Problem Set 6 - Waze Shiny Dashboard

Sijie Wu

2024-11-24

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: **___**
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **___** (2 point)
3. Late coins used this pset: **___** Late coins left after submission: **___**
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!*

```python
def print_file_contents(file_path):
    """Print contents of a file."""
    try:
        with open(file_path, 'r') as f:
            content = f.read()
            print("```python")
            print(content)
            print("```")
    except FileNotFoundError:
        print("```python")
        print(f"Error: File '{file_path}' not found")
        print("```")
    except Exception as e:
        print("```python")
        print(f"Error reading file: {e}")
        print("```")


print_file_contents("./top_alerts_map_byhour/app.py")  # Change accordingly
```

Background

Data Download and Exploration (20 points)

1.

```python
import pandas as pd

file_path = 'waze_data/waze_data_sample.csv'
waze_data_sample = pd.read_csv(file_path)
print(waze_data_sample.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7781 entries, 0 to 7780
Data columns (total 16 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
```

```
 0   Unnamed: 0    7781 non-null   int64
 1   city          7781 non-null   object
 2   confidence    7781 non-null   int64
 3   nThumbsUp     10 non-null     float64
 4   street        7630 non-null   object
 5   uuid          7781 non-null   object
 6   country       7781 non-null   object
 7   type          7781 non-null   object
 8   subtype       6777 non-null   object
 9   roadType      7781 non-null   int64
 10  reliability   7781 non-null   int64
 11  magvar        7781 non-null   int64
 12  reportRating  7781 non-null   int64
 13  ts            7781 non-null   object
 14  geo           7781 non-null   object
 15  geoWKT        7781 non-null   object
dtypes: float64(1), int64(6), object(9)
memory usage: 972.8+ KB
None
```

Variable Name	Altair Data Type
Unnamed: 0	Nominal Ordinal
city	Nominal
confidence	Quantitative
nThumbsUp	Quantitative
street	Nominal
uuid	Nominal Ordinal
country	Nominal
type	Nominal
subtype	Nominal
roadType	Nominal Quantitative
reliability	Quantitative
magvar	Nominal Quantitative
reportRating	Nominal Quantitative
ts	
geo	
geoWKT	

2.

I ask ChatGPT “how to calculate the null values and not-null values”, and “how to plot the bar chart where the x-axis is each variable and the stacked bar has two categories”.

```
file_path = 'waze_data/waze_data.csv'
waze_data = pd.read_csv(file_path)

# Calculate missing and non-missing values
null_counts = waze_data.isnull().sum()
non_null_counts = waze_data.notnull().sum()

# Combine into a DataFrame for plotting
missing_data = pd.DataFrame({
    'Variable': null_counts.index,
    'NULL': null_counts.values,
    'Non-NULL': non_null_counts.values
})

missing_data_long = missing_data.melt(id_vars='Variable',
                                      value_vars=['NULL', 'Non-NULL'],
                                      var_name='Observation Status',
                                      value_name='Count')

chart = alt.Chart(missing_data_long).mark_bar().encode(
    x=alt.X('Variable:N', title='Variables', sort=null_counts.index),
    y=alt.Y('Count:Q', title='Number of Observations'),
    color=alt.Color('Observation Status:N',
                    scale=alt.Scale(domain=['NULL', 'Non-NULL'],
                                    range=['#1f77b4', '#ff7f0e']),
                    legend=alt.Legend(title="Observation Status")),
    tooltip=['Variable', 'Observation Status', 'Count']
).properties(
    title='Stacked Bar Chart of Missing and Non-Missing Values',
    width=400,
    height=200
).configure_axis(
    labelAngle=45
)

chart.save(
    '/Users/wsjsmac/Desktop/Autumn/PPHA_30538/mine/Pset_6/chart1.png')
```

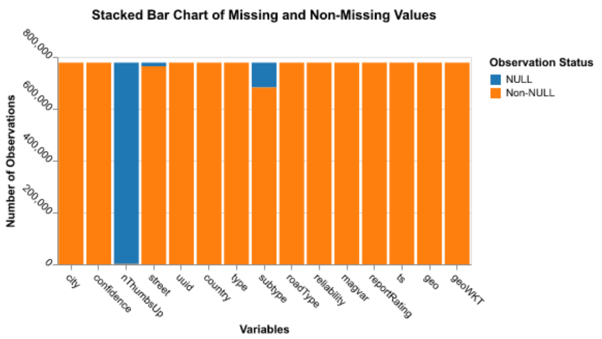


Figure 1: Stacked Bar Chart of Missing and Non-Missing Values

From the chart we can see that, variable `nThumbsUp`, `street`, and `subtype` have null values. From the chart we can see that, `nThumbsUp` has the highest proportion of missing values.

3.

```python
# Unique values in the 'type' column
unique_types = waze_data['type'].unique()
print("Unique values in 'type':", unique_types)

# Unique values in the 'subtype' column
unique_subtypes = waze_data['subtype'].unique()
print("Unique values in 'subtype':", unique_subtypes)
```

```
Unique values in 'type': ['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']
Unique values in 'subtype': [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']
```

I use ChatGPT to solve how many types have a sub- type that is NA.

```
# Filter rows where 'subtype' is NA
missing_subtype = waze_data[waze_data['subtype'].isnull()]

# Find the unique 'type' values in those rows
types_with_missing_subtype = missing_subtype['type'].unique()

# Count the unique types
num_types_with_missing_subtype = len(types_with_missing_subtype)

print(
    f"Number of types with at least one missing subtype:
     ↪  {num_types_with_missing_subtype}")
print("Types with missing subtypes:", types_with_missing_subtype)
```

```
Number of types with at least one missing subtype: 4
Types with missing subtypes: ['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']
```

All of the 4 types have at least one missing subtype.

Based on the printed unique value of type, I think the type `HAZARD` has subtypes that have enough information to generate a sub-subtype. Since in the codebook, `HAZARD` has subtypes of `HAZARD_ON_ROAD`, `HAZARD_ON_SHOULDER`, `HAZARD_WEATHER`, it is possible to generate 3 sub-subtypes.

- JAM
    - MODERATE_TRAFFIC
    - HEAVY_TRAFFIC
    - STAND_STILL_TRAFFIC
    - LIGHT_TRAFFIC
- ACCIDENT
    - MINOR
    - MAJOR
- ROAD_CLOSED
    - HAZARD
    - CONSTRUCTION
    - EVENT
- HAZARD
    - ON_ROAD
        - OBJECT

  - POT_HOLE
  - ROAD_KILL
  - LANE_CLOSED
  - OIL
  - ICE
  - CONSTRUCTION
  - CAR_STOPPED
  - TRAFFIC_LIGHT_FAULT
  - EMERGENCY_VEHICLE
- ON_SHOULDER
  - CAR_STOPPED
  - ANIMALS
  - MISSING_SIGN
- WEATHER
  - FOG
  - HAIL
  - HEAVY_RAIN
  - HEAVY_SNOW
  - FLOOD
  - MONSOON
  - TORNADO
  - HEAT_WAVE
  - HURRICANE
  - FREEZING_RAIN

I ask ChatGPT how to code NA values as "Unclassified".

