Problem Set 6 - Waze Shiny Dashboard

Peter Ganong, Maggie Shi, and Andre Oviedo

2024-11-23

1. **ps6:** Due Sat 23rd at 5:00PM Central. Worth 100 points (80 points from questions, 10 points for correct submission and 10 points for code style) + 10 extra credit.

We use (*) to indicate a problem that we think might be time consuming.

Steps to submit (10 points on PS6)

1. "This submission is my work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: **sv**
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **sv** (2 point)
3. Late coins used this pset: **1/4** Late coins left after submission: **3/4**
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!*

```
/Users/saravanvalkenburgh/Library/Python/3.9/lib/python/site-packages/urllib3/__init__.py:35:
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.

```
# read in the sample data
df_sample = pd.read_csv("waze_data/waze_data_sample.csv")
```

```
print(df_sample.columns)
```

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

Variable Names/Data Types:

- Unnamed:0: Ordinal
- city: Nominal
- confidence: Ordinal
- nThumbsUp: Ordinal
- street: Nominal
- uuid: Ordinal
- country: Nominal
- type: Nominal
- subtype: Ordinal
- roadType: Ordinal
- reliability: Ordinal
- magvar: Ordinal

- reportRating: Ordinal

2.

```
# read in the data
df_full = pd.read_csv("waze_data/waze_data.csv")
```

```
# data frame for plotting with null counts for each variable
data = pd.DataFrame({
    'variable': df_full.columns,
    'null_count': df_full.isnull().sum(),
    'non_null_count': df_full.notnull().sum()
})

# melt the data
data_melted = data.melt(
    id_vars='variable', var_name='status', value_name='count')

# Create the bar chart
chart=alt.Chart(data_melted).mark_bar().encode(
    x=alt.X('variable:N', title='Variable'),
    y=alt.Y('count:Q', title='Number of Observations'),
    color=alt.Color('status:N', scale=alt.Scale(domain=['null_count', 'non_null_count'],
 ↪  range=['pink', 'purple']),
                    title='Status', legend=alt.Legend(title="Status"))
).properties(
    title='Null vs. Non-Null Observations'
)
```

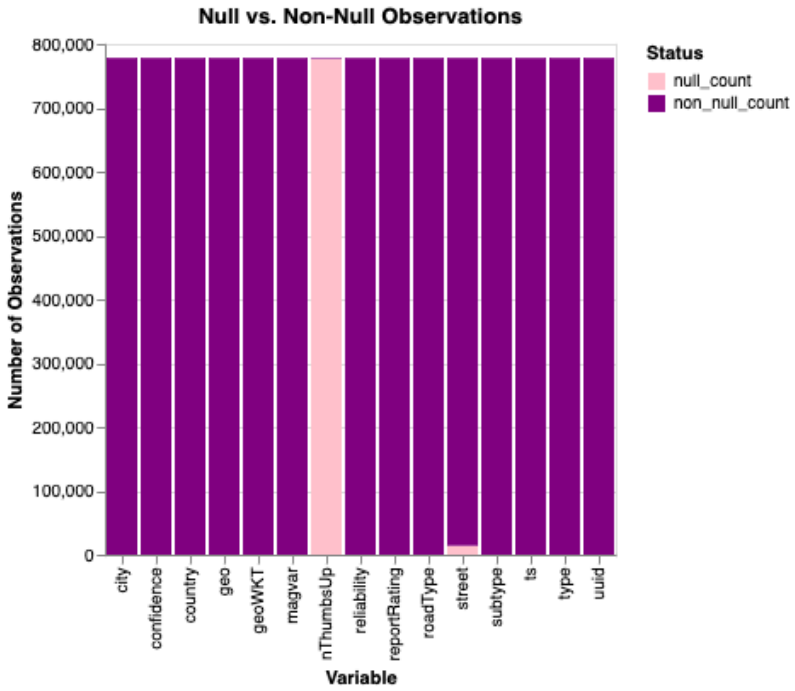

Figure 1: Bar Chart

nThumbsUp, street, and subtype have missing values. nThumbsUp has almost all values missing, while subtype has a much smaller number of missing values and street has a few missing values.

3.

```
#find unique types
unique_types = df_full['type'].unique()
print("Unique values in 'type':", unique_types)
```

```
Unique values in 'type': ['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']
```

```
# find unique subtypes
unique_subtypes = df_full['subtype'].unique()
print("Unique values in 'subtype':", unique_subtypes)
```

```
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']
```

```
# how many types have a subtype that is NA?
subtypes_count_nas = df_full[df_full['subtype'].isna()]['type'].nunique()
print(f"Number of types with NA subtypes: {subtypes_count_nas}")
```

```
Number of types with NA subtypes: 4
```

All four categories in the type columns have subtypes that have NAs.

- Traffic
  - Light
  - Moderate
  - Heavy
  - Stand-still
  - Unclassified
- Accident
  - Major
  - Minor
  - Unclassified
- Road Closed
  - Event
  - Construction
  - Hazard
  - Unclassified
- Hazard
  - On Road
    - Stopped car
    - Construction
    - Emergency vehicle
    - Ice
    - Object
    - Pot hole
    - Faulty traffic light
    - Lane closed
    - Road kill
  - On Shoulder
    - Stopped car
    - Animals
    - Missing Sign
  - Weather
    - Flood
    - Fog
    - Heavy snow
    - Hail
  - Unclassified

NAs: 12% of the subtype columns - 96,086 rows - have missing values. This seems like a lot of rows to omit from the data, so I think they should be kept and categorized as 'unclassified'.

```
# replace NAs with "unclassified"
df_full['subtype'] = df_full['subtype'].fillna('Unclassified')
```

4.

a.

```
# create the df
crosswalk = df_full.drop_duplicates(subset=["type", "subtype"]).copy()

# keep type and subtype columns - we will add the updated type, subtybe, and subsubtype in next
 ↪  step
crosswalk = crosswalk.loc[:, ["type", "subtype"]]
```

b.

```
crosswalk_data = []

# Loop over the combinations of type and subtype to populate the updated columns
for _, row in crosswalk.iterrows():
    type_val = row['type']
    subtype_val = row['subtype']

    updated_type = None
    updated_subtype = None
    updated_subsubtype = None

    if type_val == 'JAM':
        updated_type = 'Traffic'
        if subtype_val == 'JAM_LIGHT_TRAFFIC':
            updated_subtype = 'Light'
        elif subtype_val == 'JAM_MODERATE_TRAFFIC':
            updated_subtype = 'Moderate'
        elif subtype_val == 'JAM_HEAVY_TRAFFIC':
            updated_subtype = 'Heavy'
        elif subtype_val == 'JAM_STAND_STILL_TRAFFIC':
            updated_subtype = 'Stand-still'
        else:
            updated_subtype = 'Unclassified'
            updated_subsubtype = None

    elif type_val == 'ACCIDENT':
        updated_type = 'Accident'
        if subtype_val == 'ACCIDENT_MAJOR':
            updated_subtype = 'Major'
        elif subtype_val == 'ACCIDENT_MINOR':
            updated_subtype = 'Minor'
        else:
            updated_subtype = 'Unclassified'
            updated_subsubtype = None

    elif type_val == 'ROAD_CLOSED':
        updated_type = 'Road Closed'
        if subtype_val == 'ROAD_CLOSED_EVENT':
            updated_subtype = 'Event'
        elif subtype_val == 'ROAD_CLOSED_CONSTRUCTION':
            updated_subtype = 'Construction'
        elif subtype_val == 'ROAD_CLOSED_HAZARD':
            updated_subtype = 'Hazard'
        else:
            updated_subtype = 'Unclassified'
            updated_subsubtype = None
```

