Problem Set 6 - Waze Shiny Dashboard

AUTHOR
Zheng Cui

PUBLISHED
November 24, 2001

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: **_ZC_**
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **_ZC_** (2 point)
3. Late coins used this pset: **_1_** Late coins left after submission: **_1_**
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. Push 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 your Github repo (5 points). It is fine to use Github Desktop.
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!

```
#top_alerts_map
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("/Users/zhengcui/Desktop/python
        2/student30538/problem_sets/ps6/top_alerts_map/basic-app/app.py")
```

```python

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


#load the essential data
top_10_alert_df = pd.read_csv('/Users/zhengcui/Desktop/python
2/student30538/problem_sets/ps6/top_alerts_map/top_alerts_map.csv')

file_path = "/Users/zhengcui/Desktop/python 2/student30538/problem_sets/ps6/Boundaries -
Neighborhoods.geojson"
with open(file_path) as f:
    chicago_geojson = json.load(f)

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



top_10_alert_df['type_subtype'] = top_10_alert_df['updated_type'].str.capitalize() + ' - ' + \
                                  top_10_alert_df['updated_subtype'].str.replace('_', ' ').str.title()


options = top_10_alert_df['type_subtype'].drop_duplicates().tolist()


#prepare UI,server and diagram
app_ui = ui.page_fluid(
    ui.panel_title("Traffic Alert Dashboard"),
    ui.input_select("selected_option", "Select options", choices=options, selected=options[0]),
    output_widget("traffic_plot")
)


def server(input, output, session):
    @output
    @render_altair
    def traffic_plot():
        selected_option = input.selected_option()
        filtered_data = top_10_alert_df[top_10_alert_df['type_subtype'] == selected_option]


        top_10_data = filtered_data.nlargest(10, 'count')




        geo_map = alt.Chart(alt.Data(geo_data)).mark_geoshape(
            fill='lightgray', stroke='white'
        ).encode(
```

```
            tooltip='properties.pri_neigh:N'
        ).properties(
            width=600, height=400,
            title='Traffic Alerts Report-Chicago'
        ).project(type='mercator')


        scatter_plot = alt.Chart(top_10_data).mark_circle(
            fill=None, stroke='red', strokeWidth=2
        ).encode(
            longitude='longitude:Q',
            latitude='latitude:Q',
            size=alt.Size('count:Q', title='Number of Alerts',
                          scale=alt.Scale(domain=[top_10_data['count'].min(),
top_10_data['count'].max()]),
                          legend=alt.Legend(title="Traffic Alerts")
            ),
            color=alt.value('red'),
            tooltip=['longitude', 'latitude', 'count']
        ).properties(
            width=600, height=400
        )


        return geo_map + scatter_plot


app = App(app_ui, server)
```

```
#top_alerts_map_byhour
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("/Users/zhengcui/Desktop/python
        2/student30538/problem_sets/ps6/top_alerts_map_byhour/basic-app/app2.py")
```

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


# Loading the essential data
top_alerts_map_byhour = pd.read_csv('/Users/zhengcui/Desktop/python
2/student30538/problem_sets/ps6/top_alerts_map_byhour/top_alerts_map_byhour.csv')
top_alerts_map_byhour['type_subtype'] = top_alerts_map_byhour['updated_type'].str.capitalize() + ' - '
+ top_alerts_map_byhour['updated_subtype'].str.replace('_', ' ').str.title()
type_subtype_choices = top_alerts_map_byhour['type_subtype'].drop_duplicates().sort_values().tolist()



file_path = "/Users/zhengcui/Desktop/python 2/student30538/problem_sets/ps6/Boundaries -
Neighborhoods.geojson"
with open(file_path) as f:
    chicago_geojson = json.load(f)

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



#prepare UI,server and diagram
app_ui = ui.page_fluid(
    ui.panel_title("Traffic Alert Dashboard"),
    ui.input_select("selected_type_subtype", "Select Type-Subtype", choices=type_subtype_choices,
selected=type_subtype_choices[0]),
    ui.input_slider("selected_hour", "Select Hour", min=0, max=23, value=12, step=1),
    output_widget("traffic_map")
)


def server(input, output, session):
    @output
    @render_altair
    def traffic_map():

        selected_hour = f"{input.selected_hour():02}:00"


        essential_data = top_alerts_map_byhour[
            (top_alerts_map_byhour['type_subtype'] == input.selected_type_subtype()) &
            (top_alerts_map_byhour['hour'] == selected_hour)
        ]
        if essential_data.empty:
            return alt.Chart().mark_text(text="No data available").properties(width=600, height=400)

        top10_data = essential_data.nlargest(10, 'count')


        return diagram(top10_data)

def diagram(data):

    unique_number = sorted(data['count'].unique())

    geo_map = alt.Chart(alt.Data(geo_data)).mark_geoshape(
        fill='lightgray', stroke='white'
```

```
    ).encode(
        tooltip='properties.pri_neigh:N'
    ).properties(
        width=600, height=400,
        title='Traffic Alerts Report-Chicago'
    ).project(type='mercator')

    scatter_plot = alt.Chart(data).mark_circle(
        stroke='black', strokeWidth=1,fill='red'
    ).encode(
        longitude='longitude:Q',
        latitude='latitude:Q',
        size=alt.Size('count:Q', title='Number of Alerts',
                      scale=alt.Scale(domain=[min(unique_number), max(unique_number)], range=[20, 500],
clamp=True),
                      legend=alt.Legend(title="Number of Alerts", values=unique_number)),
        tooltip=[alt.Tooltip('longitude', title='Longitude'),
                 alt.Tooltip('latitude', title='Latitude'),
                 alt.Tooltip('count', title='Number of Alerts', format=".0f")]
    ).properties(
        width=600, height=400
    )

    return geo_map + scatter_plot
```

```
app = App(app_ui, server)
```
```

```
#top_alerts_map_byhour_sliderrange
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("/Users/zhengcui/Desktop/python
        2/student30538/problem_sets/ps6/top_alerts_map_byhour_sliderrange/basic-app/app3.py")
```

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

# Loading the essential data
top_alerts_map_byhour_sliderrange = pd.read_csv('/Users/zhengcui/Desktop/python
2/student30538/problem_sets/ps6/top_alerts_map_byhour_sliderrange/top_alerts_map_byhour.csv')
top_alerts_map_byhour_sliderrange['type_subtype'] =
top_alerts_map_byhour_sliderrange['updated_type'].str.capitalize() + ' - ' +
top_alerts_map_byhour_sliderrange['updated_subtype'].str.replace('_', ' ').str.title()
type_subtype_choices =
top_alerts_map_byhour_sliderrange['type_subtype'].drop_duplicates().sort_values().tolist()


top_alerts_map_byhour_sliderrange['hour'] = top_alerts_map_byhour_sliderrange['hour'].astype(str)


file_path = "/Users/zhengcui/Desktop/python 2/student30538/problem_sets/ps6/Boundaries -
Neighborhoods.geojson"
with open(file_path) as f:
    chicago_geojson = json.load(f)

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

#prepare UI,server and diagram
app_ui = ui.page_fluid(
    ui.panel_title("Traffic Alerts Dashboard"),
    ui.input_select(
        "selected_type_subtype",
        "Select Type-Subtype",
        choices=type_subtype_choices,
        selected=type_subtype_choices[0],
    ),
    ui.input_switch("switch_button", "Toggle to switch to range of hours", value=False),
    ui.output_ui("dynamic_slider"),
    output_widget("traffic_map"),
)

def server(input, output, session):
    @render.ui
    def dynamic_slider():
        if input.switch_button():
            return ui.input_slider(
                "selected_single_hour",
                "Select Single Hour",
                min=0,
                max=23,
                value=12,
                step=1,
            )
        else:
            return ui.input_slider(
                "selected_hour_range",
                "Select Hour Range",
                min=0,
                max=23,
                value=(6, 18),
                step=1,
```

```
        )

@output
@render_altair
def traffic_map():
    type_subtype = input.selected_type_subtype()
    if input.switch_button():

        selected_hour = f"{input.selected_single_hour():02}:00"

        essential_data = top_alerts_map_byhour_sliderrange[
            (top_alerts_map_byhour_sliderrange["type_subtype"] == type_subtype)
            & (top_alerts_map_byhour_sliderrange["hour"] == selected_hour)
        ]
    else:
        selected_hour_range = input.selected_hour_range()
        start_hour = f"{selected_hour_range[0]:02}:00"
        end_hour = f"{selected_hour_range[1]:02}:00"


        essential_data = top_alerts_map_byhour_sliderrange[
            (top_alerts_map_byhour_sliderrange["type_subtype"] == type_subtype)
            & (top_alerts_map_byhour_sliderrange["hour"] >= start_hour)
            & (top_alerts_map_byhour_sliderrange["hour"] <= end_hour)
        ]
    if essential_data.empty:
        return alt.Chart().mark_text(
            text="No data available for the selected range"
        ).properties(width=600, height=400)


        top_10_data = essential_data.nlargest(10, "count")
        return diagram(top_10_data)


def diagram(data):
    geo_map = alt.Chart(alt.Data(geo_data )).mark_geoshape(
        fill="lightgray", stroke="white"
    ).encode(
        tooltip="properties.pri_neigh:N"
    ).properties(
        width=600,
        height=400,
        title="Traffic Alerts Report-Chicago",
    ).project(type="mercator")


    scatter_plot = alt.Chart(data).mark_circle(
        stroke="black", strokeWidth=2
    ).encode(
        longitude="longitude:Q",
        latitude="latitude:Q",
        size=alt.Size(
            "count:Q",
            scale=alt.Scale(range=[20, 500]),
            title="Number of Alerts",
        ),
        color=alt.Color(
            "hour:O", scale=alt.Scale(scheme="category10"), title="Hour"
        ),
        tooltip=["longitude", "latitude", "count", "hour"],
    ).properties(
        width=600, height=400
    )

    return geo_map + scatter_plot


app = App(app_ui, server)


```

DataTransformerRegistry.enable('default')

Background

Data Download and Exploration (20 points)

1.

```
waze_data_sample = pd.read_csv('/Users/zhengcui/Desktop/python
        2/student30538/problem_sets/ps6/waze_data/waze_data_sample.csv')

print(waze_data_sample.head())

waze_data_sample_reported = waze_data_sample.drop(columns=['ts', 'geo', 'geoWKT'])
print(waze_data_sample_reported.dtypes)
```

```
   Unnamed: 0         city  confidence  nThumbsUp        street  \
0      584358  Chicago, IL           0         NaN           NaN
1      472915  Chicago, IL           0         NaN        I-90 E
2      550891  Chicago, IL           0         NaN        I-90 W
3      770659  Chicago, IL           0         NaN           NaN
4      381054  Chicago, IL           0         NaN  N Pulaski Rd

                                uuid country         type  \
0  c9b88a12-79e8-44cb-aadd-a75855fc4bcb      US          JAM
1  7c634c0a-099c-4262-b57f-e893bdebce73      US  ROAD_CLOSED
2  7aa3c61a-f8dc-4fe8-bbb0-db6b9e0dc53b      US       HAZARD
3  3b95dd2f-647c-46de-b4e1-8ebc73aa9221      US       HAZARD
4  13a5e230-a28a-4bf4-b928-bc1dd38850e0      US          JAM

                        subtype  roadType  reliability  magvar  \
0                           NaN        17            5     116
1             ROAD_CLOSED_EVENT         3            6     173
2  HAZARD_ON_SHOULDER_CAR_STOPPED         3            5     308
3                HAZARD_ON_ROAD        20            5     155
```

```
4               JAM_HEAVY_TRAFFIC         7             5     178

   reportRating                        ts                          geo  \
0             5  2024-07-02 18:27:40 UTC   POINT(-87.64577 41.892743)
1             0  2024-06-16 10:13:19 UTC  POINT(-87.646359 41.886295)
2             5  2024-05-02 19:01:47 UTC   POINT(-87.695982 41.93272)
3             2  2024-03-25 18:53:24 UTC  POINT(-87.669253 41.904497)
4             2  2024-06-03 21:17:33 UTC  POINT(-87.728322 41.978769)

                       geoWKT
0   Point(-87.64577 41.892743)
1  Point(-87.646359 41.886295)
2   Point(-87.695982 41.93272)
3  Point(-87.669253 41.904497)
4  Point(-87.728322 41.978769)
Unnamed: 0        int64
city             object
confidence        int64
nThumbsUp       float64
street           object
uuid             object
country          object
type             object
subtype          object
roadType          int64
reliability       int64
magvar            int64
reportRating      int64
dtype: object
```

Explanation

Variable Names

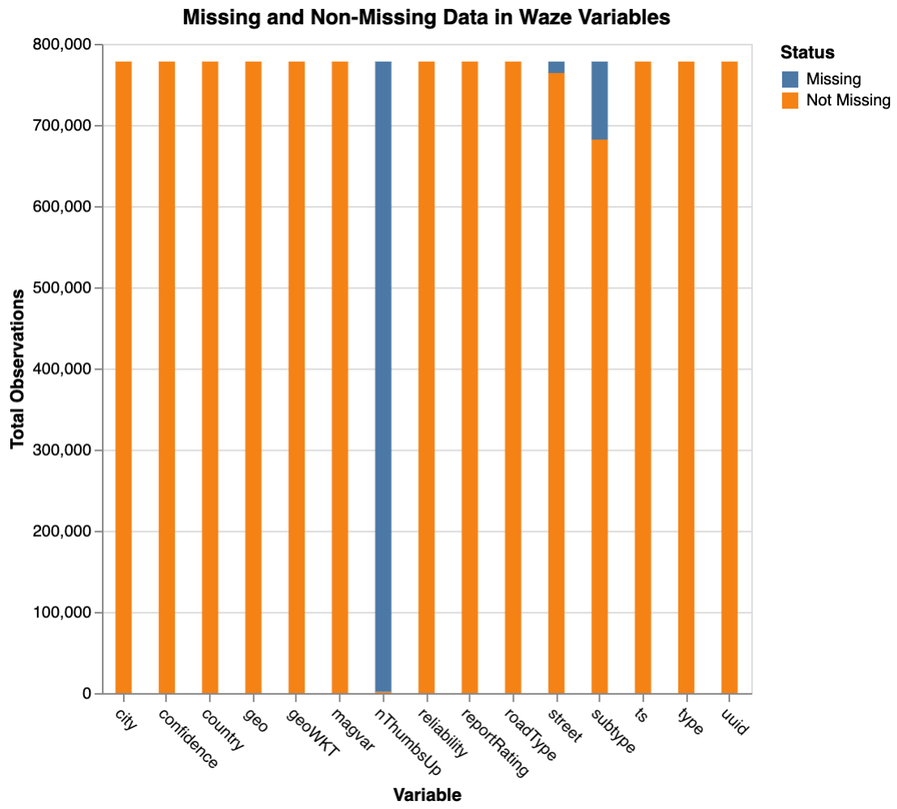
There are 16 different variables in this data set and they are : city, confidence, nThumbsUp, street, uuid, country, type, subtype, roadType, reliability, magvar, reportRating, ts, geo, geoWKT

Data Type

Variable name	Variable types
city	Nominal
confidence	Ordinal
nThumbsUp	Quantitative
street	Nominal
uuid	Nominal
country	Nominal
type	Nominal
subtype	Nominal
roadType	Nominal
reliability	Ordinal
magvar	Quantitative
reportRating	Ordinal

2.

