Problem Set 6 - Waze Shiny Dashboard

Kishika Mahajan

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: KM
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **___** (2 point)
3. Late coins used this pset: 1 Late coins left after submission: 0
4. Before starting the problem set, make sure to read and agree to the terms of data usage for the Waze data here.
5. Knit your `ps6.qmd` as a pdf document and name it `ps6.pdf`.
6. 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("./top_alerts_map_byhour/app.py") # Change accordingly
```

Background

Data Download and Exploration (20 points)

1.

```
# Unzipping the file
import zipfile

with
 ↪  zipfile.ZipFile("/Users/kishikamahajan/Desktop/GitHub/student30538/problem_sets/ps6/waze_
 ↪  "r") as zip_ref:
    zip_ref.extractall("extracted_zip")

# importing the csv
sample_data =
 ↪  pd.read_csv("/Users/kishikamahajan/Desktop/GitHub/student30538/problem_sets/ps6/extracted
```

```
data_types_altair  = {
    "column_name": [
        "uuid", "magvar", "type", "subtype", "street", "city", "country",
        "roadType", "reportRating", "Reliability", "confidence", "nThumbsUp",
        "ts", "geo", "geoWKT"
    ],
    "data_type": [
        "Nominal", "Quantitative or Ordinal", "Nominal", "Nominal",
        ↪  "Nominal", "Nominal",
        "Nominal", "Quantitative", "Ordinal", "Ordinal",
        "Ordinal", "Quantitative", "Temporal", "Nominal", "Nominal"
    ]
}

data_types_altair_df = pd.DataFrame(data_types_altair)
data_types_altair_df
```

	column_name	data_type
0	uuid	Nominal
1	magvar	Quantitative or Ordinal
2	type	Nominal
3	subtype	Nominal
4	street	Nominal
5	city	Nominal
6	country	Nominal
7	roadType	Quantitative
8	reportRating	Ordinal
9	Reliability	Ordinal
10	confidence	Ordinal
11	nThumbsUp	Quantitative
12	ts	Temporal
13	geo	Nominal
14	geoWKT	Nominal

2.

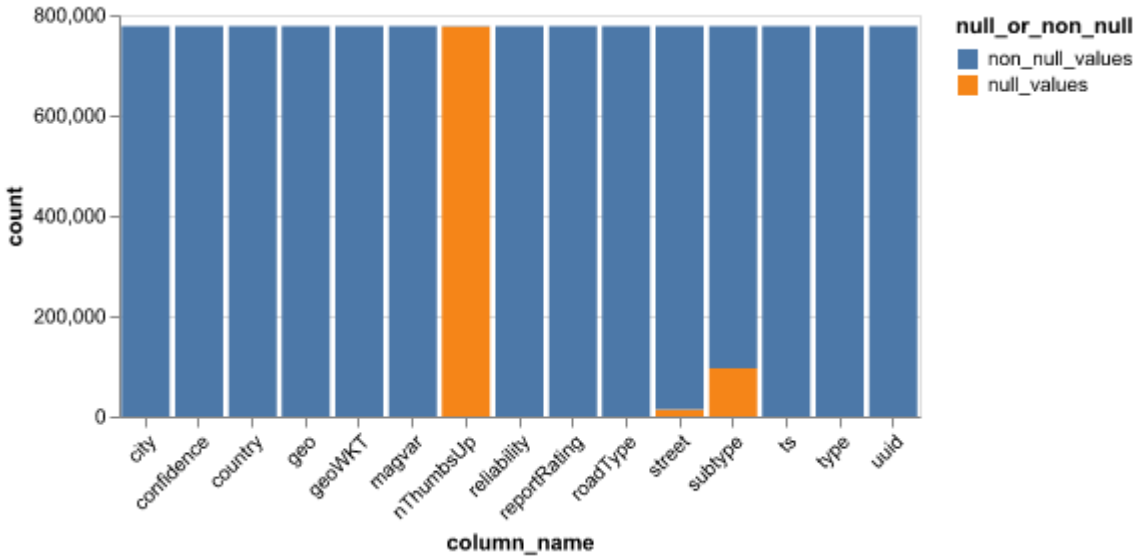
```
# loading the complete data
data =
 ↪  pd.read_csv("/Users/kishikamahajan/Desktop/GitHub/student30538/problem_sets/ps6/extracted
```

```
# getting the null and non-null values
null_values = data.isnull().sum()
non_null_values = data.notnull().sum()

# putting the two together into a df for easier plotting
df_null_analysis = pd.DataFrame({"null_values": null_values,
"non_null_values": non_null_values}).reset_index()

df_null_analysis.rename(columns = {"index": "column_name"}, inplace=True)
df_null_analysis.head()
```

	column_name	null_values	non_null_values
0	city	0	778094
1	confidence	0	778094
2	nThumbsUp	776723	1371
3	street	14073	764021
4	uuid	0	778094

Making the stacked bar plot

```
# making the df into a long format
df_null_analysis_melted = df_null_analysis.melt(id_vars = ["column_name"],
 ↪  value_vars = ["null_values", "non_null_values"], var_name =
 ↪  "null_or_non_null", value_name = "count")

# making the plot
alt.Chart(df_null_analysis_melted).mark_bar().encode(
    alt.X("column_name:N"),
    alt.Y("count:Q"),
    color = "null_or_non_null:N",
).properties(
    width = 400,
    height = 200
).configure_axisX(
    labelAngle=-45
)
```



The variables which have missing values are nThumbsUp, street and subtype.

```
# getting share of missing values for nThumbsUp

nThumbsUp = df_null_analysis[df_null_analysis["column_name"] == "nThumbsUp"]

missing_percent = (nThumbsUp["null_values"].iloc[0] /
 ↪  (nThumbsUp["null_values"].iloc[0] +
 ↪  nThumbsUp["non_null_values"].iloc[0])) * 100

print(f"Of these the variable with the most missing values is nThumbsUp with
 ↪  approximately {missing_percent} percent values missing.")
```

```
Of these the variable with the most missing values is nThumbsUp with
approximately 99.82380020922922 percent values missing.
```

3.

a.

```
# printing unique values for the columns type and subtype
print(f"The unique values in the type column are{data["type"].unique()}")
print(f"The unique values in the subtype column
 ↪  are{data["subtype"].unique()}")
```

```
The unique values in the type column are['JAM' 'ACCIDENT' 'ROAD_CLOSED'
'HAZARD']
The unique values in the subtype column 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']
```

```
# getting subtypes which are NAs
na_subtypes = data[data["subtype"].isna()]

# now from this table, we get the number of types which are present
types_with_na_subtypes = len(na_subtypes["type"].unique())

print(f"There are {types_with_na_subtypes} types that have subtypes with
 ↪  NAs.")
```

```
There are 4 types that have subtypes with NAs.
```

From the unique values, we can see that the one type which has subtypes that can have sub-subtypes seems to be hazard alone. For example, the subtype **HAZARD_WEATHER** can have sub-subtypes like Fog, Flood, Heavy Snow, Hail etc. Further, the subtype **HAZARD_ON_ROAD** can have sub-subtypes like Car Stopped, Construction, Emergency Vehicle, Ice, Object, Pot Hole, Traffic Light Fault, Lane Closed, Road Kill and Other (this will contain observations in the HAZARD_ON_ROAD subtype). Further the subtype **HAZARD_ON_SHOULDER** can have sub-subtypes like Car Stopped, Animals, Missing Sign and Other (this will contain observations in the HAZARD_ON_SHOULDER subtype).

The below codes before the bulletted list are for my reference only.

```
accident_only = data[data["type"] == "ACCIDENT"]
# getting the subtypes for accident
accident_only_subtype = accident_only["subtype"].unique()
```

```
hazard_only = data[data["type"] == "HAZARD"]
# getting the subtypes for hazard
hazard_only_subtype = hazard_only["subtype"].unique()
```

```
jam_only = data[data["type"] == "JAM"]
# getting the subtypes for hazard
jam_only_subtype = jam_only["subtype"].unique()
```

```
road_closed_only = data[data["type"] == "ROAD_CLOSED"]
# getting the subtypes for hazard
road_closed_only_subtype = road_closed_only["subtype"].unique()
```

b.

```
# Adding the hierarchy in a dictionary
hierarchy = {
    "Accident": ["Major", "Minor"],
    "Jam": ["Heavy Traffic", "Moderate Traffic", "Stand-Still Traffic",
     ↪  "Light Traffic"],
    "Road Closed": ["Event", "Construction", "Hazard"],
    "Hazard": {
        "Weather": ["Fog", "Flood", "Heavy Snow", "Hail"],
        "On Road": [
            "Car Stopped", "Construction", "Emergency Vehicle", "Ice",
            "Object", "Pot Hole", "Traffic Light Fault", "Lane Closed", "Road
             ↪  Kill", "Other"],
        "On Shoulder": ["Car Stopped", "Animals", "Missing Sign", "Other"]
    }
}

def print_hierarchy(hierarchy, indent = 0):
    for key, value in hierarchy.items():
        if isinstance(value, dict):
            print("  " * indent + f"- **{key}**")
            print_hierarchy(value, indent + 1)
        elif isinstance(value, list):
            print("  " * indent + f"- **{key}**")
            for item in value:
                print("  " * (indent + 1) + f"- {item}")
```

```
# Print the hierarchy
print_hierarchy(hierarchy)
```

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

Attrition: I used ChatGPT to understand the code to print out a bulletted list.

c.

I think we should keep the NA subtypes as although we have types for alerts/jams/irregularities, it is very much possible there are some cases which don't fall directly in the existing subtypes. Hence, just because they don't fall in the pre-defined subtypes, doesn't mean that these can be important observations too.

```
# coding NAs to "Unclassified"
data["subtype"] = data["subtype"].fillna("Unclassified")
```

4.
5.

```
# Defining a pandas df with the given columns
crosswalk_df = pd.DataFrame(columns=["type", "subtype", "updated_type",
 ↪  "updated_subtype", "updated_subsubtype"])
```

2.

Creating the crosswalk table

Note: I created this table manually as I felt that would be the easiest way to do this.

```
crosswalk_dict = {"type": ["ACCIDENT" , "ACCIDENT" , "ACCIDENT", "JAM", "JAM"
 ↪  , "JAM" , "JAM" , "JAM" , "ROAD_CLOSED" , "ROAD_CLOSED" , "ROAD_CLOSED" ,
 ↪  "ROAD_CLOSED","HAZARD","HAZARD","HAZARD","HAZARD","HAZARD","HAZARD","HAZARD","HAZARD","HA
                  "subtype" : ['Unclassified', 'ACCIDENT_MAJOR',
                   ↪  'ACCIDENT_MINOR', 'Unclassified', 'JAM_HEAVY_TRAFFIC',
                   ↪  'JAM_MODERATE_TRAFFIC',
       'JAM_STAND_STILL_TRAFFIC', 'JAM_LIGHT_TRAFFIC','Unclassified',
        ↪  'ROAD_CLOSED_EVENT', 'ROAD_CLOSED_CONSTRUCTION',
       'ROAD_CLOSED_HAZARD', 'Unclassified', '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'],
                  "updated_type" : ["Accident", "Accident" , "Accident",
                   ↪  "Jam" , "Jam" , "Jam" , "Jam", "Jam", "Road Closed",
                   ↪  "Road Closed" ,"Road Closed", "Road Closed","Hazard",
                   ↪  "Hazard", "Hazard", "Hazard", "Hazard", "Hazard",
                   ↪  "Hazard", "Hazard", "Hazard", "Hazard", "Hazard",
                   ↪  "Hazard", "Hazard", "Hazard", "Hazard", "Hazard"],
```

```
                    "updated_subtype": ["Unclassified" , "Major" , "Minor"
                     ↪  ,"Unclassified", "Heavy Traffic", "Moderate Traffic" ,
                     ↪  "Stand-Still Traffic" , "Light Traffic", "Unclassified"
                     ↪  , "Event" , "Construction" , "Hazard","Other","On
                     ↪  Road", "On Road", "On Road", "On Road", "On Road", "On
                     ↪  Road", "On Road","On Shoulder","On Shoulder",
                     ↪  "Weather","Weather","On Road","Weather","On Road","On
                     ↪  Shoulder","On Shoulder","On
                     ↪  Shoulder","Weather","Weather"],
                    "updated_subsubtype" : ["Na" , "Na" , "Na", "Na" , "Na" ,
                     ↪  "Na","Na" , "Na", "Na" , "Na","Na" ,
                     ↪  "Na","Other","Other","Car Stopped", "Construction",
                     ↪  "Emergency Vehicle", "Ice","Object","Pot Hole",
                     ↪  "Traffic Light Fault","Other","Car Stopped", "Other",
                     ↪  "Flood","Lane Closed", "Fog", "Road Kill","Animals",
                     ↪  "Missing Sign", "Heavy Snow","Hail"]}

crosswalk_df = pd.DataFrame(crosswalk_dict)
crosswalk_df.head()
```

	type	subtype	updated_type	updated_subtype	updated_subsubtype
0	ACCIDENT	Unclassified	Accident	Unclassified	Na
1	ACCIDENT	ACCIDENT_MAJOR	Accident	Major	Na
2	ACCIDENT	ACCIDENT_MINOR	Accident	Minor	Na
3	JAM	Unclassified	Jam	Unclassified	Na
4	JAM	JAM_HEAVY_TRAFFIC	Jam	Heavy Traffic	Na

3.

