Problem Set 6 - Waze Shiny Dashboard

Sarah Morrison

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

```python
def print_file_contents(file_path):
    """Print contents of a file."""
    try:
        with open(file_path, 'r') as f:
            content = f.read()
            print("```python")
            print(content)
            print("```")
    except FileNotFoundError:
        print("```python")
        print(f"Error: File '{file_path}' not found")
        print("```")
    except Exception as e:
        print("```python")
        print(f"Error reading file: {e}")
        print("```")

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

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

app_ui = ui.page_fluid(
    ui.input_select(id = 'type_subtype', label = 'Select a Type-Subtype
    Combination:',
    choices = ["Accident - Major", "Accident - Minor", "Accident -
    Unclassified",
        "Jam - Heavy Traffic", "Jam - Light Traffic", "Jam - Moderate
        Traffic",
        "Jam - Stand-Still Traffic", "Jam - Unclassified", "Hazard - On
        Road",
        "Hazard - On Shoulder", "Hazard - Weather", "Hazard - Unclassified",
```

```
        "Road Closed - Construction", "Road Closed - Event", "Road Closed -
        Hazard",
        "Road Closed - Unclassified"]),
    ui.input_slider('hour', 'Hour of Day', 0, 23, 1),
    output_widget('scatter_plot'),
    ui.output_table("filtered_table"),
)


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


    @reactive.calc
    def filtered_data():
        # Filter data based on user input
        df = full_data()
        selected_combination = input.type_subtype()
        selected_hour = input.hour()

        # Apply filters for type_subtype and hour
        filtered_df = df[(df["type_subtype"] == selected_combination) &
        (df['hour'] == selected_hour)]
        return filtered_df

    @render.table()
    def filtered_table():
        return filtered_data()


    @render_altair
    def scatter_plot():
        # Read and convert GeoDataFrame to the appropriate format for Altair
        geo_data =
        gpd.read_file("./top_alerts_map/chicago-boundaries.geojson")

        # Confirm longitude and latitude are numeric
        filtered = filtered_data()
        filtered['longitude'] = pd.to_numeric(filtered['longitude'])
        filtered['latitude'] = pd.to_numeric(filtered['latitude'])
```

```
        # Create base map chart
        map_chart = alt.Chart(geo_data).mark_geoshape(
            fill=None,  # Transparent fill
            stroke="lightgray"
        ).properties(
            width=350,
            height=350
        ).project(
            type='identity',  # Assume coordinates are in WGS84
            reflectY=True
        )

        scatter_chart = alt.Chart(filtered).mark_circle().encode(
            longitude='longitude:Q',
            latitude='latitude:Q',
            size=alt.Size(
                'Count:Q',
                title='Number of Observations',
                scale=alt.Scale(type='pow', exponent=2, range=[1, 200]))
        )


        combined_chart = (map_chart + scatter_chart).configure_view(
            stroke=None  # Remove the border of the map chart
        ).properties(
            title=f"Chicago Neighborhood Map with {input.type_subtype()}
            Observations at {input.hour()} O'Clock"
        ).configure_mark(
            opacity=0.7,
            color='darkblue'
        )

        return combined_chart

app = App(app_ui, server)


```
```python
from shiny import App, render, ui, reactive
import pandas as pd
import anywidget
from shinywidgets import render_altair, output_widget
```

```
import altair as alt
import json
import geopandas as gpd

app_ui = ui.page_fluid(
    ui.input_select(id = 'type_subtype', label = 'Select a Type-Subtype
    Combination:',
    choices = ["Accident - Major", "Accident - Minor", "Accident -
    Unclassified",
        "Jam - Heavy Traffic", "Jam - Light Traffic", "Jam - Moderate
        Traffic",
        "Jam - Stand-Still Traffic", "Jam - Unclassified", "Hazard - On
        Road",
        "Hazard - On Shoulder", "Hazard - Weather", "Hazard - Unclassified",
        "Road Closed - Construction", "Road Closed - Event", "Road Closed -
        Hazard",
        "Road Closed - Unclassified"]),
    output_widget('scatter_plot'),
    ui.output_table("filtered_table"),
)


def server(input, output, session):
    @reactive.calc
    def full_data():
        return pd.read_csv("top_alerts_map/top_10_alerts.csv")


    @reactive.calc
    def filtered_data():
        df = full_data()
        selected_combination = input.type_subtype()
        return df[df["type_subtype"] == selected_combination]

    @render.table()
    def filtered_table():
        return filtered_data()


    @render_altair
    def scatter_plot():
        # Read and convert GeoDataFrame to the appropriate format for Altair
```

```
geo_data =
gpd.read_file("./top_alerts_map/chicago-boundaries.geojson")

# Confirm longitude and latitude are numeric
filtered = filtered_data()
filtered['longitude'] = pd.to_numeric(filtered['longitude'])
filtered['latitude'] = pd.to_numeric(filtered['latitude'])

# Create base map chart
map_chart = alt.Chart(geo_data).mark_geoshape(
    fill=None,  # Transparent fill
    stroke="lightgray"
).properties(
    width=350,
    height=350
).project(
    type='identity',  # Assume coordinates are in WGS84
    reflectY=True
)

scatter_chart = alt.Chart(filtered, title='Top 10 Locations for Heavy
Traffic Jams').mark_circle().encode(
    longitude='longitude:Q',
    latitude='latitude:Q',
    size=alt.Size(
        'Count:Q',
        title='Number of Observations',
        scale=alt.Scale(type='pow', exponent=2, range=[1, 200]))
)


combined_chart = (map_chart + scatter_chart).configure_view(
    stroke=None  # Remove the border of the map chart
).properties(
    title=f"Chicago Neighborhood Map with {input.type_subtype()}
    Observations"  # Use f-string to combine all parts into the title
).configure_mark(
    opacity=0.7,
    color='darkblue'
)

return combined_chart
```

```
app = App(app_ui, server)


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

app_ui = ui.page_fluid(
    ui.input_select(id='type_subtype',
                    label='Select a Type-Subtype Combination:',
                    choices=["Accident - Major", "Accident - Minor",
                    "Accident - Unclassified",
                    "Jam - Heavy Traffic", "Jam - Light Traffic", "Jam -
                    Moderate Traffic",
                    "Jam - Stand-Still Traffic", "Jam - Unclassified",
                    "Hazard - On Road",
                    "Hazard - On Shoulder", "Hazard - Weather", "Hazard -
                    Unclassified",
                    "Road Closed - Construction", "Road Closed - Event",
                    "Road Closed - Hazard",
                    "Road Closed - Unclassified"]),
    ui.input_switch(id='switch',
                    label='Toggle to switch to range of hours'),
    ui.panel_conditional(
                    "input.switch",
                    ui.input_slider('hour', 'Hour of Day', 0, 23, 1),
                    output_widget('scatter_plot_single'),
                    ui.output_table("filtered_table_single"),
    ),
    ui.panel_conditional(
                    "!input.switch",
                    ui.input_slider(id='hour_range',
                    label='Hours of the Day',
                    min=0,
                    max=23,
                    value=[6, 9],
                    step=1),
```

```
                    output_widget('scatter_plot_range'),
                    ui.output_table("filtered_table_range"),
    )
)


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

    @reactive.calc
    def filtered_data_range():
        # Filter data based on user input
        df = full_data()
        selected_combination = input.type_subtype()
        hour_range = input.hour_range()
        start_hour, end_hour = hour_range[0], hour_range[1]

        # Filter data based on the selected hour range

        # Apply filters for type_subtype and hour
        filtered_df = df[df["type_subtype"] == selected_combination]
        filtered_df = filtered_df[(filtered_df['hour'] >= start_hour) &
        (filtered_df['hour'] <= end_hour)]

        ultra_filtered_df = filtered_df.groupby(
            ['type_subtype', 'longitude', 'latitude'],
            as_index=False)['Count'].sum()

        ultra_filtered_df = ultra_filtered_df.nlargest(10, 'Count')

        return ultra_filtered_df

    @reactive.calc
    def filtered_data_single():
        # Filter data based on user input
        df = full_data()
        selected_combination = input.type_subtype()
        selected_hour = input.hour()

        # Apply filters for type_subtype and hour
```

```
    filtered_df = df[(df["type_subtype"] == selected_combination) &
    (df['hour'] == selected_hour)]
    return filtered_df

@render.table()
def filtered_table_range():
    return filtered_data_range()

@render.table()
def filtered_table_single():
    return filtered_data_single()

@render_altair
def scatter_plot_range():
    # Read and convert GeoDataFrame to the appropriate format for Altair
    geo_data =
    gpd.read_file("./top_alerts_map/chicago-boundaries.geojson")

    # Confirm longitude and latitude are numeric
    filtered = filtered_data_range()
    filtered['longitude'] = pd.to_numeric(filtered['longitude'])
    filtered['latitude'] = pd.to_numeric(filtered['latitude'])

    # Create base map chart
    map_chart = alt.Chart(geo_data).mark_geoshape(
        fill=None,  # Transparent fill
        stroke="lightgray"
    ).properties(
        width=350,
        height=350
    ).project(
        type='identity',  # Assume coordinates are in WGS84
        reflectY=True
    )

    scatter_chart = alt.Chart(filtered).mark_circle().encode(
        longitude='longitude:Q',
        latitude='latitude:Q',
        size=alt.Size(
            'Count:Q',
            title='Number of Observations',
            scale=alt.Scale(type='pow', exponent=2, range=[1, 200]))
    )
```

```
    combined_chart = (map_chart + scatter_chart).configure_view(
        stroke=None  # Remove the border of the map chart
    ).properties(
        title=f"Chicago Neighborhood Map with {input.type_subtype()}
        Observations from {input.hour_range()} O'Clock"
    ).configure_mark(
        opacity=0.7,
        color='darkblue'
    )

    return combined_chart

@render_altair
def scatter_plot_single():
    # Read and convert GeoDataFrame to the appropriate format for Altair
    geo_data =
    gpd.read_file("./top_alerts_map/chicago-boundaries.geojson")

    # Confirm longitude and latitude are numeric
    filtered = filtered_data_single()
    filtered['longitude'] = pd.to_numeric(filtered['longitude'])
    filtered['latitude'] = pd.to_numeric(filtered['latitude'])

    # Create base map chart
    map_chart = alt.Chart(geo_data).mark_geoshape(
        fill=None,  # Transparent fill
        stroke="lightgray"
    ).properties(
        width=350,
        height=350
    ).project(
        type='identity',  # Assume coordinates are in WGS84
        reflectY=True
    )

    scatter_chart = alt.Chart(filtered).mark_circle().encode(
        longitude='longitude:Q',
        latitude='latitude:Q',
        size=alt.Size(
            'Count:Q',
            title='Number of Observations',
```

```
                scale=alt.Scale(type='pow', exponent=2, range=[1, 200]))
        )


        combined_chart = (map_chart + scatter_chart).configure_view(
            stroke=None  # Remove the border of the map chart
        ).properties(
            title=f"Chicago Neighborhood Map with {input.type_subtype()}
            Observations at {input.hour()} O'Clock"
        ).configure_mark(
            opacity=0.7,
            color='darkblue'
        )

        return combined_chart

app = App(app_ui, server)