```
#loading data
waze_data = pd.read_csv('/Users/zhengcui/Desktop/python
        2/student30538/problem_sets/ps6/waze_data/waze_data.csv')

waze_miss = waze_data.isnull().sum()

waze_non_miss = waze_data.notnull().sum()

data_to_plot = pd.DataFrame({
    'Missing': waze_miss,
    'Not Missing': waze_non_miss
}).reset_index()

#plotting
data_to_plot = data_to_plot.melt(id_vars='index', var_name='Status', value_name='Count')
data_to_plot.rename(columns={'index': 'Variable'}, inplace=True)

chart_to_plot = alt.Chart(data_to_plot).mark_bar(size=10).encode(
    x=alt.X('Variable:N', axis = alt.Axis(labelAngle=45,labelFontSize=10)),
    y=alt.Y('sum(Count):Q', title='Total Observations'),
    color='Status:N',
    tooltip=[alt.Tooltip('Variable:N'), alt.Tooltip('sum(Count):Q', title='Total'),
        alt.Tooltip('Status:N')]
).properties(
    title='Missing and Non-Missing Data in Waze Variables',
    width=400,
    height=400
)

chart_to_plot
```



Explanation

Based on the plot we have, there are three variables are having null values. They are nThumbsUp, street, subtype. And, the nThumbsUp has the highest share of observations that are missing.

3.

a

```
#3a
#Subset the data which only have type and subtype
type_data = waze_data[['type', 'subtype']]

#Find out how many types have a sub-type that is NA
na_subtype = type_data[type_data['subtype'].isnull()]

unique_na_subtype = na_subtype['type'].nunique()

print(f"Number of types with NA subtype: {unique_na_subtype}")


#Prepare the crosswalk table
subtype_counts =
        type_data[type_data['subtype'].notnull()].groupby('type').size().reset_index(name='subtype
        count')

subtype_miss_counts =
        type_data[type_data['subtype'].isnull()].groupby('type').size().reset_index(name='subtype
        NA')

total_subtype_counts = type_data.groupby('type').size().reset_index(name = 'Total counts')

#Merge to form a crosswalk table and identify which one has completed informaiton
crosswalk_table = pd.merge(total_subtype_counts, subtype_counts, on='type', how='outer')
crosswalk_table = pd.merge(crosswalk_table, subtype_miss_counts, on='type', how='outer')
crosswalk_table['completed rate %'] = (crosswalk_table['subtype count'] / crosswalk_table['Total
        counts']).round(4) * 100

print(crosswalk_table)
```

```
Number of types with NA subtype: 4
          type  Total counts  subtype count  subtype NA  completed rate %
0     ACCIDENT         33537           9178       24359             27.37
1       HAZARD        316063         312851        3212             98.98
2          JAM        372485         317444       55041             85.22
3  ROAD_CLOSED         56009          42535       13474             75.94
```

Explanation

There are 4 types types(ACCIDENT, HAZARD, JAM, and ROAD_CLOSED) have number of sub-type data that are NA. Based on the completed rate we have calculated, the HAZARD, JAM, and ROAD_CLOSED have prettry much completed rate of subtype data. And ACCIDENT has lowest completed rate which means ACCIDENT might also has lowest number of sub-subtypes data. Moreover, HAZARD, JAM and Road_CLOSED might have more sub-subtypes data.

b

```
#3b
#Bulleted list
#In order to make bulletd list, we firstly divide the dataframe by each type

#Accident
type_data_clean = type_data.copy()
type_data_clean['subtype'] = type_data_clean['subtype'].fillna('Unclassified')

accident_df = type_data_clean[type_data_clean['type'] == 'ACCIDENT'].copy()

accident_df= accident_df.groupby('type')['subtype'].unique().reset_index()

accident_df = pd.DataFrame({
    'type': accident_df.loc[0, 'type'],
    'subtype': accident_df.loc[0, 'subtype']
})

accident_df['updated_subtype'] = accident_df['subtype'].apply(lambda x: x.split('_')[1].title() if
        '_' in x else x.title())
accident_df['updated_subsubtype'] = 'Unclassified'



#Hazard
hazard_df = type_data_clean[type_data_clean['type'] == 'HAZARD'].copy()

hazard_df= hazard_df.groupby('type')['subtype'].unique().reset_index()

hazard_df = pd.DataFrame({
    'type': hazard_df.loc[0, 'type'],
    'subtype': hazard_df.loc[0, 'subtype']
```

```
})

def classify_subtype(subtype):

    subtype_lower = subtype.lower()


    if 'on_road' in subtype_lower:
        primary_subtype = 'On Road'
    elif 'on_shoulder' in subtype_lower:
        primary_subtype = 'On Shoulder'
    elif 'weather' in subtype_lower:
        primary_subtype = 'Weather'
    else:
        primary_subtype = 'Unclassified'


    if primary_subtype != 'Unclassified':

        subsubtype = subtype_lower.replace('hazard_', '').replace(primary_subtype.lower().replace('
        ', '_'), '').strip('_')
        subsubtype = ' '.join(subsubtype.split('_')).title() if subsubtype else 'Unclassified'
    else:
        subsubtype = 'Unclassified'

    return primary_subtype, subsubtype
```

```
hazard_df['updated_subtype'], hazard_df['updated_subsubtype'] =
        zip(*hazard_df['subtype'].map(classify_subtype))


#Jam

jam_df = type_data_clean[type_data_clean['type'] == 'JAM'].copy()

jam_df= jam_df.groupby('type')['subtype'].unique().reset_index()

jam_df = pd.DataFrame({
    'type': jam_df.loc[0, 'type'],
    'subtype': jam_df.loc[0, 'subtype']
})



def classify_jam(subtype):
    if 'TRAFFIC' in subtype:
        formatted_subtype = ' '.join(subtype.split('_')[1:]).title()
        return (formatted_subtype, 'Unclassified')
    return ('Unclassified', 'Unclassified')


jam_df[['updated_subtype', 'updated_subsubtype']] = jam_df['subtype'].apply(lambda x:
        classify_jam(x)).tolist()
```

```
#Road Closed

road_closed_df = type_data_clean[type_data_clean['type'] == 'ROAD_CLOSED'].copy()

road_closed_df= road_closed_df.groupby('type')['subtype'].unique().reset_index()

road_closed_df= pd.DataFrame({
    'type': road_closed_df.loc[0, 'type'],
    'subtype': road_closed_df.loc[0, 'subtype']
})


def classify_road_closed(subtype):
    parts = subtype.split('_')
    if len(parts) > 2:

        primary_subtype = parts[2].title()
    else:

        primary_subtype = 'Unclassified'


    subsubtype = 'Unclassified'
    return primary_subtype, subsubtype

# Apply classification to the dataframe
road_closed_df[['updated_subtype', 'updated_subsubtype']] = road_closed_df['subtype'].apply(lambda x:
        classify_road_closed(x)).tolist()
```

```
completed_df = pd.concat([accident_df, hazard_df, jam_df, road_closed_df])

completed_df['updated_type'] = completed_df['type']

completed_df['updated_type'] = completed_df['updated_type'].replace('ROAD_CLOSED', 'ROAD CLOSED',
        regex=True)

completed_df = completed_df[['type', 'subtype', 'updated_type', 'updated_subtype',
        'updated_subsubtype']]

completed_df.reset_index(drop=True, inplace=True)


# Creating a dictionary and be prepared to be used to create bulleted list
hierarchy_dict = {}
for i, row in completed_df.iterrows():
    type_level = row['updated_type'].capitalize()
    subtype_level = row['updated_subtype'].replace('_', ' ').title()
    subsubtype_level = row['updated_subsubtype'].replace('_', ' ').title()

    if type_level not in hierarchy_dict:
        hierarchy_dict[type_level] = {}
    if subtype_level not in hierarchy_dict[type_level]:
        hierarchy_dict[type_level][subtype_level] = []
    hierarchy_dict[type_level][subtype_level].append(subsubtype_level)
```

```
# Printing bulleted list
for type_key, subtypes in hierarchy_dict.items():
    print(f"- {type_key}")
    for subtype_key, subsubtypes in subtypes.items():
        print(f"  - {subtype_key}")
        for subsubtype in subsubtypes:
            print(f"    - {subsubtype}")
```

```
- Accident
  - Unclassified
    - Unclassified
  - Major
    - Unclassified
  - Minor
    - Unclassified
- Hazard
  - Unclassified
    - Unclassified
  - On Road
    - Unclassified
    - Car Stopped
    - Construction
    - Emergency Vehicle
    - Ice
    - Object
    - Pot Hole
    - Traffic Light Fault
    - Lane Closed
    - Road Kill
  - On Shoulder
    - Unclassified
    - Car Stopped
    - Animals
    - Missing Sign
  - Weather
    - Unclassified
    - Flood
    - Fog
    - Heavy Snow
    - Hail
- Jam
  - Unclassified
    - Unclassified
  - Heavy Traffic
    - Unclassified
  - Moderate Traffic
    - Unclassified
  - Stand Still Traffic
    - Unclassified
  - Light Traffic
    - Unclassified
- Road closed
  - Unclassified
    - Unclassified
  - Event
    - Unclassified
  - Construction
    - Unclassified
  - Hazard
    - Unclassified
```

c

```
#3c
#Mark NA as unclassfied:

waze_data['subtype'] = waze_data['subtype'].fillna('Unclassified')

#all the subtype NA values have been replaced by 'Unclassified'
waze_data.head(10)
```

	city	confidence	nThumbsUp	street	uuid	country	type	subtype	roadType	reliability	magv
0	Chicago, IL	0	NaN	NaN	004025a4-5f14-4cb7-9da6-2615daafbf37	US	JAM	Unclassified	20	5	139
1	Chicago, IL	1	NaN	NaN	ad7761f8-d3cb-4623-951d-dafb419a3ec3	US	ACCIDENT	Unclassified	4	8	2
2	Chicago, IL	0	NaN	NaN	0e5f14ae-7251-46af-a7f1-53a5272cd37d	US	ROAD_CLOSED	Unclassified	1	5	344
3	Chicago, IL	0	NaN	Alley	654870a4-a71a-450b-9f22-bc52ae4f69a5	US	JAM	Unclassified	20	5	264
4	Chicago, IL	0	NaN	Alley	926ff228-7db9-4e0d-b6cf-6739211ffc8b	US	JAM	Unclassified	20	5	359
5	Chicago, IL	0	NaN	Alley	7889ce93-b70d-4da5-8b3d-9c5c0f240f28	US	ROAD_CLOSED	Unclassified	20	5	344
6	Chicago, IL	0	NaN	DuSable Lake Shore Dr	49c16f14-3c68-445e-9597-f93ed399b724	US	JAM	Unclassified	6	5	153
7	Chicago, IL	1	NaN	DuSable Lake Shore Dr	aba17ef8-bbd2-4bd2-a422-f398840f0654	US	ACCIDENT	Unclassified	6	9	166
8	Chicago, IL	0	NaN	DuSable Lake Shore Dr	845fd8f0-2542-42d2-a39a-d81bd238ab7e	US	ACCIDENT	Unclassified	6	5	311

	city	confidence	nThumbsUp	street	uuid	country	type	subtype	roadType	reliability	magv
9	Chicago, IL	0	NaN	DuSable Lake Shore Dr	c0204977-8f76-48f0-acfe-42bbe7f75dc4	US	ACCIDENT	Unclassified	6	5	141

Explanation

I think that we should keep all the NA values because if we remove them, we will lose the valuable data in other column(type) which are not missing. In other words, the noncompleted data would likely cause us to conclude biased result, Thus, we would keep the NA values and mark them(subtype-NA) as Unclassfied.

4.

a.

```
#4a create crosswalk
crosswalk = pd.DataFrame(columns=['type', 'subtype', 'updated_type', 'updated_subtype',
        'updated_subsubtype'])

print(crosswalk)
```

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

b.

```
#4b
#We can use the completed_df which we come up from the previous question to form a crosswalk. Becasue
        the completed_df contains 32 observation which question required:
crosswalk = completed_df
print(crosswalk.head(32))
```

```
          type                          subtype updated_type  \
0      ACCIDENT                     Unclassified     ACCIDENT
1      ACCIDENT                   ACCIDENT_MAJOR     ACCIDENT
2      ACCIDENT                   ACCIDENT_MINOR     ACCIDENT
3        HAZARD                     Unclassified       HAZARD
4        HAZARD                   HAZARD_ON_ROAD       HAZARD
5        HAZARD          HAZARD_ON_ROAD_CAR_STOPPED       HAZARD
6        HAZARD        HAZARD_ON_ROAD_CONSTRUCTION       HAZARD
7        HAZARD    HAZARD_ON_ROAD_EMERGENCY_VEHICLE       HAZARD
8        HAZARD                 HAZARD_ON_ROAD_ICE       HAZARD
9        HAZARD              HAZARD_ON_ROAD_OBJECT       HAZARD
10       HAZARD            HAZARD_ON_ROAD_POT_HOLE       HAZARD
11       HAZARD  HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT       HAZARD
12       HAZARD                 HAZARD_ON_SHOULDER       HAZARD
13       HAZARD     HAZARD_ON_SHOULDER_CAR_STOPPED       HAZARD
14       HAZARD                   HAZARD_WEATHER       HAZARD
15       HAZARD              HAZARD_WEATHER_FLOOD       HAZARD
16       HAZARD          HAZARD_ON_ROAD_LANE_CLOSED       HAZARD
17       HAZARD                 HAZARD_WEATHER_FOG       HAZARD
18       HAZARD           HAZARD_ON_ROAD_ROAD_KILL       HAZARD
19       HAZARD          HAZARD_ON_SHOULDER_ANIMALS       HAZARD
20       HAZARD     HAZARD_ON_SHOULDER_MISSING_SIGN       HAZARD
21       HAZARD          HAZARD_WEATHER_HEAVY_SNOW       HAZARD
22       HAZARD               HAZARD_WEATHER_HAIL       HAZARD
23          JAM                     Unclassified          JAM
24          JAM                JAM_HEAVY_TRAFFIC          JAM
25          JAM             JAM_MODERATE_TRAFFIC          JAM
26          JAM           JAM_STAND_STILL_TRAFFIC          JAM
27          JAM                JAM_LIGHT_TRAFFIC          JAM
28  ROAD_CLOSED                     Unclassified  ROAD CLOSED
29  ROAD_CLOSED              ROAD_CLOSED_EVENT  ROAD CLOSED
30  ROAD_CLOSED         ROAD_CLOSED_CONSTRUCTION  ROAD CLOSED
31  ROAD_CLOSED               ROAD_CLOSED_HAZARD  ROAD CLOSED

       updated_subtype   updated_subsubtype
0         Unclassified         Unclassified
1                Major         Unclassified
2                Minor         Unclassified
3         Unclassified         Unclassified
4              On Road         Unclassified
5              On Road          Car Stopped
6              On Road         Construction
7              On Road    Emergency Vehicle
8              On Road                  Ice
9              On Road               Object
10             On Road             Pot Hole
11             On Road  Traffic Light Fault
12          On Shoulder         Unclassified
13          On Shoulder          Car Stopped
14              Weather         Unclassified
15              Weather                Flood
16             On Road          Lane Closed
17              Weather                  Fog
18             On Road            Road Kill
19          On Shoulder              Animals
20          On Shoulder         Missing Sign
21              Weather           Heavy Snow
22              Weather                 Hail
23        Unclassified         Unclassified
24       Heavy Traffic         Unclassified
25    Moderate Traffic         Unclassified
26  Stand Still Traffic         Unclassified
27       Light Traffic         Unclassified
28        Unclassified         Unclassified
29               Event         Unclassified
30        Construction         Unclassified
31              Hazard         Unclassified
```

c.

