Problem Set 6 - Waze Shiny Dashboard

Peter Ganong, Maggie Shi, and Andre Oviedo

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: **KT**
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **KT** (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 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!

```
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/katherinetu/Desktop/Pset6/top_alerts_map/01-basic-app/app.py")
```

Background

Data Download and Exploration (20 points)

1.

```
# Load the data for waza_data_sample.csv
df_sample =
 ↪  pd.read_csv("/Users/katherinetu/Desktop/Pset6/waze_data/waze_data_sample.csv")
print(df_sample.head(5))
```

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

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

                        subtype  roadType  reliability  magvar  \
0                           NaN        17            5     116
1             ROAD_CLOSED_EVENT         3            6     173
2  HAZARD_ON_SHOULDER_CAR_STOPPED         3            5     308
3                HAZARD_ON_ROAD        20            5     155
4             JAM_HEAVY_TRAFFIC         7            5     178

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

                       geoWKT
0   Point(-87.64577 41.892743)
1  Point(-87.646359 41.886295)
2   Point(-87.695982 41.93272)
3  Point(-87.669253 41.904497)
4  Point(-87.728322 41.978769)
```

```
# Variable names and data types in altair
name =
 ↪  ["Unamed","city","confidence","nThumbsUp","street","uuid","country","type","subtype","roa
types =
 ↪  ["Ordinal","Nominal","Quantitative","Quantitative","Nominal","Nominal","Nominal","Nominal
df_varnames = pd.DataFrame({"Variable Name":name,"Data Type":types})
print(df_varnames)
```

```
   Variable Name             Data Type
0         Unamed               Ordinal
1           city               Nominal
2     confidence          Quantitative
3      nThumbsUp          Quantitative
4         street               Nominal
```

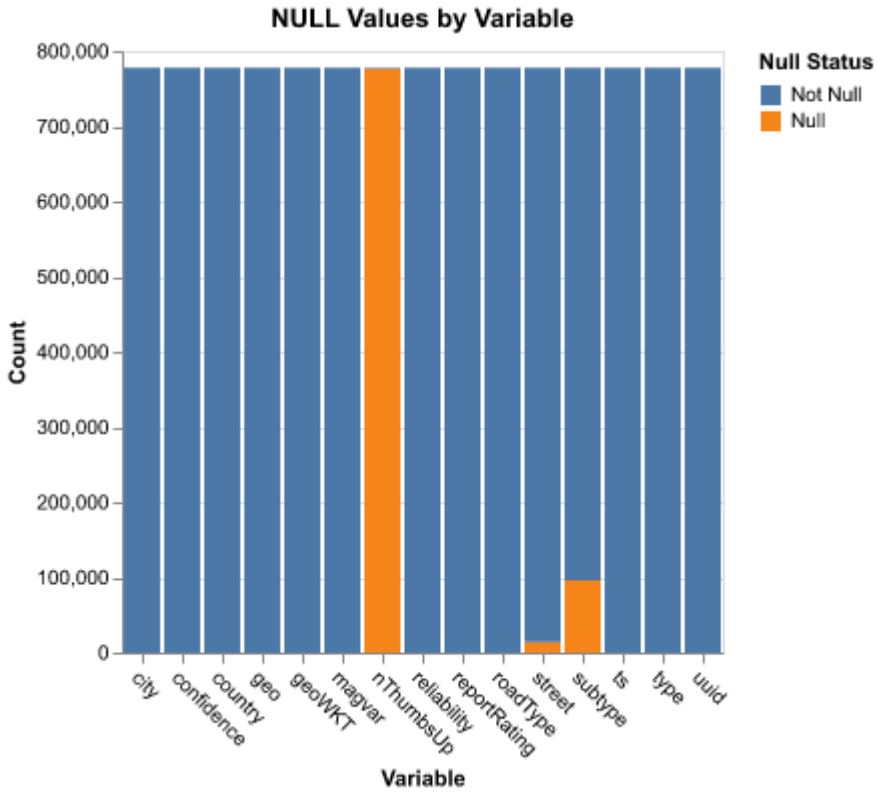
```
5           uuid                Nominal
6        country                Nominal
7           type                Nominal
8        subtype                Nominal
9       roadType                Nominal
10   reliability           Quantitative
11        magvar           Quantitative
12  reportRating  Quantitative/Ordinal
```

2.

```
# Load the data for waza_data.csv
df_waza =
 ↪  pd.read_csv("/Users/katherinetu/Desktop/Pset6/waze_data/waze_data.csv")
```

```
# Count the number of nulls and nonnulls for each variable
waza_null = df_waza.isnull().sum()
waza_nonnull = df_waza.count()
waza_total = pd.DataFrame({"Variable":df_waza.columns,"Null":waza_null,"Not
 ↪  Null": waza_nonnull, "Total":df_waza.shape[0]})
waza_total_long = waza_total.melt(id_vars="Variable",value_vars=["Null","Not
 ↪  Null"], var_name="Null Status", value_name="Count")
```

```
# Create a bar chart for the null counts
stacked_null = alt.Chart(waza_total_long, title = "NULL Values by
 ↪  Variable").mark_bar().encode(
  alt.X("Variable:N").axis(labelAngle=45),
  alt.Y("Count:Q"),
  alt.Color("Null Status:N")
)

stacked_null
```



nThumbsUp, street, and subtype variables have NULL values. nThumbsUp has the highest share of values that are missing.

3.