```

Background

Data Download and Exploration (20 points)

1.

```
waze_sample = pd.read_csv('waze_data_sample.csv')
print(waze_sample.columns)
```

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

Variable	Data Type
city	Nominal
confidence	Quantitative
nThumpsUp	Quantitative

Variable	Data Type
street	Nominal
uuid	Nominal
country	Nominal
type	Nominal
subtype	Nominal
roadType	Nominal
reliability	Ordinal
magvar	Quantitative
reportRating	Ordonal

Double checked with ChatGPT

2.

```
# Read in data
waze = pd.read_csv('waze_data.csv')
```

```
# create new dataframe that has the number of null and non-null values for
 ↪  each variable
waze_nulls = pd.DataFrame({
    'Column Name': waze.columns,
    'Null Values': waze.isnull().sum(),
    'Non-Null Values': waze.notnull().sum(),
    'Total Values': len(waze)
})

# make data long to create a nominal column for the stacked bar chart
waze_nulls_long = waze_nulls.melt(
    id_vars=['Column Name', 'Total Values'],
    value_vars=['Null Values', 'Non-Null Values'],
    var_name='Null/Non-Null',
    value_name='Count'
)

# create chart
alt.Chart(waze_nulls_long).mark_bar().encode(
  x='Column Name:N',
  y='Count:Q',
  color='Null/Non-Null:N'
)
```

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

```
# print the columns that have null values
variables_with_nulls = waze_nulls[waze_nulls['Null Values'] > 0]['Column
 ↪  Name']
print("Variables with NULL values:", variables_with_nulls.tolist())

# sort variables to put the variable with the most null values up top
waze_nulls_max_values = waze_nulls['Null Values'
    ].sort_values(ascending=False).reset_index()
waze_nulls_max_values = waze_nulls_max_values.rename(columns={'index':
 ↪  'Column Name'})

# print the variable name at the top of the sorted dataframe
print("Variable with the highest share of NULL values:",
 ↪  waze_nulls_max_values['Column Name'].loc[0])
```

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

3.

```
# unique values in the 'type' column
waze['type'].unique()

# unique values in the 'subtype' column
waze['subtype'].unique()

# how many subtypes are missing for each unique type
missing_subtype_counts = waze[waze['subtype'].isna()].groupby('type').size()
print(missing_subtype_counts)

# see how many of each subtype there are
relevant_subtypes = waze.groupby('subtype').size().reset_index()
```

```
type
ACCIDENT       24359
HAZARD          3212
JAM            55041
ROAD_CLOSED    13474
dtype: int64
```

Even though some subtypes only appear a few times, I still think they should be included in the hierarchy. It makes more sense to include all the options available and let users decide if there are too few values to look at any charts.

Hierarchy:

- ACCIDENT
  - Major
  - Minor
- HAZARD
  - On Road
    - Car Stopped
    - Construction
    - Emergency Vehicle
    - Ice
    - Object
    - Pot Hole
    - Traffic Light Fault
    - Lane Closed
    - Road Kill
  - On Shoulder
    - Animals
    - Car Stopped
    - Missing Sign
    - Weather
      - Flood
      - Fog
      - Heavy Snow
      - Hail
- JAM
  - Light Traffic
  - Moderate Traffic
  - Heavy Traffic
  - Stand-Still Traffic
- ROAD CLOSED
  - Event Construction

– Hazard

```
# fill NAs with 'Unclassified'
waze['subtype'] = waze['subtype'].fillna('Unclassified')
```

4.

```
# add type and subtype to cross walk
crosswalk = waze[['type', 'subtype']].drop_duplicates()

# update formatting for type
crosswalk['updated_type'] = crosswalk['type'].str.capitalize()
crosswalk.loc[crosswalk['type'] == 'ROAD_CLOSED', 'updated_type'] = 'Road
 ↪  Closed'

# entered unique subtypes to ChatGPT and asked it to create a hierarchy map
subtype_clean_mapping = {
    'ACCIDENT_MAJOR': 'Major',
    'ACCIDENT_MINOR': 'Minor',
    'HAZARD_ON_ROAD': 'On Road',
    'HAZARD_ON_ROAD_CAR_STOPPED': 'On Road',
    'HAZARD_ON_ROAD_CONSTRUCTION': 'On Road',
    'HAZARD_ON_ROAD_EMERGENCY_VEHICLE': 'On Road',
    'HAZARD_ON_ROAD_ICE': 'On Road',
    'HAZARD_ON_ROAD_OBJECT': 'On Road',
    'HAZARD_ON_ROAD_POT_HOLE': 'On Road',
    'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT': 'On Road',
    'HAZARD_ON_ROAD_LANE_CLOSED': 'On Road',
    'HAZARD_ON_ROAD_ROAD_KILL': 'On Road',
    'HAZARD_ON_SHOULDER': 'On Shoulder',
    'HAZARD_ON_SHOULDER_CAR_STOPPED': 'On Shoulder',
    'HAZARD_ON_SHOULDER_ANIMALS': 'On Shoulder',
    'HAZARD_ON_SHOULDER_MISSING_SIGN': 'On Shoulder',
    'HAZARD_WEATHER': 'Weather',
    'HAZARD_WEATHER_FLOOD': 'Weather',
    'HAZARD_WEATHER_FOG': 'Weather',
    'HAZARD_WEATHER_HEAVY_SNOW': 'Weather',
    'HAZARD_WEATHER_HAIL': 'Weather',
    'JAM_LIGHT_TRAFFIC': 'Light Traffic',
    'JAM_MODERATE_TRAFFIC': 'Moderate Traffic',
    'JAM_HEAVY_TRAFFIC': 'Heavy Traffic',
    'JAM_STAND_STILL_TRAFFIC': 'Stand-Still Traffic',
```

```
    'ROAD_CLOSED_EVENT': 'Event',
    'ROAD_CLOSED_CONSTRUCTION': 'Construction',
    'ROAD_CLOSED_HAZARD': 'Hazard',
}

# update subtype column
crosswalk['updated_subtype'] = waze['subtype'].replace(subtype_clean_mapping)

# create subsubtype column with none, to be updated later if necessary
crosswalk['updated_subsubtype'] = None

# update subsubtype for the required rows
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_ON_ROAD',
    'updated_subsubtype'] = 'General'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_ON_ROAD_CAR_STOPPED',
    'updated_subsubtype'] = 'Car Stopped'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_ON_ROAD_CONSTRUCTION',
    'updated_subsubtype'] = 'Construction'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_ON_ROAD_EMERGENCY_VEHICLE',
    'updated_subsubtype'] = 'Emergency Vehicle'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_ON_ROAD_ICE',
    'updated_subsubtype'] = 'Ice'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_ON_ROAD_OBJECT',
    'updated_subsubtype'] = 'Object'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_ON_ROAD_POT_HOLE',
    'updated_subsubtype'] = 'Pot Hole'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT',
    'updated_subsubtype'] = 'Traffic Light Fault'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_ON_ROAD_LANE_CLOSED',
    'updated_subsubtype'] = 'Lane Closed'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_ON_ROAD_ROAD_KILL',
    'updated_subsubtype'] = 'Road Kill'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_ON_SHOULDER',
    'updated_subsubtype'] = 'General'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_ON_SHOULDER_CAR_STOPPED',
    'updated_subsubtype'] = 'Car Stopped'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_ON_SHOULDER_ANIMALS',
    'updated_subsubtype'] = 'Animals'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_ON_SHOULDER_MISSING_SIGN',
    'updated_subsubtype'] = 'Missing Sign'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_WEATHER',
    'updated_subsubtype'] = 'General'
```

```
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_WEATHER_FLOOD',
    'updated_subsubtype'] = 'Flood'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_WEATHER_FOG',
    'updated_subsubtype'] = 'Fog'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_WEATHER_HEAVY_SNOW',
    'updated_subsubtype'] = 'Heavy Snow'
crosswalk.loc[crosswalk['subtype'] == 'HAZARD_WEATHER_HAIL',
    'updated_subsubtype'] = 'Hail'
crosswalk['updated_subsubtype'] =
 ↪  crosswalk['updated_subsubtype'].fillna('Unclassified')

# merge crosswalk and dataframe
waze_merged = waze.merge(crosswalk, on=['type', 'subtype'], how='left')

# find how many rows there are where type = Accident and subtype =
 ↪  Unclassified
accident_unclassified_count = waze_merged[
    (waze_merged['updated_type'] == 'Accident') &
    (waze_merged['updated_subtype'] == 'Unclassified')
].shape[0]
print(f"\nNumber of rows for Accident - Unclassified:
 ↪  {accident_unclassified_count}")
```

