

Problem Set 6 - Waze Shiny Dashboard

Lauren Laine

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: `**__** LL`
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: `**__** 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 corresponding-section for the code style rubric.

Notes: see the [Quarto documentation \(link\)](#) for directions on inserting images into your knitted document.

IMPORTANT: For the App portion of the PS, in case you can not arrive to the expected functional dashboard we will need to take a look at your `app.py` file. You can use the following code chunk template to “import” and print the content of that file. Please, don’t forget to also tag the corresponding code chunk as part of your submission!

```
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(r"C:\Users\laine\OneDrive\Documents\GitHub\problem-set-6\top_alerts_map\basic-app\app.p
```

```
↪
```

```
```python
```

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

```
app_ui = ui.page_fluid(
 ui.input_select(id='type_subtype', label='Choose a type and subtype',
 choices=[
 'JAM: Unclassified',
 'JAM: JAM_HEAVY_TRAFFIC',
 'JAM: JAM_MODERATE_TRAFFIC',
 'JAM: JAM_STAND_STILL_TRAFFIC',
 'JAM: JAM_LIGHT_TRAFFIC',
 'ACCIDENT: Unclassified',
 'ACCIDENT: ACCIDENT_MAJOR',
 'ACCIDENT: ACCIDENT_MINOR',
 'ROAD_CLOSED: Unclassified',
 'ROAD_CLOSED: ROAD_CLOSED_EVENT',
 'ROAD_CLOSED: ROAD_CLOSED_CONSTRUCTION',
 'ROAD_CLOSED: ROAD_CLOSED_HAZARD',
 'HAZARD: Unclassified',
 'HAZARD: ON_ROAD',
 'HAZARD: On Shoulder',
 'HAZARD: Weather']),
 output_widget('plot'),
)
```

```
def server(input, output, session):
 @reactive.calc
 def choice():
 return input.type_subtype()

 @reactive.calc
 def type():
 return choice().split(':')[0]

 @reactive.calc
 def subtype():
 subtype = choice().split(':')[1]
 subtype = subtype.replace(" ", "") # Remove extra spaces
 return subtype

 @reactive.calc
 def df():
```

```
df=pd.read_csv(r"C:\Users\laine\OneDrive\Documents\GitHub\problem-set-6\top_alerts_map\top_alerts_m
```

```

 return df

@reactive.calc
def subset():
 # Filter the dataframe based on selected type and subtype
 current_type = type()
 current_subtype = subtype()
 full_df=df()
 type_subset = full_df[full_df['type'] == current_type]
 subset = type_subset[type_subset['subtype'] == current_subtype]
 return subset

@reactive.calc
def chi_geo_data():
 file_path = r"C:\Users\laine\OneDrive\Documents\GitHub\problem-set-6\Boundaries -
 Neighborhoods .geojson"
 with open(file_path) as f:
 chicago_geojson = json.load(f)

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

@render_altair
def plot():
 data=subset()
 min_lat=41.65
 max_lat=42.0
 min_long=-87.84
 max_long=-87.58

 points=chart = alt.Chart(data).mark_point().encode(
 alt.X('Longitude:Q', scale=alt.Scale(domain=[min_long, max_long])),
 alt.Y('Latitude:Q', scale=alt.Scale(domain=[min_lat, max_lat])),
 alt.Size('Count:Q')
)

 chi_map=alt.Chart(chi_geo_data()).mark_geoshape(
 fill='lightgray',
 stroke='white'
).encode().properties(
 width=350,
 height=500)

 full_plot=chi_map+points
 return full_plot

```

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

```
```
```

App 2 Code:

```
print_file_contents(r"C:\Users\laine\OneDrive\Documents\GitHub\problem-set-6\top_alerts_map_byhour\basic-ap
```

```
```python
from shiny import App, render, ui, reactive

```

```

from shinywidgets import render_altair, output_widget
import pandas as pd
import altair as alt
import json
import re

app_ui = ui.page_fluid(
 ui.input_select(id='type_subtype', label='Choose a type and subtype',
 choices=[
 'JAM: Unclassified',
 'JAM: JAM_HEAVY_TRAFFIC',
 'JAM: JAM_MODERATE_TRAFFIC',
 'JAM: JAM_STAND_STILL_TRAFFIC',
 'JAM: JAM_LIGHT_TRAFFIC',
 'ACCIDENT: Unclassified',
 'ACCIDENT: ACCIDENT_MAJOR',
 'ACCIDENT: ACCIDENT_MINOR',
 'ROAD_CLOSED: Unclassified',
 'ROAD_CLOSED: ROAD_CLOSED_EVENT',
 'ROAD_CLOSED: ROAD_CLOSED_CONSTRUCTION',
 'ROAD_CLOSED: ROAD_CLOSED_HAZARD',
 'HAZARD: Unclassified',
 'HAZARD: ON_ROAD',
 'HAZARD: On Shoulder',
 'HAZARD: Weather'
]),
 ui.input_slider(id='hour', label= "Choose an Hour (UTC)", min=1, max=24, value=1),
 output_widget('plot'))

def server(input, output, session):
 @reactive.calc
 def df():
 df=pd.read_csv(r"C:\Users\laine\OneDrive\Documents\GitHub\problem-set-6\top_alerts_map_byhour\top_a
 return df

 @reactive.calc
 def choice():
 return input.type_subtype()

 @reactive.calc
 def type():
 return choice().split(':')[0]

 @reactive.calc
 def subtype():
 subtype = choice().split(':')[1]
 subtype = subtype.replace(" ", "") # Remove extra spaces
 return subtype

 @reactive.calc
 def hour():
 return input.hour()

 @render.text
 def hour_check():
 return input.hour()

```

```

@reactive.calc
def subset():
 # Filter the dataframe based on selected type and subtype and hour
 current_type = type()
 current_subtype = subtype()
 current_hour= hour()

 full_df=df()
 hour_subset=full_df[full_df['hour']==current_hour]
 type_subset = hour_subset[hour_subset['type'] == current_type]
 subset = type_subset[type_subset['subtype'] == current_subtype]
 return subset

@reactive.calc
def chi_geo_data():
 file_path = r"C:\Users\laine\OneDrive\Documents\GitHub\problem-set-6\Boundaries -
 Neighborhoods .geojson"
 with open(file_path) as f:
 chicago_geojson = json.load(f)

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

@render_altair
def plot():
 data=subset()
 min_long=-87.93
 max_long=-87.56
 min_lat=41.65
 max_lat=42.01

 points=chart = alt.Chart(data).mark_point().encode(
 alt.X('Longitude:Q', scale=alt.Scale(domain=[min_long, max_long])),
 alt.Y('Latitude:Q', scale=alt.Scale(domain=[min_lat, max_lat])),
 alt.Size('Count:Q')
)

 chi_map=alt.Chart(chi_geo_data()).mark_geoshape(
 fill='lightgray',
 stroke='white'
).encode().properties(
 width=350,
 height=500)

 full_plot=chi_map+points
 return full_plot

```

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

```
```
```

App 3 Code:

```
print_file_contents(r"C:\Users\laine\OneDrive\Documents\GitHub\problem-set-6\top_alerts_map_byhour_sliderra
```

```