```
# Find unique values for types and subtypes
df_types = df_waza[["type","subtype"]].drop_duplicates()
df_types = df_types.sort_values(by="type").reset_index(drop = True)
print(df_types)
```

```
          type                              subtype
0      ACCIDENT                                  NaN
1      ACCIDENT                       ACCIDENT_MAJOR
2      ACCIDENT                       ACCIDENT_MINOR
3        HAZARD      HAZARD_ON_SHOULDER_CAR_STOPPED
4        HAZARD            HAZARD_WEATHER_HEAVY_SNOW
5        HAZARD     HAZARD_ON_SHOULDER_MISSING_SIGN
6        HAZARD          HAZARD_ON_SHOULDER_ANIMALS
7        HAZARD            HAZARD_ON_ROAD_ROAD_KILL
8        HAZARD                  HAZARD_WEATHER_FOG
```

```
9         HAZARD           HAZARD_ON_ROAD_LANE_CLOSED
10        HAZARD                 HAZARD_WEATHER_FLOOD
11        HAZARD                       HAZARD_WEATHER
12        HAZARD                   HAZARD_ON_SHOULDER
13        HAZARD                  HAZARD_WEATHER_HAIL
14        HAZARD              HAZARD_ON_ROAD_POT_HOLE
15        HAZARD                                 NaN
16        HAZARD                       HAZARD_ON_ROAD
17        HAZARD           HAZARD_ON_ROAD_CAR_STOPPED
18        HAZARD          HAZARD_ON_ROAD_CONSTRUCTION
19        HAZARD  HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT
20        HAZARD    HAZARD_ON_ROAD_EMERGENCY_VEHICLE
21        HAZARD                    HAZARD_ON_ROAD_ICE
22        HAZARD                HAZARD_ON_ROAD_OBJECT
23           JAM                 JAM_MODERATE_TRAFFIC
24           JAM                    JAM_HEAVY_TRAFFIC
25           JAM                    JAM_LIGHT_TRAFFIC
26           JAM              JAM_STAND_STILL_TRAFFIC
27           JAM                                 NaN
28  ROAD_CLOSED                    ROAD_CLOSED_EVENT
29  ROAD_CLOSED             ROAD_CLOSED_CONSTRUCTION
30  ROAD_CLOSED                   ROAD_CLOSED_HAZARD
31  ROAD_CLOSED                                 NaN
```

```python
# Identify how many of the types have NA subtypes
na_types = df_types[df_types["subtype"].isna()]["type"].unique()
number_na = len(na_types)
print(f"{number_na} types have NA subtypes.")
```

```
4 types have NA subtypes.
```

Which type have subtypes have enough information to consider that they could have sub-subtypes?

Hazard seems like it could have subtypes with sub-subtypes, because there are repeating keywords at the beginning of the classification acting like prefix, such as “Hazard on Shoulder Car Stopped” and “Hazard on Shoulder Missing Sign” should be two sub-subtypes under the subtype “On Shoulder”

```python
# Write a bulleted list for the hierarchy

# Accident
#   - Major
```

```
#   - Minor
# Hazard
#   - On Shoulder
#     - Car stopped
#     - Missing sign
#     - Animals
#   - Weather
#     - Heavy Snow
#     - Fog
#     - Flood
#     - Hail
#   - On Road
#     - Road Kill
#     - Lane Closed
#     - Pot Hole
#     - Car Stopped
#     - Construction
#     - Traffic Light Fault
#     - Emergency Vehicle
#     - Ice
#     - Object
# Jam
#   - Heavy Traffic
#   - Moderate Traffic
#   - Light Traffic
#   - Stand-Still Traffic
# Road_Closed
#   - Event
#   - Construction
#   - Hazard
```

I think NA subtypes should be kept because it might just contain information that is unclear on which subcategory it belongs to, and it could still contribute to the total amount of reports in each alert.

```
# Recoding NA as Unclassified
df_waza["subtype"] = df_waza["subtype"].fillna("Unclassified")
```

4.
5.

```
# Create a dataframe
df_crosswalk = pd.DataFrame(
    columns=["type", "subtype", "updated_type", "updated_subtype",
 ↪  "updated_subsubtype"])
```

2.

```
# Fill in the dataframe
cw_type = df_types["type"]
cw_subtype = df_types["subtype"]
cw_subtype = cw_subtype.fillna("Unclassified")
updated_type = df_types["type"].str.capitalize()
df_crosswalk = pd.DataFrame({
    "type": cw_type,
    "subtype": cw_subtype,
    "updated_type": updated_type
})
```