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

4.

a.

```python
crosswalk = waze_data[['type', 'subtype']].copy()

# Add new columns: updated_type, updated_subtype, updated_subsubtype
crosswalk['updated_type'] = None
crosswalk['updated_subtype'] = None
crosswalk['updated_subsubtype'] = None
crosswalk = crosswalk.drop_duplicates().reset_index(drop=True)
```

b.

I ask ChatGPT on how to make only the first letter capitalized.

```
mapping = {
    "JAM": {
        "JAM_HEAVY_TRAFFIC": {"updated_type": "JAM", "updated_subtype":
         ↪  "HEAVY_TRAFFIC", "updated_subsubtype": "Unclassified"},
        "JAM_MODERATE_TRAFFIC": {"updated_type": "JAM", "updated_subtype":
         ↪  "MODERATE_TRAFFIC", "updated_subsubtype": "Unclassified"},
        "JAM_STAND_STILL_TRAFFIC": {"updated_type": "JAM", "updated_subtype":
         ↪  "STAND_STILL_TRAFFIC", "updated_subsubtype": "Unclassified"},
        "JAM_LIGHT_TRAFFIC": {"updated_type": "JAM", "updated_subtype":
         ↪  "LIGHT_TRAFFIC", "updated_subsubtype": "Unclassified"},
        "Unclassified": {"updated_type": "JAM", "updated_subtype":
         ↪  "Unclassified", "updated_subsubtype": "Unclassified"}
    },
    "ACCIDENT": {
        "ACCIDENT_MINOR": {"updated_type": "ACCIDENT", "updated_subtype":
         ↪  "MINOR", "updated_subsubtype": "Unclassified"},
        "ACCIDENT_MAJOR": {"updated_type": "ACCIDENT", "updated_subtype":
         ↪  "MAJOR", "updated_subsubtype": "Unclassified"},
        "Unclassified": {"updated_type": "ACCIDENT", "updated_subtype":
         ↪  "Unclassified", "updated_subsubtype": "Unclassified"}
    },
    "ROAD_CLOSED": {
        "ROAD_CLOSED_HAZARD": {"updated_type": "ROAD_CLOSED",
         ↪  "updated_subtype": "HAZARD", "updated_subsubtype":
         ↪  "Unclassified"},
        "ROAD_CLOSED_CONSTRUCTION": {"updated_type": "ROAD_CLOSED",
         ↪  "updated_subtype": "CONSTRUCTION", "updated_subsubtype":
         ↪  "Unclassified"},
        "ROAD_CLOSED_EVENT": {"updated_type": "ROAD_CLOSED",
         ↪  "updated_subtype": "EVENT", "updated_subsubtype":
         ↪  "Unclassified"},
        "Unclassified": {"updated_type": "ROAD_CLOSED", "updated_subtype":
         ↪  "Unclassified", "updated_subsubtype": "Unclassified"}
    },
    "HAZARD": {
        "HAZARD_ON_ROAD": {"updated_type": "HAZARD", "updated_subtype":
         ↪  "ON_ROAD", "updated_subsubtype": "Unclassified"},
        "HAZARD_ON_SHOULDER": {"updated_type": "HAZARD", "updated_subtype":
         ↪  "ON_SHOULDER", "updated_subsubtype": "Unclassified"},
        "HAZARD_WEATHER": {"updated_type": "HAZARD", "updated_subtype":
         ↪  "WEATHER", "updated_subsubtype": "Unclassified"},
        "HAZARD_ON_ROAD_OBJECT": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "ON_ROAD", "updated_subsubtype": "OBJECT"},
        "HAZARD_ON_ROAD_POT_HOLE": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "ON_ROAD", "updated_subsubtype": "POT_HOLE"},
        "HAZARD_ON_ROAD_ROAD_KILL": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "ON_ROAD", "updated_subsubtype": "ROAD_KILL"},
        "HAZARD_ON_ROAD_LANE_CLOSED": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "ON_ROAD", "updated_subsubtype":
         ↪  "LANE_CLOSED"},
        "HAZARD_ON_ROAD_OIL": {"updated_type": "HAZARD", "updated_subtype":
         ↪  "ON_ROAD", "updated_subsubtype": "OIL"},
        "HAZARD_ON_ROAD_ICE": {"updated_type": "HAZARD", "updated_subtype":
         ↪  "ON_ROAD", "updated_subsubtype": "ICE"},
        "HAZARD_ON_ROAD_CONSTRUCTION": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "ON_ROAD", "updated_subsubtype":
         ↪  "CONSTRUCTION"},
        "HAZARD_ON_ROAD_CAR_STOPPED": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "ON_ROAD", "updated_subsubtype":
         ↪  "CAR_STOPPED"},
        "HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "ON_ROAD", "updated_subsubtype":
         ↪  "TRAFFIC_LIGHT_FAULT"},
        "HAZARD_ON_ROAD_EMERGENCY_VEHICLE": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "ON_ROAD", "updated_subsubtype":
         ↪  "EMERGENCY_VEHICLE"},
        "HAZARD_ON_SHOULDER_CAR_STOPPED": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "ON_SHOULDER", "updated_subsubtype":
         ↪  "CAR_STOPPED"},
        "HAZARD_ON_SHOULDER_ANIMALS": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "ON_SHOULDER", "updated_subsubtype":
         ↪  "ANIMALS"},
        "HAZARD_ON_SHOULDER_MISSING_SIGN": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "ON_SHOULDER", "updated_subsubtype":
         ↪  "MISSING_SIGN"},
        "HAZARD_WEATHER_FOG": {"updated_type": "HAZARD", "updated_subtype":
         ↪  "WEATHER", "updated_subsubtype": "FOG"},
        "HAZARD_WEATHER_HAIL": {"updated_type": "HAZARD", "updated_subtype":
         ↪  "WEATHER", "updated_subsubtype": "HAIL"},
        "HAZARD_WEATHER_HEAVY_RAIN": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "WEATHER", "updated_subsubtype":
         ↪  "HEAVY_RAIN"},
        "HAZARD_WEATHER_HEAVY_SNOW": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "WEATHER", "updated_subsubtype":
         ↪  "HEAVY_SNOW"},
        "HAZARD_WEATHER_FLOOD": {"updated_type": "HAZARD", "updated_subtype":
         ↪  "WEATHER", "updated_subsubtype": "FLOOD"},
        "HAZARD_WEATHER_MONSOON": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "WEATHER", "updated_subsubtype": "MONSOON"},
        "HAZARD_WEATHER_TORNADO": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "WEATHER", "updated_subsubtype": "TORNADO"},
        "HAZARD_WEATHER_HEAT_WAVE": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "WEATHER", "updated_subsubtype": "HEAT_WAVE"},
        "HAZARD_WEATHER_HURRICANE": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "WEATHER", "updated_subsubtype": "HURRICANE"},
        "HAZARD_WEATHER_FREEZING_RAIN": {"updated_type": "HAZARD",
         ↪  "updated_subtype": "WEATHER", "updated_subsubtype":
         ↪  "FREEZING_RAIN"},
        "Unclassified": {"updated_type": "HAZARD", "updated_subtype":
         ↪  "Unclassified", "updated_subsubtype": "Unclassified"}
    }
}


# Update crosswalk with the mapping
for idx, row in crosswalk.iterrows():
    type_val = row['type']
    subtype_val = row['subtype']

    # Check if the type and subtype are in the mapping
    if type_val in mapping and subtype_val in mapping[type_val]:
        # Update the columns using the mapping
        crosswalk.at[idx,
                     'updated_type'] =
                      ↪  mapping[type_val][subtype_val]["updated_type"]
        crosswalk.at[idx, 'updated_subtype'] =
 ↪  mapping[type_val][subtype_val]["updated_subtype"]
        crosswalk.at[idx, 'updated_subsubtype'] =
 ↪  mapping[type_val][subtype_val]["updated_subsubtype"]
    else:
        # For subtypes not found in the mapping, set to "Unclassified"
        crosswalk.at[idx, 'updated_type'] = "Unclassified"
        crosswalk.at[idx, 'updated_subtype'] = "Unclassified"
        crosswalk.at[idx, 'updated_subsubtype'] = "Unclassified"

crosswalk['updated_type'] = crosswalk['updated_type'].str.capitalize()
crosswalk['updated_subtype'] = crosswalk['updated_subtype'].str.capitalize()
```

```
crosswalk['updated_subsubtype'] =
 ↪  crosswalk['updated_subsubtype'].str.capitalize()
```