```
    elif type_val == 'HAZARD':
        updated_type = 'Hazard'

        if 'HAZARD_ON_ROAD' in subtype_val:
            updated_subtype = 'On Road'
            if 'CAR_STOPPED' in subtype_val:
                updated_subsubtype = 'Stopped car'
            elif 'CONSTRUCTION' in subtype_val:
                updated_subsubtype = 'Construction'
            elif 'EMERGENCY_VEHICLE' in subtype_val:
                updated_subsubtype = 'Emergency vehicle'
            elif 'ICE' in subtype_val:
                updated_subsubtype = 'Ice'
            elif 'OBJECT' in subtype_val:
                updated_subsubtype = 'Object'
            elif 'POT_HOLE' in subtype_val:
                updated_subsubtype = 'Pot hole'
            elif 'TRAFFIC_LIGHT_FAULT' in subtype_val:
                updated_subsubtype = 'Faulty traffic light'
            elif 'LANE_CLOSED' in subtype_val:
                updated_subsubtype = 'Lane closed'
            elif 'ROAD_KILL' in subtype_val:
                updated_subsubtype = 'Road kill'
            else:
                updated_subsubtype = 'Unclassified'

        elif 'HAZARD_ON_SHOULDER' in subtype_val:
            updated_subtype = 'On Shoulder'
            if 'CAR_STOPPED' in subtype_val:
                updated_subsubtype = 'Stopped car'
            elif 'ANIMALS' in subtype_val:
                updated_subsubtype = 'Animals'
            elif 'MISSING_SIGN' in subtype_val:
                updated_subsubtype = 'Missing Sign'
            else:
                updated_subsubtype = 'Unclassified'

        elif 'HAZARD_WEATHER' in subtype_val:
            updated_subtype = 'Weather'
            if 'FLOOD' in subtype_val:
                updated_subsubtype = 'Flood'
            elif 'FOG' in subtype_val:
                updated_subsubtype = 'Fog'
            elif 'HEAVY_SNOW' in subtype_val:
                updated_subsubtype = 'Heavy snow'
            elif 'HAIL' in subtype_val:
                updated_subsubtype = 'Hail'
            else:
                updated_subsubtype = 'Unclassified'

        else:
            updated_subtype = 'Unclassified'
            updated_subsubtype = None

    crosswalk_data.append(
        [type_val, subtype_val, updated_type, updated_subtype, updated_subsubtype])
```

```
# Create final crosswalk df
crosswalk = pd.DataFrame(crosswalk_data, columns=[
                         'type', 'subtype', 'updated_type', 'updated_subtype',
                          ↪  'updated_subsubtype'])

crosswalk.head()
```

	type	subtype	updated_type	updated_subtype	updated_subsubtype
0	JAM	Unclassified	Traffic	Unclassified	None
1	ACCIDENT	Unclassified	Accident	Unclassified	None
2	ROAD_CLOSED	Unclassified	Road Closed	Unclassified	None
3	HAZARD	Unclassified	Hazard	Unclassified	None
4	ACCIDENT	ACCIDENT_MAJOR	Accident	Major	None

c.

```
# merge with original data on "type" and "subtype"
merged_data = df_full.merge(crosswalk[['type', 'subtype', 'updated_type', 'updated_subtype',
 ↪  'updated_subsubtype']],
                            on=['type', 'subtype'], how='left')
```

```
merged_data.query(
    "updated_type == 'Accident' and updated_subtype == 'Unclassified'").shape[0]
```

24359

There are 24359 rows for Accident - Unclassified.

d. Extra Credit

```
# check that the crosswalk and new merged data have the same values in type and subtype
type_merged = merged_data['type'].unique()
type_crosswalk = crosswalk['type'].unique()

subtype_merged = merged_data['subtype'].unique()
subtype_crosswalk = crosswalk['subtype'].unique()

print("\nUnique 'type' values in merged dataset:", type_merged)
print("Unique 'type' values in crosswalk:", type_crosswalk)

print("\nUnique 'subtype' values in merged dataset:", subtype_merged)
print("Unique 'subtype' values in crosswalk:", subtype_crosswalk)
```

```
Unique 'type' values in merged dataset: ['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']
Unique 'type' values in crosswalk: ['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']

Unique 'subtype' values in merged dataset: ['Unclassified' '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']
Unique 'subtype' values in crosswalk: ['Unclassified' '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']
```

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

1.

a.

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


# apply the function to create a new column 'latlon_list'
merged_data['latlon_list'] = merged_data['geo'].apply(extract_coordinates)

# create new columns for latitude and longitude from the list column
merged_data[['longitude', 'latitude']] = pd.DataFrame(
    merged_data['latlon_list'].tolist(), index=merged_data.index)
```

Source: I put the following prompt into ChatGPT –
"I have a geoWKT column that contains data in this format: POINT(-87.624816 41.753358). Put together a regular expression that extracts the coordinates." and it gave me the expression in the code above.

I also used this post: https://stackoverflow.com/questions/35491274/split-a-pandas-column-of-lists-into-multiple-columns

b.

```python
# create the bins
step = 0.01
merged_data["latBin"] = (
    (merged_data.latitude / step).round()).astype(float) * step
```

```
merged_data["lonBin"] = (
    (merged_data.longitude / step).round()).astype(float) * step

# make sure all bins rounded to two decimal places
merged_data["latBin"] = merged_data["latBin"].map(lambda x: f"{x:.2f}")
merged_data["lonBin"] = merged_data["lonBin"].map(lambda x: f"{x:.2f}")
```

```
merged_data.to_csv('top_alerts_map/merged_data.csv', index=False)
```

```
# create the groups and see which combination has the greatest number of observations
groups = merged_data.groupby(["latBin", "lonBin"]).size()
max_group = groups.idxmax()
max_count = groups.max()

print(f"The latitude-longitude combination with the greatest number of observations is
 ↪  {max_group} with {max_count} observations.")
```

```
The latitude-longitude combination with the greatest number of observations is ('41.88',
'-87.65') with 21325 observations.
```

Source: https://stackoverflow.com/questions/39254704/pandas-group-bins-of-data-per-longitude-latitude

c.

```
# collapse data to top ten bins for minor accidents and create df
top_alerts_df = (merged_data[(merged_data['type'] == 'ACCIDENT') & (merged_data['subtype'] ==
 ↪  'ACCIDENT_MINOR')]
                 .groupby(['latBin', 'lonBin'])
                 .size()
                 .reset_index(name="alert_count")
                 .sort_values(by="alert_count", ascending=False)
                 .head(10))

# save to csv
top_alerts_df.to_csv('top_alerts_map/top_alerts_map.csv', index=False)
```

The level of aggregation in this case is the number of minor accidents per certain lat/lon bins. There are 10 rows in the dataframe because we filtered to the top 10 lat/lon bins in terms of number of alerts for this chosen type and subtype.