```
#4c
#Merge crosswalk into orignal data
merged_df = waze_data.merge(crosswalk, on=['type', 'subtype'], how='left')

#Find out how many rows are there for Accident - Uncalssified
num_rows = merged_df[(merged_df['type'] == 'ACCIDENT') & (merged_df['updated_subtype'] ==
        'Unclassified')].shape[0]
print("Number of rows for Accident - Unclassified:", num_rows)
```

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

d.

```
#Extra Credit:
crosswalk['type'].nunique() == merged_df['type'].nunique()

crosswalk['subtype'].nunique() == merged_df['subtype'].nunique()
```

```
True
```

Explanation

Based on the result we have, the new merged dataset have the same number values in type and subtype.

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

1.

a.

```
import re

#write a function to split the points into latitude and longitude.
def extract_lat_lon(point):
    split_point = point.replace('POINT(', '').replace(')', '')
    parts = split_point.split()
    return float(parts[0]), float(parts[1])

#create two columns 'latitude' and longitude to
merged_df['longitude'], merged_df['latitude'] = zip(*merged_df['geo'].apply(extract_lat_lon))

#Here is the result to show the points have been splitted into latitude and longitude.
print(merged_df.head(2))

#Here is the GPT response:

#If you have multiple coordinates combined in a single string or column of a dataframe and you want
        to split them into separate latitude and longitude values, you can use Python to achieve
        this. Here's how you can handle it in different scenarios:

# List of coordinate pairs as strings
# coordinates = ["34.0522, -118.2437", "40.7128, -74.0060"]

# # Splitting each pair into latitude and longitude
# latitudes = []
# longitudes = []
# for coord in coordinates:
#     lat, lon = coord.split(',')
#     latitudes.append(float(lat.strip()))
#     longitudes.append(float(lon.strip()))

# print(latitudes, longitudes)

#Here is conversation link:
# https://chatgpt.com/share/673bf16a-3aa4-8012-97a3-dfc3dde9b401
```

```
          city  confidence  nThumbsUp street  \
0  Chicago, IL           0        NaN    NaN
1  Chicago, IL           1        NaN    NaN

                                uuid country      type       subtype  \
0  004025a4-5f14-4cb7-9da6-2615daafbf37      US       JAM  Unclassified
1  ad7761f8-d3cb-4623-951d-dafb419a3ec3      US  ACCIDENT  Unclassified

   roadType  reliability  magvar  reportRating                       ts  \
0        20            5     139             3  2024-02-04 16:40:41 UTC
1         4            8       2             2  2024-02-04 20:01:27 UTC

                          geo                     geoWKT updated_type  \
0  POINT(-87.676685 41.929692)  Point(-87.676685 41.929692)          JAM
1  POINT(-87.624816 41.753358)  Point(-87.624816 41.753358)     ACCIDENT

  updated_subtype updated_subsubtype  longitude   latitude
0    Unclassified       Unclassified -87.676685  41.929692
1    Unclassified       Unclassified -87.624816  41.753358
```

b.

```
def bin (value):
  return np.round(value,2)

merged_df['longitude'] = merged_df['longitude'].apply(bin)
merged_df['latitude'] = merged_df['latitude'].apply(bin)

coordinates_counts = merged_df.groupby(['longitude', 'latitude']).size().reset_index(name='count')

coordinates_counts[coordinates_counts['count'] == coordinates_counts['count'].max()]
```

	longitude	latitude	count
492	-87.65	41.88	21325

Explanation

(-87.65,41.88) has the greatest number of observations in the overall dataset.

c.

```
#Here is the DataFrame
top_10_alert_df = merged_df[['updated_type','updated_subtype','longitude','latitude']]

top_10_alert_df = top_10_alert_df.groupby(['updated_type', 'updated_subtype', 'longitude',
        'latitude']).size().reset_index(name='count')

#Saved as csv in the local path
top_10_alert_df.to_csv('/Users/zhengcui/Desktop/python
        2/student30538/problem_sets/ps6/top_alerts_map/top_alerts_map.csv', index=False)

#Here is the DataFrame row
top_10_alert_df.shape[0]
```

```
6675
```

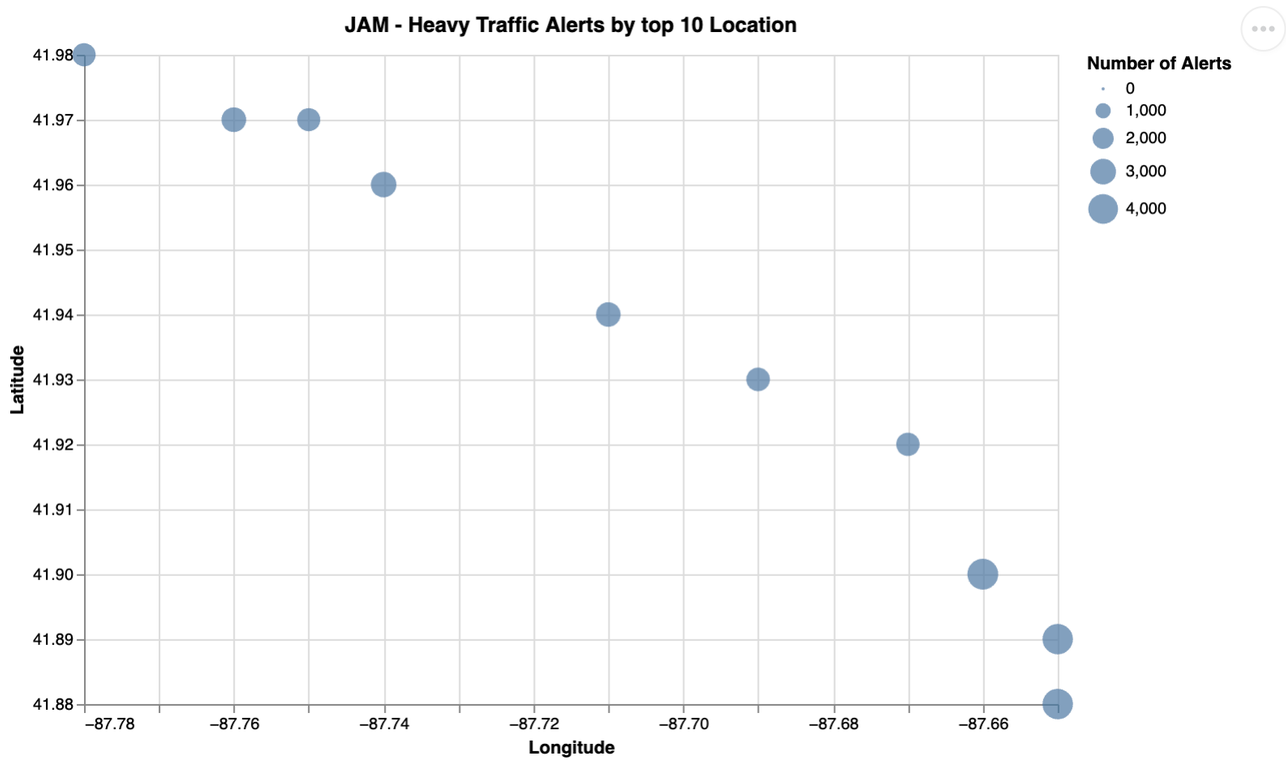
Explanation

For this level of aggregation, we are counting the number of alerts based on the combination of latitude-longitude, alert type, and alert subtype. This approach helps us focus on the top 10 locations (the combination of latitude and longitude) with the highest number of alerts for a particular alert type and subtype.

Based on the observations, there are 6675 rows in this DataFrame.

2.

```
#Extracting the essential data to plot the diagram
plot_jam_df = top_10_alert_df[(top_10_alert_df['updated_type'] == 'JAM') &
        (top_10_alert_df['updated_subtype'] == 'Heavy Traffic')]

top_10_jam = plot_jam_df.sort_values(by='count', ascending=False).head(10)


#plotting diagram
scatter_plot = alt.Chart(top_10_jam).mark_circle().encode(
    x=alt.X('longitude:Q', title='Longitude', scale=alt.Scale(domain=[top_10_jam['longitude'].min(),
        top_10_jam['longitude'].max()])),
    y=alt.Y('latitude:Q', title='Latitude', scale=alt.Scale(domain=[top_10_jam['latitude'].min(),
        top_10_jam['latitude'].max()])),
    size=alt.Size('count:Q', title='Number of Alerts')
).properties(
    width=600,
    height=400,
    title='JAM - Heavy Traffic Alerts by top 10 Location'
)

scatter_plot
```



3.

a.

```
#Extra credit
import requests

url = 'https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON'

web = requests.get(url)

print(web)
```

```
<Response [200]>
```

Explanation

Based on the response code, we can download the package successfully.

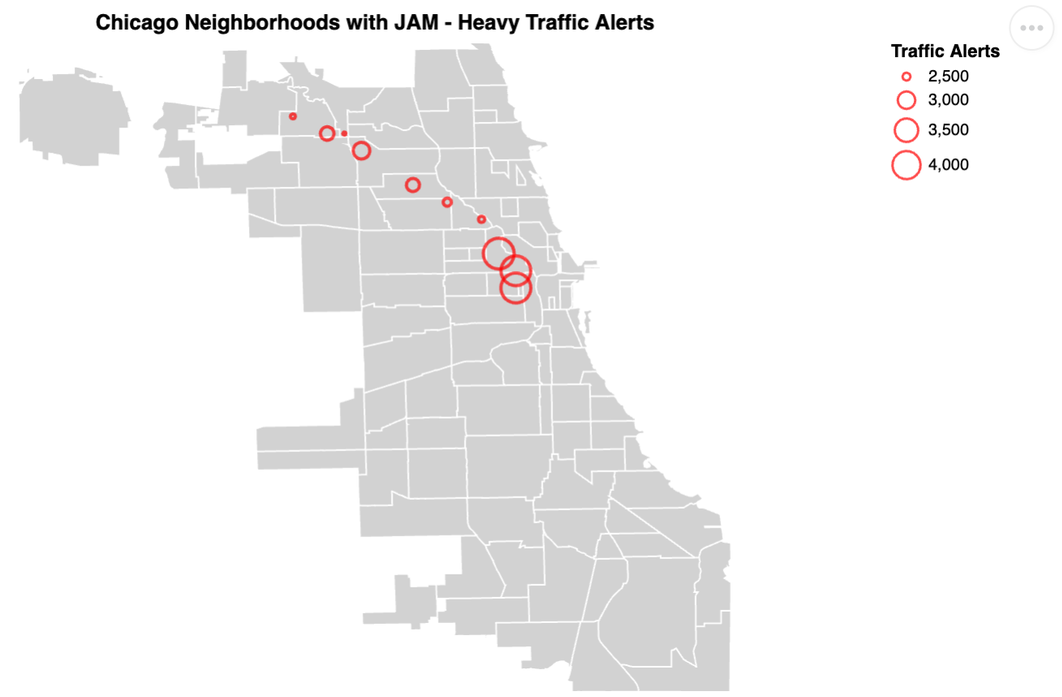
b.