```
# merging the crosswalk with the original dataset
merged_data = pd.merge(data , crosswalk_df , on = ["type" , "subtype"])

# looking at the number of rows for which updated type is Accident and the
 ↪  updated_subtype is Unclassified

accident_unclassified = merged_data[
    (merged_data["updated_type"] == "Accident") &
    (merged_data["updated_subtype"] == "Unclassified")
]
print(f"The number of rows for Accident - Unclassified are
 ↪  {accident_unclassified.shape[0]}.")
```

```
The number of rows for Accident - Unclassified are 24359.
```

4.

```
# checking if the crosswalk and the merged dataset have the same values for
 ↪  type and subtype

# for merged
unique_type_merged = merged_data["type"].unique()
unique_subtype_merged = merged_data["subtype"].unique()

# for crosswalk
unique_type_crosswalk_df = crosswalk_df["type"].unique()
unique_subtype_crosswalk_df = crosswalk_df["subtype"].unique()

# Using sets as the comparing the lists might not give us the corrct answer
 ↪  if the lists are ordered differently
if set(unique_type_merged) == set(unique_type_crosswalk_df):
    print("The type values are the same in the merged and crosswalk df.")
else:
    print("The type values are not the same in the merged and crosswalk df.")

# Compare unique "subtype" values
if set(unique_subtype_merged) == set(unique_subtype_crosswalk_df):
    print("The subtype values are the same in the merged and crosswalk df.")
else:
    print("The subtype values are not the same in the merged and crosswalk
     ↪  df.")
```

```
The type values are the same in the merged and crosswalk df.
The subtype values are the same in the merged and crosswalk df.
```

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

1.

a.

Prompt for ChatGPT: I have a dataframe which has a column “geo” which holds coordinates data, but they are stored in a string that represents the Well-Known Text representation of the point. Write me a regex function to extract the latitude and longitude from this and make two separate columns for latitude and longitude.

```
import re

# function to extract latitude and longitude
def extract_lat_lon(wkt):
    match = re.match(r"POINT\(([-\d\.]+) ([-\d\.]+)\)", wkt)
    if match:
        lon, lat = match.groups()
        return float(lat), float(lon)
    return None, None

# apply the function to the dataframe
merged_data[["latitude", "longitude"]] = merged_data["geo"].apply(
    lambda x: pd.Series(extract_lat_lon(x))
)
```

b.

Binning latitude and longitude or essentially, rounding them to 2 digits

```
merged_data["binned_latitude"] = merged_data["latitude"].round(2)
merged_data["binned_longitude"] = merged_data["longitude"].round(2)

# to get the most number of observations in a combination
binned_lat_long_count = merged_data.groupby(["binned_latitude",
 ↪  "binned_longitude"]).size().reset_index(name = "count")

# getting the maximum number
binned_lat_long_count = binned_lat_long_count.sort_values(by = "count",
 ↪  ascending = False)
print(f"The combination which has the most number of observations is
 ↪  latitude:{binned_lat_long_count.iloc[0]["binned_latitude"]} and
 ↪  longitude: {binned_lat_long_count.iloc[0]["binned_longitude"]}. The
 ↪  number of observations in this combination are
 ↪  {binned_lat_long_count.iloc[0]["count"]}.")
```

```
The combination which has the most number of observations is latitude:41.88
and longitude: -87.65. The number of observations in this combination are
21325.0.
```

c.

The level of aggregation of this dataset is the type level and further the subtypes of each type level.

```
types_and_subtypes = {
    "Accident": ['Unclassified', 'Major', 'Minor'],
    "Jam": ['Unclassified', 'Heavy Traffic', 'Moderate Traffic', 'Stand-Still
     ↪  Traffic', 'Light Traffic'],
    "Road Closed": ['Unclassified', 'Event', 'Construction', 'Hazard'],
    "Hazard": ['Other', 'On Road', 'On Shoulder', 'Weather']
}

# Function to get top 10 latitude-longitude bins for each type-subtype
 ↪  combination
def get_top_bins_for_each_type(df, types_and_subtypes):

    """
    This function gets the top 10 lat and longs for each type-subtype
 ↪  combination
    """
    results = []

    # Iterate over each type and its subtypes
    for type_name, subtypes in types_and_subtypes.items():
        for subtype in subtypes:
            # Filter the dataframe based on the current type and subtype
            filtered_df = df[(df["updated_type"] == type_name) &
 ↪  (df["updated_subtype"] == subtype)]

            # Group by latitude-longitude bins and count the number of
             ↪  observations
            grouped_df = filtered_df.groupby(["binned_latitude",
 ↪  "binned_longitude"]).size().reset_index(name = "count")

            # Sort by alert_count in descending order
            sorted_df = grouped_df.sort_values(by = "count", ascending =
 ↪  False)

            # Select the top 10 latitude-longitude bins
            top_10_bins = sorted_df.head(10)

            # Add the type and subtype to the result
            top_10_bins["type"] = type_name
            top_10_bins["subtype"] = subtype

            # Append to results
```

```
            results.append(top_10_bins)

    # Combine all results into a single DataFrame
    final_df = pd.concat(results, ignore_index = True)

    return final_df

# Get the top bins for each type-subtype combination
top_alerts_map = get_top_bins_for_each_type(merged_data, types_and_subtypes)

top_alerts_map.head()

# saving as a csv in the folder
save_path =
 ↪  "/Users/kishikamahajan/Desktop/GitHub/Problem_Set_6/top_alerts_map/top_alerts_map.csv"
top_alerts_map.to_csv(save_path, index = False)
```

```
/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_60650/4015636622.py:32:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus

/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_60650/4015636622.py:33:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus

/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_60650/4015636622.py:32:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus

/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_60650/4015636622.py:33:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus

/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_60650/4015636622.py:32:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus

/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_60650/4015636622.py:33:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus

/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_60650/4015636622.py:32:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

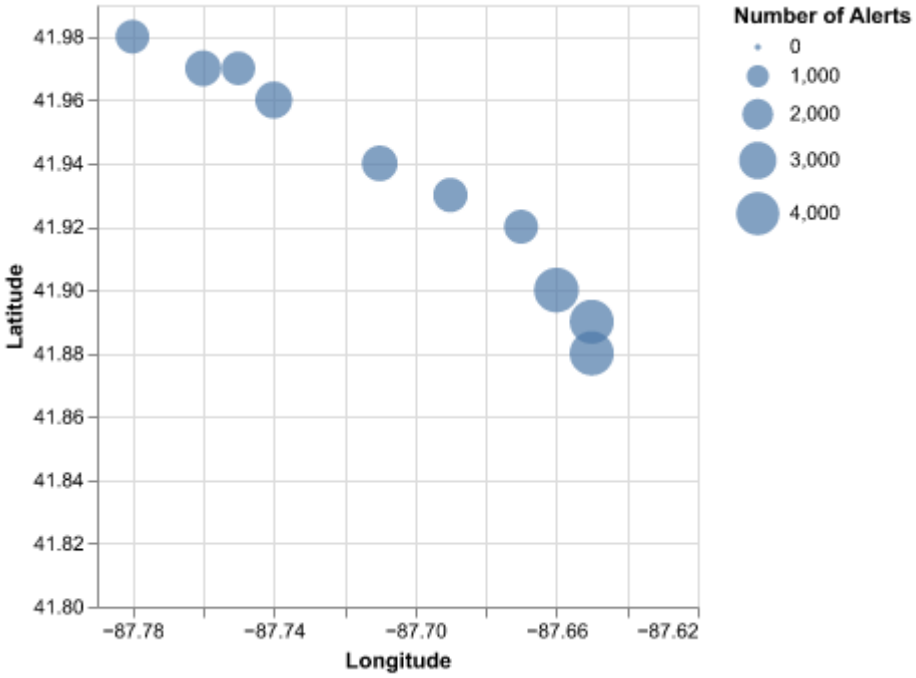
```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus

/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_60650/4015636622.py:33:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus

/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_60650/4015636622.py:32:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus

/var/folders/w0/cccpmsxn11z4l65wxv069v7m0000gn/T/ipykernel_60650/4015636622.py:33:
SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation:
https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
```

```
print(f"The number of rows this dataframe has are
 ↪  {top_alerts_map.shape[0]}.")
```

```
The number of rows this dataframe has are 155.
```

2.

```
# Subsetting only Jam
jam_top_10 = top_alerts_map[top_alerts_map["type"] == "Jam"]
jam_heavy_traffic = jam_top_10[jam_top_10["subtype"] == "Heavy Traffic"]

# Create a scatter plot with Altair
scatter_plot = alt.Chart(jam_heavy_traffic).mark_circle().encode(
    x=alt.X(
        "binned_longitude:Q",
        scale = alt.Scale(domain = [-87.79, -87.62]),
        title = "Longitude",
    ),
    y=alt.Y(
        "binned_latitude:Q",
        scale = alt.Scale(domain = [41.8, 41.99]),
        title = "Latitude",
    ),
    size=alt.Size(
        "count:Q",
        scale = alt.Scale(range = [10, 500]),
        title = "Number of Alerts"
    )
).project(
    type = "equirectangular"
)
scatter_plot
```



3.

a.

Dowanloading the file directly from python

```
import requests

url =
 ↪  "https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON"
output_path =
 ↪  "/Users/kishikamahajan/Desktop/GitHub/Problem_Set_6/top_alerts_map/chicago-boundaries.geo

response = requests.get(url)
with open(output_path, "wb") as f:
  f.write(response.content)
```

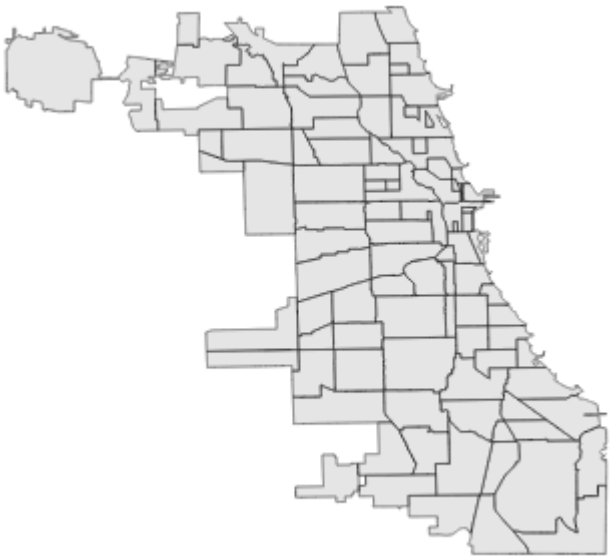
b.

```
# loading the file
file_path =
 ↪  "/Users/kishikamahajan/Desktop/GitHub/Problem_Set_6/top_alerts_map/chicago-boundaries.geo
```