```python
from shiny import App, render, ui, reactive

```

```

from shinywidgets import render_altair, output_widget
import pandas as pd
import altair as alt
import json
import re

app_ui = ui.page_fluid(
 ui.input_select(id='type_subtype', label='Choose a type and subtype',
 choices=[
 'JAM: Unclassified',
 'JAM: JAM_HEAVY_TRAFFIC',
 'JAM: JAM_MODERATE_TRAFFIC',
 'JAM: JAM_STAND_STILL_TRAFFIC',
 'JAM: JAM_LIGHT_TRAFFIC',
 'ACCIDENT: Unclassified',
 'ACCIDENT: ACCIDENT_MAJOR',
 'ACCIDENT: ACCIDENT_MINOR',
 'ROAD_CLOSED: Unclassified',
 'ROAD_CLOSED: ROAD_CLOSED_EVENT',
 'ROAD_CLOSED: ROAD_CLOSED_CONSTRUCTION',
 'ROAD_CLOSED: ROAD_CLOSED_HAZARD',
 'HAZARD: Unclassified',
 'HAZARD: ON_ROAD',
 'HAZARD: On Shoulder',
 'HAZARD: Weather'
]),
 ui.input_switch('switch', 'Toggle to switch to range of hours', False),
 ui.panel_conditional('input.switch',
 ui.input_slider(id='hour_range', label= "Choose a Range of Hours (UTC)",
 min=1, max=24, value=[1,3]),
 output_widget('plot_range')
),
 ui.panel_conditional('!input.switch',
 ui.input_slider(id='hour', label= "Choose an Hour (UTC)", min=1, max=24,
 value=1),
 output_widget('plot'))
)

def server(input, output, session):
 @reactive.calc
 def df():
 df=pd.read_csv(r"C:\Users\laine\OneDrive\Documents\GitHub\problem-set-6\top_alerts_map_byhour\top_a
 return df

 @reactive.calc
 def choice():
 return input.type_subtype()

 @reactive.calc
 def type():
 return choice().split(':')[0]

 @reactive.calc
 def subtype():
 subtype = choice().split(':')[1]
 subtype = subtype.replace(" ", "") # Remove extra spaces

```

```

 return subtype

@reactive.calc
def hour():
 return input.hour()

@render.text
def hour_check():
 return input.hour()

@reactive.calc
def subset_range():
 # Filter the dataframe based on selected type and subtype and hour
 hour_start=input.hour_range()[0]
 hour_end=input.hour_range()[1]
 current_type = type()
 current_subtype = subtype()

 full_df=df()
 hour_subset=full_df[(full_df['hour']>=hour_start) & (full_df['hour']<=hour_end)]
 type_subset = hour_subset[hour_subset['type'] == current_type]
 subset = type_subset[type_subset['subtype'] == current_subtype]
 return subset

@reactive.calc
def subset():
 # Filter the dataframe based on selected type and subtype and hour
 current_hour=hour()
 current_type = type()
 current_subtype = subtype()

 full_df=df()
 hour_subset=full_df[full_df['hour']==current_hour]
 type_subset = hour_subset[hour_subset['type'] == current_type]
 subset = type_subset[type_subset['subtype'] == current_subtype]
 return subset

@reactive.calc
def chi_geo_data():
 file_path = r"C:\Users\laine\OneDrive\Documents\GitHub\problem-set-6\Boundaries -
 Neighborhoods .geojson"
 with open(file_path) as f:
 chicago_geojson = json.load(f)

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

@render_altair
def plot():
 data=subset()
 min_long=-87.93
 max_long=-87.56
 min_lat=41.65
 max_lat=42.01

 points=chart = alt.Chart(data).mark_point().encode(

```

```

alt.X('Longitude:Q', scale=alt.Scale(domain=[min_long, max_long])),
alt.Y('Latitude:Q', scale=alt.Scale(domain=[min_lat, max_lat])),
alt.Size('Count:Q')
)

chi_map=alt.Chart(chi_geo_data()).mark_geoshape(
fill='lightgray',
stroke='white'
).encode().properties(
width=350,
height=500)

full_plot=chi_map+points
return full_plot

@render_altair
def plot_range():
data=subset_range()
min_long=-87.93
max_long=-87.56
min_lat=41.65
max_lat=42.01

points=chart = alt.Chart(data).mark_point().encode(
alt.X('Longitude:Q', scale=alt.Scale(domain=[min_long, max_long])),
alt.Y('Latitude:Q', scale=alt.Scale(domain=[min_lat, max_lat])),
alt.Size('Count:Q')
)

chi_map=alt.Chart(chi_geo_data()).mark_geoshape(
fill='lightgray',
stroke='white'
).encode().properties(
width=350,
height=500)

full_plot=chi_map+points
return full_plot

```

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

```
...
```

## Background

### Data Download and Exploration (20 points)

1.

```
sample=pd.read_csv(r"C:\Users\laine\OneDrive\Documents\GitHub\problem-set-6\waze_data_sample.csv")
sample.head()
```



	Unnamed: 0	city	confidence	nThumbsUp	street	uuid	country
0	584358	Chicago, IL	0	NaN	NaN	c9b88a12-79e8-44cb-aadd-a75855fc4bcb	US
1	472915	Chicago, IL	0	NaN	I-90 E	7c634c0a-099c-4262-b57f-e893bdebce73	US
2	550891	Chicago, IL	0	NaN	I-90 W	7aa3c61a-f8dc-4fe8-bbb0-db6b9e0dc53b	US
3	770659	Chicago, IL	0	NaN	NaN	3b95dd2f-647c-46de-b4e1-8ebc73aa9221	US
4	381054	Chicago, IL	0	NaN	N Pulaski Rd	13a5e230-a28a-4bf4-b928-bc1dd38850e0	US

Unnamed 0: Index city: Nominal confidence: Quantitative nThumbsUp: Quantitative street: Nominal uuid: Nominal country: Nominal type: Nominal subtype: Nominal RoadType: Nominal reliability: Quantitative magvar: Nominal reportRating: Quantitative

2.

```
df=pd.read_csv(r"C:\Users\laine\OneDrive\Documents\GitHub\problem-set-6\waze_data.csv")
```

```
def count_null(data):
 count_null=[]
 for column in data:
 num_null=len(data[data[column]=='NULL'])
 count_null.append(num_null)
 return count_null
```

```
df_null_counts=count_null(df)
print(df_null_counts)
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
def count_na(data):
 counts=[]
 for column in data:
 num_na=len(data[data[column].isna()==True])
 counts.append(num_na)
 return counts
```

```
def not_na(data):
 count_not_na=[]
 for column in data:
 num_not_na=len(data[data[column].isna()==False])
 count_not_na.append(num_not_na)
 return count_not_na
```

```
na_counts=count_na(df)
not_na_counts=not_na(df)
variables=df.columns
counts = pd.DataFrame({'Variable': variables, 'Na_Count': na_counts, 'Not_Na_Count':
 ↪ not_na_counts
})
```

```
counts_melt=pd.melt(counts, id_vars=['Variable'],value_vars=['Na_Count', 'Not_Na_Count'])
counts_melt.columns=['Variable', 'Type', 'Count']
counts_melt.head()
```

	Variable	Type	Count
0	city	Na_Count	0
1	confidence	Na_Count	0
2	nThumbsUp	Na_Count	776723
3	street	Na_Count	14073
4	uuid	Na_Count	0

```
stacked_bar_na=alt.Chart(counts_melt, title='Number of Missing and Not Missing Observations by
↪ Variable').mark_bar().encode(
 alt.X('Variable:N'),
 alt.Y('Count:Q'),
 alt.Color('Type:N')
)
stacked_bar_na
```

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

nThumbsUp, street, and subtype all have missing values. nThumbsUp has the highest share of missing observations.

3.