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

```
check_type = waze_merged['type'].isin(crosswalk['type']).all()
check_subtype = waze_merged['subtype'].isin(crosswalk['subtype']).all()

if check_type and check_subtype:
    print("The crosswalk and merged dataset have consistent values in 'type'
     ↪  and 'subtype'.")
else:
    print("There is a mismatch between the crosswalk and merged dataset in
     ↪  'type' or 'subtype'.")
```

```
The crosswalk and merged dataset have consistent values in 'type' and
'subtype'.
```

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

1.

a.

```
import re
# define regex code
pattern = r"POINT\(([-+]?[0-9]*\.?[0-9]+) ([-+]?[0-9]*\.?[0-9]+)\)"

# apply regex to geo
waze_merged[['latitude', 'longitude']] =
 ↪  waze_merged['geo'].str.extract(pattern).astype(float)
```

ChatGPT Prompt: how to write regex code to extract longitude and latitude from a wkt string in a column in a dataframe

ChatGPT Response:

To extract longitude and latitude from a Well-Known Text (WKT) string in a column of a Pandas DataFrame, we can use **regular expressions** along with the `pandas` library. Here's how to do it:

Step-by-Step Plan:

1. Import the necessary libraries (`pandas` and `re`).
2. Define a regex pattern to extract the latitude and longitude values from the WKT string.
3. Use `pandas.DataFrame.apply()` or `pandas.Series.str.extract()` to apply the regex extraction on the column containing WKT strings.
4. Store the extracted latitude and longitude values in separate DataFrame columns.

Here's the Code:

```python
import pandas as pd
import re

# Example DataFrame with WKT strings in a column
data = {
    "geo": [
        "POINT (30.2672 -97.7431)",
        "POINT (40.7128 -74.0060)",
        "POINT (34.0522 -118.2437)"
    ]
}

df = pd.DataFrame(data)

# Define the regex pattern to extract latitude and longitude
pattern = r"POINT \(([-+]?[0-9]*\.?[0-9]+) ([-+]?[0-9]*\.?[0-9]+)\)"

# Extract latitude and longitude using .str.extract()
df[['latitude', 'longitude']] = df['geo'].str.extract(pattern).astype(float)

# Display the updated DataFrame
print(df)
```

```
                        geo  latitude  longitude
0   POINT (30.2672 -97.7431)   30.2672   -97.7431
1   POINT (40.7128 -74.0060)   40.7128   -74.0060
2  POINT (34.0522 -118.2437)   34.0522  -118.2437
```

Output:

The resulting DataFrame will have two new columns for `latitude` and `longitude`:

geo	latitude	longitude
POINT (30.2672 -97.7431)	30.2672	-97.7431
POINT (40.7128 -74.0060)	40.7128	-74.0060
POINT (34.0522 -118.2437)	34.0522	-118.2437

Explanation:

1. **Regex Pattern**:
   - `POINT \(`: Matches the literal text `POINT (`.
   - `([-+]?[0-9]*\.?[0-9]+)`: Matches a signed or unsigned floating-point number (latitude).
   - A space separates latitude and longitude.
   - `([-+]?[0-9]*\.?[0-9]+)`: Matches a signed or unsigned floating-point number (longitude).
2. **`str.extract()`**:
   - Extracts groups from the matched regex and creates separate columns for each group.
3. **`.astype(float)`**:
   - Converts the extracted strings into floating-point numbers.

Suggestions:

a. Add error handling for invalid WKT strings in the DataFrame.
b. Use a unit test to validate the regex extraction for various WKT formats.

b.

```
# bin longitude
waze_merged['binned_longitude'] = waze_merged['longitude'].round(2)

# bin latitude
waze_merged['binned_latitude'] = waze_merged['latitude'].round(2)

# ChatGPT Question: how to combine two columns
waze_merged['binned_coordinates'] = list(zip(waze_merged['binned_latitude'],
 ↪  waze_merged['binned_longitude']))

# find greatest number of observations
highest_observed_bin =
 ↪  waze_merged.groupby('binned_coordinates').size().reset_index(name='Count')
```

```
highest_observed_bin = highest_observed_bin.sort_values('Count',
 ↪  ascending=False)
print('The binned latitude-longitude combination')
print('that has the greatest number of observations')
print('in the overall dataset is',
 ↪  highest_observed_bin['binned_coordinates'].loc[0])
```

```
The binned latitude-longitude combination
that has the greatest number of observations
in the overall dataset is (-87.94, 41.99)
```

c.

```
# create new dataframe based on the coordinates, type, and subtype
top_alerts_map = waze_merged.groupby(['binned_coordinates', 'updated_type',
 ↪  'updated_subtype']).size().reset_index(name='Count')

# ChatGPT Questions: i have a dataframe with type, subtype, binned
 ↪  coordinates, and count of observations for each combination of the three,
 ↪  i want to create a new dataset that subsets the top 10 highest numbers in
 ↪  count for each combination for type and subtype


# group by each type and subtype, sort by top 10 counts
top_10_alerts = top_alerts_map.sort_values('Count',
 ↪  ascending=False).groupby(['updated_type', 'updated_subtype'],
 ↪  as_index=False).head(10)

# sort by type and subtype
top_10_alerts = top_10_alerts.sort_values(by=['updated_type',
 ↪  'updated_subtype'], ascending=[True, True])

# save all the type and subtype top 10s
folder_path = 'top_alerts_map'
os.makedirs(folder_path, exist_ok=True)
file_path = os.path.join(folder_path, 'top_10_alerts.csv')
top_10_alerts.to_csv(file_path, index=False)

# choose one type and subtype
top_10_jam = top_10_alerts[top_10_alerts['updated_subtype']=='Heavy Traffic']
```

```
folder_path = 'top_alerts_map'
os.makedirs(folder_path, exist_ok=True)
file_path = os.path.join(folder_path, 'top_alerts_map.csv')
top_10_jam.to_csv(file_path, index=False)
```