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

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

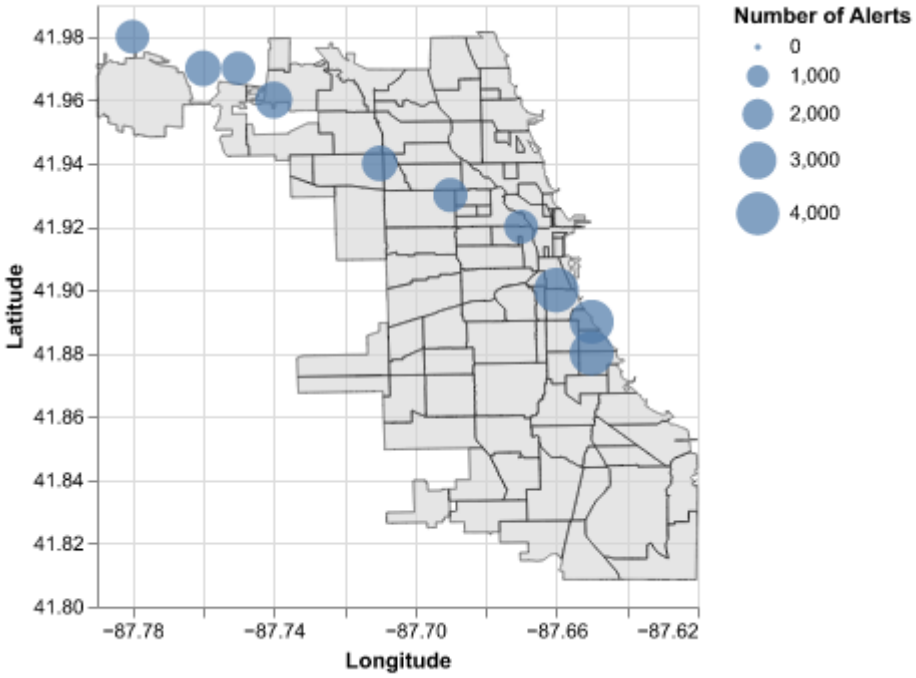
4.

```
# Making the Chicago boundaries
map_chart = alt.Chart(geo_data).mark_geoshape(
    fill = "lightgray",
    stroke = "black",
    opacity = 0.6
).project(
    type = "equirectangular"
)
map_chart
```



Layering the plot

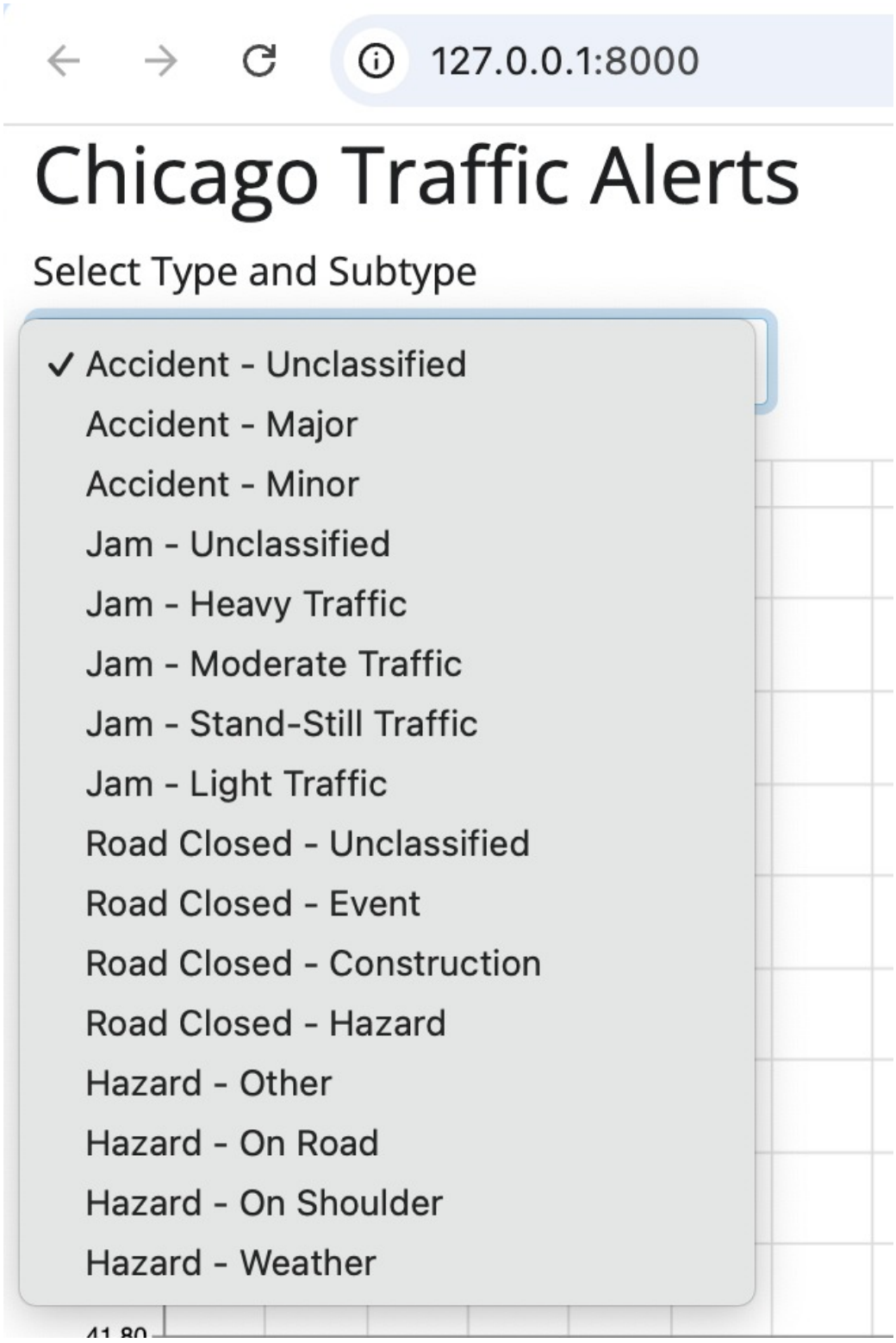
```
combined_plot = map_chart + scatter_plot

combined_plot
```



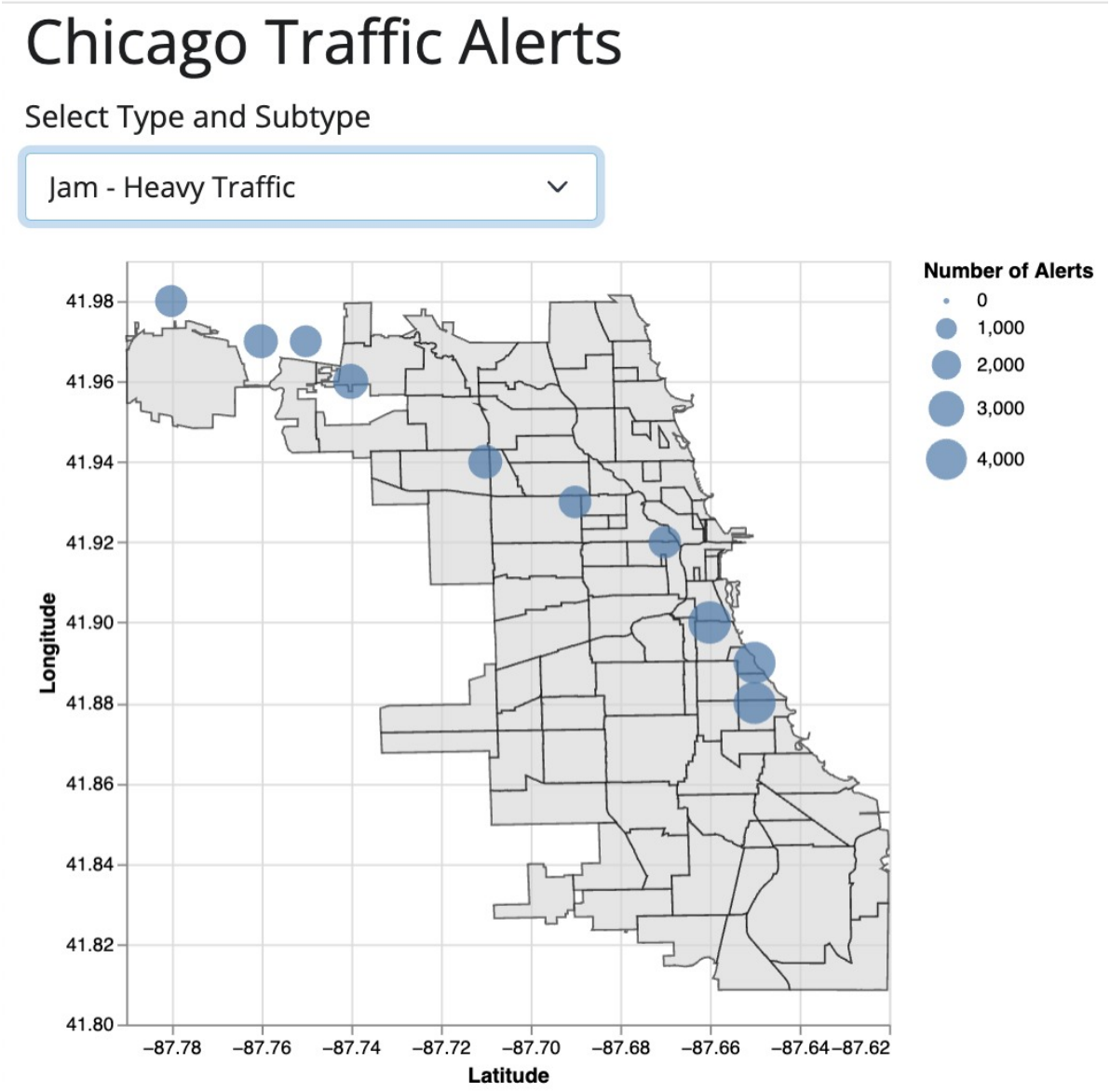
5.

a.

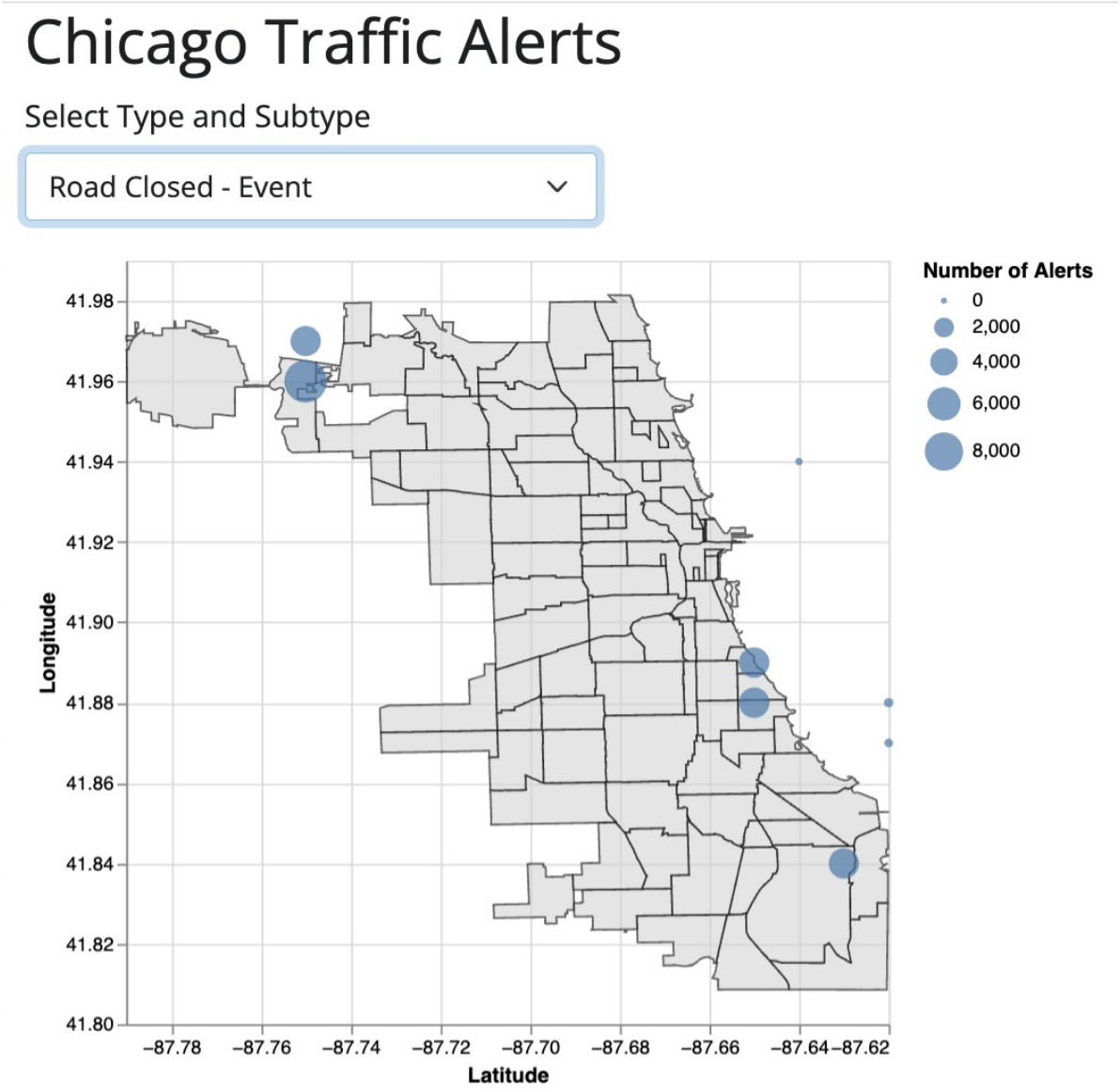
The total type-subtype combinations in my drop down menu are 16.


