'Unclassified'],
        'HAZARD_WEATHER_FLOOD': ['Hazard', 'Weather', 'Flood'],
        'HAZARD_WEATHER_FOG': ['Hazard', 'Weather', 'Fog'],
        'HAZARD_WEATHER_HEAVY_SNOW': ['Hazard', 'Weather', 'Heavy Snow'],
        'HAZARD_WEATHER_HAIL': ['Hazard', 'Weather', 'Hail'],
        # Unclassified
        'Unclassified': ['Hazard', 'Unclassified', 'Unclassified']
    },
    'JAM': {
        'JAM_HEAVY_TRAFFIC': ['Jam', 'Heavy Traffic', 'Unclassified'],
        'JAM_MODERATE_TRAFFIC': ['Jam', 'Moderate Traffic', 'Unclassified'],
        'JAM_STAND_STILL_TRAFFIC': ['Jam', 'Stand Still Traffic',
         ↪  'Unclassified'],
        'JAM_LIGHT_TRAFFIC': ['Jam', 'Light Traffic', 'Unclassified'],
        'Unclassified': ['Jam', 'Unclassified', 'Unclassified']
    },
    'ROAD_CLOSED': {
        'ROAD_CLOSED_EVENT': ['Road Closed', 'Event', 'Unclassified'],
        'ROAD_CLOSED_CONSTRUCTION': ['Road Closed', 'Construction',
         ↪  'Unclassified'],
        'ROAD_CLOSED_HAZARD': ['Road Closed', 'Hazard', 'Unclassified'],
        'Unclassified': ['Road Closed', 'Unclassified', 'Unclassified']
    }
}
# flatten the dictionary into a list of rows
rows = []
for category, subtypes in crosswalk_dict.items():
    for key, value in subtypes.items():
        rows.append([category, key] + value)

# reassign the DataFrame
crosswalk_df = pd.DataFrame(rows, columns=['type', 'subtype', 'updated_type',
↪  'updated_subtype', 'updated_subsubtype'])

print(crosswalk_df.head())
print(f'Number of rows in crosswalk DataFrame: {crosswalk_df.shape[0]}')
# ChatGPT reference for "make a crosswalk based on given structure"
```

```
       type                     subtype updated_type updated_subtype  \
0  ACCIDENT              ACCIDENT_MAJOR     Accident           Major
1  ACCIDENT              ACCIDENT_MINOR     Accident           Minor
2  ACCIDENT                Unclassified     Accident    Unclassified
3    HAZARD              HAZARD_ON_ROAD       Hazard         On Road
4    HAZARD  HAZARD_ON_ROAD_CAR_STOPPED       Hazard         On Road

  updated_subsubtype
0       Unclassified
1       Unclassified
2       Unclassified
3       Unclassified
4        Car Stopped
Number of rows in crosswalk DataFrame: 32
```

c.

```
# apply the crosswalk to full waze dataset
merged_waze_data = waze_data.merge(crosswalk_df,
on=['type', 'subtype'], how='left')

# filter rows for Accident - Unclassified
accident_unclassified_rows = merged_waze_data[
    (merged_waze_data['updated_type'] == 'Accident') &
    (merged_waze_data['updated_subtype'] == 'Unclassified')]
```

```python
print(f'Number of rows for Accident-Unclassified:
 ↪  {accident_unclassified_rows.shape[0]}')
```

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

d.

```python
# drop duplicates based on type and subtype in both DataFrames
crosswalk_cleaned = crosswalk_df.drop_duplicates(subset=['type', 'subtype'])
merged_waze_cleaned = merged_waze_data.drop_duplicates(subset=['type',
 ↪  'subtype'])

# check for mismatches using 'outer' to include all rows
merged_check = pd.merge(crosswalk_cleaned[['type', 'subtype']],
                        merged_waze_cleaned[['type', 'subtype']],
                        on=['type', 'subtype'],
                        how='outer',
                        indicator=True)

# display the rows with mismatches & print result
mismatches = merged_check[merged_check['_merge'] != 'both']
if mismatches.empty:
    print("No mismatches detected. Values in 'type' and 'subtype' are
     ↪  consistent.")
else:
    print('There are mismatches. Please review for further analysis.')
```

```
No mismatches detected. Values in 'type' and 'subtype' are consistent.
```

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

1.

a.

```python
def extract_coordinates(wkt):
    """function to extract latitude and longitude from WKT"""
    match = re.match(r"Point\((-?\d+\.\d+) (-?\d+\.\d+)\)", wkt)
    if match:
        return float(match.group(1)), float(match.group(2))
    return None, None

# apply the function to the geoWKT column
merged_waze_data['latitude'], merged_waze_data['longitude'] =
 ↪  zip(*merged_waze_data['geoWKT'].apply(extract_coordinates))
print(merged_waze_data[['geoWKT', 'latitude', 'longitude']].head())
# ChatGPT reference for "extracting the latitude and longitude from the
 ↪  Well-Known Text string"
```

```
                        geoWKT   latitude  longitude
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.

```
# bin the latitude and longitude into bins of step size 0.01
merged_waze_data['latitude_binned'] = merged_waze_data['latitude'].round(2)
merged_waze_data['longitude_binned'] = merged_waze_data['longitude'].round(2)

# display the binned coordinates
print(merged_waze_data[['latitude', 'longitude', 'latitude_binned',
 ↪  'longitude_binned']].head())

# group by binned longitude and latitude, count obs, and sort
binned_counts = (
    merged_waze_data.groupby(['latitude_binned', 'longitude_binned'])
    .size()
    .reset_index(name='count')
    .reindex(columns=['latitude_binned', 'longitude_binned', 'count'])
)

binned_max = binned_counts.sort_values(by='count', ascending=False).iloc[0]
print(f'Binned latitude-longitude with greatest number of observations:
 ↪  \n{binned_max}')
# ChatGPT reference for "binning variables into bins of step size 0.01"
# ChatGPT reference for "ensuring column name order when groupby"
```

```
    latitude  longitude  latitude_binned  longitude_binned
0 -87.676685  41.929692           -87.68             41.93
1 -87.624816  41.753358           -87.62             41.75
2 -87.614122  41.889821           -87.61             41.89
3 -87.680139  41.939093           -87.68             41.94
4 -87.735235  41.916580           -87.74             41.92
Binned latitude-longitude with greatest number of observations:
latitude_binned       -87.65
longitude_binned       41.88
count               21325.00
Name: 492, dtype: float64
```

The binned latitude-longitude combination (-87.65, 41.88) has the greatest number of observations in the overall dataset.

c.

```
# collapse the data: group by and count number of alerts
aggregated_data = (
    merged_waze_data.groupby(['updated_type',
 ↪  'updated_subtype','latitude_binned', 'longitude_binned'])
    .size().reset_index(name='alert_count')
).sort_values('alert_count', ascending=False)
print(f'Number of rows in the DataFrame: {len(aggregated_data)}')

# save the DataFrame to a CSV file in the top_alerts_map folder
output_path = 'top_alerts_map/top_alerts_map.csv'
aggregated_data.to_csv(output_path, index=False)
# GhatGPT reference for "save DataFrae to a csv file"
```