```
# Create function to apply updated subtype
def subtype(row):
    if "MAJOR" in row["subtype"]:
        return "Major"
    elif "MINOR" in row["subtype"]:
        return "Minor"
    elif "ON_SHOULDER" in row["subtype"]:
        return "On Shoulder"
    elif "WEATHER" in row["subtype"]:
        return "Weather"
    elif "ON_ROAD" in row["subtype"]:
        return "On Road"
    elif "HEAVY_TRAFFIC" in row["subtype"]:
        return "Heavy Traffic"
    elif "MODERATE_TRAFFIC" in row["subtype"]:
        return "Moderate Traffic"
    elif "LIGHT_TRAFFIC" in row["subtype"]:
        return "Light Traffic"
    elif "STAND_STILL" in row["subtype"]:
        return "Stand-Still Traffic"
    elif "EVENT" in row["subtype"]:
        return "Event"
    elif "ROAD_CLOSED_CONSTRUCTION" in row["subtype"]:
```

```
        return "Construction"
    elif "ROAD_CLOSED_HAZARD" in row["subtype"]:
        return "Hazard"
    return "Unclassified"


df_crosswalk["updated_subtype"] = df_crosswalk.apply(subtype, axis=1)
```

```
# Code for subsubtype
hazard = df_types[df_types["type"] == "HAZARD"].copy()
hazard["subtype"] = hazard["subtype"].astype(str)
replacements = ["HAZARD_", "ON_SHOULDER", "ON_ROAD", "WEATHER"]
for pattern in replacements:
    hazard["subtype"] = hazard["subtype"].str.replace(pattern, "",
 ↪  regex=False)

hazard["subtype"] = hazard["subtype"].str.replace("_", " ", regex=False)
hazard["subtype"] = hazard["subtype"].replace("", np.nan)
hazard["subtype"] = hazard["subtype"].str.strip().str.capitalize()
```

```
#Add hazard subsubtype to the larger crosswalk
df_crosswalk["updated_subsubtype"] = pd.NA
df_crosswalk.loc[df_crosswalk["type"] == "HAZARD","updated_subsubtype"] =
 ↪  hazard["subtype"]

#check observations
print(f"The crosswalk dataframe has {len(df_crosswalk)} observations")
```

```
The crosswalk dataframe has 32 observations
```

3.

```
# Merge the crosswalk with orginal data
df_merge = df_waza.merge(df_crosswalk, left_on = ["type","subtype"], right_on
 ↪  = ["type","subtype"])
accident = df_merge[df_merge["updated_type"] == "Accident"]
acc_unclass = accident[accident["updated_subtype"]=="Unclassified"].shape[0]
print(f"{acc_unclass} rows are Accident - Unclassified.")
```

```
24359 rows are Accident - Unclassified.
```

4. Extra Credit

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

1.

a.

```
#ChatGPT Prompt: "I have a column with geodata seperated by a space, how do a
 ↪  create two new columns, one for the first value and one for the second,
 ↪  using regex"

df_merge[["Longitude","Latitude"]] =
 ↪  df_merge["geo"].str.extract(r"([-\d.]+)\s+([-\d.]+)")

df_merge["Longitude"] = df_merge["Longitude"].astype(float)
df_merge["Latitude"] = df_merge["Latitude"].astype(float)
```

b.

```
# bin the latitude and longitude to make sure it only shows 2 digits of
 ↪  decimals

df_merge["lat_bin"] = ((df_merge["Latitude"]//0.01)*0.01).round(2)
df_merge["lon_bin"] = ((df_merge["Longitude"]//0.01)*0.01).round(2)

# aggregate the data to find the highest number of observations
lon_lat_count =
 ↪  df_merge.groupby(["lon_bin","lat_bin"]).size().reset_index(name =
 ↪  "count")
most_count_lon_lat = lon_lat_count.sort_values(by = "count", ascending =
 ↪  False).head(1).iloc[:, 0:2].values.tolist()

print(f"The most frequent binned longitude-latitude is
 ↪  {most_count_lon_lat}.")
```

```
The most frequent binned longitude-latitude is [[-87.75, 41.96]].
```

c.

```
# group the data by type and subtype to find the frequency of lon-lat
 ↪  combinations for each type and subtype

df_grouped =
 ↪  df_merge.groupby(["updated_type","updated_subtype","lon_bin","lat_bin"]).size().reset_in
 ↪  = "count")

# sort to find the most frequent latitude and longitude bins for each type
 ↪  and subtype

sorted_grouped =
 ↪  df_grouped.sort_values(["updated_type","updated_subtype","count"],
 ↪  ascending = [True, True, False])

# group to only include the top 10

top_10 = sorted_grouped.groupby(["updated_type","updated_subtype"]).head(10)

top_10["type-subtype"] = top_10["updated_type"] + "-" +
 ↪  top_10["updated_subtype"]

top_10.to_csv("/Users/katherinetu/Desktop/Pset6/top_alerts_map/top_alerts_map.csv")
```