127.0.0.1:8000
Chicago Traffic Alerts
Select Type and Subtype
✓ Accident - Unclassified
Accident - Major
Accident - Minor
Jam - Unclassified
Jam - Heavy Traffic
Jam - Moderate Traffic
Jam - Stand-Still Traffic
Jam - Light Traffic
Road Closed - Unclassified
Road Closed - Event
Road Closed - Construction
Road Closed - Hazard
Hazard - Other
Hazard - On Road
Hazard - On Shoulder
Hazard - Weather

b.


Chicago Traffic Alerts
Select Type and Subtype
Jam - Heavy Traffic
Number of Alerts
0
1,000
2,000
3,000
4,000
Longitude
41.98
41.96
41.94
41.92
41.90
41.88
41.86
41.84
41.82
41.80
−87.78
−87.76
−87.74
−87.72
−87.70
−87.68
−87.66
−87.64
−87.62
Latitude

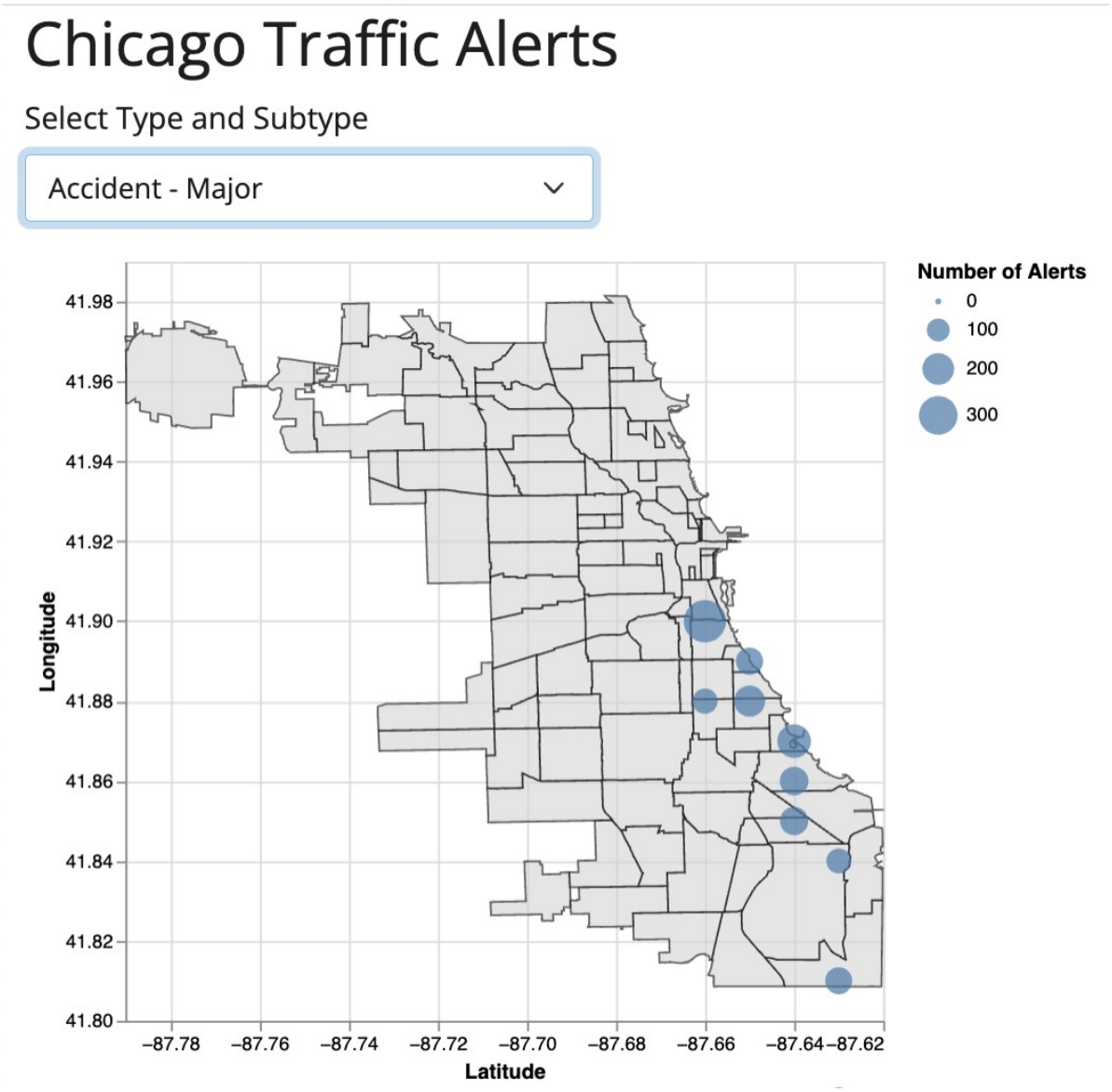
c.



As can be seen, the most road closures due to events happen around O'Hare and Nordwood Park.

d.

Another question that can be answered is where is where do maximum major accidents happen.



As can be seen, the maximum number of major accidents happen in the central-eastern part of Chicago.

e. Another column that can be added to the dashboard is what time of the day the alerts were made so that a deeper analysis of the most vulnerable times can be done.

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

1.

a. ts shows the timestamp of the reported alert. I don't think it will be particularly useful to collapse the data by the exact time (upto minutes and seconds) at which the alert was reported. If anything, it will just be useful to know during what time of the day (morning, afternoon, evening or night) were the most alerts reported to understand the most vulnerable times during the day.

b.

```
import datetime as datetime
# creating a new column called hour which is extracted from each ts
# converting the column to datetime
merged_data["ts"] = pd.to_datetime(merged_data["ts"])

# extracting the hour
merged_data["hour"] = merged_data["ts"].dt.strftime("%H:00")
```

```
# Group by latitude, longitude, hour, type, and subtype
top_alerts_map_byhour = merged_data.groupby(
    ['binned_latitude', 'binned_longitude', 'hour', 'updated_type',
 ↪  'updated_subtype']
).size().reset_index(name='count')

# Sort by count in descending order for each group
top_alerts_map_byhour = top_alerts_map_byhour.sort_values(by='count',
 ↪  ascending=False)

# Get the top 10 for each hour, updated_type, and updated_subtype combination
top_alerts_map_byhour = top_alerts_map_byhour.groupby(['hour',
 ↪  'updated_type', 'updated_subtype']).head(10)

top_alerts_map_byhour = top_alerts_map_byhour.rename(columns={
    'updated_type': 'type',
    'updated_subtype': 'subtype'
})

# saving as a csv in the folder
save_path2 =
 ↪  "/Users/kishikamahajan/Desktop/GitHub/Problem_Set_6/top_alerts_map_byhour/top_alerts_map_
```

```
top_alerts_map_byhour.to_csv(save_path2, index = False)

top_alerts_map_byhour.head()
```

	binned_latitude	binned_longitude	hour	type	subtype	count
32701	41.88	-87.65	22:00	Jam	Stand-Still Traffic	760
32687	41.88	-87.65	21:00	Jam	Stand-Still Traffic	740
37159	41.90	-87.66	21:00	Jam	Stand-Still Traffic	612
37171	41.90	-87.66	22:00	Jam	Stand-Still Traffic	579
34917	41.89	-87.65	22:00	Jam	Stand-Still Traffic	574

```
print(f"The number of rows this dataset has are
 ↪  {top_alerts_map_byhour.shape[0]}.")
```

```
The number of rows this dataset has are 3202.
```

c.

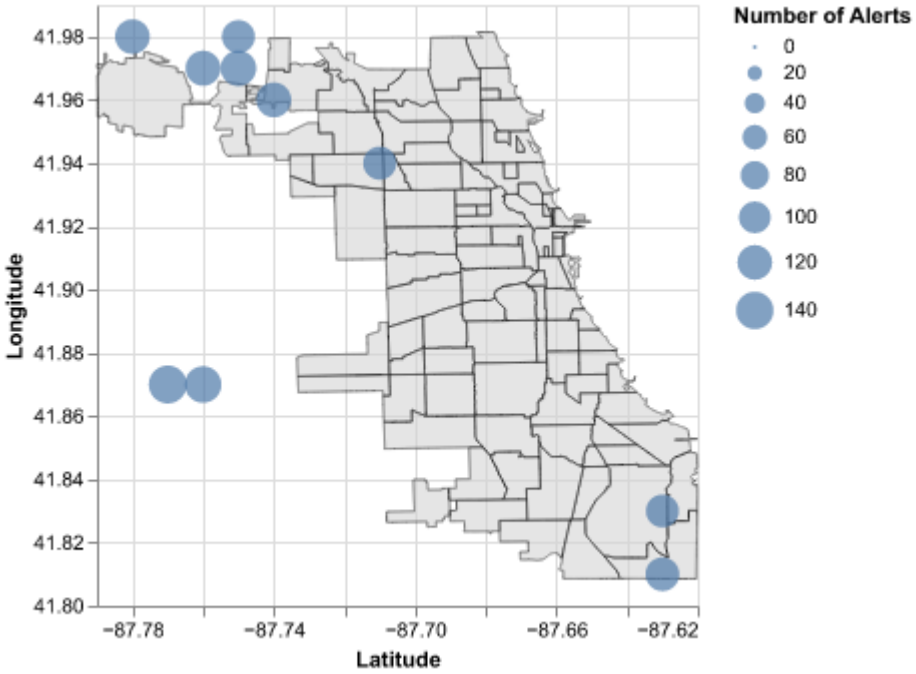
Subsetting jam-heavy traffic

```
# Subsetting only Jam
jam_byhour = top_alerts_map_byhour[top_alerts_map_byhour["type"] == "Jam"]
jam_heavy_traffic_byhour = jam_byhour[jam_byhour["subtype"] == "Heavy
 ↪  Traffic"]
```

Choosing 3 times of the day

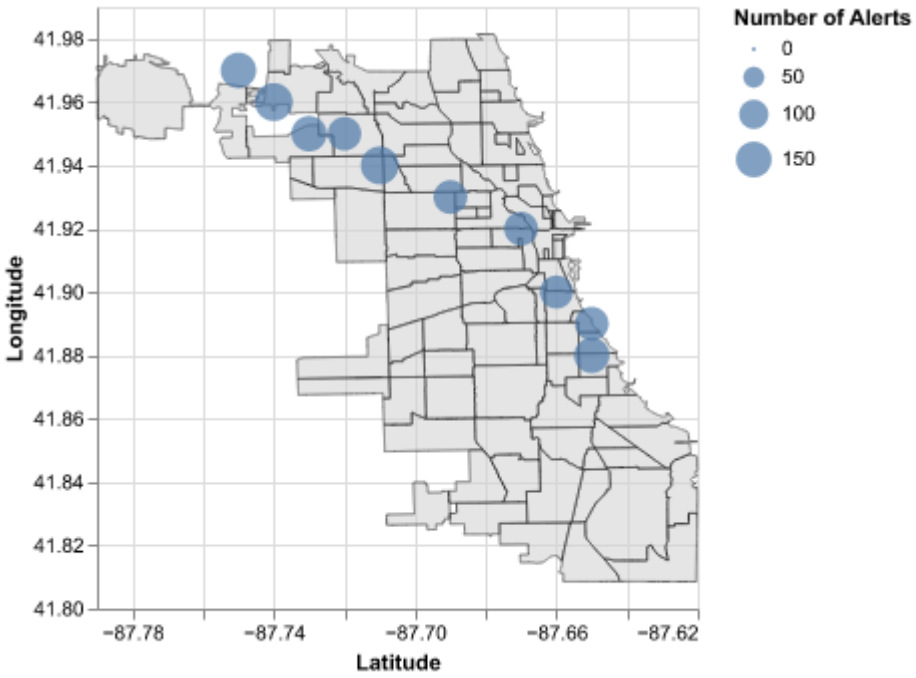
```
# for the purposes of morning, we can pick 11:00

morning_jam_heavy_traffic =
 ↪  jam_heavy_traffic_byhour[jam_heavy_traffic_byhour["hour"] == "11:00"]

# plotting this on the map and layering it
scatter_plot_morning =
 ↪  alt.Chart(morning_jam_heavy_traffic).mark_circle().encode(
    x=alt.X(
        "binned_longitude:Q",
        scale = alt.Scale(domain = [-87.79, -87.62]),
        title = "Latitude",
```

```
    ),
    y=alt.Y(
        "binned_latitude:Q",
        scale = alt.Scale(domain = [41.8, 41.99]),
        title = "Longitude",
    ),
    size=alt.Size(
        "count:Q",
        title = "Number of Alerts"
    )
).project(
    type = "equirectangular"
)

combined_morning = map_chart + scatter_plot_morning
combined_morning
```