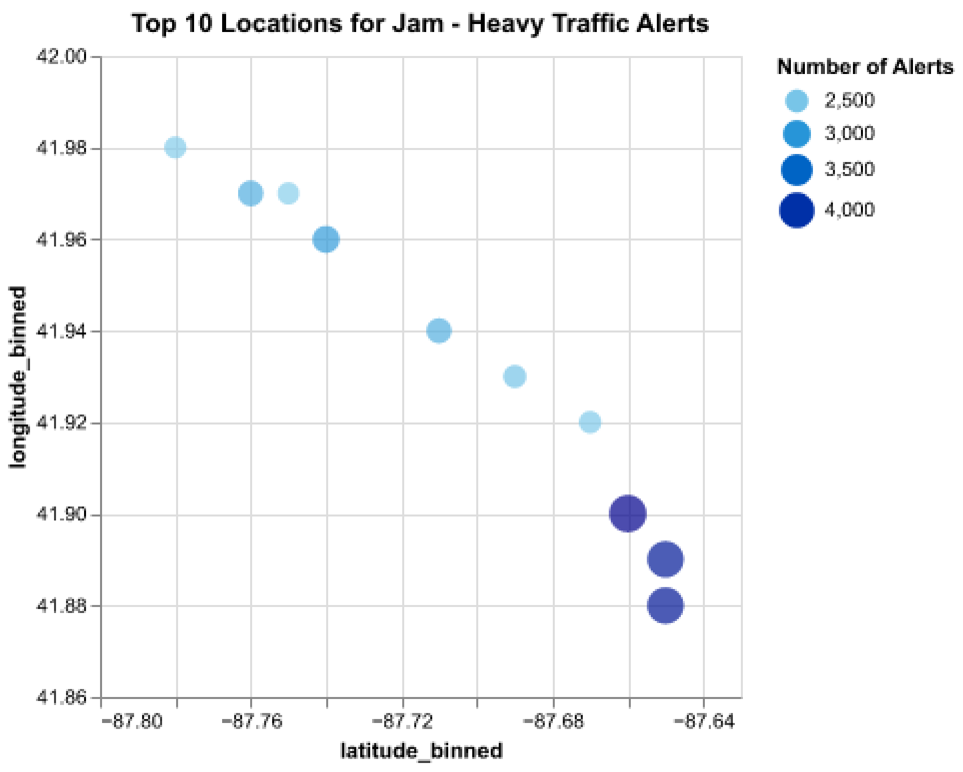
```
Number of rows in the DataFrame: 6675
```

The level of aggregation in this case is at the unique combination of 'updated_type', 'updated_subtype','latitude_binned', and 'longitude_binned', where alert_count represents the total number of alerts for each unique combination of latitude-longitude bin, type, and subtype. Overall, this DataFrame contains 6,675 rows in total.

2.

```
# filter data for Jam - Heavy Traffic
jam_heavy_traffic = aggregated_data[(aggregated_data['updated_type'] ==
 ↪  'Jam') & (
    aggregated_data['updated_subtype'] == 'Heavy Traffic')]
```

```
top_10_jam_alerts = jam_heavy_traffic.sort_values(
    'alert_count', ascending=False).head(10)

# set the domain for latitude and longitude based on the data
lat_domain = [top_10_jam_alerts['latitude_binned'].min(
) - 0.02, top_10_jam_alerts['latitude_binned'].max() + 0.02]
long_domain = [top_10_jam_alerts['longitude_binned'].min(
) - 0.02, top_10_jam_alerts['longitude_binned'].max() + 0.02]

top_10_jam_scatter_plot = alt.Chart(top_10_jam_alerts).mark_circle().encode(
    x=alt.X('latitude_binned:Q', scale=alt.Scale(domain=lat_domain)),
    y=alt.Y('longitude_binned:Q', scale=alt.Scale(domain=long_domain)),
    size=alt.Size('alert_count:Q', title='Number of Alerts',

 ↪  scale=alt.Scale(domain=[top_10_jam_alerts['alert_count'].min(),
 ↪  top_10_jam_alerts['alert_count'].max()], range=[120, 350])),
    color=alt.Color('alert_count:Q', scale=alt.Scale(
        domain=[top_10_jam_alerts['alert_count'].min(),
                top_10_jam_alerts['alert_count'].max()],
        range=['skyblue', 'darkblue'])),
    tooltip=['latitude_binned', 'longitude_binned', 'alert_count']
).properties(
    title='Top 10 Locations for Jam - Heavy Traffic Alerts',
    width=320, height=320
)

top_10_jam_scatter_plot.display()
# ChatGPT reference for "set domain of the x and y axis for the latitude and
 ↪  longitude for Altair plotting"
# Geeksforgeeks reference for changing scale ranges in Altair:
 ↪  https://www.geeksforgeeks.org/adjusting-scale-ranges-in-altair/
# Website reference for changing color range:
 ↪  https://github.com/vega/altair/issues/921
```



3.

a.

```
# download the GeoJSON data
geojson_url =
↪  'https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON'
response = requests.get(geojson_url)

# check if the request was successful (status code 200)
if response.status_code == 200:
    # write the content directly to a file
    with open('./top_alerts_map/chicago-boundaries.geojson', 'wb') as f:
        f.write(response.content)
    print('GeoJSON file downloaded and saved successfully.')
else:
    print(f'Failed to download. Status code: {response.status_code}')
# ChatGDP reference for "download file with Python using requests package"
```

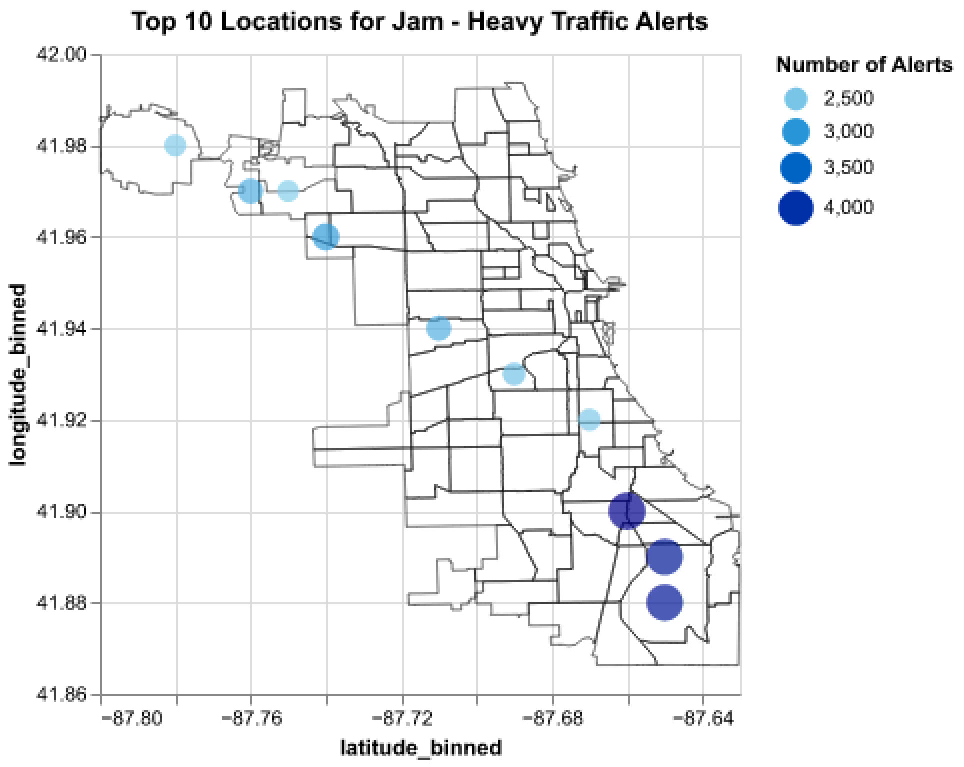
```
GeoJSON file downloaded and saved successfully.
```

b.

```
file_path = './top_alerts_map/chicago-boundaries.geojson'
with open(file_path) as f:
    chicago_geojson = json.load(f)
geo_data = alt.Data(values=chicago_geojson['features'])
```

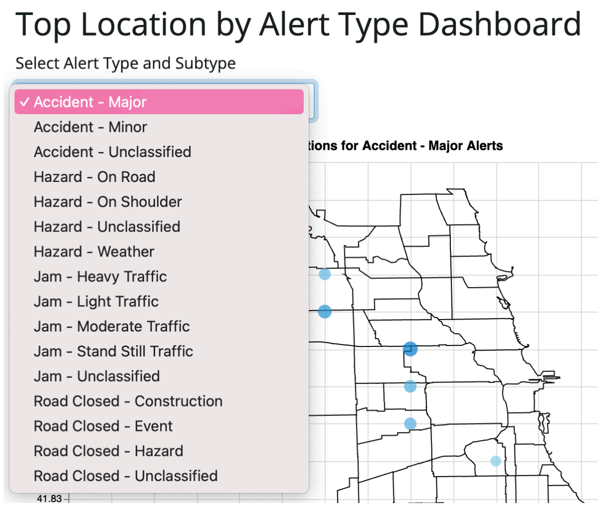
4.

```
# create the map (background) using the GeoJSON data
background = alt.Chart(geo_data).mark_geoshape(
    fillOpacity=0, stroke='black', strokeWidth=0.6
).project(type = 'equirectangular').properties(
    width=320, height=320
) # alternatively: project(type = 'identity', reflectY = True)

combined_plot = background + top_10_jam_scatter_plot
combined_plot.display()
# ChatGPT reference for "plot geo_data in Altair"
```



5.

a.