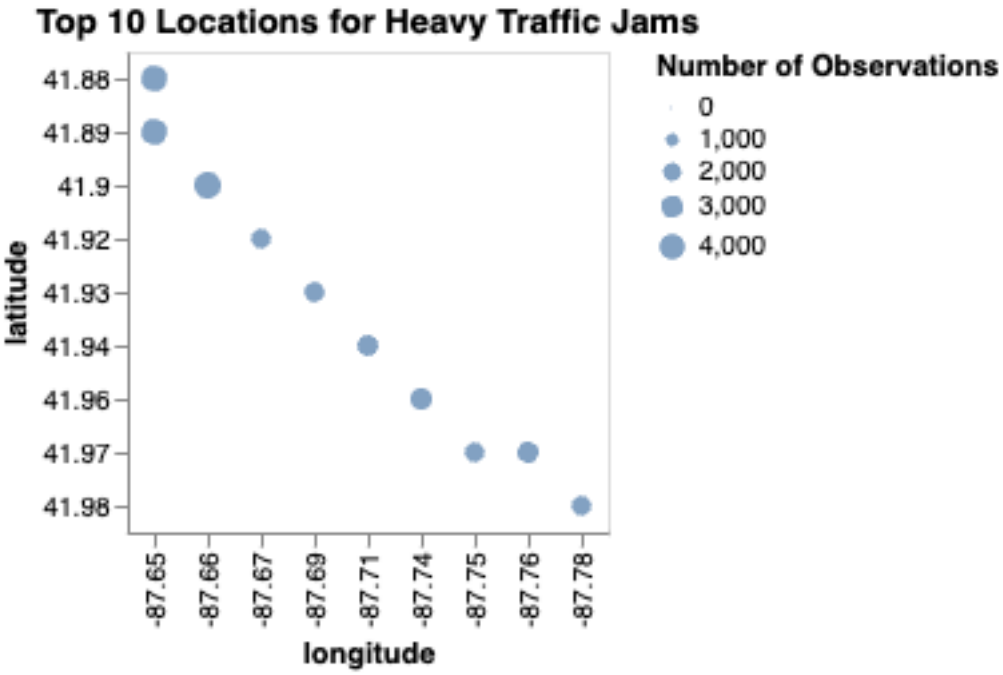
Each row represents a unique combination of type, subtype, and binned_coordinates. I then paired this down to just be one type and subtype (Jam, Heavy Traffic). There are 10 rows in this dataframe because it is the top 10 most observed coordinates for Heavy Traffic. There were 155 rows when each combination of type and subtype had at most 10 coordinates.

2.

```
top_10_jam['binned_coordinates'] =
 ↪  top_10_jam['binned_coordinates'].astype(str)
top_10_jam[['longitude', 'latitude']] =
 ↪  top_10_jam['binned_coordinates'].str.strip('()').str.split(', ',
 ↪  expand=True)
```

```
heavy_traffic_chart = alt.Chart(top_10_jam, title='Top 10 Locations for Heavy
 ↪  Traffic Jams').mark_circle().encode(
    alt.X('longitude'),
    alt.Y('latitude'),
    alt.Size('Count:Q', scale=alt.Scale(range=[1, 100]), title='Number of
 ↪  Observations')
)

heavy_traffic_chart
```

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



3.

a.

```
import requests

url =
 ↪  'https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON'
geo_data = requests.get(url)
```

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"])

print("Top-level keys in GeoJSON:", chicago_geojson.keys())

# Number of features
print("Number of features:", len(chicago_geojson['features']))
```

```
Top-level keys in GeoJSON: dict_keys(['type', 'features'])
Number of features: 98
```

4.

```
map_chart = alt.Chart(geo_data).mark_geoshape(
    fill='None',  # Change to "None" or transparent for layering
    stroke='lightgray'
).encode(
    tooltip=[alt.Tooltip('properties.neighborhood:N', title='Neighborhood')]
 ↪  # Adjust as per GeoJSON properties
).properties(
    width=400,
    height=400
).project(
    type='identity',  # Assume coordinates are in WGS84
    reflectY=True     # Adjust Y-axis to match standard map orientation
)
scatter_chart = alt.Chart(top_10_jam, title='Top 10 Locations for Heavy
 ↪  Traffic Jams').mark_circle().encode(
    longitude='longitude:Q',
    latitude='latitude:Q',
    size=alt.Size(
        'Count:Q',
        title='Number of Observations',
        scale=alt.Scale(type='pow', exponent=2, range=[1, 200]))
)


combined_chart = (map_chart + scatter_chart).configure_view(
    stroke=None  # Remove the border of the map chart
).properties(
    title="Chicago Neighborhood Map with Heavy Traffic Jam Observations"
).configure_mark(
    opacity=0.7,
    color='darkblue'
)

combined_chart
```

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

5.

a.

```
top_10_alerts["type_subtype"] = top_10_alerts["updated_type"] + " - " +
 ↪  top_10_alerts["updated_subtype"]
print(top_10_alerts['type_subtype'].nunique())
```

16

```
from shiny import App, render, ui, reactive
import pandas as pd
import anywidget
from shinywidgets import render_altair, output_widget
import altair as alt
import json
import geopandas as gpd

app_ui = ui.page_fluid(
    ui.input_select(id = 'type_subtype', label = 'Select a Type-Subtype Combination:',
    choices = ["Accident - Major", "Accident - Minor", "Accident - Unclassified",
        "Jam - Heavy Traffic", "Jam - Light Traffic", "Jam - Moderate Traffic",
        "Jam - Stand-Still Traffic", "Jam - Unclassified", "Hazard - On Road",
        "Hazard - On Shoulder", "Hazard - Weather", "Hazard - Unclassified",
        "Road Closed - Construction", "Road Closed - Event", "Road Closed - Hazard",
        "Road Closed - Unclassified"]),
    output_widget('scatter_plot'),
    ui.output_table("filtered_table"),
)


def server(input, output, session):
    @reactive.calc
    def full_data():
        return pd.read_csv("top_alerts_map/top_10_alerts.csv")


    @reactive.calc
    def filtered_data():
        df = full_data()
        selected_combination = input.type_subtype()
        return df[df["type_subtype"] == selected_combination]
```

app.py code for the drop-down menu in app #1

Enter this into terminal to run shiny app: shiny run top_alerts_map/app.py

b.

```
top_10_alerts['binned_coordinates'] =
 ↪  top_10_alerts['binned_coordinates'].astype(str)
top_10_alerts[['longitude', 'latitude']] =
 ↪  top_10_alerts['binned_coordinates'].str.strip('()').str.split(', ',
 ↪  expand=True)
top_10_alerts[['longitude', 'latitude']] = top_10_alerts[['longitude',
 ↪  'latitude']].astype(float)
```