```
print(df['type'].unique())
for i in (df['type'].unique()):
 subset=df[df['type']==i]
 print(f'{i}: {subset['subtype'].unique()}')
 }
```

```
['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']
JAM: [nan 'JAM_HEAVY_TRAFFIC' 'JAM_MODERATE_TRAFFIC' 'JAM_STAND_STILL_TRAFFIC'
 'JAM_LIGHT_TRAFFIC']
ACCIDENT: [nan 'ACCIDENT_MAJOR' 'ACCIDENT_MINOR']
ROAD_CLOSED: [nan 'ROAD_CLOSED_EVENT' 'ROAD_CLOSED_CONSTRUCTION' 'ROAD_CLOSED_HAZARD']
HAZARD: [nan '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'
 'HAZARD_ON_ROAD_LANE_CLOSED' 'HAZARD_WEATHER_FOG'
 'HAZARD_ON_ROAD_ROAD_KILL' 'HAZARD_ON_SHOULDER_ANIMALS'
 'HAZARD_ON_SHOULDER_MISSING_SIGN' 'HAZARD_WEATHER_HEAVY_SNOW'
 'HAZARD_WEATHER_HAIL']
```

All 4 types have a nan subtype. Hazard could have sub-subtypes- on road, on shoulder, and weather.

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

I think that we shouldn't keep NA subtypes because we still know that it fits in that general type, but we don't have more specific information, or it doesn't fit into any of the other categories. Dashboard users may still care about outliers that don't fit into the subtype categories.

```
df['subtype']=df['subtype'].fillna('Unclassified')
```

4.

5.

```
empty_list=[]
thirty_two=empty_list*32
hierarchical=pd.DataFrame({'type': thirty_two, 'subtype': thirty_two, 'updated_type':thirty_two,
 ↪ 'updated_subtype':thirty_two, 'updated_subsubtype':thirty_two
})
```

2.

```
jam=['JAM']
jam_subset=df[df['type']=='JAM']
jam_array=jam_subset['subtype'].unique()
jam_subtypes=jam_array.tolist()
jam_type=len(jam_subtypes)*jam
```

```
accident=['ACCIDENT']
accident_subset=df[df['type']=='ACCIDENT']
accident_array=(accident_subset['subtype'].unique())
accident_subtypes=accident_array.tolist()
accident_type=len(accident_subtypes)*accident
```

```
road_closed=['ROAD_CLOSED']
road_closed_subset=df[df['type']=='ROAD_CLOSED']
road_closed_array=road_closed_subset['subtype'].unique()
road_closed_subtypes=road_closed_array.tolist()
road_closed_type=len(road_closed_subtypes)*road_closed
```

```
hazard=['HAZARD']
hazard_subset=df[df['type']=='HAZARD']
hazard_array=hazard_subset['subtype'].unique()
hazard_subtypes=hazard_array.tolist()
hazard_type=len(hazard_subtypes)*hazard
```

```
all_types=jam_type+accident_type+road_closed_type+hazard_type
all_subtypes=jam_subtypes+accident_subtypes+road_closed_subtypes+hazard_subtypes
```

```
#create list for updated subtype col
on_road=['ON_ROAD']
on_road_list=on_road*10
on_shoulder=['On Shoulder']
on_shoulder_list=on_shoulder*4
weather=['Weather']
weather_list=weather*5
unclassified=['Unclassified']
hazard_breakout=unclassified+on_road_list+on_shoulder_list+weather_list
new_subtypes=jam_subtypes+accident_subtypes+road_closed_subtypes+hazard_breakout
```

```
#create list for updated_subsubtype col
hazard_road_subsubtypes=['General', 'Car Stopped', 'Construction', 'Emergency Vehicle', 'Ice',
 ↪ 'Object', 'Pot Hole', 'Traffic Light Fault', 'Lane Closed', 'Road Kill']
hazard_shoulder_subsubtypes=['General', 'Car Stopped', 'Animals', 'Missing Sign']
hazard_weather_subsubtypes=['General', 'Flood', 'Fog', 'Heavy Snow', 'Hail']
new_subsubtypes=(unclassified*13)+hazard_road_subsubtypes+hazard_shoulder_subsubtypes+hazard_weather_subsub
```

```
#fill in heirarchical
hierarchical['type']=all_types
hierarchical['subtype']=all_subtypes
hierarchical['updated_type']=all_types
hierarchical['updated_subtype']=new_subtypes
hierarchical['updated_subsubtype']=new_subsubtypes
```

```
hierarchical.tail(10)
```

	type	subtype	updated_type	updated_subtype	updated_subsubtype
22	HAZARD	HAZARD_ON_SHOULDER_CAR_STOPPED	HAZARD	ON_ROAD	Road Kill
23	HAZARD	HAZARD_WEATHER	HAZARD	On Shoulder	General
24	HAZARD	HAZARD_WEATHER_FLOOD	HAZARD	On Shoulder	Car Stopped
25	HAZARD	HAZARD_ON_ROAD_LANE_CLOSED	HAZARD	On Shoulder	Animals
26	HAZARD	HAZARD_WEATHER_FOG	HAZARD	On Shoulder	Missing Sign
27	HAZARD	HAZARD_ON_ROAD_ROAD_KILL	HAZARD	Weather	General
28	HAZARD	HAZARD_ON_SHOULDER_ANIMALS	HAZARD	Weather	Flood
29	HAZARD	HAZARD_ON_SHOULDER_MISSING_SIGN	HAZARD	Weather	Fog
30	HAZARD	HAZARD_WEATHER_HEAVY_SNOW	HAZARD	Weather	Heavy Snow
31	HAZARD	HAZARD_WEATHER_HAIL	HAZARD	Weather	Hail

3.

```
merged=df.merge(hierarchical, on=['type','subtype'], how='left')
```

4.

```
assert all(df['type']==merged['type']), 'Diff Type'
assert all(df['subtype']==merged['subtype']), 'Diff Subtype'
```

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

1.

a.

```
import re
merged.head()
```

	city	confidence	nThumbsUp	street	uuid	country	type	s
0	Chicago, IL	0	NaN	NaN	004025a4-5f14-4cb7-9da6-2615daafbf37	US	JAM	
1	Chicago, IL	1	NaN	NaN	ad7761f8-d3cb-4623-951d-dafb419a3ec3	US	ACCIDENT	
2	Chicago, IL	0	NaN	NaN	0e5f14ae-7251-46af-a7f1-53a5272cd37d	US	ROAD_CLOSED	
3	Chicago, IL	0	NaN	Alley	654870a4-a71a-450b-9f22-bc52ae4f69a5	US	JAM	
4	Chicago, IL	0	NaN	Alley	926ff228-7db9-4e0d-b6cf-6739211ffc8b	US	JAM	

```
test='Point(-87.676685 41.929692)'
```

```
check=re.split(r'\(', test, 1)
re.split('\s', check[1], 1)
```

<>:4: SyntaxWarning:

invalid escape sequence '\s'

<>:4: SyntaxWarning:

invalid escape sequence '\s'

C:\Users\laine\AppData\Local\Temp\ipykernel\_8332\2330649517.py:4: SyntaxWarning:

invalid escape sequence '\s'

['-87.676685', '41.929692)']

```
def longitude(txt):
 x=re.split(r'\(', txt, 1)
 y=re.split(r'\s', x[1], 1)
 lat= y[0]
 return lat
```

```
def latitude(txt):
 x=re.split(r'\(', txt, 1)
 y=re.split(r'\s', x[1], 1)
 long=y[1]
 long=long.replace(r')', '')
 return long
```

```
merged['latitude']=merged['geoWKT'].map(latitude)
merged['longitude']=merged['geoWKT'].map(longitude)
```

b.