c.

I ask ChatGPT on how to calculate the # of rows where “updated_type” = “Accident”, and “updated_subtype” = “Unclassified”.

```
merged_data = pd.merge(waze_data, crosswalk, on=[
                       'type', 'subtype'], how='left')

# Filter rows where 'updated_type' is 'Accident' and 'updated_subtype' is
 ↪  'Unclassified'
filtered_data = merged_data[(merged_data['updated_type'] == 'Accident') & (
    merged_data['updated_subtype'] == 'Unclassified')]

# Get the number of rows in the filtered data
num_rows = filtered_data.shape[0]

# Display the number of rows
print(f'Number of rows for "Accident - Unclassified": {num_rows}')
```

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

d. I ask ChatGPT on how to check variables in crosswalk and merged_data.

```
# Check if the values in 'type' and 'subtype' from the crosswalk are present
 ↪  in the merged dataset
check_crosswalk_in_merged = crosswalk[['type',
 ↪  'subtype']].isin(merged_data[['type', 'subtype']])

# If all values are in the merged dataset, the result will be True for each
 ↪  row
all_values_match = check_crosswalk_in_merged.all(axis=1)

# Check if the merged dataset and crosswalk match for type and subtype
if all(all_values_match):
    print("The values in 'type' and 'subtype' from the crosswalk are present
     ↪  in the merged dataset.")
else:
    print("There are discrepancies between the 'type' and 'subtype' values in
     ↪  the crosswalk and merged dataset.")
```

```
There are discrepancies between the 'type' and 'subtype' values in the
crosswalk and merged dataset.
```

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

1.

a. I ask ChatGPT “How to use regex to extract latitude and longitude from the coordinates data, and create two variable latitude and longitude?”

Its response in below: ChatGPT pormpt

```python
import re

# Function to extract latitude and longitude


def extract_coordinates(coord):
    pattern = r"POINT\((-?\d+\.\d+)\s(-?\d+\.\d+)\)"
    match = re.match(pattern, coord)
    if match:
        lon, lat = match.groups()
        return float(lat), float(lon)
    return None, None


# Apply the function to create new columns
merged_data['latitude'], merged_data['longitude'] = zip(
    *merged_data['geo'].apply(extract_coordinates))

# Display the DataFrame
merged_data.head()
```

	city	confidence	nThumbsUp	street	uuid	country	type
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	ACC
2	Chicago, IL	0	NaN	NaN	0e5f14ae-7251-46af-a7f1-53a5272cd37d	US	ROA
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

b.

I ask ChatGPT by “how to Bin the latitude and longitude variables into bins of step size 0.01? That is, coordinats with values of (-41.9232, -87.4251) should become (-41.92, -87.43).”

I ask ChatGPT how to “make a latitude-longitude combination with the binned data”.

I ask ChatGPT how to “find the binned latitude-longitude combination with the greatest number of observations in a dataset”.

```
# Binning
merged_data['binned_latitude'] = merged_data['latitude'].round(2)
merged_data['binned_longitude'] = merged_data['longitude'].round(2)

# Combine into a new column as tuple
merged_data['coordinates_tuple'] = list(
    zip(merged_data['binned_latitude'], merged_data['binned_longitude']))

# Combine into a new column as string
merged_data['coordinates_string'] = merged_data.apply(
    lambda row: f"({row['binned_latitude']}, {row['binned_longitude']})",
     ↪  axis=1
)

merged_data.head()

# Group by binned coordinates and count occurrences
grouped_data = merged_data.groupby(
    "coordinates_string").size().reset_index(name='count')

# Find the combination with the greatest number of observations
max_combination = grouped_data.loc[grouped_data['count'].idxmax()]

print("Binned Latitude-Longitude Combination with the Most Observations:")
print(max_combination)
```

```
Binned Latitude-Longitude Combination with the Most Observations:
coordinates_string    (41.88, -87.65)
count                           21325
Name: 388, dtype: object
```

