Problem Set 6 - Waze Shiny Dashboard

George Wang

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: **GW**
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **GW** (2 point)
3. Late coins used this pset: **3** Late coins left after submission: **1**
4. Before starting the problem set, make sure to read and agree to the terms of data usage for the Waze data here.
5. Knit your `ps6.qmd` as a pdf document and name it `ps6.pdf`.
6. Submit 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 the gradescope repo assignment (5 points).
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.

Background

Data Download and Exploration (20 points)

1.

```
# Read in CSV file
df = pd.read_csv('waze_data/waze_data_sample.csv')

# Check data type
data_types = df.dtypes

# Ignore columns
ignore_columns = ['ts', 'geo', 'geoWKT']

# Altair data type mapping
altair_types = {
    'int64': 'Quantitative',
```

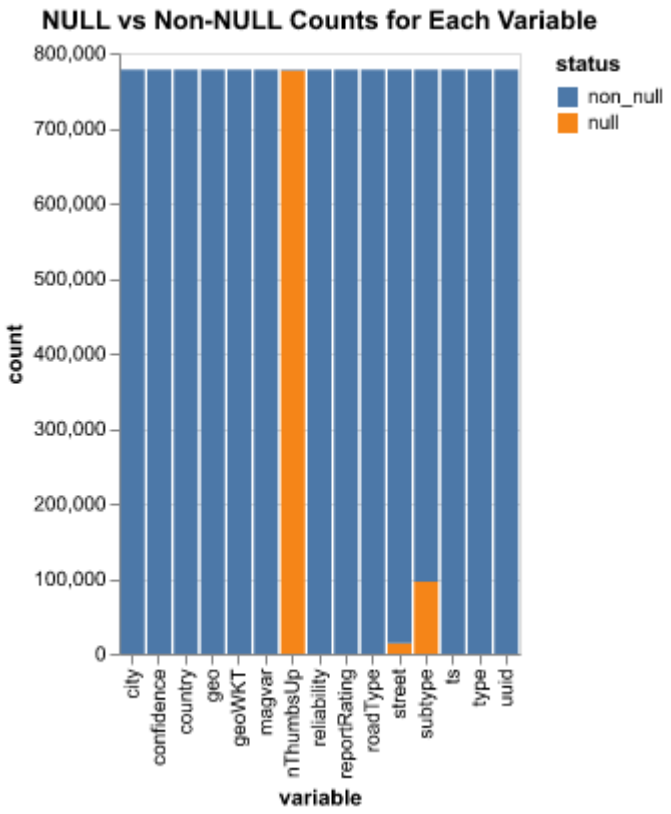
```
    'float64': 'Quantitative',
    'object': 'Nominal',
    'bool': 'Nominal',
    'datetime64[ns]': 'Temporal'
}

# Report data types
for column in df.columns:
  if column not in ignore_columns:
    if 'Unnamed' in column:
      print(f"{column}: Nominal")
    else:
      print(f"{column}: {altair_types[str(data_types[column])]}")
```

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

2.

```
# Read the wze data in geo dataframe
df_full = pd.read_csv('waze_data/waze_data.csv')

# Check for NULL values
null_counts = df_full.isnull().sum()
non_null_counts = df_full.notnull().sum()

# Create a DataFrame for visualization
null_data = pd.DataFrame({
    'variable': df_full.columns,
    'null': null_counts,
    'non_null': non_null_counts
})

# Melt the DataFrame for Altair
null_data_melted = null_data.melt(id_vars='variable', value_vars=['null', 'non_null'], var_name='status',
↪  value_name='count')

# Create the stacked bar chart
chart = alt.Chart(null_data_melted).mark_bar().encode(
    x='variable',
    y='count',
    color='status'
).properties(
    title='NULL vs Non-NULL Counts for Each Variable',
    width=200
)

chart.display()

# Print variables with NULL values and the variable with the highest share of missing observations
variables_with_nulls = null_counts[null_counts > 0]
```

```
print("Variables with NULL values:")
print(variables_with_nulls)

variable_with_highest_null_share = (null_counts / len(df_full)).idxmax()
print(f"Variable with the highest share of missing observations: {variable_with_highest_null_share}")
```




```
Variables with NULL values:
nThumbsUp    776723
street        14073
subtype       96086
dtype: int64
Variable with the highest share of missing observations: nThumbsUp
```

3.

a.

```
# Inspect the variable types and subtypes
data_types = df_full.dtypes
print(data_types)

# Create a concise crosswalk table
crosswalk = pd.DataFrame({
    'original_name': df_full.columns,
    'clean_name': [
        'city', 'confidence', 'thumbs_up', 'street', 'unique_id', 'country_id',
        ↪  'alert_type','alert_subtype', 'road_type', 'reliability', 'alert_direction', 'report_rating',
        ↪  'timestamp', 'geo', 'geoWKT'
```

```
    ],
    'description': [
        'City and state name',
        'Confidence in alert (user reactions)',
        'Number of thumbs up',
        'Street name',
        'Unique system ID',
        'Country code (ISO 3166-1)',
        'Alert type',
        'Alert sub type',
        'Road type',
        'Confidence in alert (user input)',
        'Alert direction (0-359 degrees)',
        'User rank (1-6)',
        'Timestamp of alert',
        'Geography of alert',
        'Geography of alert (WKT format)'
    ]
})

# Print the crosswalk table
print(crosswalk)
```

```
city             object
confidence        int64
nThumbsUp       float64
street           object
uuid             object
country          object
type             object
subtype          object
roadType          int64
reliability       int64
magvar            int64
reportRating      int64
ts               object
geo              object
geoWKT           object
dtype: object
   original_name       clean_name                          description
0           city             city                   City and state name
1     confidence       confidence  Confidence in alert (user reactions)
2      nThumbsUp        thumbs_up                   Number of thumbs up
3         street           street                           Street name
4           uuid        unique_id                     Unique system ID
5        country       country_id            Country code (ISO 3166-1)
6           type       alert_type                            Alert type
7        subtype    alert_subtype                        Alert sub type
8       roadType        road_type                             Road type
9    reliability      reliability     Confidence in alert (user input)
10       magvar  alert_direction      Alert direction (0-359 degrees)
11  reportRating    report_rating                      User rank (1-6)
12            ts        timestamp                   Timestamp of alert
13           geo              geo                   Geography of alert
14        geoWKT           geoWKT      Geography of alert (WKT format)
```

```
# Print unique values for 'type' and 'subtype'
unique_types = df_full['type'].unique()
unique_subtypes = df_full['subtype'].unique()

print("Unique values for 'type':")
print(unique_types)
```

```
print("\nUnique values for 'subtype':")
print(unique_subtypes)

# Count types with NA subtypes
na_subtype_count = df_full[df_full['subtype'].isna()]['type'].nunique()
print(f"\nNumber of types with a subtype that is NA: {na_subtype_count}")

# Identify types with subtypes that have enough information to consider sub-subtypes
type_subtype_combinations = df_full.groupby('type')['subtype'].nunique()
print(type_subtype_combinations)
```

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

Unique values for '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']

Number of types with a subtype that is NA: 4
type
ACCIDENT        2
HAZARD         19
JAM             4
ROAD_CLOSED     3
Name: subtype, dtype: int64
```

HAZARD is has 19 subtypes that have enough information to consider that they could have sub-subtypes. For other vairables, they do not contain much complicated information that necessitates sub-categories.

b.

