Problem Set 6 - Waze Shiny Dashboard

AUTHOR
Prashanthi Subbiah

PUBLISHED
November 24, 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: PS
2. "I have uploaded the names of anyone I worked with on the problem set **here**" PS (2 point)
3. Late coins used this pset: \0\1_\0\2 Late coins left after submission: \0\1 _ \0\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. Push your `ps6.qmd`, `ps6.pdf`, `requirements.txt`, and all created folders (we will create three Shiny apps so you will have at least three additional folders) to your Github repo (5 points). It is fine to use Github Desktop.
7. Submit `ps6.pdf` and also link your Github repo via Gradescope (5 points)
8. Tag your submission in Gradescope. For the Code Style part (10 points) please tag the whole correspondingsection for the code style rubric.

Notes: see the Quarto documentation (link) for directions on inserting images into your knitted document.

IMPORTANT: For the App portion of the PS, in case you can not arrive to the expected functional dashboard we will need to take a look at your `app.py` file. You can use the following code chunk template to "import" and print the content of that file. Please, don't forget to also tag the corresponding code chunk as part of your submission!

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

```
    except Exception as e:
        print("```python")
        print(f"Error reading file: {e}")
        print("```")

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

Background

Data Download and Exploration (20 points)

1.

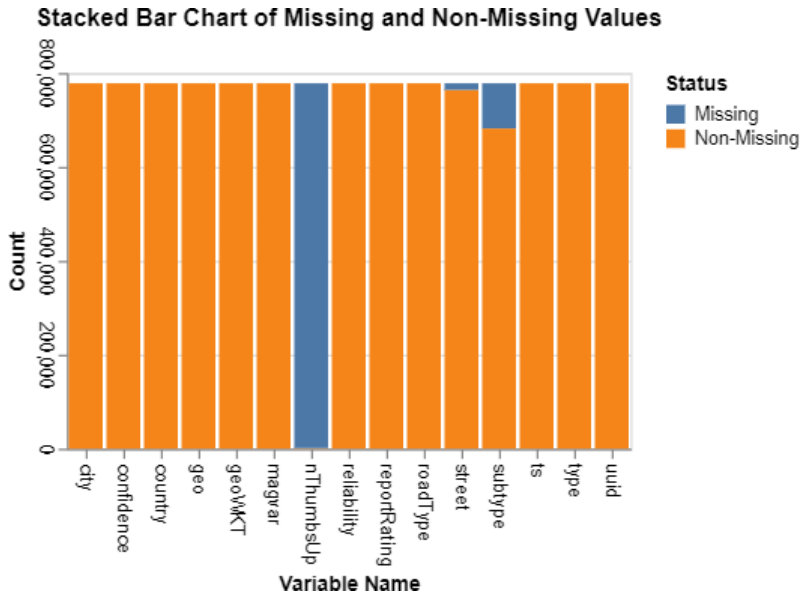
```
# Importing and Loading Packages
import zipfile
import pandas as pd

# Loading Data
waze_data_sample = pd.read_csv("C:/Users/prash/Downloads/waze_data/waze_data_sample.csv")

# Function to determine Altair data types
def get_altair_types(df):
    altair_types = {}
    for column in df.columns:
        if pd.api.types.is_numeric_dtype(df[column]):
            altair_types[column] = 'Quantitative' # Quantitative
        elif pd.api.types.is_categorical_dtype(df[column]) or df[column].dtype == 'object':
            altair_types[column] = 'Nominal' # Nominal
        elif pd.api.types.is_datetime64_any_dtype(df[column]):
            altair_types[column] = 'Temporal' # Temporal
        elif pd.api.types.is_integer_dtype(df[column]):
            altair_types[column] = 'Quantitative'  # Quantitative (integers)
        else:
            altair_types[column] = 'Ordinal'  # Ordinal or fallback

    return altair_types

# Get Altair data types for the datafame
altair_data_types = get_altair_types(waze_data_sample)

# Creating dataframe for variable and data type
waze_data_types = pd.DataFrame(list(altair_data_types.items()), columns=['Variable Name',
         'Altair Data Types'])

# Printing dataframe
print(waze_data_types)

# BingChat Citation: queryed function to trouble-shoot for errors
```

```
   Variable Name Altair Data Types
0     Unnamed: 0      Quantitative
```

```
1            city           Nominal
2      confidence      Quantitative
3       nThumbsUp      Quantitative
4          street           Nominal
5            uuid           Nominal
6         country           Nominal
7            type           Nominal
8         subtype           Nominal
9        roadType      Quantitative
10    reliability      Quantitative
11         magvar      Quantitative
12   reportRating      Quantitative
13             ts           Nominal
14            geo           Nominal
15         geoWKT           Nominal
```

2.

```
# Importing and Loading Packages
import zipfile
import pandas as pd
import matplotlib.pyplot as plt
import altair as alt

# Loading Data
waze_full = pd.read_csv("C:/Users/prash/Downloads/waze_data/waze_data.csv")

waze_data_counts = pd.DataFrame({'Variable Name': waze_full.columns, 'Sum of Entries':
         len(waze_full), 'Missing': waze_full.isna().sum()})
waze_data_counts['Non-Missing'] = waze_data_counts['Sum of Entries'] -
         waze_data_counts['Missing']
final_waze_data_counts = waze_data_counts[['Variable Name', 'Non-Missing', 'Missing']]
final_waze_data_counts

# Melt the dataframe to long format
melted_df = final_waze_data_counts.melt(id_vars=['Variable Name'], value_vars=['Non-
         Missing', 'Missing'], var_name='Status', value_name='Count')

# Plotting Stacked Bar Chart in Altair
chart = alt.Chart(melted_df).mark_bar().encode(
    x='Variable Name:N',
    y='Count:Q',
    color='Status:N'
).properties(
    title='Stacked Bar Chart of Missing and Non-Missing Values',
    width=300,
    height=200
).configure_axis(
    labelAngle=90
)

# Show plot
chart.display()
```



The variables with NULL values are “subtype”, “street”, and “nThumbsUp”. The variable with the highest share of NULL values is “nThumbsUp”.

3.

a.

```
# Importing and Loading Packages
import zipfile
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Loading Data
waze_data = pd.read_csv("C:/Users/prash/Downloads/waze_data/waze_data.csv")

# Getting the unique types and subtypes
unique_type = waze_data['type'].unique()
unique_subtype = waze_data['subtype'].unique()
print("Unique types are:", unique_type)
print("Unique subtypes are:", unique_subtype)
```

```
Unique types are: ['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']
Unique subtypes are: [nan 'ACCIDENT_MAJOR' 'ACCIDENT_MINOR' 'HAZARD_ON_ROAD'
 'HAZARD_ON_ROAD_CAR_STOPPED' 'HAZARD_ON_ROAD_CONSTRUCTION'
 'HAZARD_ON_ROAD_EMERGENCY_VEHICLE' 'HAZARD_ON_ROAD_ICE'
 'HAZARD_ON_ROAD_OBJECT' 'HAZARD_ON_ROAD_POT_HOLE'
 'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT' 'HAZARD_ON_SHOULDER'
 'HAZARD_ON_SHOULDER_CAR_STOPPED' 'HAZARD_WEATHER' 'HAZARD_WEATHER_FLOOD'
 'JAM_HEAVY_TRAFFIC' 'JAM_MODERATE_TRAFFIC' 'JAM_STAND_STILL_TRAFFIC'
 'ROAD_CLOSED_EVENT' 'HAZARD_ON_ROAD_LANE_CLOSED' 'HAZARD_WEATHER_FOG'
 'ROAD_CLOSED_CONSTRUCTION' 'HAZARD_ON_ROAD_ROAD_KILL'
 'HAZARD_ON_SHOULDER_ANIMALS' 'HAZARD_ON_SHOULDER_MISSING_SIGN'
 'JAM_LIGHT_TRAFFIC' 'HAZARD_WEATHER_HEAVY_SNOW' 'ROAD_CLOSED_HAZARD'
 'HAZARD_WEATHER_HAIL']
```

All 4 types have a subtype called NA. Hazards has sub-types with enough information to consider sub-

subtypes. An example is HAZARD_ON_ROAD_ICE. Type would be “HAZARD”, subtype, “ON_ROAD”, and sub-subtype, “ICE”.

b.