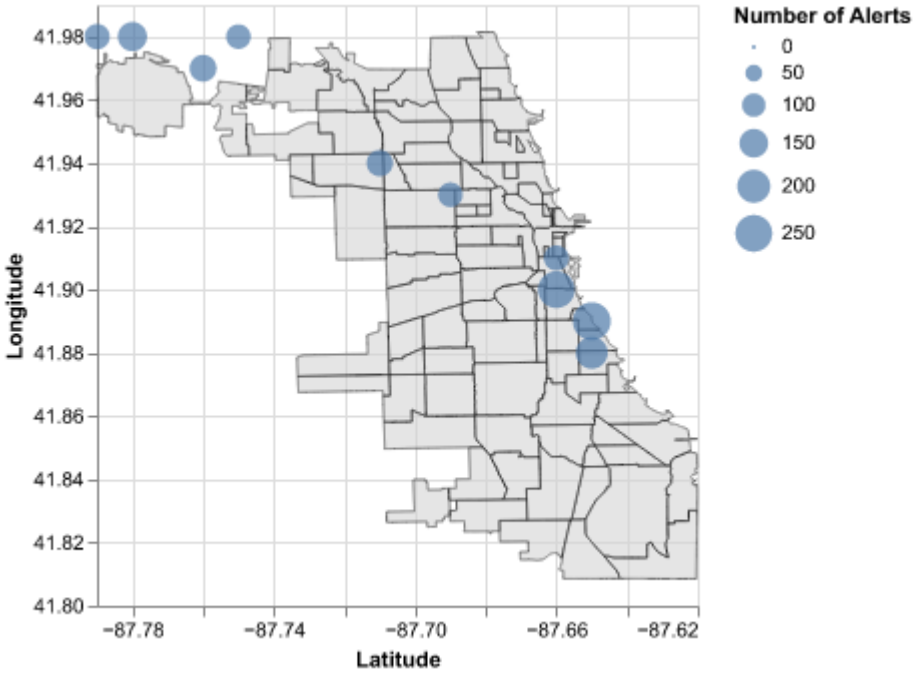

```
# for the purposes of afternoon, we can pick 14:00

afternoon_jam_heavy_traffic =
 ↪  jam_heavy_traffic_byhour[jam_heavy_traffic_byhour["hour"] == "14:00"]
```

```
# plotting this on the map and layering it
# plotting this on the map and layering it
scatter_plot_afternoon =
 ↪  alt.Chart(afternoon_jam_heavy_traffic).mark_circle().encode(
    x=alt.X(
        "binned_longitude:Q",
        scale = alt.Scale(domain = [-87.79, -87.62]),
        title = "Latitude",
    ),
    y=alt.Y(
        "binned_latitude:Q",
        scale = alt.Scale(domain = [41.8, 41.99]),
        title = "Longitude",
    ),
    size=alt.Size(
        "count:Q",
        title = "Number of Alerts"
    )
).project(
    type = "equirectangular"
)

combined_afternoon = map_chart + scatter_plot_afternoon
combined_afternoon
```



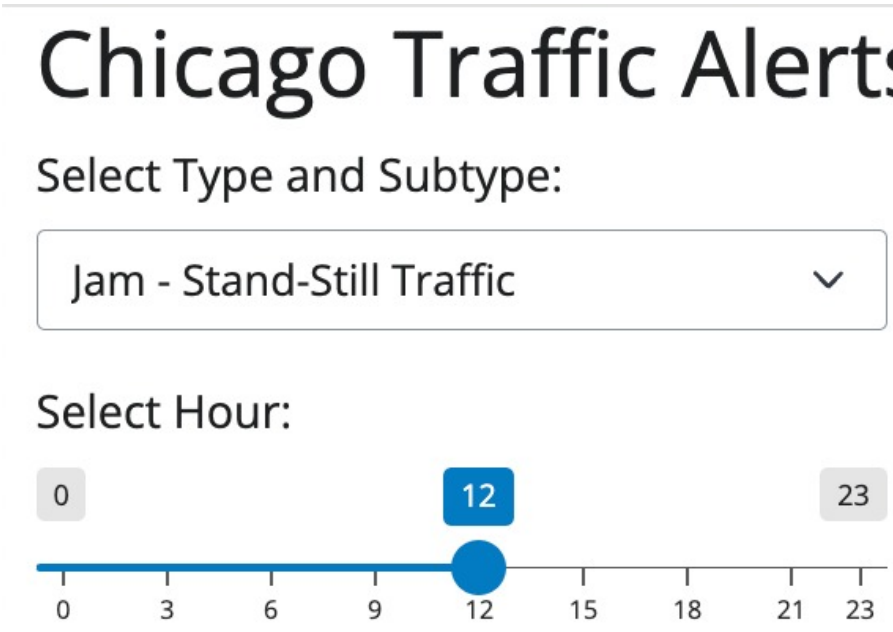

```
# for the purposes of night, we can pick 00:00

night_jam_heavy_traffic =
 ↪  jam_heavy_traffic_byhour[jam_heavy_traffic_byhour["hour"] == "00:00"]

# plotting this on the map and layering it
scatter_plot_night = alt.Chart(night_jam_heavy_traffic).mark_circle().encode(
    x=alt.X(
        "binned_longitude:Q",
        scale = alt.Scale(domain = [-87.79, -87.62]),
        title = "Latitude",
    ),
    y=alt.Y(
        "binned_latitude:Q",
        scale = alt.Scale(domain = [41.8, 41.99]),
        title = "Longitude",
    ),
    size=alt.Size(
        "count:Q",
        title = "Number of Alerts"
    )
).project(
```

```
    type = "equirectangular"
)

combined_night = map_chart + scatter_plot_night
combined_night
```

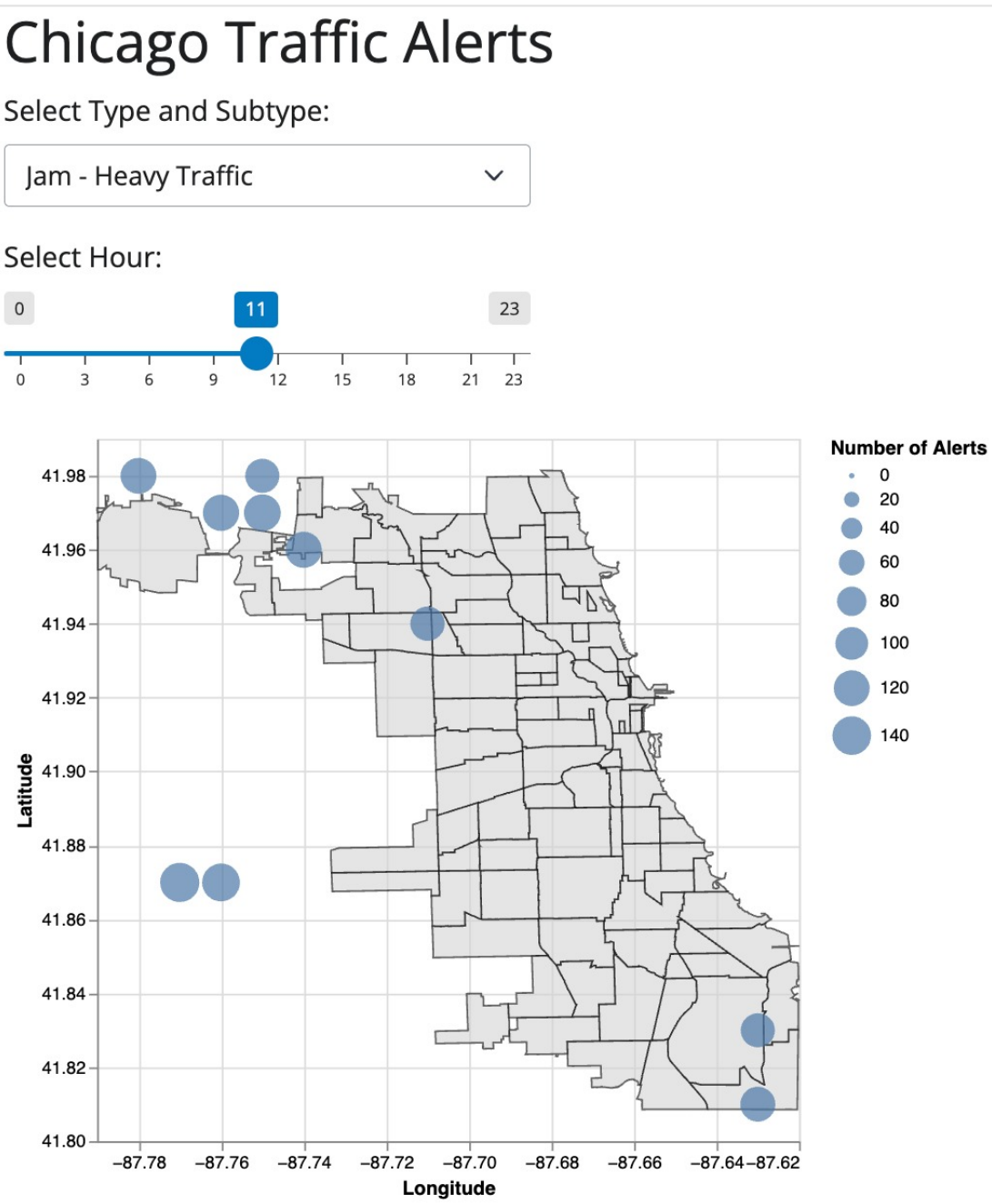


2.

a.


Chicago Traffic Alerts
Select Type and Subtype:
Jam - Stand-Still Traffic
Select Hour:
0
12
23
0
3
6
9
12
15
18
21
23

b.


Chicago Traffic Alerts
Select Type and Subtype:
Jam - Heavy Traffic
Select Hour:
0
11
23
0 3 6 9 12 15 18 21 23
Number of Alerts
0
20
40
60
80
100
120
140
Latitude
41.98
41.96
41.94
41.92
41.90
41.88
41.86
41.84
41.82
41.80
−87.78 −87.76 −87.74 −87.72 −87.70 −87.68 −87.66 −87.64 −87.62
Longitude

c.