```
# MODIFY ACCORDINGLY
file_path = "/Users/zhengcui/Desktop/python 2/student30538/problem_sets/ps6/Boundaries -
        Neighborhoods.geojson"

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

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

4.

```
#combining map and scatter plot
geo_map = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).encode(
    tooltip='properties.pri_neigh:N'
).properties(
    width=600,
    height=400,
    title='Chicago Neighborhoods with JAM - Heavy Traffic Alerts'
).project(
    type='equirectangular'
)


scatter_plot = alt.Chart(top_10_jam).mark_circle(
    fill=None,
    stroke='red',
    strokeWidth=2
).encode(
    longitude='longitude:Q',
```

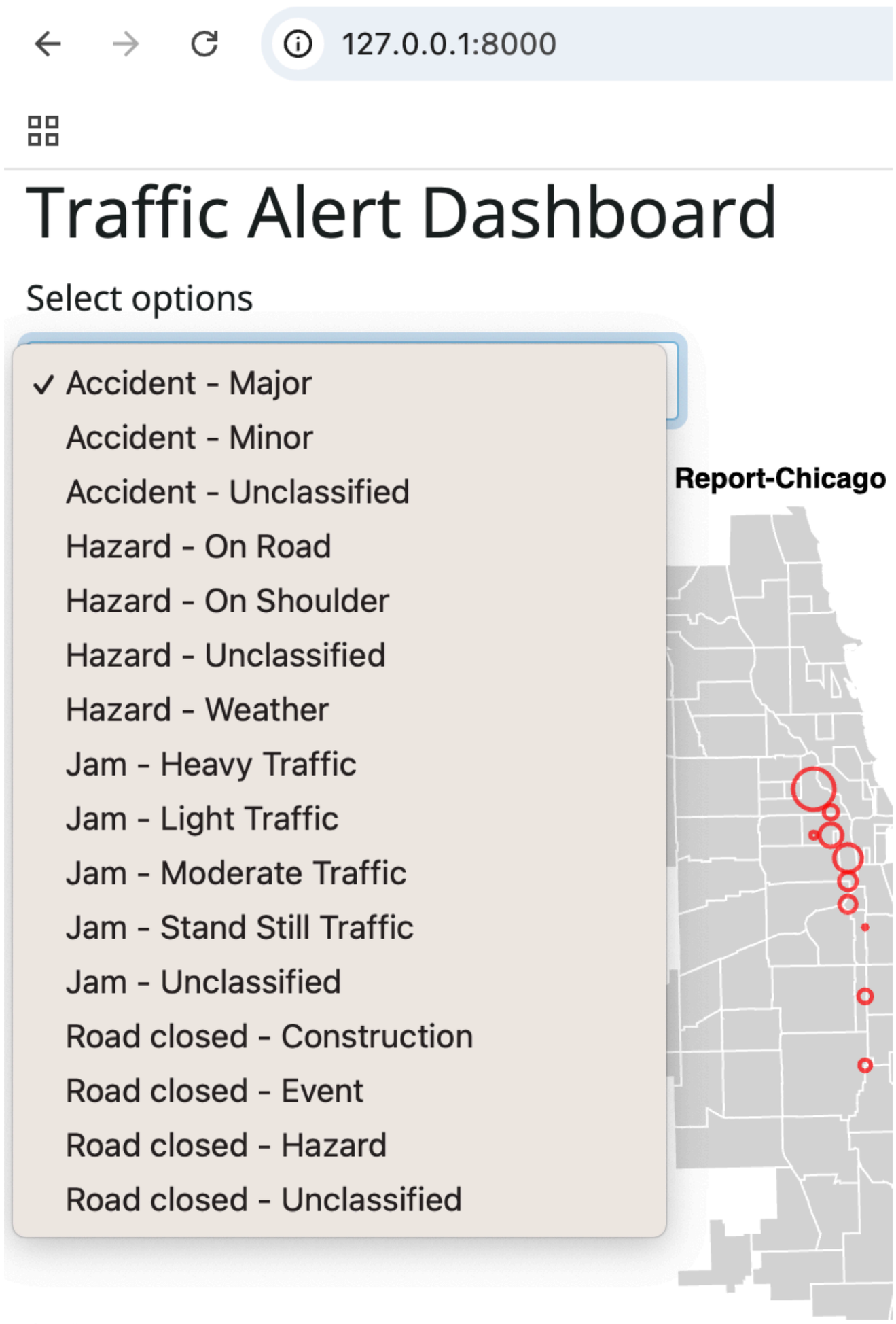
```
    latitude='latitude:Q',
    size=alt.Size('count:Q', title='Number of Alerts',
                  scale=alt.Scale(domain=[top_10_jam['count'].min(), top_10_jam['count'].max()]),
                  legend=alt.Legend(title="Traffic Alerts")),
    color=alt.value('red'),
    tooltip=['longitude', 'latitude', 'count']
).properties(
    width=600,
    height=400
)

geo_map + scatter_plot
```



5.

a.

Here is the screenshot of dropdown menu

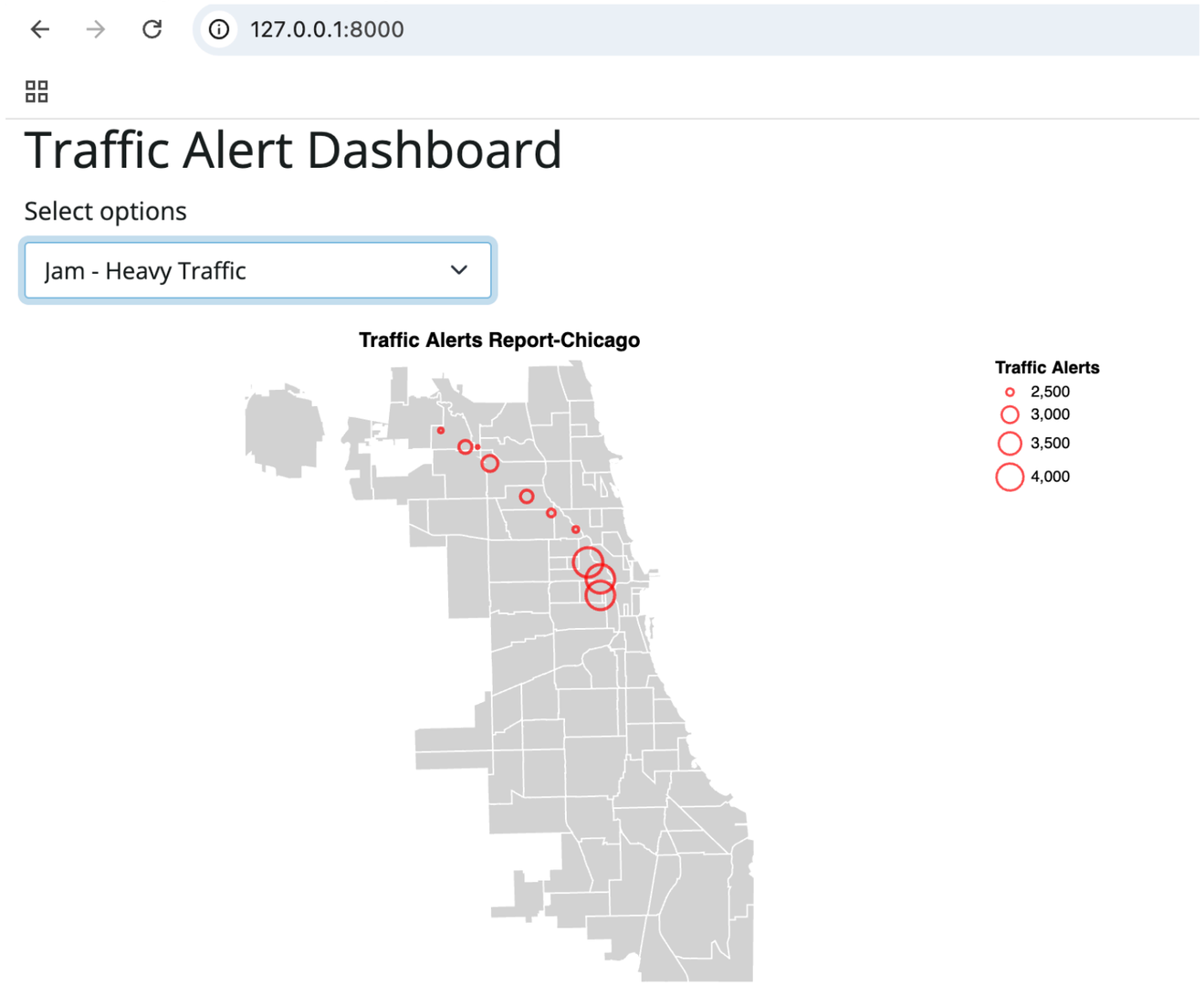


Explanation

Based on the diagram, there are 16 combinations of types & subtypes

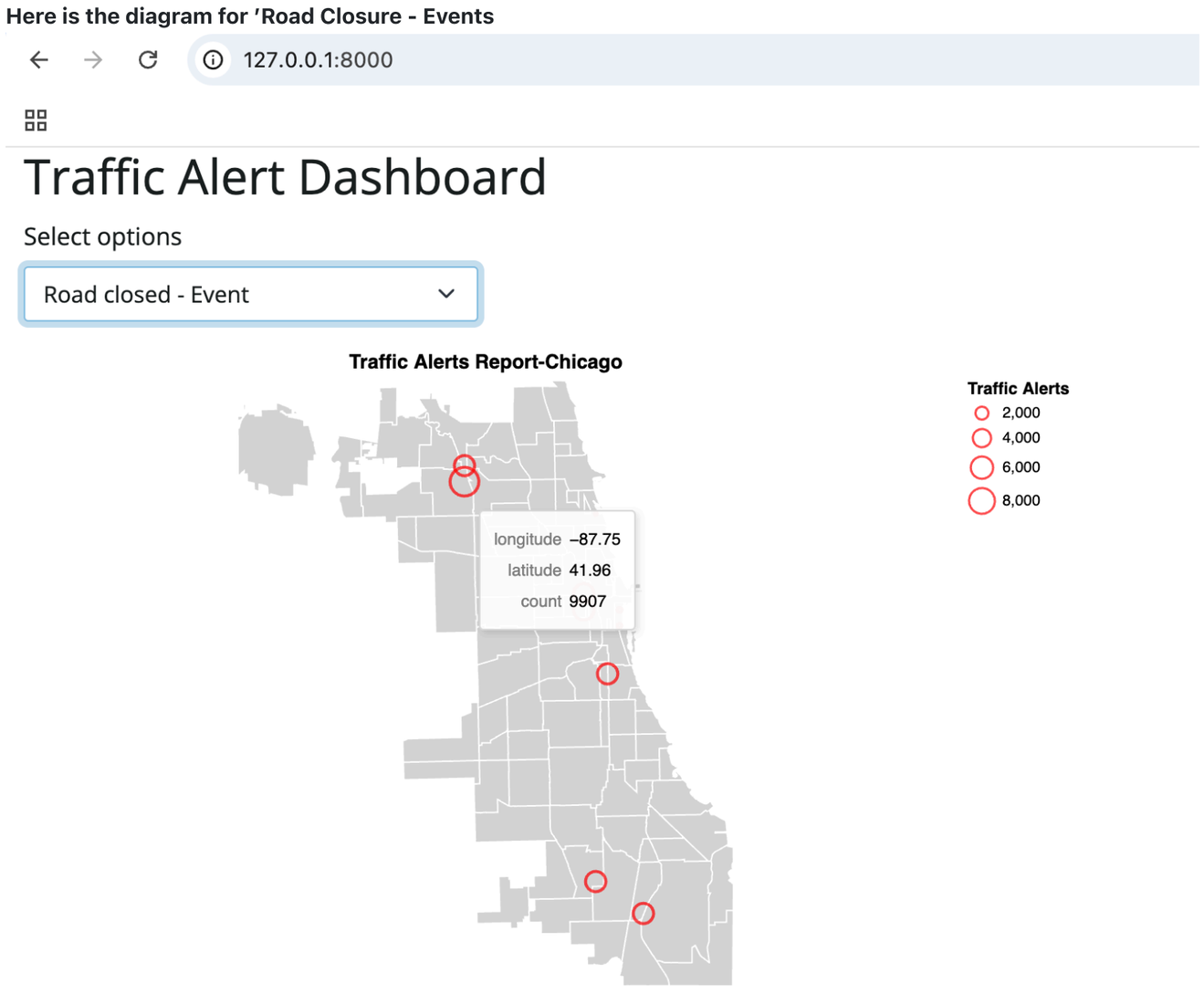
b.

Here is the diagram for "Jam - Heavy Traffic" plot



c.

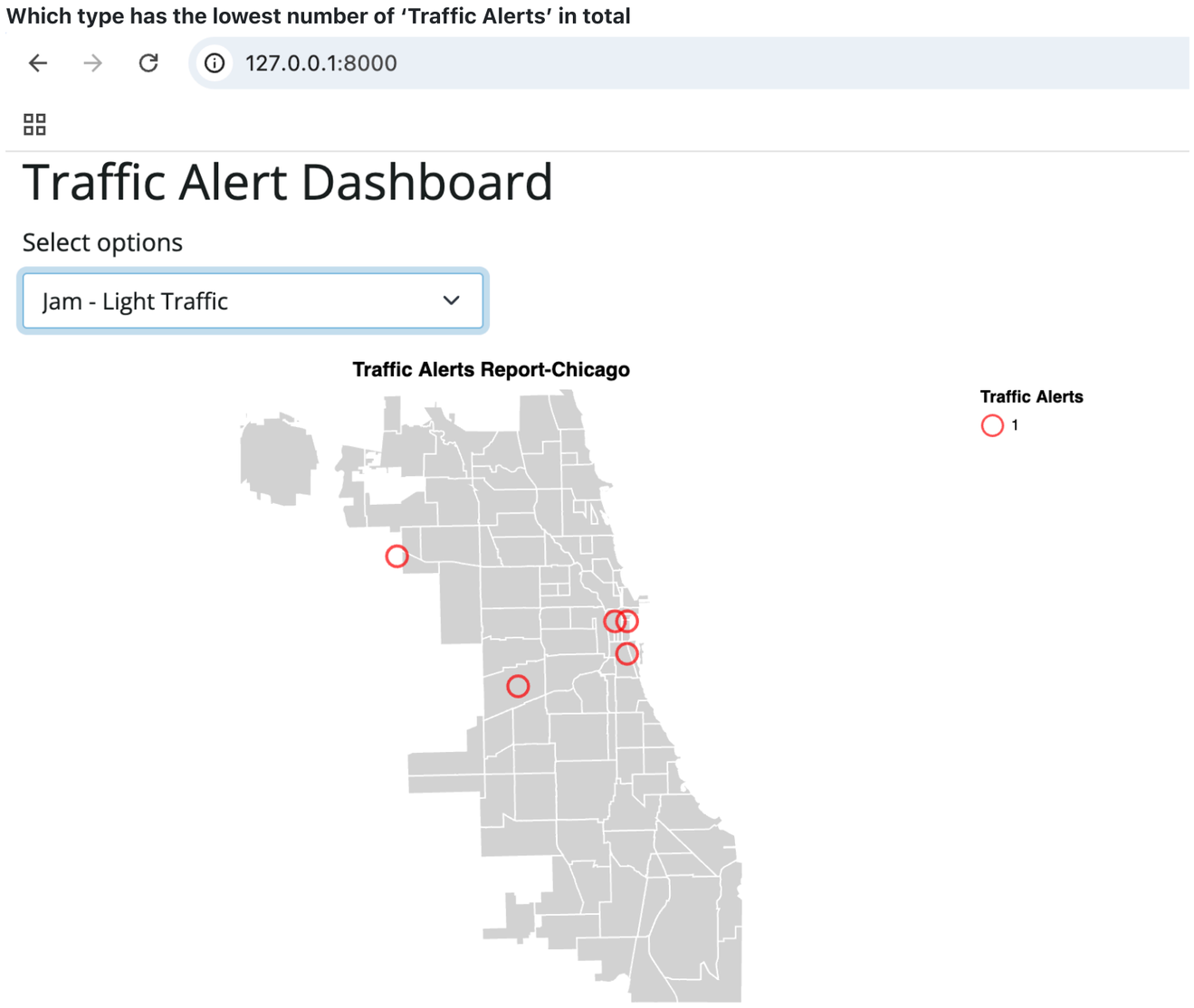
Here is the diagram for 'Road Closure - Events



Explanation

Based on the diagram, we can see the area with longitude -87.75 and latitude 41.96 where has the highest number of alerts 9907. In other words, it means that is the place where are alerts for road closures due to events most common

d.



e.

We can also add weather conditions as another column to the dashboard, allowing users to see which weather conditions have the most incidents.

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

1.

a.

