title: "Problem Set 6 - Waze Shiny Dashboard"

author: "Peter Ganong, Maggie Shi, and Andre Oviedo"

date: 11/19/2024

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

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!

Data Download and Exploration (20 points)

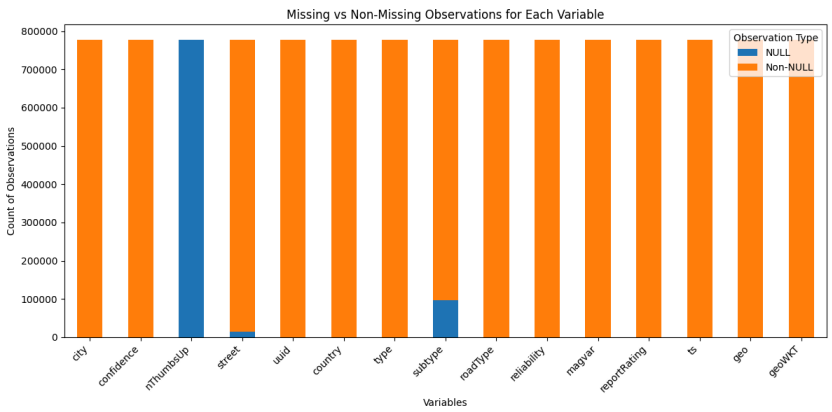
1.

```
import pandas as pd

sample_csv_path = '/content/waze_data_sample.csv'  # Replace with the correct path
waze_sample_df = pd.read_csv(sample_csv_path)

excluded_columns = ['ts', 'geo', 'geoWKT']
filtered_columns = waze_sample_df.drop(columns=excluded_columns, errors='ignore')

variable_data_types = filtered_columns.dtypes

altair_mapping = {
    'int64': 'Quantitative',
    'float64': 'Quantitative',
    'object': 'Nominal',
    'datetime64[ns]': 'Temporal',
    'bool': 'Nominal'
}
altair_data_types = {col: altair_mapping[str(dtype)] for col, dtype in variable_data_types.items()}

print(altair_data_types)
```

```
{'Unnamed: 0': 'Quantitative', 'city': 'Nominal', 'confidence': 'Quantitative', 'nThumbsUp': 'Quantitative', 'street': 'Nominal', 'uuid': 'Nominal', 'country': 'Nominal', 'type': 'Nominal', 'subtype': 'Nominal', 'roadType': 'Quantitative', 'reliability': 'Quantitative'
```

2.

```
import pandas as pd
import matplotlib.pyplot as plt

full_csv_path = '/content/waze_data.csv'
waze_full_df = pd.read_csv(full_csv_path)

null_counts = waze_full_df.isnull().sum()
non_null_counts = waze_full_df.notnull().sum()

null_summary_df = pd.DataFrame({
    'NULL': null_counts,
    'Non-NULL': non_null_counts
})

ax = null_summary_df.plot(kind='bar', stacked=True, figsize=(12, 6))
plt.title("Missing vs Non-Missing Observations for Each Variable")
plt.ylabel("Count of Observations")
plt.xlabel("Variables")
plt.xticks(rotation=45, ha='right')
plt.legend(title="Observation Type")
plt.tight_layout()
plt.show()

variables_with_nulls = null_counts[null_counts > 0]
highest_null_share_variable = (variables_with_nulls / len(waze_full_df)).idxmax()

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

print("\nVariable with the highest share of missing values:")
print(highest_null_share_variable)
```




```
Variables with NULL values:
nThumbsUp    776723
street        14073
subtype       96086
```

3. a & b & c

```
full_csv_path = '/content/waze_data.csv'
waze_full_df = pd.read_csv(full_csv_path)

unique_types = waze_full_df['type'].unique()
unique_subtypes = waze_full_df['subtype'].unique()

types_with_na_subtype = waze_full_df[waze_full_df['subtype'].isnull()]['type'].unique()
count_types_with_na_subtype = len(types_with_na_subtype)

types_with_meaningful_subtypes = waze_full_df[waze_full_df['subtype'].notnull()].groupby('type')['subtype'].nunique()
types_with_meaningful_subtypes = types_with_meaningful_subtypes[types_with_meaningful_subtypes > 1]
```

```
hierarchy = {}
for type_value in types_with_meaningful_subtypes.index:
    subtypes = waze_full_df[waze_full_df['type'] == type_value]['subtype'].dropna().unique()
    hierarchy[type_value.replace('_', ' ').title()] = [subtype.replace('_', ' ').title() for subtype in subtypes]

waze_full_df['subtype'] = waze_full_df['subtype'].fillna('Unclassified')

print("Unique values for 'type':", unique_types)
print("\nUnique values for 'subtype':", unique_subtypes)
print(f"\nNumber of types with NA subtypes: {count_types_with_na_subtype}\n")
print("Hierarchy of types and subtypes:")
for key, values in hierarchy.items():
    print(f"- {key}:")
    for value in values:
        print(f"  - {value}")
print("\nNA subtypes have been replaced with 'Unclassified'.")
```

```
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 NA subtypes: 4

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

NA subtypes have been replaced with 'Unclassified'.
```

Yes, I think we should keep the NA subtypes because first, we need to preserve Information that NA subtypes might represent meaningful but unclassified or unspecified events. Dropping them could lead to a loss of valuable context or data for analysis. Second, we need to ensure completeness that if NA subtypes are removed, datasets may become incomplete and inconsistent for visualization and further analysis. Retaining NA subtypes ensures that all events are included and appropriately handled.

4

a.

```
import pandas as pd

crosswalk = waze_full_df[['type', 'subtype']].drop_duplicates().reset_index(drop=True)

crosswalk['updated_type'] = crosswalk['type'].str.replace('_', ' ').str.title()
crosswalk['updated_subtype'] = crosswalk['subtype'].str.replace('_', ' ').str.title()

crosswalk['updated_subsubtype'] = crosswalk['updated_subtype']

print(crosswalk.head())

crosswalk.to_csv('crosswalk_with_hierarchy.csv', index=False)
```

```
           type         subtype updated_type updated_subtype updated_subsubtype
0           JAM    Unclassified          Jam    Unclassified       Unclassified
1     ACCIDENT    Unclassified     Accident    Unclassified       Unclassified
2  ROAD_CLOSED    Unclassified  Road Closed    Unclassified       Unclassified
3       HAZARD    Unclassified       Hazard    Unclassified       Unclassified
4     ACCIDENT  ACCIDENT_MAJOR     Accident  Accident Major     Accident Major
```

