Problem Set 6 - Waze Shiny Dashboard

AUTHOR
Xiaotian Tang

PUBLISHED
November 23, 2024

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

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

Steps to submit (10 points on PS6)

1. "This submission is my work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: ** XT**
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **__** (2 point)
3. Late coins used this pset: **\0 ** Late coins left after submission: ** 3 **
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!

Background

Data Download and Exploration (20 points)

1.

```
# $ cd /Users/tang/Desktop/1Python II/ProblemSet6
# Extract zip
zip_path = 'waze_data.zip'
ext_path = 'waze_data'

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(ext_path)

# load the waze_data_sample.csv file into a dataframe waze_sample
smpl_path = "waze_data/waze_data_sample.csv"
waze_sample = pd.read_csv(smpl_path)


# Report Data Type
columns_to_ignore = ["ts", "geo", "geoWKT"] # ignore the three columns
waze_sample_selected = waze_sample.drop(columns=columns_to_ignore, errors="ignore")
print(waze_sample_selected.dtypes)
```

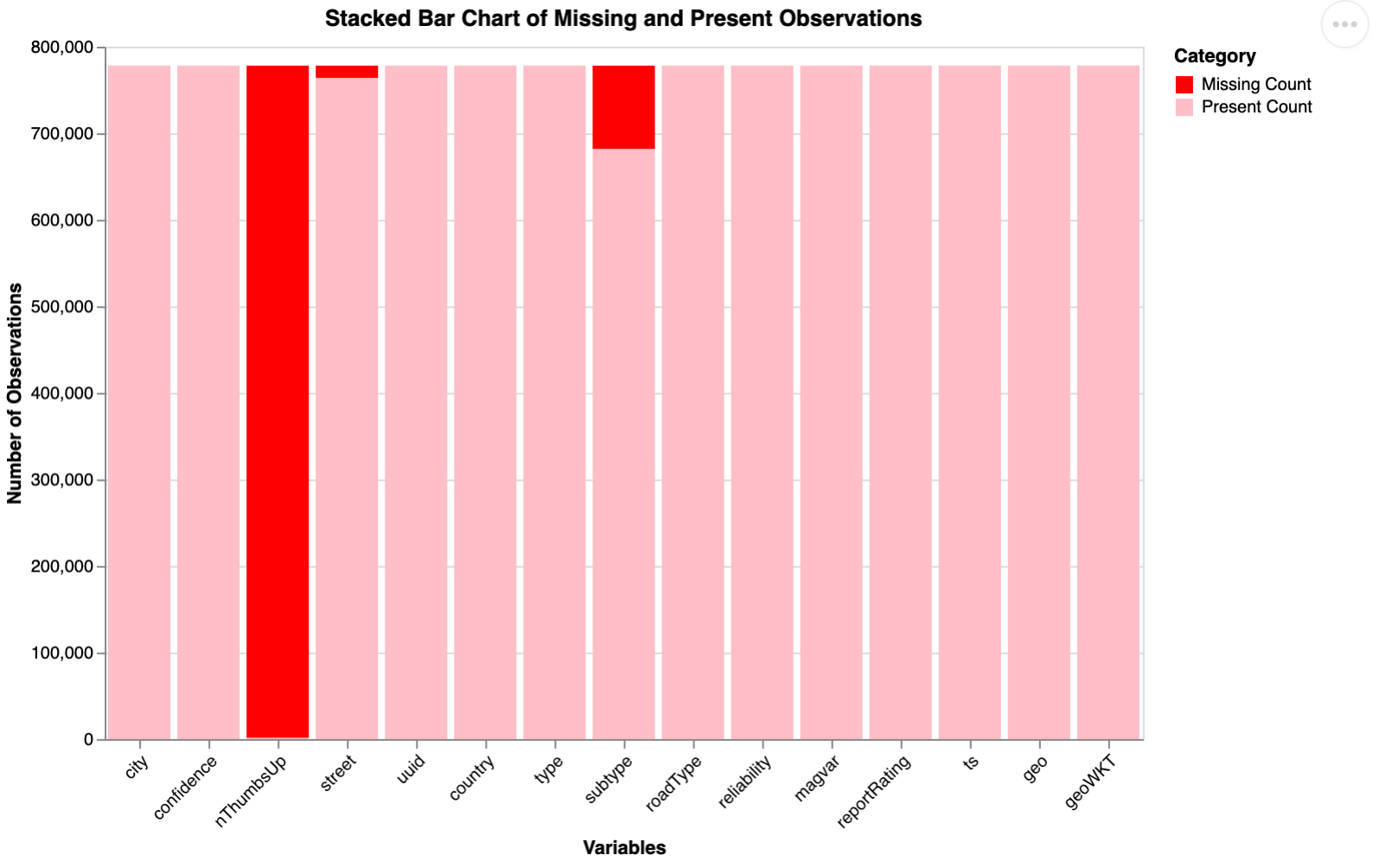
```
Unnamed: 0        int64
city             object
confidence        int64
nThumbsUp       float64
street           object
uuid             object
country          object
type             object
subtype          object
roadType          int64
reliability       int64
magvar            int64
reportRating      int64
dtype: object
```

Report: The data types is as follows:

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

2.

```
# load the waze file into the dataframe 'waze'
waze_path = "waze_data/waze_data.csv"
waze = pd.read_csv(waze_path)

# derive the number of missing rows
missing_info = pd.DataFrame({
    "Missing Count": waze.isnull().sum(),
    "Present Count": waze.notnull().sum()
}).reset_index(names='Variable')

# convert into long form
missing_long = pd.melt(
    missing_info,
    id_vars=["Variable"],
    value_vars=["Missing Count", "Present Count"],
    var_name="Category",
    value_name="Count"
)

# Use altair to draw the graph
alt.Chart(missing_long).mark_bar().encode(
    x=alt.X("Variable:N",
            title="Variables",
            sort=missing_info["Variable"].tolist(),
            axis=alt.Axis(labelAngle=-45)),
    y=alt.Y("Count:Q",
             title="Number of Observations"),
    color=alt.Color("Category:N",
                    title="Category",
                    scale=alt.Scale(domain=["Missing Count","Present Count"],
                    range=["red", "pink"]))
).properties(
    title="Stacked Bar Chart of Missing and Present Observations",
    width=600,
    height=400
)
```



Answer: Variable *nThumbsUp*, *street*, and *subtype* have missing valuess; and variable *nThumbsUp* has the largest share of missing observations.

3.

```
# print the unique values
unique_types = waze['type'].unique()
unique_subtypes = waze['subtype'].unique()
print(f'The unique types are {unique_types}')
print(f'The unique subtypes are {unique_subtypes}')

# create df 'waze_type' that only contains types
waze_type = waze[['type','subtype']]
# fill each NA with 'Unclassified'
waze_type['subtype'] = waze_type['subtype'].fillna('Unclassified')
has_unclassified = waze_type.groupby('type')['subtype'].apply(
  lambda x: 'Unclassified' in x.values)
print(f'There are {len(has_unclassified)} types have a subtype that is NA.')
```

```
The unique types are ['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']
The unique subtypes are [nan 'ACCIDENT_MAJOR' 'ACCIDENT_MINOR' 'HAZARD_ON_ROAD'
 'HAZARD_ON_ROAD_CAR_STOPPED' 'HAZARD_ON_ROAD_CONSTRUCTION'
 'HAZARD_ON_ROAD_EMERGENCY_VEHICLE' 'HAZARD_ON_ROAD_ICE'
 'HAZARD_ON_ROAD_OBJECT' 'HAZARD_ON_ROAD_POT_HOLE'
 'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT' 'HAZARD_ON_SHOULDER'
 'HAZARD_ON_SHOULDER_CAR_STOPPED' 'HAZARD_WEATHER' 'HAZARD_WEATHER_FLOOD'
 'JAM_HEAVY_TRAFFIC' 'JAM_MODERATE_TRAFFIC' 'JAM_STAND_STILL_TRAFFIC'
 'ROAD_CLOSED_EVENT' 'HAZARD_ON_ROAD_LANE_CLOSED' 'HAZARD_WEATHER_FOG'
 'ROAD_CLOSED_CONSTRUCTION' 'HAZARD_ON_ROAD_ROAD_KILL'
 'HAZARD_ON_SHOULDER_ANIMALS' 'HAZARD_ON_SHOULDER_MISSING_SIGN'
 'JAM_LIGHT_TRAFFIC' 'HAZARD_WEATHER_HEAVY_SNOW' 'ROAD_CLOSED_HAZARD'
 'HAZARD_WEATHER_HAIL']
There are 4 types have a subtype that is NA.
```

Yes,I can identify which type has subtypes that have enough information to consider that they could have sub-subtypes. This type is **Hazard**.

Write out the bulleted:

- Accident
  - Major
  - Minor
- Hazard
  - On Road
    - Car Stopped
    - Construction
    - Emergency Vehicle
    - Ice
    - Object
    - Pot Hole
    - Traffic Light Fault
    - Lane Closed
    - Road Kill

    - On Shoulder
        - Car Stopped
        - Animals
        - Missing Sign
    - Weather
        - Flood
        - Fog
        - Heavy Snow
        - Hail
- Jam
    - Heacy Traffic
    - Moderate Traffic
    - Stand Still Traffic
    - Light Traffic
- Road Closed
    - Event
    - Construction
    - Hazard

I think we should keep the NA subtypes. Because these subtypes may contain events unable to be classified, but still meaningful for our research.

4.