There are 16 total type x subtype combinations in my dropdown menu.

b.

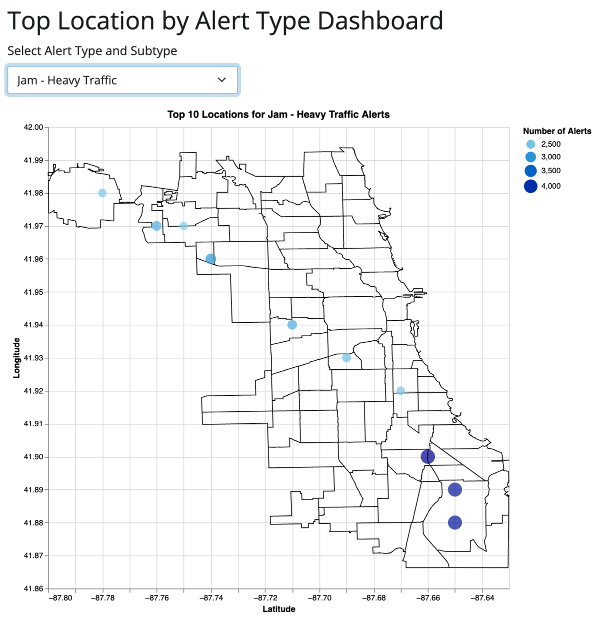
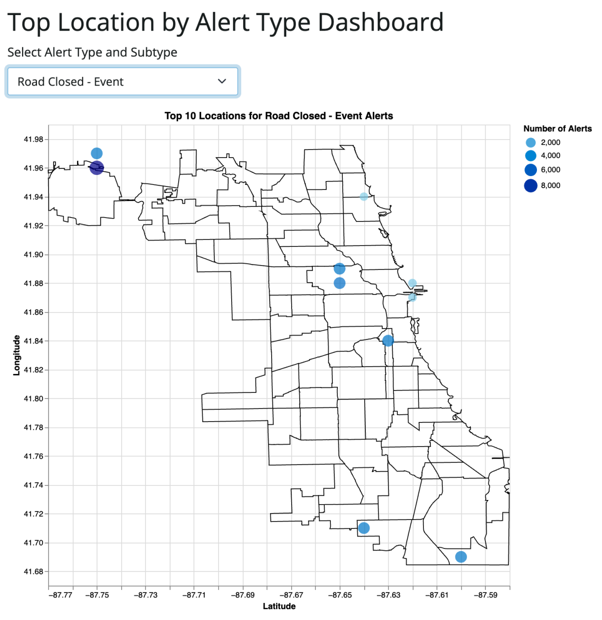


Figure 1: App1.b. Map

c.

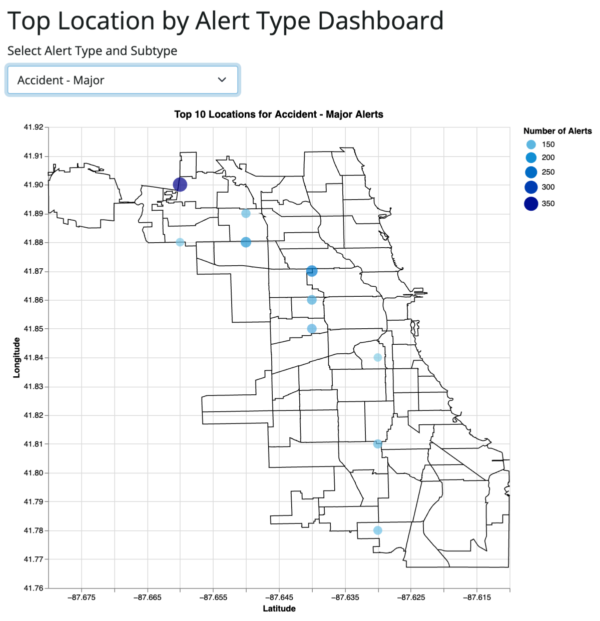

Top Location by Alert Type Dashboard

Select Alert Type and Subtype

Road Closed - Event

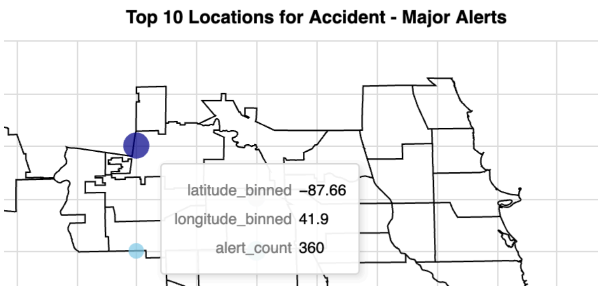
Figure 2: App1.c. Map

d.


Top Location by Alert Type Dashboard

Select Alert Type and Subtype

Accident - Major



latitude_binned −87.66
longitude_binned 41.9
alert_count 360

Question: which location has the most alerts for major accident, and what is the total count at that location?

Answer: The location (-87.66, 41.9) has the most alerts for major accidents, with a total count of 360.

e. To enhance our analysis, we could add additional columns such as ts (the timestamp of the reported alert), roadType (which serves as a reference guide for different types of alerts, jams, and irregularities), reliability (indicating the confidence in the alert based on user input), and confidence (reflecting the level of confidence in the alert based on user reactions), etc. These enhancements could provide a more comprehensive view in the dashboard.

```
# app.py for APP 1
print_file_contents("./top_alerts_map/basic-app/app.py")
```

~~~
```python
import altair as alt
import pandas as pd
import json
import requests
from shiny import App, ui, render
from shinywidgets import output_widget, render_altair

top_alerts_map = pd.read_csv(

    '/Users/boyalin/Documents/GitHub/ppha30538_ps/problem-set-6-boyalin/top_alerts_map/top_al

url =
'https://data.cityofchicago.org/api/geospatial/igwz-8jzy?method=export&format=GeoJSON'
response = requests.get(url)
file_path = 'chicago_neighborhoods.geojson'

with open(file_path, 'wb') as file:
    file.write(response.content)
with open(file_path) as f:
    chicago_geojson = json.load(f)

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

# define function to get the top alerts based on selected type and subtype
def get_top_alerts(selected_type, selected_subtype):
    # filter based on type and subtype
    filtered_df = top_alerts_map[(top_alerts_map['updated_type'] ==
    selected_type) & (
        top_alerts_map['updated_subtype'] == selected_subtype)]
    # sort by the highest alert count and return the top 10
~~~

```
    top_alerts = filtered_df.sort_values(
        'alert_count', ascending=False).head(10)
    return top_alerts

# dropdown choices with unique combinations of alert type and subtype
dropdown_choices = (
    top_alerts_map[['updated_type', 'updated_subtype']]
    .drop_duplicates()
    .sort_values(by=['updated_type', 'updated_subtype'])
    .apply(lambda row: f"{row['updated_type']} - {row['updated_subtype']}",
    axis=1)
    .tolist()
)

# UI side
app_ui = ui.page_fluid(
    ui.panel_title('Top Location by Alert Type Dashboard'),
    ui.input_select(
        'alert_type_subtype',
        'Select Alert Type and Subtype',
        choices=dropdown_choices,
        selected=dropdown_choices[0]
    ),
    output_widget('top_alerts_plot')
)

# server logic
def server(input, output, session):
    @output
    @render_altair
    def top_alerts_plot():
        # parse user selection
        selected_type, selected_subtype = input.alert_type_subtype().split("
        - ")
        top_alerts = get_top_alerts(selected_type, selected_subtype)