Chicago Traffic Alerts

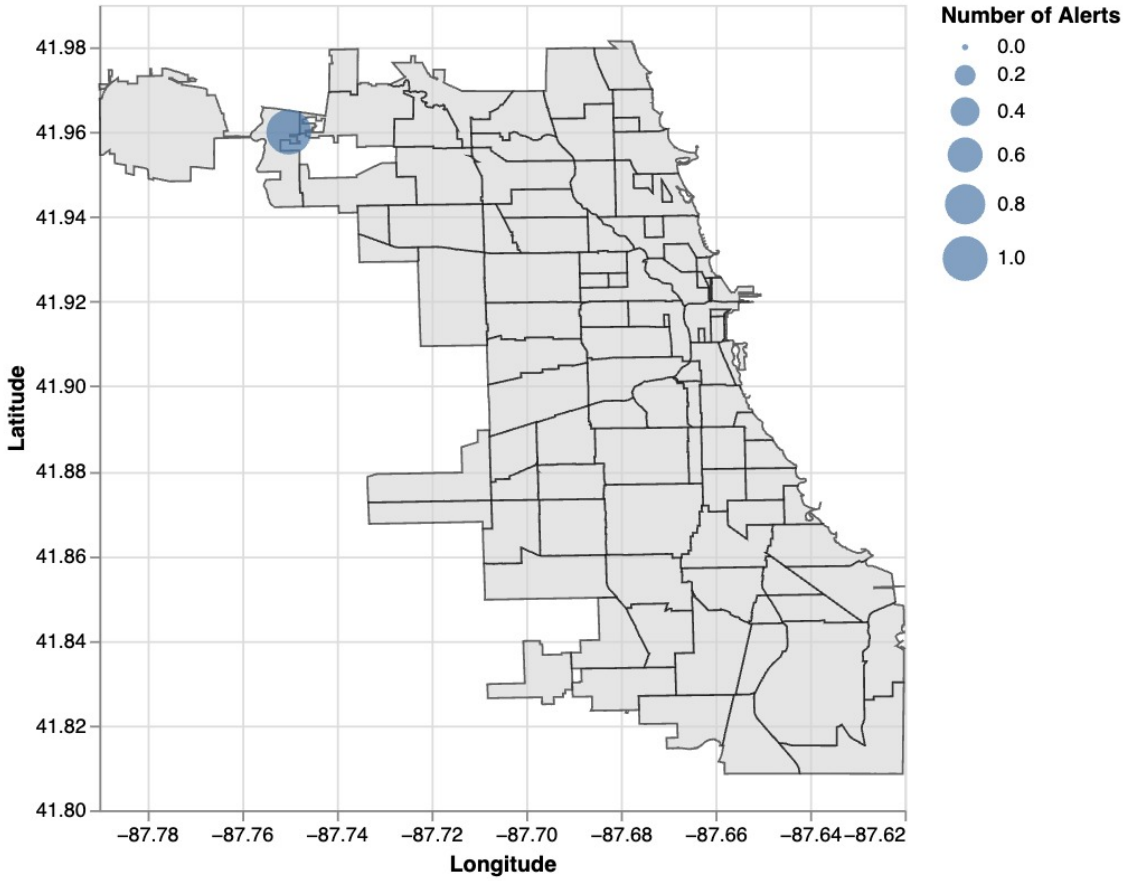
Select Type and Subtype:

Road Closed - Construction

Select Hour:

0 11 23

0 3 6 9 12 15 18 21 23



Chicago Traffic Alerts

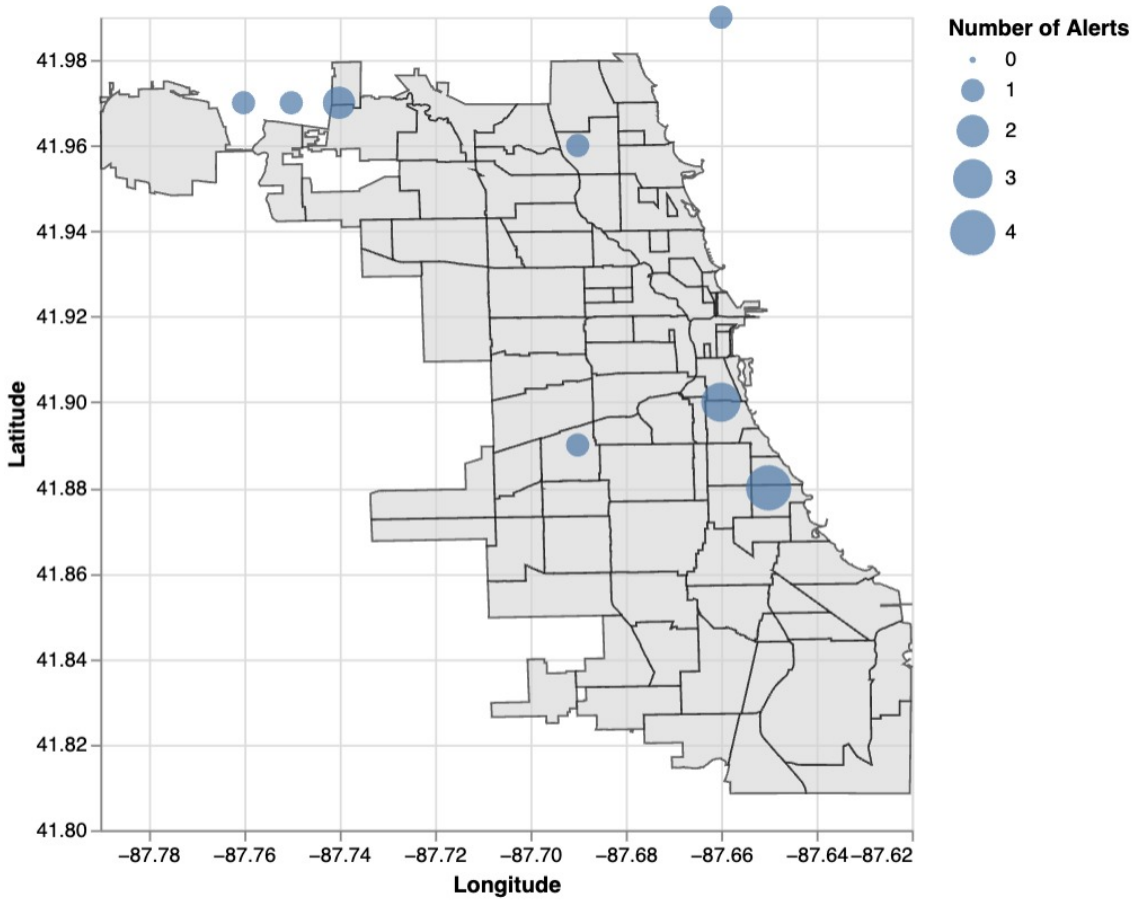
Select Type and Subtype:

Road Closed - Construction

Select Hour:

0 21

0 3 6 9 12 15 18 21 23



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

1.

a. I think it will be a good idea to collapse the data by the range of hours because it will make the analysis way easier, in that, it will be easier to determine which part of the day, morning, afternoon/evening or night sees the most amount of alerts.

I believe, it will be more informative for the user to do this type of analysis than for a particular single time in the day.

b.

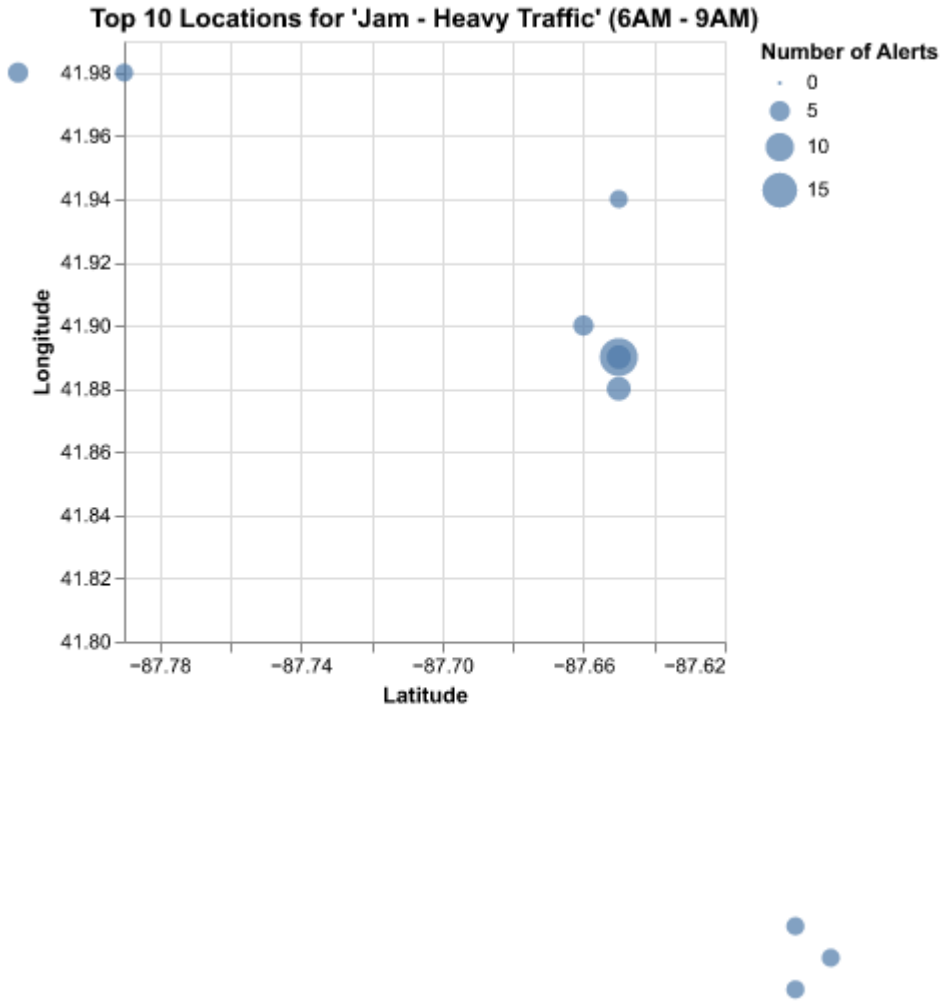
```
# subsetting the dataset for the time from 6 AM to 9 AM

subset_6am_9am =
 ↪  top_alerts_map_byhour[top_alerts_map_byhour["hour"].isin(["06:00",
 ↪  "07:00", "08:00", "09:00"])]
subset_6am_9am_heavy_traffic = subset_6am_9am[(subset_6am_9am["type"] ==
 ↪  "Jam") & (subset_6am_9am["subtype"] == "Heavy Traffic")]

# getting the 10 largest
subset_6am_9am_heavy_traffic_top10 =
 ↪  subset_6am_9am_heavy_traffic.nlargest(10, "count")
subset_6am_9am_heavy_traffic_top10.head()
```

	binned_latitude	binned_longitude	hour	type	subtype	count
34737	41.89	-87.65	06:00	Jam	Heavy Traffic	18
32520	41.88	-87.65	06:00	Jam	Heavy Traffic	7
34746	41.89	-87.65	07:00	Jam	Heavy Traffic	7
55343	41.98	-87.82	09:00	Jam	Heavy Traffic	5
37004	41.90	-87.66	06:00	Jam	Heavy Traffic	5

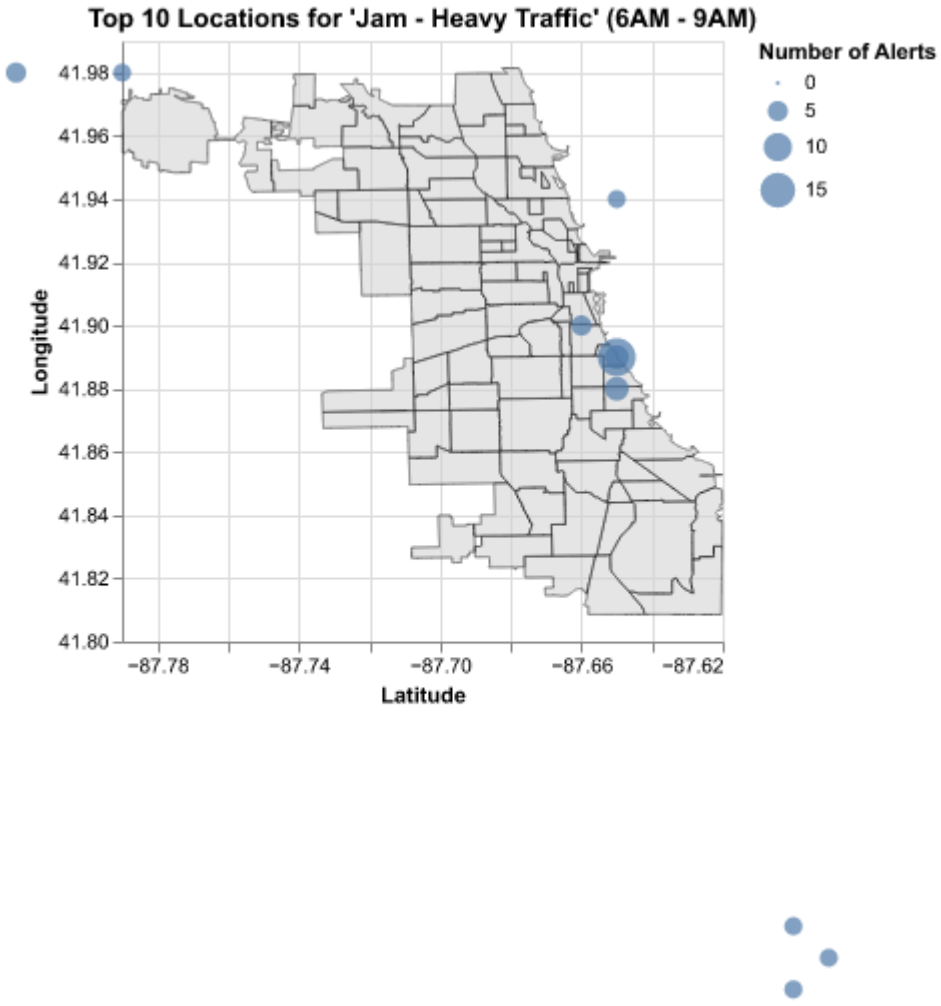
Creating the plot

```
# Create the Altair plot
top_10_range_scatterplot =
 ↪  alt.Chart(subset_6am_9am_heavy_traffic_top10).mark_circle().encode(
    x=alt.X(
```

```
        "binned_longitude:Q",
        scale = alt.Scale(domain = [-87.79, -87.62]),
        title = "Latitude"),
    y=alt.Y(
        "binned_latitude:Q",
        scale = alt.Scale(domain = [41.8, 41.99]),
        title = "Longitude"),
    size=alt.Size(
        "count:Q",
        title="Number of Alerts"
)).properties(
    title="Top 10 Locations for 'Jam - Heavy Traffic' (6AM - 9AM)"
).project(type = "equirectangular")
top_10_range_scatterplot
```