```
# Replace NA with 'Unclassified'
df_full['subtype'] = df_full['subtype'].fillna('Unclassified')

# Create a mapping for clean and readable names
type_mapping = {
    'ACCIDENT': 'Accident',
    'JAM': 'Traffic Jam',
    'HAZARD': 'Hazard',
    'ROAD_CLOSED': 'Road Closed',
    'CONSTRUCTION': 'Construction',
    'EVENT': 'Event',
    'CHIT_CHAT': 'Chit Chat'
}

subtype_mapping = {
    'ACCIDENT_MAJOR': ('Major', 'Unclassified'),
    'ACCIDENT_MINOR': ('Minor', 'Unclassified'),
    'JAM_HEAVY_TRAFFIC': ('Heavy Traffic', 'Unclassified'),
    'JAM_MODERATE_TRAFFIC': ('Moderate Traffic', 'Unclassified'),
    'JAM_STAND_STILL_TRAFFIC': ('Stand Still Traffic', 'Unclassified'),
```

```
    'HAZARD_ON_ROAD': ('On Road', 'Unclassified'),
    'HAZARD_ON_ROAD_CAR_STOPPED': ('On Road', 'Car Stopped'),
    'HAZARD_ON_ROAD_CONSTRUCTION': ('On Road', 'Construction'),
    'HAZARD_ON_ROAD_EMERGENCY_VEHICLE': ('On Road', 'Emergency Vehicle'),
    'HAZARD_ON_ROAD_ICE': ('On Road', 'Ice'),
    'HAZARD_ON_ROAD_OBJECT': ('On Road', 'Object'),
    'HAZARD_ON_ROAD_POT_HOLE': ('On Road', 'Pot Hole'),
    'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT': ('On Road', 'Traffic Light Fault'),
    'HAZARD_ON_SHOULDER': ('On Shoulder', 'Unclassified'),
    'HAZARD_ON_SHOULDER_CAR_STOPPED': ('On Shoulder', 'Car Stopped'),
    'HAZARD_ON_SHOULDER_ANIMALS': ('On Shoulder', 'Animals'),
    'HAZARD_ON_SHOULDER_MISSING_SIGN': ('On Shoulder', 'Missing Sign'),
    'HAZARD_WEATHER': ('Weather', 'Unclassified'),
    'HAZARD_WEATHER_FLOOD': ('Weather', 'Flood'),
    'HAZARD_WEATHER_FOG': ('Weather', 'Fog'),
    'HAZARD_WEATHER_HEAVY_SNOW': ('Weather', 'Heavy Snow'),
    'HAZARD_WEATHER_HAIL': ('Weather', 'Hail'),
    'ROAD_CLOSED_EVENT': ('Event', 'Unclassified'),
    'ROAD_CLOSED_CONSTRUCTION': ('Construction', 'Unclassified'),
    'ROAD_CLOSED_HAZARD': ('Hazard', 'Unclassified'),
    'HAZARD_ON_ROAD_LANE_CLOSED': ('On Road', 'Lane Closed'),
    'HAZARD_ON_ROAD_ROAD_KILL': ('On Road', 'Road Kill'),
    'JAM_LIGHT_TRAFFIC': ('Light Traffic', 'Unclassified'),
    'Unclassified': ('Unclassified', 'Unclassified')
}

# Get unique combinations of type and subtype
unique_combinations = df_full[['type', 'subtype']].drop_duplicates()

# Create the hierarchical list
hierarchy = {}

for t in unique_combinations['type'].unique():
    clean_type = type_mapping.get(t, t)
    subtypes = unique_combinations[unique_combinations['type'] == t]['subtype'].unique()
    clean_subtypes = [subtype_mapping.get(st, (st, '')) for st in subtypes]
    hierarchy[clean_type] = clean_subtypes

# Print the hierarchical list
print("Hierarchical List:")
for t, subtypes in hierarchy.items():
    print(f"- {t}")
    subcategory_dict = {}
    for st, subsub in subtypes:
        if subsub:
            if st not in subcategory_dict:
                subcategory_dict[st] = []
            subcategory_dict[st].append(subsub)
        else:
            print(f"  - {st}")
    for st, subsubs in subcategory_dict.items():
        print(f"  - {st}")
        for subsub in subsubs:
            print(f"    - {subsub}")
```

```
Hierarchical List:
- Traffic Jam
  - Unclassified
    - Unclassified
  - Heavy Traffic
    - Unclassified
  - Moderate Traffic
    - Unclassified
```

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

c.

```
# The mapping of unclassified variable has been updated above (please refer to the last mapping process)
subtype_mapping

# Calculate the percentage distribution of 'Unclassified' subtypes
unclassified_percentage = (df_full['subtype'] == 'Unclassified').mean() * 100

print(f"Percentage distribution of 'Unclassified' subtypes: {unclassified_percentage:.2f}%")
```

```
Percentage distribution of 'Unclassified' subtypes: 12.35%
```

We should keep the NA subtypes because df_full have 12.35% of the whole dataset, which is a significant portion. Keeping them can make our result more representative and comprehensive.

4.

a.

::: {.cell execution_count=8}
``` {.python .cell-code}
# Define the crosswalk DataFrame with the specified columns
crosswalk_df = pd.DataFrame(columns=['type', 'subtype', 'updated_type', 'updated_subtype',
'updated_subsubtype'])

:::