b.

```
import pandas as pd

crosswalk = waze_full_df[['type', 'subtype']].drop_duplicates().reset_index(drop=True)

crosswalk['subtype'] = crosswalk['subtype'].fillna('Unclassified')

crosswalk['updated_type'] = crosswalk['type'].str.replace('_', ' ').str.title()
crosswalk['updated_subtype'] = crosswalk['subtype'].str.replace('_', ' ').str.title()
crosswalk['updated_subsubtype'] = crosswalk['updated_subtype']  # Assign similar to subtype for simplicity

print("Number of rows in the crosswalk:", len(crosswalk))
print(crosswalk)

crosswalk.to_csv('crosswalk_with_hierarchy.csv', index=False)
```

```
Number of rows in the crosswalk: 32
            type                            subtype updated_type  \
0            JAM                       Unclassified          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          Jam
19          JAM           JAM_MODERATE_TRAFFIC          Jam
20          JAM         JAM_STAND_STILL_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          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                    Accident Major                    Accident Major
5                    Accident Minor                    Accident Minor
6                      Hazard On Road                    Hazard On Road
```

```
7          Hazard On Road Car Stopped          Hazard On Road Car Stopped
8         Hazard On Road Construction         Hazard On Road Construction
9    Hazard On Road Emergency Vehicle    Hazard On Road Emergency Vehicle
10                Hazard On Road Ice                  Hazard On Road Ice
11             Hazard On Road Object               Hazard On Road Object
12           Hazard On Road Pot Hole             Hazard On Road Pot Hole
13  Hazard On Road Traffic Light Fault  Hazard On Road Traffic Light Fault
14                Hazard On Shoulder                  Hazard On Shoulder
15     Hazard On Shoulder Car Stopped      Hazard On Shoulder Car Stopped
16                     Hazard Weather                      Hazard Weather
17               Hazard Weather Flood                Hazard Weather Flood
18                  Jam Heavy Traffic                   Jam Heavy Traffic
19               Jam Moderate Traffic                Jam Moderate Traffic
20            Jam Stand Still Traffic             Jam Stand Still Traffic
21                  Road Closed Event                   Road Closed Event
```

c.

```
import pandas as pd

merged_df = pd.merge(
    waze_full_df,
    crosswalk,
    on=['type', 'subtype'],
    how='left'
)

accident_unclassified_rows = merged_df[
    (merged_df['updated_type'] == 'Accident') &
    (merged_df['updated_subtype'] == 'Unclassified')
]

num_accident_unclassified = len(accident_unclassified_rows)

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

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

d.

```
crosswalk_unique_values = set(crosswalk[['type', 'subtype']].apply(tuple, axis=1))
merged_unique_values = set(merged_df[['type', 'subtype']].apply(tuple, axis=1))

consistent = crosswalk_unique_values == merged_unique_values

mismatched_values = crosswalk_unique_values.symmetric_difference(merged_unique_values)

print(f"Is the crosswalk consistent with the merged dataset? {consistent}")
if not consistent:
    print("Mismatched values:")
    print(mismatched_values)
```

```
Is the crosswalk consistent with the merged dataset? True
```

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

1.

a.

```
import pandas as pd
import re

def extract_coordinates(geo):
    if isinstance(geo, str):
        match = re.match(r"POINT\s*\((-?\d+\.\d+)\s+(-?\d+\.\d+)\)", geo)
        if match:
            return float(match.group(2)), float(match.group(1))
    return None, None

merged_df['latitude'], merged_df['longitude'] = zip(*merged_df['geo'].apply(extract_coordinates))
print(merged_df[['geo', 'latitude', 'longitude']].head())
```

```
                            geo   latitude  longitude
0  POINT(-87.676685 41.929692)  41.929692 -87.676685
1  POINT(-87.624816 41.753358)  41.753358 -87.624816
2  POINT(-87.614122 41.889821)  41.889821 -87.614122
3  POINT(-87.680139 41.939093)  41.939093 -87.680139
4   POINT(-87.735235 41.91658)  41.916580 -87.735235
```

b.

```
merged_df['latitude_bin'] =  merged_df['latitude'].apply(lambda x: round(x, 2) if not pd.isnull(x) else None)
merged_df['longitude_bin'] = merged_df['longitude'].apply(lambda x: round(x, 2) if not pd.isnull(x) else None)

binned_counts = merged_df.groupby(['latitude_bin', 'longitude_bin']).size().reset_index(name='count')

max_binned_combination = binned_counts.loc[binned_counts['count'].idxmax()]

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

```
Binned Latitude-Longitude Combination with the Most Observations:
latitude_bin        41.88
longitude_bin      -87.65
count            21325.00
Name: 396, dtype: float64
```

c.

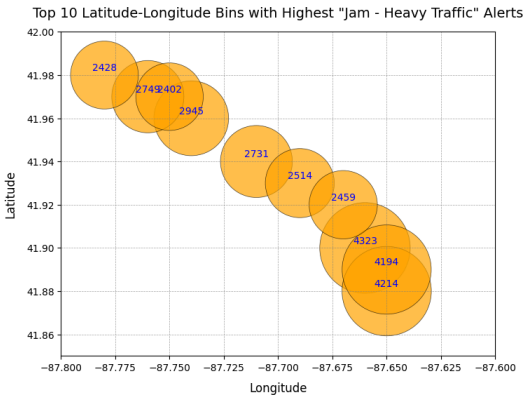
```
import os
import pandas as pd

chosen_type = 'Accident'
chosen_subtype = 'Unclassified'

filtered_df = merged_df[
    (merged_df['updated_type'] == chosen_type) &
    (merged_df['updated_subtype'] == chosen_subtype)
]

filtered_df['lat_long_bin'] = list(zip(filtered_df['latitude_bin'], filtered_df['longitude_bin']))

top_alerts = (
    filtered_df.groupby('lat_long_bin')
    .size()
    .reset_index(name='count')
)
top_10_alerts = top_alerts.nlargest(10, 'count')

os.makedirs('top_alerts_map', exist_ok=True)

top_10_alerts.to_csv('top_alerts_map/top_alerts_map.csv', index=False)

level_of_aggregation = 'latitude-longitude bin'
number_of_rows = top_10_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: 10
          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
<ipython-input-11-f021697ef177>:12: 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-a-copy
  filtered_df['lat_long_bin'] = list(zip(filtered_df['latitude_bin'], filtered_df['longitude_bin']))
```