```
# save all the type and subtype top 10s
folder_path = 'top_alerts_map'
os.makedirs(folder_path, exist_ok=True)
file_path = os.path.join(folder_path, 'top_10_alerts.csv')
top_10_alerts.to_csv(file_path, index=False)
```

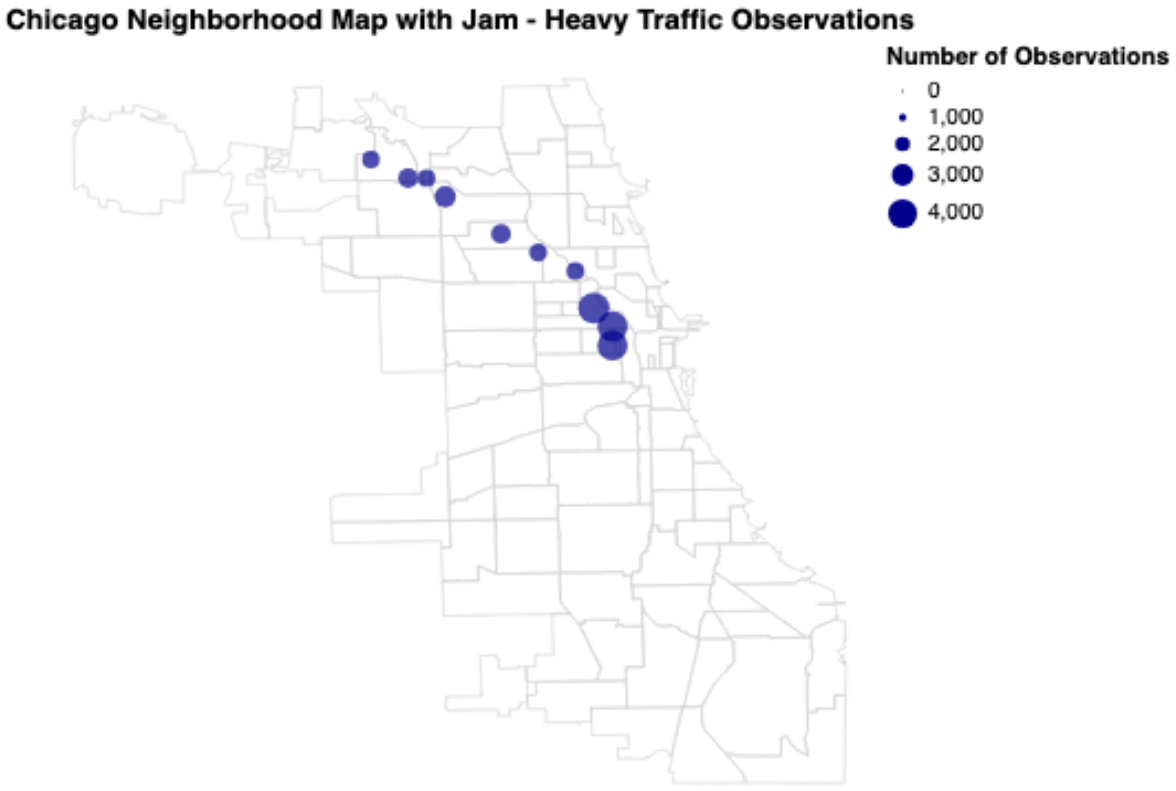
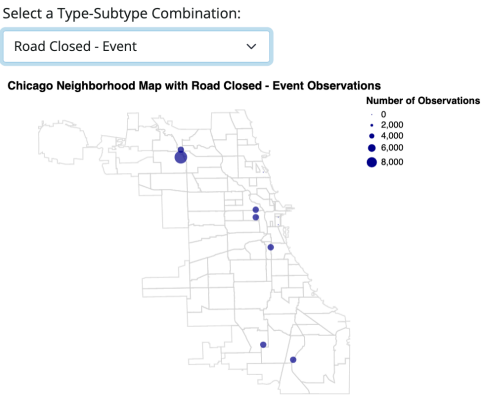


Figure 1: Jam-Heavy Traffic Plot from shiny app #1

c.



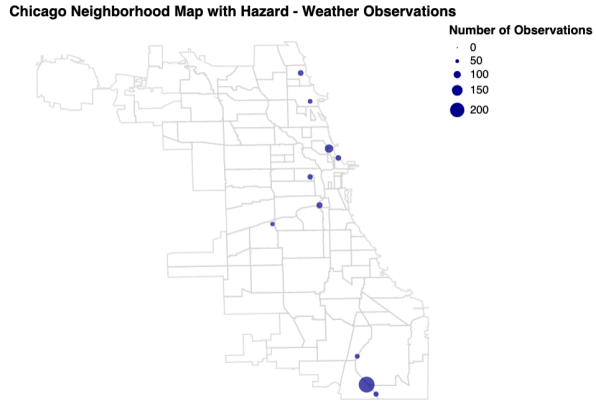
binned_coordinates	updated_type	updated_subtype	Count	longitude	latitude	type_subtype
(-87.75, 41.96)	Road Closed	Event	9907	-87.75	41.96	Road Closed - Event
(-87.65, 41.88)	Road Closed	Event	5002	-87.65	41.88	Road Closed - Event
(-87.63, 41.84)	Road Closed	Event	4974	-87.63	41.84	Road Closed - Event
(-87.65, 41.89)	Road Closed	Event	4964	-87.65	41.89	Road Closed - Event
(-87.6, 41.69)	Road Closed	Event	4964	-87.60	41.69	Road Closed - Event
(-87.64, 41.71)	Road Closed	Event	4947	-87.64	41.71	Road Closed - Event
(-87.75, 41.97)	Road Closed	Event	4932	-87.75	41.97	Road Closed - Event
(-87.62, 41.88)	Road Closed	Event	291	-87.62	41.88	Road Closed - Event
(-87.62, 41.87)	Road Closed	Event	233	-87.62	41.87	Road Closed - Event
(-87.64, 41.94)	Road Closed	Event	116	-87.64	41.94	Road Closed - Event

The largest dot and first row in the DataFrame indicate where the most road-closed due to events alerts happen.

d. Where is weather most likely to cause a driving hazard in Chicago?

Select a Type-Subtype Combination:

Hazard - Weather



binned_coordinates	updated_type	updated_subtype	Count	longitude	latitude	type_subtype
(-87.59, 41.66)	Hazard	Weather	221	-87.59	41.66	Hazard - Weather
(-87.63, 41.91)	Hazard	Weather	117	-87.63	41.91	Hazard - Weather
(-87.64, 41.85)	Hazard	Weather	83	-87.64	41.85	Hazard - Weather
(-87.66, 41.99)	Hazard	Weather	79	-87.66	41.99	Hazard - Weather
(-87.62, 41.9)	Hazard	Weather	77	-87.62	41.90	Hazard - Weather
(-87.65, 41.88)	Hazard	Weather	73	-87.65	41.88	Hazard - Weather
(-87.58, 41.65)	Hazard	Weather	71	-87.58	41.65	Hazard - Weather
(-87.6, 41.69)	Hazard	Weather	67	-87.60	41.69	Hazard - Weather
(-87.65, 41.96)	Hazard	Weather	64	-87.65	41.96	Hazard - Weather
(-87.69, 41.83)	Hazard	Weather	59	-87.69	41.83	Hazard - Weather

Weather is most likely to cause a driving hazard in South Chicago.

e. Providing the most common road type for each binned coordinates might help identify more details about the accidents rather than just where they were located.

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

1.

a. It would not be a good idea to collapse by this column because each row is a unique value, so there is nothing to collapse.

b.

```
# convert ts to datetime
waze_merged['ts'] = pd.to_datetime(waze_merged['ts'])

# filter out hour
waze_merged['hour'] = waze_merged['ts'].dt.hour

# collapse data by type, subtype, binned_coordinates, and hour
alerts_byhour = waze_merged.groupby(['binned_coordinates', 'updated_type',
 ↪  'updated_subtype', 'hour']).size().reset_index(name='Count')

# count for each type, subtype, and hour combo
alerts_by_hour = alerts_byhour.sort_values('Count',
 ↪  ascending=False).groupby(['updated_type', 'updated_subtype', 'hour'],
 ↪  as_index=False).head(10)

# sort values
alerts_by_hour = alerts_by_hour.sort_values(by=['updated_type',
 ↪  'updated_subtype', 'hour'], ascending=[True, True, True])

# create type-subtype column
alerts_by_hour["type_subtype"] = alerts_by_hour["updated_type"] + " - " +
 ↪  alerts_by_hour["updated_subtype"]

# create longitude and latitude columns
alerts_by_hour['binned_coordinates'] =
 ↪  alerts_by_hour['binned_coordinates'].astype(str)
alerts_by_hour[['longitude', 'latitude']] =
 ↪  alerts_by_hour['binned_coordinates'].str.strip('()').str.split(', ',
 ↪  expand=True)

# save hour grouped dataframe
folder_path = 'top_alerts_map_byhour'
os.makedirs(folder_path, exist_ok=True)
file_path = os.path.join(folder_path, 'top_alerts_map_byhour.csv')
alerts_by_hour.to_csv(file_path, index=False)
```

c.

```
# filter for heavy traffic jam
top_10_jam_byhour = alerts_by_hour[alerts_by_hour['type_subtype']=='Jam -
 ↪  Heavy Traffic']
```

```
# filter for 9 am
top_10_jam_9 = top_10_jam_byhour[top_10_jam_byhour['hour']==9]

# filter for 12 pm
top_10_jam_12 = top_10_jam_byhour[top_10_jam_byhour['hour']==12]

# filter for 5 pm
top_10_jam_17 = top_10_jam_byhour[top_10_jam_byhour['hour']==17]
```

9 A.M. Heavy Traffic Jam Chart

```
scatter_9_chart = alt.Chart(top_10_jam_9).mark_circle().encode(
    longitude='longitude:Q',
    latitude='latitude:Q',
    size=alt.Size(
        'Count:Q',
        title='Number of Observations',
        scale=alt.Scale(type='pow', exponent=2, range=[1, 200]))
)


combined_9_chart = (map_chart + scatter_9_chart).configure_view(
    stroke=None  # Remove the border of the map chart
).properties(
    title='Top 10 Locations for Heavy Traffic Jam Observations at 9 A.M.'
).configure_mark(
    opacity=0.7,
    color='darkblue'
)

combined_9_chart
```

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

12 P.M. Heavy Traffic Jam Chart

```
scatter_12_chart = alt.Chart(top_10_jam_12).mark_circle().encode(
    longitude='longitude:Q',
    latitude='latitude:Q',
    size=alt.Size(
        'Count:Q',
        title='Number of Observations',
        scale=alt.Scale(type='pow', exponent=2, range=[1, 200]))
)


combined_12_chart = (map_chart + scatter_12_chart).configure_view(
    stroke=None  # Remove the border of the map chart
).properties(
    title='Top 10 Locations for Heavy Traffic Jam Observations at 12 P.M.'
).configure_mark(
    opacity=0.7,
    color='darkblue'
)

combined_12_chart
```

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

5 P.M. Heavy Traffic Jam Chart

```
scatter_17_chart = alt.Chart(top_10_jam_17).mark_circle().encode(
    longitude='longitude:Q',
    latitude='latitude:Q',
    size=alt.Size(
        'Count:Q',
        title='Number of Observations',
        scale=alt.Scale(type='pow', exponent=2, range=[1, 200]))
)


combined_17_chart = (map_chart + scatter_17_chart).configure_view(
    stroke=None  # Remove the border of the map chart
).properties(
    title='Top 10 Locations for Heavy Traffic Jam Observations at 5 P.M.'
```

```
).configure_mark(
    opacity=0.7,
    color='darkblue'
)

combined_17_chart
```

`alt.LayerChart(...)`

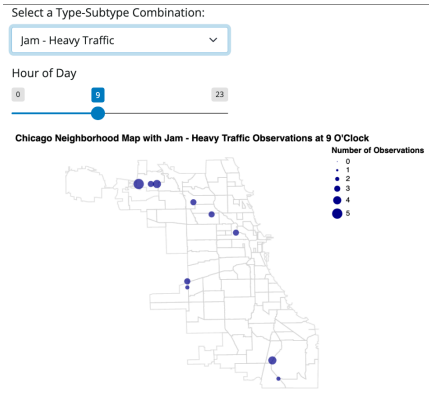
2.

Enter this into terminal to run shiny app: shiny run top_alerts_map_byhour/app.py

b. a.