2.

```
# create the top ten df for Jam - Heavy Traffic alerts
jam_heavy_traffic_df = (merged_data[(merged_data['type'] == 'JAM') & (merged_data['subtype'] ==
 ↪  'JAM_HEAVY_TRAFFIC')]
                 .groupby(['latBin', 'lonBin'])
                 .size()
                 .reset_index(name="alert_count")
                 .sort_values(by="alert_count", ascending=False)
                 .head(10))

# make the plot
alt.Chart(jam_heavy_traffic_df).mark_circle().encode(
    latitude='latBin:Q',
    longitude='lonBin:Q',
```

```
    size=alt.Size(
        'alert_count:Q',
        scale=alt.Scale(domain=[2000, 4500], range=[50, 500]),
        title='Number of Alerts')
).properties(
    title='Top 10 Latitude-Longitude Bins for "Jam - Heavy Traffic" Alerts')
```

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

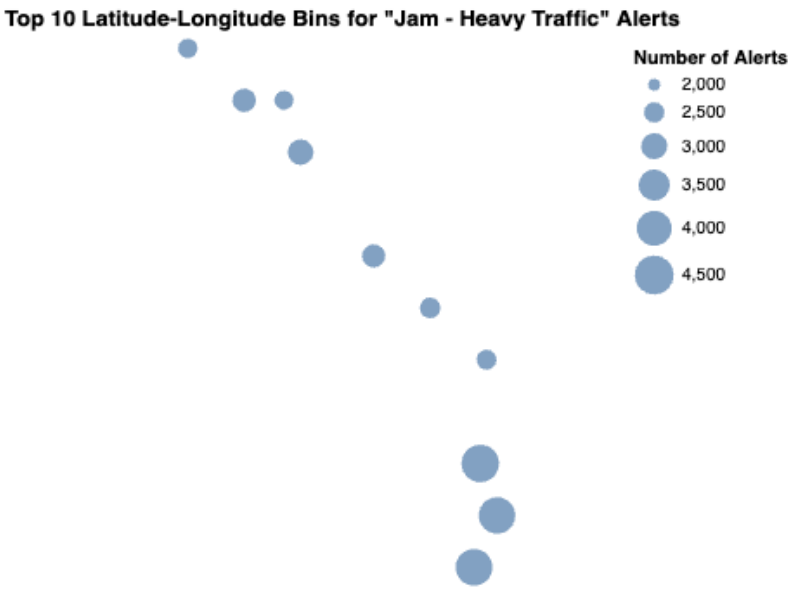

Figure 2: scatter_plot

3.

a.

```
url = "https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON"
output_file =
 ↪  "/Users/saravanvalkenburgh/Documents/GitHub/pset6/top_alerts_map/chicago_neighborhood_boundaries.geojso
response = requests.get(url)

if response.status_code == 200:
    with open(output_file, "wb") as file:
        file.write(response.content)
```

Source: https://realpython.com/python-download-file-from-url/

b.

```
file_path =
 ↪  "/Users/saravanvalkenburgh/Documents/GitHub/pset6/top_alerts_map/chicago_neighborhood_boundaries.geojso

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

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

4.

a.

```
# create the map from the geoJSON file
background = alt.Chart(geo_data).mark_geoshape(
    fill="lightgray",
    stroke="black"
).project(type="equirectangular")


# create the scatter plot layer
points = alt.Chart(jam_heavy_traffic_df).mark_circle().encode(
    latitude='latBin:Q',
    longitude='lonBin:Q',
    size=alt.Size(
        'alert_count:Q',
        scale=alt.Scale(domain=[2400, 4400], range=[50, 500]),
        title='Number of Alerts'
    ),
    color=alt.value('blue')
)

# Layer the two plots
layered_plot = (background + points).properties(
    title="Chicago Neighborhood Boundaries with 'Jam - Heavy Traffic' Alerts"
)
```

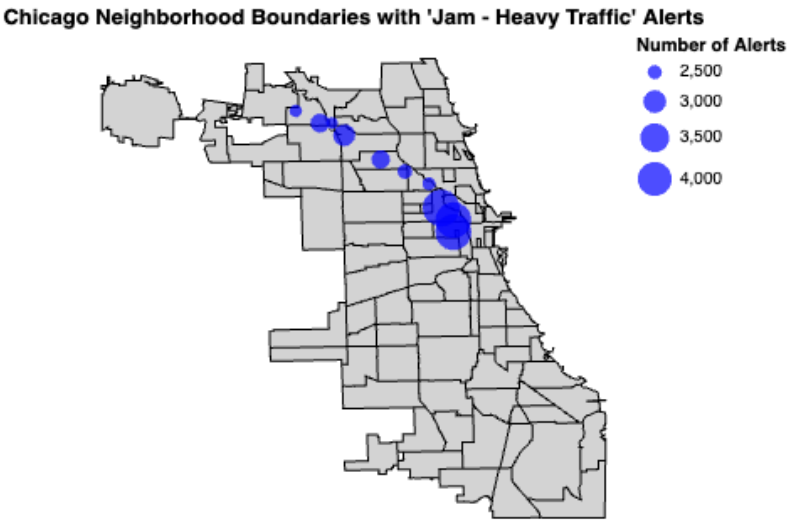

Figure 3: layered_plot

5.

a. As you can see in the screenshot below, there are 16 combinations of type and subtype in the dropdown menu.

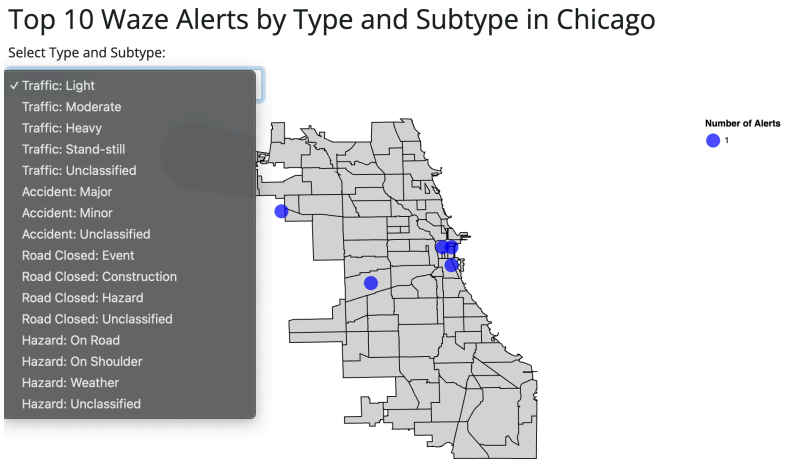

Figure 4: Dropdown Menu

b. Jam - Heavy Traffic plot from above

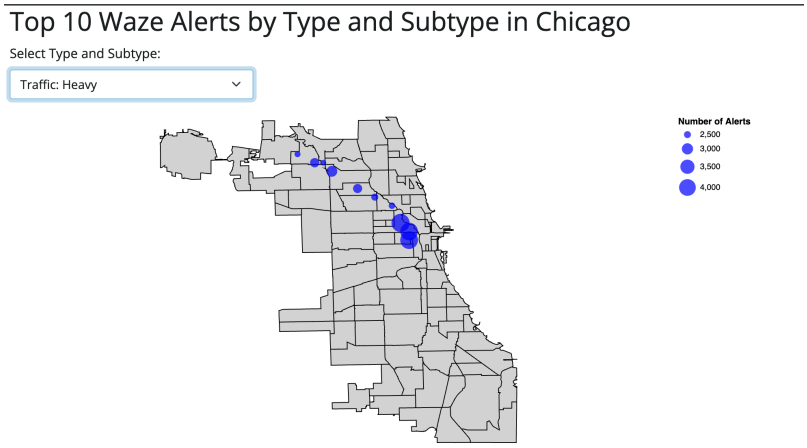

Figure 5: Jam - Heavy Traffic

c. Alerts for road closures due to events are more common in the Northwest part of the city, as well as around what looks like the United Center and some in the Southwest part of the city. There are also some around Wrigley Field and Guaranteed Rate Field, and fewer but still some around Soldier Field.

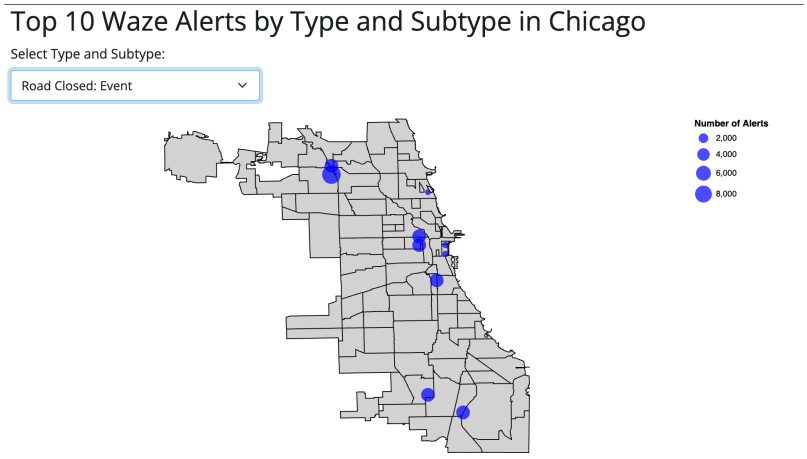

Figure 6: Road Closed - Event

d. Another question could be: Where are alerts for road closures due to construction the most common?

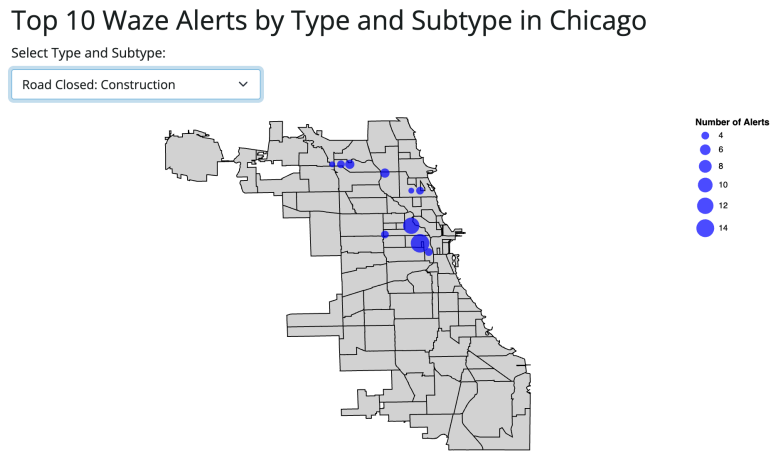

Figure 7: Road Closed - Construction

Answer: Alerts for road closures due to construction are more common on the West and North sides of the city. There are no alerts for road closures due to contruction on the Southside in the top 10, which could say something about the city's priorities in terms of infrastructure projects.

e. Another column that could enhance our analysis is the road type column. This would enable us to analyze which types of alerts are more prevalent on specific kinds of roads, such as primary streets, secondary streets, freeway, exit ramps, etc, which could be helpful in terms of urban and city planning and where to target infrastructure improvements.

App.py for App #1