2.

```
import altair as alt
import matplotlib.pyplot as plt

filtered_df = merged_df[
    (merged_df['updated_type'] == 'Jam') &
    (merged_df['updated_subtype'] == 'Jam Heavy Traffic')
]

filtered_df['lat_long_bin'] = list(zip(filtered_df['latitude_bin'], filtered_df['longitude_bin']))

top_alerts = (
    filtered_df.groupby(['latitude_bin', 'longitude_bin'])
    .size()
    .reset_index(name='count')
)

top_10_alerts = top_alerts.nlargest(10, 'count')

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=[41.8, 42.0])),
    size=alt.Size('count:Q', title='Number of Alerts'),
    tooltip=['latitude_bin', 'longitude_bin', 'count']
).properties(
    title='Top 10 Latitude-Longitude Bins with Highest "Jam - Heavy Traffic" Alerts',
    width=400,
    height=300
)

scatter_plot.save('scatter_plot.html')

import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))
scatter = plt.scatter(
    top_10_alerts['longitude_bin'],
    top_10_alerts['latitude_bin'],
    s=top_10_alerts['count'] * 2,
    alpha=0.7,
    color='orange',
    edgecolor='black',
    linewidth=0.5
)

plt.grid(color='gray', linestyle='--', linewidth=0.5, alpha=0.7)

plt.title('Top 10 Latitude-Longitude Bins with Highest "Jam - Heavy Traffic" Alerts', fontsize=14, pad=15)
plt.xlabel('Longitude', fontsize=12, labelpad=10)
plt.ylabel('Latitude', fontsize=12, labelpad=10)

for index, row in top_10_alerts.iterrows():
    plt.text(
        row['longitude_bin'],
        row['latitude_bin'] + 0.002,
        f"{int(row['count'])}",
        fontsize=10,
        ha='center',
        color='blue'
    )

plt.xlim(-87.8, -87.6)
plt.ylim(41.85, 42.0)
plt.savefig('scatter_plot_smaller_circles.png', dpi=300)
plt.show()
```

```
<ipython-input-12-656d4fc09d45>:9: 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-a-copy
  filtered_df['lat_long_bin'] = list(zip(filtered_df['latitude_bin'], filtered_df['longitude_bin']))
```



3.

a.

```
import requests
geojson_url = "https://data.cityofchicago.org/resource/igwz-8jzy.geojson"

response = requests.get(geojson_url)

if response.status_code == 200:
    with open("chicago_neighborhoods.geojson", "wb") as file:
        file.write(response.content)
    print("GeoJSON file successfully downloaded and saved as 'chicago_neighborhoods.geojson'.")
else:
    print(f"Failed to download the GeoJSON file. Status code: {response.status_code}")
```

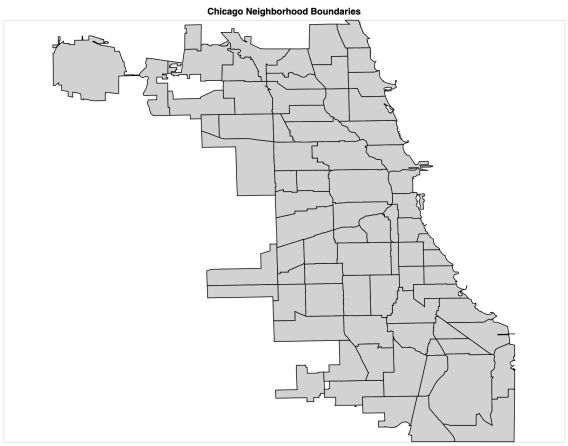
```
GeoJSON file successfully downloaded and saved as 'chicago_neighborhoods.geojson'.
```

b.

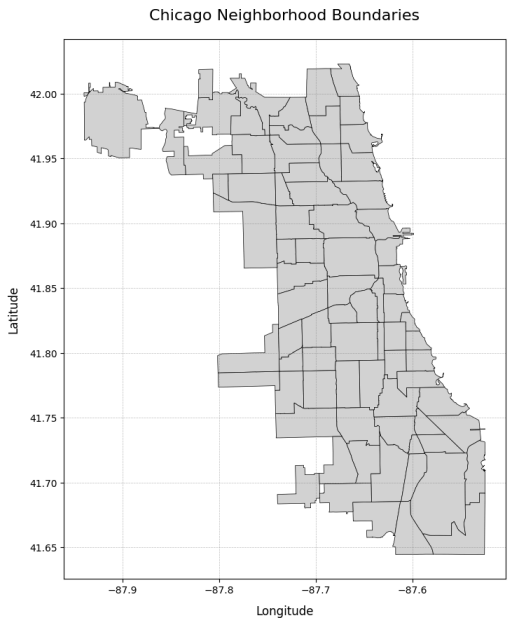
```
import json
import altair as alt
file_path = "/content/chicago_neighborhoods.geojson"

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

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

map_chart = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='black'
).project(
    type='equirectangular'
).properties(
    title="Chicago Neighborhood Boundaries",
    width=800,
    height=600
)
map_chart.save('chicago_neighborhoods_map.html')
print("The map has been saved as 'chicago_neighborhoods_map.html'.")
```

```
The map has been saved as 'chicago_neighborhoods_map.html'.
```

```
from IPython.core.display import display, HTML
html_file_path = "chicago_neighborhoods_map.html"

with open(html_file_path, "r") as file:
    display(HTML(file.read()))
```




```
import geopandas as gpd
import matplotlib.pyplot as plt

# Step 1: Load the GeoJSON file into a GeoDataFrame
gdf = gpd.read_file("chicago_neighborhoods.geojson")  # Ensure the file path is correct

# Step 2: Plot the GeoDataFrame
fig, ax = plt.subplots(figsize=(10, 10))
gdf.plot(ax=ax, color="lightgray", edgecolor="black", linewidth=0.5)

# Step 3: Customize the plot
ax.set_title("Chicago Neighborhood Boundaries", fontsize=16, pad=20)
ax.set_xlabel("Longitude", fontsize=12, labelpad=10)
ax.set_ylabel("Latitude", fontsize=12, labelpad=10)
ax.grid(color='gray', linestyle='--', linewidth=0.5, alpha=0.5)

# Step 4: Save the plot
plt.savefig("chicago_neighborhoods_map.png", dpi=300)
plt.show()
```

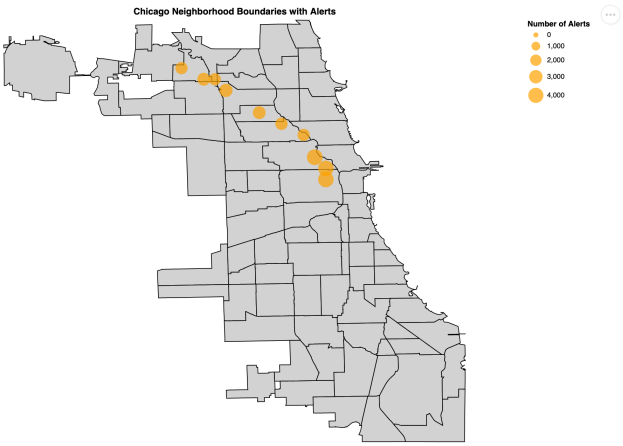


4.

```
import altair as alt

base_map = alt.Chart(geo_data).mark_geoshape(
    fill='lightgray',
    stroke='black'
).project(
    type='equirectangular'
).properties(
    width=800,
    height=600,
    title="Chicago Neighborhood Boundaries with Alerts"
)

scatter_layer = alt.Chart(top_10_alerts).mark_circle(
    color='orange',
    opacity=0.7
).encode(
    longitude='longitude_bin:Q',
    latitude='latitude_bin:Q',
    size=alt.Size('count:Q', scale=alt.Scale(range=[50, 500]), title="Number of Alerts"),
    tooltip=['latitude_bin', 'longitude_bin', 'count']
)

layered_map = (base_map + scatter_layer).configure_axis(
    titleFontSize=14,
    labelFontSize=12,
    grid=True,
    gridColor="gray",
    gridOpacity=0.5
).configure_view(
    stroke=None
).properties(
    width=800,
    height=600
).encode(
    latitude=alt.Latitude('latitude_bin:Q', title='Latitude'),
    longitude=alt.Longitude('longitude_bin:Q', title='Longitude')
)

layered_map.save('layered_chicago_map_with_labels.html')
print("The layered map with latitude and longitude labels has been saved as 'layered_chicago_map_with_labels.html'.")
```

```
The layered map with latitude and longitude labels has been saved as 'layered_chicago_map_with_labels.html'.
```

```
from IPython.core.display import display, HTML

html_file_path = "layered_chicago_map_with_labels.html"

with open(html_file_path, "r") as file:
    display(HTML(file.read()))
```