```
# Summarizing type and subtypes
waze_data_type_sum = waze_data.groupby('type')['subtype'].unique()
print("Table summarizing type and subtype is:", waze_data_type_sum)

# Creating bulleted list using a loop
for main_type, subtypes in waze_data_type_sum.items():
    print(main_type)
    for subtype in subtypes:
        print(f" - {subtype}")

# BingChat Citation: How do I generate a bulleted list in Python?
```

```
Table summarizing type and subtype is: type
  ACCIDENT                    [nan,                ACCIDENT_MAJOR,
  ACCIDENT_MINOR]    HAZARD               [nan,    HAZARD_ON_ROAD,
  HAZARD_ON_ROAD_CAR_STOPP...   JAM   [nan,    JAM_HEAVY_TRAFFIC,
  JAM_MODERATE_TRAFFIC,...     ROAD_CLOSED                  [nan,
  ROAD_CLOSED_EVENT, ROAD_CLOSED_CONSTRUCT...
  Name: subtype, dtype: object
  ACCIDENT
    - nan
    - ACCIDENT_MAJOR
    - ACCIDENT_MIN
  OR HAZARD
    - nan
    - 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_HA
  IL JAM
    - nan
    - JAM_HEAVY_TRAFFIC
    - JAM_MODERATE_TRAFFIC
    - JAM_STAND_STILL_TRAFFIC
```

c.

```
# Summarizing count of NAs and total counts for each type of alert
na_counts_per_type = waze_data.groupby('type').apply(lambda x:
         x.isna().sum().sum()).reset_index()
na_counts_per_type.rename(columns={0: 'NA_count'}, inplace=True)
waze_data['total_cases_per_type'] = waze_data.groupby('type')['type'].transform('count')
na_counts_per_type['total_cases_per_type'] = waze_data['total_cases_per_type']

# Printing table
print(na_counts_per_type)
```

```
# Replacing NA with "Unclassified"
waze_data['subtype'] = waze_data['subtype'].fillna('Unclassified')

# BingChat Citation: Used BingChat to troubleshoot the formation of the table - had used
         .agg instead of transform in original code
```

	type	NA_count	total_cases_per_type
0	ACCIDENT	58522	372485
1	HAZARD	325058	33537
2	JAM	433526	56009
3	ROAD_CLOSED	69776	372485

I decided to keep the NAs, as the proportion of cases that are NA per type is large, as shown in the table above.

4.

a.

```
import pandas as pd
waze_data = pd.read_csv("C:/Users/prash/Downloads/waze_data/waze_data.csv")
waze_data['subtype'] = waze_data['subtype'].fillna('Unclassified')

# Create the DataFrame with the specified columns
type_subtype_df = pd.DataFrame({
    "type": waze_data['type'],
    "subtype": waze_data['subtype'],
    "updated_type": waze_data['type']
})

# Extracting Updated Subtypes
# Create a function to identify updated_subtypes
def identify_updated_subtype(subtype):
    if pd.isna(subtype):
        return 'Unclassified'
    elif 'ACCIDENT' in subtype and 'ACCIDENT_MAJOR' in subtype:
        return "MAJOR"
    elif 'ACCIDENT' in subtype and 'ACCIDENT_MINOR' in subtype:
        return "MINOR"
    elif 'HAZARD_ON_ROAD' in subtype and 'HAZARD_ON_ROAD' in subtype:
        return "ON_ROAD"
```

```
    elif 'HAZARD' in subtype and 'HAZARD_ON_SHOULDER' in subtype:
        return "ON_SHOULDER"
    elif 'HAZARD' in subtype and 'HAZARD_WEATHER' in subtype:
        return "WEATHER"
    elif 'JAM' in subtype and 'JAM_HEAVY_TRAFFIC' in subtype:
        return "HEAVY_TRAFFIC"
    elif 'JAM' in subtype and 'JAM_MODERATE_TRAFFIC' in subtype:
        return "MODERATE_TRAFFIC"
    elif 'JAM' in subtype and 'JAM_LIGHT_TRAFFIC' in subtype:
        return "LIGHT_TRAFFIC"
    elif 'JAM' in subtype and 'JAM_STAND_STILL_TRAFFIC' in subtype:
        return "STAND_STILL_TRAFFIC"
    elif 'ROAD' in subtype and 'ROAD_CLOSED_EVENT' in subtype:
        return "EVENT"
    elif 'ROAD' in subtype and 'ROAD_CLOSED_CONSTRUCTION' in subtype:
        return "CONSTRUCTION"
    elif 'ROAD' in subtype and 'ROAD_CLOSED_HAZARD' in subtype:
        return "HAZARD"
    else:
        return "Unclassified"

# Apply the function to create a new column 'sub_subtype'
waze_data['updated_subtype'] = waze_data['subtype'].apply(identify_updated_subtype)

# Display the updated DataFrame
type_subtype_df['updated_subtype'] = waze_data['updated_subtype']

# BingChat Citation: How do I use a function to extract the subtype from a type?
```

Adding sub-subtypes

```
# Create a function to identify sub_subtypes
def identify_sub_subtype(subtype):
    if pd.isna(subtype):
        return None
    elif 'ACCIDENT' in subtype and 'ACCIDENT_MAJOR' in subtype:
        return None
    elif 'ACCIDENT' in subtype and 'ACCIDENT_MINOR' in subtype:
        return None
    elif 'HAZARD' in subtype and 'HAZARD_ON_ROAD_LANE_CLOSED' in subtype:
        return "LANE_CLOSED"
    elif 'HAZARD' in subtype and 'HAZARD_ON_ROAD_ROAD_KILL' in subtype:
        return "ROAD_KILL"
    elif 'HAZARD' in subtype and 'HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT' in subtype:
        return "TRAFFIC_LIGHT_FAULT"
    elif 'HAZARD' in subtype and 'HAZARD_ON_ROAD_POT_HOLE' in subtype:
        return "TRAFFIC_POT_HOLE"
    elif 'HAZARD' in subtype and 'HAZARD_ON_ROAD_OBJECT' in subtype:
        return "OBJECT"
    elif 'HAZARD' in subtype and 'HAZARD_ON_ROAD_ICE' in subtype:
        return "ICE"
    elif 'HAZARD' in subtype and 'HAZARD_ON_ROAD_EMERGENCY_VEHICLE' in subtype:
        return "EMERGENCY_VEHICLE"
    elif 'HAZARD' in subtype and 'HAZARD_ON_ROAD_CONSTRUCTION' in subtype:
        return "CONSTRUCTION"
    elif 'HAZARD' in subtype and 'HAZARD_ON_ROAD_CAR_STOPPED' in subtype:
```

```
        return "CAR_STOPPED"
    elif 'HAZARD' in subtype and 'HAZARD_ON_SHOULDER_CAR_STOPPED' in subtype:
        return "CAR_STOPPED"
    elif 'HAZARD' in subtype and 'HAZARD_ON_SHOULDER_ANIMALS' in subtype:
        return "ANIMALS"
    elif 'HAZARD' in subtype and 'HAZARD_ON_SHOULDER_MISSING_SIGN' in subtype:
        return "MISSING_SIGN"
    elif 'HAZARD' in subtype and 'HAZARD_WEATHER_FLOOD' in subtype:
        return "FLOOD"
    elif 'HAZARD' in subtype and 'HAZARD_WEATHER_FOG' in subtype:
        return "FOG"
    elif 'HAZARD' in subtype and 'HAZARD_WEATHER_HAIL' in subtype:
        return "HAIL"
    elif 'HAZARD' in subtype and 'HAZARD_WEATHER_HEAVY_SNOW' in subtype:
        return "HEAVY_SNOW"
    elif 'JAM' in subtype and 'JAM_HEAVY_TRAFFIC' in subtype:
        return None
    elif 'JAM' in subtype and 'JAM_MODERATE_TRAFFIC' in subtype:
        return None
    elif 'JAM' in subtype and 'JAM_LIGHT_TRAFFIC' in subtype:
        return None
    elif 'JAM' in subtype and 'JAM_STAND_STILL_TRAFFIC' in subtype:
        return None
    elif 'ROAD' in subtype and 'ROAD_CLOSED_EVENT' in subtype:
        return None
    elif 'ROAD' in subtype and 'ROAD_CLOSED_CONSTRUCTION' in subtype:
        return None
    elif 'ROAD' in subtype and 'ROAD_CLOSED_HAZARD' in subtype:
        return None
    else:
        return None

waze_data['updated_sub_subtype'] = waze_data['subtype'].apply(identify_sub_subtype)
type_subtype_df['updated_sub_subtype'] = waze_data['updated_sub_subtype']

# Printing dataframe's first 5 rows with needed columns
print(type_subtype_df.head(5))
```

```
           type       subtype updated_type updated_subtype updated_sub_subtype
0           JAM  Unclassified          JAM    Unclassified                 None
1      ACCIDENT  Unclassified      ACCIDENT    Unclassified                 None
2   ROAD_CLOSED  Unclassified   ROAD_CLOSED    Unclassified                 None
3           JAM  Unclassified          JAM    Unclassified                 None
4           JAM  Unclassified          JAM    Unclassified                 None
```

b.