b.

```
# Get unique combinations of type and subtype
unique_combinations = df_full[['type', 'subtype']].drop_duplicates()

# Create the crosswalk DataFrame
crosswalk_data = []

for _, row in unique_combinations.iterrows():
    original_type = row['type']
    original_subtype = row['subtype']
    updated_type = type_mapping.get(original_type, original_type)
    updated_subtype, updated_subsubtype = subtype_mapping.get(original_subtype, (original_subtype, ''))
    crosswalk_data.append([original_type, original_subtype, updated_type, updated_subtype,
↪  updated_subsubtype])

crosswalk_df = pd.DataFrame(crosswalk_data, columns=['type', 'subtype', 'updated_type',
↪  'updated_subtype', 'updated_subsubtype'])

# Print the crosswalk DataFrame
print(crosswalk_df)
```

```
           type                            subtype updated_type  \
0           JAM                        Unclassified  Traffic Jam
1      ACCIDENT                        Unclassified     Accident
2   ROAD_CLOSED                        Unclassified  Road Closed
3        HAZARD                        Unclassified       Hazard
4      ACCIDENT                      ACCIDENT_MAJOR     Accident
5      ACCIDENT                      ACCIDENT_MINOR     Accident
6        HAZARD                      HAZARD_ON_ROAD       Hazard
7        HAZARD          HAZARD_ON_ROAD_CAR_STOPPED       Hazard
8        HAZARD         HAZARD_ON_ROAD_CONSTRUCTION       Hazard
9        HAZARD    HAZARD_ON_ROAD_EMERGENCY_VEHICLE       Hazard
10       HAZARD                 HAZARD_ON_ROAD_ICE       Hazard
11       HAZARD               HAZARD_ON_ROAD_OBJECT       Hazard
12       HAZARD             HAZARD_ON_ROAD_POT_HOLE       Hazard
13       HAZARD  HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT       Hazard
14       HAZARD                 HAZARD_ON_SHOULDER       Hazard
15       HAZARD      HAZARD_ON_SHOULDER_CAR_STOPPED       Hazard
16       HAZARD                      HAZARD_WEATHER       Hazard
17       HAZARD                HAZARD_WEATHER_FLOOD       Hazard
18          JAM                   JAM_HEAVY_TRAFFIC  Traffic Jam
19          JAM                JAM_MODERATE_TRAFFIC  Traffic Jam
20          JAM             JAM_STAND_STILL_TRAFFIC  Traffic Jam
21  ROAD_CLOSED                   ROAD_CLOSED_EVENT  Road Closed
22       HAZARD          HAZARD_ON_ROAD_LANE_CLOSED       Hazard
23       HAZARD                 HAZARD_WEATHER_FOG       Hazard
24  ROAD_CLOSED            ROAD_CLOSED_CONSTRUCTION  Road Closed
25       HAZARD            HAZARD_ON_ROAD_ROAD_KILL       Hazard
26       HAZARD          HAZARD_ON_SHOULDER_ANIMALS       Hazard
27       HAZARD     HAZARD_ON_SHOULDER_MISSING_SIGN       Hazard
28          JAM                   JAM_LIGHT_TRAFFIC  Traffic Jam
```
```

```
29       HAZARD           HAZARD_WEATHER_HEAVY_SNOW       Hazard
30  ROAD_CLOSED                 ROAD_CLOSED_HAZARD  Road Closed
31       HAZARD                HAZARD_WEATHER_HAIL       Hazard

        updated_subtype   updated_subsubtype
0          Unclassified         Unclassified
1          Unclassified         Unclassified
2          Unclassified         Unclassified
3          Unclassified         Unclassified
4                 Major         Unclassified
5                 Minor         Unclassified
6               On Road         Unclassified
7               On Road          Car Stopped
8               On Road         Construction
9               On Road    Emergency Vehicle
10              On Road                  Ice
11              On Road               Object
12              On Road             Pot Hole
13              On Road  Traffic Light Fault
14           On Shoulder         Unclassified
15           On Shoulder          Car Stopped
16              Weather         Unclassified
17              Weather                Flood
18        Heavy Traffic         Unclassified
19     Moderate Traffic         Unclassified
20  Stand Still Traffic         Unclassified
21                Event         Unclassified
22              On Road          Lane Closed
23              Weather                  Fog
24         Construction         Unclassified
25              On Road            Road Kill
26           On Shoulder              Animals
27           On Shoulder         Missing Sign
28        Light Traffic         Unclassified
29              Weather           Heavy Snow
30               Hazard         Unclassified
31              Weather                 Hail
```

c.

```
# Merge the crosswalk with the original data
merged_df = pd.merge(df_full, crosswalk_df, on=['type', 'subtype'])

# Save the merged DataFrame to a CSV file
merged_df.to_csv("merged_data.csv", index=False)

# Count the rows for Accident - Unclassified
accident_unclassified_count = merged_df[(merged_df['updated_type'] == 'Accident') &
↪  (merged_df['updated_subtype'] == 'Unclassified')].shape[0]

print(f"Number of rows for Accident - Unclassified: {accident_unclassified_count}")
```

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

d.

```
# Check if the values in type and subtype are the same in both datasets
type_check = (merged_df['type'] == merged_df['updated_type']).all()
subtype_check = (merged_df['subtype'] == merged_df['updated_subtype']).all()

print(f"Do the 'type' values match in both datasets? {'Yes' if type_check else 'No'}")
print(f"Do the 'subtype' values match in both datasets? {'Yes' if subtype_check else 'No'}")
```

```
Do the 'type' values match in both datasets? No
Do the 'subtype' values match in both datasets? No
```

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

1.

a.

```python
# Define a function to extract each of them
def extract_coordinates(geo):
    pattern = r"POINT\(([-+]?\d*\.\d+|\d+) ([-+]?\d*\.\d+|\d+)\)"
    match = re.match(pattern, geo)
    if match:
        return float(match.group(2)), float(match.group(1))  # latitude, longitude
    return None, None

# Extract coordinates into a new DataFrame
coordinates = merged_df["geo"].apply(extract_coordinates)

coordinates_df = pd.DataFrame(coordinates.tolist(), columns=["latitude", "longitude"])

# Display the new DataFrame
print(coordinates_df)
```

```
         latitude  longitude
0       41.929692 -87.676685
1       41.939093 -87.680139
2       41.916580 -87.735235
3       41.898673 -87.618122
4       41.900655 -87.619477
...           ...        ...
778089  41.927996 -87.641237
778090  41.976969 -87.876461
778091  41.982756 -87.806445
778092  41.661887 -87.589164
778093  41.970322 -87.759834

[778094 rows x 2 columns]
```

b.

```python
# Bin the latitude and longitude variables
coordinates_df['latitude_bin'] = coordinates_df['latitude'].apply(lambda x: round(x, 2))
coordinates_df['longitude_bin'] = coordinates_df['longitude'].apply(lambda x: round(x, 2))

# Combine the binned latitude and longitude into a single column
coordinates_df['lat_long_bin'] = list(zip(coordinates_df['latitude_bin'],
↪  coordinates_df['longitude_bin']))
merged_df['lat_long_bin'] = list(zip(coordinates_df['latitude_bin'], coordinates_df['longitude_bin']))

# Save the merged DataFrame to a CSV file
coordinates_df.to_csv("coordinates_df.csv", index=False)

# Find the binned latitude-longitude combination with the greatest number of observations
most_common_bin = coordinates_df['lat_long_bin'].value_counts().idxmax()
most_common_bin_count = coordinates_df['lat_long_bin'].value_counts().max()

print(f"The binned latitude-longitude combination with the greatest number of observations is:
↪  {most_common_bin}")
print(f"Number of observations: {most_common_bin_count}")
```