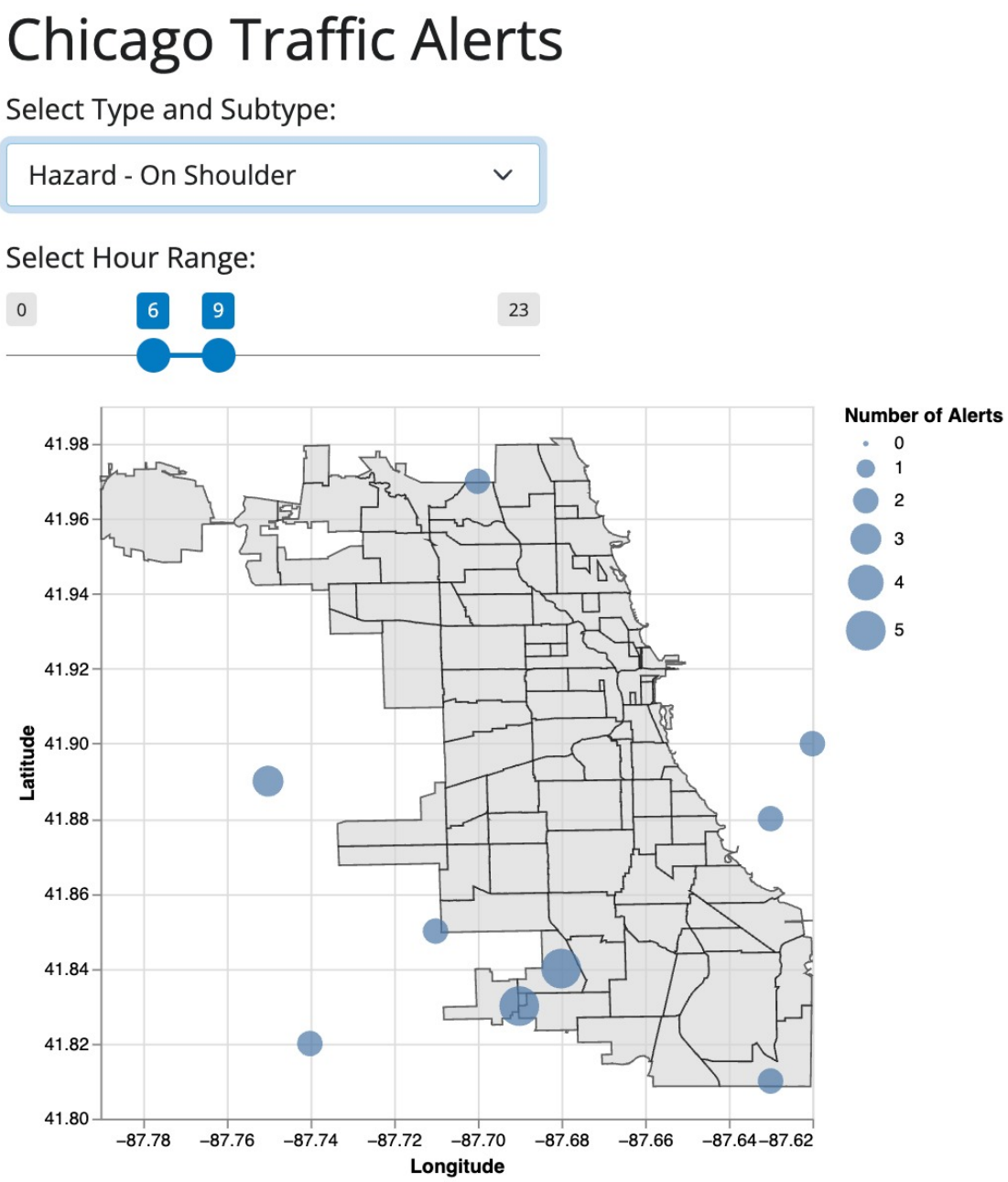



```
combined_range = map_chart + top_10_range_scatterplot
combined_range
```

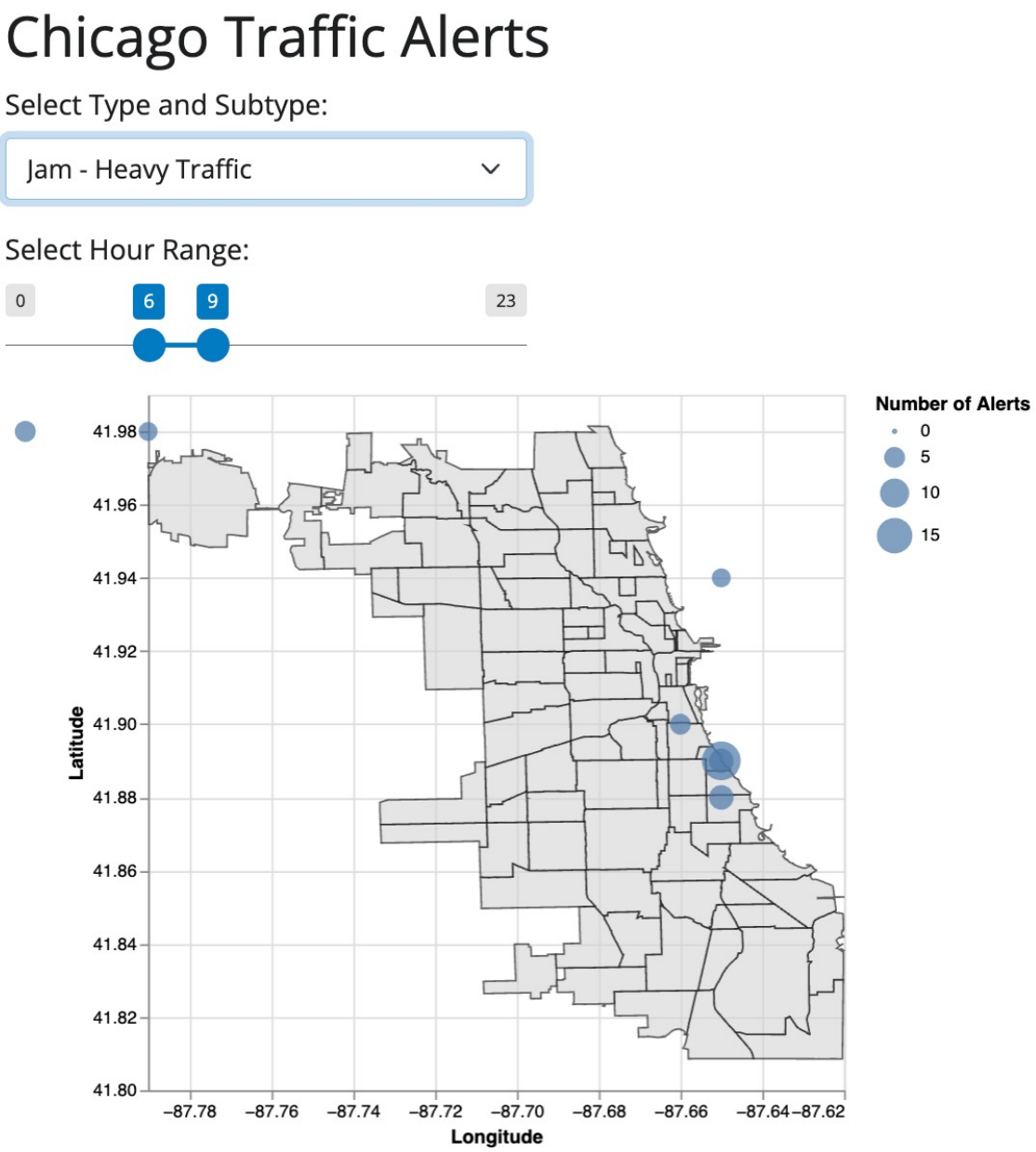

Top 10 Locations for 'Jam - Heavy Traffic' (6AM - 9AM)
Number of Alerts
0
5
10
15
Longitude
41.98
41.96
41.94
41.92
41.90
41.88
41.86
41.84
41.82
41.80
−87.78
−87.74
−87.70
−87.66
−87.62
Latitude

2.

a.

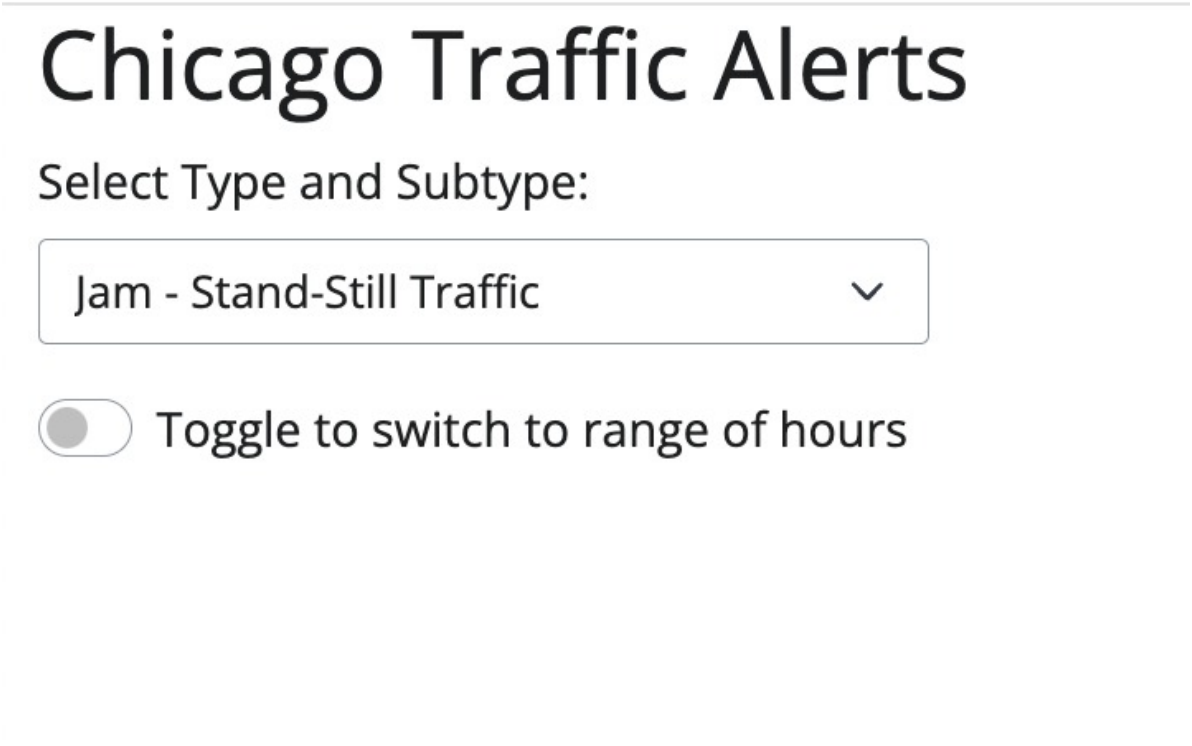

Chicago Traffic Alerts
Select Type and Subtype:
Hazard - On Shoulder
Select Hour Range:
0
6
9
23
Number of Alerts
0
1
2
3
4
5
Latitude
41.98
41.96
41.94
41.92
41.90
41.88
41.86
41.84
41.82
41.80
−87.78
−87.76
−87.74
−87.72
−87.70
−87.68
−87.66
−87.64
−87.62
Longitude

b.