1.

```
# define a new df 'crosswalk'
crosswalk = pd.DataFrame(
  columns=['type',
           'subtype',
           'updated_type',
           'updated_subtype',
           'updated_subsubtype'])
# first two columns from the original dataset
crosswalk[['type', 'subtype']] = waze[['type', 'subtype']]
```

2.

```
# create a temporary cloumn 'combined_type'
crosswalk['subtype'] = crosswalk['subtype'].fillna('Unclassified')
crosswalk['combined_type'] = crosswalk["type"] + '_' + crosswalk['subtype']
crosswalk = crosswalk.drop_duplicates(subset=['combined_type']).reset_index(drop=True)

### First, deal with the duplicated type in some rows
# create a replacement dictionary
replacement_dict ={
  'HAZARD_HAZARD'           : 'HAZARD',
  'JAM_JAM'                 : 'JAM',
  'ROAD_CLOSED_ROAD_CLOSED' : 'ROAD_CLOSED',
  'ACCIDENT_ACCIDENT'       :'ACCIDENT'
}

# create a function to apply replacement
def adjustment_function(subtype, adjustment_dict):
    for key, value in adjustment_dict.items():
        if key in subtype:
            subtype = subtype.replace(key, value)
    return subtype

# use the function
crosswalk['combined_type'] = crosswalk['combined_type'].apply(
  lambda x: adjustment_function(x, replacement_dict)
  )

### Second, same logic, based on the bulleted, make some adjustments
adjustment_dict = {
  'ON_ROAD'             : 'ON ROAD',
  'ON_SHOULDER'         : 'ON SHOULDER',
  'CAR_STOPPED'         : 'CAR STOPPED',
  'EMERGENCY_VEHICLE'   : 'EMERGENCY VEHICLE',
  'POT_HOLE'            : 'POT HOLE',
  'TRAFFIC_LIGHT_FAULT' : 'TRAFFIC LIGHT FAULT',
  'LANE_CLOSED'         : 'LANE CLOSED',
  'ROAD_KILL'           : 'ROAD KILL',
  'CAR_STOPPED'         : 'CAR STOPPED',
  'MISSING_SIGN'        : 'MISSING SIGN',
  'HEAVY_SNOW'          : 'HEAVY SNOW',
  'HEAVY_TRAFFIC'       : 'HEAVY TRAFFIC',
```

```
  'MODERATE_TRAFFIC'    : 'MODERATE TRAFFIC',
  'STAND_STILL_TRAFFIC' : 'STAND STILL TRAFFIC',
  'LIGHT_TRAFFIC'       : 'LIGHT TRAFFIC',
  'ROAD_CLOSED'         : 'ROAD CLOSED'
}

# use the function
crosswalk['combined_type'] = crosswalk['combined_type'].apply(
  lambda x: adjustment_function(x, adjustment_dict)
  )

### Third, fill in the three columns
# define a function to make it possible
def split_combined_type(combined_type):
    parts = combined_type.split('_')  # split based on the underscore
    updated_type = parts[0]
    updated_subtype = parts[1]
    updated_subsubtype = parts[2] if len(parts) > 2 else 'Unclassified'
    return updated_type, updated_subtype, updated_subsubtype

# use the function
crosswalk[
  ['updated_type','updated_subtype','updated_subsubtype']
  ] = crosswalk['combined_type'].apply(
    lambda x: pd.Series(split_combined_type(x))
    )

### Drop the temoporary column, and title the content in each column
crosswalk = crosswalk.drop(columns=['combined_type'])
crosswalk[
  ['updated_type', 'updated_subtype', 'updated_subsubtype']
  ] = crosswalk[
    ['updated_type', 'updated_subtype', 'updated_subsubtype']
    ].applymap(lambda x: x.title())
```

3.

```
# first, manipulate waze
waze['subtype'] = waze['subtype'].fillna('Unclassified')

# then, use left merge
waze = waze.merge(crosswalk,
                  on = ['type','subtype'],
                  how = 'left'
)

# Calculate rows for Accident - Unclassified
accident_unclassified = waze[
        (waze['updated_type'] == 'Accident') & \
        (waze['updated_subtype'] == 'Unclassified')]
print(f'There are {accident_unclassified.shape[0]} rows for Accident - Unclassified')
```

```
There are 24359 rows for Accident - Unclassified
```

4.

```
# in the crosswork
crosswork_unique_cmbn = crosswalk[['type', 'subtype']
                        ].drop_duplicates().reset_index(drop=True)
# in the newly merged waze
waze_unique_cmbn = waze[['type', 'subtype']
                        ].drop_duplicates().reset_index(drop=True)
# compare if it's equal
is_equal = crosswork_unique_cmbn.equals(waze_unique_cmbn)
print(f"Have the same value: {is_equal}")
```

```
Have the same value: True
```

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

1.

a.

```
# use regex to capture longitude and latitude from geo
waze[['longitude', 'latitude']] = waze['geo'].str.extract(
```

```
  r'POINT\((-?\d+\.\d+)\s(-?\d+\.\d+)\)'
  )
```

b.

```
# round to two decimal places
waze['latitude_bin'] = waze['latitude'].astype(float).round(2)
waze['longitude_bin'] = waze['longitude'].astype(float).round(2)

# groupby (latitude_bin, longitude_bin) and count
bin_counts = waze.groupby(
    ['latitude_bin', 'longitude_bin']
    ).size().reset_index(name='count')

# find out the max count bin
max_bin = bin_counts.loc[bin_counts['count'].idxmax()]
print(f'When latitude = {max_bin.iloc[0]}, altitude = {max_bin.iloc[1]}, \
the combination has the greatest number of observations: {max_bin.iloc[2]}.')
```

```
When latitude = 41.88, altitude = -87.65, the combination has the greatest number of observations:
21325.0.
```

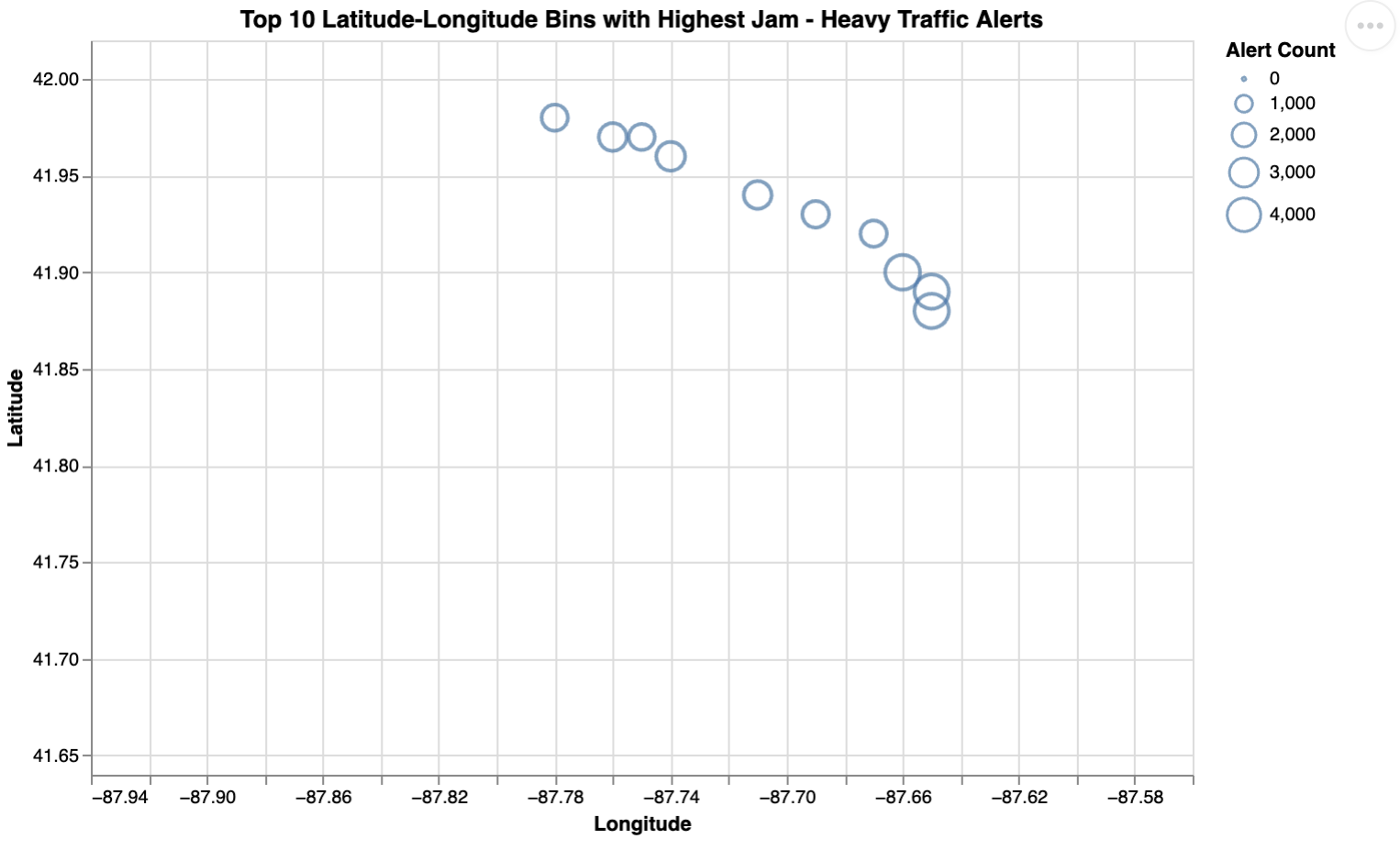
c.

```
# collapse the data
top_alerts_map = waze[['latitude_bin','longitude_bin','updated_type','updated_subtype']]

# aggregate to the level needed
top_alerts_map = top_alerts_map.groupby(['latitude_bin', 'longitude_bin', \
'updated_type','updated_subtype']).size().reset_index(name = 'count')

# save in the appointed folder
top_alerts_map.to_csv('top_alerts_map/top_alerts_map.csv', index=False)

print(f'The level of aggragation in this case is 4')
print(f'the dataframe has {top_alerts_map.shape[0]} observations.')
```

```
The level of aggragation in this case is 4
the dataframe has 6675 observations.
```