```
The binned latitude-longitude combination with the greatest number of observations is: (41.88, -87.65)
Number of observations: 21325
```

c.

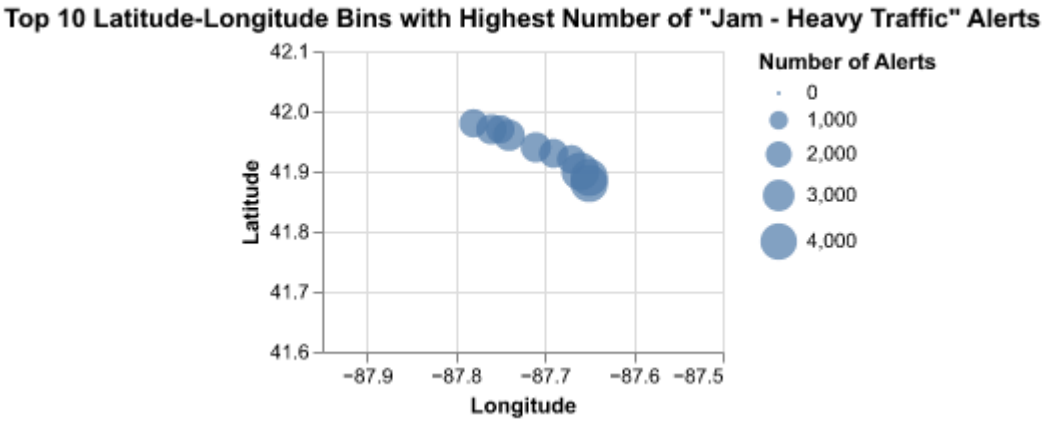
```
# Filter the data for a chosen type and subtype
chosen_type = 'Accident'
chosen_subtype = 'Unclassified'
filtered_df = merged_df[(merged_df['updated_type'] == chosen_type) & (merged_df['updated_subtype'] ==
↪  chosen_subtype)]

# Aggregate the data to find the top 10
top_alerts =
↪  coordinates_df.loc[filtered_df.index].groupby('lat_long_bin').size().reset_index(name='count')
top_10_alerts = top_alerts.nlargest(10, 'count')

# Create the directory
os.makedirs('top_alerts_map', exist_ok=True)

# Save the DataFrame as a CSV file
top_10_alerts.to_csv('top_alerts_map/top_alerts_map.csv', index=False)

# Display the level of aggregation and the number of rows
level_of_aggregation = 'latitude-longitude bin'
number_of_rows = top_alerts.shape[0]

print(f"Level of aggregation: {level_of_aggregation}")
print(f"Number of rows: {number_of_rows}")
print(top_10_alerts)
```

```
Level of aggregation: latitude-longitude bin
Number of rows: 630
       lat_long_bin  count
391   (41.9, -87.66)    561
357  (41.88, -87.65)    536
374  (41.89, -87.65)    452
395   (41.9, -87.62)    437
310  (41.85, -87.64)    363
323  (41.86, -87.64)    352
341  (41.87, -87.64)    340
252  (41.81, -87.63)    307
532  (41.97, -87.75)    279
297  (41.84, -87.63)    268
```

2.

```
# Filter the data for 'Jam - Heavy Traffic'
chosen_type = 'Traffic Jam'
chosen_subtype = 'Heavy Traffic'
filtered_df = merged_df[(merged_df['updated_type'] == chosen_type) & (merged_df['updated_subtype'] ==
↪  chosen_subtype)]

# Aggregate the data to find the top 10
top_alerts =
↪  coordinates_df.loc[filtered_df.index].groupby('lat_long_bin').size().reset_index(name='count')
top_10_alerts = top_alerts.nlargest(10, 'count')

# Extract latitude and longitude from the bins
top_10_alerts[['latitude_bin', 'longitude_bin']] = pd.DataFrame(top_10_alerts['lat_long_bin'].tolist(),
↪  index=top_10_alerts.index)

# Create the scatter plot
```

```
scatter_plot = alt.Chart(top_10_alerts).mark_circle().encode(
    x=alt.X('longitude_bin:Q', title='Longitude', scale=alt.Scale(domain=[-87.95, -87.5])),
    y=alt.Y('latitude_bin:Q', title='Latitude', scale=alt.Scale(domain=[41.6, 42.1])),
    size=alt.Size('count:Q', title='Number of Alerts'),
    tooltip=['latitude_bin', 'longitude_bin', 'count']
).properties(
    title='Top 10 Latitude-Longitude Bins with Highest Number of "Jam - Heavy Traffic" Alerts',
    width=200,
    height=150
)

scatter_plot.display()
```



3.

a.

```
# URL of the GeoJSON file
url = "https://data.cityofchicago.org/resource/igwz-8jzy.geojson"

# Send a GET request to download the file
response = requests.get(url)

# Save the GeoJSON file
geojson_path = "./top_alerts_map/chicago-boundaries.geojson"
with open(geojson_path, 'wb') as f:
    f.write(response.content)

print(f"GeoJSON file downloaded and saved to {geojson_path}")
```

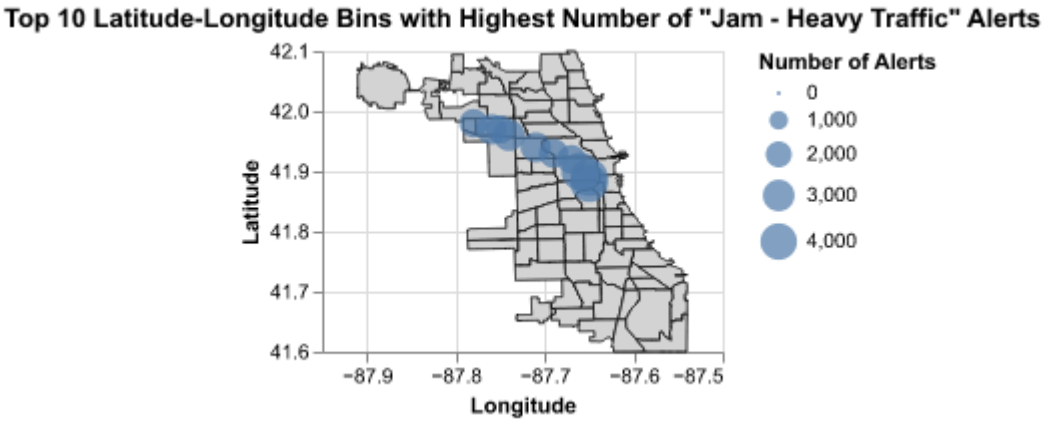
```
GeoJSON file downloaded and saved to ./top_alerts_map/chicago-boundaries.geojson
```

b.