```
import pandas as pd

# Making crosswalk manually
crosswalk = [
    {'updated_type': 'ACCIDENT', 'updated_subtype': "Unclassified", 'Sub-Subtype': None},
    {'updated_type': 'ACCIDENT', 'updated_subtype': 'MAJOR', 'Sub-Subtype': None},
    {'updated_type': 'ACCIDENT', 'updated_subtype': 'MINOR', 'Sub-Subtype': None},
    {'updated_type': 'HAZARD', 'updated_subtype': "Unclassified", 'Sub-Subtype': None},
    {'updated_type': 'HAZARD', 'updated_subtype': 'ON_ROAD', 'Sub-Subtype': None},
    {'updated_type': 'HAZARD', 'updated_subtype': 'ON_ROAD', 'Sub-Subtype': 'CAR_STOPPED'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'ON_ROAD', 'Sub-Subtype': 'CONSTRUCTION'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'ON_ROAD', 'Sub-Subtype':
        'EMERGENCY_VEHICLE'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'ON_ROAD', 'Sub-Subtype': 'ICE'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'ON_ROAD', 'Sub-Subtype': 'OBJECT'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'ON_ROAD', 'Sub-Subtype': 'POT_HOLE'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'ON_ROAD', 'Sub-Subtype':
        'TRAFFIC_LIGHT_FAULT'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'ON_SHOULDER', 'Sub-Subtype': None},

    {'updated_type': 'HAZARD', 'updated_subtype': 'ON_SHOULDER', 'Sub-Subtype':
        'CAR_STOPPED'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'WEATHER', 'Sub-Subtype': None},
    {'updated_type': 'HAZARD', 'updated_subtype': 'WEATHER', 'Sub-Subtype': 'FLOOD'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'ON_ROAD', 'Sub-Subtype': 'LANE_CLOSED'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'WEATHER', 'Sub-Subtype': 'FOG'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'ON_ROAD', 'Sub-Subtype': 'ROAD_KILL'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'ON_SHOULDER', 'Sub-Subtype': 'ANIMALS'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'ON_SHOULDER', 'Sub-Subtype':
        'MISSING_SIGN'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'WEATHER', 'Sub-Subtype': 'HEAVY_SNOW'},
    {'updated_type': 'HAZARD', 'updated_subtype': 'WEATHER', 'Sub-Subtype': 'HAIL'},
    {'updated_type': 'JAM', 'updated_subtype': "Unclassified", 'Sub-Subtype': None},
    {'updated_type': 'JAM', 'updated_subtype': 'HEAVY_TRAFFIC', 'Sub-Subtype': None},
    {'updated_type': 'JAM', 'updated_subtype': 'MODERATE_TRAFFIC', 'Sub-Subtype': None},
    {'updated_type': 'JAM', 'updated_subtype': 'STAND_STILL_TRAFFIC', 'Sub-Subtype': None},
    {'updated_type': 'JAM', 'updated_subtype': 'LIGHT_TRAFFIC', 'Sub-Subtype': None},
    {'updated_type': 'ROAD_CLOSED', 'updated_subtype': "Unclassified", 'Sub-Subtype': None},
    {'updated_type': 'ROAD_CLOSED', 'updated_subtype': 'EVENT', 'Sub-Subtype': None},
    {'updated_type': 'ROAD_CLOSED', 'updated_subtype': 'CONSTRUCTION', 'Sub-Subtype': None},
    {'updated_type': 'ROAD_CLOSED', 'updated_subtype': 'HAZARD', 'Sub-Subtype': None},
]

# Convert the list of dictionaries into a DataFrame
crosswalk_df = pd.DataFrame(crosswalk)

# Taking only the relevant columns
updated_crosswalk = crosswalk_df[['updated_type', 'updated_subtype', 'Sub-Subtype']]
updated_crosswalk['sub-subtype'] = crosswalk_df['Sub-Subtype']
updated_crosswalk['type'] = crosswalk_df['updated_type']
updated_crosswalk = updated_crosswalk[['type', 'updated_subtype', 'sub-subtype']]

# Display the first 5 rows of the updated DataFrame
```

```
print(updated_crosswalk.sort_values(by = ['type', 'updated_subtype']).reset_index(drop =
        True).head(5))
```

```
      type updated_subtype  sub-subtype
0  ACCIDENT            MAJOR         None
1  ACCIDENT            MINOR         None
2  ACCIDENT    Unclassified          None
3    HAZARD          ON_ROAD         None
4    HAZARD          ON_ROAD  CAR_STOPPED
```

c.

```
# Merge the original data with the crosswalk
merged_df = pd.merge(waze_data, updated_crosswalk, on=['type', 'updated_subtype'],
        how='left')

# Filter for Accident - Unclassified
acc_unc = merged_df[(merged_df['type'] == "ACCIDENT") & (merged_df['updated_subtype'] ==
        "Unclassified")]

# The number of Accident - Unclassified cases
print( "The number of Accident - Unclassified cases is = ", len(acc_unc))
```

```
The number of Accident - Unclassified cases is =  24359
```

d.

```
print("The type values in the original dataframe are:", merged_df['type'].unique())
print("The subtype values in the original dataframe are:",
        merged_df['updated_subtype'].unique())
print("The type values in the crosswalk dataframe are:", updated_crosswalk['type'].unique())
print("The subtype values in the crosswalk dataframe are:",
        updated_crosswalk['updated_subtype'].unique())
```

```
The type values in the original dataframe are: ['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD'] The
subtype values in the original dataframe are: ['Unclassified' 'MAJOR' 'MINOR' 'ON_ROAD'
'ON_SHOULDER' 'WEATHER'
 'HEAVY_TRAFFIC' 'MODERATE_TRAFFIC' 'STAND_STILL_TRAFFIC' 'EVENT'
 'CONSTRUCTION' 'LIGHT_TRAFFIC' 'HAZARD']
The type values in the crosswalk dataframe are: ['ACCIDENT' 'HAZARD' 'JAM' 'ROAD_CLOSED'] The
subtype values in the crosswalk dataframe are: ['Unclassified' 'MAJOR' 'MINOR' 'ON_ROAD'
'ON_SHOULDER' 'WEATHER'
 'HEAVY_TRAFFIC' 'MODERATE_TRAFFIC' 'STAND_STILL_TRAFFIC' 'LIGHT_TRAFFIC'
 'EVENT' 'CONSTRUCTION' 'HAZARD']
```

Therefore, the crosswalk and merged dataset have the same types and subtypes.

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

1.

a.