2.

```
alt.Chart(top_alerts_map).mark_point().transform_filter(
        (alt.datum.updated_type == 'Jam') &
        (alt.datum.updated_subtype == 'Heavy Traffic')
    ).transform_window(
        rank='rank(count)',
        sort=[alt.SortField('count', order='descending')]
    ).transform_filter(
        alt.datum.rank <= 10
    ).encode(
        x=alt.X('longitude_bin:Q',
                scale=alt.Scale(domain=[-87.94, -87.56]),
                title='Longitude'),
        y=alt.Y('latitude_bin:Q',
                scale=alt.Scale(domain=[41.64, 42.02]),
                title='Latitude'),
        size=alt.Size('count:Q', title='Alert Count')
    ).properties(
        title='Top 10 Latitude-Longitude Bins with Highest Jam - Heavy Traffic Alerts',
        width=600,
        height=400)
```



3.

a.

```
# URL of the GeoJSON file
url = "https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON"

# Filepath to save the GeoJSON file
output_file = "Boundaries - Neighborhoods.geojson"

# Download the GeoJSON file
response = requests.get(url)

# Check if the request was successful
if response.status_code == 200:
    # Save the file locally
    with open(output_file, "wb") as file:
        file.write(response.content)
    print(f"GeoJSON file successfully downloaded and saved as {output_file}")
else:
    print(f"Failed to download the file. HTTP Status Code: {response.status_code}")
```

```
GeoJSON file successfully downloaded and saved as Boundaries - Neighborhoods.geojson
```

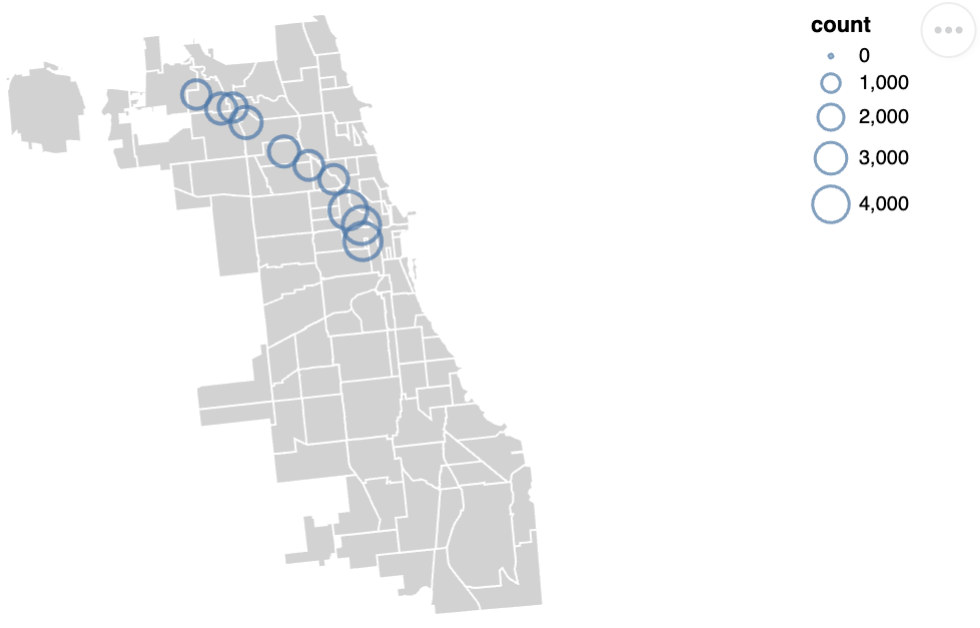
b.

```
file_path = "/Users/tang/Desktop/1Python II/ProblemSet6/Boundaries - Neighborhoods.geojson"

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

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

4.

```
background = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).project('albersUsa').properties(
    width=500,
    height=300
)

points = alt.Chart(top_alerts_map).mark_point().transform_filter(
        (alt.datum.updated_type == 'Jam') &
        (alt.datum.updated_subtype == 'Heavy Traffic')
    ).transform_window(
        rank='rank(count)',
        sort=[alt.SortField('count', order='descending')]
    ).transform_filter(
        alt.datum.rank <= 10
    ).encode(
```

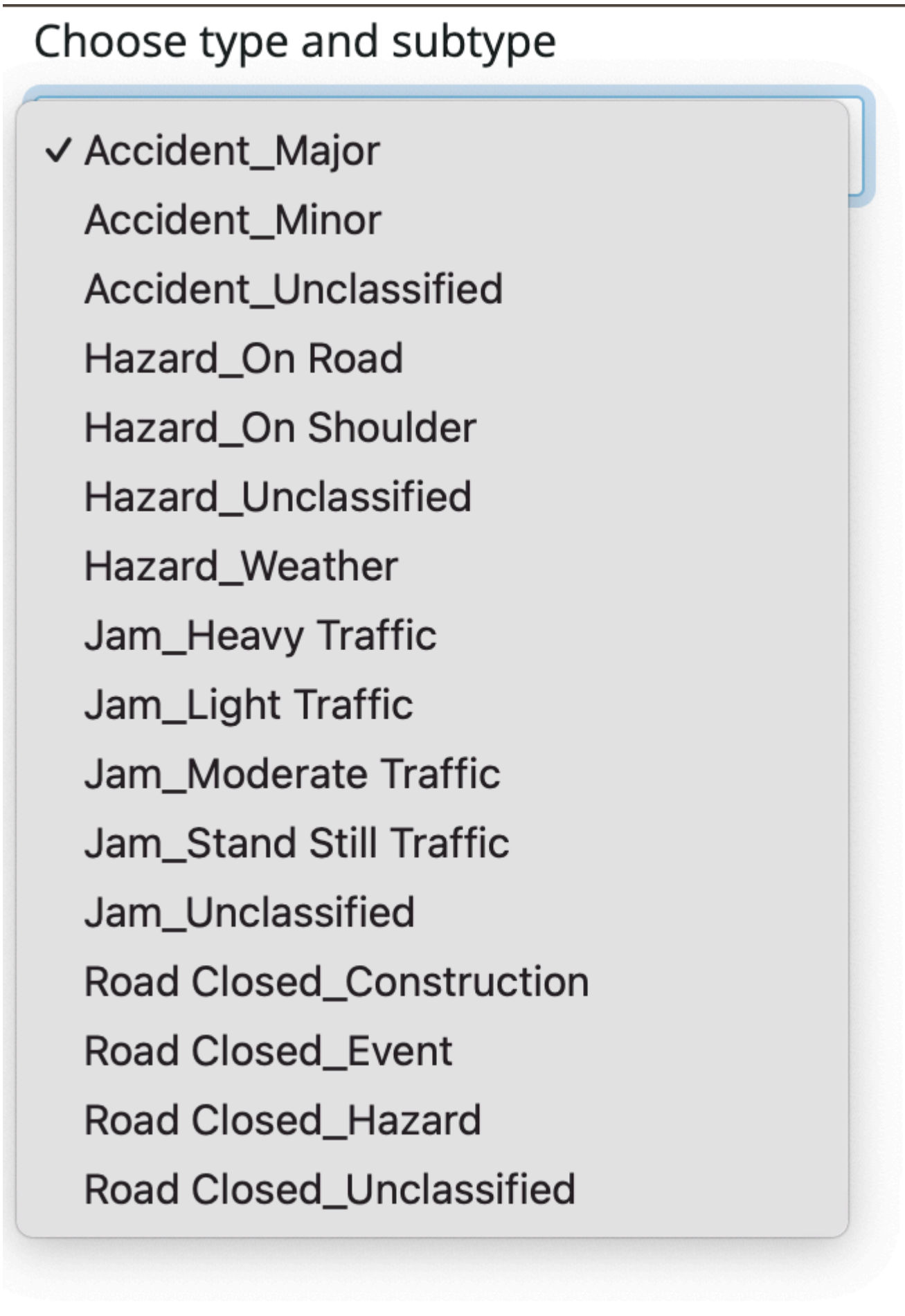
```
    longitude='longitude_bin:Q',
    latitude='latitude_bin:Q',
    size='count:Q'
)