```
/var/folders/wb/j78hfj4x4w7g6f94hhf5svk00000gn/T/ipykernel_8819/3581771930.py:13:
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
  top_10["type-subtype"] = top_10["updated_type"] + "-" +
  top_10["updated_subtype"]
```

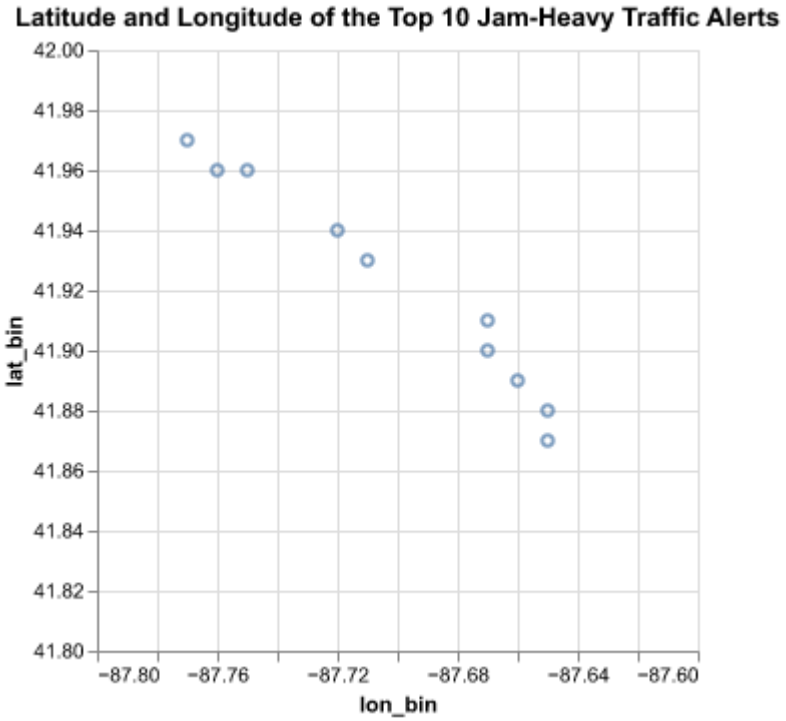
Including collapsing the longitude and latitude to bins, this is 3 levels of aggregation (binning, groupping to find the frequency of lat-lon combination for each subtype, and including only the top 10 by each type and subtype.)

```
print(f"This dataframe has {top_10.shape[0]} rows.")
```

```
This dataframe has 154 rows.
```

d.

```
jam_heavy = top_10[top_10["type-subtype"] == "Jam-Heavy Traffic"]

chart_jam = alt.Chart(jam_heavy, title = "Latitude and Longitude of the Top
↪  10 Jam-Heavy Traffic Alerts").mark_point().encode(
  alt.X("lon_bin:Q").scale(domain=(-87.80, -87.60)),
  alt.Y("lat_bin:Q").scale(domain=(41.80, 42))
)

chart_jam
```



3.

a.

```
import requests

url =
↪  "http://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON"

response = requests.get(url)
```

```
if response.status_code == 200:
    with open("./top_alerts_map/chicago-boundaries.geojson", "wb") as file:
        file.write(response.content)
    print("GeoJSON file downloaded successfully!")
else:
    print(f"Failed to download file. Status code: {response.status_code}")
```

```
GeoJSON file downloaded successfully!
```

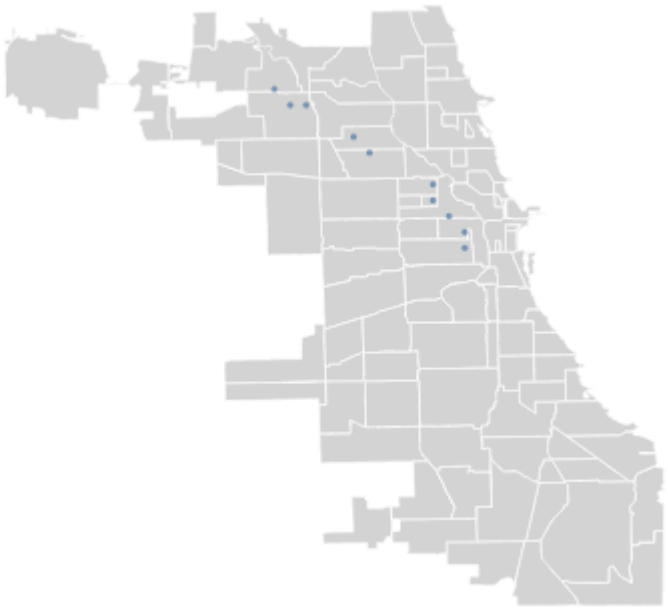
b.

```
# MODIFY ACCORDINGLY
file_path = "./top_alerts_map/chicago-boundaries.geojson"
#----

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

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

4.

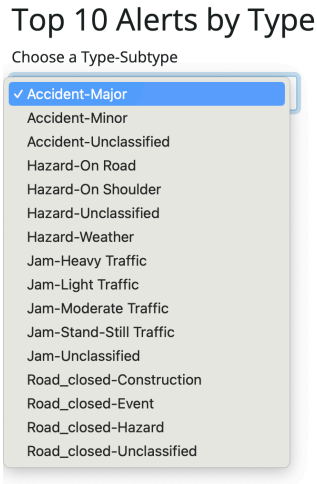
```
#create plot for background
background = alt.Chart(geo_data).mark_geoshape(
    fill='lightgrey',
    stroke='white').project(type='equirectangular').properties(width=500,
 ↪  height=300)
```

```
#create plot for points
points = alt.Chart(jam_heavy).mark_circle(size=10).encode(
    longitude='lon_bin:Q',
    latitude='lat_bin:Q',
    tooltip = ['lon_bin:Q', 'lat_bin:Q']
).project(type='equirectangular').properties(
    width=500,
    height=300
)

layered_chart = background + points
layered_chart
```



5.

a.