        # set the domain for latitude and longitude based on the data
        lat_domain = [top_alerts['latitude_binned'].min(
        ) - 0.02, top_alerts['latitude_binned'].max() + 0.02]
        long_domain = [top_alerts['longitude_binned'].min(
        ) - 0.02, top_alerts['longitude_binned'].max() + 0.02]

        # scatter plot for top alerts
        scatter_plot = alt.Chart(top_alerts).mark_circle().encode(
            x=alt.X('latitude_binned:Q', title='Latitude',
                    scale=alt.Scale(domain=lat_domain)),
            y=alt.Y('longitude_binned:Q', title='Longitude',
                    scale=alt.Scale(domain=long_domain)),
            size=alt.Size('alert_count:Q', title='Number of Alerts'),
            color=alt.Color('alert_count:Q', scale=alt.Scale(
                range=['skyblue', 'darkblue'])),
            tooltip=['latitude_binned', 'longitude_binned', 'alert_count']
        ).properties(
            title=f'Top 10 Locations for {selected_type} - {selected_subtype}
            Alerts', width=600, height=600
        )

        map_layer = alt.Chart(geo_data).mark_geoshape(
            fillOpacity=0, stroke='black'
        ).project(type = 'equirectangular').properties(
            width=600, height=600
        )

        combined_plot = map_layer + scatter_plot
        return combined_plot

app = App(app_ui, server)

# ChatGPT reference for "make a dropdown choices with unique combinations of
alert type and subtype" and "parse the combination when plotting using
Altair"
```

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

1.

a. I think collapsing the dataset by the ts (timestamp) column would not be effective, because it has high granularity with second-level precision, which could result in each row being unique and make aggregation impractical. Instead, aggregating by broader time frames, such as hours, would be more suitable for meaningful analysis.

b.

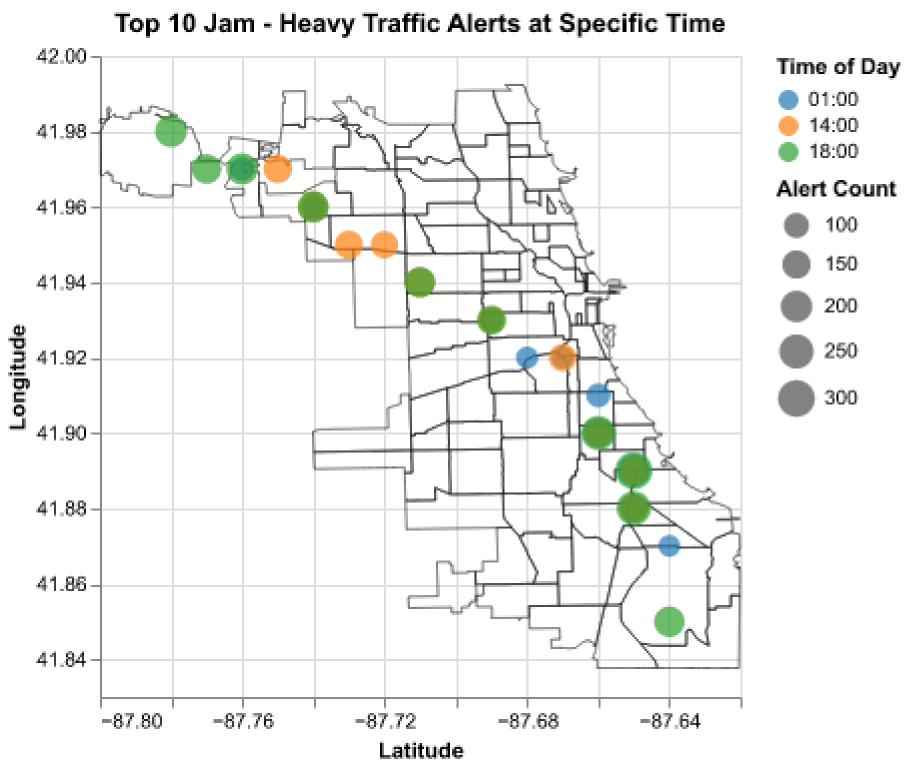
```python
# convert ts column to datetime format & extract hour from timestamp
merged_waze_data['ts'] = pd.to_datetime(merged_waze_data['ts'])
merged_waze_data['hour'] = merged_waze_data['ts'].dt.strftime('%H:00')

# group by hour, type, subtype, binned location, and hour
collapsed_data = (merged_waze_data.groupby(
    ['hour', 'updated_type', 'updated_subtype', 'latitude_binned',
 ↪  'longitude_binned']).size().reset_index(name='alert_count'))

# save the collapsed dataset
output_path = 'top_alerts_map_byhour/top_alerts_map_byhour.csv'
collapsed_data.to_csv(output_path, index=False)

print(f'Number of rows in collapsed dataset: {len(collapsed_data)}')
# ChatGPT reference for "extracting hour from timestemp like 2024-07-02
 ↪  18:27:40 UTC"
```

```
Number of rows in collapsed dataset: 62825
```

c.

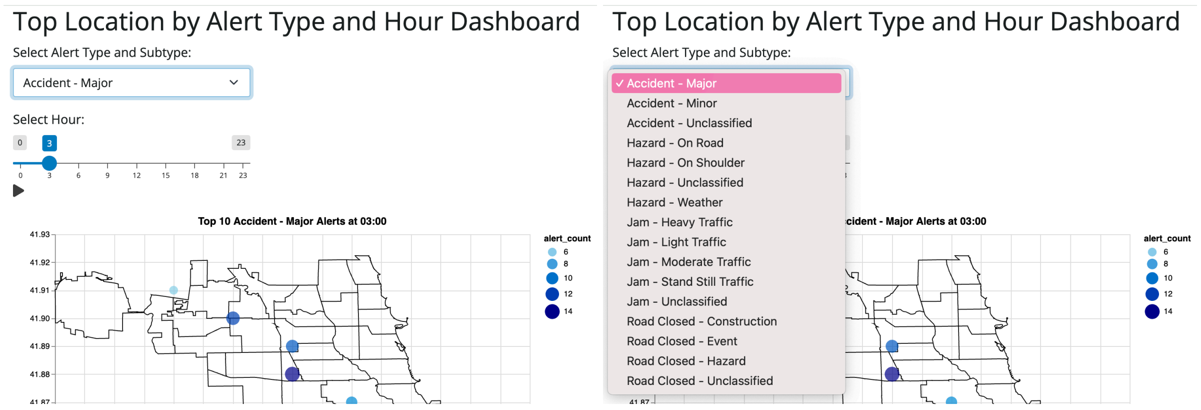
```python
# filter collapsed_data for type, subtype, and specific hour
specific_hours = ['01:00', '14:00', '18:00']
filtered_data_byhour = collapsed_data[
    (collapsed_data['updated_type'] == 'Jam') &
    (collapsed_data['updated_subtype'] == 'Heavy Traffic') &
    (collapsed_data['hour'].isin(specific_hours))
].copy()

# find top 10 locations for each hour
top_10_by_hour = (
    filtered_data_byhour.groupby('hour', group_keys=False)
    .apply(lambda x: x.nlargest(10, 'alert_count'))
)

lat_domain_byhour = [top_10_by_hour['latitude_binned'].min(
) - 0.02, top_10_by_hour['latitude_binned'].max() + 0.02]
long_domain_byhour = [top_10_by_hour['longitude_binned'].min(
) - 0.02, top_10_by_hour['longitude_binned'].max() + 0.02]

scatter_plot_combined = alt.Chart(top_10_by_hour).mark_circle().encode(
    x=alt.X('latitude_binned:Q', scale=alt.Scale(
        domain=lat_domain_byhour), title="Latitude"),
    y=alt.Y('longitude_binned:Q', scale=alt.Scale(
        domain=long_domain_byhour), title="Longitude"),
    size=alt.Size('alert_count:Q', title='Alert Count',