```
merged['latitude']=pd.to_numeric(merged['latitude'])
merged['longitude']=pd.to_numeric(merged['longitude'])
```

```
merged['binned_lat'] = merged['latitude'].round(2)
merged['binned_long'] = merged['longitude'].round(2)
```

```
grouped=merged.groupby(['binned_lat', 'binned_long']).size()
grouped=grouped.reset_index()
grouped.columns=['latitude', 'longitude', 'count']
```

```
grouped.max()
```

```
latitude 42.02
longitude -87.56
count 21325.00
dtype: float64
```

(-87.56, 42.02) has the most observations with 21325.

c.

```
subset=merged[(merged['updated_type'] == 'JAM') & (merged['updated_subtype'] == 'Unclassified')]
subset_grouped=subset.groupby(['binned_lat', 'binned_long']).size()
subset_grouped=subset_grouped.reset_index()
subset_grouped.columns=['Latitude', 'Longitude', 'Count']
subset_grouped=subset_grouped.sort_values(by='Count', ascending=False)
subset_grouped=subset_grouped.head(10)
subset_grouped['type']='JAM'
subset_grouped['subtype']='Unclassified'
print(subset_grouped)
```

	Latitude	Longitude	Count	type	subtype
392	41.89	-87.65	1086	JAM	Unclassified
410	41.90	-87.66	973	JAM	Unclassified
375	41.88	-87.65	944	JAM	Unclassified
486	41.94	-87.71	624	JAM	Unclassified
449	41.92	-87.67	612	JAM	Unclassified
576	41.98	-87.79	610	JAM	Unclassified
577	41.98	-87.78	604	JAM	Unclassified
466	41.93	-87.69	593	JAM	Unclassified
327	41.85	-87.64	582	JAM	Unclassified
430	41.91	-87.66	581	JAM	Unclassified

```
top_tens=[]
```

```
for i in merged['updated_type'].unique():
 type_set=merged[merged['updated_type']==i]
 for j in type_set['updated_subtype'].unique():
 subset = merged[(merged['updated_type'] == i) & (merged['updated_subtype'] == j)]
 subset_grouped=subset.groupby(['binned_lat', 'binned_long']).size()
 subset_grouped=subset_grouped.reset_index()
 subset_grouped.columns=['Latitude', 'Longitude', 'Count']
 subset_grouped=subset_grouped.sort_values(by='Count', ascending=False)
 subset_grouped=subset_grouped.head(10)
 subset_grouped['type']=i
 subset_grouped['subtype']=j
 top_tens.append(subset_grouped)
```

```
top_ten_df=pd.concat(top_tens, ignore_index=True)
```

```
check=merged[merged['updated_type']=='JAM']
test=check[check['updated_subtype']=='JAM_LIGHT_TRAFFIC']
```

```
top_ten_df.to_csv(r"C:\Users\laine\OneDrive\Documents\GitHub\problem-set-6\top_alerts_map\top_alerts_map.csv")
```

The data is filtered based on type and subtype and then aggregated by binned latitude and binned longitude. The dataframe has 155 rows. It has 16 combinations of types and subtypes but type Jam with subtype Jam Light Traffic only has 5 observations so the final length isn't divisible by 10.

2.

```

max_long=top_ten_df['Longitude'].max()
min_long=top_ten_df['Longitude'].min()
print(min_long, max_long)
max_lat=top_ten_df['Latitude'].max()
min_lat=top_ten_df['Latitude'].min()
print(min_lat, max_lat)

```

```

-87.84 -87.58
41.65 42.0

```

```

subset_jam_heavy_traffic=top_ten_df[(top_ten_df['type']=='JAM')
↪ &(top_ten_df['subtype']=='JAM_HEAVY_TRAFFIC')]
plot_heavy_traffic=alt.Chart(subset_jam_heavy_traffic).mark_point().encode(
 alt.X('Longitude:Q').scale(domain=(min_long, max_long)),
 alt.Y('Latitude:Q').scale(domain=(min_lat, max_lat)),
 alt.Size('Count:Q')
)
plot_heavy_traffic

```

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

3.

a.

b.

```

MODIFY ACCORDINGLY
file_path = r"C:\Users\laine\OneDrive\Documents\GitHub\problem-set-6\Boundaries - Neighborhoods
↪ .geojson"
#----

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

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

```

4.

```

chi_map=alt.Chart(geo_data).mark_geoshape(
 fill='lightgray',
 stroke='white'
).properties(
 width=375,
 height=450
)
chi_map+plot_heavy_traffic

```

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

5.

a.

```

choices=[]
for i in merged['updated_type'].unique():
 type_set=merged[merged['updated_type']==i]
 for j in type_set['updated_subtype'].unique():
 choice=f'{i}: {j}'
 choices.append(choice)

```

```

#print choices to put in choices list in the app
choices

```

```

['JAM: Unclassified',
 'JAM: JAM_HEAVY_TRAFFIC',
 'JAM: JAM_MODERATE_TRAFFIC',
 'JAM: JAM_STAND_STILL_TRAFFIC',
 'JAM: JAM_LIGHT_TRAFFIC',
 'ACCIDENT: Unclassified',
 'ACCIDENT: ACCIDENT_MAJOR',
 'ACCIDENT: ACCIDENT_MINOR',
 'ROAD_CLOSED: Unclassified',
 'ROAD_CLOSED: ROAD_CLOSED_EVENT',
 'ROAD_CLOSED: ROAD_CLOSED_CONSTRUCTION',
 'ROAD_CLOSED: ROAD_CLOSED_HAZARD',
 'HAZARD: Unclassified',
 'HAZARD: ON_ROAD',
 'HAZARD: On Shoulder',
 'HAZARD: Weather']

```



Figure 1: Dropdown Menu



There are 16 type and subtype combinations.