There are 16 type x subtypes in my dropdown menu.

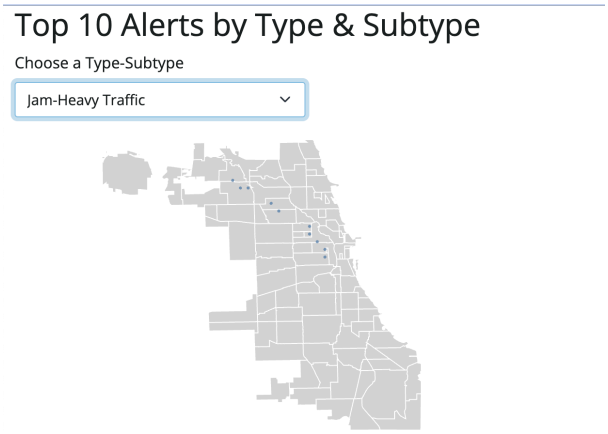


Figure 1: Jam-Heavy Traffic Dynamic Plot Screenshot

b.

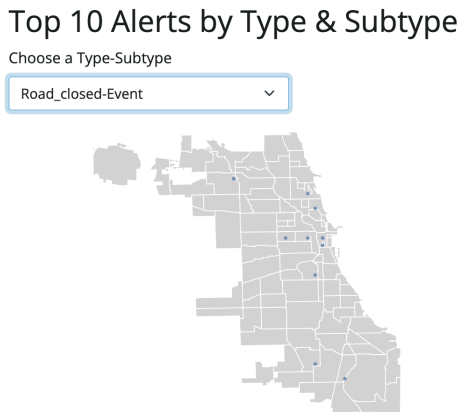


Figure 2: Road Closure Due to Events Dynamic Plot Screenshot

c.

As demonstrated in the map, it seems like areas around the loop, west loop, and some roads scattered on the outskirts tend to be most frequent to these types of alerts.

d. Where are alerts for Hazard On Roads most common?

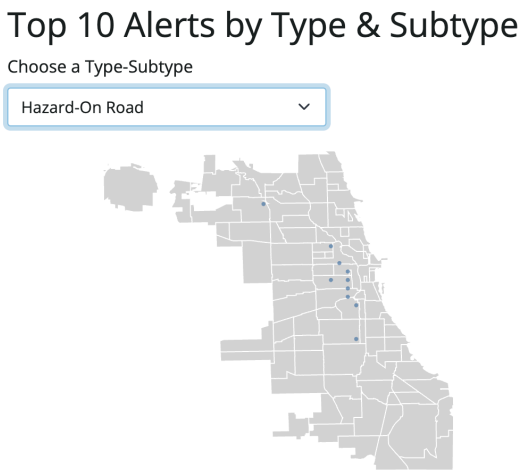


Figure 3: Hazard On Road Dynamic Plot Screenshot

The hazard on road seem to be concentrated downtown, around the western high ways and streets.

e. I think adding a column or table for the count for the top 10 alerts can enhance the analysis through showing how many alerts exactly are there for these top 10 locations.

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

1.

a. I think it would not be a good idea to collapse directly from this column, because the ts column contains information that is unncessary, such as the year, month, date, etc. Since we only want to know the time of day, collapsing by this column without specifying the hour value would not be very helpful.

b.

```
#create an hour column
df_merge["ts"] = pd.to_datetime(df_merge["ts"])
df_merge["hour"] = df_merge["ts"].dt.hour.astype(str).str.zfill(2) + ":00"
```

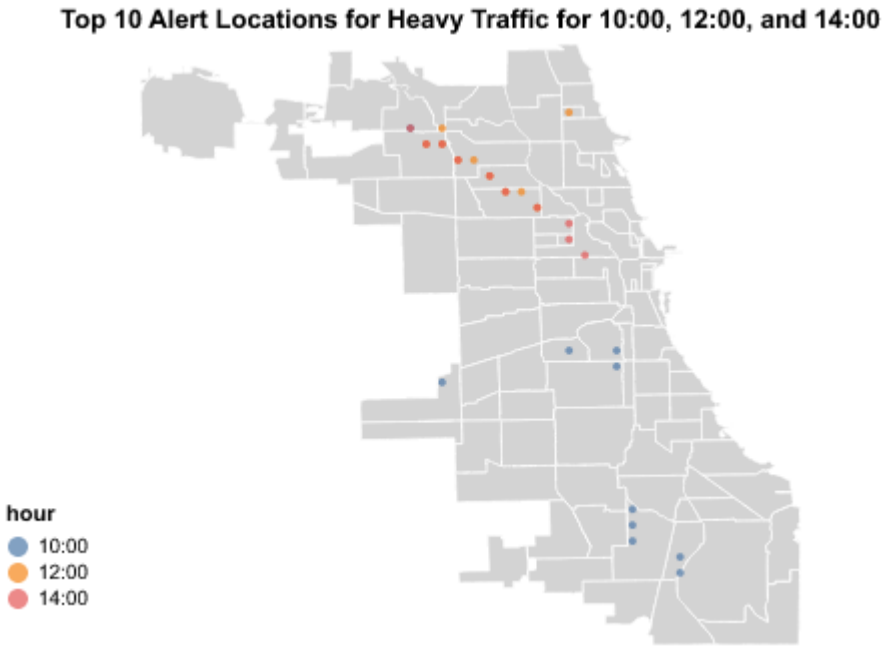
```
#collapse by hour to find the count

df_merge["type-subtype"] = df_merge["updated_type"] + "-" +
 ↪  df_merge["updated_subtype"]

df_hour_count =
 ↪  df_merge.groupby(["type-subtype","hour","lon_bin","lat_bin"]).size().reset_index(name
 ↪  = "count")

# sort to find the most frequent latitude and longitude bins for each hour

sorted_hour_count =
 ↪  df_hour_count.sort_values(["type-subtype","hour","count"], ascending =
 ↪  [True, True, False])

# group to only include the top 10

top_alerts_map_byhour =
 ↪  sorted_hour_count.groupby(["type-subtype","hour"]).head(10)

top_alerts_map_byhour.to_csv("/Users/katherinetu/Desktop/Pset6/top_alerts_map_byhour/top_ale

print(f"This dataset has {top_alerts_map_byhour.shape[0]} rows.")
```

```
This dataset has 3201 rows.
```