```
# Locate the directory
file_path = "./top_alerts_map/chicago-boundaries.geojson"

# Open and load the geojson
with open(file_path) as f:
    chicago_geojson = json.load(f)

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

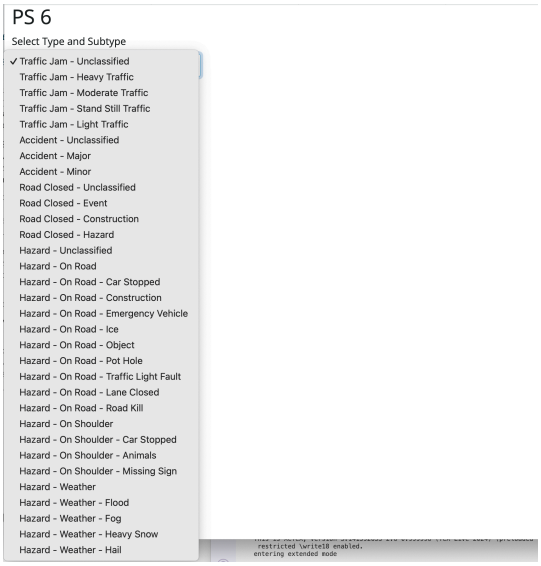
4.

```
# Create the base map using the GeoJSON data
base_map = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='black'
).encode(
    tooltip=['properties.name:N']
).project(
    type='equirectangular'
).properties(
    width=200,
    height=150
)

# Layer the scatter plot on top of the base map
combined_plot = base_map + scatter_plot

# Display the combined plot
combined_plot.display()
```

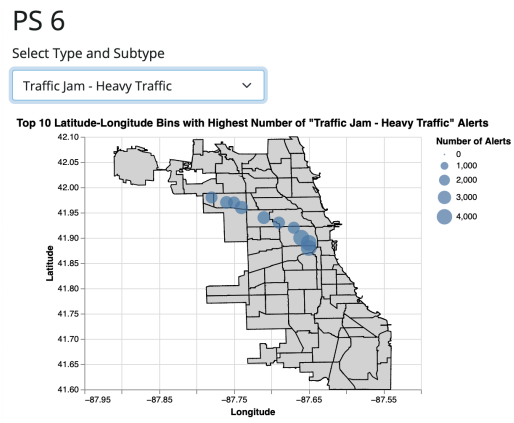


5.

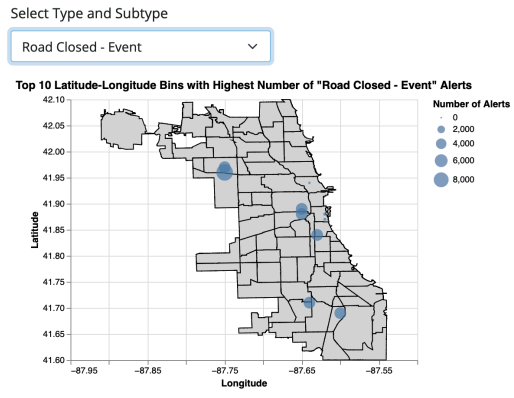
a.



b.



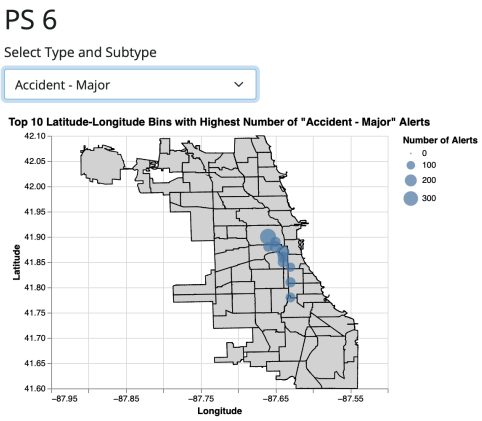
c.



The most common locations for road closures

due to events are represented by the largest blue circles, indicating the areas with the highest number of alerts. Central and North-Central Chicago has the highest density of road closure alerts due to events. South-Central Chicago also has smaller clusters scattered across other regions. This suggests that road closures for events are concentrated in areas with higher activity, likely due to event venues or major roads in these regions.

d.



It can answer where do major accident usually occur. Based on the provided visualization, major accidents are most commonly reported in South-Central Chicago, particularly along major traffic corridors (very likely due to high way traffic and complicated road signs). The clusters indicate hotspots that might benefit from additional traffic monitoring.

e. To make the dashboard more insightful, we can include a column for the hour of the day or day of the week. This will help visualize patterns of incidents over time and identify peak periods for different types of alerts.

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

1.

a. It would not be a good idea to collapse the dataset solely by ts. Timestamps are typically very granular, often down to the second. Collapsing the dataset by such a granular column will result inmany unique groups, making it difficult to analyze the data effectively. Thus, if the goal is to analyze patterns over time, it would be more useful to aggregate the data into larger time intervals, such as hours, allowing for more meaningful insights into trends.

b.

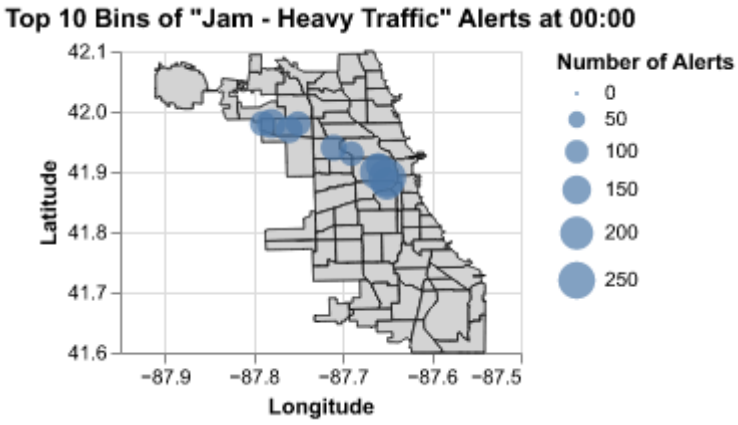
```
# Parse the 'ts' column to datetime and extract the hour
merged_df['hour'] = pd.to_datetime(merged_df['ts']).dt.strftime('%H:00')

# Create the directory
os.makedirs('top_alerts_map_byhour', exist_ok=True)

# Step 3: Collapse the dataset
# Group by hour, type, subtype, and location to calculate the count of alerts
collapsed_df = merged_df.groupby(['hour', 'updated_type', 'updated_subtype',
↪  'lat_long_bin']).size().reset_index(name='count')

# Save the collapsed dataset as a CSV file
collapsed_df.to_csv('top_alerts_map_byhour/top_alerts_map_byhour.csv', index=False)

# Print
number_of_rows = collapsed_df.shape[0]
print(f"Number of rows in the collapsed dataset: {number_of_rows}")
```

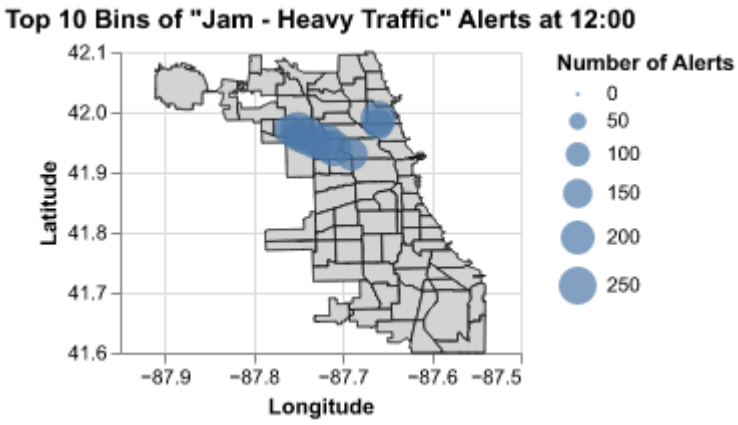
```
Number of rows in the collapsed dataset: 62824
```