```
import re
import pandas as pd

# Define the regex pattern
pattern = r"POINT\((-?\d+\.\d+) (-?\d+\.\d+)\)"

# Define the function to extract coordinates
def extract_lat_long(point_str):
    matches = re.match(pattern, point_str)
    if matches:
        longitude = matches.group(1) latitude =
        matches.group(2)
        return pd.Series([latitude, longitude])
    else:
        return pd.Series([None, None])

# Read the CSV file into a DataFrame - NOT NEEDED IN FINAL
waze_data = pd.read_csv("C:/Users/prash/Downloads/waze_data/waze_data.csv")

# Apply the function to the 'geo' column
waze_data[['latitude','longitude']] = waze_data['geo'].apply(extract_lat_long)
```

```
# Display the updated DataFrame
print(waze_data.head(5))

# BingChat Citation: How do I use regex to extract longitude and latitude from geo data?
# Output was a pseudo-code:
# import re
# import pandas as pd

# Define the regex pattern
# pattern = r"POINT\((-?\d+\.\d+) (-?\d+\.\d+)\)"

# Define the function to extract coordinates
# def extract_lat_long(point_str):
#     matches = re.match(pattern, point_str)
#     if matches:
#         longitude = matches.group(1)
#         latitude = matches.group(2)
#         return pd.Series([latitude, longitude])
#     else:
#         return pd.Series([None, None])

# Apply the function to the 'geo' column
# df[['latitude','longitude']] = df['geo'].apply(extract_lat_long)
```

```
          city  confidence  nThumbsUp street  \
0  Chicago, IL           0        NaN    NaN
1  Chicago, IL           1        NaN    NaN
2  Chicago, IL           0        NaN    NaN
3  Chicago, IL           0        NaN  Alley
4  Chicago, IL           0        NaN  Alley

                                   uuid country          type subtype  \
0  004025a4-5f14-4cb7-9da6-2615daafbf37      US           JAM     NaN
1  ad7761f8-d3cb-4623-951d-dafb419a3ec3      US     ACCIDENT     NaN
2  0e5f14ae-7251-46af-a7f1-53a5272cd37d      US   ROAD_CLOSED     NaN
3  654870a4-a71a-450b-9f22-bc52ae4f69a5      US           JAM     NaN
4  926ff228-7db9-4e0d-b6cf-6739211ffc8b      US           JAM     NaN

   roadType  reliability  magvar  reportRating                       ts  \
0        20            5     139             3  2024-02-04 16:40:41 UTC
1         4            8       2             2  2024-02-04 20:01:27 UTC
2         1            5     344             2  2024-02-04 02:15:54 UTC
3        20            5     264             2  2024-02-04 00:30:54 UTC
4        20            5     359             0  2024-02-04 03:27:35 UTC

                          geo                       geoWKT   latitude  \
0  POINT(-87.676685 41.929692)  Point(-87.676685 41.929692)  41.929692
1  POINT(-87.624816 41.753358)  Point(-87.624816 41.753358)  41.753358
2  POINT(-87.614122 41.889821)  Point(-87.614122 41.889821)  41.889821
3  POINT(-87.680139 41.939093)  Point(-87.680139 41.939093)  41.939093
4   POINT(-87.735235 41.91658)   Point(-87.735235 41.91658)   41.91658

    longitude
0  -87.676685
1  -87.624816
2  -87.614122
3  -87.680139
4  -87.735235
```

b.

```
import pandas as pd

# Ensure latitude and longitude columns are numeric
waze_data['latitude'] = pd.to_numeric(waze_data['latitude'], errors='coerce') # BingChat
        helped me do this, yielded errors without this step
waze_data['longitude'] = pd.to_numeric(waze_data['longitude'], errors='coerce')

# Round the latitude and longitude values to 2 decimal places
waze_data['latitude'] = waze_data['latitude'].round(2)
waze_data['longitude'] = waze_data['longitude'].round(2)

# Grouping by latitude and longitude to find top 10 latitude-longitude bins
lat_long_comb_counts = waze_data.groupby(['latitude',
        'longitude']).size().reset_index(name='count')

# Most common longitude latitude bins
print("The most common combination is", lat_long_comb_counts.max())
```

```
# BingChat Citation: I am getting an error when I run just this:
# waze_data['latitude'] = waze_data['latitude'].round(2)
# waze_data['longitude'] = waze_data['longitude'].round(2)
```

```
The most common combination is latitude         42.02
longitude       -87.56
count         21325.00
dtype: float64
```

c.

```
import pandas as pd
chosen_type = 'ACCIDENT'
chosen_subtype = 'ACCIDENT_MAJOR'
chosen_data = waze_data[(waze_data['type'] == chosen_type) & (waze_data['subtype'] ==
          chosen_subtype)]

# Group by latitude and longitude, and count the number of alerts
grouped_data = chosen_data.groupby(['latitude',
          'longitude']).size().reset_index(name='count')

# Sort by the number of alerts and select the top 10 bins
top_alerts_map = grouped_data.sort_values(by='count', ascending=False)

output_folder_alerts = 'C:/Users/prash/OneDrive/Documents/Python_II/PSet_6/top_alerts_map'
output_file_alerts = f"{output_folder_alerts}/top_alerts_map.csv"
top_alerts_map.to_csv(output_file_alerts, index=False)

# Save the result to a CSV file
```

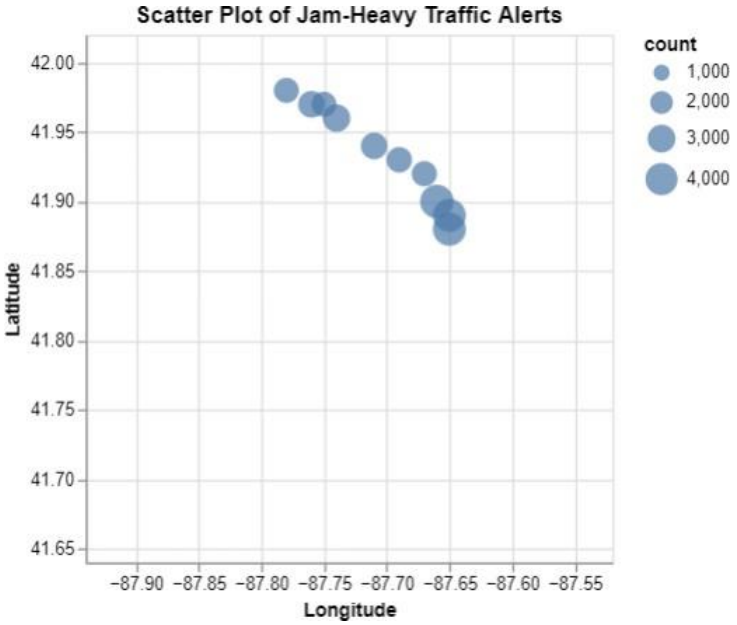
```
top_alerts_map_file =
          "C:/Users/prash/OneDrive/Documents/Python_II/top_alerts_map/top_alerts_map.csv"
top_alerts_map.to_csv(top_alerts_map_file, index=False)

# Display the level of aggregation and number of rows
print("The level of aggregation: Latitude-Longitude Bins")
print(f"Number of Rows in the DataFrame: ", len(top_alerts_map))

# BingChat Citation: How do I save a dataframe as a csv?
# This yielded a pseudo-code, which I adapted to this
```

```
The level of aggregation: Latitude-Longitude Bins
Number of Rows in the DataFrame:  425
```

2.

```
import pandas as pd
import altair as alt

chosen_type = 'JAM'
chosen_subtype = 'JAM_HEAVY_TRAFFIC'
chosen_data = waze_data[(waze_data['type'] == chosen_type) & (waze_data['subtype'] ==
          chosen_subtype)]

# Group by latitude and longitude, and count the number of alerts
```

```
grouped_data = chosen_data.groupby(['latitude',
          'longitude']).size().reset_index(name='count')

# Sort by the number of alerts and select the top 10 bins
jam_top_alerts_map = grouped_data.sort_values(by='count', ascending=False).head(10)

# Create a scatter plot
scatter_plot = alt.Chart(jam_top_alerts_map).mark_circle(size=60).encode(
    x=alt.X('longitude', scale=alt.Scale(domain=[-87.94, -87.52]), title='Longitude'),
    y=alt.Y('latitude', scale=alt.Scale(domain=[41.64, 42.02]), title='Latitude'),
    size='count',
    tooltip=['latitude', 'longitude', 'count']
).properties(
    title='Scatter Plot of Jam-Heavy Traffic Alerts',
)

# Display the plot
scatter_plot.display()

# BingChat Citation: What are the minimum and maximum longitudinal values in Chicago?
```



3.

a.

```
import geopandas as gpd
# Define the path to your GeoJSON file
file_path = "C:/Users/prash/Downloads/Boundaries - Neighborhoods.geojson"