background + points
```



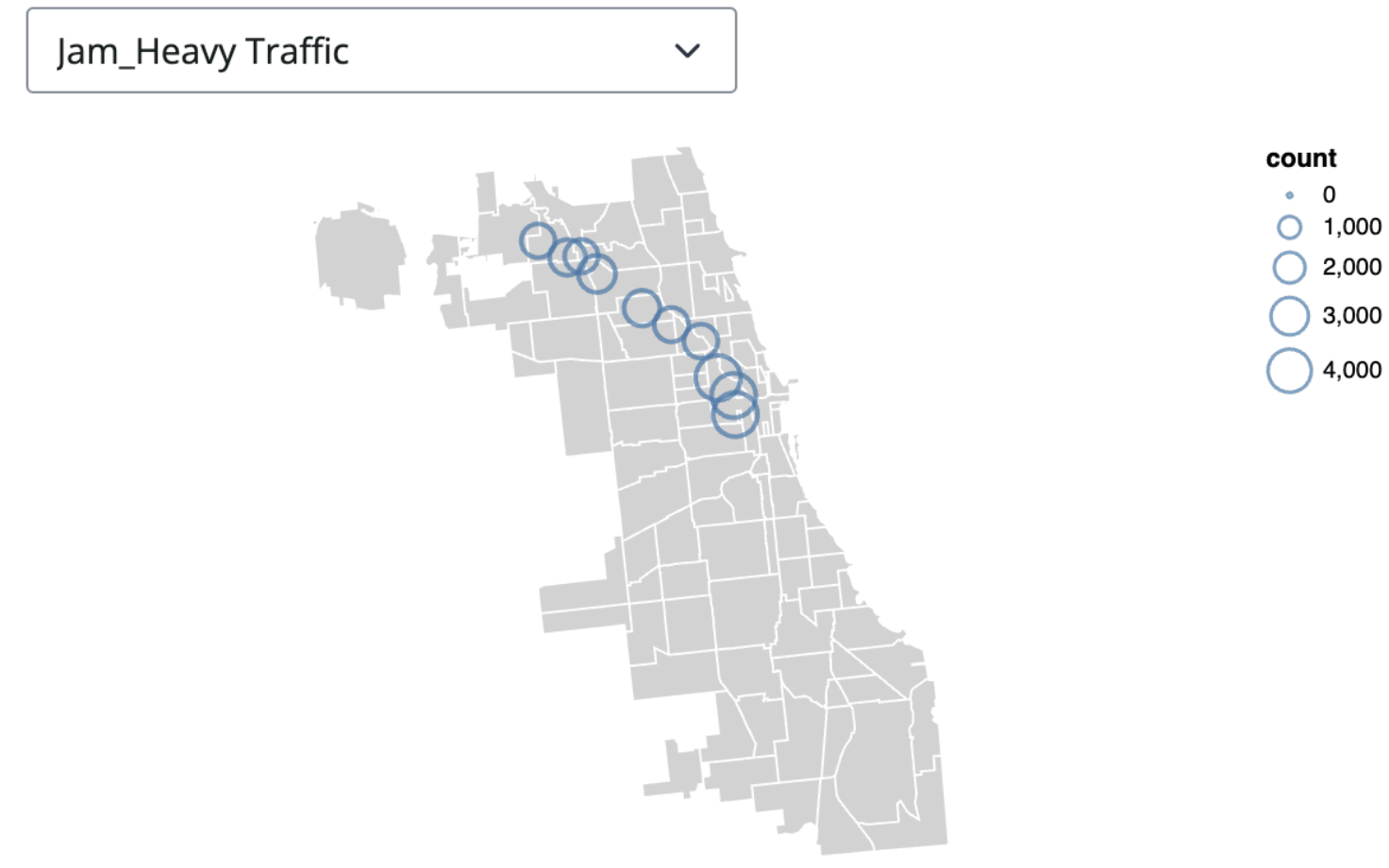
5.

a.

There are 16 unique combinations in my dropdown menu.

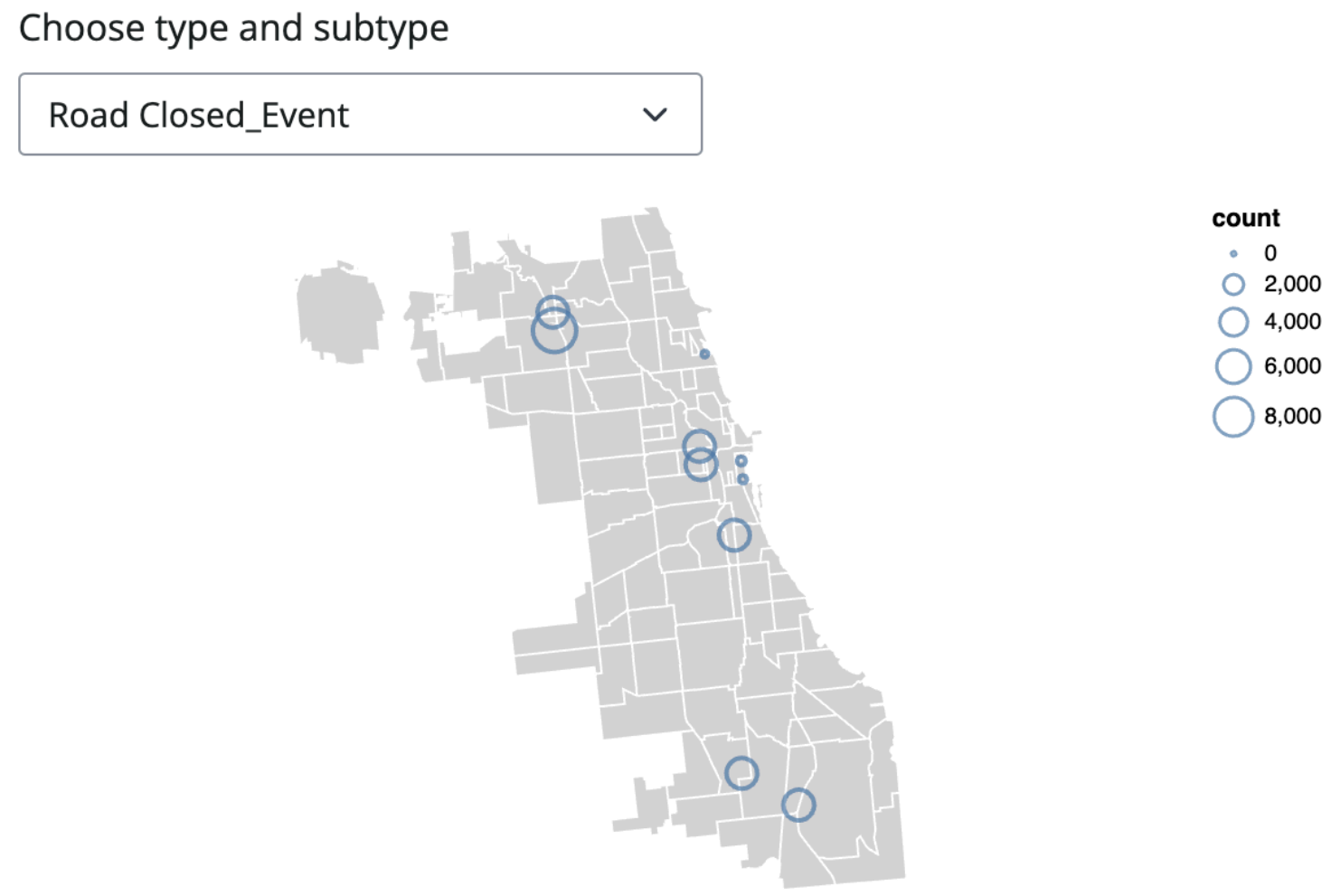


b.



c.

The most frequent road closure alerts due to events occur at longitude -87.75 and latitude 41.96.



d.

An Example of Question:

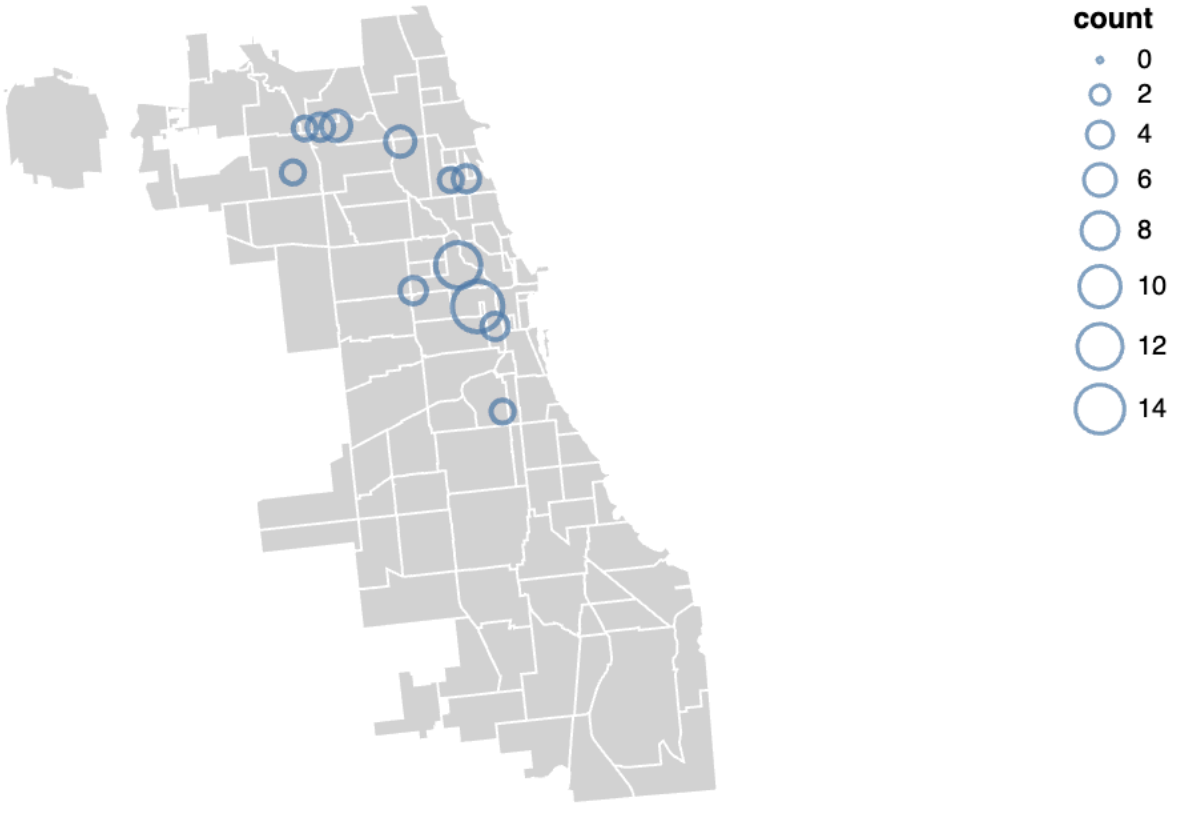
Where are alerts for road closures due to construction most common?

Answer:

The most frequent road closure alerts due to construction occur at longitude -87.65 and latitude 41.88.

Choose type and subtype

Road Closed_Construction



e.

We can add our updated_subsubtype column to the dashboard to make it more specific.

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

1.

a.

It will not be a good idea since the time is too detailed. We need first extract the date and hour out of it.

b.

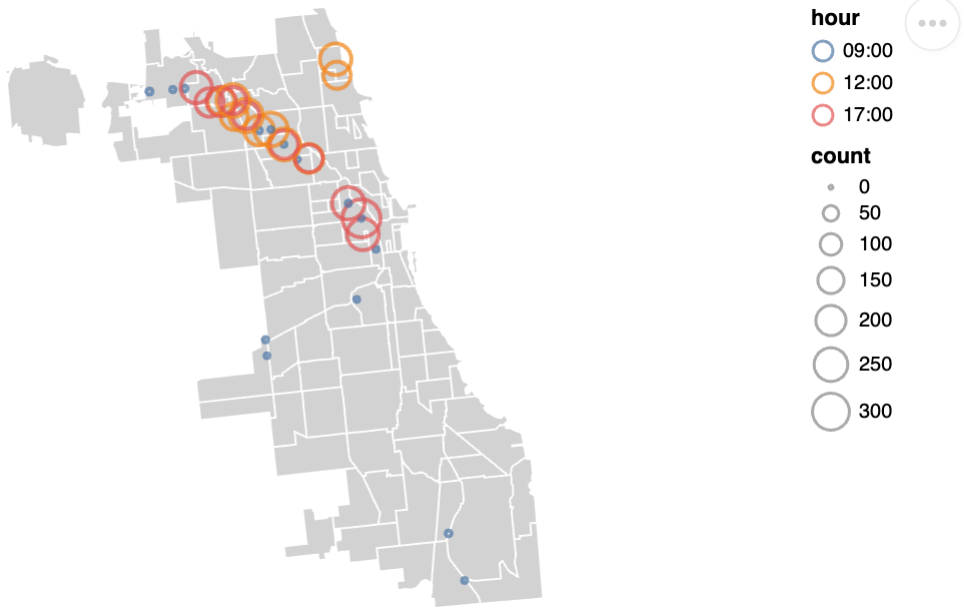
```
waze['ts'] = pd.to_datetime(waze['ts'])
waze['hour'] = waze['ts'].dt.strftime('%H:00')