5.

a.

```
!pip install dash
!pip install dash-bootstrap-components
```

```
Collecting dash
  Downloading dash-2.18.2-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from dash) (3.0.3)
Collecting Werkzeug<3.1 (from dash)
  Downloading werkzeug-3.0.6-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from dash) (5.24.1)
Collecting dash-html-components==2.0.0 (from dash)
  Downloading dash_html_components-2.0.0-py3-none-any.whl.metadata (3.8 kB)
Collecting dash-core-components==2.0.0 (from dash)
  Downloading dash_core_components-2.0.0-py3-none-any.whl.metadata (2.9 kB)
Collecting dash-table==5.0.0 (from dash)
  Downloading dash_table-5.0.0-py3-none-any.whl.metadata (2.4 kB)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-packages (from dash) (8.5.0)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/dist-packages (from dash) (4.12.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from dash) (2.32.3)
Collecting retrying (from dash)
  Downloading retrying-1.3.4-py3-none-any.whl.metadata (6.9 kB)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages (from dash) (1.6.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from dash) (75.1.0)
Requirement already satisfied: Jinja2>=3.1.2 in /usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4->dash) (3.1.4)
Requirement already satisfied: itsdangerous>=2.1.2 in /usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4->dash) (2.2.0)
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4->dash) (8.1.7)
Requirement already satisfied: blinker>=1.6.2 in /usr/local/lib/python3.10/dist-packages (from Flask<3.1,>=1.0.4->dash) (1.9.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly>=5.0.0->dash) (9.0.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly>=5.0.0->dash) (24.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from Werkzeug<3.1->dash) (3.0.2)
Requirement already satisfied: zipp>=3.20 in /usr/local/lib/python3.10/dist-packages (from importlib-metadata->dash) (3.21.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->dash) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->dash) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->dash) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->dash) (2024.8.30)
Requirement already satisfied: six>=1.7.0 in /usr/local/lib/python3.10/dist-packages (from retrying->dash) (1.16.0)
Downloading dash-2.18.2-py3-none-any.whl (7.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 83.6 MB/s eta 0:00:00
Downloading dash_core_components-2.0.0-py3-none-any.whl (3.8 kB)
Downloading dash_html_components-2.0.0-py3-none-any.whl (4.1 kB)
Downloading dash_table-5.0.0-py3-none-any.whl (3.9 kB)
Downloading werkzeug-3.0.6-py3-none-any.whl (227 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 228.0/228.0 kB 16.3 MB/s eta 0:00:00
Downloading retrying-1.3.4-py3-none-any.whl (11 kB)
Installing collected packages: dash-table, dash-html-components, dash-core-components, Werkzeug, retrying, dash
  Attempting uninstall: Werkzeug
    Found existing installation: Werkzeug 3.1.3
    Uninstalling Werkzeug-3.1.3:
      Successfully uninstalled Werkzeug-3.1.3
Successfully installed Werkzeug-3.0.6 dash-2.18.2 dash-core-components-2.0.0 dash-html-components-2.0.0 dash-table-5.0.0 retrying-1.3.4
Collecting dash-bootstrap-components
  Downloading dash_bootstrap_components-1.6.0-py3-none-any.whl.metadata (5.2 kB)
Requirement already satisfied: dash>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from dash-bootstrap-components) (2.18.2)
Requirement already satisfied: Flask<3.1,>=1.0.4 in /usr/local/lib/python3.10/dist-packages (from dash>=2.0.0->dash-bootstrap-components) (3.0.3)
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.10/dist-packages (from dash>=2.0.0->dash-bootstrap-components) (3.0.6)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.10/dist-packages (from dash>=2.0.0->dash-bootstrap-components) (5.24.1)
Requirement already satisfied: dash-html-components==2.0.0 in /usr/local/lib/python3.10/dist-packages (from dash>=2.0.0->dash-bootstrap-components) (2.0.0)
Requirement already satisfied: dash-core-components==2.0.0 in /usr/local/lib/python3.10/dist-packages (from dash>=2.0.0->dash-bootstrap-components) (2.0.0)
Requirement already satisfied: dash-table==5.0.0 in /usr/local/lib/python3.10/dist-packages (from dash>=2.0.0->dash-bootstrap-components) (5.0.0)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.10/dist-packages (from dash>=2.0.0->dash-bootstrap-components) (8.5.0)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.10/dist-packages (from dash>=2.0.0->dash-bootstrap-components) (4.12.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from dash>=2.0.0->dash-bootstrap-components) (2.32.3)
```

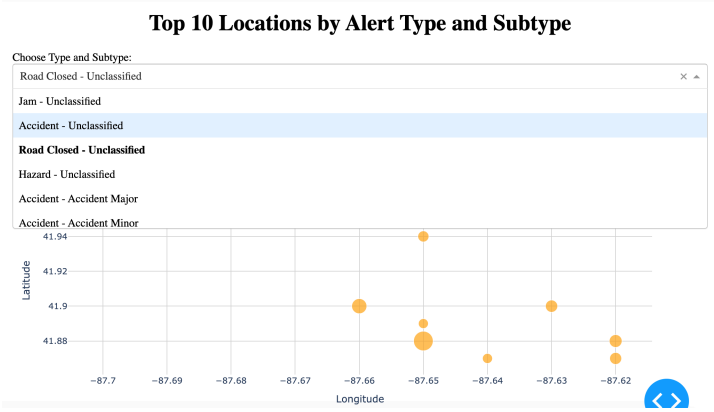
```
import pandas as pd
import dash
from dash import dcc, html, Input, Output
import plotly.express as px

data = merged_df
data["type_subtype"] = data["updated_type"] + " - " + data["updated_subtype"]
unique_combinations = data["type_subtype"].unique()

app = dash.Dash(__name__)

app.layout = html.Div([
    html.H1("Top 10 Locations by Alert Type and Subtype", style={"textAlign": "center"}),

    html.Label("Choose Type and Subtype:"),
    dcc.Dropdown(
        id="type_subtype_dropdown",
        options=[{"label": combo, "value": combo} for combo in unique_combinations],
        value=unique_combinations[0]
    ),

    html.Div(id="selected_combination", style={"marginTop": "20px", "fontSize": "18px"}),

    dcc.Graph(id="top10_plot")
])

@app.callback(
    [Output("selected_combination", "children"),
     Output("top10_plot", "figure")],
    [Input("type_subtype_dropdown", "value")]
)
def update_dashboard(selected_combination):
    filtered_data = data[data["type_subtype"] == selected_combination]
    top10_data = (
        filtered_data.groupby(["latitude_bin", "longitude_bin"])
        .size()
        .reset_index(name="count")
        .sort_values(by="count", ascending=False)
        .head(10)
    )

    fig = px.scatter(
        top10_data,
        x="longitude_bin",
        y="latitude_bin",
        size="count",
        color_discrete_sequence=["orange"],
        labels={"longitude_bin": "Longitude", "latitude_bin": "Latitude", "count": "Alert Count"},
        title=f"Top 10 Locations for {selected_combination}",
        template="plotly_white"
    )
    fig.update_traces(marker=dict(opacity=0.7))
    fig.update_layout(
        title={"x": 0.5},  # Center the title
        xaxis=dict(gridcolor="lightgray"),
        yaxis=dict(gridcolor="lightgray")
    )

    return f"You selected: {selected_combination}", fig
```

```
# Run the app
if __name__ == "__main__":
    app.run_server(debug=True)
```