c.

```python
# Filter for chosen type and subtype
chosen_type = "JAM"
chosen_subtype = "JAM_STAND_STILL_TRAFFIC"
filtered_data = merged_data[(merged_data["type"] == chosen_type) &
 ↪  (merged_data["subtype"] == chosen_subtype)]

# Aggregate by binned latitude-longitude and count alerts
aggregated_data = (
    filtered_data.groupby("coordinates_string")
    .size()
    .reset_index(name='alert_count')
)

# Sort by alert_count and select the top 10 bins
top_10_bins = aggregated_data.nlargest(10, 'alert_count')

# Display the top 10 bins
print("Top 10 Latitude-Longitude Bins with the Most Alerts with chosen_type =
 ↪  'Jam', chosen_subtype = 'Heavy_traffic':")
print(top_10_bins)
```

```
Top 10 Latitude-Longitude Bins with the Most Alerts with chosen_type = 'Jam',
chosen_subtype = 'Heavy_traffic':
    coordinates_string  alert_count
339    (41.88, -87.65)         4666
357    (41.89, -87.65)         4278
374     (41.9, -87.66)         3563
523    (41.97, -87.76)         2503
544    (41.98, -87.75)         2347
501    (41.96, -87.74)         2287
522    (41.97, -87.75)         2079
452    (41.94, -87.71)         1983
475    (41.95, -87.73)         1972
547    (41.98, -87.78)         1913
```

```python
df_alert_counts = (
    merged_data.groupby(["type", "subtype", "updated_type",
 ↪  "updated_subtype",
                        "updated_subsubtype", "binned_latitude",
                         ↪  "binned_longitude"])
    .size()
    .reset_index(name="alert_count")
    .sort_values(by="alert_count", ascending=False)
)

df_alert_counts_path = './top_alerts_map/df_alert_counts.csv'
df_alert_counts.to_csv(df_alert_counts_path, index=False)
```

```
merged_data.to_csv('./merged_data.csv', index=False)
```

The level of aggregation is at `type`, `subtype`, `updated_type`, `updated_subtype`, `updated_subsubtype`, `binned_latitude`, `binned_longitude`.

```
# Count the number of unique latitude-longitude bins
num_rows = df_alert_counts.shape[0]
print(f"The alert_counts has {num_rows} rows.")
```

```
The alert_counts has 11060 rows.
```

2.

```
chosen_type = "Jam"
chosen_subtype = "Heavy_traffic"
filtered_data = merged_data[(merged_data["updated_type"] == chosen_type) & (
    merged_data["updated_subtype"] == chosen_subtype)]

# Aggregate by binned latitude-longitude and count alerts
aggregated_data = (
    filtered_data.groupby(
        ["coordinates_string", "binned_longitude", "binned_latitude"])
    .size()
    .reset_index(name='alert_count')
    .sort_values(by="alert_count", ascending=False)
)

top_10 = aggregated_data.head(10)

# Create scatter plot
scatter_plot = (
    alt.Chart(top_10)
    .mark_circle()
    .encode(
        x=alt.X("binned_longitude:Q", title="Longitude",
                scale=alt.Scale(domain=[-87.52, -87.96])),
        y=alt.Y("binned_latitude:Q", title="Latitude",
                scale=alt.Scale(domain=[41.66, 42.02])),
        color=alt.Color("alert_count:Q", title="Number of Alerts"),
        tooltip=["binned_longitude", "binned_latitude", "alert_count"],
    )
    .properties(
        title="Top 10 Latitude-Longitude Bins for 'Jam - Heavy Traffic'
↪   Alerts",
        width=200,
        height=200,
    )
)

scatter_plot.save(
    '/Users/wsjsmac/Desktop/Autumn/PPHA_30538/mine/Pset_6/chart2.png')
```

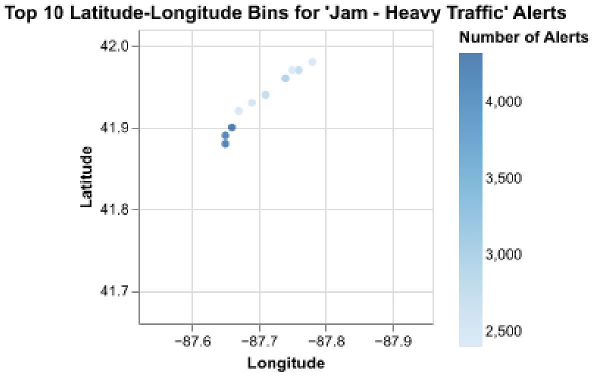


Figure 2: Top 10 Latitude-Longitude Bins for ‘Jam - Heavy Traffic’ Alerts

3.

a.

I ask ChatGPT how to “use request package to download and open the file in the link”

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

# Step 1: Download the GeoJSON file
```

```python
response = requests.get(url)
if response.status_code == 200:
    chicago_geojson = response.json()  # Directly parse JSON response
else:
    print(f"Failed to download data: {response.status_code}")
    chicago_geojson = None

# Step 2: Process the GeoJSON data (if successful)
if chicago_geojson:
    # Extract the 'features' from the GeoJSON
    geo_data = alt.Data(values=chicago_geojson["features"])
    print("GeoJSON data successfully loaded and processed.")
else:
    print("Failed to load GeoJSON data.")
```

```
GeoJSON data successfully loaded and processed.
```

b.

```python
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.

```python
points = alt.Chart(top_10).mark_circle().encode(
    longitude='binned_longitude:Q',
    latitude='binned_latitude:Q',
    size=alt.Size('alert_count', scale=alt.Scale(range=[10, 100])),
    tooltip=["binned_latitude", "binned_longitude", "alert_count"]
)
map_layer = (
    alt.Chart(geo_data).mark_geoshape(
        fill="lightgray",
        stroke="white",
        strokeWidth=1
    )
    .properties(
        width=400,
        height=600
    )
    .project("identity", reflectY=True)
)

combined_plot = (
    map_layer + points
).properties(title="Top 10 Jam-Heavy Traffic with Geo Data")

combined_plot.save(
    '/Users/wsjsmac/Desktop/Autumn/PPHA_30538/mine/Pset_6/chart3.png')
```

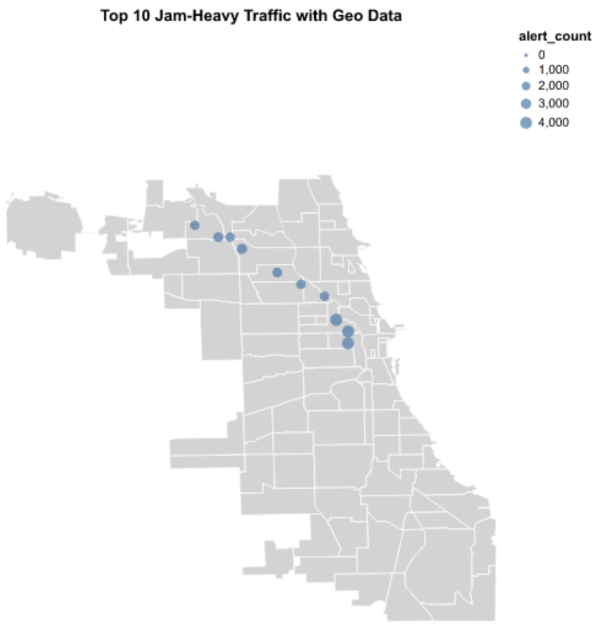
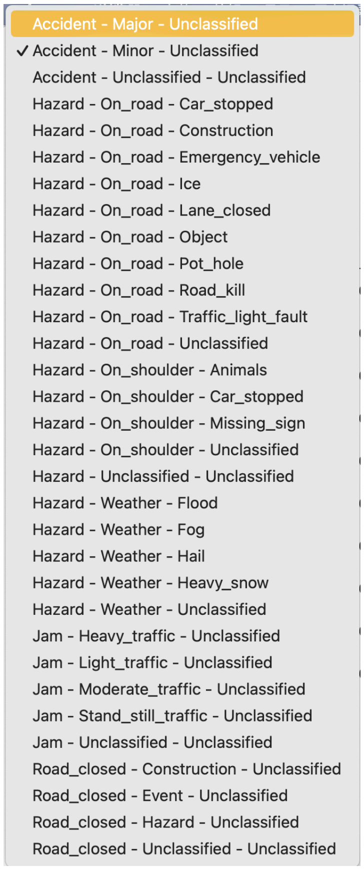