```
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/app.py")
```

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

types_and_subtypes = {
    "Traffic": ["Light", "Moderate", "Heavy", "Stand-still", "Unclassified"],
    "Accident": ["Major", "Minor", "Unclassified"],
    "Road Closed": ["Event", "Construction", "Hazard", "Unclassified"],
    "Hazard": ["On Road", "On Shoulder", "Weather", "Unclassified"]
}

app_ui = ui.page_fluid(
    ui.h2("Top 10 Waze Alerts by Type and Subtype in Chicago"),
    ui.input_select(
        id="type_subtype_dropdown",
        label="Select Type and Subtype:",
        choices=[f"{t}: {s}" for t, subtypes in types_and_subtypes.items()
                 for s in subtypes]
    ),
    output_widget("layered_plot")
)


def server(input, output, session):
    @reactive.calc
    def full_data():
        return pd.read_csv("merged_data.csv")

    @reactive.calc
    def geo_data():
        with open("chicago_neighborhood_boundaries.geojson", "r") as f:
            chicago_geojson = json.load(f)
        return alt.Data(values=chicago_geojson["features"])

    @reactive.calc
    def filtered_data():
        selected = input.type_subtype_dropdown()
```

```
        selected_type, selected_subtype = selected.split(": ")

        data = full_data()
        filtered = (
            data[
                (data["updated_type"] == selected_type) &
                (data["updated_subtype"] == selected_subtype)
            ]
            .groupby(["latBin", "lonBin"])
            .size()
            .reset_index(name="alert_count")
            .sort_values(by="alert_count", ascending=False)
            .head(10)
        )
        return filtered

    @render_altair
    def layered_plot():
        data = filtered_data()
        geo_data_values = geo_data()

        alert_min = data['alert_count'].min()
        alert_max = data['alert_count'].max()

        map_layer = alt.Chart(geo_data_values).mark_geoshape(
            fill="lightgray",
            stroke="black"
        ).project(type='equirectangular'
        ).properties(
            width=800,
            height=400
        )

        scatter_layer = alt.Chart(data).mark_circle(size=100).encode(
            latitude='latBin:Q',
            longitude='lonBin:Q',
            size=alt.Size(
                'alert_count:Q',
                scale=alt.Scale(
                    domain=[alert_min, alert_max], range=[50, 500]),
                title='Number of Alerts'
            ),
            color=alt.value('blue')
        )

        return map_layer + scatter_layer


app = App(app_ui, server)

```

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

1.

a. The ts column is the timestamp of the reported alert. I think it makes sense to collapse the data by this column, but we need to be careful not to collapse our data TOO much, such as by date. It makes more sense to collapse the data by hour so that it does not result in too significant of data reduction, which could impact our analysis and ability to see patterns.

b.

```
# create a new variable called hour
merged_data['hour'] = pd.to_datetime(merged_data['ts']).dt.strftime('%H')

# convert to integers to better interact with slider
merged_data['hour'] = merged_data['hour'].astype(int)
```

```
# new collapsed dataset
collapsed_df = merged_data.groupby(
    ['hour', 'updated_type', 'updated_subtype', 'latBin', 'lonBin']
).size(
).reset_index(name="alert_count"
).sort_values(by="alert_count", ascending=False)
```

```
# count the number of rows
print(len(collapsed_df))
```

62825

This dataset has 62825 rows.

```
# save to csv
collapsed_df.to_csv('top_alerts_map_byhour/top_alerts_map_byhour.csv', index=False)
```

c.

```
# filter data for top ten locations by hour for "Jam - Heavy Traffic"

#7pm
filter_1900 = collapsed_df[(collapsed_df['updated_type'] == 'Traffic') &
 ↪  (collapsed_df['updated_subtype'] == 'Heavy') & (collapsed_df['hour'] == '19')].head(10)

#3pm
filter_1500 = collapsed_df[(collapsed_df['updated_type'] == 'Traffic') &
 ↪  (collapsed_df['updated_subtype'] == 'Heavy') & (collapsed_df['hour'] == '15')].head(10)

#12pm
filter_1200 = collapsed_df[(collapsed_df['updated_type'] == 'Traffic') &
 ↪  (collapsed_df['updated_subtype'] == 'Heavy') & (collapsed_df['hour'] == '12')].head(10)
```

```
# 7pm plot
background = alt.Chart(geo_data).mark_geoshape(
    fill="lightgray",
    stroke="black"
).project(type="equirectangular")

points = alt.Chart(filter_1900).mark_circle(
    stroke="black",
    strokeWidth=1
```

```
).encode(
    latitude='latBin:Q',
    longitude='lonBin:Q',
    size=alt.Size(
        'alert_count:Q',
        title='Number of Alerts'
    )
)

layered_plot1900 = (background + points).properties(
    title="Top 10 Locations for 'Jam - Heavy Traffic' Alerts: 7pm"
)
# 3pm plot
background = alt.Chart(geo_data).mark_geoshape(
    fill="lightgray",
    stroke="black"
).project(type="equirectangular")

points = alt.Chart(filter_1500).mark_circle(
    stroke="black",
    strokeWidth=1
).encode(
    latitude='latBin:Q',
    longitude='lonBin:Q',
    size=alt.Size(
        'alert_count:Q',
        title='Number of Alerts'
    )
)

layered_plot1500 = (background + points).properties(
    title="Top 10 Locations for 'Jam - Heavy Traffic' Alerts: 3pm"
)

# 12pm plot
background = alt.Chart(geo_data).mark_geoshape(
    fill="lightgray",
    stroke="black"
).project(type="equirectangular")

points = alt.Chart(filter_1200).mark_circle(
    stroke="black",
    strokeWidth=1
).encode(
    latitude='latBin:Q',
    longitude='lonBin:Q',
    size=alt.Size(
        'alert_count:Q',
        title='Number of Alerts'
    )
)

layered_plot1200 = (background + points).properties(
    title="Top 10 Locations for 'Jam - Heavy Traffic' Alerts: 12pm"
)
```