 ↪  scale=alt.Scale(domain=[top_10_by_hour['alert_count'].min(),
 ↪  top_10_by_hour['alert_count'].max()], range=[120, 350])),
    color=alt.Color('hour:N', scale=alt.Scale(
        domain=['01:00', '14:00', '18:00'],
        range=['#1f77b4', '#ff7f0e', '#2ca02c']), title="Time of Day"),
    tooltip=['hour', 'latitude_binned', 'longitude_binned', 'alert_count']
).properties(
    title="Top 10 Jam - Heavy Traffic Alerts at Specific Time",
    width=320, height=320
)

combined_plot_byhour = background + scatter_plot_combined
combined_plot_byhour.display()
# ChatGDP reference for "revise to generate an individual plot for 3
 ↪  different times" and "find top 10 for each hour"
# Color code reference: https://htmlcolorcodes.com/
```

```
/var/folders/r5/j3b0xjqs7h9bw5yxznft8cl00000gn/T/ipykernel_20327/3109504024.py:11:
DeprecationWarning:

DataFrameGroupBy.apply operated on the grouping columns. This behavior is
deprecated, and in a future version of pandas the grouping columns will be
excluded from the operation. Either pass `include_groups=False` to exclude
the groupings or explicitly select the grouping columns after groupby to
silence this warning.
```



2.

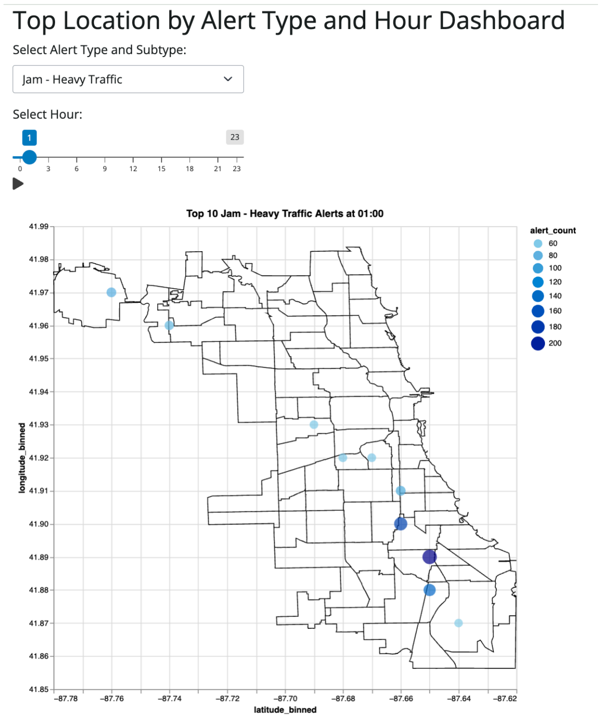
a.


Top Location by Alert Type and Hour Dashboard

Select Alert Type and Subtype: Accident - Major

Select Hour: 3

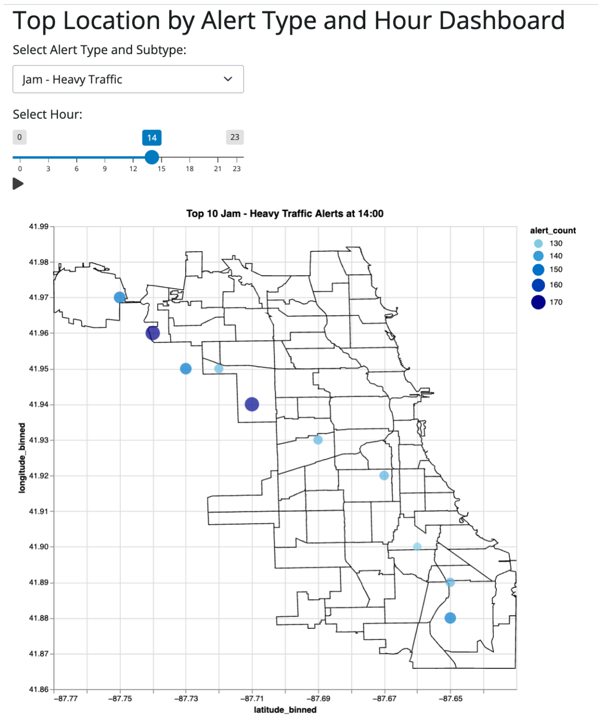
b.



Top Location by Alert Type and Hour Dashboard

Select Alert Type and Subtype: Jam - Heavy Traffic

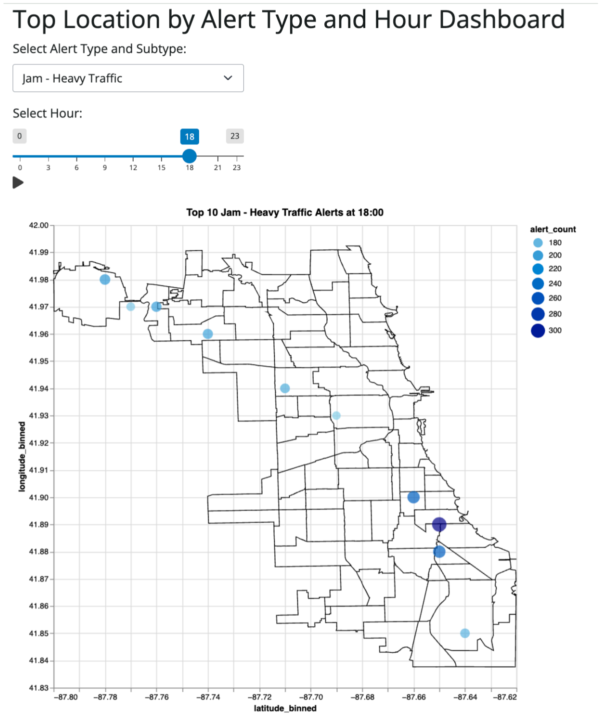
Select Hour: 1



Top Location by Alert Type and Hour Dashboard

Select Alert Type and Subtype: Jam - Heavy Traffic

Select Hour: 14

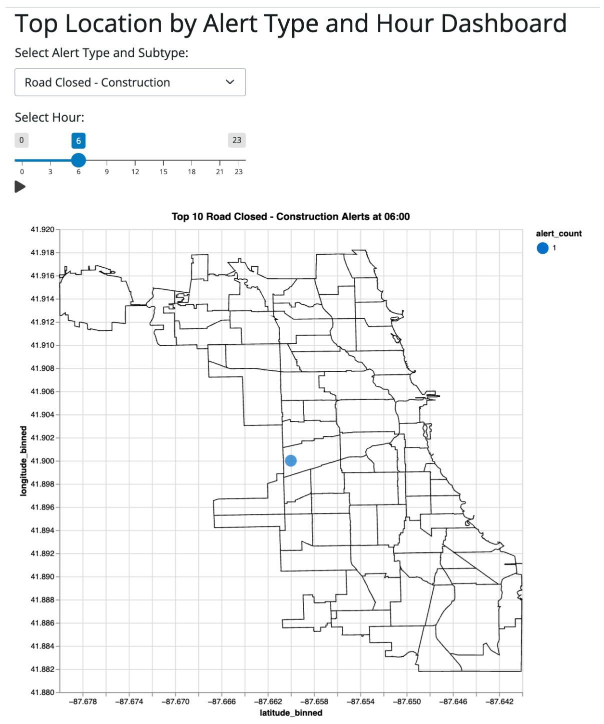


Top Location by Alert Type and Hour Dashboard

Select Alert Type and Subtype: Jam - Heavy Traffic

Select Hour: 18

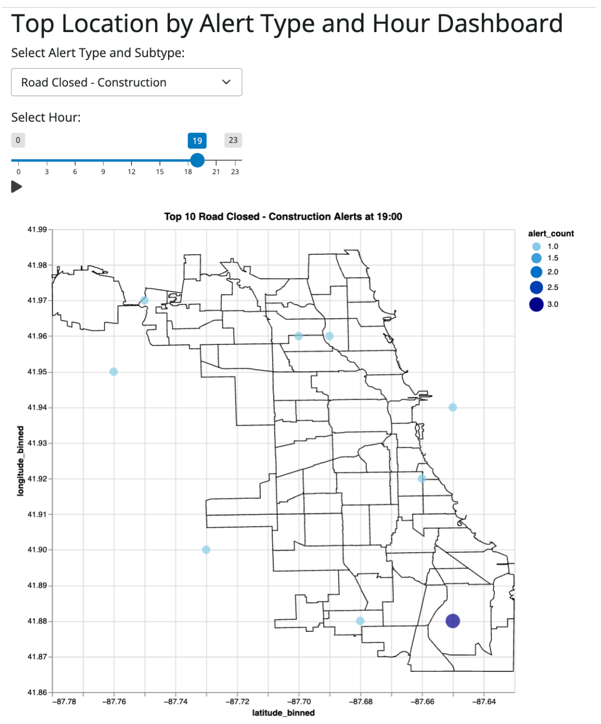
c.


Top Location by Alert Type and Hour Dashboard

Select Alert Type and Subtype:

Road Closed - Construction

Select Hour: 6


Top Location by Alert Type and Hour Dashboard

Select Alert Type and Subtype:

Road Closed - Construction

Select Hour: 19

It seems like road construction is done more during night hours.