# collapse the data
top_alerts_map_byhour = waze[
    ['latitude_bin','longitude_bin','updated_type','updated_subtype','hour']
    ]

# aggregate to the level needed
top_alerts_map_byhour = top_alerts_map_byhour.groupby(
    ['latitude_bin', 'longitude_bin','updated_type', \
    'updated_subtype','hour']).size().reset_index(name = 'count')

# save in the appointed folder
top_alerts_map_byhour.to_csv('top_alerts_map_byhour/top_alerts_map_byhour.csv', \
    index=False)

print('The level of aggragation in this case is 5')
print(f'the dataframe has {top_alerts_map_byhour.shape[0]} observations.')
```

```
The level of aggragation in this case is 5
the dataframe has 62825 observations.
```

c.

```
background = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).project('albersUsa').properties(
    width=500,
    height=300
)
```

```
pointcombo = alt.Chart(top_alerts_map_byhour
).mark_point().transform_filter(
    (alt.datum.updated_type == 'Jam') &
    (alt.datum.updated_subtype == 'Heavy Traffic') &
     ((alt.datum.hour == '09:00') | (
        alt.datum.hour == '12:00') | (
            alt.datum.hour == '17:00'))
).transform_window(
    groupby=['hour'],
    rank='rank(count)',
    sort=[alt.SortField('count', order='descending')]
).transform_filter(
    alt.datum.rank <= 10
).encode(
    longitude='longitude_bin:Q',
    latitude='latitude_bin:Q',
    size=alt.Size('count:Q'),
    color=alt.Color('hour:N'),
    tooltip=['longitude_bin', 'latitude_bin', 'count', 'hour']
)