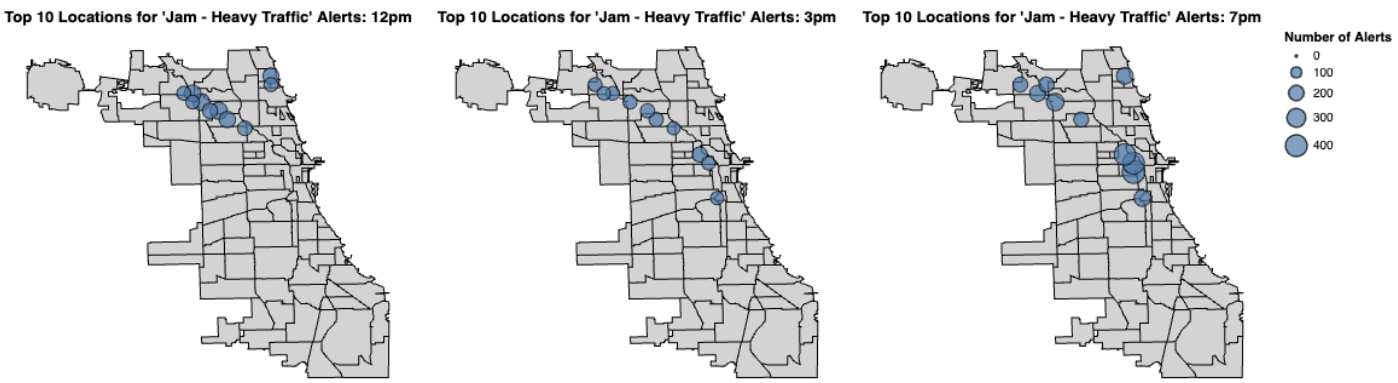

Figure 8: layered_plot1200 | layered_plot1500 | layered_plot1900

2.

a. What does the UI look like?

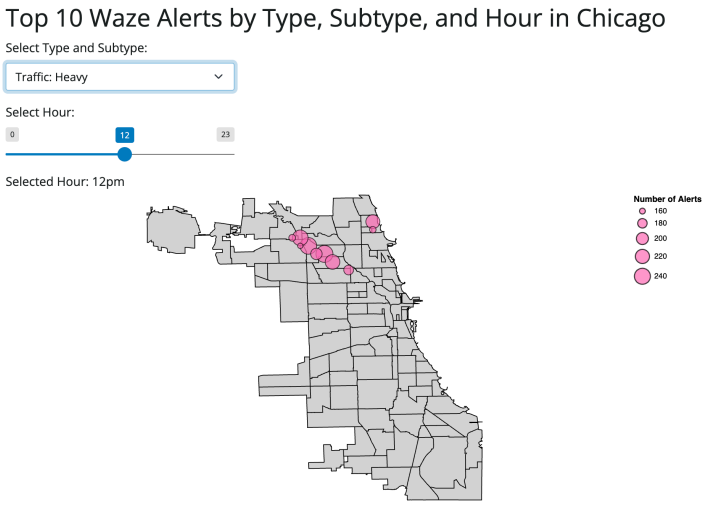

Figure 9: App2 UI

b. Recreate plots from above

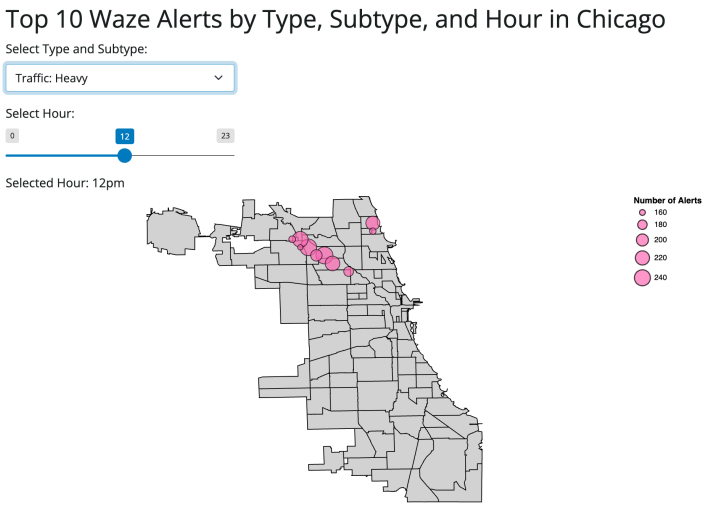

Figure 10: Jam - Heavy Traffic - 12pm

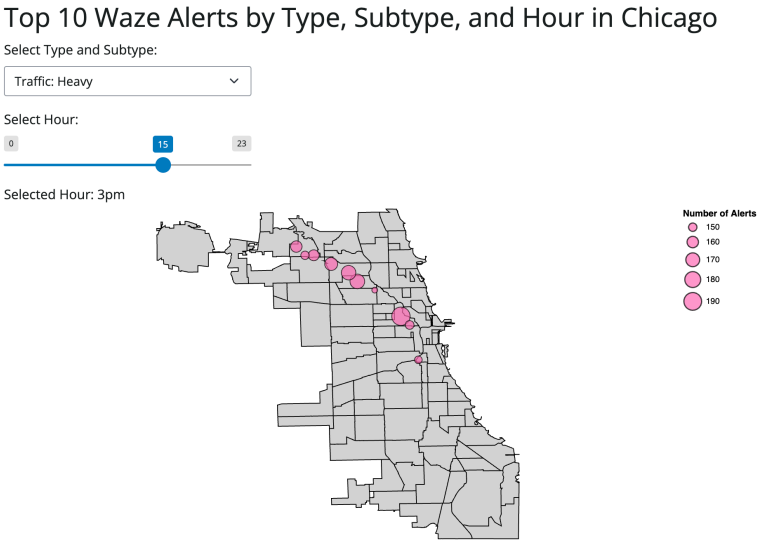

Figure 11: Jam - Heavy Traffic - 3pm

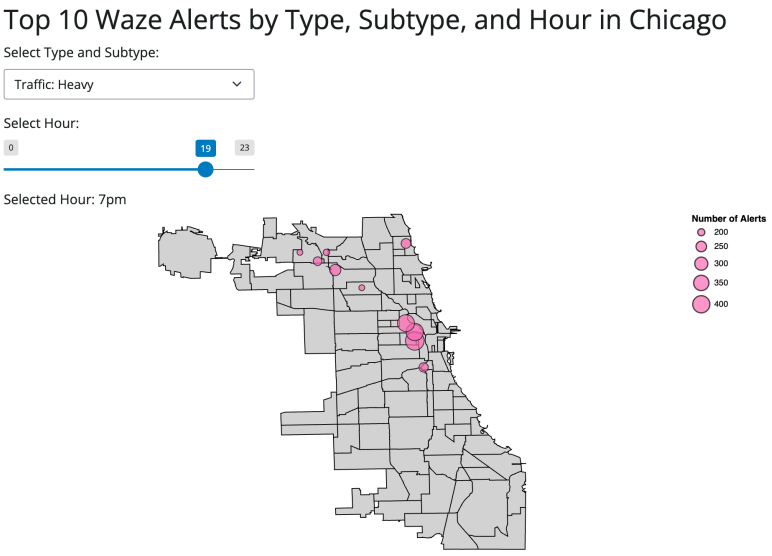

Figure 12: Jam - Heavy Traffic - 7pm

c. It seems as though road construction is done more during the night hours, as seen by the plots below.