```
import pandas as pd
import anywidget
from shinywidgets import render_altair, output_widget
import altair as alt
import json
import geopandas as gpd

app_ui = ui.page_fluid(
    ui.input_select(id = 'type_subtype', label = 'Select a Type-Subtype Combination:',
    choices = ["Accident - Major", "Accident - Minor", "Accident - Unclassified",
        "Jam - Heavy Traffic", "Jam - Light Traffic", "Jam - Moderate Traffic",
        "Jam - Stand-Still Traffic", "Jam - Unclassified", "Hazard - On Road",
        "Hazard - On Shoulder", "Hazard - Weather", "Hazard - Unclassified",
        "Road Closed - Construction", "Road Closed - Event", "Road Closed - Hazard",
        "Road Closed - Unclassified"]),
    ui.input_slider('hour', 'Hour of Day', 0, 23, 1),
    output_widget('scatter_plot'),
    ui.output_table("filtered_table"),
)


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


    @reactive.calc
    def filtered_data():
        df = full_data()
        selected_combination = input.type_subtype()
        selected_hour = input.hour()
        return df[[df["type_subtype"] == selected_combination],
            [df['hour'] == selected_hour]]
```

b. Heavy Traffic Jam 9 A.M.



binned_coordinates	updated_type	updated_subtype	hour	Count	type_subtype	longitude	latitude
(-87.82, 41.98)	Jam	Heavy Traffic	9	5	Jam - Heavy Traffic	-87.82	41.98
(-87.79, 41.98)	Jam	Heavy Traffic	9	4	Jam - Heavy Traffic	-87.79	41.98
(-87.6, 41.69)	Jam	Heavy Traffic	9	4	Jam - Heavy Traffic	-87.60	41.69
(-87.66, 41.9)	Jam	Heavy Traffic	9	3	Jam - Heavy Traffic	-87.66	41.90
(-87.8, 41.98)	Jam	Heavy Traffic	9	3	Jam - Heavy Traffic	-87.80	41.98
(-87.7, 41.93)	Jam	Heavy Traffic	9	3	Jam - Heavy Traffic	-87.70	41.93
(-87.73, 41.95)	Jam	Heavy Traffic	9	3	Jam - Heavy Traffic	-87.73	41.95
(-87.74, 41.82)	Jam	Heavy Traffic	9	3	Jam - Heavy Traffic	-87.74	41.82
(-87.74, 41.81)	Jam	Heavy Traffic	9	2	Jam - Heavy Traffic	-87.74	41.81
(-87.59, 41.66)	Jam	Heavy Traffic	9	2	Jam - Heavy Traffic	-87.59	41.66

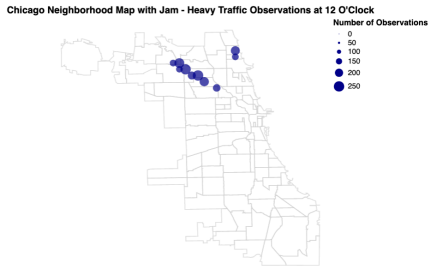
Heavy Traffic Jam 12 P.M.

Select a Type-Subtype Combination:

Jam - Heavy Traffic

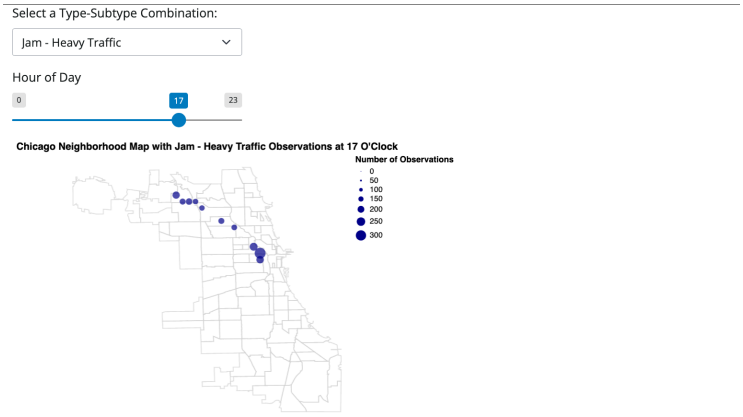
Hour of Day

0 12 23



binned_coordinates	updated_type	updated_subtype	hour	Count	type_subtype	longitude	latitude
(-87.72, 41.95)	Jam	Heavy Traffic	12	252	Jam - Heavy Traffic	-87.72	41.95
(-87.74, 41.96)	Jam	Heavy Traffic	12	248	Jam - Heavy Traffic	-87.74	41.96
(-87.75, 41.97)	Jam	Heavy Traffic	12	236	Jam - Heavy Traffic	-87.75	41.97
(-87.71, 41.94)	Jam	Heavy Traffic	12	225	Jam - Heavy Traffic	-87.71	41.94
(-87.66, 41.99)	Jam	Heavy Traffic	12	218	Jam - Heavy Traffic	-87.66	41.99
(-87.73, 41.95)	Jam	Heavy Traffic	12	195	Jam - Heavy Traffic	-87.73	41.95
(-87.69, 41.93)	Jam	Heavy Traffic	12	182	Jam - Heavy Traffic	-87.69	41.93
(-87.76, 41.97)	Jam	Heavy Traffic	12	164	Jam - Heavy Traffic	-87.76	41.97
(-87.66, 41.98)	Jam	Heavy Traffic	12	163	Jam - Heavy Traffic	-87.66	41.98
(-87.75, 41.96)	Jam	Heavy Traffic	12	158	Jam - Heavy Traffic	-87.75	41.96

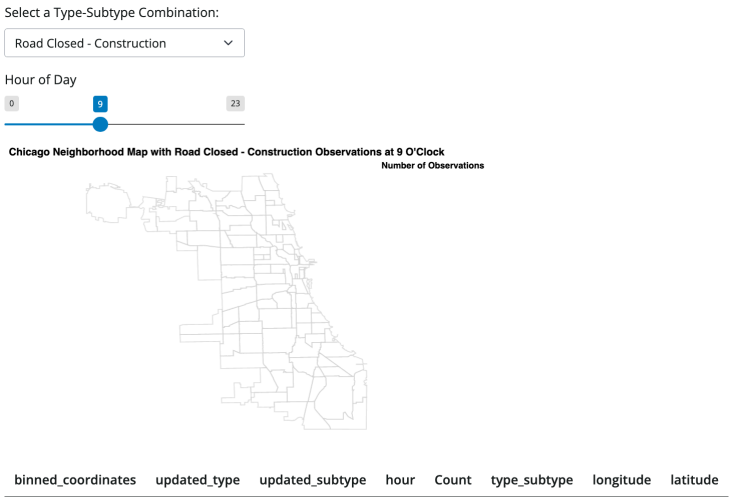
Heavy Traffic Jam 5 P.M.



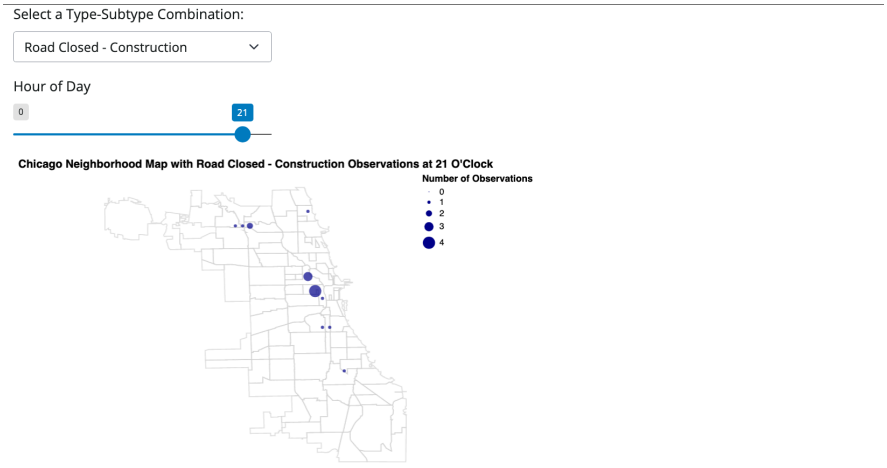
binned_coordinates	updated_type	updated_subtype	hour	Count	type_subtype	longitude	latitude
(-87.65, 41.89)	Jam	Heavy Traffic	17	318	Jam - Heavy Traffic	-87.65	41.89
(-87.66, 41.9)	Jam	Heavy Traffic	17	233	Jam - Heavy Traffic	-87.66	41.90
(-87.65, 41.88)	Jam	Heavy Traffic	17	220	Jam - Heavy Traffic	-87.65	41.88
(-87.78, 41.98)	Jam	Heavy Traffic	17	216	Jam - Heavy Traffic	-87.78	41.98
(-87.76, 41.97)	Jam	Heavy Traffic	17	190	Jam - Heavy Traffic	-87.76	41.97
(-87.77, 41.97)	Jam	Heavy Traffic	17	176	Jam - Heavy Traffic	-87.77	41.97
(-87.71, 41.94)	Jam	Heavy Traffic	17	175	Jam - Heavy Traffic	-87.71	41.94
(-87.69, 41.93)	Jam	Heavy Traffic	17	167	Jam - Heavy Traffic	-87.69	41.93
(-87.75, 41.97)	Jam	Heavy Traffic	17	160	Jam - Heavy Traffic	-87.75	41.97
(-87.74, 41.96)	Jam	Heavy Traffic	17	159	Jam - Heavy Traffic	-87.74	41.96

c. Based on the limited reports, there appears to be no road construction during the morning hours and more reports at night

   Road Construction at 9 A.M.



Road Construction 9 P.M.



binned_coordinates	updated_type	updated_subtype	hour	Count	type_subtype	longitude	latitude
(-87.65, 41.88)	Road Closed	Construction	21	4	Road Closed - Construction	-87.65	41.88
(-87.66, 41.9)	Road Closed	Construction	21	3	Road Closed - Construction	-87.66	41.90
(-87.74, 41.97)	Road Closed	Construction	21	2	Road Closed - Construction	-87.74	41.97
(-87.61, 41.77)	Road Closed	Construction	21	1	Road Closed - Construction	-87.61	41.77
(-87.63, 41.83)	Road Closed	Construction	21	1	Road Closed - Construction	-87.63	41.83
(-87.75, 41.97)	Road Closed	Construction	21	1	Road Closed - Construction	-87.75	41.97
(-87.76, 41.97)	Road Closed	Construction	21	1	Road Closed - Construction	-87.76	41.97
(-87.66, 41.99)	Road Closed	Construction	21	1	Road Closed - Construction	-87.66	41.99
(-87.64, 41.83)	Road Closed	Construction	21	1	Road Closed - Construction	-87.64	41.83
(-87.64, 41.87)	Road Closed	Construction	21	1	Road Closed - Construction	-87.64	41.87

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

1.

a. I think it makes the most sense to sum the Count column based on the hours chosen rather than collapsing by a range of hours. Since the user will pick the range of hours, it does not make sense for the dataframe to be collapsed by a set range of hours.

b.