    # Read the GeoJSON file
    gdf = gpd.read_file(file_path)
```

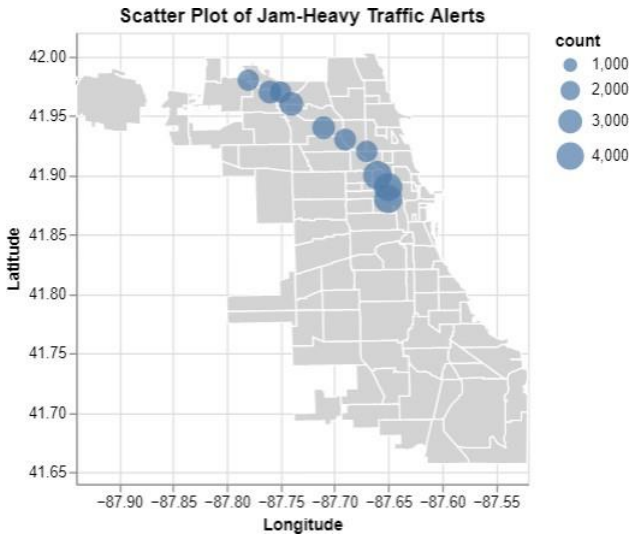
b.

```
import json
import altair as alt

# Define the path to your GeoJSON file
file_path = "C:/Users/prash/Downloads/Boundaries - Neighborhoods.geojson"

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

# Prepare the GeoJSON data for use in Altair
geo_data = alt.Data(values=chicago_geojson["features"])

# BingChat Citation: How do I troubleshoot from here:
# file_path = "C:/Users/prash/Downloads/Boundaries - Neighborhoods.geojson"
#.....
# with open(file_path) as f:
# chicago_geojson = json.load(f)
# geo_data = alt.Data(values=chicago_geojson["features"])
```

4.

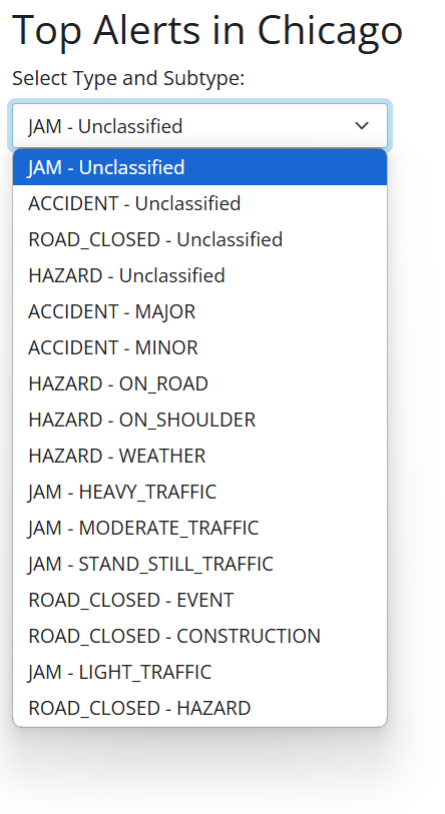
```
import altair as alt
import geopandas as gpd
import pandas as pd
import json

# Load the GeoJSON file using GeoPandas
file_path = "C:/Users/prash/Downloads/Boundaries - Neighborhoods.geojson"
chicago_geojson = gpd.read_file(file_path)

# Ensure using the same coordinate system
chicago_geojson = chicago_geojson.to_crs(epsg=4326)

# Convert GeoDataFrame to JSON format suitable for Altair
chicago_geojson_json = json.loads(chicago_geojson.to_json())

# Ensure latitude and longitude are numeric
jam_top_alerts_map['latitude'] = pd.to_numeric(jam_top_alerts_map['latitude'])
jam_top_alerts_map['longitude'] = pd.to_numeric(jam_top_alerts_map['longitude'])
jam_top_alerts_map['count'] = pd.to_numeric(jam_top_alerts_map['count'])

# Create the map layer
map_layer = alt.Chart(alt.Data(values=chicago_geojson_json["features"])).mark_geoshape(
    fill='lightgray',
    stroke='white'
).project(type = "equirectangular")

# Create final chart layering
```

```
final_chart = map_layer + scatter_plot

# Display the final chart
final_chart.display()
```



5.

a. App 1 UI Dropdown:



App1_UI

From this, we can see that there are 16 type-subtype combinations.

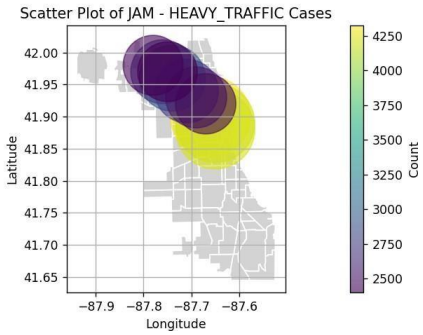
BingChat Citation 1: How do I use a lambda function to return type and updated_subtype in each dropdown? BingChat Citation 2: How do I convert the map layer plot and scatterplot to matplotlib?

b. Jam-Heavy Traffic Plot:

Top Alerts in Chicago

Select Type and Subtype:

JAM - HEAVY_TRAFFIC



App1_Jam_HEAVY

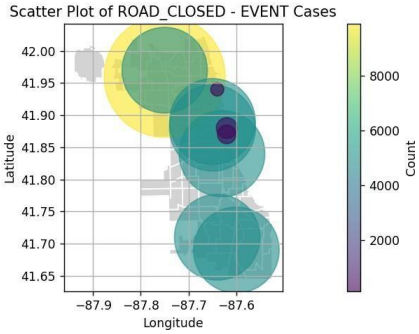
c. They are most common in the northern part of Chicago.

Road Closed-Event Plot:

Top Alerts in Chicago

Select Type and Subtype:

ROAD_CLOSED - EVENT



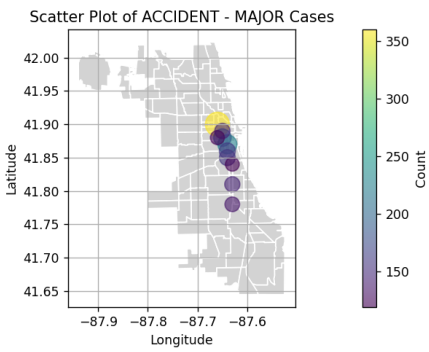
App1_Event

d. A question like this could be: Where do the highest number of Major Accidents occur in Chicago?
Answer: The highest number of Major Accidents occur in Northeastern Chicago.

Top Alerts in Chicago

Select Type and Subtype:

ACCIDENT - MAJOR



App1_Accidents

e. Adding a column on the duration of the alert would aid in analysis.

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

1.

a.

```
import pandas as pd

print(waze_data['ts'].head(5))
```

```
0    2024-02-04 16:40:41 UTC
1    2024-02-04 20:01:27 UTC
2    2024-02-04 02:15:54 UTC
3    2024-02-04 00:30:54 UTC
4    2024-02-04 03:27:35 UTC
Name: ts, dtype: object
```

No, it would not be a good idea to collapse on this column, as it has very specific information each alert. We want to use only the hour to see how many alerts there are of a specified type and subtype in a particular hour.

b.

```
import re
import pandas as pd

# Define the regex pattern
pattern = r"POINT\((-?\d+\.\d+) (-?\d+\.\d+)\)"

# Define the function to extract coordinates
def extract_lat_long(point_str):
    matches = re.match(pattern, point_str)
    if matches:
        longitude = matches.group(1)
        latitude = matches.group(2)
        return pd.Series([latitude, longitude])
    else:
        return pd.Series([None, None])

# Read the CSV file into a DataFrame - NOT NEEDED IN FINAL
waze_data = pd.read_csv("C:/Users/prash/Downloads/waze_data/waze_data.csv")

# Apply the function to the 'geo' column
waze_data[['latitude','longitude']] = waze_data['geo'].apply(extract_lat_long)

waze_data['latitude'] = pd.to_numeric(waze_data['latitude'], errors='coerce')
waze_data['longitude'] = pd.to_numeric(waze_data['longitude'], errors='coerce')

# Round the latitude and longitude values to 2 decimal places
waze_data['latitude'] = waze_data['latitude'].round(2)
waze_data['longitude'] = waze_data['longitude'].round(2)

# Creating 'hour' variable by extracting from 'ts'
waze_data['hour'] = waze_data['ts'].str.replace(' UTC', '')
```

```
# Converting to datetime format
waze_data['hour'] = pd.to_datetime(waze_data['hour'])