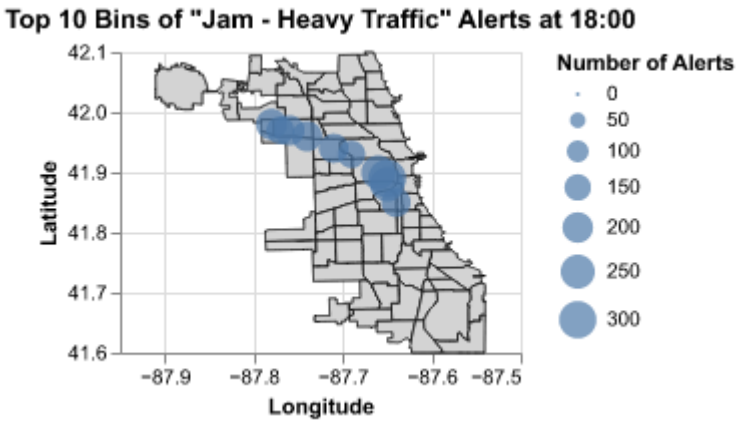
c.

```
# Define the hours to plot
hours_to_plot = ['00:00', '12:00', '18:00']

# Filter the data for 'Jam - Heavy Traffic' and the specified hours
chosen_type_subtype = 'Traffic Jam - Heavy Traffic'
chosen_type, chosen_subtype = chosen_type_subtype.split(' - ')

filtered_df = collapsed_df[(collapsed_df['updated_type'] == chosen_type) &
↪  (collapsed_df['updated_subtype'] == chosen_subtype) & (collapsed_df['hour'].isin(hours_to_plot))]
print(filtered_df)
# Create individual plots for each hour
for hour in hours_to_plot:
    hour_df = filtered_df[filtered_df['hour'] == hour]

    # Extract latitude and longitude from the bins and select top 10
    hour_df[['latitude_bin', 'longitude_bin']] = pd.DataFrame(hour_df['lat_long_bin'].tolist(),
↪   index=hour_df.index, columns=['latitude_bin', 'longitude_bin'])
    top_10_bins = hour_df.nlargest(10, 'count')


    # Create the scatter plot
    scatter_plot = alt.Chart(top_10_bins).mark_circle().encode(
        x=alt.X('longitude_bin:Q', title='Longitude', scale=alt.Scale(domain=[-87.95, -87.5])),
        y=alt.Y('latitude_bin:Q', title='Latitude', scale=alt.Scale(domain=[41.6, 42.1])),
        size=alt.Size('count:Q', title='Number of Alerts'),
        tooltip=['latitude_bin', 'longitude_bin', 'count']
    ).properties(
        title=f'Top 10 Bins of "Jam - Heavy Traffic" Alerts at {hour}',
        width=200,
        height=150
    )

    # Layer the scatter plot on top of the base map
    combined_plot = base_map + scatter_plot

    # Display the combined plot
    combined_plot.display()
```

```
       hour updated_type updated_subtype     lat_long_bin  count
1802   00:00  Traffic Jam   Heavy Traffic  (41.65, -87.58)      9
1803   00:00  Traffic Jam   Heavy Traffic  (41.66, -87.62)      2
1804   00:00  Traffic Jam   Heavy Traffic  (41.66, -87.59)     14
1805   00:00  Traffic Jam   Heavy Traffic  (41.66, -87.58)      1
1806   00:00  Traffic Jam   Heavy Traffic  (41.66, -87.57)      1
...      ...          ...             ...              ...    ...
42885  18:00  Traffic Jam   Heavy Traffic  (42.01, -87.69)     13
42886  18:00  Traffic Jam   Heavy Traffic  (42.01, -87.68)      6
42887  18:00  Traffic Jam   Heavy Traffic  (42.01, -87.67)      2
42888  18:00  Traffic Jam   Heavy Traffic  (42.01, -87.66)      5
42889  18:00  Traffic Jam   Heavy Traffic  (42.02, -87.68)      2

[1268 rows x 5 columns]
```


Top 10 Bins of "Jam - Heavy Traffic" Alerts at 00:00
Latitude
42.1
42.0
41.9
41.8
41.7
41.6
−87.9
−87.8
−87.7
−87.6
−87.5
Longitude
Number of Alerts
0
50
100
150
200
250


Top 10 Bins of "Jam - Heavy Traffic" Alerts at 12:00
Latitude
42.1
42.0
41.9
41.8
41.7
41.6
−87.9
−87.8
−87.7
−87.6
−87.5
Longitude
Number of Alerts
0
50
100
150
200
250


Top 10 Bins of "Jam - Heavy Traffic" Alerts at 18:00
Latitude
42.1
42.0
41.9
41.8
41.7
41.6
−87.9
−87.8
−87.7
−87.6
−87.5
Longitude
Number of Alerts
0
50
100
150
200
250
300

2.

a.

Shiny App

Select Type and Subtype

Option 2

Select Hour

0 5 23

b.