```
# subset heavy traffic jam data to between 6 ad 9 am
top_10_jam_6_9 = top_10_jam_byhour[(top_10_jam_byhour['hour']>=6) &
 ↪  (top_10_jam_byhour['hour']<= 9)]

# add all observations for each coordinate and type-subtype
```

```
top_10_jam_6_9_range = top_10_jam_6_9.groupby(['type_subtype', 'longitude',
 ↪  'latitude'], as_index=False)['Count'].sum()

top_10_jam_6_9_range = top_10_jam_6_9_range.nlargest(10, 'Count')
```

```
scatter_6_9_chart = alt.Chart(top_10_jam_6_9_range).mark_circle().encode(
    longitude='longitude:Q',
    latitude='latitude:Q',
    size=alt.Size(
        'Count:Q',
        title='Number of Observations',
        scale=alt.Scale(type='pow', exponent=2, range=[1, 200]))
)


combined_6_9_chart = (map_chart + scatter_6_9_chart).configure_view(
    stroke=None  # Remove the border of the map chart
).properties(
    title='Top 10 Locations for Heavy Traffic Jam Observations from 6-9 A.M.'
).configure_mark(
    opacity=0.7,
    color='darkblue'
)

combined_6_9_chart
```

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

2.

Enter this into terminal to run shiny app: shiny run top_alerts_map_byhour_sliderrange/app.py

a. I chose not to include all the type-subtype choices in my screenshot, but they are all still there. I wanted to only put in one screenshot, and I wanted it to include the most new and relevant code.

top_alerts_map_byhour_sliderrange > app.py

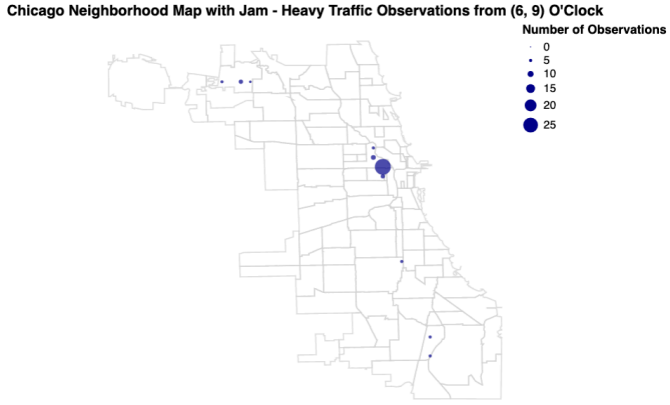
```
app_ui = ui.page_fluid(
    ui.input_select(id='type_subtype',
                    "Road Closed - Construction", "Road Closed - Event", "Road Closed - Hazard",
                    "Road Closed - Unclassified"]),
    ui.input_slider(id='hour_range',
                    label='Hour of Day',
                    min=0,
                    max=23,
                    value=[6, 9],
                    step=1),
    output_widget('scatter_plot'),
    ui.output_table("filtered_table"),
)


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


    @reactive.calc
    def filtered_data():
        # Filter data based on user input
        df = full_data()
        selected_combination = input.type_subtype()
        hour_range = input.hour_range()
        start_hour, end_hour = hour_range[0], hour_range[1]

        # Filter data based on the selected hour range

        # Apply filters for type_subtype and hour
        filtered_df = df[df["type_subtype"] == selected_combination]
        filtered_df = filtered_df[(filtered_df['hour'] >= start_hour) & (filtered_df['hour'] <= end_hour)]

        ultra_filtered_df = filtered_df.groupby(
            ['type_subtype', 'longitude', 'latitude'],
            as_index=False)['Count'].sum()

        ultra_filtered_df = ultra_filtered_df.nlargest(10, 'Count')