Figure 3: Top 10 Jam-Heavy Traffic with Geo Data

5.

See in "/top_alerts_map/app.py" for codes.

a.



Accident - Major - Unclassified
✓ Accident - Minor - Unclassified
Accident - Unclassified - Unclassified
Hazard - On_road - Car_stopped
Hazard - On_road - Construction
Hazard - On_road - Emergency_vehicle
Hazard - On_road - Ice
Hazard - On_road - Lane_closed
Hazard - On_road - Object
Hazard - On_road - Pot_hole
Hazard - On_road - Road_kill
Hazard - On_road - Traffic_light_fault
Hazard - On_road - Unclassified
Hazard - On_shoulder - Animals
Hazard - On_shoulder - Car_stopped
Hazard - On_shoulder - Missing_sign
Hazard - On_shoulder - Unclassified
Hazard - Unclassified - Unclassified
Hazard - Weather - Flood
Hazard - Weather - Fog
Hazard - Weather - Hail
Hazard - Weather - Heavy_snow
Hazard - Weather - Unclassified
Jam - Heavy_traffic - Unclassified
Jam - Light_traffic - Unclassified
Jam - Moderate_traffic - Unclassified
Jam - Stand_still_traffic - Unclassified
Jam - Unclassified - Unclassified
Road_closed - Construction - Unclassified
Road_closed - Event - Unclassified
Road_closed - Hazard - Unclassified
Road_closed - Unclassified - Unclassified

Figure 4: screenshot of the dropdown menu

There are 32 type * subtype combinations in my dropdown menu.

b.

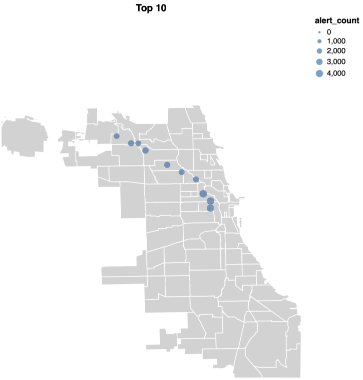


Figure 5: screenshot of “Jam - Heavy Traffic” plot

c.

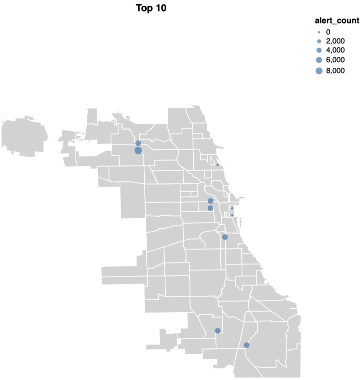
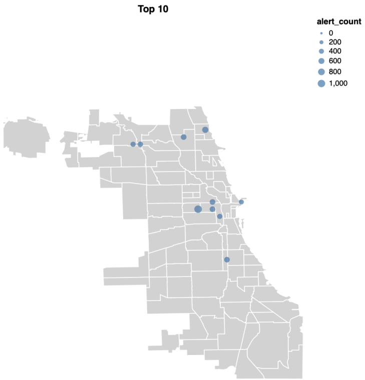


Figure 6: screenshot of “Road Closed - Event” plot

The road closed events mostly happen near northwestern of Chicago, near downtown, and around south side.

d. Question: Where does the “Hazard on Road Construction” happen most?



Plot:

Answer: The hazard on road construction most happens around downtown and the north side.

e.

We can add a new column of time based on the data from the `waze_data` to know when alerts happens, together with using the map to know where the alerts happen.

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

1.

a. I don't think it will be a good idea to use `ts` column to show the time of the alert. Because this variable is too exact to the second, and will generate too many distinct values for readers to ues.

b. I ask ChatGPT how to "extract the hour from the ts column".

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

```
hour_alert_counts = (
    merged_data.groupby(["type", "subtype", "updated_type",
 ↪  "updated_subtype",
                        "updated_subsubtype", "binned_latitude",
                         ↪  "binned_longitude", "hour"])
    .size()
    .reset_index(name="alert_count")
    .sort_values(by="alert_count", ascending=False)
)
```

```
hour_alert_counts['hour'] =
 ↪  hour_alert_counts['hour'].str.split(":").str[0].astype(int)
hour_alert_counts_path = './top_alerts_map_byhour/hour_alert_counts.csv'
hour_alert_counts.to_csv(hour_alert_counts_path, index=False)
```

```
num_rows = hour_alert_counts.shape[0]
print(f"The hour_alert_counts has {num_rows} rows.")
```

```
The hour_alert_counts has 87923 rows.
```

c.

chosen_hour = “02:00”

```
chosen_type = "Jam"
chosen_subtype = "Heavy_traffic"
chosen_hour = "02:00"
filtered_data = merged_data[(merged_data["updated_type"] == chosen_type) & (
    merged_data["updated_subtype"] == chosen_subtype) & (merged_data["hour"]
 ↪  == chosen_hour)]

aggregated_data = (
    filtered_data.groupby(
        ["coordinates_string", "binned_longitude", "binned_latitude"])
    .size()
    .reset_index(name='alert_count')
    .sort_values(by="alert_count", ascending=False)
)

top_10 = aggregated_data.head(10)

points = alt.Chart(top_10).mark_circle().encode(
    longitude='binned_longitude:Q',
    latitude='binned_latitude:Q',
    size=alt.Size('alert_count', scale=alt.Scale(range=[10, 100])),
    tooltip=["binned_latitude", "binned_longitude", "alert_count"]
)
map_layer = (
    alt.Chart(geo_data).mark_geoshape(
        fill="lightgray",
        stroke="white",
        strokeWidth=1
    )
    .properties(
        width=400,
        height=600
    )
    .project("identity", reflectY=True)
)

combined_plot = (
    map_layer + points
).properties(title="Top 10 Jam-Heavy Traffic with Geo Data at 02:00")

combined_plot.save(
    '/Users/wsjsmac/Desktop/Autumn/PPHA_30538/mine/Pset_6/chart4.png')
```