No, it would not be a great idea if we simply collapse the dataset by the 'ts' column. Since the 'ts' column records different series of times, it would be difficult to provide meaningful aggregation. Also, with many distinct numbers showing up in this column, it would be difficult to keep track of what's going on with the traffic over time. It would be challenging to provide useful information and conclusions if we collapse the dataset by the 'ts' column. However, if we choose to collapse the dataset by considering to pick an hour of the day for the ts column, it would be a great idea because it would allow the users to keep tracking the number of alerts over time to time.

b.

```
#create a new dataframe
top_alerts_map_byhour =
        merged_df[['updated_type','updated_subtype','longitude','latitude','ts']].copy()

top_alerts_map_byhour['ts'] = pd.to_datetime(top_alerts_map_byhour['ts'])

#create an new variable 'hour'
top_alerts_map_byhour['hour'] = top_alerts_map_byhour['ts'].dt.strftime('%H:00')

top_alerts_map_byhour = top_alerts_map_byhour.groupby(['updated_type', 'updated_subtype',
        'longitude', 'latitude','hour']).size().reset_index(name='count')

#save the csv file into top_alerts_map_byhour folder
top_alerts_map_byhour.to_csv('/Users/zhengcui/Desktop/python
        2/student30538/problem_sets/ps6/top_alerts_map_byhour/top_alerts_map_byhour.csv',
        index=False)
#find out how many rows
top_alerts_map_byhour.shape[0]
```

```
62825
```

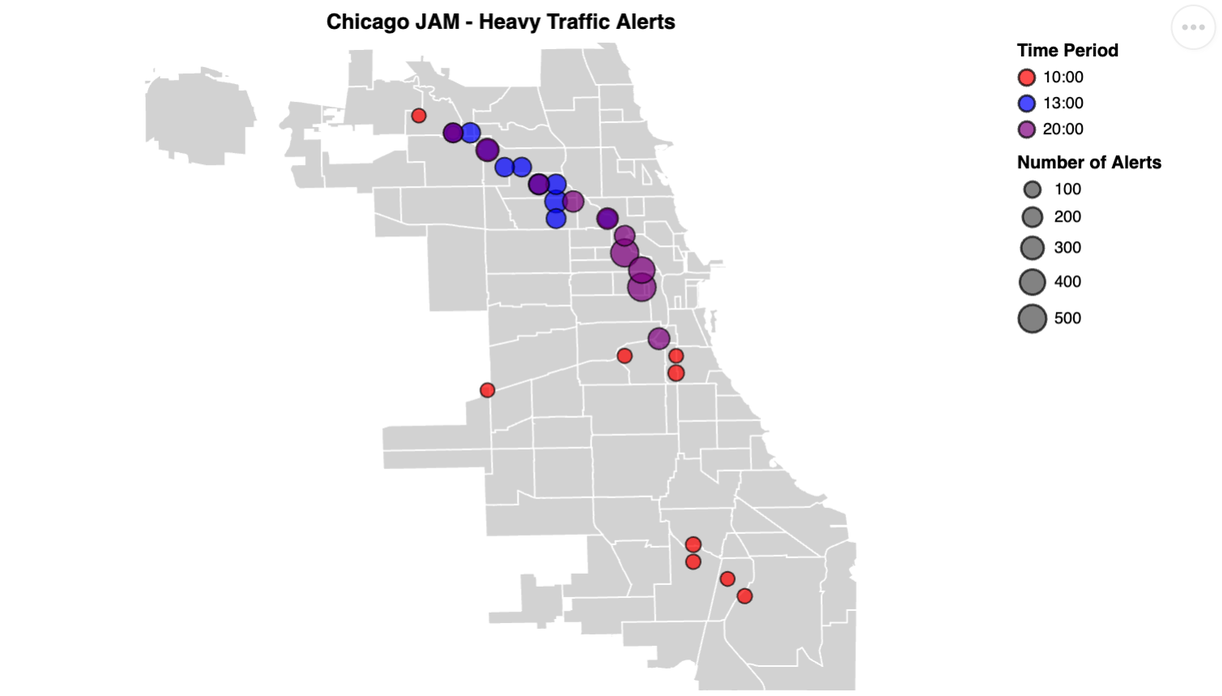
Explanation

Based on the results we have, there are 62825 rows in this dataset(top_alerts_map_byhour)

c.

```
#selecting the essential data
traffic_jam_hour = top_alerts_map_byhour[
    (top_alerts_map_byhour['updated_type'] == 'JAM') &
    (top_alerts_map_byhour['updated_subtype'] == 'Heavy Traffic')]

#pick up the time which we interest in
traffic_jam_hour = traffic_jam_hour[traffic_jam_hour['hour'].isin(['10:00', '13:00', '20:00'])]

#find out top 10 locations
top_locations_hour = traffic_jam_hour.groupby('hour').apply(lambda x: x.nlargest(10,
        'count')).reset_index(drop=True)

#plotting the diagram
geo_map_hour = alt.Chart(alt.Data(values=chicago_geojson['features'])).mark_geoshape(
    fill='lightgray',
    stroke='white'
).encode(
    tooltip='properties.pri_neigh:N'
).properties(
    width=600,
    height=400,
    title='Chicago JAM - Heavy Traffic Alerts'
).project('equirectangular')
```

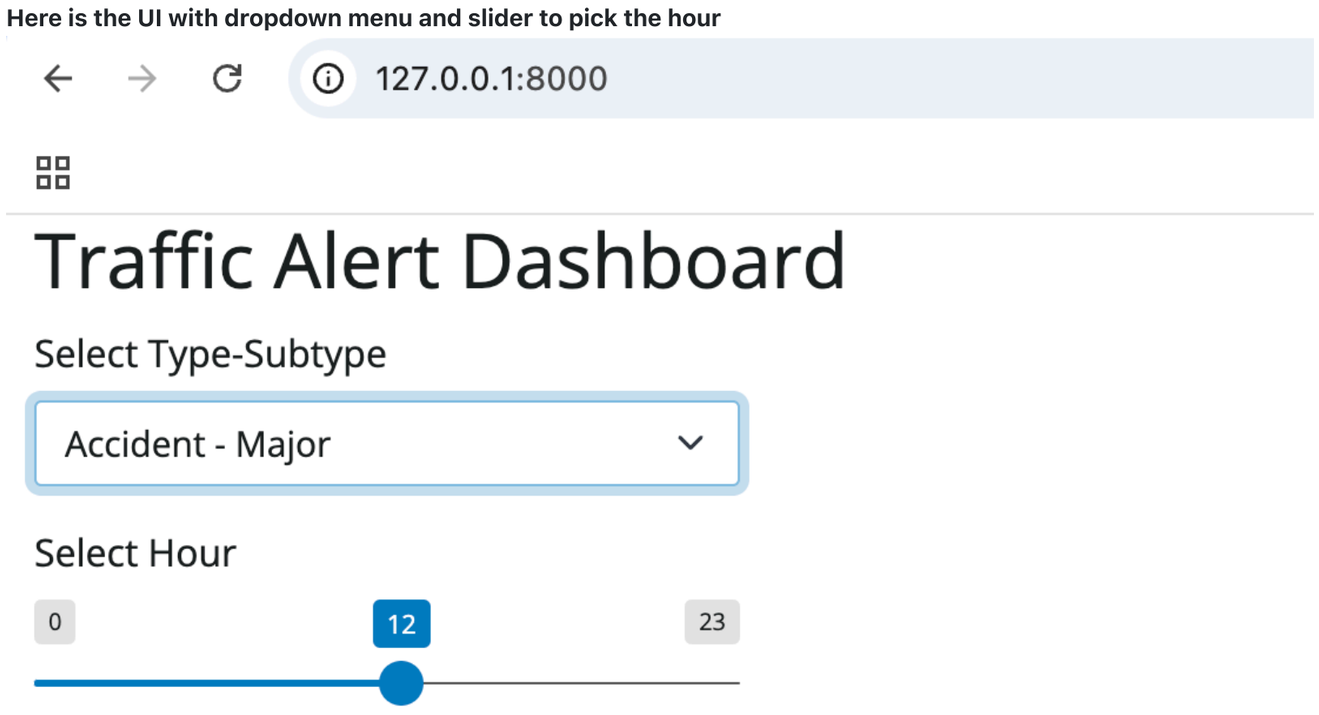
```
scatter_plot_hour = alt.Chart(top_locations_hour).mark_circle(
    stroke='black',
    strokeWidth=1
).encode(
    longitude='longitude:Q',
    latitude='latitude:Q',
    size=alt.Size('count:Q', title="Number of Alerts", scale=alt.Scale(domain=[1,
        top_locations_hour['count'].max()], range=[50, 300])),
    color=alt.Color('hour:N', scale=alt.Scale(domain=['10:00', '13:00', '20:00'], range=['red',
        'blue', 'purple']), legend=alt.Legend(title="Time Period")),
    tooltip=['count:Q', 'longitude:Q', 'latitude:Q', 'hour:N']
).properties(
    title='Top Traffic Jam Locations by Hour'
)


combined_chart_hour = alt.layer(geo_map_hour, scatter_plot_hour)

combined_chart_hour
```



2.

a.

Here is the UI with dropdown menu and slider to pick the hour



b.