c.

```
#Generate a plot for jam-heavy traffic for 3 different times
jam_heavy_hour =
 ↪  top_alerts_map_byhour[top_alerts_map_byhour["type-subtype"]== "Jam-Heavy
 ↪  Traffic"]

jam_heavy_hour_points = alt.Chart(
    jam_heavy_hour, title = "Top 10 Alert Locations for Heavy Traffic for
 ↪  10:00, 12:00, and 14:00"
    ).mark_circle(size=15).encode(
    longitude='lon_bin:Q',
    latitude='lat_bin:Q',
    color = alt.Color('hour:N').legend(orient = 'bottom-left'),
    tooltip = ['lon_bin:Q', 'lat_bin:Q']
).transform_filter(
```

```
    alt.FieldOneOfPredicate(
        field='hour',
    oneOf=["10:00","12:00","14:00"])).project(type='equirectangular').properties(
    width=500,
    height=300
)

layered_threetimes_jam = background + jam_heavy_hour_points
layered_threetimes_jam
```



2.

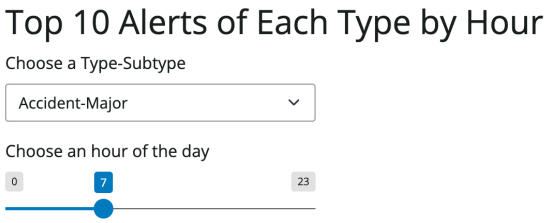


Figure 4: Slider and Chooser UI Screenshot

a.

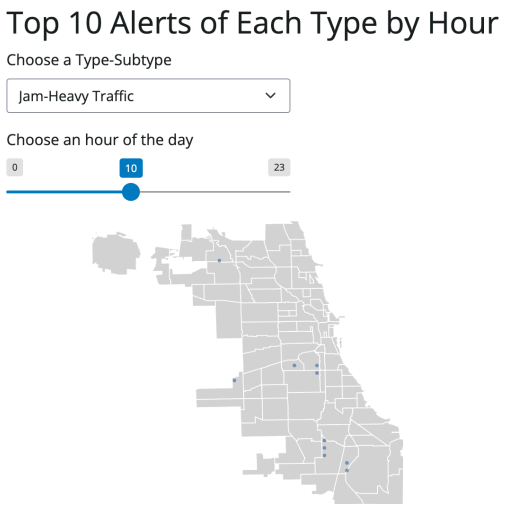


Figure 5: Jam Heavy Traffic 10:00 Screenshot

b.