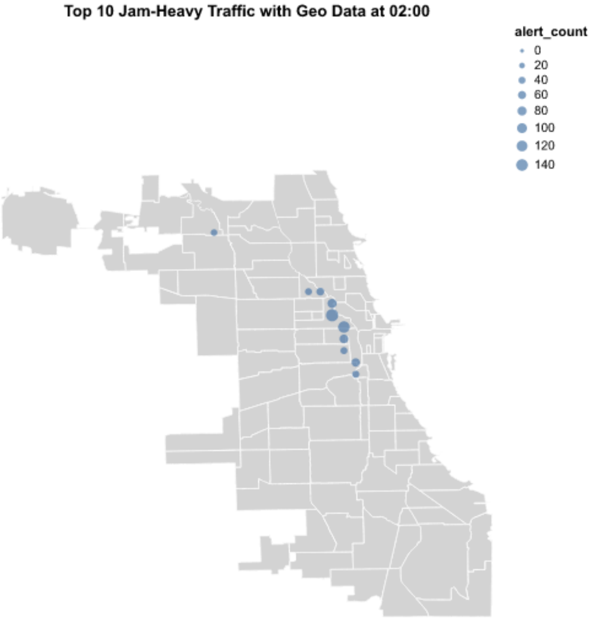


Figure 7: Top 10 Jam-Heavy Traffic with Geo Data at 02:00

chosen_hour = "10:00"

```
chosen_type = "Jam"
chosen_subtype = "Heavy_traffic"
chosen_hour = "10:00"
filtered_data = merged_data[(merged_data["updated_type"] == chosen_type) & (
    merged_data["updated_subtype"] == chosen_subtype) & (merged_data["hour"]
↪   == chosen_hour)]

aggregated_data = (
    filtered_data.groupby(
        ["coordinates_string", "binned_longitude", "binned_latitude"])
    .size()
    .reset_index(name='alert_count')
    .sort_values(by="alert_count", ascending=False)
)

top_10 = aggregated_data.head(10)
```

```
points = alt.Chart(top_10).mark_circle().encode(
    longitude='binned_longitude:Q',
    latitude='binned_latitude:Q',
    size=alt.Size('alert_count', scale=alt.Scale(range=[10, 100])),
    tooltip=["binned_latitude", "binned_longitude", "alert_count"]
)
map_layer = (
    alt.Chart(geo_data).mark_geoshape(
        fill="lightgray",
        stroke="white",
        strokeWidth=1
    )
    .properties(
        width=400,
        height=600
    )
    .project("identity", reflectY=True)
)

combined_plot = (
    map_layer + points
).properties(title="Top 10 Jam-Heavy Traffic with Geo Data at 10:00")

combined_plot.save(
    '/Users/wsjsmac/Desktop/Autumn/PPHA_30538/mine/Pset_6/chart5.png')
```

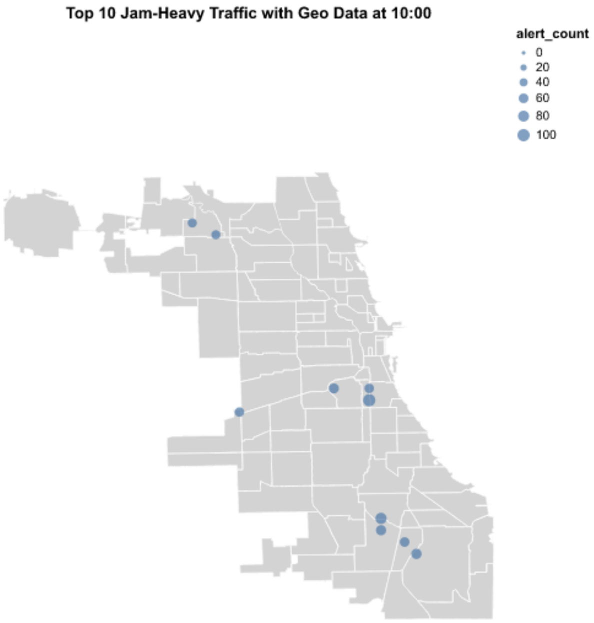


Figure 8: Top 10 Jam-Heavy Traffic with Geo Data at 10:00

chosen_hour = "16:00"

```
chosen_type = "Jam"
chosen_subtype = "Heavy_traffic"
chosen_hour = "16:00"
filtered_data = merged_data[(merged_data["updated_type"] == chosen_type) & (
    merged_data["updated_subtype"] == chosen_subtype) & (merged_data["hour"]
↪   == chosen_hour)]

aggregated_data = (
    filtered_data.groupby(
        ["coordinates_string", "binned_longitude", "binned_latitude"])
    .size()
    .reset_index(name='alert_count')
    .sort_values(by="alert_count", ascending=False)
)

top_10 = aggregated_data.head(10)
```

```
points = alt.Chart(top_10).mark_circle().encode(
    longitude='binned_longitude:Q',
    latitude='binned_latitude:Q',
    size=alt.Size('alert_count', scale=alt.Scale(range=[10, 100])),
    tooltip=["binned_latitude", "binned_longitude", "alert_count"]
)
map_layer = (
    alt.Chart(geo_data).mark_geoshape(
        fill="lightgray",
        stroke="white",
        strokeWidth=1
    )
    .properties(
        width=400,
        height=600
    )
    .project("identity", reflectY=True)
)

combined_plot = (
    map_layer + points
).properties(title="Top 10 Jam-Heavy Traffic with Geo Data at 16:00")

combined_plot.save(
    '/Users/wsjsmac/Desktop/Autumn/PPHA_30538/mine/Pset_6/chart6.png')
```

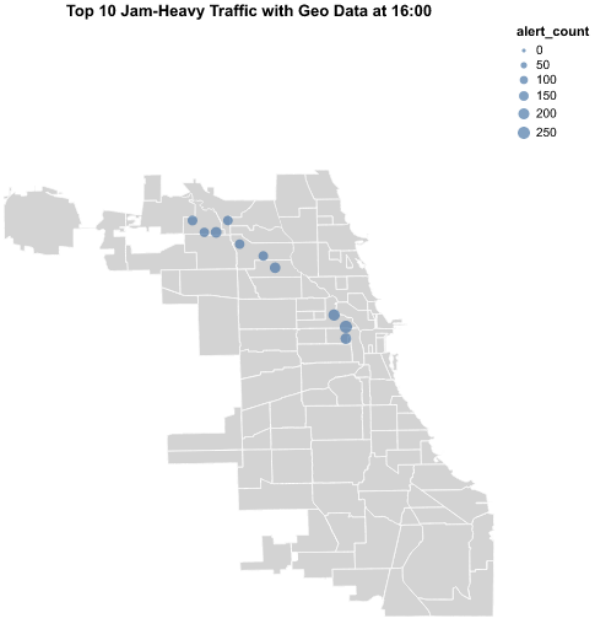
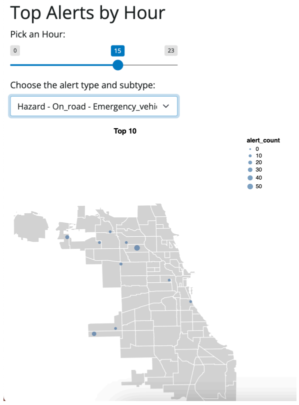


Figure 9: Top 10 Jam-Heavy Traffic with Geo Data at 16:00

2.