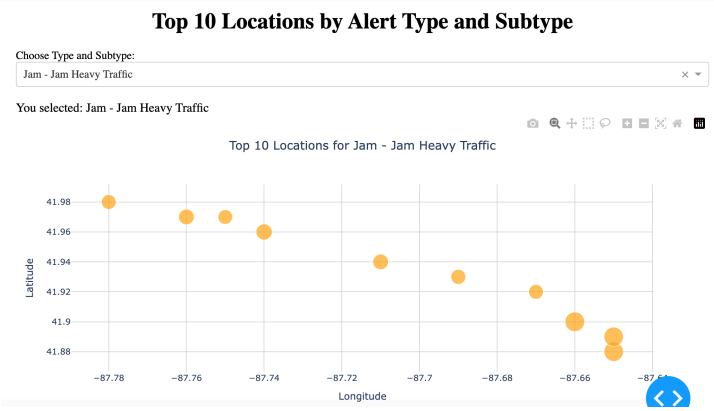
Show hidden output

```
len(unique_combinations)
```

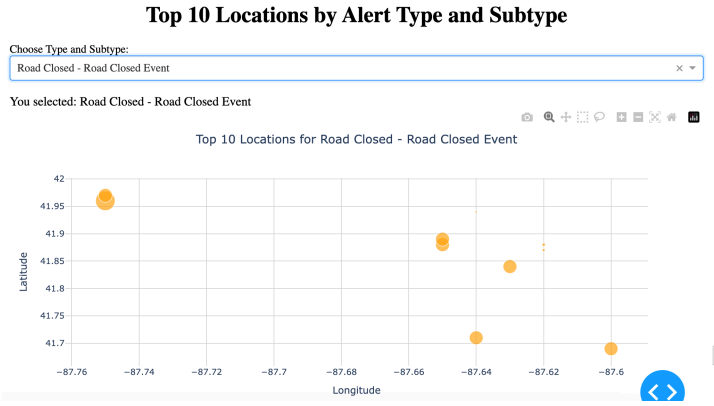
32



b.



c.



Based on the map displayed, the alerts for road closures due to events are most common at:

Latitude: ~41.95 and Longitude: ~-87.76. This location has the largest circle, indicating the highest number of alerts for road closures due to events.

d.

The question is "Where are the top 10 locations with the highest number of 'Hazard - Hazard On Road Ice' alerts in Chicago?"

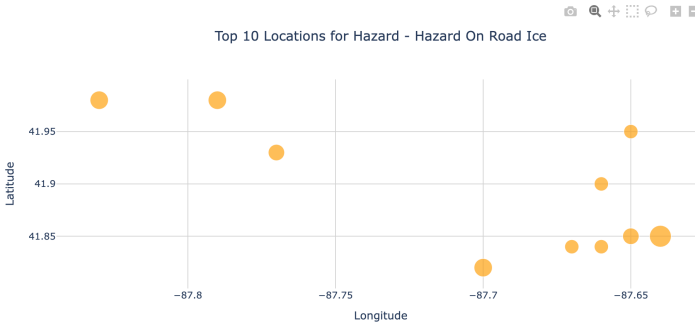
The answer is the alerts for road ice hazards are most commonly observed in areas around Latitude ~41.9 and Longitude ~-87.7. Other hotspots include areas slightly west and south of this region. This map can help city planners prioritize areas for road salt distribution and hazard warnings during icy conditions.

Top 10 Locations by Alert Type and Subtype

Choose Type and Subtype:

Hazard - Hazard On Road Ice

You selected: Hazard - Hazard On Road Ice



e. To enhance the analysis, I suggest adding a "Time of Day" column or filter to the dashboard. This column can categorize alerts by time ranges (e.g., morning, afternoon, evening, night) or specific time stamps. With this, we could analyze whether certain alerts (e.g., traffic jams or hazards) are more common during specific times of the day and helps authorities plan interventions (e.g., deploying traffic patrols or maintenance teams) based on peak times.

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 directly by the ts column unless the analysis specifically requires granular time-based patterns. Since ts likely records unique timestamps for each alert, collapsing by this column might result in too many unique values, making the dataset very large and difficult to manage and analyze. Also,for finding top 10 locations for a given alert type, collapsing by ts would require further aggregation later, increasing computational complexity.

b.

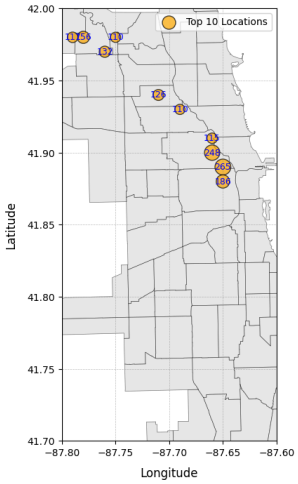
```
import os
import pandas as pd

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

os.makedirs('top_alerts_map_byhour', exist_ok=True)

collapsed_df = (
    data.groupby(['hour', 'updated_type', 'updated_subtype', 'latitude_bin', 'longitude_bin'])
    .size()
    .reset_index(name='count')
)

collapsed_df.to_csv('top_alerts_map_byhour.csv', index=False)

rows_count = collapsed_df.shape[0]
print(f"The dataset has been saved to top_alerts_map_byhour.csv and contains {rows_count} rows.")
```

```
The dataset has been saved to top_alerts_map_byhour.csv and contains 87920 rows.
```