background + pointcombo
```



2.

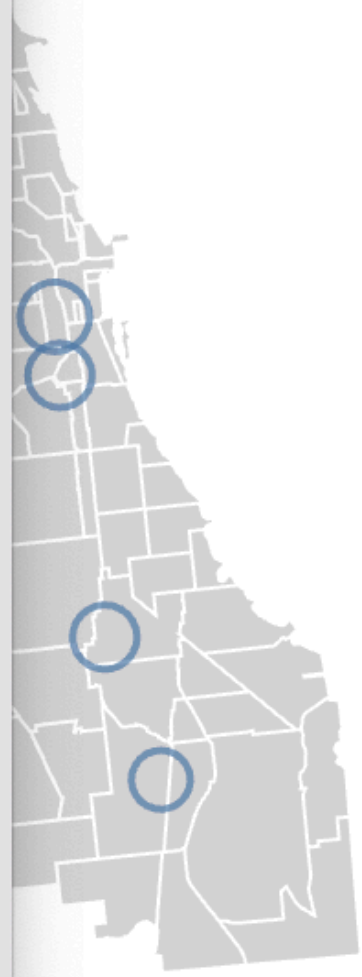
a.

Select hour

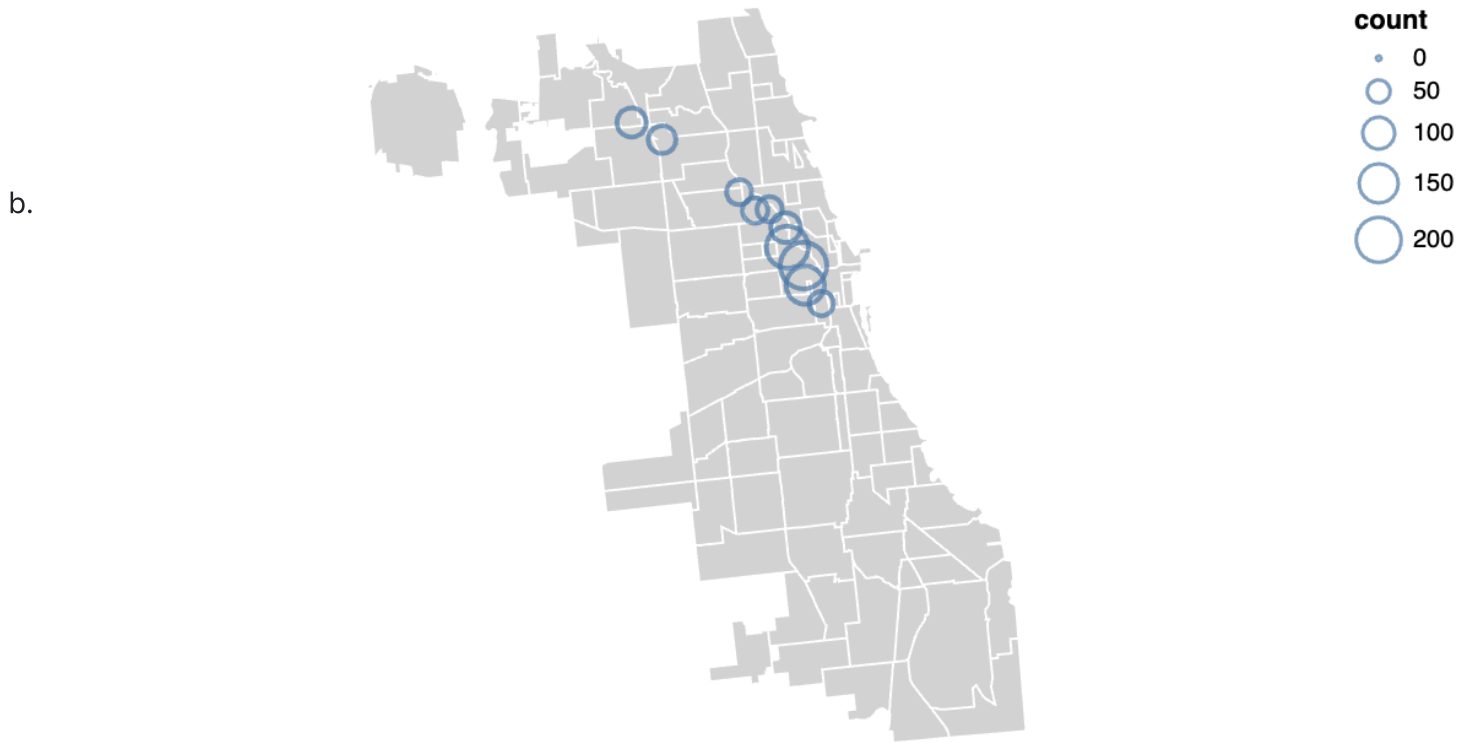
0 12 23

Choose type and subtype

- ✓ Accident_Major
- Accident_Minor
- Accident_Unclassified
- Hazard_On Road
- Hazard_On Shoulder
- Hazard_Unclassified
- Hazard_Weather
- Jam_Heavy Traffic
- Jam_Light Traffic
- Jam_Moderate Traffic
- Jam_Stand Still Traffic
- Jam_Unclassified
- Road Closed_Construction
- Road Closed_Event
- Road Closed_Hazard
- Road Closed_Unclassified



b.



c.

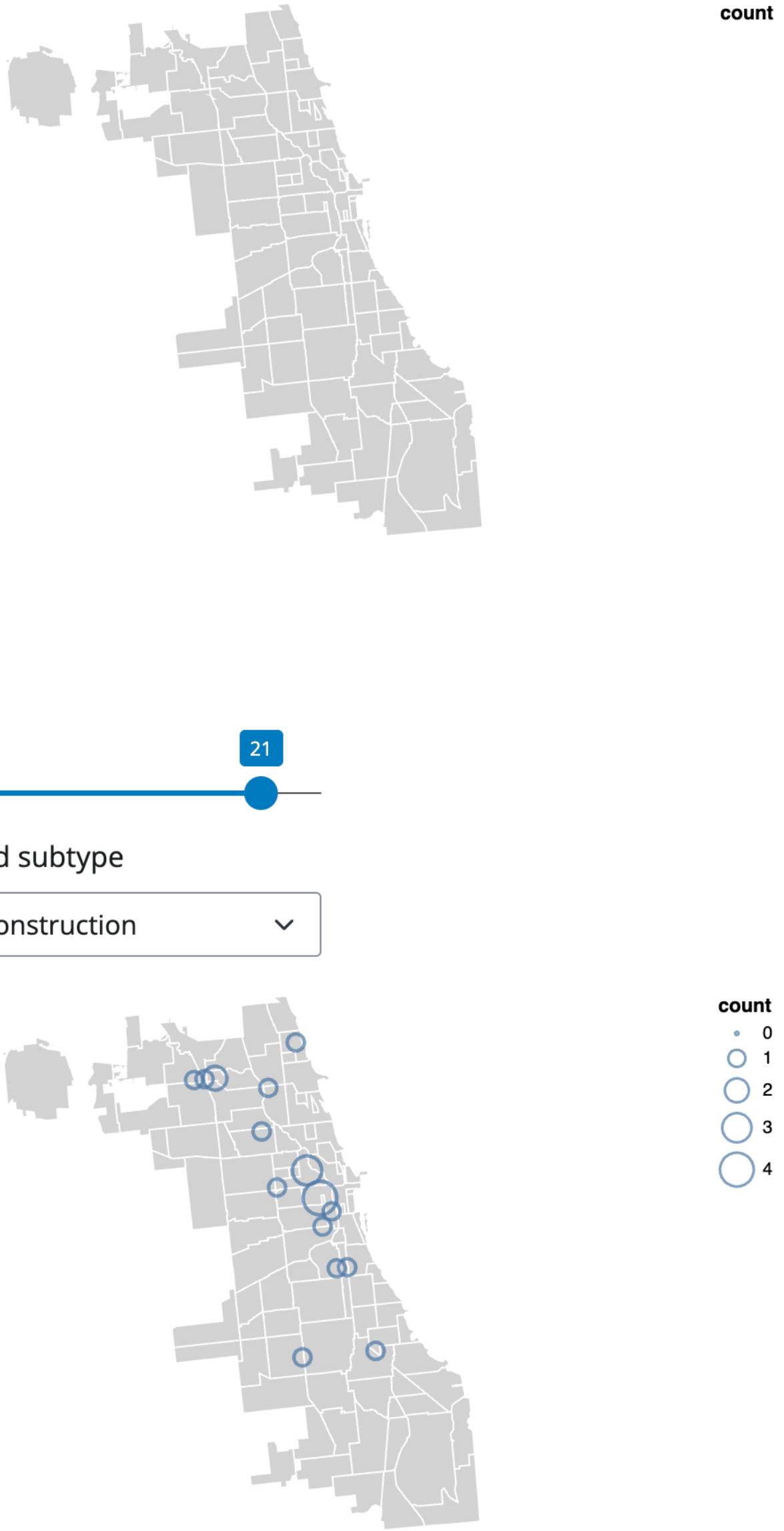
It seems that road construction is done more during night hours.

Select hour

0 7 23

Choose type and subtype

Road Closed_Construction



Select hour

0 21

Choose type and subtype

Road Closed_Construction

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

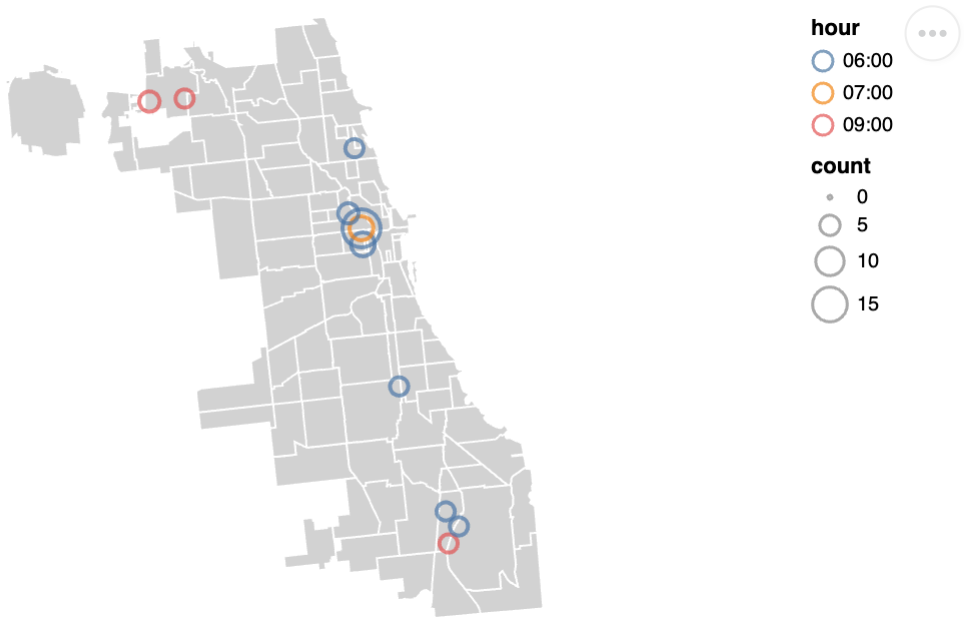
1.

a.

it would NOT be a good idea to collapse the data by range of hours. It is neither practical nor efficient. If we consider all potential data ranges, it would have nearly 2^{24} possibilities; but if we provide some data ranges (for example, 6-9, 10-12) we would lose flexibility. Therefore, a dynamic, real-time computations provide a better balance between flexibility and computational efficiency.

b.

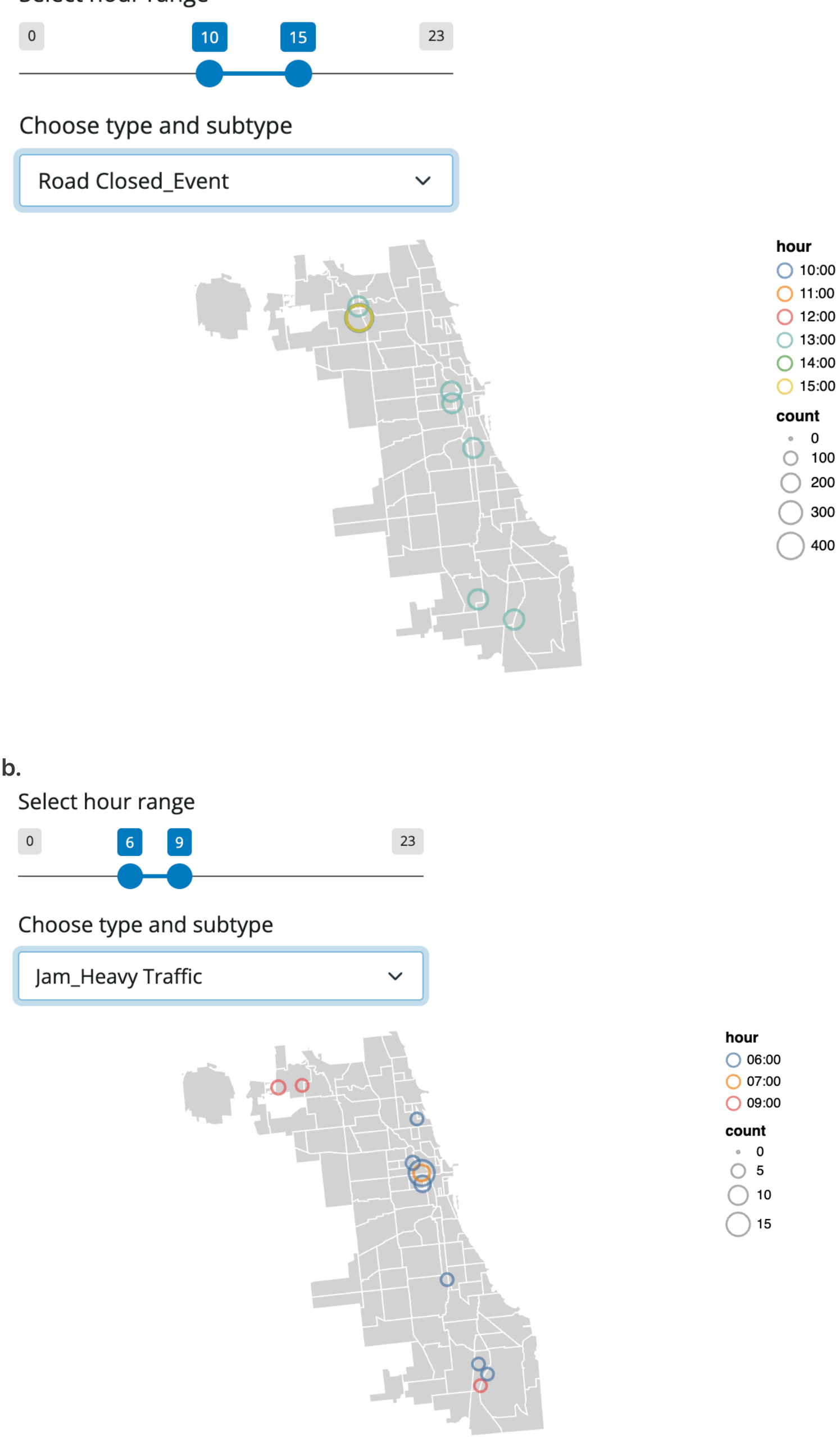
```
background = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='white'
).project('albersUsa').properties(
    width=500,
    height=300
)

pointrange = alt.Chart(top_alerts_map_byhour).mark_point().transform_filter(
    (alt.datum.updated_type == 'Jam') &
    (alt.datum.updated_subtype == 'Heavy Traffic') &
    (alt.datum.hour >= '06:00') &
    (alt.datum.hour <= '09:00')
).transform_window(
    # groupby=['hour'],
    rank='rank(count)',
    sort=[alt.SortField('count', order='descending')]
).transform_filter(
    alt.datum.rank <= 10
).encode(
    longitude='longitude_bin:Q',
    latitude='latitude_bin:Q',
    size=alt.Size('count:Q'),
    color=alt.Color('hour:N'),
    tooltip=['longitude_bin', 'latitude_bin', 'count', 'hour']
)

background + pointrange
```



2.

a.