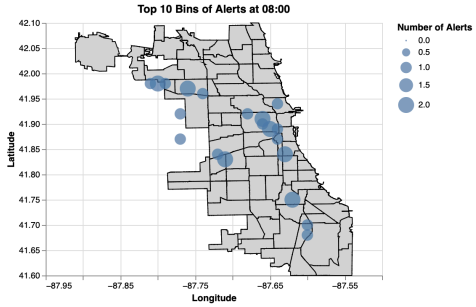
PS6 App2

Select Type and Subtype

Traffic Jam - Heavy Traffic

Select Hour

08:00



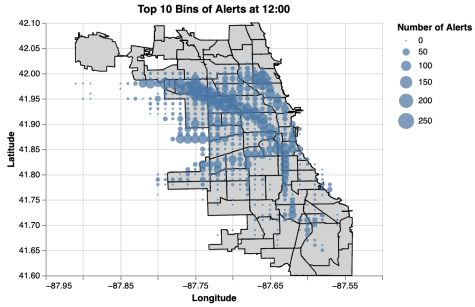
PS6 App2

Select Type and Subtype

Traffic Jam - Heavy Traffic

Select Hour

12:00



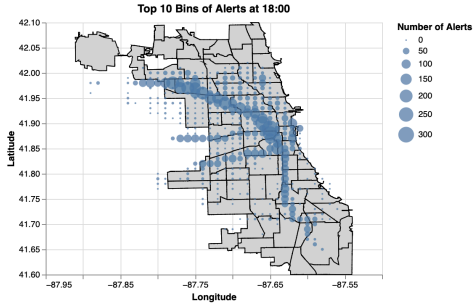
PS6 App2

Select Type and Subtype

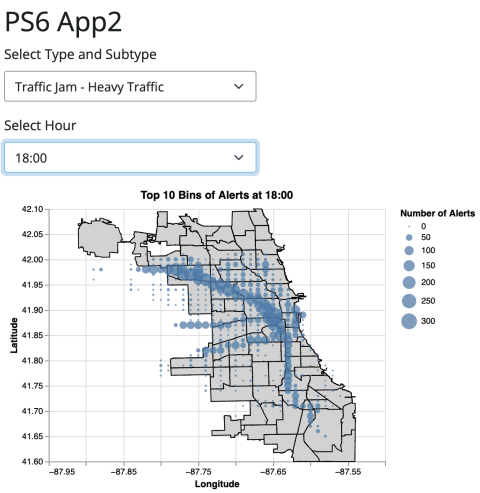
Traffic Jam - Heavy Traffic

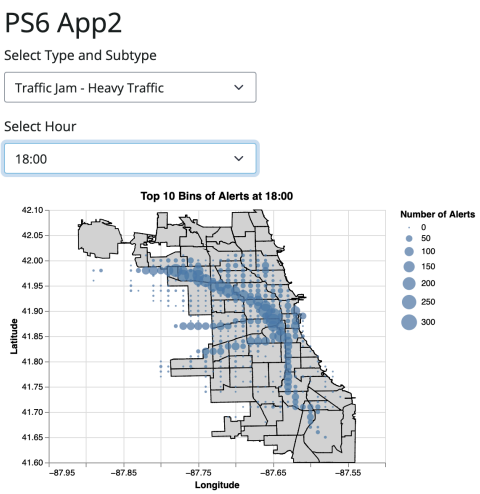
Select Hour

18:00



c.





Road construction is done more during night hours than morning hours.

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

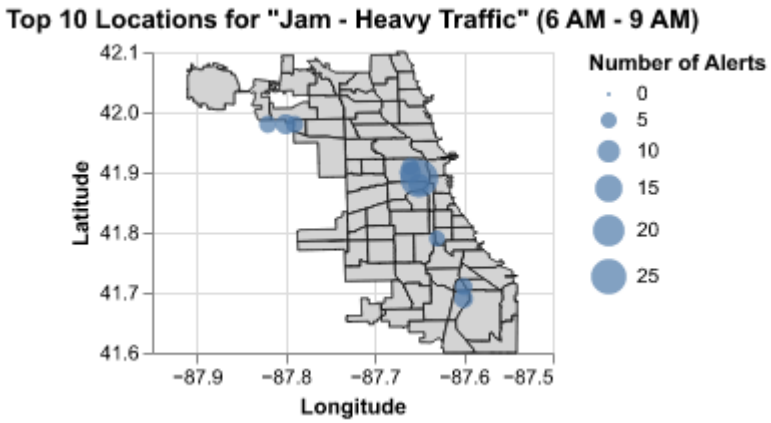
1.

a. Collapsing the dataset by a range of hours can be a useful approach. First, it reduces the size of the dataset, which makes data processing and visualization in the Shiny app more efficient. Additionally, it helps identify broader patterns, such as traffic jams or road hazards that occur more frequently during specific time blocks like the morning hours. This approach also simplifies user interaction by allowing users to focus on larger time intervals rather than examining data for each hour individually, making visualizations clearer by reducing noise.

b.

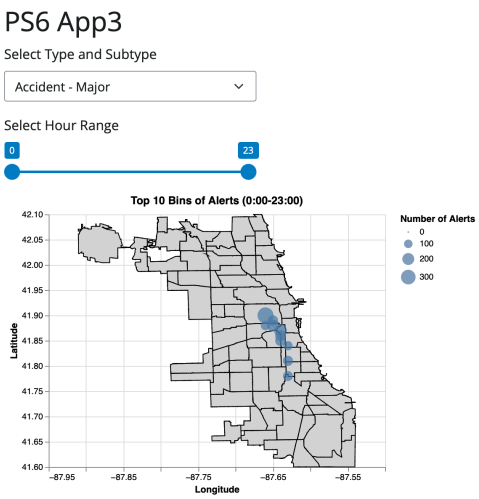
```
# Define the range of hours
hours_range = ['06:00', '07:00', '08:00', '09:00']

# Filter the dataset for 'Jam - Heavy Traffic' and the specified hours
chosen_type_subtype = 'Traffic Jam - Heavy Traffic'
```

```
chosen_type, chosen_subtype = chosen_type_subtype.split(' - ')

filtered_df = collapsed_df[
    (collapsed_df['updated_type'] == chosen_type) &
    (collapsed_df['updated_subtype'] == chosen_subtype) &
    (collapsed_df['hour'].isin(hours_range))
]

# Aggregate counts across the hour range
aggregated_df = filtered_df.groupby('lat_long_bin').agg({'count': 'sum'}).reset_index()

# Select the top 10 locations
top_10_bins = aggregated_df.nlargest(10, 'count')

# Extract latitude and longitude
top_10_bins[['latitude_bin', 'longitude_bin']] = pd.DataFrame(
    top_10_bins['lat_long_bin'].tolist(),
    index=top_10_bins.index,
    columns=['latitude_bin', 'longitude_bin']
)

# Create the scatter plot
scatter_plot = alt.Chart(top_10_bins).mark_circle().encode(
    x=alt.X('longitude_bin:Q', title='Longitude', scale=alt.Scale(domain=[-87.95, -87.5])),
    y=alt.Y('latitude_bin:Q', title='Latitude', scale=alt.Scale(domain=[41.6, 42.1])),
    size=alt.Size('count:Q', title='Number of Alerts'),
    tooltip=['latitude_bin', 'longitude_bin', 'count']
).properties(
    title='Top 10 Locations for "Jam - Heavy Traffic" (6 AM - 9 AM)',
    width=200,
    height=150
)

# Layer the scatter plot on top of the base map
combined_plot = base_map + scatter_plot

# Display
combined_plot.display()
```

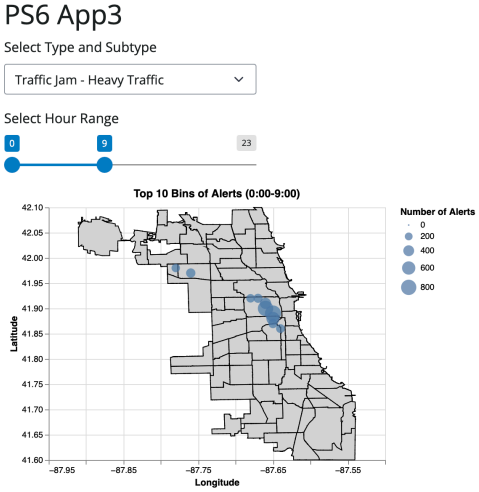


2.

a.