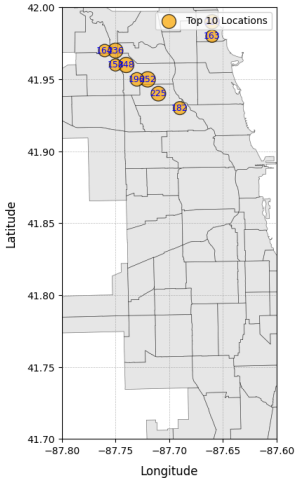
c.

```
import matplotlib.pyplot as plt

filtered_df = collapsed_df[
    (collapsed_df['updated_type'] == 'Jam') &
    (collapsed_df['updated_subtype'] == 'Jam Heavy Traffic')
]
times_to_plot = ['00:00', '12:00', '18:00']

longitude_range = (-87.8, -87.6)
latitude_range = (41.7, 42.0)

for time in times_to_plot:
    hourly_data = filtered_df[filtered_df['hour'] == time].nlargest(10, 'count')
    fig, ax = plt.subplots(figsize=(10, 8))
    gdf.plot(ax=ax, color='lightgray', edgecolor='black', linewidth=0.5, alpha=0.5)
    scatter = ax.scatter(
        hourly_data['longitude_bin'],
        hourly_data['latitude_bin'],
        s=hourly_data['count'],
        color='orange',
        alpha=0.7,
        edgecolor='black',
        label='Top 10 Locations'
    )
    for _, row in hourly_data.iterrows():
        ax.text(row['longitude_bin'], row['latitude_bin'], str(int(row['count'])), fontsize=9, ha='center', va='center', color='blue')

    ax.set_title(f"Top 10 Locations for 'Jam - Heavy Traffic' at {time}", fontsize=16, pad=20)
    ax.set_xlim(longitude_range)
    ax.set_ylim(latitude_range)
    ax.set_xlabel("Longitude", fontsize=12, labelpad=10)
    ax.set_ylabel("Latitude", fontsize=12, labelpad=10)
    ax.grid(color='gray', linestyle='--', linewidth=0.5, alpha=0.5)
    ax.legend(loc='upper right')

    plt.savefig(f"top_10_locations_{time.replace(':', '')}.png", dpi=300)
    plt.show()
```

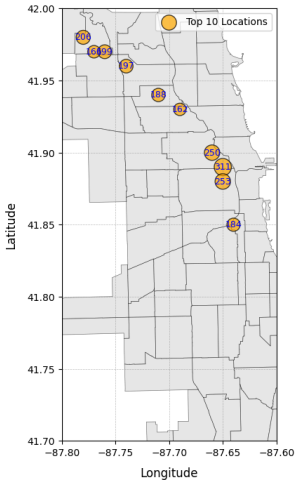
Top 10 Locations for 'Jam - Heavy Traffic' at 00:00



Top 10 Locations for 'Jam - Heavy Traffic' at 12:00



Top 10 Locations for 'Jam - Heavy Traffic' at 18:00



2a.

```
from dash import Dash, dcc, html

app = Dash(__name__)

dropdown_options = [
    {'label': f"{row['updated_type']} - {row['updated_subtype']}", 'value': f"{row['updated_type']} - {row['updated_subtype']}"}
    for _, row in collapsed_df[['updated_type', 'updated_subtype']].drop_duplicates().iterrows()
]

app.layout = html.Div([
    html.H1("Top 10 Alert Locations by Type, Subtype, and Hour", style={'textAlign': 'center'}),

    html.Div([
        html.Label("Choose Type and Subtype:"),
        dcc.Dropdown(
            id='type-subtype-dropdown',
            options=dropdown_options,
            placeholder="Select Type and Subtype",
            style={'width': '50%'}
        )
    ], style={'padding': '20px'}),

    html.Div([
        html.Label("Select Hour:"),
        dcc.Slider(
            id='hour-slider',
            min=0,
            max=23,
            step=1,
            value=12,
            marks={i: f"{i}:00" for i in range(0, 24)},
            tooltip={"placement": "bottom", "always_visible": True}
        )
    ], style={'padding': '20px'})
])

if __name__ == '__main__':
```

```
    app.run_server(debug=True)
```

Show hidden output

Top 10 Alert Locations by Type, Subtype, and Hour

Choose Type and Subtype:

Hazard - Hazard On Road

Select Hour:

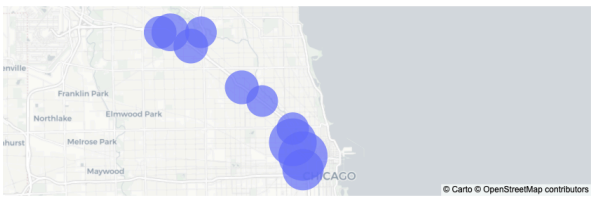
0:00 1:00 2:00 3:00 4:00 5:00 6:00 7 8:00 9:00 10:00 11:00 12:00 13:00 14:00 15:00 16:00 17:00 18:00 19:00 20:00 21:00 22:00 23:00

b.

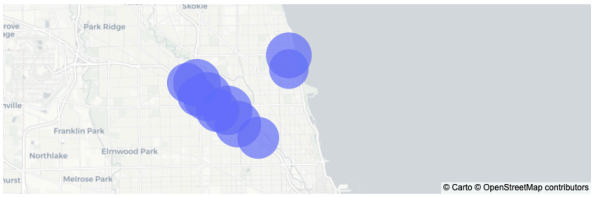
```
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt
from dash import Dash, dcc, html, Input, Output
import plotly.express as px

app = Dash(__name__)

collapsed_df = pd.read_csv('top_alerts_map_byhour.csv')
geojson_path = "/content/chicago_neighborhoods.geojson"
gdf = gpd.read_file(geojson_path)

dropdown_options = [
    {'label': f"{row['updated_type']} - {row['updated_subtype']}", 'value': f"{row['updated_type']} - {row['updated_subtype']}"}
    for _, row in collapsed_df[['updated_type', 'updated_subtype']].drop_duplicates().iterrows()
]

app.layout = html.Div([
    html.H1("Dynamic Plot: Top 10 Locations by Type, Subtype, and Hour", style={'textAlign': 'center'}),

    html.Div([
        html.Label("Choose Type and Subtype:"),
        dcc.Dropdown(
            id='type-subtype-dropdown',
            options=dropdown_options,
            placeholder="Select Type and Subtype",
            style={'width': '50%'}
        )
    ], style={'padding': '20px'}),

    html.Div([
        html.Label("Select Hour:"),
        dcc.Slider(
            id='hour-slider',
            min=0,
            max=23,
            step=1,
            value=12,  # Default hour
            marks={i: f"{i}:00" for i in range(0, 24)},  # Hour labels
            tooltip={"placement": "bottom", "always_visible": True}
        )
    ], style={'padding': '20px'}),

    html.Div([
        dcc.Graph(id='dynamic-map-plot')
    ])
])

@app.callback(
    Output('dynamic-map-plot', 'figure'),
    [Input('type-subtype-dropdown', 'value'),
     Input('hour-slider', 'value')]
)
def update_plot(selected_type_subtype, selected_hour):
    if not selected_type_subtype:
        return px.scatter(title="Please select a Type and Subtype")

    selected_type, selected_subtype = selected_type_subtype.split(" - ")

    filtered_data = collapsed_df[
        (collapsed_df['updated_type'] == selected_type) &
        (collapsed_df['updated_subtype'] == selected_subtype) &
        (collapsed_df['hour'] == f"{selected_hour:02}:00")
    ].nlargest(10, 'count')

    fig = px.scatter_mapbox(
        filtered_data,
        lat='latitude_bin',
        lon='longitude_bin',
        size='count',
        hover_name='count',
        size_max=50,
        title=f"Top 10 Locations for '{selected_type} - {selected_subtype}' at {selected_hour:02}:00",
        mapbox_style="carto-positron",
        zoom=10,
        center={"lat": 41.85, "lon": -87.65}
    )

    return fig

if __name__ == '__main__':
    app.run_server(debug=True)
```