```python
# app.py for APP 2
print_file_contents("./top_alerts_map_byhour/basic-app/app.py")
```

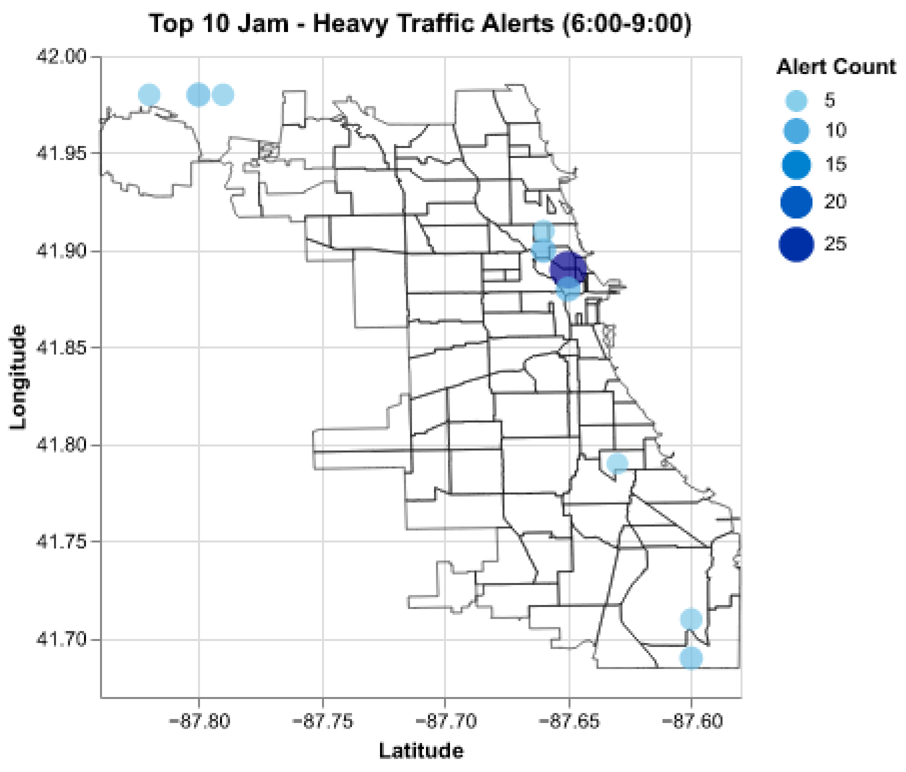
~~~
```python
import altair as alt
import pandas as pd
import json
import requests
from shiny import App, ui, render
from shinywidgets import output_widget, render_altair

collapsed_data = pd.read_csv(

    '/Users/boyalin/Documents/GitHub/ppha30538_ps/problem-set-6-boyalin/top_alerts_map_byhou

url =
'https://data.cityofchicago.org/api/geospatial/igwz-8jzy?method=export&format=GeoJSON'
response = requests.get(url)
file_path = "chicago_neighborhoods.geojson"

with open(file_path, 'wb') as file:
~~~

```
    file.write(response.content)
with open(file_path) as f:
    chicago_geojson = json.load(f)

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

# define function to get the top alerts based on selected type, subtype, and
hour
def filter_data(type, subtype, hour):
    filtered_data = collapsed_data[
        (collapsed_data['updated_type'] == type) &
        (collapsed_data['updated_subtype'] == subtype) &
        (collapsed_data['hour'] == hour)
    ]
    top_10_byhour = filtered_data.sort_values(
        'alert_count', ascending=False).head(10)
    return top_10_byhour

# dropdown choices with unique combinations of alert type and subtype
dropdown_choices = (
    collapsed_data[['updated_type', 'updated_subtype']]
    .drop_duplicates()
    .sort_values(by=['updated_type', 'updated_subtype'])
    .apply(lambda row: f"{row['updated_type']} - {row['updated_subtype']}",
    axis=1)
    .tolist()
)

# UI side
app_ui = ui.page_fluid(
    ui.panel_title('Top Location by Alert Type and Hour Dashboard'),
    ui.input_select(
        'alert_type_subtype',
        'Select Alert Type and Subtype:',
        choices=dropdown_choices,
        selected=dropdown_choices[0]),
    # slider for selecting the hour
    ui.input_slider('hour', 'Select Hour:', min=0, max=23, value=3, step=1,
                    animate=True, ticks=True),
    output_widget('top_alerts_plot')
)

# server logic
```

```
def server(input, output, session):
    @output()
    @render_altair
    def top_alerts_plot():
        # parse user selection
        # get the selected type and subtype
        selected_type, selected_subtype = input.alert_type_subtype().split("
        - ")
        # get the selected hour
        specific_hour = f'{input.hour():02}:00'
        # filter the data based on user input
        filtered_data = filter_data(
            selected_type, selected_subtype, specific_hour)

        lat_domain = [filtered_data['latitude_binned'].min(
        ) - 0.02, filtered_data['latitude_binned'].max() + 0.02]
        long_domain = [filtered_data['longitude_binned'].min(
        ) - 0.02, filtered_data['longitude_binned'].max() + 0.02]

        scatter_plot_by_hour = alt.Chart(filtered_data).mark_circle().encode(
            x=alt.X('latitude_binned:Q', scale=alt.Scale(domain=lat_domain)),
            y=alt.Y('longitude_binned:Q',
            scale=alt.Scale(domain=long_domain)),
            size=alt.Size('alert_count:Q', scale=alt.Scale(
                domain=[filtered_data['alert_count'].min(
                ), filtered_data['alert_count'].max()],
                range=[120, 350])),
            color=alt.Color('alert_count:Q', scale=alt.Scale(
                domain=[filtered_data['alert_count'].min(
                ), filtered_data['alert_count'].max()],
                range=['skyblue', 'darkblue'])),
            tooltip=['latitude_binned', 'longitude_binned', 'alert_count']
        ).properties(
            title=f"Top 10 {selected_type} - {selected_subtype} Alerts at
            {specific_hour}",
            width=600, height=600
        )

        map_layer = alt.Chart(geo_data).mark_geoshape(
            fillOpacity=0, stroke='black'
        ).project(type='equirectangular').properties(
            width=600, height=600
        )
        combined_plot_by_hour = map_layer + scatter_plot_by_hour
        return combined_plot_by_hour

app = App(app_ui, server)

# ChatGPT reference for "make a ui.input_slider for 24 hours"
# ChatGDP reference for "parsing hour input as 01:00 format"

```
```

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

1.

a. I think collapsing the dataset by range of hours is not necessary, because the dataset is already aggregated at the individual hour level. To plot the top 10 locations by alert type and range of hours, we can simply filter and group certain data. The Shiny app can also dynamically filter and group the data based on the selected range without pre-aggregation. Maintaining the dataset at the hourly level can ensure flexibility and simplify the data preparation process.

b.