See in “/top_alerts_map_byhour/app.py” for codes.

a.

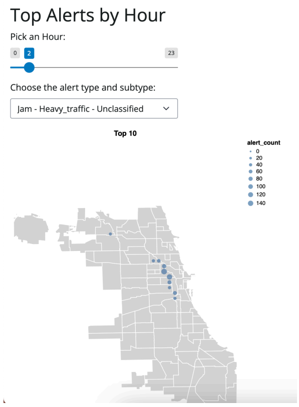

Top Alerts by Hour

Pick an Hour: 15

Choose the alert type and subtype: Hazard - On_road - Emergency_vehi

Figure 10: screenshot of “Top Alerts by Hour” plot

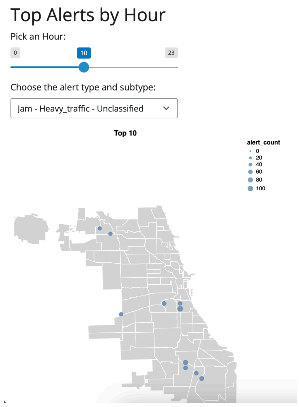
b.


Top Alerts by Hour

Pick an Hour: 0 2 23

Choose the alert type and subtype: Jam - Heavy_traffic - Unclassified

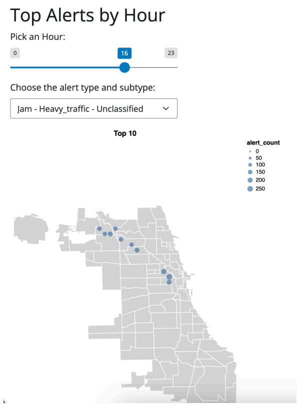
Top 10


Top Alerts by Hour

Pick an Hour: 0 10 23

Choose the alert type and subtype: Jam - Heavy_traffic - Unclassified

Top 10

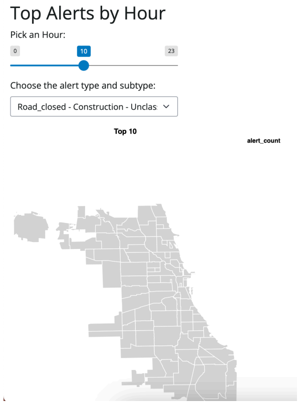

Top Alerts by Hour

Pick an Hour: 0 16 23

Choose the alert type and subtype: Jam - Heavy_traffic - Unclassified

Top 10

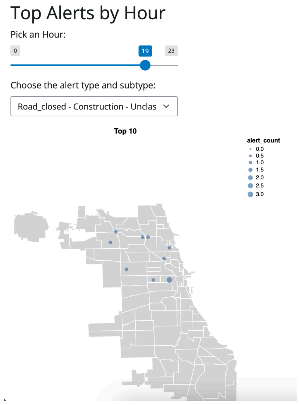
c. Road construction is done more in the night than in the morning. I choose 10:00 and 19:00 as the time for the road construction alerts. It can be seen that, at 10 a.m., there is zero road construction alerts, while at 7 p.m., there are a dozen of road construction alerts.


Top Alerts by Hour

Pick an Hour: 0 10 23

Choose the alert type and subtype: Road_closed - Construction - Unclas

Top 10


Top Alerts by Hour

Pick an Hour: 0 19 23

Choose the alert type and subtype: Road_closed - Construction - Unclas

Top 10

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

1.

a.

I think it will be a good idea to collapse data by range of hours. Because we need to plot the alerts of hour-range, so collagpse data by one-hour interval is a good idea.

b.

```
chosen_type = "Jam"
chosen_subtype = "Heavy_traffic"
chosen_hour_start = "06:00"
chosen_hour_end = "09:00"
filtered_data = merged_data[(merged_data["updated_type"] == chosen_type) & (
    merged_data["updated_subtype"] == chosen_subtype) & (merged_data["hour"]
↪   >= chosen_hour_start) & (merged_data["hour"] < chosen_hour_end)]

aggregated_data = (
    filtered_data.groupby(
        ["coordinates_string", "binned_longitude", "binned_latitude"])
    .size()
    .reset_index(name='alert_count')
    .sort_values(by="alert_count", ascending=False)
)

top_10 = aggregated_data.head(10)

points = alt.Chart(top_10).mark_circle().encode(
    longitude='binned_longitude:Q',
    latitude='binned_latitude:Q',
    size=alt.Size('alert_count', scale=alt.Scale(range=[10, 100])),
    tooltip=["binned_latitude", "binned_longitude", "alert_count"]
)
map_layer = (
    alt.Chart(geo_data).mark_geoshape(
        fill="lightgray",
        stroke="white",
        strokeWidth=1
    )
    .properties(
        width=400,
        height=600
    )
    .project("identity", reflectY=True)
)

combined_plot = (
    map_layer + points
).properties(title="Top 10 Jam-Heavy Traffic with Geo Data from 6:00 AM to
↪   9:00 AM")
```

```
combined_plot.save(
    '/Users/wsjsmac/Desktop/Autumn/PPHA_30538/mine/Pset_6/chart7.png')
```