```
test=merged[merged['type']=='ACCIDENT']
test=test[test['subtype']=='Unclassified']
test.head()
```

	city	confidence	nThumbsUp	street	uuid	country	ty
1	Chicago, IL	1	NaN	NaN	ad7761f8-d3cb-4623-951d-dafb419a3ec3	US	A
7	Chicago, IL	1	NaN	DuSable Lake Shore Dr	aba17ef8-bbd2-4bd2-a422-f398840f0654	US	A
8	Chicago, IL	0	NaN	DuSable Lake Shore Dr	845fd8f0-2542-42d2-a39a-d81bd238ab7e	US	A
9	Chicago, IL	0	NaN	DuSable Lake Shore Dr	c0204977-8f76-48f0-acfe-42bbe7f75dc4	US	A
10	Chicago, IL	5	NaN	DuSable Lake Shore Dr	1b466926-c8e7-49f6-8f64-c3021b3f97a7	US	A

Choose a type and subtype

JAM: JAM\_HEAVY\_TRAFFIC

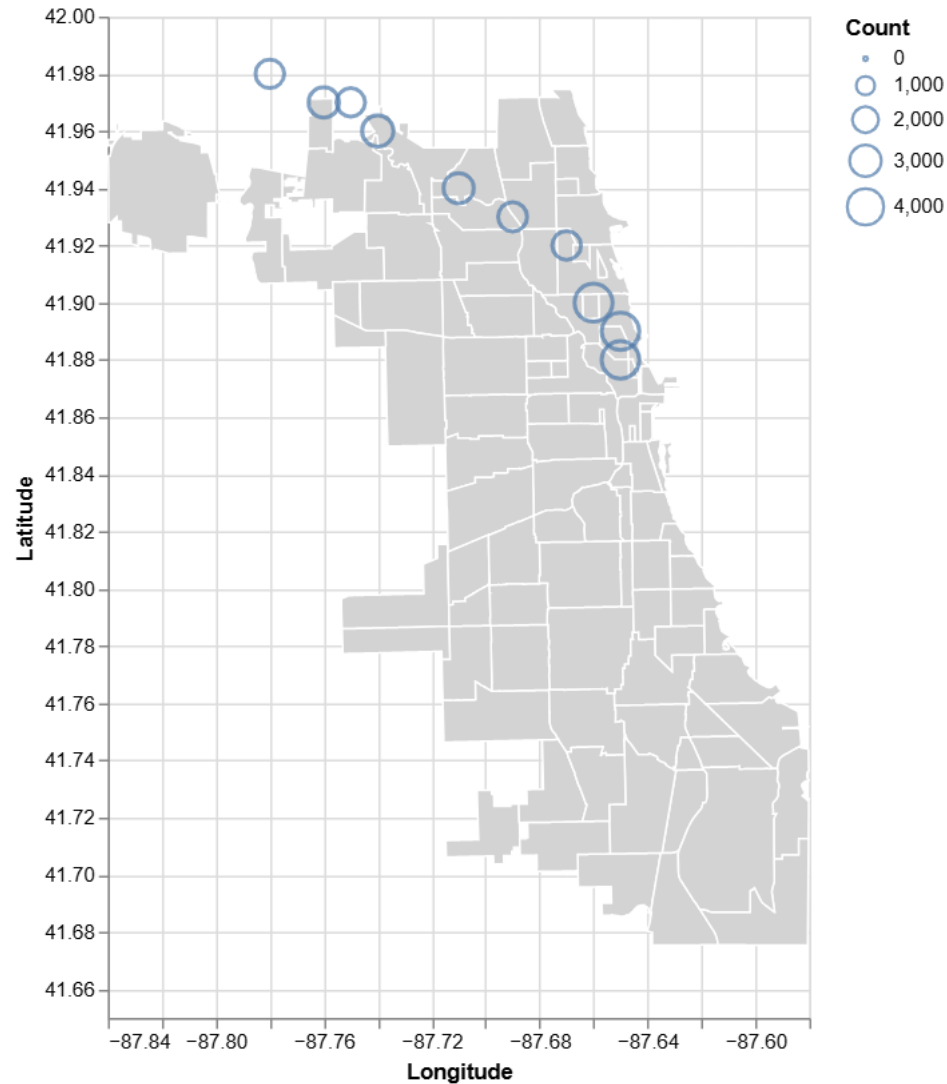


Figure 2: App1 Heavy Traffic

b.

Choose a type and subtype

ROAD\_CLOSED: ROAD\_CLOSED\_EVENT ▾

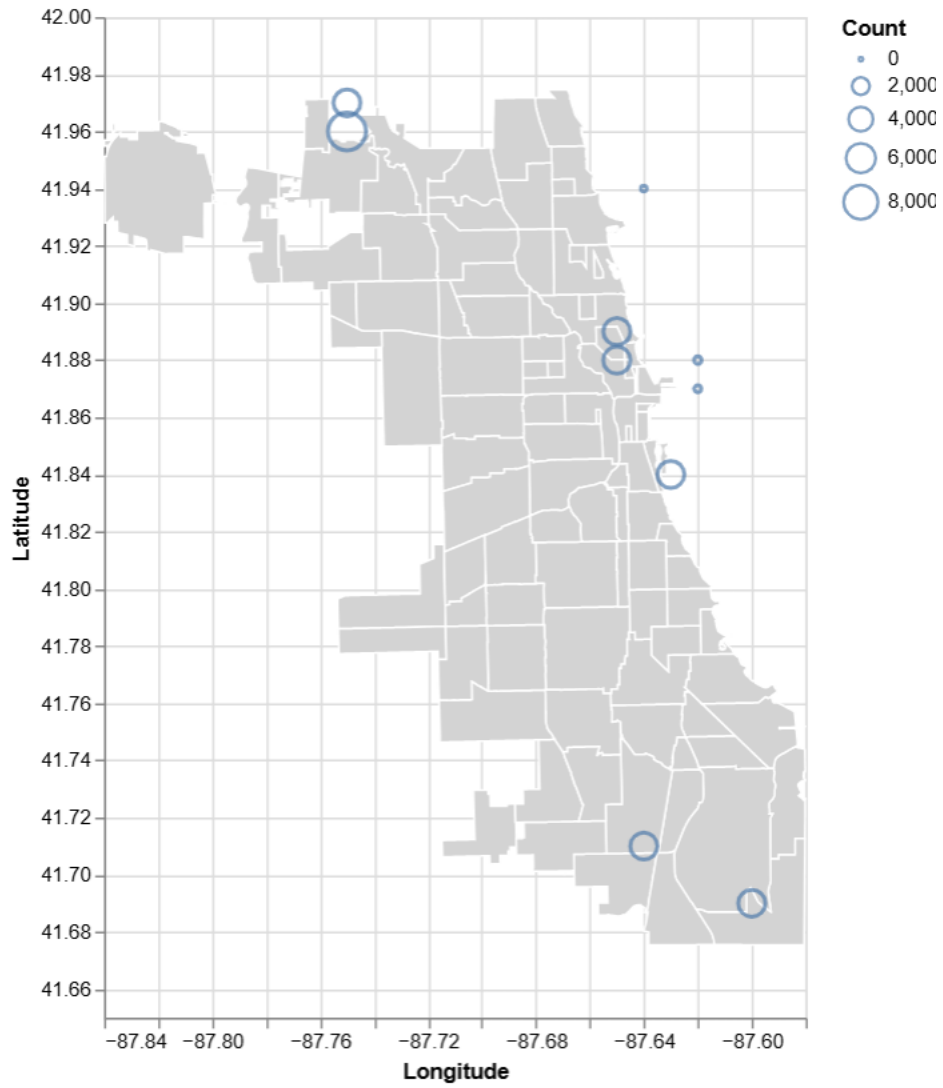
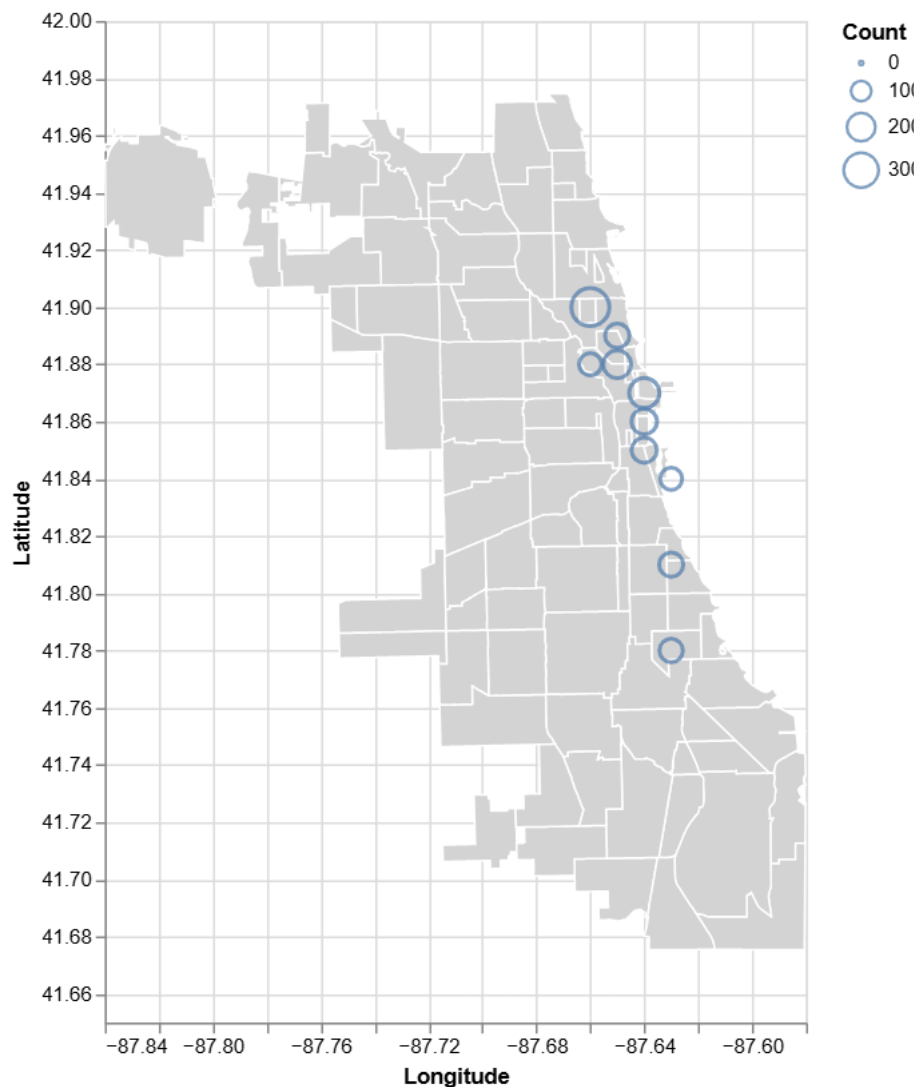


Figure 3: Road Closed Events

c.

Choose a type and subtype

ACCIDENT: ACCIDENT\_MAJOR



- d. Where do the most major accidents occur?
- e. It may also be interesting to look at roadType to see if there are any common road types that involved in accidents, or other alerts. That could be incorporated into the app using color as roadtype.

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

- 1.
  - a. I don't think it would be a good idea to collapse the dataset by this column for two reasons. First, time is very specific if grouped each different time would be separate, ex not grouped by hour or number of hours. This wouldn't be very helpful in making trends readily apparent without having to inspect each specific time. Also the column has times in UTC not in Chicago's central time so the time could actually be misleading.
  - b.