        return ultra_filtered_df
```

Select a Type-Subtype Combination:

Jam - Heavy Traffic

Hour of Day

0 6 9 23

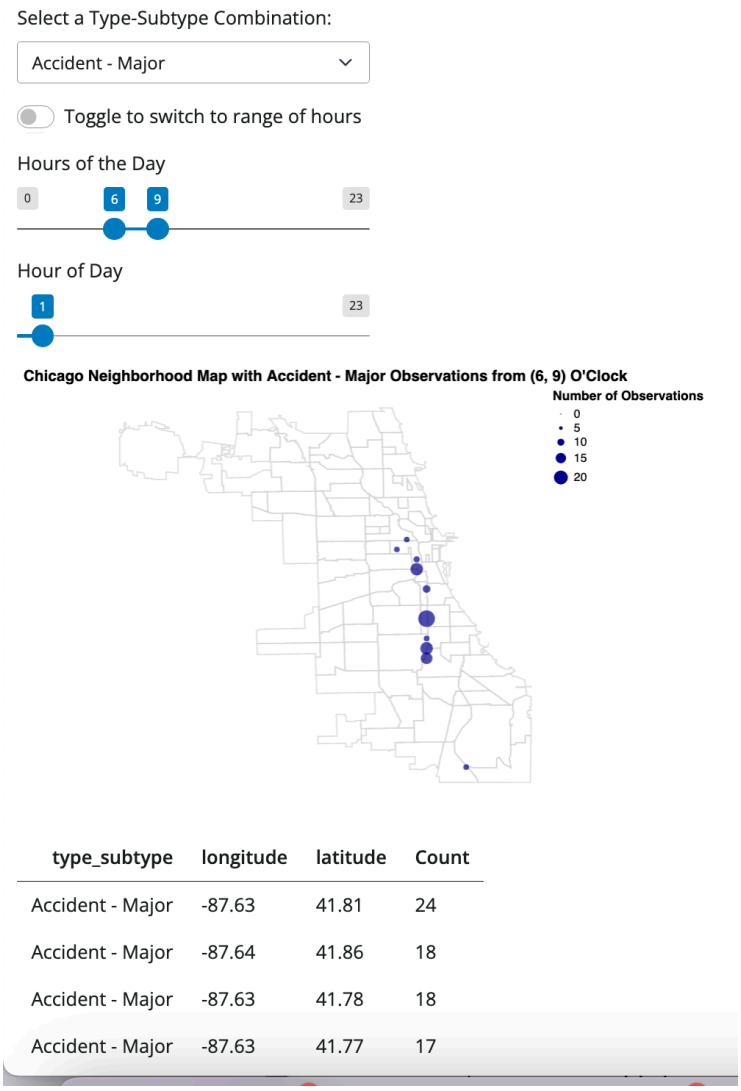


type_subtype	longitude	latitude	Count
Jam - Heavy Traffic	-87.65	41.89	27
Jam - Heavy Traffic	-87.66	41.90	8
Jam - Heavy Traffic	-87.80	41.98	7
Jam - Heavy Traffic	-87.65	41.88	7
Jam - Heavy Traffic	-87.82	41.98	5
Jam - Heavy Traffic	-87.66	41.91	5
Jam - Heavy Traffic	-87.63	41.79	5
Jam - Heavy Traffic	-87.60	41.69	5
Jam - Heavy Traffic	-87.60	41.71	5
Jam - Heavy Traffic	-87.79	41.98	4

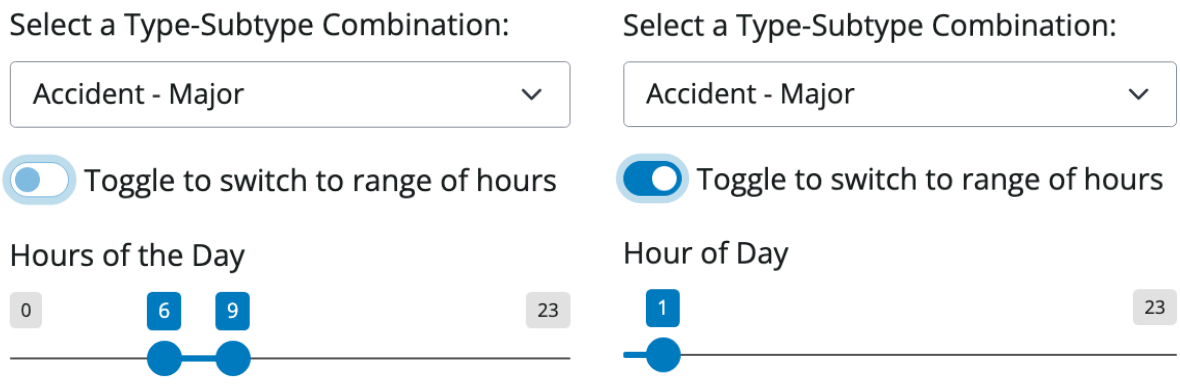
b.

3.

a. The possible values of the switch are True or False. True means that you can choose from a single hour, and False means that you choose a range of hours.


Select a Type-Subtype Combination:
Accident - Major
Toggle to switch to range of hours
Hours of the Day
0
6
9
23
Hour of Day
1
23
Chicago Neighborhood Map with Accident - Major Observations from (6, 9) O'Clock
Number of Observations
0
5
10
15
20
type_subtype longitude latitude Count
Accident - Major -87.63 41.81 24
Accident - Major -87.64 41.86 18
Accident - Major -87.63 41.78 18
Accident - Major -87.63 41.77 17

b.


Select a Type-Subtype Combination:
Accident - Major
Toggle to switch to range of hours
Hours of the Day
0
6
9
23
Select a Type-Subtype Combination:
Accident - Major
Toggle to switch to range of hours
Hour of Day
1
23

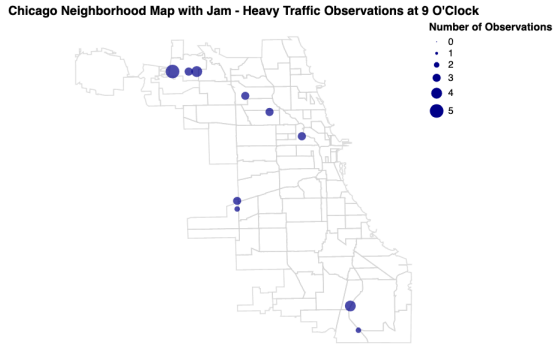
Select a Type-Subtype Combination:

Jam - Heavy Traffic

Toggle to switch to range of hours

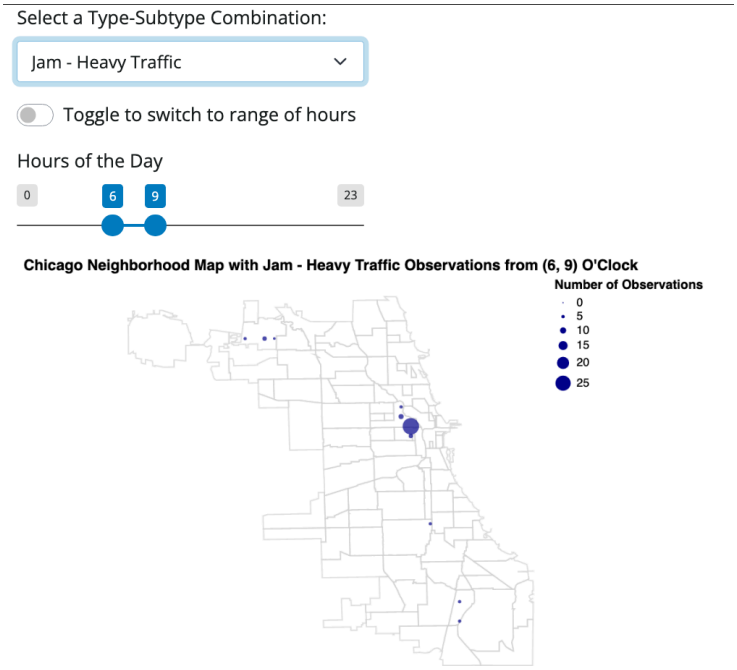
Hour of Day

0 9 23



binned_coordinates	updated_type	updated_subtype	hour	Count	type_subtype	longitude	latitude
(-87.82, 41.98)	Jam	Heavy Traffic	9	5	Jam - Heavy Traffic	-87.82	41.98
(-87.79, 41.98)	Jam	Heavy Traffic	9	4	Jam - Heavy Traffic	-87.79	41.98
(-87.6, 41.69)	Jam	Heavy Traffic	9	4	Jam - Heavy Traffic	-87.60	41.69
(-87.66, 41.9)	Jam	Heavy Traffic	9	3	Jam - Heavy Traffic	-87.66	41.90
(-87.8, 41.98)	Jam	Heavy Traffic	9	3	Jam - Heavy Traffic	-87.80	41.98
(-87.7, 41.93)	Jam	Heavy Traffic	9	3	Jam - Heavy Traffic	-87.70	41.93
(-87.73, 41.95)	Jam	Heavy Traffic	9	3	Jam - Heavy Traffic	-87.73	41.95
(-87.74, 41.82)	Jam	Heavy Traffic	9	3	Jam - Heavy Traffic	-87.74	41.82
(-87.74, 41.81)	Jam	Heavy Traffic	9	2	Jam - Heavy Traffic	-87.74	41.81
(-87.59, 41.66)	Jam	Heavy Traffic	9	2	Jam - Heavy Traffic	-87.59	41.66

c.



type_subtype	longitude	latitude	Count
Jam - Heavy Traffic	-87.65	41.89	27
Jam - Heavy Traffic	-87.66	41.90	8
Jam - Heavy Traffic	-87.80	41.98	7
Jam - Heavy Traffic	-87.65	41.88	7
Jam - Heavy Traffic	-87.82	41.98	5
Jam - Heavy Traffic	-87.66	41.91	5
Jam - Heavy Traffic	-87.63	41.79	5
Jam - Heavy Traffic	-87.60	41.69	5
Jam - Heavy Traffic	-87.60	41.71	5
Jam - Heavy Traffic	-87.79	41.98	4

d.

e. I would create a new column called time_period that labels hours 0-11 as Morning and 12-23 as Afternoon. I would then collapse the data by type-subtype, location, and the top 10 morning locations and top 10 afternoon locations. I would make the chart mark_point() and add alt.Color based on time_period.