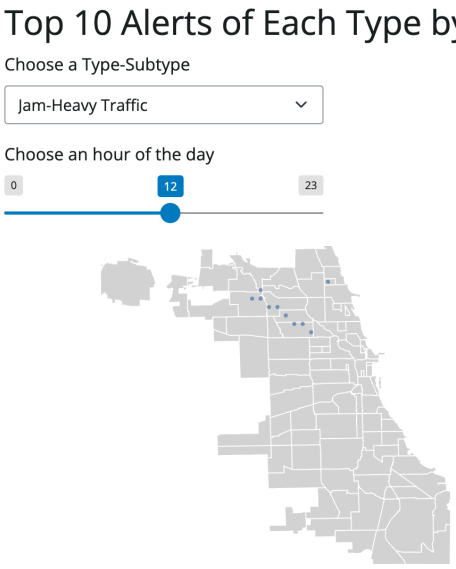


Figure 6: Jam Heavy Traffic 12:00 Screenshot

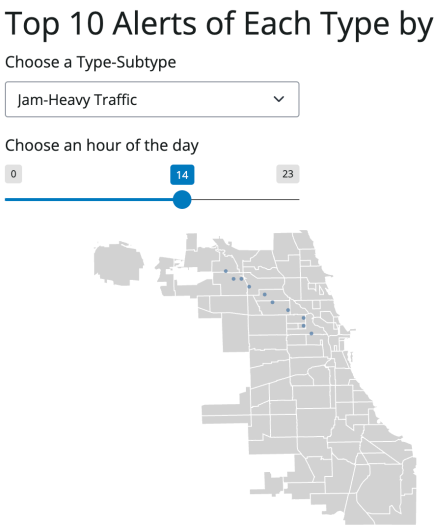


Figure 7: Jam Heavy Traffic 14:00 Screenshot

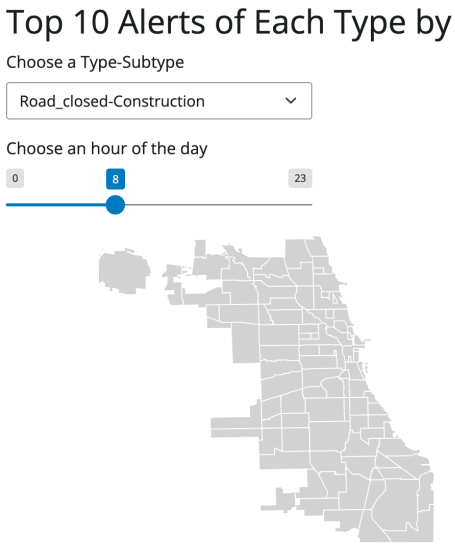


Figure 8: Jam Heavy Traffic Morning

c.

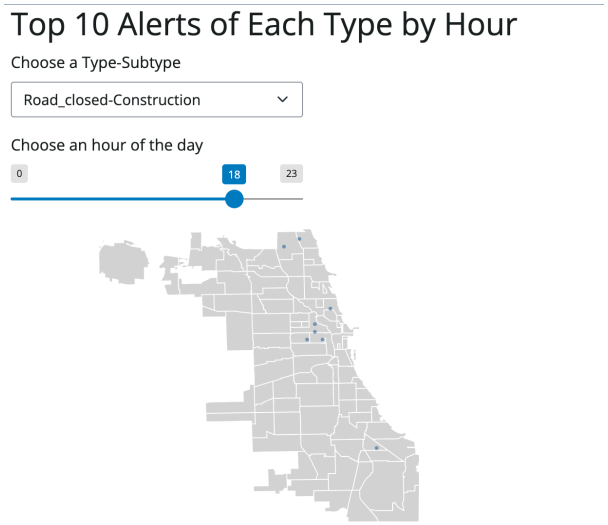


Figure 9: Jam Heavy Traffic Night

Construction is done more during the night hours.

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

1.

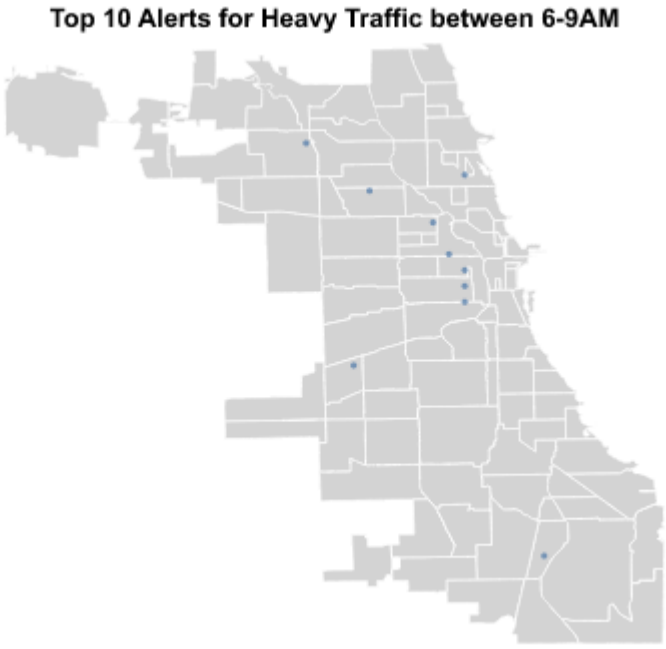
a. Because the range of hours is not set, it would be difficult to collapse by the range of hours. It might be better to do so through a dynamic process.

b.

```
# filter the jam_heavy_hour dataset for 06:00-09:00

jam_heavy_hour_6_9 = jam_heavy_hour[(jam_heavy_hour["hour"]>= "06:00") &
 ↪  (jam_heavy_hour["hour"]<"09:00")]

# aggregate to find the top 10 location

jam_heavy_hour_filtered_count =
 ↪  jam_heavy_hour_6_9.groupby(["lon_bin","lat_bin"]).agg({'count':
 ↪  'sum'}).reset_index()
```

```
top_10_jam_6_9 = jam_heavy_hour_filtered_count.sort_values(by = 'count',
↪  ascending = False).head(10)
```

```
jam_6_9 = alt.Chart(top_10_jam_6_9, title = "Top 10 Alerts for Heavy Traffic
↪  between 6-9AM").mark_circle(size=10).encode(
    longitude='lon_bin:Q',
    latitude='lat_bin:Q',
    tooltip = ['lon_bin:Q', 'lat_bin:Q']
).project(type='equirectangular').properties(
    width=500,
    height=300
)

layered_6_9_jam = background + jam_6_9
layered_6_9_jam
```



2.