```
# preparation: save the data to the appointed file.
top_alerts_map_byhour.to_csv('top_alerts_map_byhour_sliderrange/top_alerts_map_byhour.csv', index=False
```

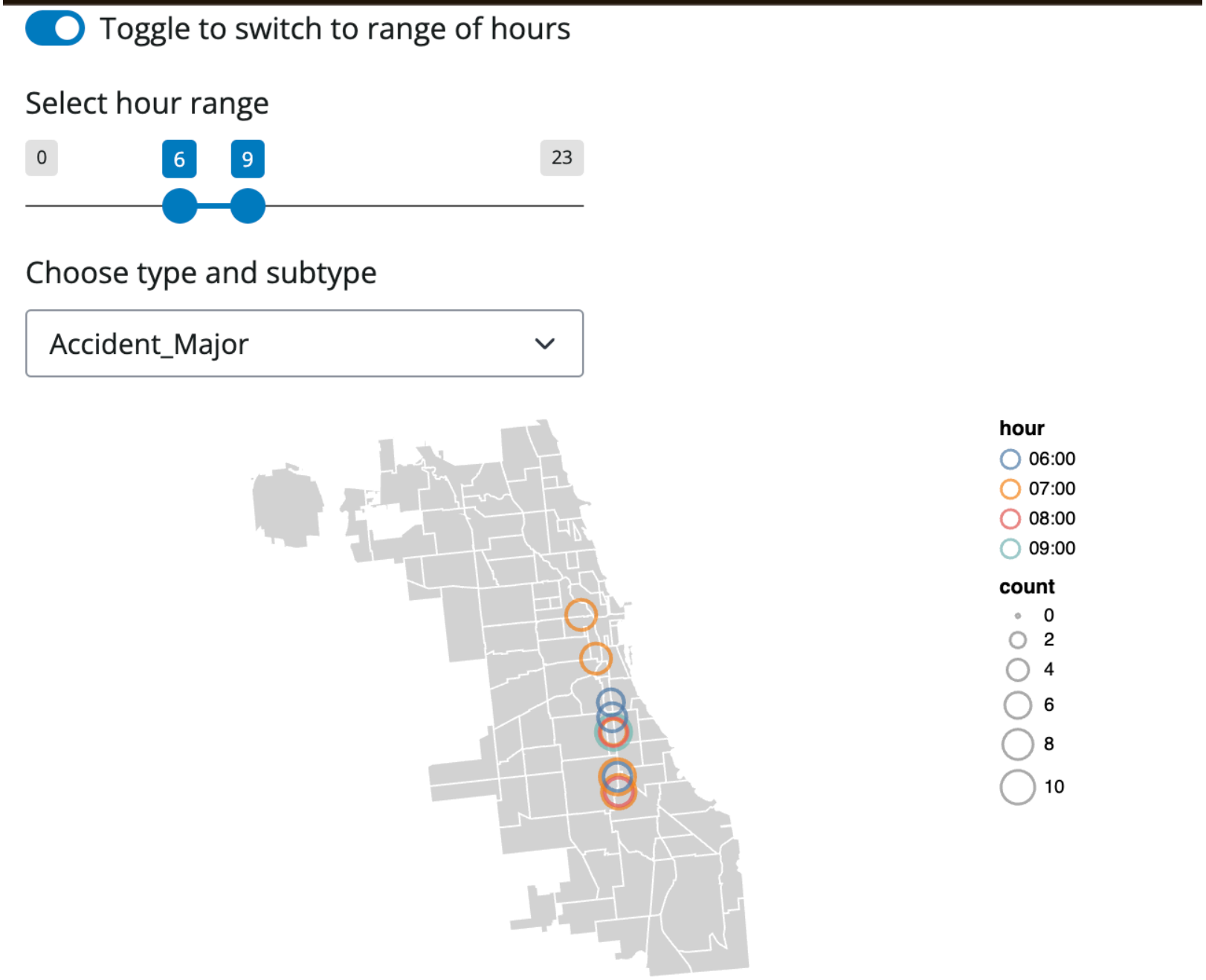


b.

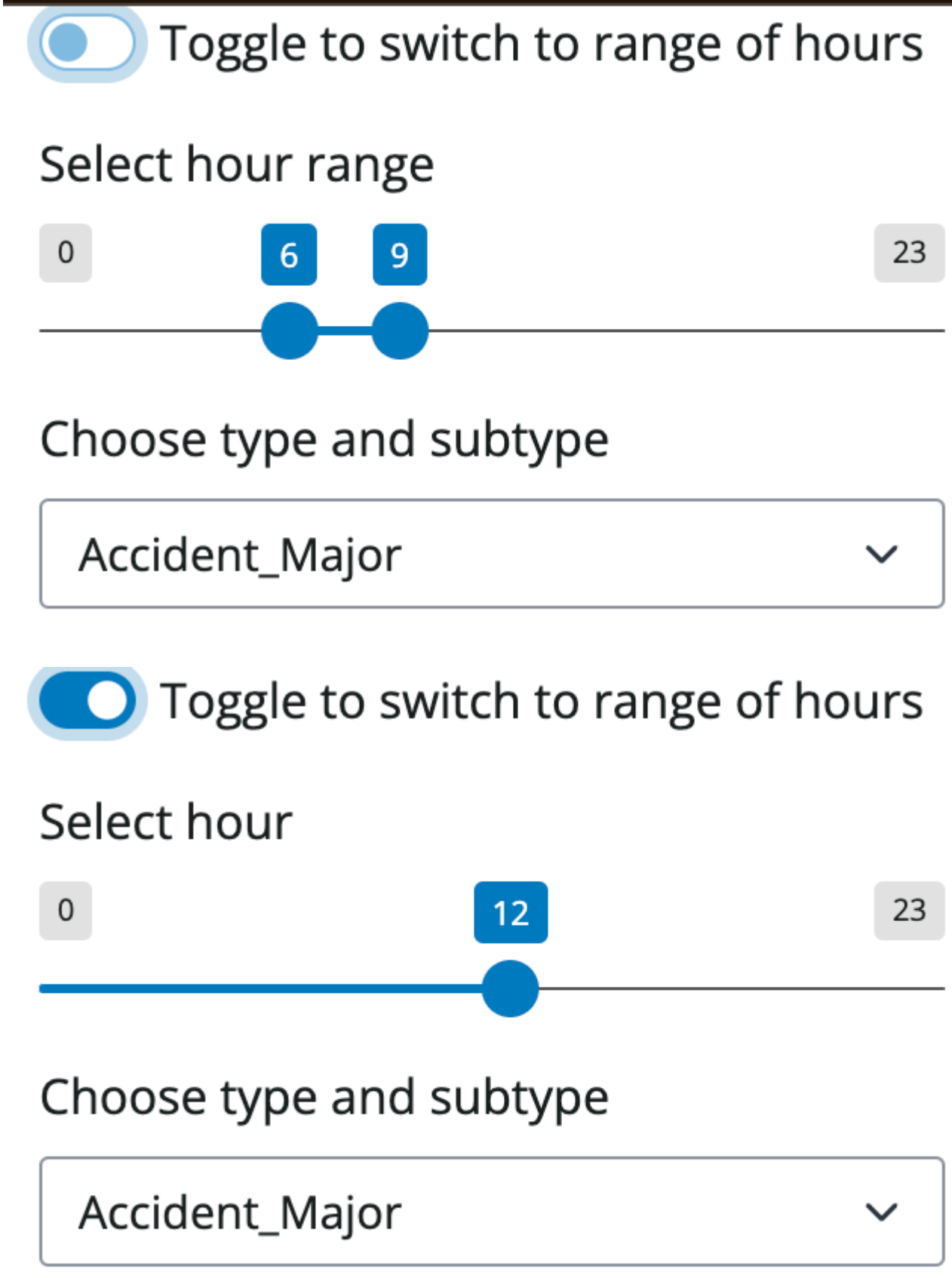
3.

a.

The possible value will be boolean value (i.e.: True when is on, False when is off.)


Toggle to switch to range of hours
Select hour range
0
6
9
23
Choose type and subtype
Accident_Major
hour
06:00
07:00
08:00
09:00
count
0
2
4
6
8
10

b.


Toggle to switch to range of hours
Select hour range
0
6
9
23
Choose type and subtype
Accident_Major
Toggle to switch to range of hours
Select hour
0
12
23
Choose type and subtype
Accident_Major

c.

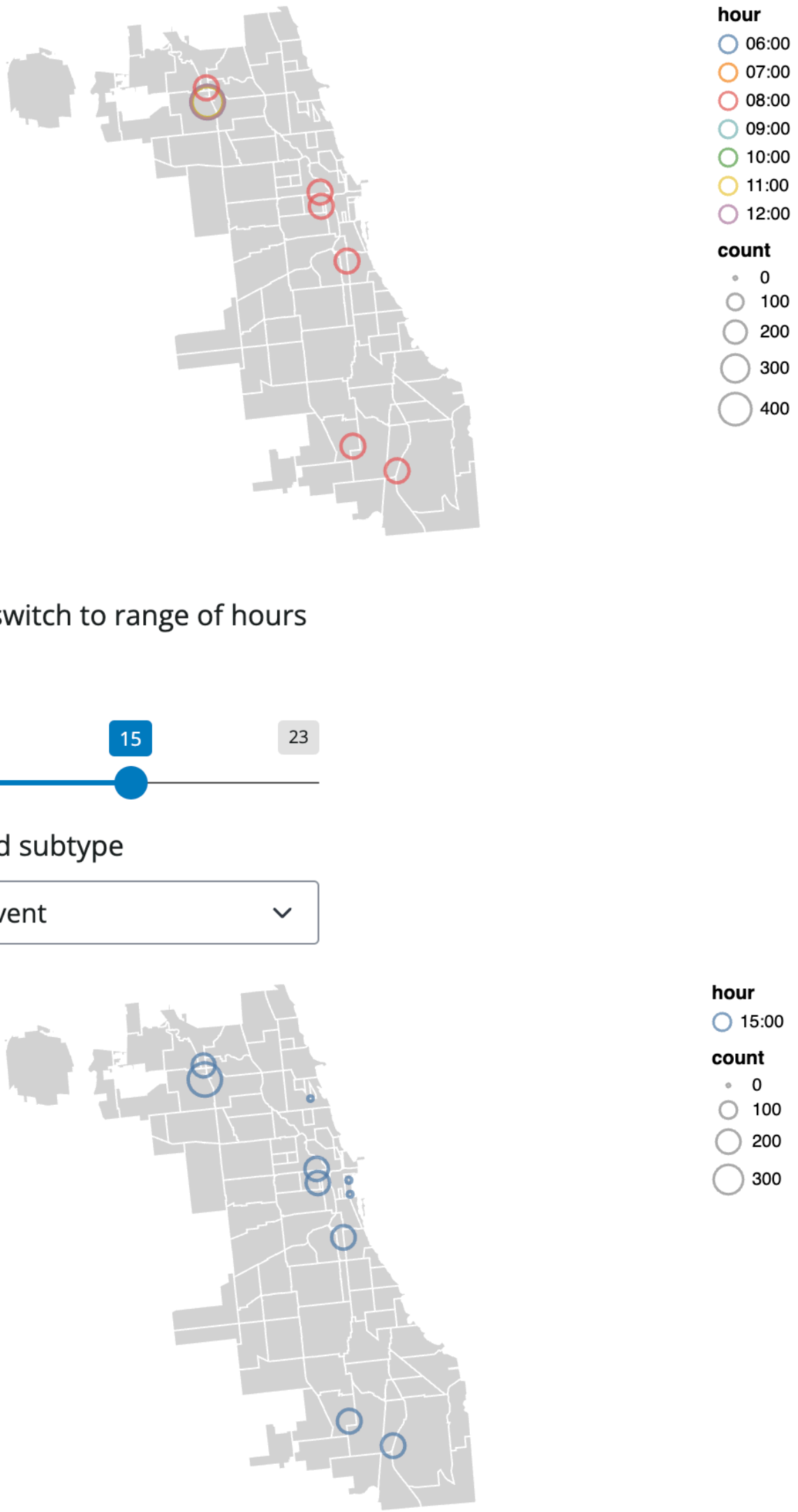
Toggle to switch to range of hours

Select hour range

0 6 12 23

Choose type and subtype

Road Closed_Event



Toggle to switch to range of hours

Select hour

0 15 23

Choose type and subtype

Road Closed_Event

d.

To achieve this, I should introduce a new column, “Time Period”, to categorize hours into broader groups, for example:
Morning: 06:00 - 12:00

Afternoon: 13:00 - 18:00

Then, use this column to group and color the data points by time period.

APPENDIX