Show hidden output

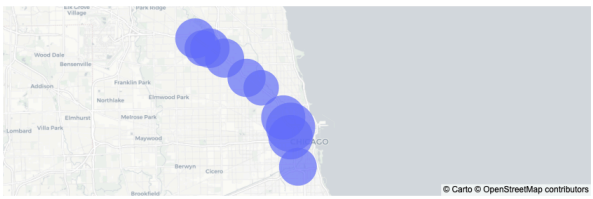
Top 10 Locations for 'Jam - Jam Heavy Traffic' at 00:00



Top 10 Locations for 'Jam - Jam Heavy Traffic' at 12:00

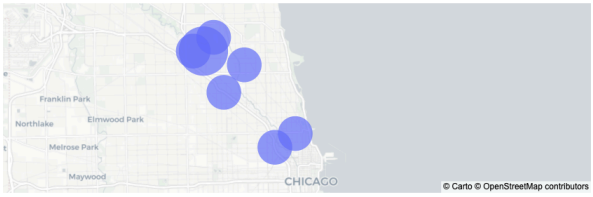


Top 10 Locations for 'Jam - Jam Heavy Traffic' at 18:00

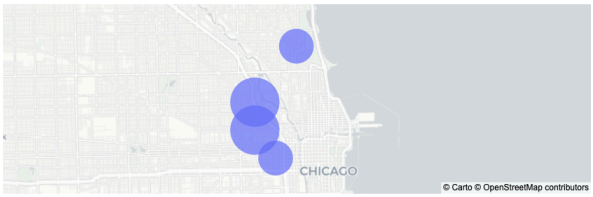


c. Road construction is done more during night hours than morning hours.

Top 10 Locations for 'Road Closed - Road Closed Construction' at 12:00



Top 10 Locations for 'Road Closed - Road Closed Construction' at 18:00



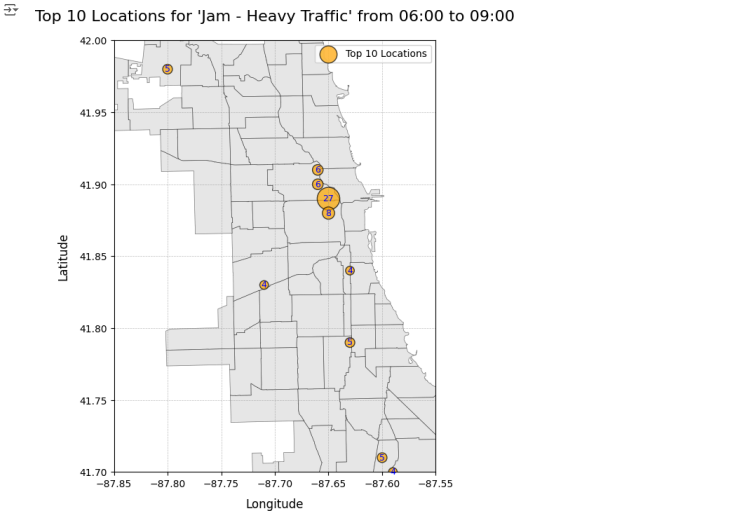
App #3: Top Location by Alert Type and Hour Dashboard (20 points)

1.a. Collapsing the dataset by a range of hours could be a good idea because it simplifies handling the data in the app. Pre-collapsing the data ensures that the app doesn't have to dynamically calculate aggregates for every range of hours. This reduces computational overhead, making the app faster and more responsive. Instead of storing data for each individual hour, collapsing by ranges (e.g., 6AM–10AM) reduces the number of rows in the dataset, optimizing storage and app efficiency.

b.

```
import matplotlib.pyplot as plt
import pandas as pd

time_range_start = "06:00"
time_range_end = "09:00"

collapsed_df['hour'] = pd.to_datetime(collapsed_df['hour'], format='%H:%M')
time_range_data = collapsed_df[
    (collapsed_df['updated_type'] == 'Jam') &
    (collapsed_df['updated_subtype'] == 'Jam Heavy Traffic') &
    (collapsed_df['hour'] >= pd.to_datetime(time_range_start, format='%H:%M')) &
    (collapsed_df['hour'] < pd.to_datetime(time_range_end, format='%H:%M'))
]

top_10_locations = time_range_data.groupby(['latitude_bin', 'longitude_bin'], as_index=False)['count'].sum()
top_10_locations = top_10_locations.nlargest(10, 'count')

longitude_range = (-87.85, -87.55)
latitude_range = (41.7, 42.0)

fig, ax = plt.subplots(figsize=(10, 8))

gdf.plot(ax=ax, color='lightgray', edgecolor='black', linewidth=0.5, alpha=0.5)

scatter = ax.scatter(
    top_10_locations['longitude_bin'],
    top_10_locations['latitude_bin'],
    s=top_10_locations['count'] * 20,
    color='orange',
    alpha=0.7,
    edgecolor='black',
    label='Top 10 Locations'
)

for _, row in top_10_locations.iterrows():
    ax.text(row['longitude_bin'], row['latitude_bin'], str(int(row['count'])), fontsize=9, ha='center', va='center', col

# Customize the plot
ax.set_title(f"Top 10 Locations for 'Jam - Heavy Traffic' from {time_range_start} to {time_range_end}", fontsize=16, pac
```

```
ax.set_xlim(longitude_range)
ax.set_ylim(latitude_range)
ax.set_xlabel("Longitude", fontsize=12, labelpad=10)
ax.set_ylabel("Latitude", fontsize=12, labelpad=10)
ax.grid(color='gray', linestyle='--', linewidth=0.5, alpha=0.5)
ax.legend(loc='upper right')

# Save the plot
plt.savefig(f"top_10_locations_{time_range_start.replace(':', '')}_{time_range_end.replace(':', '')}.png", dpi=300)
plt.show()
```