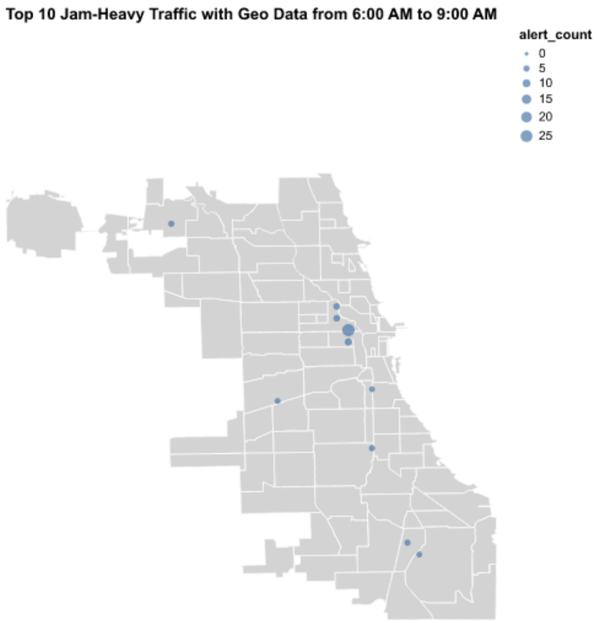
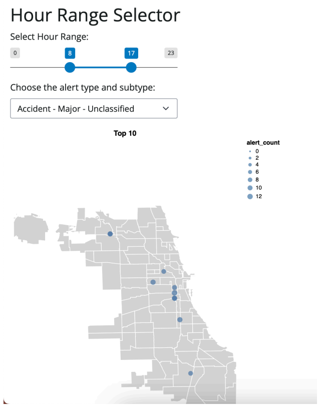

Top 10 Jam-Heavy Traffic with Geo Data from 6:00 AM to 9:00 AM
alert_count
0
5
10
15
20
25

Figure 11: Top 10 Jam-Heavy Traffic with Geo Data from 6:00 AM to 9:00 AM

2.

See in “/top_alerts_map_byhour_sliderrange/app.py” for codes.

a.



Hour Range Selector

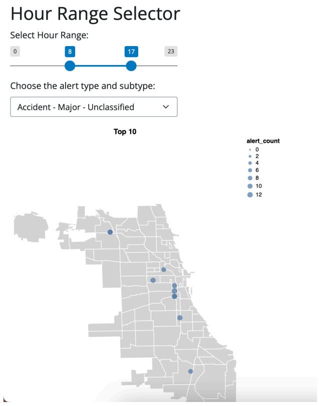
Select Hour Range:

Choose the alert type and subtype:

Accident - Major - Unclassified

Figure 12: screenshot of “Hour Range Selector” plot

b.



Hour Range Selector

Select Hour Range:

Choose the alert type and subtype:

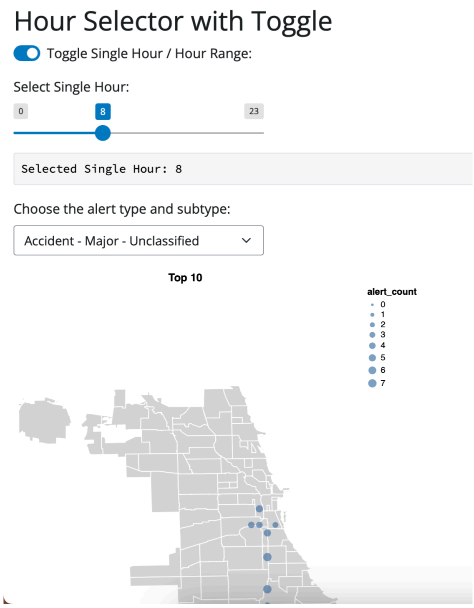
Accident - Major - Unclassified

Figure 13: screenshot of “Top 10 Jam-Heavy Traffic with Geo Data from 6:00 AM to 9:00 AM” plot

3.

See in “/top_alerts_map_byhour_sliderrange/app2.py” for updated codes.

I ask ChatGPT “How to update my codes to meet the requirement with `ui.input_switch`”.



Hour Selector with Toggle

Toggle Single Hour / Hour Range:

Select Single Hour:

0 8 23

```
Selected Single Hour: 8
```

Choose the alert type and subtype:

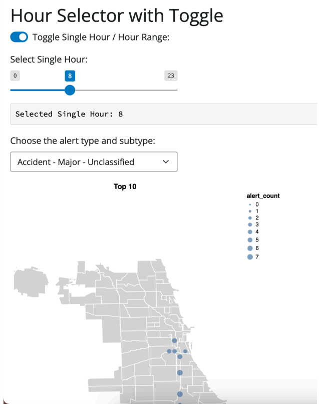
Accident - Major - Unclassified

Figure 14: screenshot of “Hour Selector with Toggle” plot, draft

a.

The possible values for `ui.input_switch` are `Ture` and `False`.

b.



Hour Selector with Toggle

Toggle Single Hour / Hour Range:

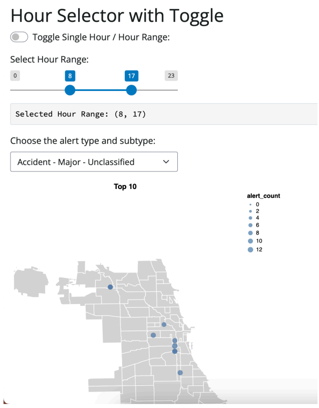
Select Single Hour:

0 8 23

```
Selected Single Hour: 8
```

Choose the alert type and subtype:

Accident - Major - Unclassified



Hour Selector with Toggle

Toggle Single Hour / Hour Range:

Select Hour Range:

0 8 17 23

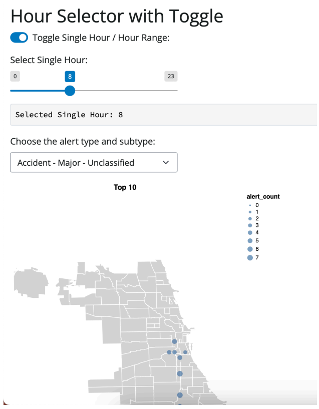
```
Selected Hour Range: (8, 17)
```

Choose the alert type and subtype:

Accident - Major - Unclassified

c.

The pictures are the same as answer in b. (Thank you ChatGPT)



Hour Selector with Toggle

● Toggle Single Hour / Hour Range:

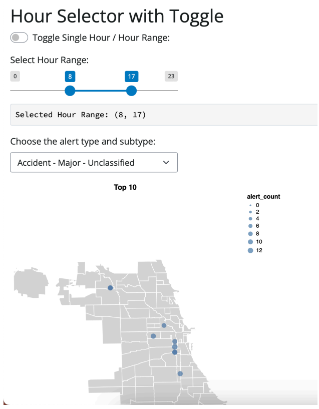
Select Single Hour:

0 8 23

```
Selected Single Hour: 8
```

Choose the alert type and subtype:

Accident - Major - Unclassified



Hour Selector with Toggle

○ Toggle Single Hour / Hour Range:

Select Hour Range:

0 8 17 23

```
Selected Hour Range: (8, 17)
```

Choose the alert type and subtype:

Accident - Major - Unclassified

d.

I plan to add a new column `period time` and categorize the hour as Morning and Afternoon. Morning time should be 6:00 AM to 12:00 PM, and afternoon time should be 12:00 PM to 18:00 PM.