```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("/Users/tang/Desktop/1Python II/ProblemSet6/top_alerts_map/app.py")
print_file_contents("/Users/tang/Desktop/1Python II/ProblemSet6/top_alerts_map_byhour/app.py")
print_file_contents("/Users/tang/Desktop/1Python II/ProblemSet6/top_alerts_map_byhour_sliderrange/app.p
```

```python
##########
# APP1   #
##########
from shiny import App, render, ui, reactive
from shinywidgets import render_altair, output_widget
import pandas as pd
import json
import altair as alt

# import top alerts data as df
df = pd.read_csv('top_alerts_map/top_alerts_map.csv')
df['type_subtype'] = df['updated_type'] + '_' + df['updated_subtype']


# import geodata
alt.data_transformers.disable_max_rows()
file_path = "Boundaries - Neighborhoods.geojson"
with open(file_path) as f:
    chicago_geojson = json.load(f)
geo_data = alt.Data(values=chicago_geojson["features"])

# UI
app_ui = ui.page_fluid(
    ui.input_select('type_subtype', 'Choose type and subtype', choices=[]),
    output_widget('top_alerts_plot')
)

# Server
def server(input, output, session):
    # Update dropdown
    @reactive.effect
    def update_dropdown():
        type_list = sorted(df['type_subtype'].unique())
        ui.update_select('type_subtype', choices=type_list)

    # Create Altair plot
    @render_altair
    def top_alerts_plot():

        # Filtered data for points
        selected_combination = input.type_subtype()
        filtered_data = df[
            df['type_subtype'] == selected_combination
        ]

        # Background map
        background = alt.Chart(geo_data).mark_geoshape(
            fill='lightgray',
            stroke='white'
        ).project('albersUsa').properties(width=500, height=300)
```

```
        # points
        points = alt.Chart(filtered_data).mark_point().transform_window(
            rank='rank(count)',
            sort=[alt.SortField('count', order='descending')]
        ).transform_filter(
            alt.datum.rank <= 10
        ).encode(
            longitude='longitude_bin:Q',
            latitude='latitude_bin:Q',
            size='count:Q',
            tooltip=['longitude_bin', 'latitude_bin', 'count']
        )

        return background + points

# App
app = App(app_ui, server)

```
```python
############
#   APP2   #
############

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

# import top alerts data as df
df = pd.read_csv('top_alerts_map_byhour/top_alerts_map_byhour.csv')
df['type_subtype'] = df['updated_type'] + '_' + df['updated_subtype']


# import geodata
alt.data_transformers.disable_max_rows()
file_path = "Boundaries - Neighborhoods.geojson"
with open(file_path) as f:
    chicago_geojson = json.load(f)
geo_data = alt.Data(values=chicago_geojson["features"])

# UI
app_ui = ui.page_fluid(
    ui.input_slider('hour','Select hour',min=0,max=23,value=12,step=1),
    ui.input_select('type_subtype', 'Choose type and subtype', choices=[]),
    output_widget('top_alerts_byhour_plot')
)


# Server
def server(input, output, session):
    # Update dropdown
    @reactive.effect
    def update_dropdown():
        type_list = sorted(df['type_subtype'].unique())
        ui.update_select('type_subtype', choices=type_list)

# Create Altair plot
@render_altair
def top_alerts_byhour_plot():

     # Filtered data for points
    selected_combination = input.type_subtype()
    selected_time = f"{input.hour():02d}:00"
    filtered_data = df[
        (df['type_subtype'] == selected_combination) &
        (df['hour'] == selected_time)
    ]

    # Background map
    background = alt.Chart(geo_data).mark_geoshape(
        fill='lightgray',
        stroke='white'
    ).project('albersUsa').properties(width=500, height=300)

    # points
    points = alt.Chart(filtered_data).mark_point().transform_window(
        rank='rank(count)',
        sort=[alt.SortField('count', order='descending')]
    ).transform_filter(
        alt.datum.rank <= 10
```

```
        ).encode(
            longitude='longitude_bin:Q',
            latitude='latitude_bin:Q',
            size='count:Q',
            tooltip=['longitude_bin', 'latitude_bin', 'count']
        )

        return background + points

# App
app = App(app_ui, server)


```
```python
#############
#   APP3   #
#############

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

# import top alerts data as df
df = pd.read_csv('top_alerts_map_byhour_sliderrange/top_alerts_map_byhour.csv')
df['type_subtype'] = df['updated_type'] + '_' + df['updated_subtype']

# import geodata
alt.data_transformers.disable_max_rows()
file_path = "Boundaries - Neighborhoods.geojson"
with open(file_path) as f:
    chicago_geojson = json.load(f)
geo_data = alt.Data(values=chicago_geojson["features"])

# UI
app_ui = ui.page_fluid(
    ui.input_switch(
        "switch_button",
        "Toggle to switch to range of hours",
        value = False),
    ui.panel_conditional(
        "input.switch_button === false",
        ui.input_slider('hour_range',
                        'Select hour range',
                        min=0,max=23,
                        value=(6,9),
                        step=1
        )
    ),
    ui.panel_conditional(
        "input.switch_button === true",
        ui.input_slider('hour',
                        'Select hour',
                        min=0,max=23,
                        value=12,
                        step=1
        )
    ),
    ui.input_select('type_subtype',
                    'Choose type and subtype',
                    choices=[]),
    output_widget('top_alerts_byhour_plot')
)


# Server
def server(input, output, session):
    # Update dropdown
    @reactive.effect
    def update_dropdown():
        type_list = sorted(df['type_subtype'].unique())
        ui.update_select('type_subtype', choices=type_list)



    # Create Altair plot
    @render_altair
    def top_alerts_byhour_plot():
```

```
    # conditional filter
        selected_combination = input.type_subtype()

        if input.switch_button():
            selected_time = f"{input.hour():02d}:00"
            filtered_data = df[
                (df['type_subtype'] == selected_combination) &
                (df['hour'] == selected_time)
            ]
        else:
            start_hour, end_hour = input.hour_range()
            start_hour = f"{start_hour:02d}:00"
            end_hour = f"{end_hour:02d}:00"
            filtered_data = df[
                (df['type_subtype'] == selected_combination) &
                ((df['hour'] >= start_hour) &
                (df['hour'] <= end_hour))
            ]

        # Background map
        background = alt.Chart(geo_data).mark_geoshape(
            fill='lightgray',
            stroke='white'
        ).project('albersUsa').properties(width=500, height=300)

        points = alt.Chart(filtered_data).mark_point().transform_window(
            rank='rank(count)',
            sort=[alt.SortField('count', order='descending')]
        ).transform_filter(
            alt.datum.rank <= 10
        ).encode(
            longitude='longitude_bin:Q',
            latitude='latitude_bin:Q',
            size='count:Q',
            color=alt.Color('hour:N'),
            tooltip=['longitude_bin', 'latitude_bin', 'count']
        )

        return background + points

# App
app = App(app_ui, server)


```