Chicago Traffic Alerts
Select Type and Subtype:
Jam - Heavy Traffic
Select Hour Range:
0
6
9
23
Number of Alerts
0
5
10
15
Latitude
41.98
41.96
41.94
41.92
41.90
41.88
41.86
41.84
41.82
41.80
−87.78
−87.76
−87.74
−87.72
−87.70
−87.68
−87.66
−87.64
−87.62
Longitude

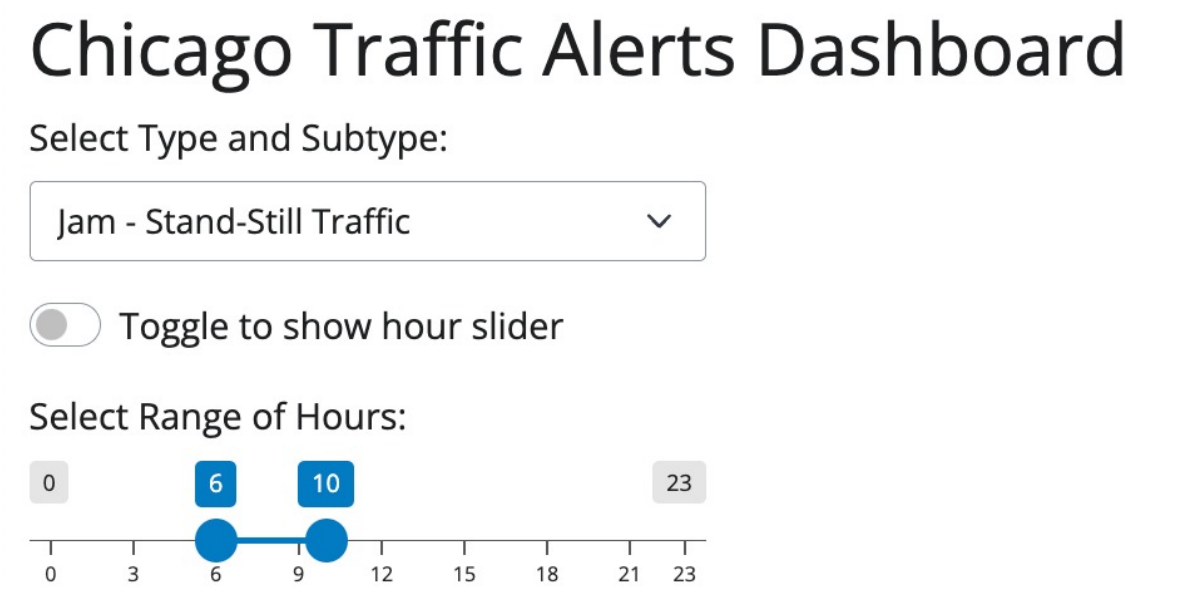
3.

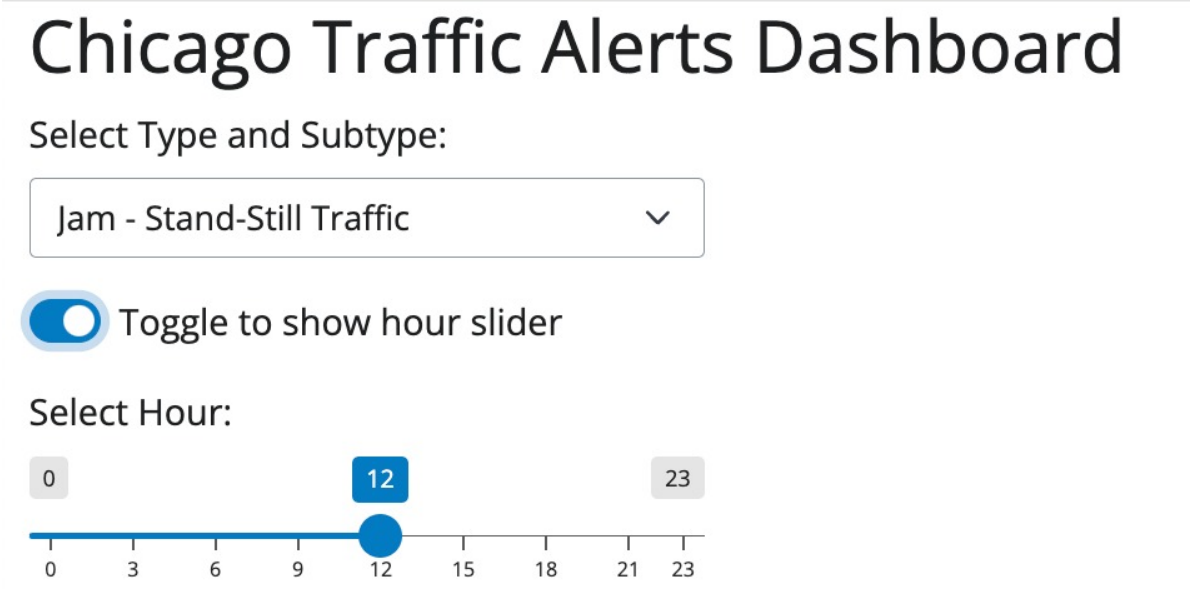
a.



The possible values for the switch button and in particular, for the input.switch_button can be True or False. Here, True means when the button is “on” and False means when the button is “off”. By default, the value is set to False.

b.


Chicago Traffic Alerts Dashboard
Select Type and Subtype:
Jam - Stand-Still Traffic
Toggle to show hour slider
Select Range of Hours:
0
6
10
23
0
3
6
9
12
15
18
21
23


Chicago Traffic Alerts Dashboard
Select Type and Subtype:
Jam - Stand-Still Traffic
Toggle to show hour slider
Select Hour:
0
12
23
0
3
6
9
12
15
18
21
23

c.

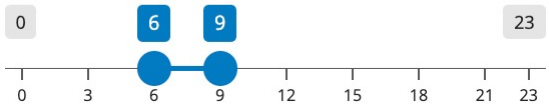
Chicago Traffic Alerts

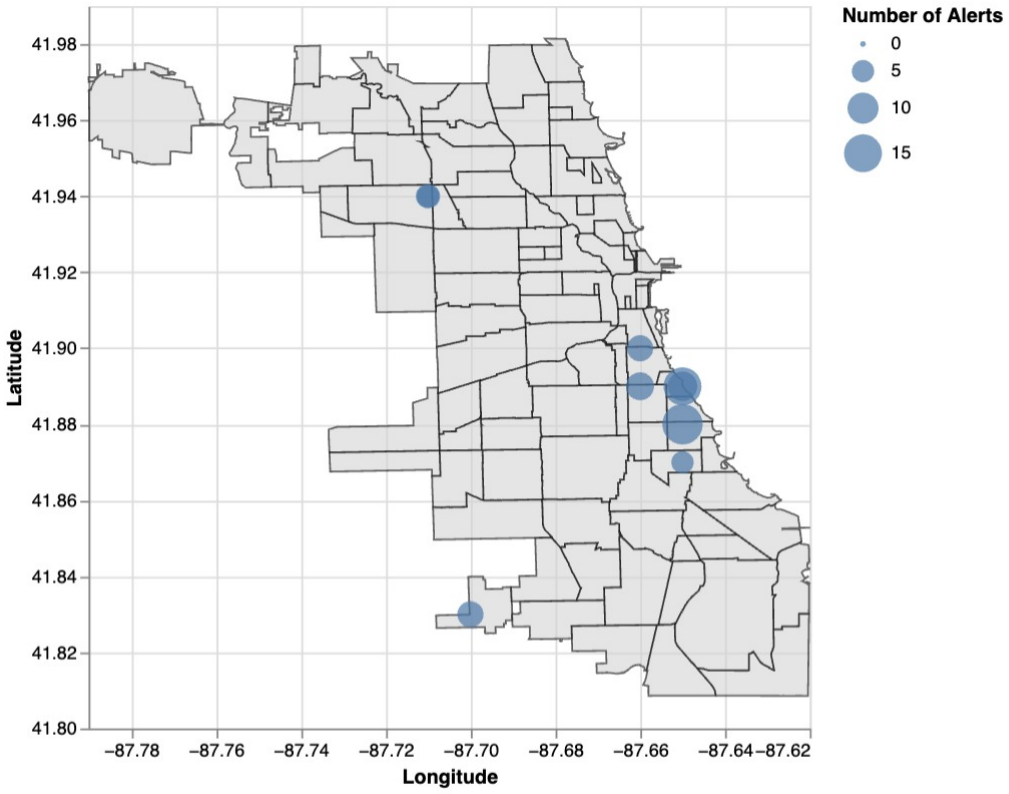
Select Type and Subtype:

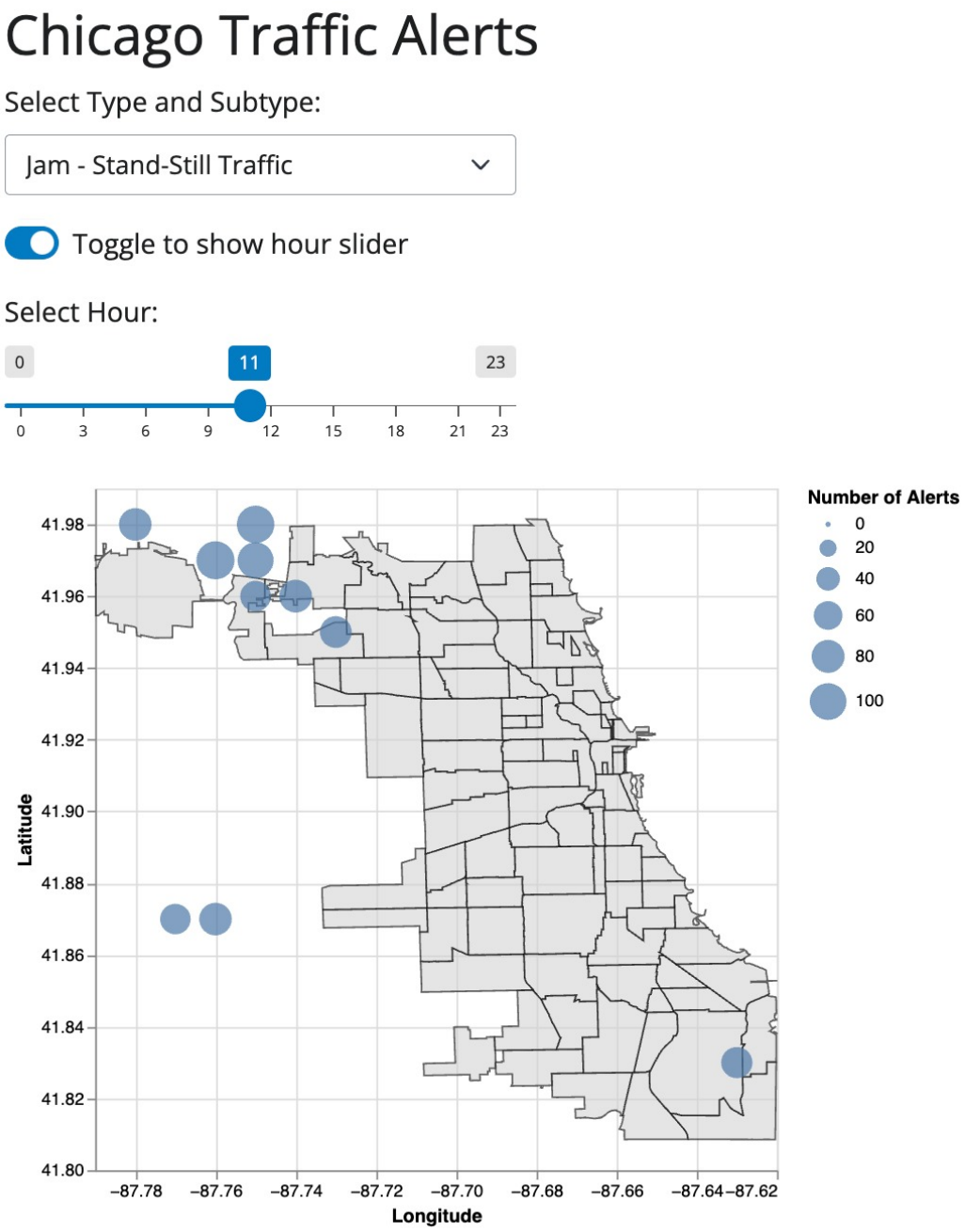
Jam - Stand-Still Traffic

Toggle to show hour slider

Select Range of Hours:







d. As of now, the scatter plot shows only the number of alerts. One modification that should be included is to color code the number of alerts by the time of the alert (as is done in the plot, wherein, different colors are allocated to morning and afternoon.) Further, it seems like in the app, the timing won't be classified by ranges or individual

hours, it will now be based on much broader catgories like morning, afternoon, evening or night. Further, instead of showing just one category, we will make changes such that each we can select multiple time categories to be viewed at once for each type-subtype combination.