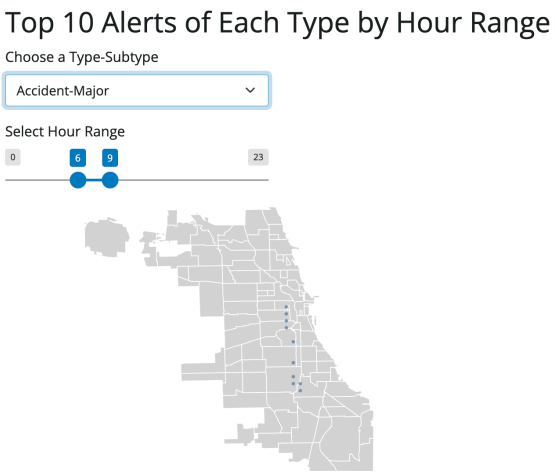


Figure 10: Dropdown and Range Slider Screenshot

a.

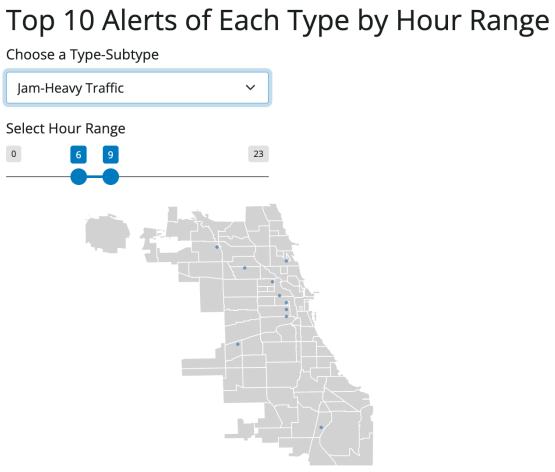


Figure 11: Dropdown and Range Slider Screenshot

b.

3.

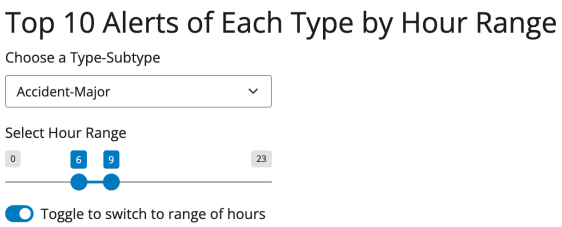


Figure 12: Switch Button Screenshot

a.

The possible values for the switch button would be True or False, indicating switching on the button or not, which could be associated to its respective conditional panels.

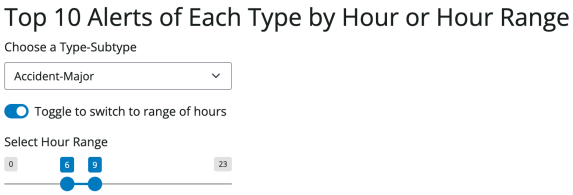


Figure 13: Switch On Screenshot

b.

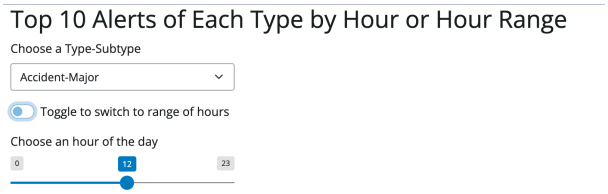


Figure 14: Switch Off Screenshot

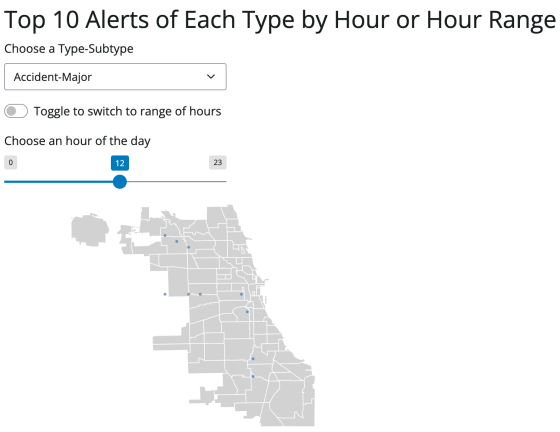


Figure 15: Single Hour Screenshot

c.

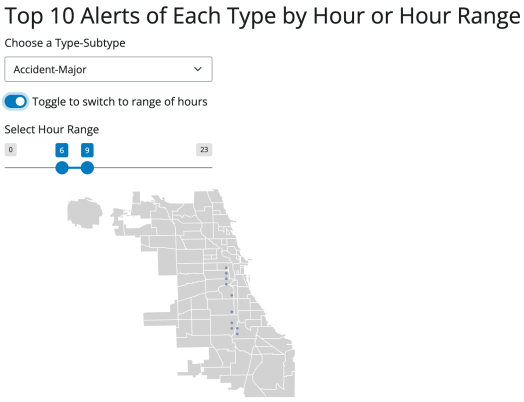


Figure 16: Ranged Hour Screenshot

d. I would need to modify the plots to make circles that vary by size to indicate the number of alerts in each location. I would also need to classify the hours into morning hours or afternoon hours, then assign colors to them to show which one they belong to.