Here are the screenshots for each time period above

10:00

127.0.0.1:8000

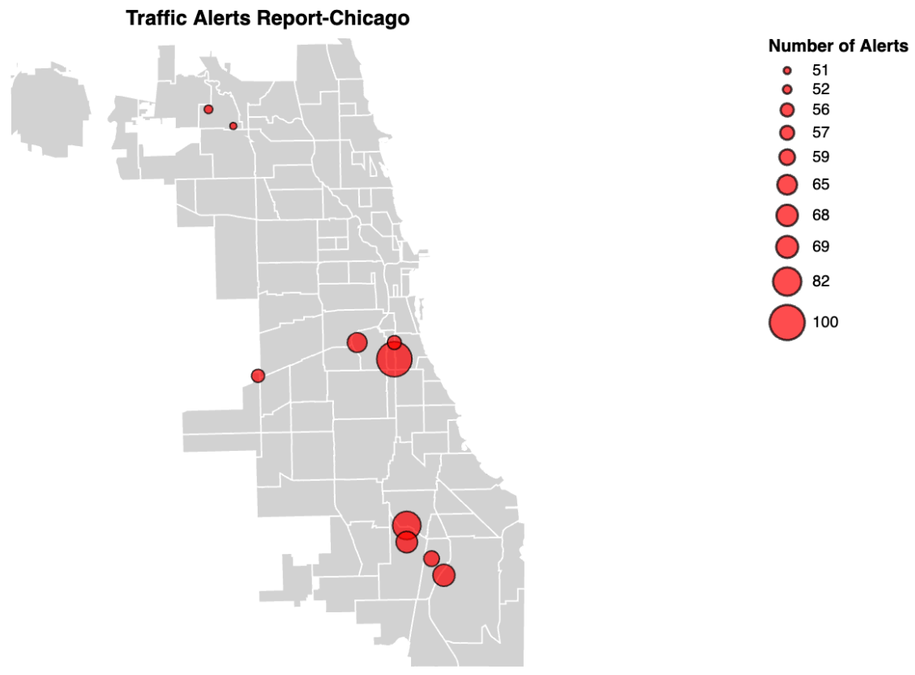
Traffic Alert Dashboard

Select Type-Subtype

Jam - Heavy Traffic

Select Hour

0 10 23



13:00

127.0.0.1:8000

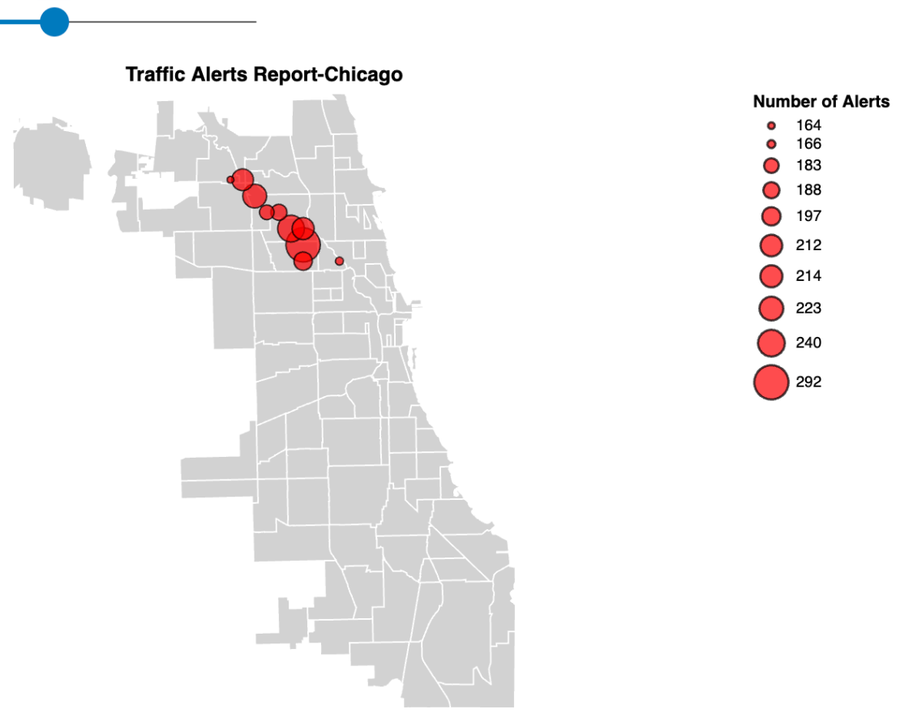
Traffic Alert Dashboard

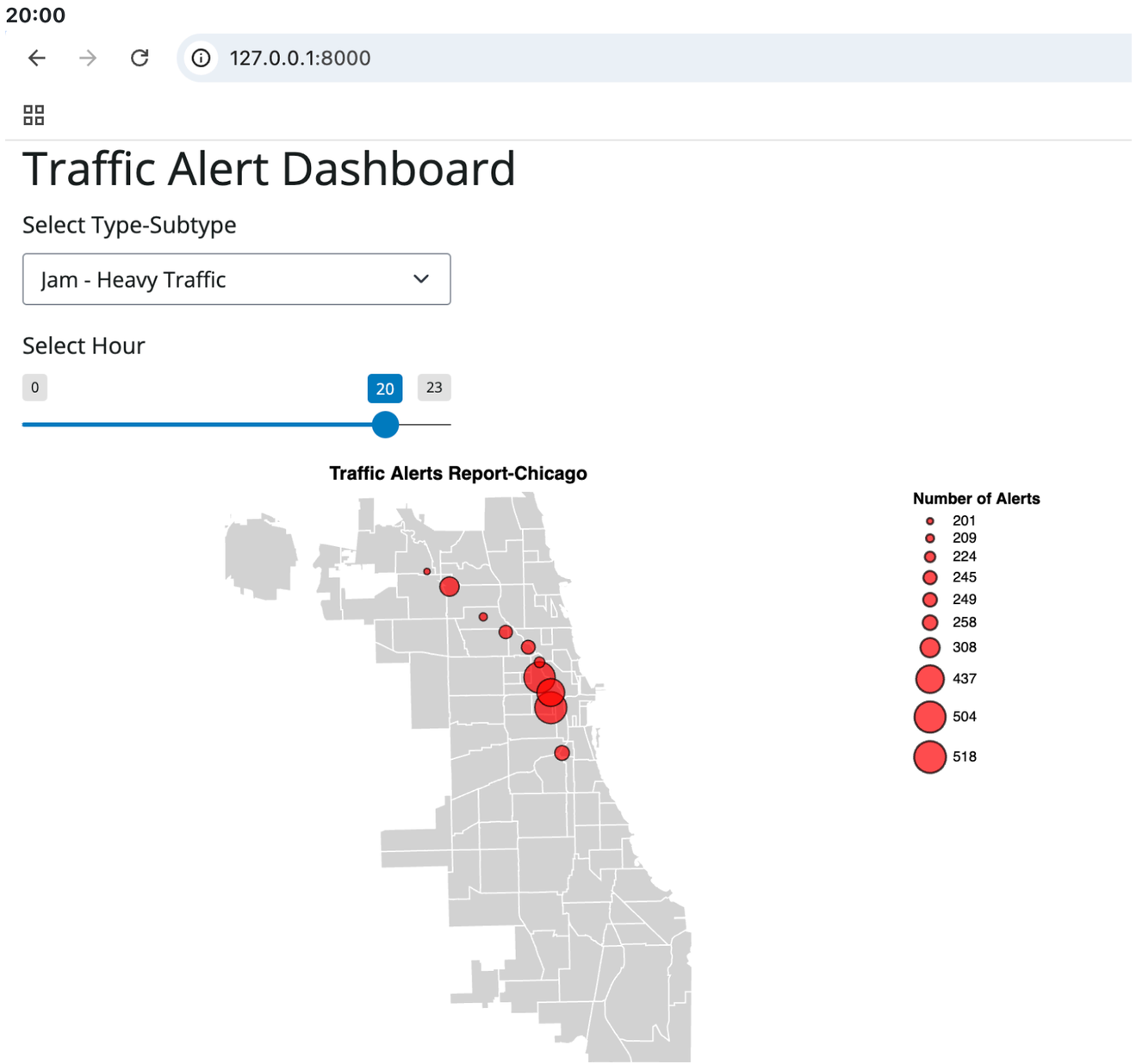
Select Type-Subtype

Jam - Heavy Traffic

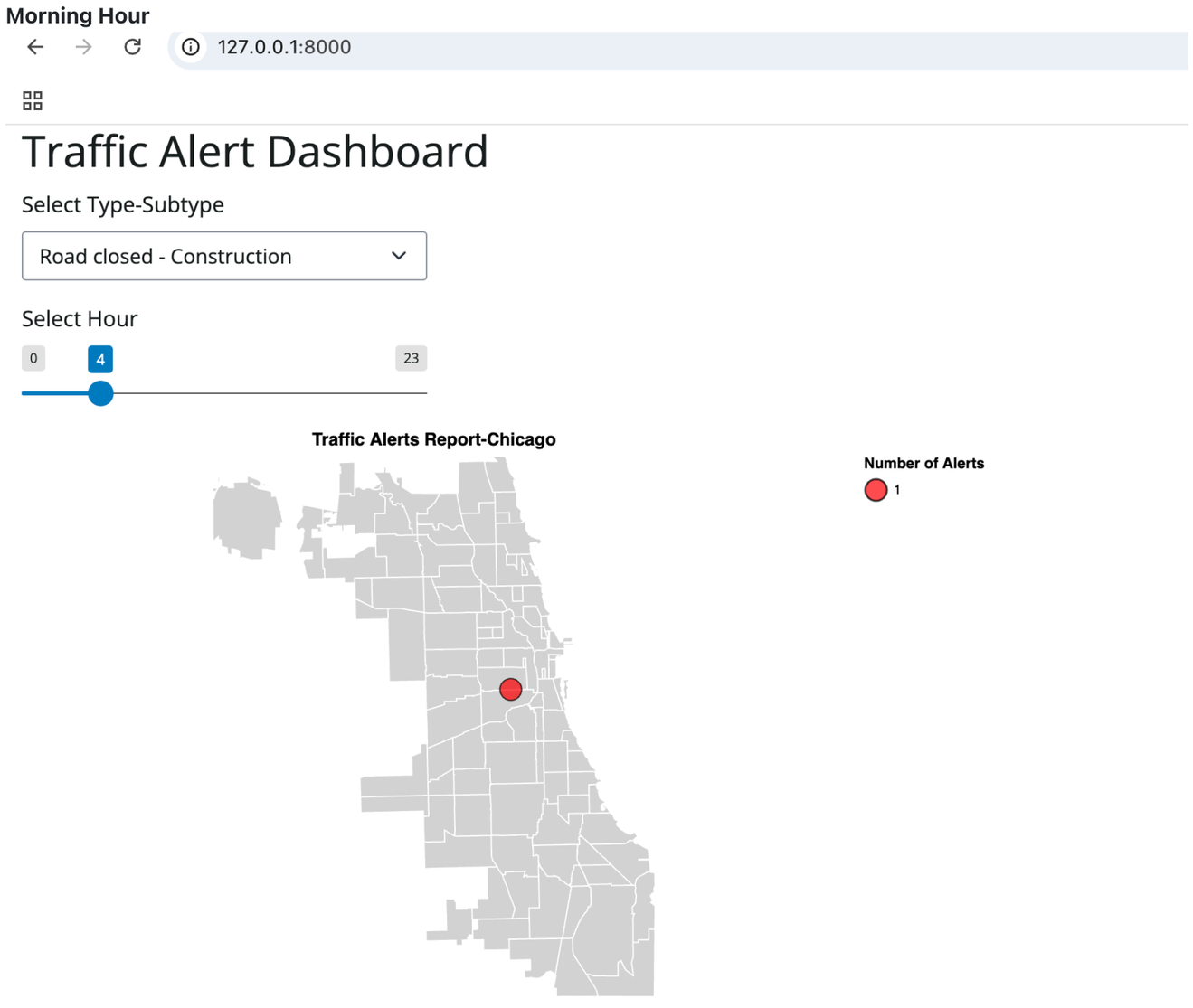
Select Hour

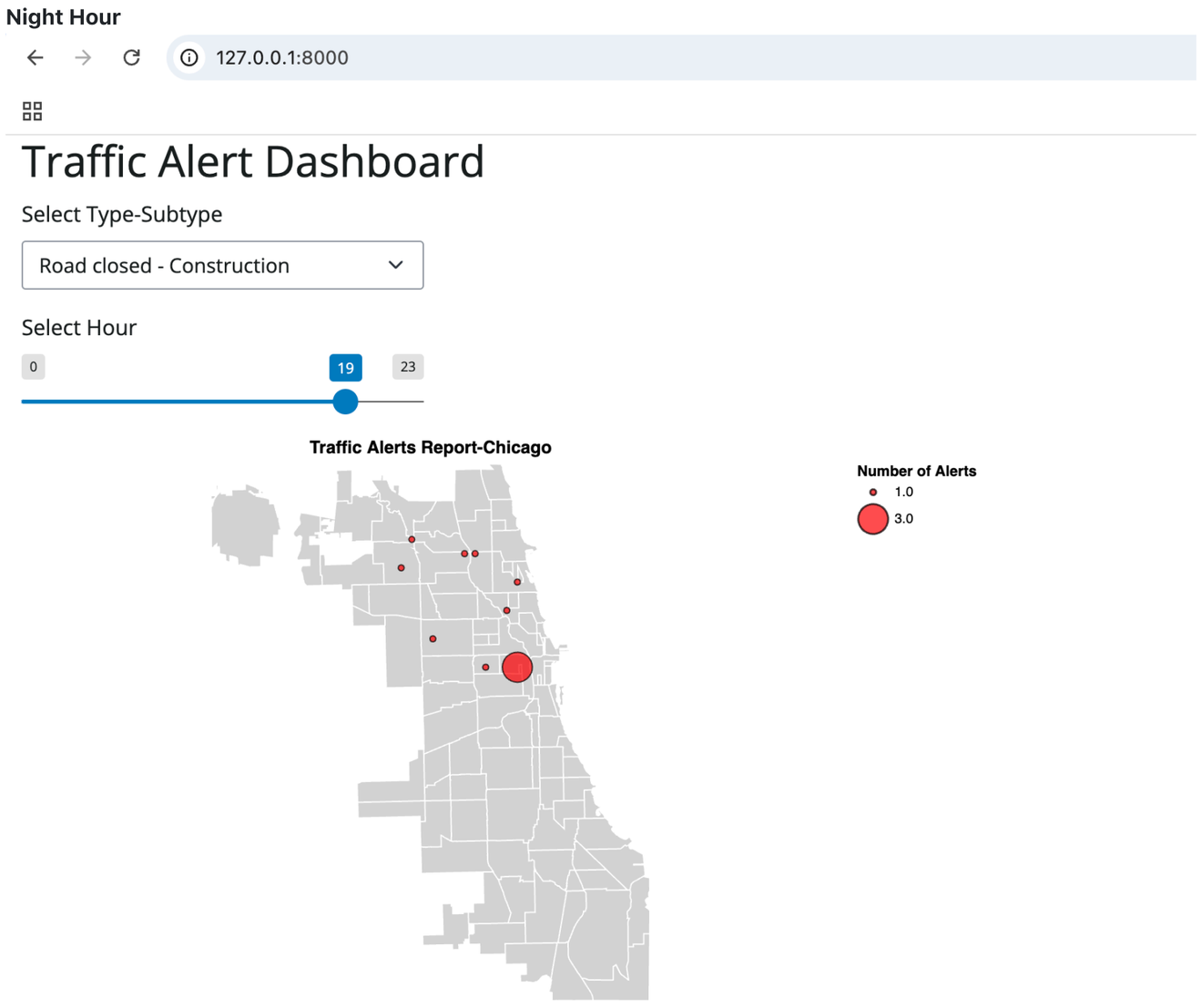
0 13 23




20:00
127.0.0.1:8000
Traffic Alert Dashboard
Select Type-Subtype
Jam - Heavy Traffic
Select Hour
0
20
23
Traffic Alerts Report-Chicago
Number of Alerts
201
209
224
245
249
258
308
437
504
518

c.


Morning Hour
127.0.0.1:8000
Traffic Alert Dashboard
Select Type-Subtype
Road closed - Construction
Select Hour
0
4
23
Traffic Alerts Report-Chicago
Number of Alerts
1



Explanation

Based on the comparison, we can see less road construction is done less during morning hours. However, we can see more road construction is done more during night hours.

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

1.

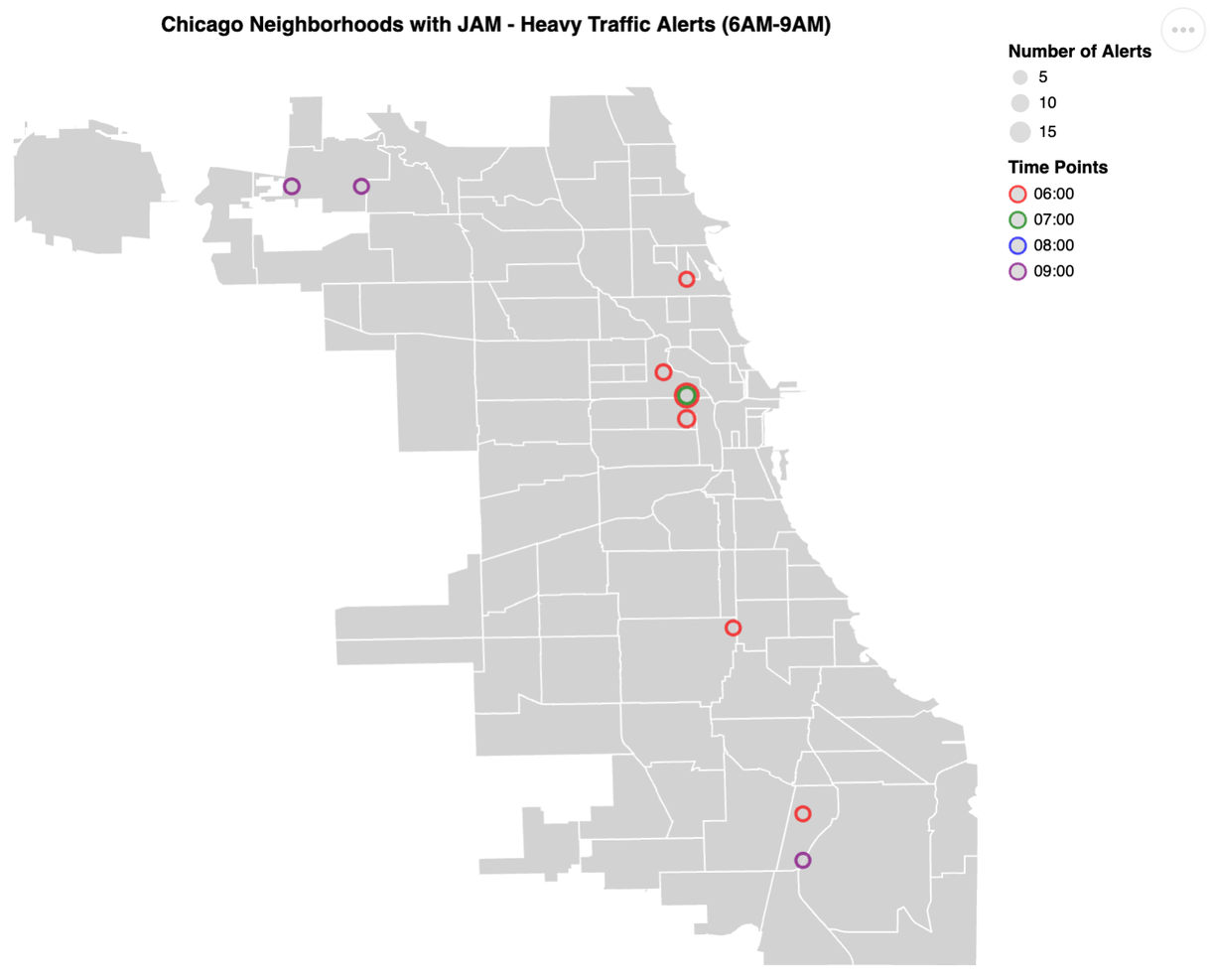
a.