```
merged['ts']=pd.to_datetime(merged['ts'])
merged['hour']=merged['ts'].map(lambda x: x.time().hour)
```

```
top_tens_by_hour=[]
```

```
for x in merged['hour'].unique():
 hour_set=merged[merged['hour']==x]
 for i in hour_set['updated_type'].unique():
 type_set=hour_set[hour_set['updated_type']==i]
 for j in type_set['updated_subtype'].unique():
 subset = type_set[type_set['updated_subtype']==j]
 subset_grouped=subset.groupby(['binned_lat', 'binned_long']).size()
 subset_grouped=subset_grouped.reset_index()
 subset_grouped.columns=['Latitude', 'Longitude', 'Count']
 subset_grouped=subset_grouped.sort_values(by='Count', ascending=False)
 subset_grouped=subset_grouped.head(10)
 subset_grouped['hour']=x
 subset_grouped['type']=i
 subset_grouped['subtype']=j
 top_tens_by_hour.append(subset_grouped)
```

```
top_ten_by_hourdf=pd.concat(top_tens_by_hour, ignore_index=True)
```

```
top_ten_by_hourdf.to_csv(r"C:\Users\laine\OneDrive\Documents\GitHub\problem-set-6\top_alerts_map_byhour\top
```

c.

```
heavy_traffic_at_1=top_ten_by_hourdf[top_ten_by_hourdf['hour']==1]
heavy_traffic_at_1=heavy_traffic_at_1[heavy_traffic_at_1['type']=='JAM']
heavy_traffic_at_1=heavy_traffic_at_1[heavy_traffic_at_1['subtype']=='JAM_HEAVY_TRAFFIC']
```

```
traffic_at_1_plot=alt.Chart(heavy_traffic_at_1).mark_point().encode(
 alt.X('Longitude:Q').scale(domain=(min_long, max_long)),
 alt.Y('Latitude:Q').scale(domain=(min_lat, max_lat)),
 alt.Size('Count:Q'))
```

```
chi_map+traffic_at_1_plot
```

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

```
heavy_traffic_at_6=top_ten_by_hourdf[top_ten_by_hourdf['hour']==6]
heavy_traffic_at_6=heavy_traffic_at_6[heavy_traffic_at_6['type']=='JAM']
heavy_traffic_at_6=heavy_traffic_at_6[heavy_traffic_at_6['subtype']=='JAM_HEAVY_TRAFFIC']
```

```
traffic_at_6_plot=alt.Chart(heavy_traffic_at_6).mark_point().encode(
 alt.X('Longitude:Q').scale(domain=(min_long, max_long)),
 alt.Y('Latitude:Q').scale(domain=(min_lat, max_lat)),
 alt.Size('Count:Q'))
```

```
chi_map+traffic_at_6_plot
```

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

```
heavy_traffic_at_12=top_ten_by_houdf[top_ten_by_houdf['hour']==12]
heavy_traffic_at_12=heavy_traffic_at_12[heavy_traffic_at_12['type']=='JAM']
heavy_traffic_at_12=heavy_traffic_at_12[heavy_traffic_at_12['subtype']=='JAM_HEAVY_TRAFFIC']
```

```
traffic_at_12_plot=alt.Chart(heavy_traffic_at_12).mark_point().encode(
 alt.X('Longitude:Q').scale(domain=(min_long, max_long)),
 alt.Y('Latitude:Q').scale(domain=(min_lat, max_lat)),
 alt.Size('Count:Q'))
```