# Extracting only hour from total timestamp
waze_data['hour'] = waze_data['hour'].dt.floor('H')
waze_data['hour'] = waze_data['hour'].dt.strftime('%H:%M') # This and previous line from
         BingChat

# Group by latitude and longitude, and count the number of alerts
grouped_data = waze_data.groupby(['latitude', 'longitude', 'type', 'subtype',
          'hour']).size().reset_index(name='count')

# Sorting data by count
top_alerts_map_hour = grouped_data.sort_values(by='count', ascending=False)

# Save the result to a CSV file
output_folder = "C:/Users/prash/OneDrive/Documents/Python_II/PSet_6/top_alerts_map_byhour"
output_file = f"{output_folder}/top_alerts_map_byhour.csv"
top_alerts_map_hour.to_csv(output_file, index=False)

# Display the level of aggregation and number of rows
print("The levels of aggregation: Hour, Type, Subtype, Latitude, and Longitude")
print(f"Number of Rows in the DataFrame: ", len(top_alerts_map_hour))

# BingChat Citation: How do convert 01:30 to 01:00, and leave it in this format?
```

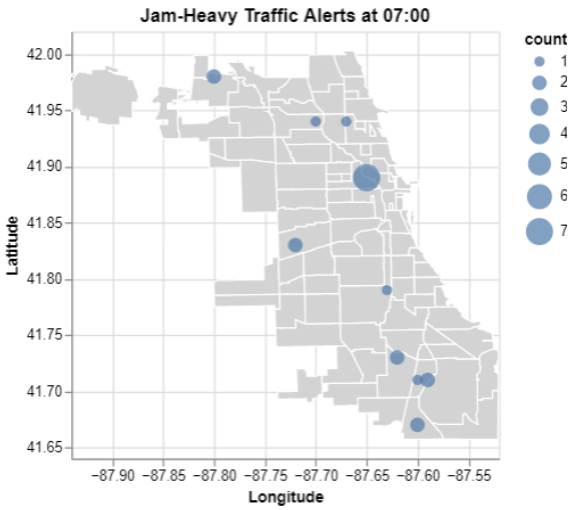
```
The levels of aggregation: Hour, Type, Subtype, Latitude, and Longitude
Number of Rows in the DataFrame: 68892
```

c.

at 07:00:

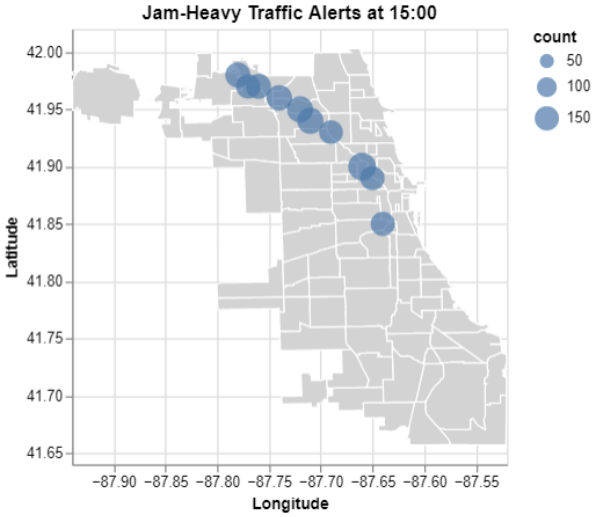
```
import pandas as pd
import geopandas as gpd
import json
import altair as alt

# Define the path to your GeoJSON file
file_path = "C:/Users/prash/Downloads/Boundaries - Neighborhoods.geojson"

# Read the GeoJSON file
gdf = gpd.read_file(file_path)

# Define the path to your GeoJSON file
file_path = "C:/Users/prash/Downloads/Boundaries - Neighborhoods.geojson"

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

# Prepare the GeoJSON data for use in Altair
geo_data = alt.Data(values=chicago_geojson["features"])

# Load the GeoJSON file using GeoPandas
file_path = "C:/Users/prash/Downloads/Boundaries - Neighborhoods.geojson"
chicago_geojson = gpd.read_file(file_path)

# Ensure using the same coordinate system
chicago_geojson = chicago_geojson.to_crs(epsg=4326)

# Convert GeoDataFrame to JSON format suitable for Altair
chicago_geojson_json = json.loads(chicago_geojson.to_json())

# Create the map layer
map_layer = alt.Chart(alt.Data(values=chicago_geojson_json["features"])).mark_geoshape(
    fill='lightgray',
    stroke='white'
).project(type = "equirectangular")

# Chosing type, subtype, and hour
chosen_type = 'JAM'
chosen_subtype = 'JAM_HEAVY_TRAFFIC'
chosen_hour = "07:00"
chosen_data_hour = waze_data[(waze_data['type'] == chosen_type) & (waze_data['subtype'] ==
         chosen_subtype) & (waze_data['hour'] == chosen_hour)]

# Group by latitude and longitude, and count the number of alerts
grouped_data_hour = chosen_data_hour.groupby(['latitude',
         'longitude']).size().reset_index(name='count')

# Sort by the number of alerts and select the top 10 bins
hour_top_alerts_map = grouped_data_hour.sort_values(by='count', ascending=False).head(10)

# Ensure latitude and longitude are numeric
hour_top_alerts_map['latitude'] = pd.to_numeric(hour_top_alerts_map['latitude'])
hour_top_alerts_map['longitude'] = pd.to_numeric(hour_top_alerts_map['longitude'])
```

```
hour_top_alerts_map['count'] = pd.to_numeric(hour_top_alerts_map['count'])

# Create a scatter plot
scatter_plot_hour = alt.Chart(hour_top_alerts_map).mark_circle(size=60).encode(
    x=alt.X('longitude', scale=alt.Scale(domain=[-87.94, -87.52]), title='Longitude'), #
        Ensure the scale doesn't force zero
    y=alt.Y('latitude', scale=alt.Scale(domain=[41.64, 42.02]), title='Latitude'), # chicago
        min max chat gpt
    size='count',
    tooltip=['latitude', 'longitude', 'count']
).properties(
    title='Jam-Heavy Traffic Alerts at 07:00'
)

# Layering with map
hour_final_plot =  map_layer + scatter_plot_hour

# Displaying scatterplot
hour_final_plot.display()
```



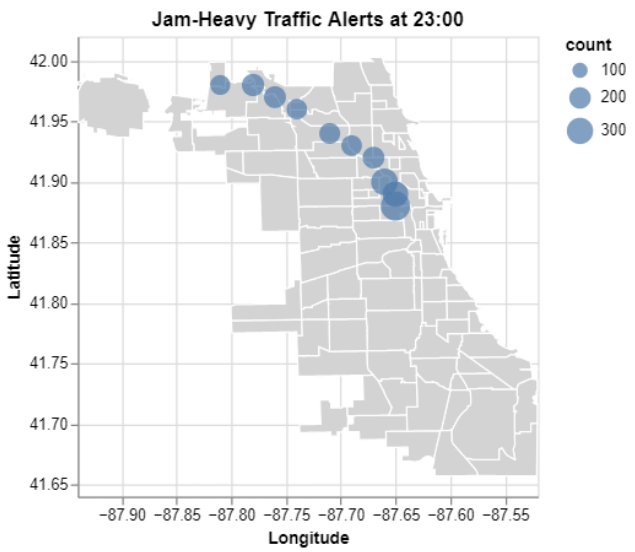
At 15:00:

```
# Chosing type, subtype, and hour
chosen_type = 'JAM'
chosen_subtype = 'JAM_HEAVY_TRAFFIC'
chosen_hour = "15:00"
chosen_data_hour = waze_data[(waze_data['type'] == chosen_type) & (waze_data['subtype'] ==
        chosen_subtype) & (waze_data['hour'] == chosen_hour)]