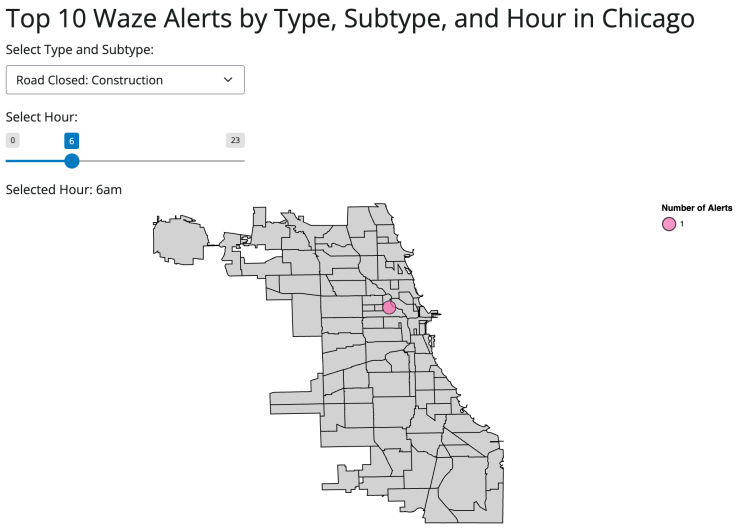

Figure 13: Road Closed: Construction - 6am

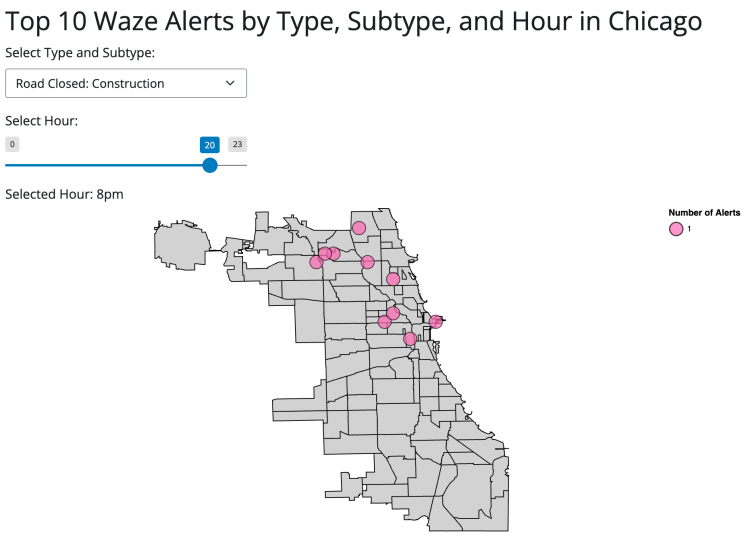

Figure 14: Road Closed: Construction - 8am

App.py for App #2

```
print_file_contents("./top_alerts_map_byhour/app.py")
```

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

types_and_subtypes = {
    "Traffic": ["Light", "Moderate", "Heavy", "Stand-still", "Unclassified"],
    "Accident": ["Major", "Minor", "Unclassified"],
    "Road Closed": ["Event", "Construction", "Hazard", "Unclassified"],
    "Hazard": ["On Road", "On Shoulder", "Weather", "Unclassified"]
}

def format_hour(hour):
    if hour == 0:
        return "12am"
    elif hour == 12:
        return "12pm"
    elif hour < 12:
        return f"{hour}am"
    else:
        return f"{hour - 12}pm"

app_ui = ui.page_fluid(
    ui.h2("Top 10 Waze Alerts by Type, Subtype, and Hour in Chicago"),
    ui.input_select(
        id="type_subtype_dropdown",
        label="Select Type and Subtype:",
```
```

```
        choices=[f"{t}: {s}" for t, subtypes in types_and_subtypes.items()
                 for s in subtypes]
    ),
    ui.input_slider("hour_slider", "Select Hour:",
                    0, 23, 12),
    ui.output_text("selected_hour_text"),
    output_widget("layered_plot"),
    ui.output_text("no_alerts_message")
)


def server(input, output, session):
    @ reactive.calc
    def full_data():
        return pd.read_csv("top_alerts_map_byhour.csv")

    @ reactive.calc
    def geo_data():
        with open("chicago_neighborhood_boundaries.geojson", "r") as f:
            chicago_geojson = json.load(f)
        return alt.Data(values=chicago_geojson["features"])

    @ reactive.calc
    def filtered_data():
        selected = input.type_subtype_dropdown()

        selected_type, selected_subtype = selected.split(": ")

        selected_hour = input.hour_slider()

        data = full_data()
        filtered = (
            data[
                (data["updated_type"] == selected_type) &
                (data["updated_subtype"] == selected_subtype) &
                (data["hour"] == selected_hour)
            ]
            .head(10)
        )
        return filtered

    @render_altair
    def layered_plot():
        data = filtered_data()
        geo_data_values = geo_data()

        if data.empty:
            return None

        alert_min = data['alert_count'].min()
        alert_max = data['alert_count'].max()

        map_layer = alt.Chart(geo_data_values).mark_geoshape(
            fill="lightgray",
            stroke="black"
        ).project(type='equirectangular'
                  ).properties(
```

```
            width=800,
            height=400
        )

        scatter_layer = alt.Chart(data).mark_circle(stroke="black",
                                                    strokeWidth=1
                                                    ).encode(
            latitude='latBin:Q',
            longitude='lonBin:Q',
            size=alt.Size(
                'alert_count:Q',
                scale=alt.Scale(
                    domain=[alert_min, alert_max], range=[50, 500]),
                title='Number of Alerts'
            ),
            color=alt.value('hotpink')
        )

        return map_layer + scatter_layer

    @render.text
    def no_alerts_message():
        data = filtered_data()
        if data.empty:
            return "No alerts found for this type, subtype, and hour combination."
        return ""

    @render.text
    def selected_hour_text():
        selected_hour = input.hour_slider()
        formatted_hour = format_hour(selected_hour)
        return f"Selected Hour: {formatted_hour}"

app = App(app_ui, server)
```

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

1.

a. Yes, I think it would be a good idea to collapse the data by a range of hours. This will provide a more general view of the data make the UI easier for viewers, who will have fewer options to cycle through. It will make it easier to get a big picture idea of the types of alerts happening in Chicago for general times of day. It will also eliminate the problem I had in app 2 where some of the alerts had 0 or only 1 observation for certain hours.