```
chi_map+traffic_at_12_plot
```

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

2.

Choose a type and subtype

ACCIDENT: ACCIDENT\_MAJOR

▼

Choose an Hour (UTC)

1

6

24




Figure 4: App2 UI

a.

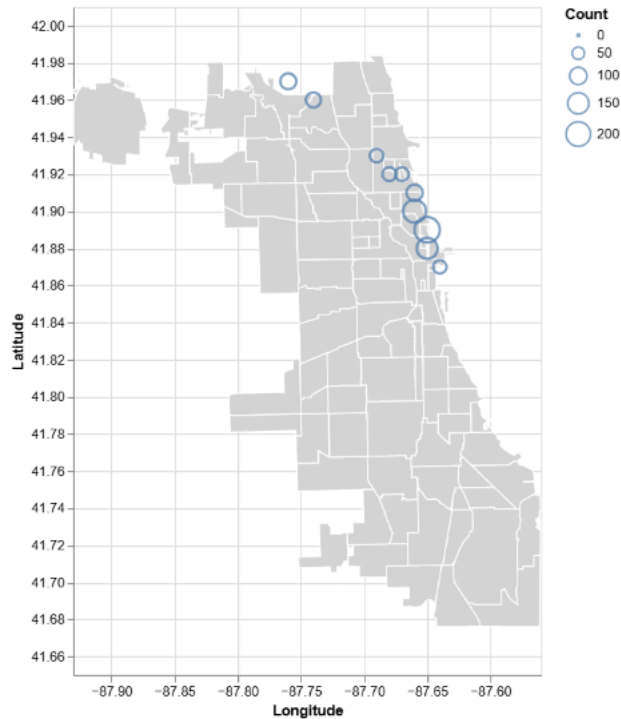
Choose a type and subtype

JAM: JAM\_HEAVY\_TRAFFIC

Choose an Hour (UTC)

1

24



Choose a type and subtype

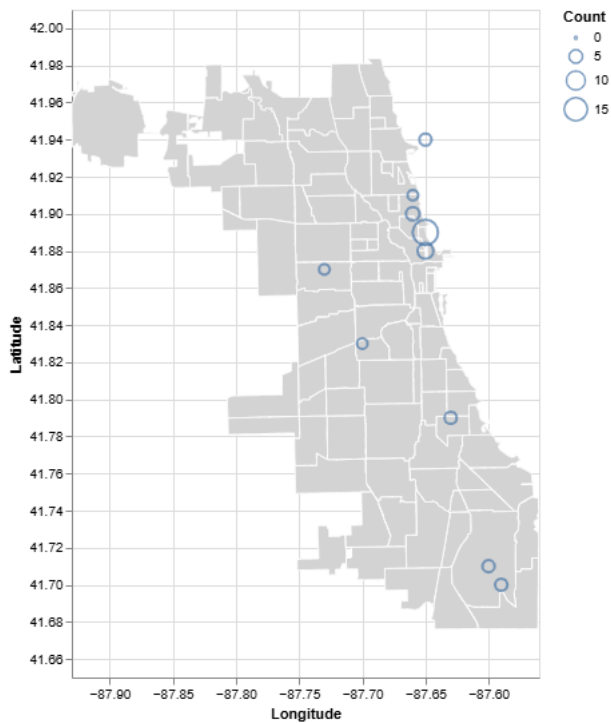
JAM: JAM\_HEAVY\_TRAFFIC

Choose an Hour (UTC)

1

6

24



b.

Choose a type and subtype

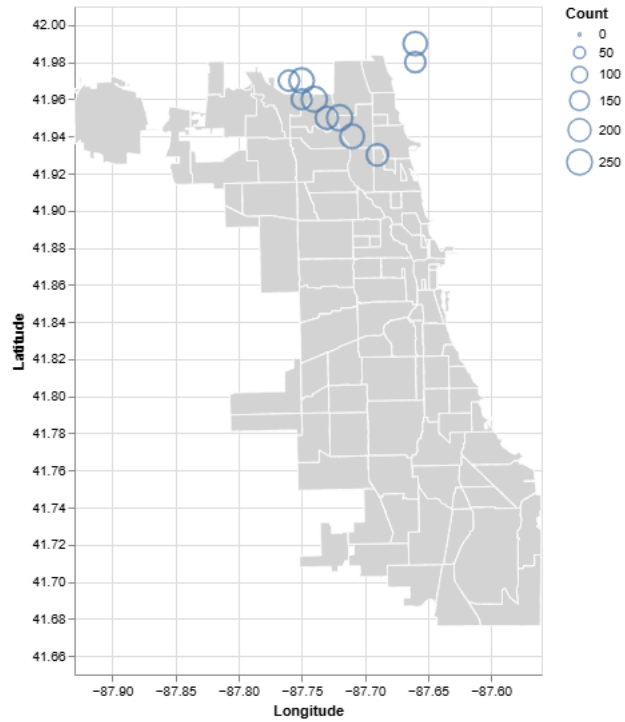
JAM: JAM\_HEAVY\_TRAFFIC

Choose an Hour (UTC)

1

12

24



- c. From the dashboard it appears that construction is done more at night than in the morning.

Choose a type and subtype

ROAD\_CLOSED: ROAD\_CLOSED\_COM

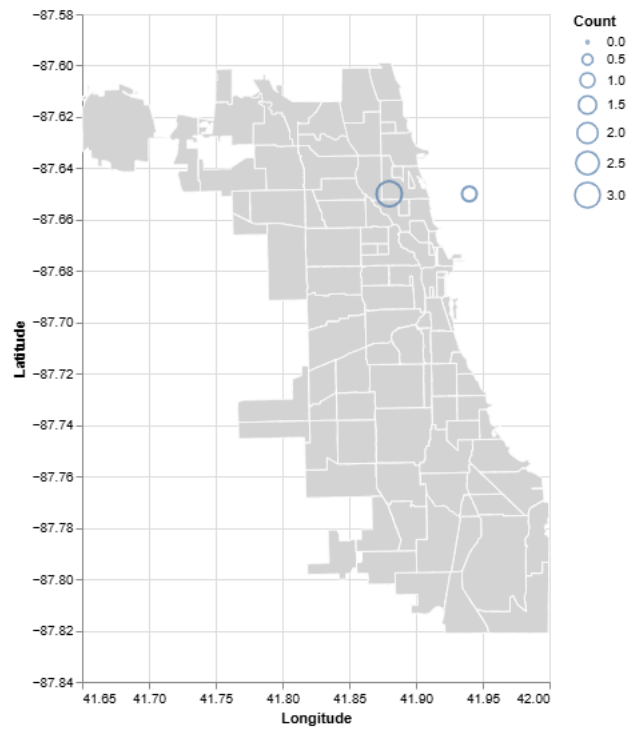
Choose an Hour (UTC)

1

3

24

3



3:00am UTC is equivalent to 9:00am CST in Chicago.



Choose a type and subtype

ROAD\_CLOSED: ROAD\_CLOSED\_COM ▾

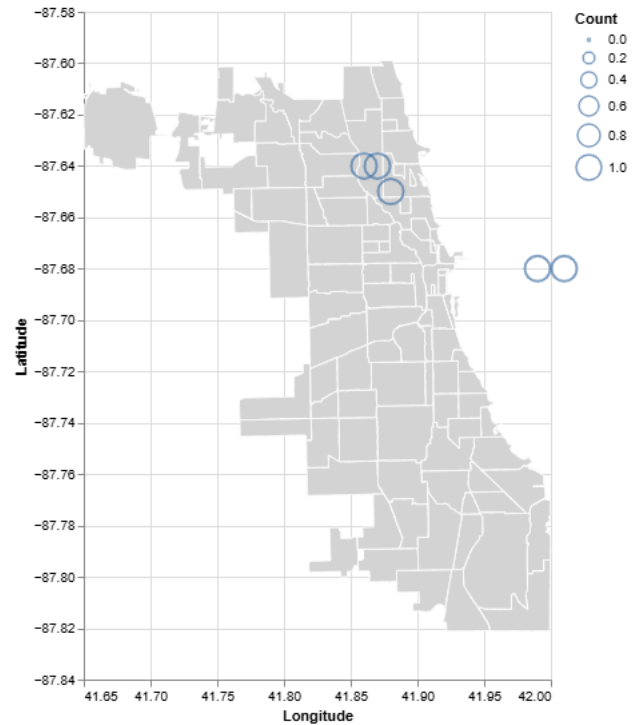
Choose an Hour (UTC)

1

15

24

15



15:00 UTC is equivalent to 3:00pm or 9:00pm CST in Chicago.

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

1.

- a. It would be impractical to collapse the data by alert type and range or hours because there are so many potential ranges of hours. The resulting dataframe would be extremely long. It would be much more efficient to filter the data using  $\geq$  and  $\leq$  operators with the existing byhour dataframe than to run the top ten alerts by type for every potential range of hours.

b.