```
# filter the data for the chosen alert type, subtype, and hour range
filtered_data_byrange = collapsed_data[
    (collapsed_data['updated_type'] == 'Jam') &
    (collapsed_data['updated_subtype'] == 'Heavy Traffic') &
    (collapsed_data['hour'] >= '06:00') &
    (collapsed_data['hour'] <= '09:00')
]
# aggregate and sum alert counts across range
aggregated_by_range = (
    filtered_data_byrange.groupby(['latitude_binned', 'longitude_binned'])
    .agg({'alert_count': 'sum'})
    .reset_index()
)

top_10_by_range = aggregated_by_range.sort_values(
    'alert_count', ascending=False).head(10)
```
```

```
lat_domain_byrange = [top_10_by_range['latitude_binned'].min(
) - 0.02, top_10_by_range['latitude_binned'].max() + 0.02]
long_domain_byrange = [top_10_by_range['longitude_binned'].min(
) - 0.02, top_10_by_range['longitude_binned'].max() + 0.02]

scatter_plot_by_range = alt.Chart(top_10_by_range).mark_circle().encode(
    x=alt.X('latitude_binned:Q', scale=alt.Scale(
        domain=lat_domain_byrange), title='Latitude'),
    y=alt.Y('longitude_binned:Q', scale=alt.Scale(
        domain=long_domain_byrange), title='Longitude'),
    size=alt.Size('alert_count:Q', scale=alt.Scale(
        domain=[top_10_by_range['alert_count'].min(
        ), top_10_by_range['alert_count'].max()],
        range=[120, 350]), title='Alert Count'),
    color=alt.Color('alert_count:Q', scale=alt.Scale(
        domain=[top_10_by_range['alert_count'].min(
        ), top_10_by_range['alert_count'].max()],
        range=['skyblue', 'darkblue'])),
    tooltip=['latitude_binned', 'longitude_binned', 'alert_count']
).properties(
    title=f'Top 10 Jam - Heavy Traffic Alerts (6:00-9:00)',
    width=320, height=320
)

combined_plot_by_range = background + scatter_plot_by_range
combined_plot_by_range.display()
```



2.

a.

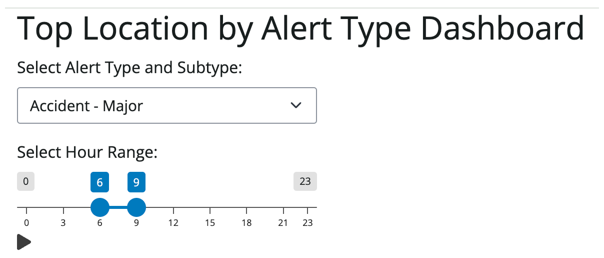
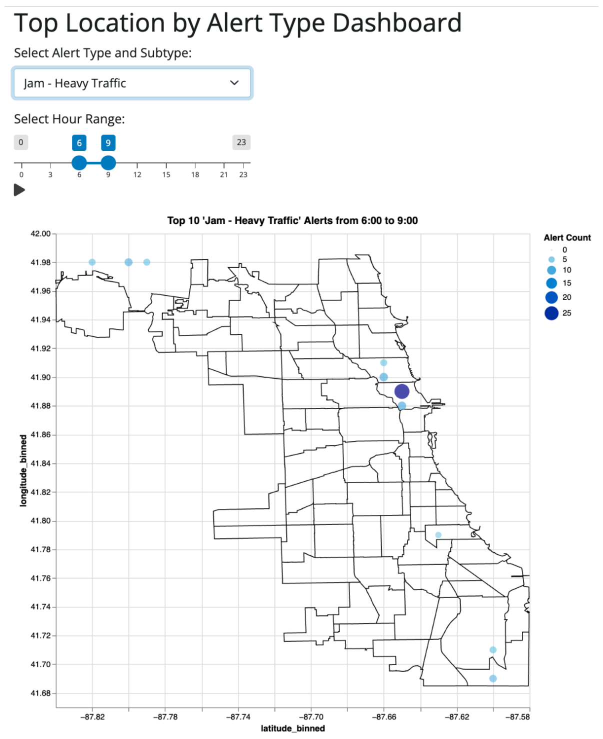

Top Location by Alert Type Dashboard
Select Alert Type and Subtype:
Accident - Major
Select Hour Range:
0
6
9
23
0 3 6 9 12 15 18 21 23

Figure 3: App3. 2.a. Dropdown Menu & Hour Range

b.


Top Location by Alert Type Dashboard

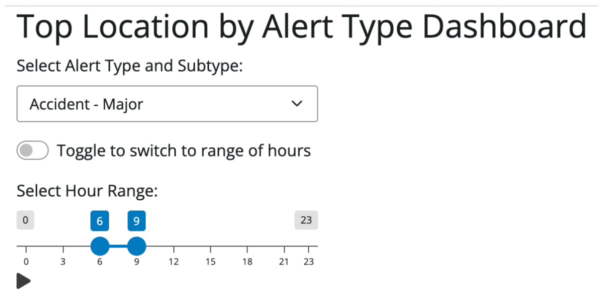
Select Alert Type and Subtype: Jam - Heavy Traffic

Select Hour Range: 6 – 9

Figure 4: App3. 2.b. Map

3.

a.


Top Location by Alert Type Dashboard

Select Alert Type and Subtype: Accident - Major

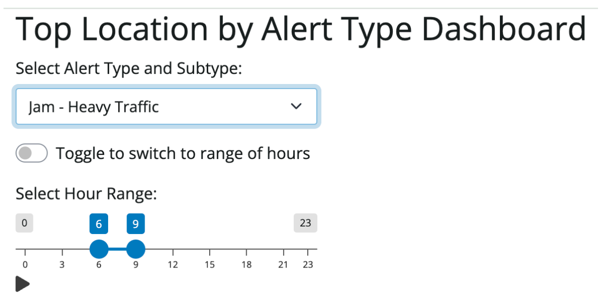
Toggle to switch to range of hours

Select Hour Range: 6 – 9

Figure 5: App3. 3.a. Switch Button

The possible values of input.switch_button are True (or 1) and False (or 0). When toggled on, the value is True; when toggled off, it is False. These binary values can be used to conditionally display the hour slider, enabling selection of either a single hour or a range of hours.

b.

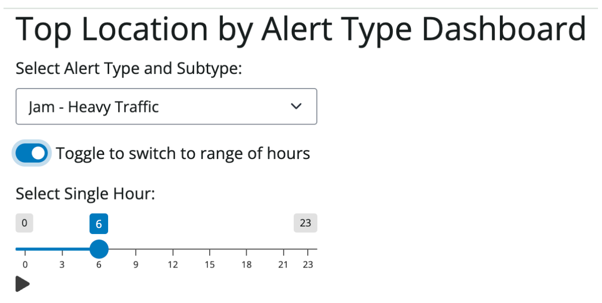


Top Location by Alert Type Dashboard

Select Alert Type and Subtype: Jam - Heavy Traffic

Toggle to switch to range of hours

Select Hour Range: 6 – 9



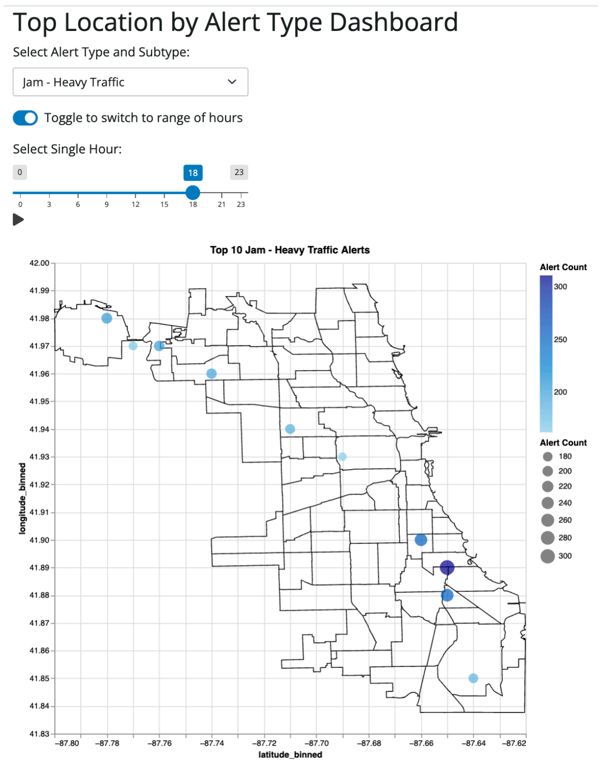
Top Location by Alert Type Dashboard

Select Alert Type and Subtype: Jam - Heavy Traffic

Toggle to switch to range of hours

Select Single Hour: 6

c.