b.

```
# using collapsed_df from before, filter the data for 6-9am
filtered_data = collapsed_df[
    (collapsed_df["updated_type"] == "Traffic") &
    (collapsed_df["updated_subtype"] == "Heavy") &
    (collapsed_df["hour"] >= 6) & (collapsed_df["hour"] <= 9)
].groupby(["latBin", "lonBin"]).agg(
    alert_count=("alert_count", "sum")
).reset_index(
```

```
).head(10)

# create the map from the geoJSON file
background = alt.Chart(geo_data).mark_geoshape(
    fill="lightgray",
    stroke="black"
).project(type="equirectangular")


# create the scatter plot layer
points = alt.Chart(filtered_data).mark_circle(stroke="black",
                                              strokeWidth=1
                                              ).encode(
    latitude='latBin:Q',
    longitude='lonBin:Q',
    size=alt.Size(
        'alert_count:Q',
        title='Number of Alerts'
    ),
    color=alt.value('green')
)

# Layer the two plots
layered_plot = (background + points).properties(
    title="Chicago Neighborhood Boundaries with 'Jam - Heavy Traffic' Alerts from 6-9am"
)
```

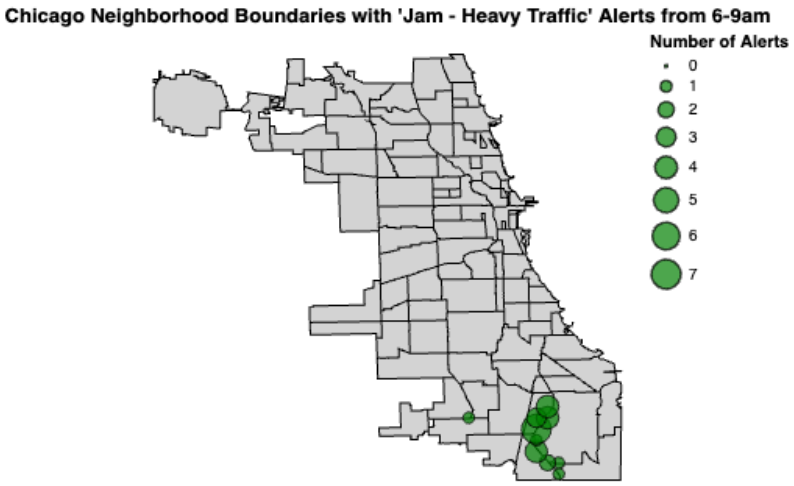

Figure 15: Jam: Heavy Traffic from 6-9am

2.

a. UI for App 3

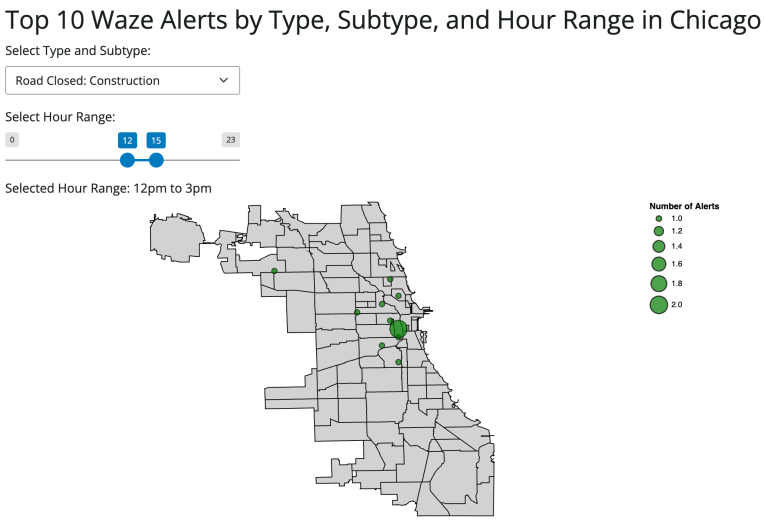

Figure 16: App 3 UI

b. Recreate plot for Jam: Heavy Traffic from 6-9am

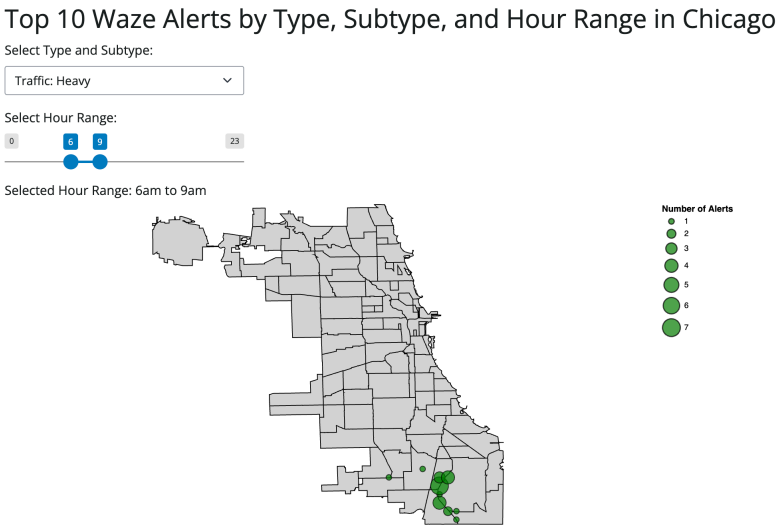

Figure 17: Jam: Heavy Traffic - 6-9am

3.

a. Screenshot of app with added switch button:

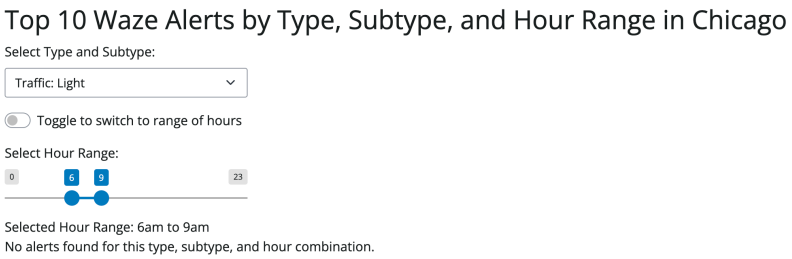

Figure 18: Added Switch Button

The possible values would be TRUE when the switch is in the "on" position and FALSE when the switch is in the "off" position

b.

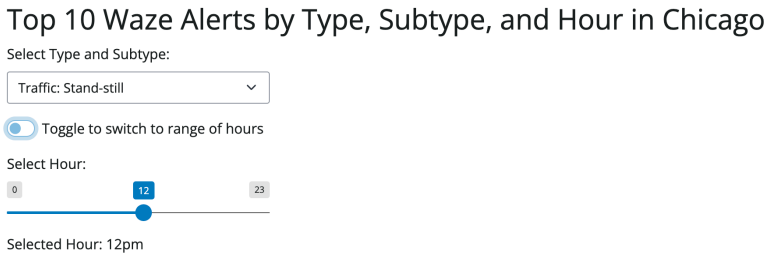

Figure 19: Not Toggled

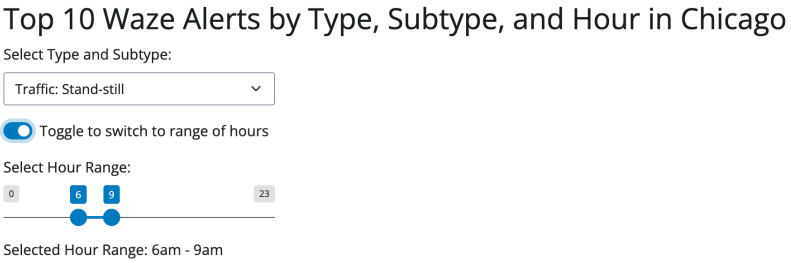

Figure 20: Toggled

c.