# Group by latitude and longitude, and count the number of alerts
grouped_data_hour = chosen_data_hour.groupby(['latitude',
        'longitude']).size().reset_index(name='count')
```


Jam-Heavy Traffic Alerts at 15:00
count
50
100
150
42.00
41.95
41.90
41.85
41.80
41.75
41.70
41.65
Latitude
−87.90 −87.85 −87.80 −87.75 −87.70 −87.65 −87.60 −87.55
Longitude

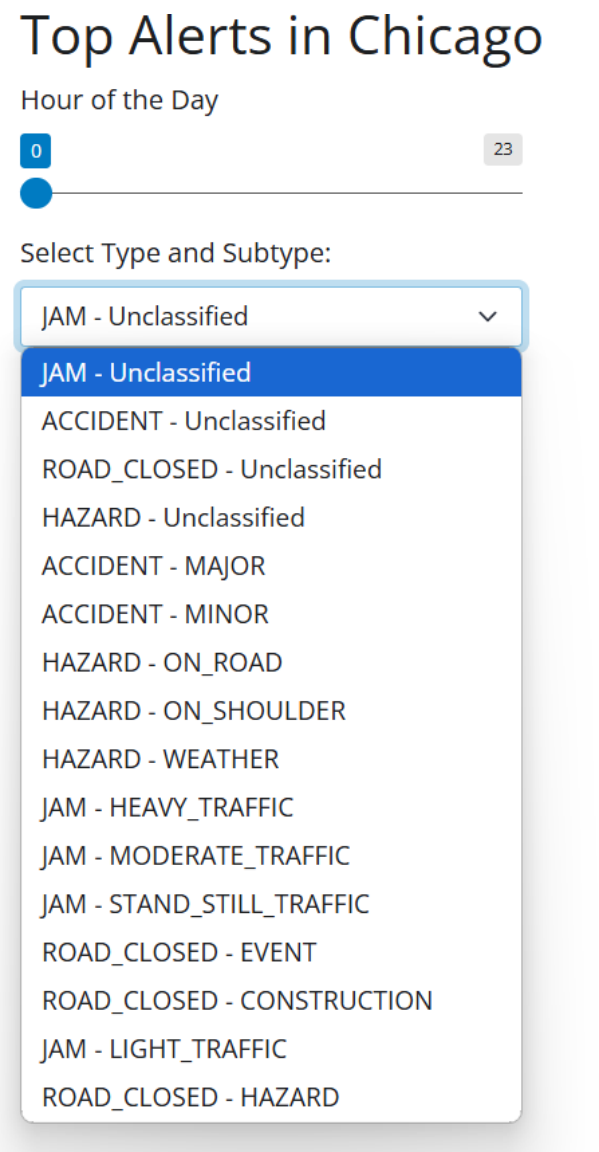
At 23:00:

```
# Chosing type, subtype, and hour
chosen_type = 'JAM'
chosen_subtype = 'JAM_HEAVY_TRAFFIC'
chosen_hour = "23:00"
chosen_data_hour = waze_data[(waze_data['type'] == chosen_type) & (waze_data['subtype'] ==
        chosen_subtype) & (waze_data['hour'] == chosen_hour)]

# Group by latitude and longitude, and count the number of alerts
grouped_data_hour = chosen_data_hour.groupby(['latitude',
        'longitude']).size().reset_index(name='count')

# Sort by the number of alerts and select the top 10 bins
hour_top_alerts_map = grouped_data_hour.sort_values(by='count', ascending=False).head(10)

# Ensure latitude and longitude are numeric
hour_top_alerts_map['latitude'] = pd.to_numeric(hour_top_alerts_map['latitude'])
hour_top_alerts_map['longitude'] = pd.to_numeric(hour_top_alerts_map['longitude'])
hour_top_alerts_map['count'] = pd.to_numeric(hour_top_alerts_map['count'])

# Create a scatter plot
scatter_plot_hour = alt.Chart(hour_top_alerts_map).mark_circle(size=60).encode(
    x=alt.X('longitude', scale=alt.Scale(domain=[-87.94, -87.52]), title='Longitude'), #
        Ensure the scale doesn't force zero
    y=alt.Y('latitude', scale=alt.Scale(domain=[41.64, 42.02]), title='Latitude'), # chicago
        min max chat gpt
    size='count',
    tooltip=['latitude', 'longitude', 'count']
).properties(
    title='Jam-Heavy Traffic Alerts at 23:00'
)

# Layering with map
hour_final_plot =  map_layer + scatter_plot_hour

# Displaying scatterplot
hour_final_plot.display()
```



2.

a. App 2 UI Dropdown and Slider:



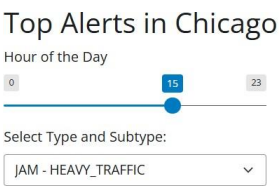
App2_UI

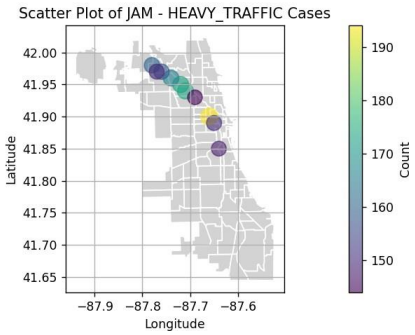
BingChat Citation: How do convert datetime to integer/numeric?

b. Jam-Heavy Traffic at 07:00:

Jam_HEAVY_700

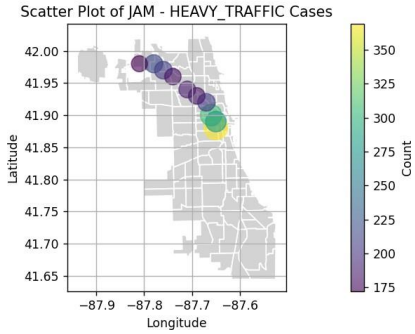
Jam-Heavy Traffic at 15:00:





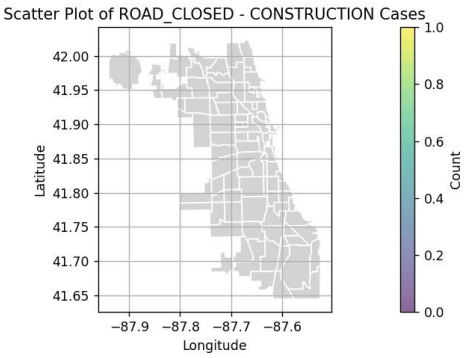
Jam_HEAVY_1500

Jam-Heavy Traffic at 23:00:



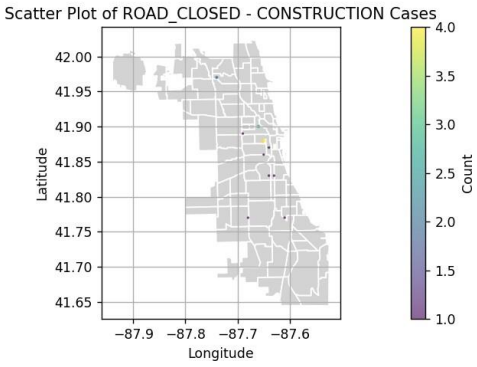
Jam_HEAVY_2300

c. Road Construction at 09:00:



App2_Road_Morning

Road Construction at 21:00:



App2_Road_Night

From these images, it seems like more construction is done at night than in the morning, as there are fewer alerts in the morning.

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

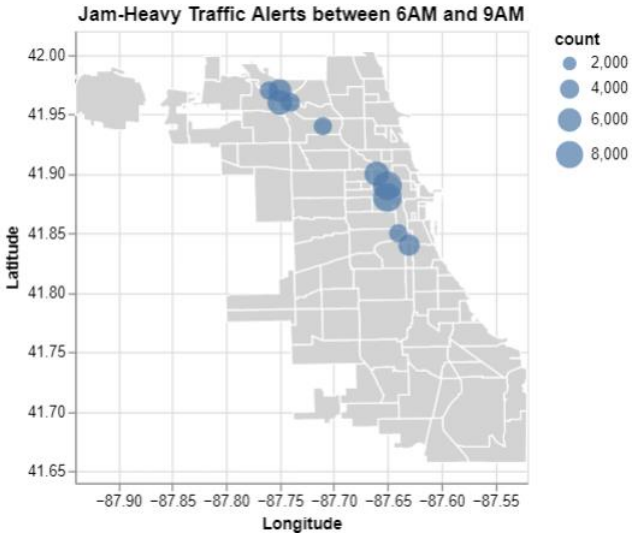
1.

a. It would be a good idea to collapse data by range of hours, as it would give one a better idea around what time of day particular alerts are at their highest or lowest. One could find out what kinds of alerts are most frequent and when during the day this is so.

b.

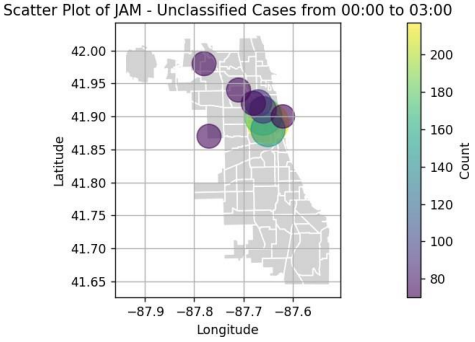
```
import pandas as pd
import geopandas as gpd
import json
import altair as alt