Yes, it would be a good idea to collapse the dataset by range of hours because it would show users which specific time periods have more alerts. Moreover, some alerts, such as traffic jams or road construction, can persist for a range of times. Therefore, it would be more beneficial to track the number of alerts over a range of time.

b.

```
#Read the data in the new folder
top_alerts_map_byhour_sliderrange = pd.read_csv('/Users/zhengcui/Desktop/python
        2/student30538/problem_sets/ps6/top_alerts_map_byhour_sliderrange/top_alerts_map_byhour.csv')

#Find the data between 6AM and 9AM
data_6AM9AM = top_alerts_map_byhour_sliderrange[
    (top_alerts_map_byhour_sliderrange['updated_type'] == 'JAM') &
    (top_alerts_map_byhour_sliderrange['updated_subtype'] == 'Heavy Traffic') &
    (top_alerts_map_byhour_sliderrange['hour'].between('06:00', '09:00'))
]



#find top10 location with highest counts between 6AM and 9AM
top_10_6AM9AM = data_6AM9AM.nlargest(10, 'count')

#classify time period into different colors
color_scale_6AM9AM = alt.Scale(
    domain=['06:00', '07:00', '08:00','09:00'],
    range=['red', 'green', 'blue','purple']
)

#create a digram for the time period between 6Am and 9AM
geo_map_6AM9AM = alt.Chart(alt.Data(values=chicago_geojson['features'])).mark_geoshape(
    fill='lightgray',
    stroke='white'
).encode(
    tooltip='properties.pri_neigh:N'
).properties(
    width=600,
    height=600,
    title='Chicago Neighborhoods with JAM - Heavy Traffic Alerts (6AM-9AM)'
).project('equirectangular')

scatter_chart_6AM9AM = alt.Chart(top_10_6AM9AM).mark_circle(
    fill='lightred',
    strokeWidth=2,
    stroke= 'black'
).encode(
    longitude='longitude:Q',
    latitude='latitude:Q',
    size=alt.Size(
        'count:Q',
        title='Number of Alerts',
        scale=alt.Scale(domain=[1, top_10_6AM9AM['count'].max()], range=[50, 200]),
        legend=alt.Legend(title="Number of Alerts")
    ),
    stroke=alt.Color(
        'hour:N',
        scale=color_scale_6AM9AM,
        title='Time Points',
        legend=alt.Legend(title="Time Points")
    ),
```

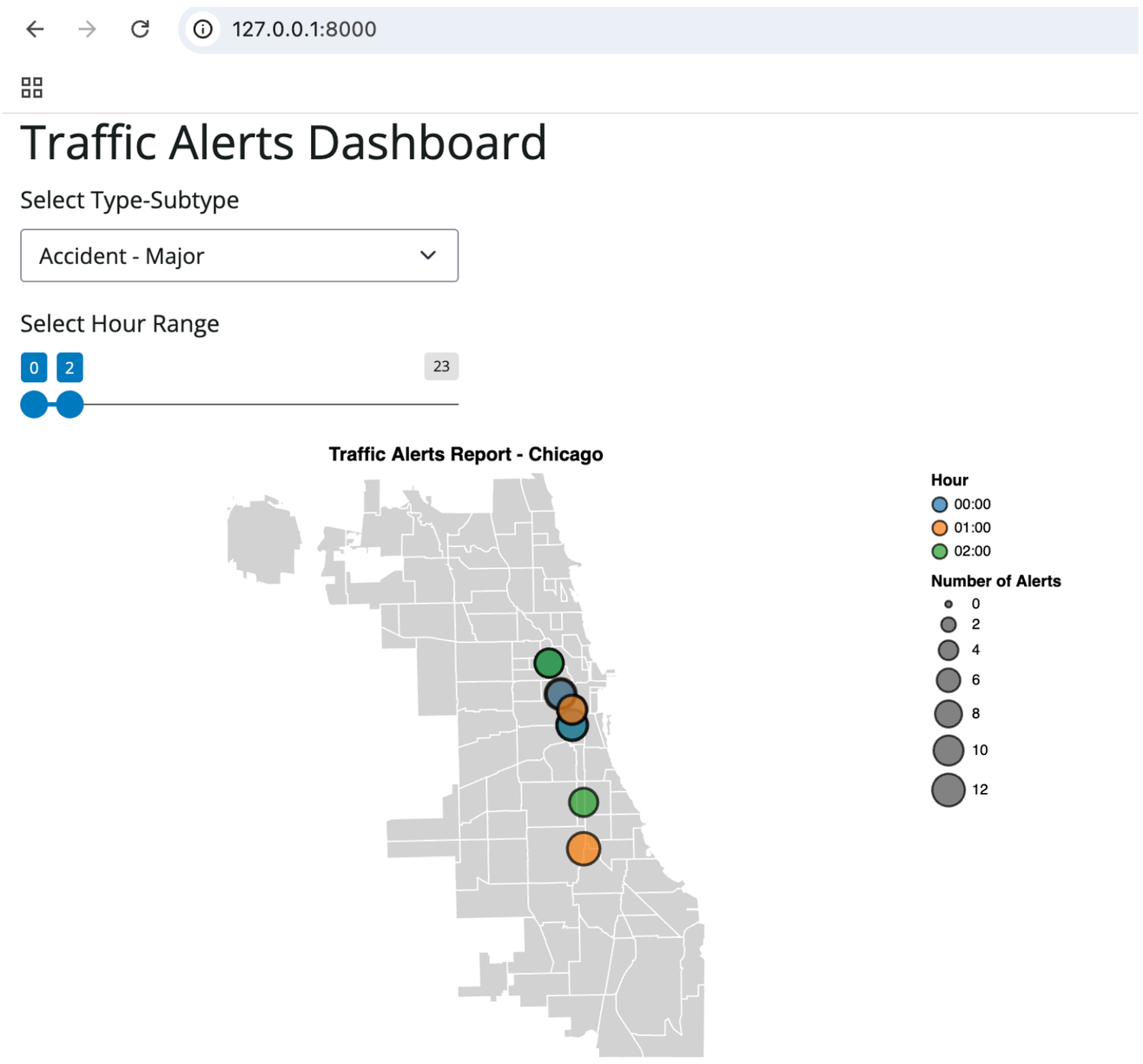
```
    tooltip=['longitude', 'latitude', 'count', 'hour']
).properties(
    width=600,
    height=600
)


geo_map_6AM9AM + scatter_chart_6AM9AM
```



2.

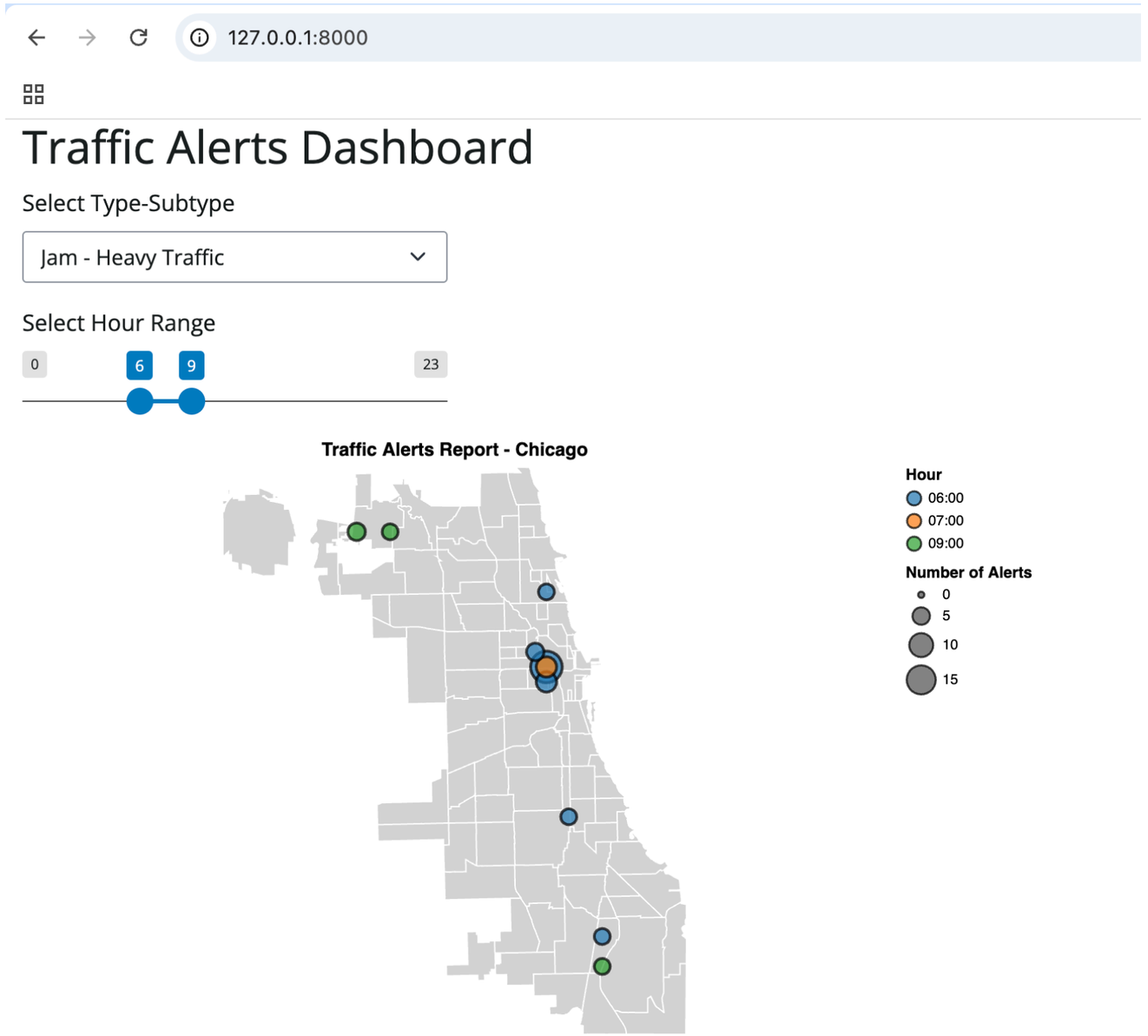
a.

Here is a screenshot of the UI and the plot



b.

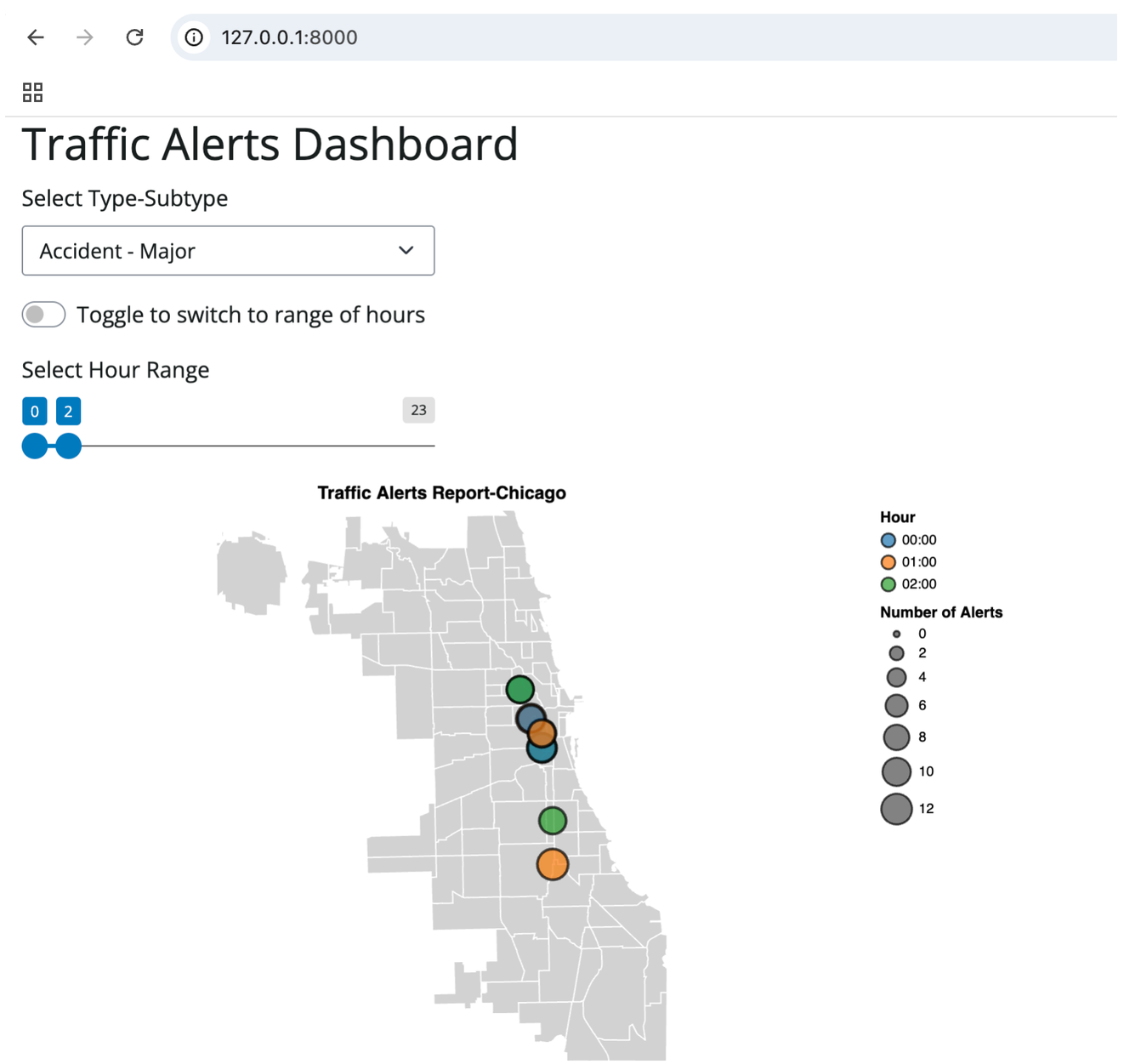
Here is a screenshot of the “Jam - Heavy Traffic” plot



3.

a.