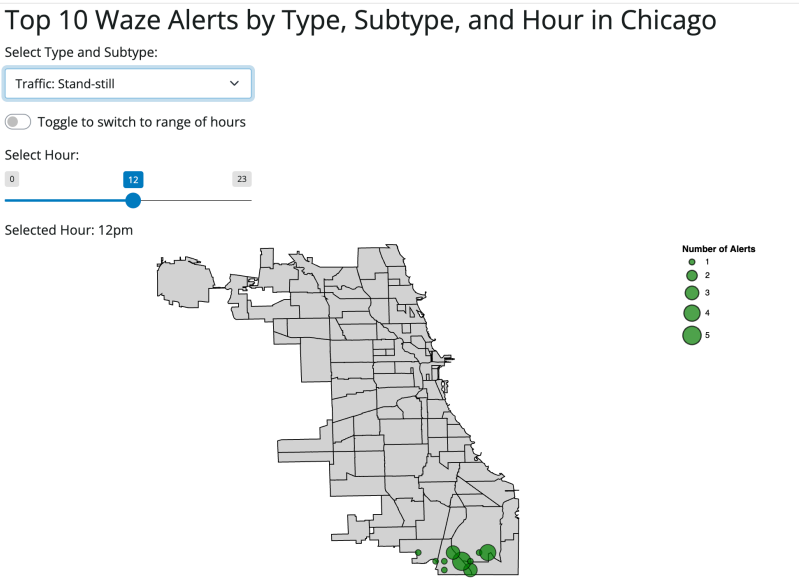

Figure 21: Not Toggled

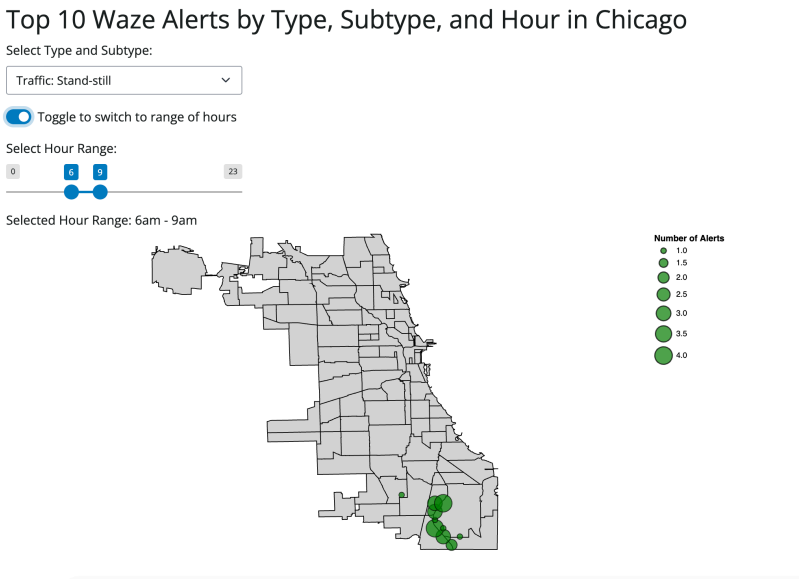

Figure 22: Toggled

d. Extra Credit

In order to achieve a plot similar to the one below, I would need to add gridlines to my plot and labels with latitude and longtitude. I would need to filter the data in the server logic to distinguish between morning and afternoon hours, and then adjust my plot to display blue open circles for afternoon and red closed circles for morning. I would get rid of the hour slider and the toggle swith; they would no longer be needed; or alternatively, I would leave those in on the first page and create a second page where the plot with morning vs. afternoon hours displays.

App.py for App #3

```
print_file_contents("./top_alerts_map_byhour_sliderrange/app.py")
```

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

types_and_subtypes = {
    "Traffic": ["Light", "Moderate", "Heavy", "Stand-still", "Unclassified"],
    "Accident": ["Major", "Minor", "Unclassified"],
    "Road Closed": ["Event", "Construction", "Hazard", "Unclassified"],
    "Hazard": ["On Road", "On Shoulder", "Weather", "Unclassified"]
}

def format_hour(hour):
    if hour == 0:
        return "12am"
    elif hour == 12:
        return "12pm"
    elif hour < 12:
        return f"{hour}am"
    else:
        return f"{hour - 12}pm"


app_ui = ui.page_fluid(
    ui.h2("Top 10 Waze Alerts by Type, Subtype, and Hour in Chicago"),
    ui.input_select(
        id="type_subtype_dropdown",
        label="Select Type and Subtype:",
        choices=[f"{t}: {s}" for t, subtypes in types_and_subtypes.items()
                 for s in subtypes]
    ),
    ui.input_switch("switch_button",
                    "Toggle to switch to range of hours", value=False),
    ui.output_ui("hour_slider_ui"),
    ui.output_text("selected_hour_text"),
    output_widget("layered_plot"),
    ui.output_text("no_alerts_message")
)

def server(input, output, session):
    @reactive.calc
    def full_data():
        return pd.read_csv("top_alerts_map_byhour.csv")

    @reactive.calc
    def geo_data():
        with open("chicago_neighborhood_boundaries.geojson", "r") as f:
            chicago_geojson = json.load(f)
        return alt.Data(values=chicago_geojson["features"])

    @reactive.calc
    def filtered_data():
        selected = input.type_subtype_dropdown()
```

```
    selected_type, selected_subtype = selected.split(": ")

    if input.switch_button():
        start_hour, end_hour = input.hour_slider_range()
    else:
        selected_hour = input.hour_slider()
        start_hour = selected_hour
        end_hour = selected_hour

    data = full_data()
    filtered = (
        data[
            (data["updated_type"] == selected_type) &
            (data["updated_subtype"] == selected_subtype) &
            (data["hour"] >= start_hour) &
            (data["hour"] <= end_hour)
        ]
        .groupby(["latBin", "lonBin"]).agg(
            alert_count=("alert_count", "sum")
        ).reset_index(
        ).head(10)
    )
    return filtered

@render_altair
def layered_plot():
    data = filtered_data()
    geo_data_values = geo_data()

    if data.empty:
        return None

    alert_min = data['alert_count'].min()
    alert_max = data['alert_count'].max()

    map_layer = alt.Chart(geo_data_values).mark_geoshape(
        fill="lightgray",
        stroke="black"
    ).project(type='equirectangular'
              ).properties(
        width=800,
        height=400
    )

    scatter_layer = alt.Chart(data).mark_circle(stroke="black",
                                                strokeWidth=1
                                                ).encode(
        latitude='latBin:Q',
        longitude='lonBin:Q',
        size=alt.Size(
            'alert_count:Q',
            scale=alt.Scale(
                domain=[alert_min, alert_max], range=[50, 500]),
            title='Number of Alerts'
        ),
        color=alt.value('green')
    )
```

```
        return map_layer + scatter_layer

    @render.text
    def no_alerts_message():
        data = filtered_data()
        if data.empty:
            return "No alerts found for this type, subtype, and hour combination."
        return ""

    @render.text
    def selected_hour_text():
        if input.switch_button():
            start_hour, end_hour = input.hour_slider_range()
            formatted_range = f"Selected Hour Range: {format_hour(start_hour)} -
            {format_hour(end_hour)}"
            return formatted_range
        else:
            selected_hour = input.hour_slider()
            formatted_hour = format_hour(selected_hour)
            return f"Selected Hour: {formatted_hour}"

    @render.ui
    def hour_slider_ui():
        if input.switch_button():
            return ui.input_slider(
                "hour_slider_range",
                "Select Hour Range:",
                min=0, max=23,
                value=[6, 9]
            )
        else:
            return ui.input_slider("hour_slider", "Select Hour:", 0, 23, 12)


app = App(app_ui, server)
```