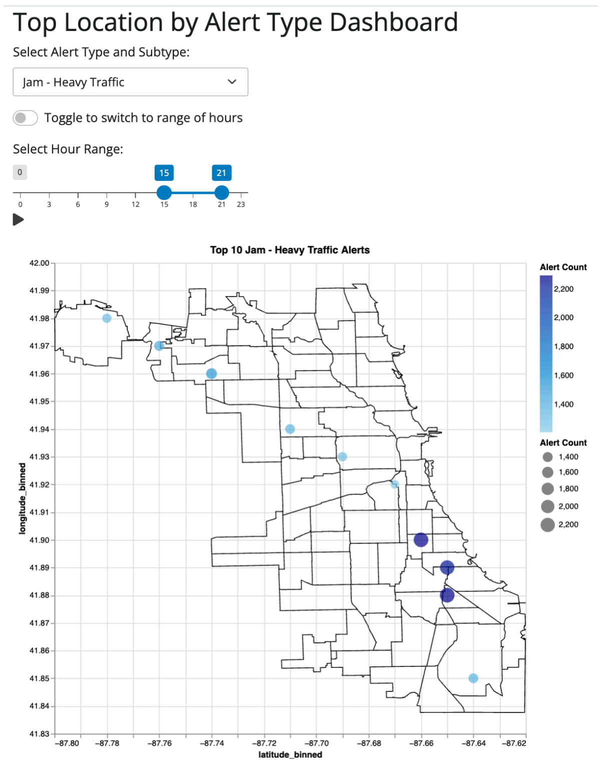


Top Location by Alert Type Dashboard

Select Alert Type and Subtype: Jam - Heavy Traffic

Toggle to switch to range of hours

Select Single Hour: 18



Top Location by Alert Type Dashboard

Select Alert Type and Subtype: Jam - Heavy Traffic

Toggle to switch to range of hours

Select Hour Range: 15 – 21

d. To create a plot similar to the one provided, we might need to re-classify data into certain time periods for morning vs. afternoon by creating a new variable based on the alert time. Moreover, points on the map should be color-coded by time period (e.g., red for "Morning" and blue for "Afternoon") and scaled in size to represent the total number of alerts at each location, with larger points indicating higher alert counts. Legends should be added to explain both the point size and color. Additionally, transparency (alpha) should be adjusted to reduce overlap and improve clarity in areas with dense clusters of points.

```python
# app.py for APP 3
print_file_contents("./top_alerts_map_byhour_sliderrange/basic-app/app.py")
```

~~~
```python
import altair as alt
import pandas as pd
import json
import requests
from shiny import App, ui, render
from shinywidgets import output_widget, render_altair

collapsed_data = pd.read_csv(

    '/Users/boyalin/Documents/GitHub/ppha30538_ps/problem-set-6-boyalin/top_alerts_map_byhou
)

url =
'https://data.cityofchicago.org/api/geospatial/igwz-8jzy?method=export&format=GeoJSON'
response = requests.get(url)
file_path = 'chicago_neighborhoods.geojson'

with open(file_path, 'wb') as file:
    file.write(response.content)
with open(file_path) as f:
    chicago_geojson = json.load(f)

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

# define function to get top alerts for a specific hour input
def filter_data_specific_hour(type, subtype, hour):
    filtered_data = collapsed_data[
        (collapsed_data['updated_type'] == type) &
        (collapsed_data['updated_subtype'] == subtype) &
        (collapsed_data['hour'] == hour)
    ]
    return filtered_data.sort_values('alert_count', ascending=False).head(10)

# define function to get top alerts for a range of hour input
def filter_data_by_range(type, subtype, hour_start, hour_end):
    hour_range = [f'{h:02}:00' for h in range(hour_start, hour_end + 1)]
    filtered_data = collapsed_data[
        (collapsed_data['updated_type'] == type) &
~~~

```
        (collapsed_data['updated_subtype'] == subtype) &
        (collapsed_data['hour'].isin(hour_range))
    ]
    return (
        filtered_data.groupby(['latitude_binned', 'longitude_binned'])
        .agg({'alert_count': 'sum'})
        .reset_index()
        .sort_values('alert_count', ascending=False)
        .head(10)
    )

# dropdown choices with unique combinations of alert type and subtype
dropdown_choices = (
    collapsed_data[['updated_type', 'updated_subtype']]
    .drop_duplicates()
    .sort_values(by=['updated_type', 'updated_subtype'])
    .apply(lambda row: f"{row['updated_type']} - {row['updated_subtype']}",
    axis=1)
    .tolist()
)

# UI side
app_ui = ui.page_fluid(
    ui.panel_title('Top Location by Alert Type Dashboard'),
    ui.input_select(
        'alert_type_subtype',
        'Select Alert Type and Subtype:',
        choices=dropdown_choices,
        selected=dropdown_choices[0]
    ),
    ui.input_switch('switch_button', 'Toggle to switch to range of hours',
    value=False),
    # show hour range first
    ui.panel_conditional('!input.switch_button',
                         ui.input_slider(id='hour_range', label='Select Hour
                         Range:', min=0, max=23, value=(6, 9), step=1,
                         animate=True, ticks=True)),
    # switch to specific hour when toggle is true
    ui.panel_conditional('input.switch_button',
                         ui.input_slider(id='single_hour', label='Select
                         Single Hour:', min=0, max=23, value=6, step=1,
                         animate=True, ticks=True)),
    output_widget('top_alerts_plot')
```

```
)

# server logic
def server(input, output, session):
    @output
    @render_altair
    def top_alerts_plot():
        # parse user selection
        selected_type, selected_subtype = input.alert_type_subtype().split('
        - ')

        # determine the hour range based on toggle
        if input.switch_button():
            # ensure hour is in 2-digit format
            hour = f'{input.single_hour():02}:00'
            filtered_data = filter_data_specific_hour(
                selected_type, selected_subtype, hour)
        else:
            hour_start, hour_end = input.hour_range()
            filtered_data = filter_data_by_range(
                selected_type, selected_subtype, hour_start, hour_end)

        lat_domain = [filtered_data['latitude_binned'].min(
        ) - 0.02, filtered_data['latitude_binned'].max() + 0.02]
        long_domain = [filtered_data['longitude_binned'].min(
        ) - 0.02, filtered_data['longitude_binned'].max() + 0.02]

        scatter_plot = alt.Chart(filtered_data).mark_circle().encode(
            x=alt.X('latitude_binned:Q', scale=alt.Scale(domain=lat_domain)),
            y=alt.Y('longitude_binned:Q',
            scale=alt.Scale(domain=long_domain)),
            size=alt.Size('alert_count:Q', title='Alert Count',
            scale=alt.Scale(
                domain=[filtered_data['alert_count'].min(),
                filtered_data['alert_count'].max()], range=[120, 350])),
            color=alt.Color('alert_count:Q', scale=alt.Scale(
                range=['skyblue', 'darkblue']), title='Alert Count'),
            tooltip=['latitude_binned', 'longitude_binned', 'alert_count']
        ).properties(
            title=f'Top 10 {selected_type} - {selected_subtype} Alerts',
            width=600, height=600
        )
        map_layer = alt.Chart(geo_data).mark_geoshape(
            fillOpacity=0, stroke='black'
        ).project(type='equirectangular').properties(
            width=600, height=600
        )

        combined_plot = map_layer + scatter_plot
        return combined_plot

app = App(app_ui, server)

# ChatGDP reference for "use ui.input_switch to switch between hour range and
specific hour and return corresponding chart"

```