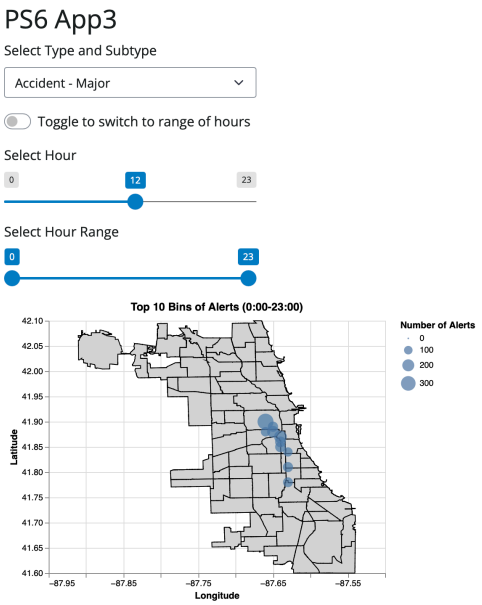

PS6 App3
Select Type and Subtype
Accident - Major
Select Hour Range
0
23
Top 10 Bins of Alerts (0:00-23:00)
Number of Alerts
0
100
200
300
Latitude
Longitude
42.10
42.05
42.00
41.95
41.90
41.85
41.80
41.75
41.70
41.65
41.60
−87.95
−87.85
−87.75
−87.65
−87.55

b.


PS6 App3
Select Type and Subtype
Traffic Jam - Heavy Traffic
Select Hour Range
0
9
23
Top 10 Bins of Alerts (0:00-9:00)
Number of Alerts
0
200
400
600
800
Latitude
Longitude
42.10
42.05
42.00
41.95
41.90
41.85
41.80
41.75
41.70
41.65
41.60
−87.95
−87.85
−87.75
−87.65
−87.55

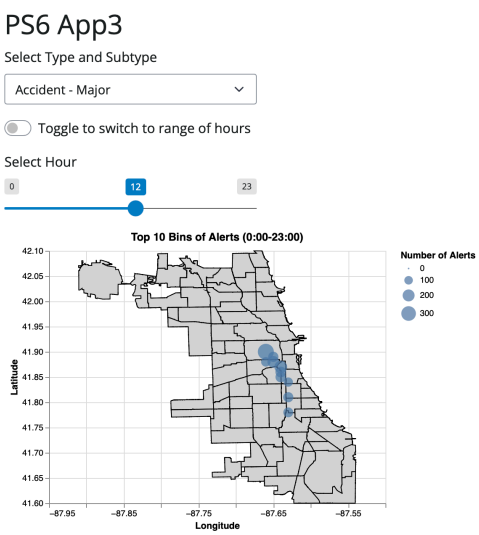
3.

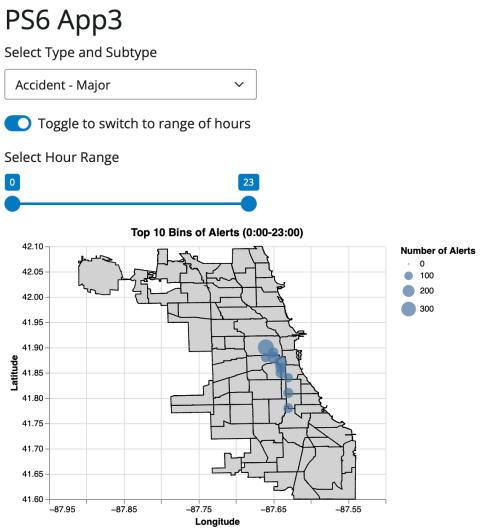
a.



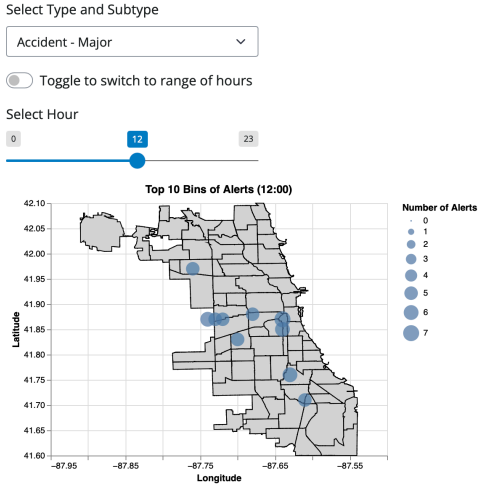
The possible values for the input.switch_button are True and False. When the switch button is toggled on, the value will be True, and when it is toggled off, the value will be False. This allows the app to conditionally display single hourslider or the hour range slider.

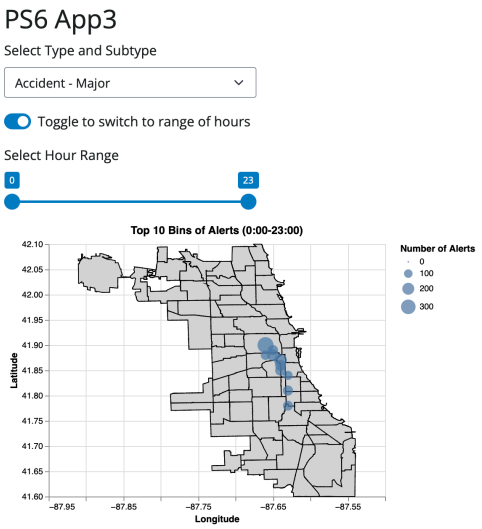
b.




PS6 App3
Select Type and Subtype
Accident - Major
Toggle to switch to range of hours
Select Hour Range
0
23
Top 10 Bins of Alerts (0:00-23:00)
Number of Alerts
0
100
200
300
Latitude
Longitude
42.10
42.05
42.00
41.95
41.90
41.85
41.80
41.75
41.70
41.65
41.60
−87.95
−87.85
−87.75
−87.65
−87.55

c.


PS6 App3
Select Type and Subtype
Accident - Major
Toggle to switch to range of hours
Select Hour
0
12
23
Top 10 Bins of Alerts (12:00)
Number of Alerts
0
1
2
3
4
5
6
7
Latitude
Longitude
42.10
42.05
42.00
41.95
41.90
41.85
41.80
41.75
41.70
41.65
41.60
−87.95
−87.85
−87.75
−87.65
−87.55



d.

To achieve this plot, the dataset needs to include a new column categorizing hours into specific time periods, such as "Morning" (6 AM–12 PM) and "Afternoon" (12 PM–6 PM).

Next, the plot should use color encoding in Altair to visually differentiate time periods, assigning distinct colors (red for Morning and blue for Afternoon) to make patterns more obvious.

Additionally, the data should be aggregated for each time period, summing up counts with specific time period. The aggregated results can then be combined into a single dataset, allowing both periods to be displayed simultaneously on the map.