2.a.

```
import pandas as pd
import geopandas as gpd
from dash import Dash, dcc, html, Input, Output
import plotly.express as px

app = Dash(__name__)
collapsed_df = pd.read_csv('top_alerts_map_byhour.csv')
geojson_path = "/content/chicago_neighborhoods.geojson"
gdf = gpd.read_file(geojson_path)

collapsed_df['hour'] = pd.to_datetime(collapsed_df['hour'], format='%H:%M').dt.strftime('%H:%M')

dropdown_options = [
    {'label': f"{row['updated_type']} - {row['updated_subtype']}", 'value': f"{row['updated_type']} - {row['updated_subt
    for _, row in collapsed_df[['updated_type', 'updated_subtype']].drop_duplicates().iterrows()
]

app.layout = html.Div([
    html.H1("Dynamic Plot: Top 10 Locations by Type, Subtype, and Hour Range", style={'textAlign': 'center'}),

    html.Div([
        html.Label("Choose Type and Subtype:"),
        dcc.Dropdown(
            id='type-subtype-dropdown',
            options=dropdown_options,
            placeholder="Select Type and Subtype",
            style={'width': '50%'}
        )
    ], style={'padding': '20px'}),

    html.Div([
        html.Label("Select Hour Range:"),
        dcc.RangeSlider(
            id='hour-range-slider',
            min=0,
            max=23,
            step=1,
            value=[6, 9],  # Default range
            marks={i: f"{i}:00" for i in range(0, 24)},  # Hour labels
            tooltip={"placement": "bottom", "always_visible": True}
        )
    ], style={'padding': '20px'}),

    html.Div([
        dcc.Graph(id='dynamic-map-plot')
    ])
])

# Callback to update the plot
@app.callback(
    Output('dynamic-map-plot', 'figure'),
    [Input('type-subtype-dropdown', 'value'),
     Input('hour-range-slider', 'value')]
)
def update_plot(selected_type_subtype, hour_range):
    # Debugging: Print the inputs
    print(f"Selected type-subtype: {selected_type_subtype}")
    print(f"Selected hour range: {hour_range}")

    if not selected_type_subtype:
        return px.scatter(title="Please select a Type and Subtype")

    selected_type, selected_subtype = selected_type_subtype.split(" - ")
    start_hour, end_hour = hour_range

    # Filter data based on type, subtype, and hour range
    filtered_data = collapsed_df[
        (collapsed_df['updated_type'] == selected_type) &
        (collapsed_df['updated_subtype'] == selected_subtype) &
        (collapsed_df['hour'].apply(lambda x: int(x.split(':')[0]) >= start_hour)) &
        (collapsed_df['hour'].apply(lambda x: int(x.split(':')[0]) <= end_hour))
    ]

    # Debugging: Print filtered data
    print(f"Filtered Data:\n{filtered_data}")

    if filtered_data.empty:
        return px.scatter(title=f"No data available for {selected_type_subtype} from {start_hour}:00 to {end_hour}:00")

    # Group data and get top 10 locations
    top_10 = filtered_data.groupby(['latitude_bin', 'longitude_bin'], as_index=False)['count'].sum().nlargest(10, 'count

    # Debugging: Print top 10 data
    print(f"Top 10 Locations:\n{top_10}")

    # Create the map plot
    fig = px.scatter_mapbox(
        top_10,
        lat='latitude_bin',
        lon='longitude_bin',
        size='count',
        hover_name='count',
        size_max=50,
        title=f"Top 10 Locations for '{selected_type} - {selected_subtype}' from {start_hour}:00 to {end_hour}:00",
        mapbox_style="carto-positron",
        zoom=10,
        center={"lat": 41.85, "lon": -87.65}
    )

    return fig

# Run the app
if __name__ == '__main__':
    app.run_server(debug=True)
```

Show hidden output

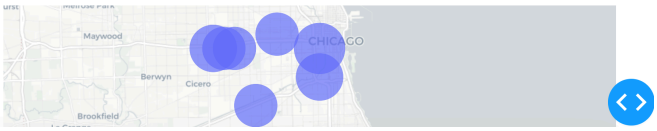
Dynamic Plot: Top 10 Locations by Type, Subtype, and Hour/Range

Choose Type and Subtype:
Accident - Accident Major

Toggle to select Single Hour or Hour Range:
Enable Range of Hours

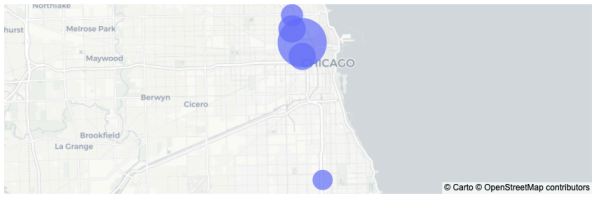
Select Hour:
0:00 1:00 2:00 3:00 4:00 5:00 6:00 7:00 8:00 9:00 10:00 11:00 12 13:00 14:00 15:00 16:00 17:00 18:00 19:00 20:00 21:00 22:00 23:00

Top 10 Locations for 'Accident - Accident Major' at 12:00



b.

Top 10 Locations for 'Jam - Jam Heavy Traffic' from 6:00 to 9:00



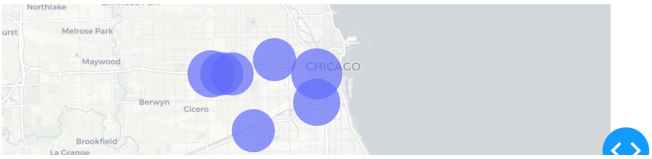
3.a.

Choose Type and Subtype:
Accident - Accident Major

Toggle to select Single Hour or Hour Range:
Enable Range of Hours

Select Hour:
0:00 1:00 2:00 3:00 4:00 5:00 6:00 7:00 8:00 9:00 10:00 11:00 12 13:00 14:00 15:00 16:00 17:00 18:00 19:00 20:00 21:00 22:00 23:00

Top 10 Locations for 'Accident - Accident Major' at 12:00



The possible values for input.switch_button if the switch button is named switch_button could be [] (an empty list) and ['range']. [] This value indicates that the switch button is in the default (off) state, meaning the user has not selected the option to toggle to a range of hours. In this case, the single-hour slider (hour-slider) should be displayed.['range'] This value indicates that the switch button is in the enabled (on) state, meaning the user has selected the option to toggle to a range of hours.

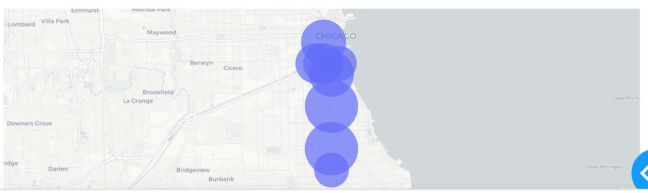
c.

Choose Type and Subtype:
Accident - Accident Major

Toggle to select Single Hour or Hour Range:
Enable Range of Hours

Select Hour:
0:00 1:00 2:00 3:00 4:00 5:00 6:00 7:00 8 9:00 10:00 11:00 12:00 13:00 14:00 15:00 16:00 17:00 18:00 19:00 20:00 21:00 22:00 23:00

Top 10 Locations for 'Accident - Accident Major' at 08:00



Choose Type and Subtype:
Accident - Accident Major

Toggle to select Single Hour or Hour Range:
Enable Range of Hours

Select Hour Range:
0:00 1:00 2:00 3:00 4:00 5:00 6 7:00 8:00 9 10:00 11:00 12:00 13:00 14:00 15:00 16:00 17:00 18:00 19:00 20:00 21:00 22:00 23:00

Top 10 Locations for 'Accident - Accident Major' from 6:00 to 9:00