```
max_long=top_ten_by_hourdf['Longitude'].max()
min_long=top_ten_by_hourdf['Longitude'].min()
print(min_long, max_long)
max_lat=top_ten_by_hourdf['Latitude'].max()
min_lat=top_ten_by_hourdf['Latitude'].min()
print(min_lat, max_lat)
```

```
-87.93 -87.56
41.65 42.01
```

```
heavy_traffic_6_to_9=top_ten_by_hourdf[(top_ten_by_hourdf['hour']>=6)&
→ (top_ten_by_hourdf['hour']<=9)]
heavy_traffic_6_to_9=heavy_traffic_6_to_9[(heavy_traffic_6_to_9['type']=='JAM')&
→ (heavy_traffic_6_to_9['subtype']=='JAM_HEAVY_TRAFFIC')]

heavy_traffic_6_to_9=alt.Chart(heavy_traffic_6_to_9).mark_point().encode(
 alt.X('Longitude:Q').scale(domain=(min_long, max_long)),
 alt.Y('Latitude:Q').scale(domain=(min_lat, max_lat)),
 alt.Size('Count:Q'))

chi_map+heavy_traffic_6_to_9
```

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

2.

Choose a type and subtype

JAM: Unclassified
▼

Choose an Hour (UTC)

1

3

10

24

Figure 5: App3 UI

a.

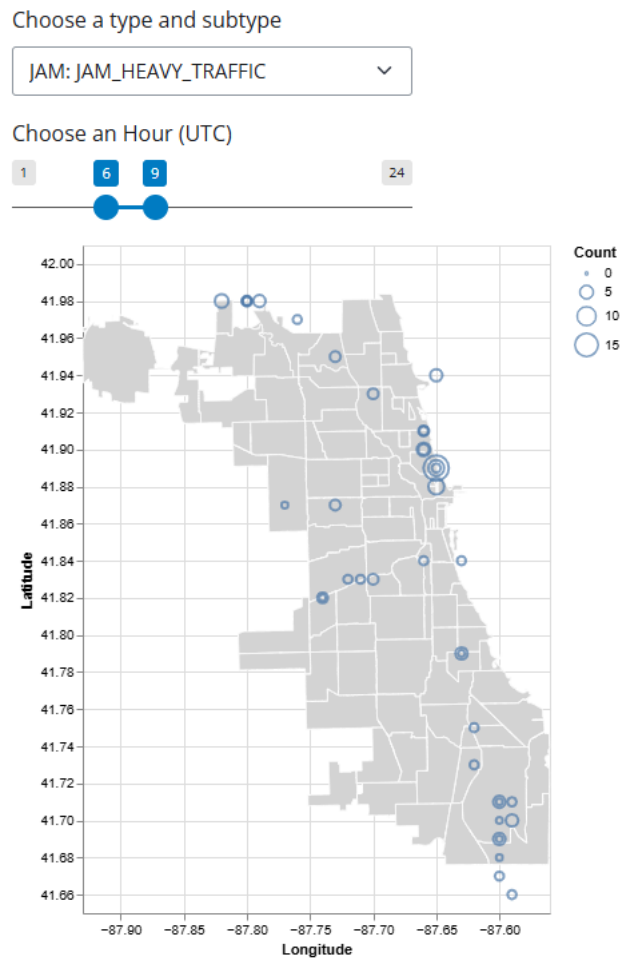


Figure 6: App3 Heavy Traffic 6 to 9

- b.
- 3.

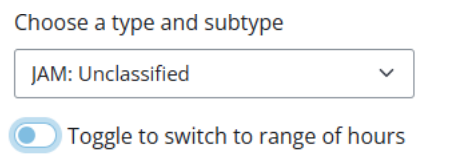


Figure 7: Switch Button

- a.

The possible values for `input.switch_button` are `True` meaning the button would start switched to hour range and `False` meaning the button would start switched to just hour.

Choose a type and subtype

JAM: Unclassified

☒ Toggle to switch to range of hours

Choose an Hour (UTC)

1 24

b.

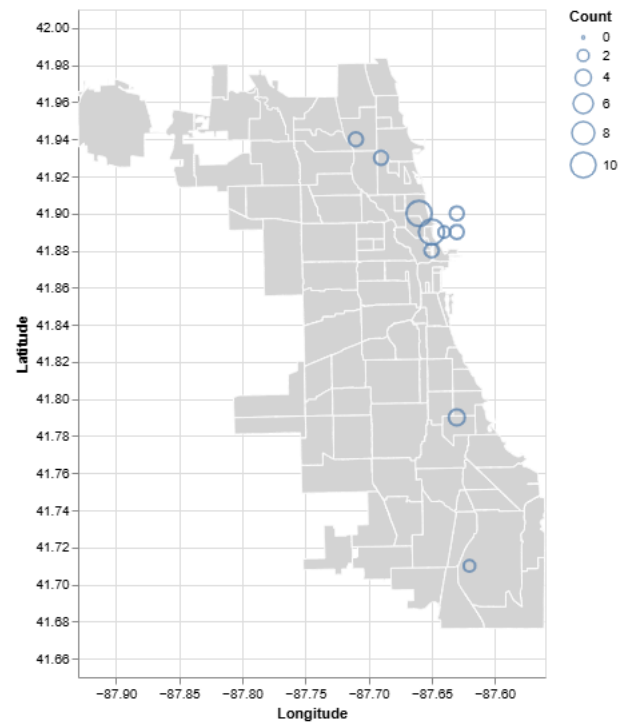
Choose a type and subtype

JAM: Unclassified

☐ Toggle to switch to range of hours

Choose an Hour (UTC)

1 6 24



Choose a type and subtype

JAM: Unclassified

☒ Toggle to switch to range of hours

Choose a Range of Hours (UTC)

1 3 24

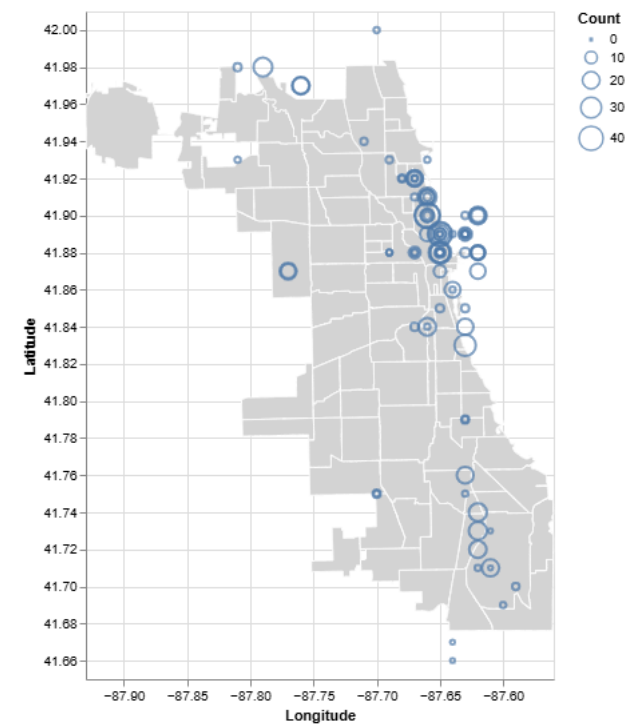
Choose a type and subtype

JAM: Unclassified

☒ Toggle to switch to range of hours

Choose a Range of Hours (UTC)

2 10 24



c.

- d. In order to make the plot displayed in the instructions, you would need to code a new variable for morning or afternoon based on hours. The hours are a little subjective but it could be from around 6:00am-12:pm is morning and 12:00pm-5:00pm is afternoon. Then you would have to plot the new variable using `alt.Color` to distinguish between morning and afternoon. You would also have to code either night, or Na for hours that fall outside of both of these ranges. You could filter out all night/Na before graphing so you are left with only morning and afternoon alerts.