Here is a screenshot of the 'switch' button

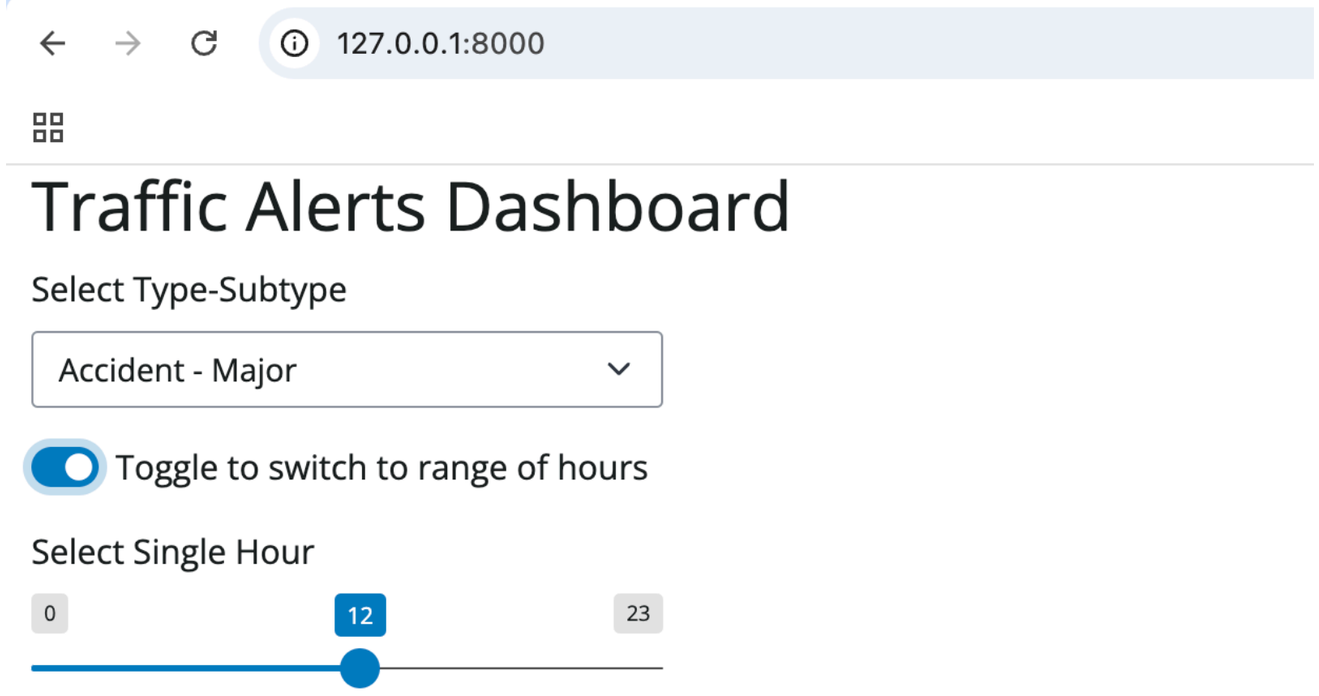


Explanation

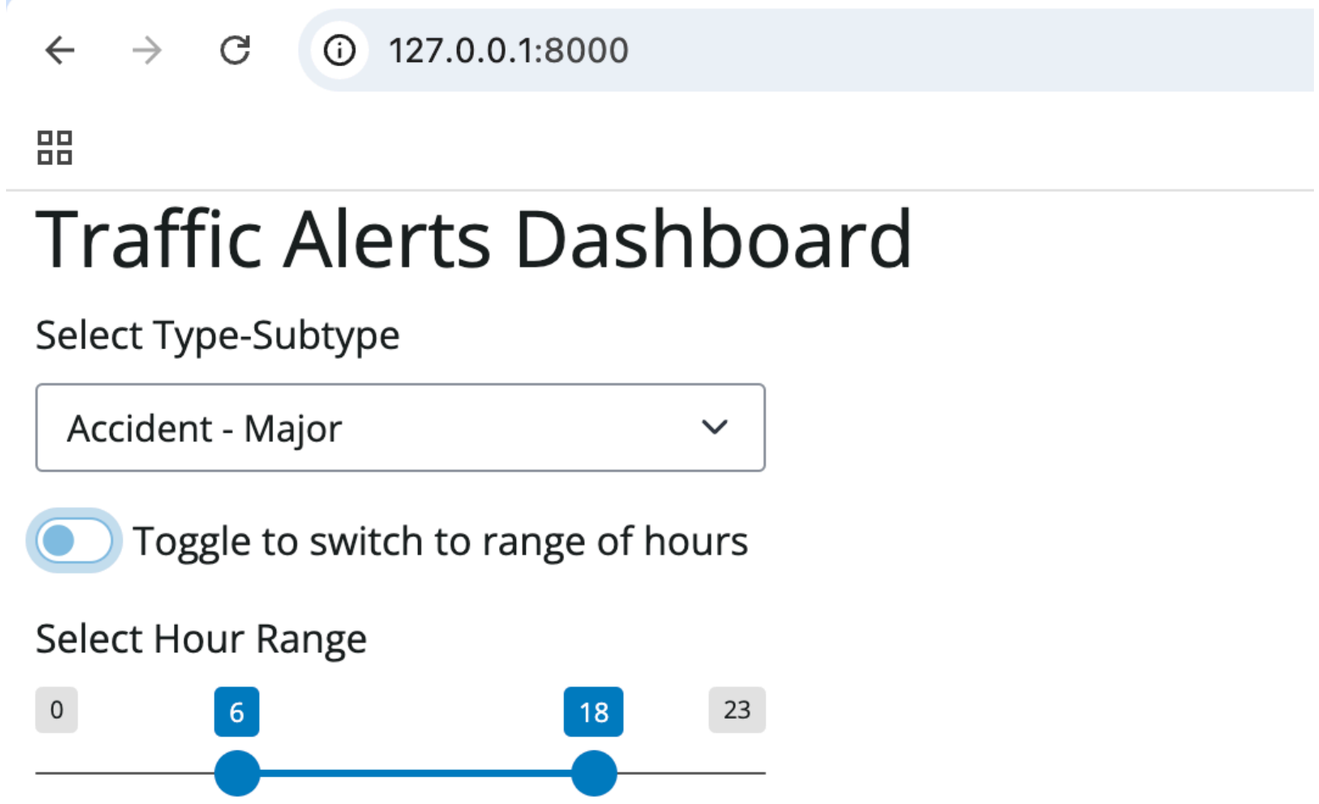
There are two possible values for the switch button. The first one is 'True' which means that the switch is toggled to the 'on' position. The second one is 'False' which means that the switch is toggled to the 'off' position.

b.

Here is a screenshot when the when the switch button is toggled, the slider for a single hour is shown

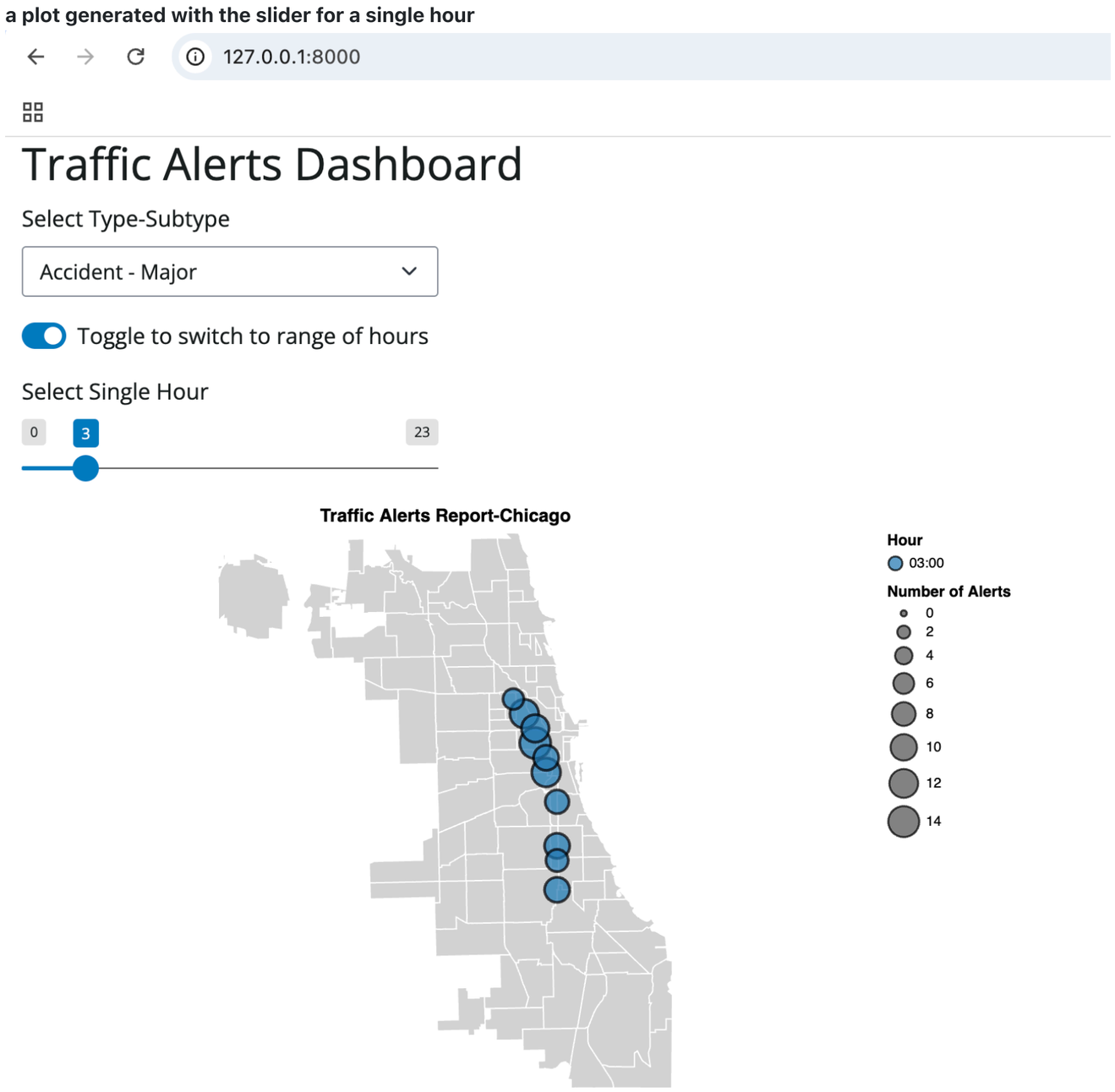


Here is a screenshot when the when the switch button is not toggled, the slider for a range of hours is shown

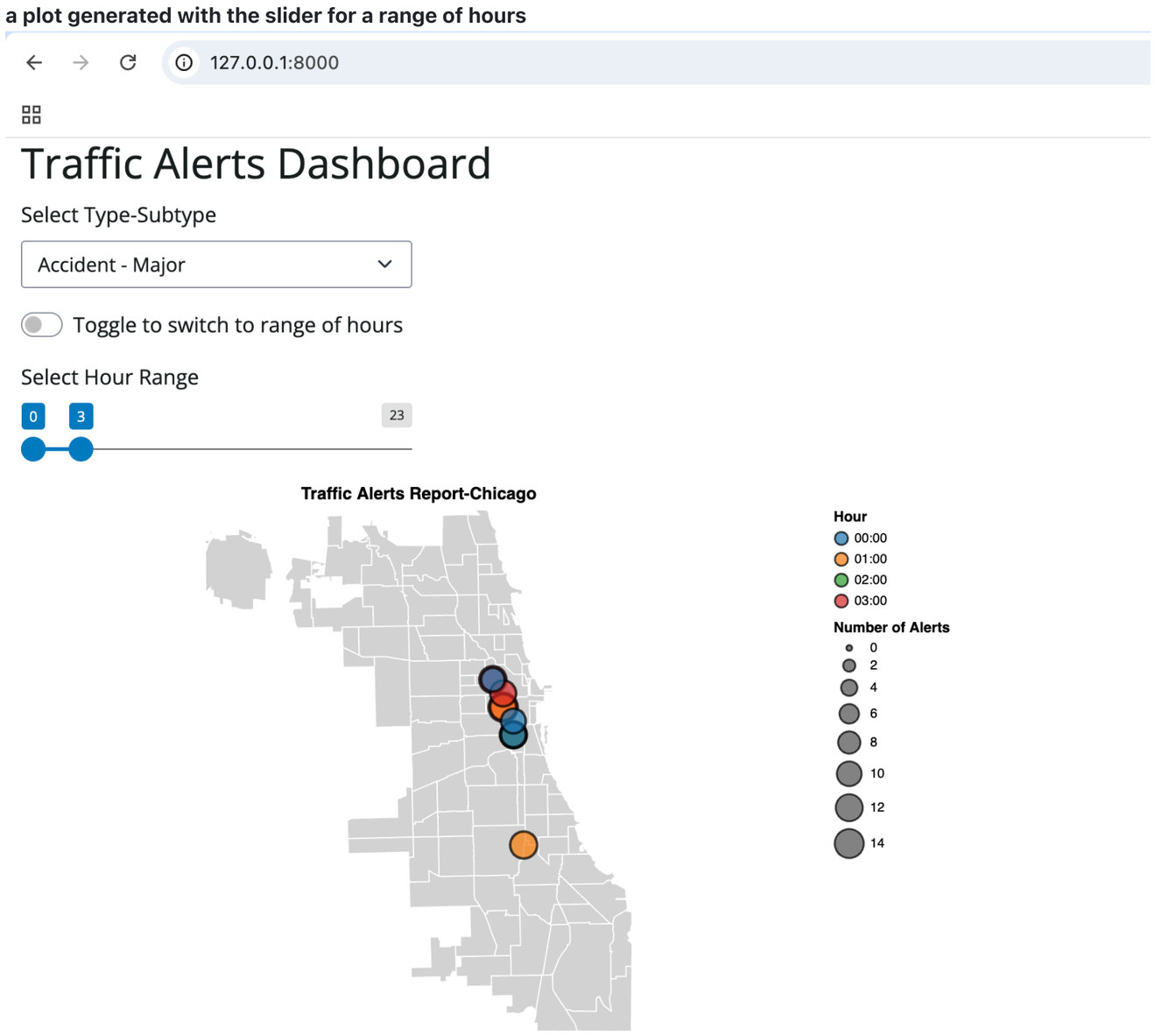


c.

a plot generated with the slider for a single hour



a plot generated with the slider for a range of hours



d.

Organizing data

We firstly need to specify the time range such as Morning:1:00AM to 12:00PM, Afternoon:13:00PM to 6:00PM, Night:7:00PM to 12:00AM. Then, we will use the time range to classify the data(put the data into different time range) such as 6:00AM data would be placed into 'Morning' group. After data classification, we need to use the data group to find top 10 locations with highest number of alerts.

Working in shiny

After classifying and filtering the data, we can add a multi-select dropdown menu to allow users to choose which time periods they are interested in viewing. This functionality will enable users to explore the data for specific periods, such as Morning, Afternoon, or Night.Additionally, the top 10 locations with the highest number of alerts can be displayed for multiple time periods simultaneously (e.g., Morning and Afternoon).

Ploting Diagram

We can use different colors to represent various time periods. For example, we can use red to display the top 10 data points with the highest number of alerts in the Morning, blue for the Afternoon, and purple for the Night. Additionally, we should include a legend named Time Period to indicate which color corresponds to each time period.To enhance the visualization, we can vary the size of the circles to represent the number of alerts—the higher the number of alerts, the larger the circle. Once we overlay this plot on a map of Chicago, we will have a diagram that effectively conveys the required information.