# Define the path to your GeoJSON file
file_path = "C:/Users/prash/Downloads/Boundaries - Neighborhoods.geojson"

# Read the GeoJSON file
gdf = gpd.read_file(file_path)

# Define the path to your GeoJSON file
file_path = "C:/Users/prash/Downloads/Boundaries - Neighborhoods.geojson"

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

# Prepare the GeoJSON data for use in Altair
geo_data = alt.Data(values=chicago_geojson["features"])

# Load the GeoJSON file using GeoPandas
file_path = "C:/Users/prash/Downloads/Boundaries - Neighborhoods.geojson"
chicago_geojson = gpd.read_file(file_path)

# Ensure using the same coordinate system
chicago_geojson = chicago_geojson.to_crs(epsg=4326)

# Convert GeoDataFrame to JSON format suitable for Altair
chicago_geojson_json = json.loads(chicago_geojson.to_json())

# Create the map layer
map_layer = alt.Chart(alt.Data(values=chicago_geojson_json["features"])).mark_geoshape(
    fill='lightgray',
    stroke='white'
).project(type = "equirectangular")

chosen_type = 'JAM'
chosen_subtype = 'JAM_HEAVY_TRAFFIC'
chosen_start_hour = "06:00"
chosen_end_hour = "09:00"
chosen_data_range = waze_data[(waze_data['type'] == chosen_type) & (waze_data['subtype'] ==
         chosen_subtype) & (waze_data['hour'] >= chosen_start_hour) | (waze_data['hour'] <=
         chosen_end_hour)]

# Group by latitude and longitude, and count the number of alerts
grouped_data_range = chosen_data_range.groupby(['latitude',
         'longitude']).size().reset_index(name='count')

# Sort by the number of alerts and select the top 10 bins
range_top_alerts_map = grouped_data_range.sort_values(by='count', ascending=False).head(10)
```

```
# Create a scatter plot
scatter_plot_range = alt.Chart(range_top_alerts_map).mark_circle(size=60).encode(
    x=alt.X('longitude', scale=alt.Scale(domain=[-87.94, -87.52]), title='Longitude'), #
        Ensure the scale doesn't force zero
    y=alt.Y('latitude', scale=alt.Scale(domain=[41.64, 42.02]), title='Latitude'), # chicago
        min max chat gpt
    size='count',
    tooltip=['latitude', 'longitude', 'count']
).properties(
    title='Jam-Heavy Traffic Alerts between 6AM and 9AM'
)

# Layering with map
range_final_plot = map_layer + scatter_plot_range

# Displaying scatterplot
range_final_plot.display()
```



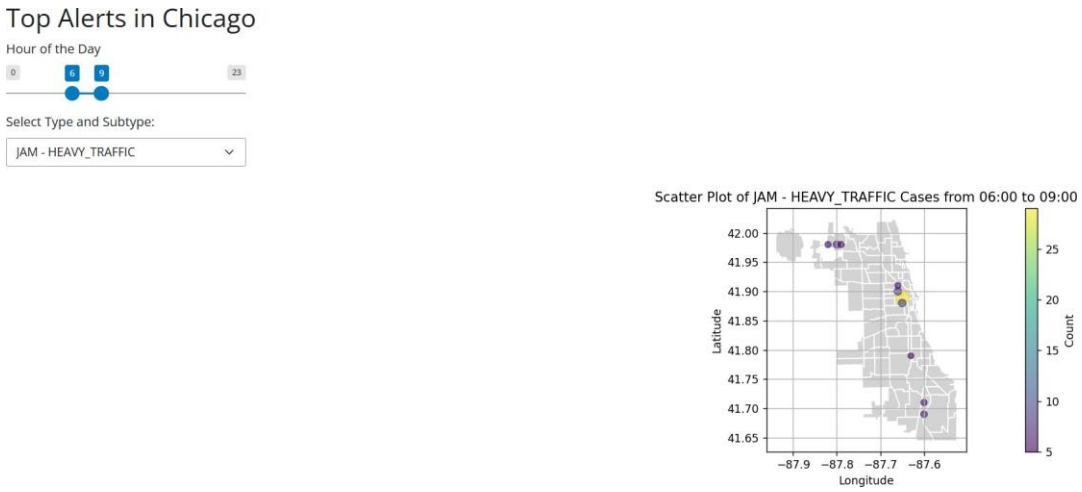
2.

a. App with hour range slider and plot:



App_Range_Plot

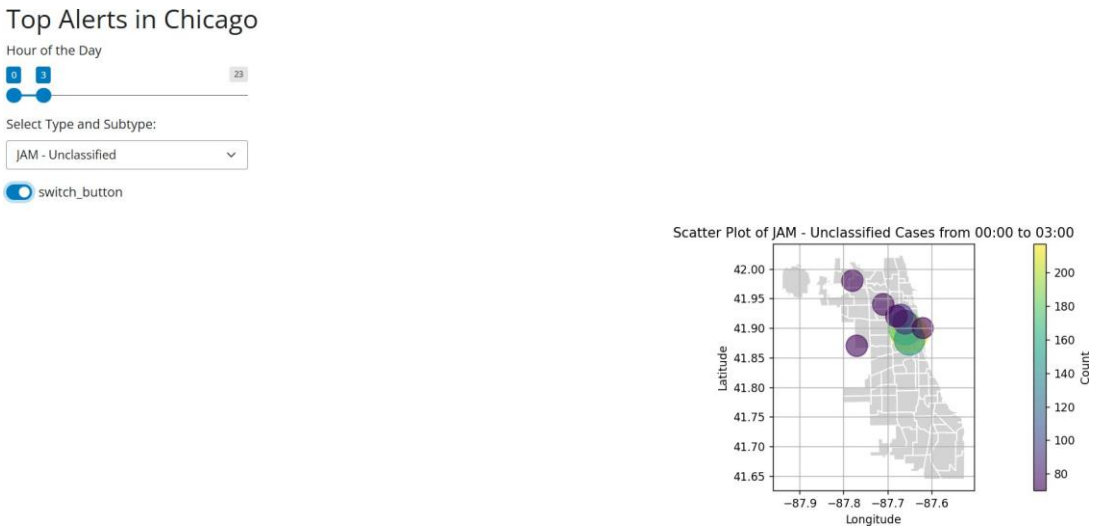
b. App with hour range slider and plot for Jam-Heavy Traffic from 6-9 AM:



App_Plot_Jam_6_9

3.

a. Dashboard with switch (no functionality):



App_Switch_Button_No_Function

b. Dashboard with switch off:

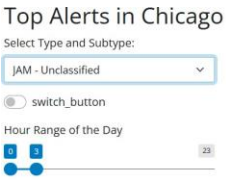
App_Switch_Button_Off

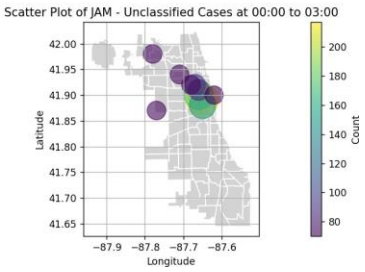
Dashboard with switch on:

App_Switch_Button_On

c.

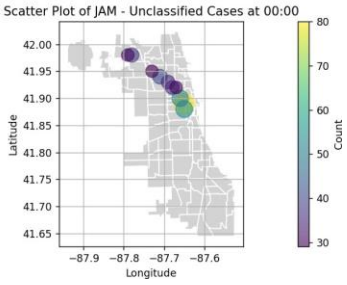
Dashboard showing plot with switch off (range of hours):





App_Switch_Button_Off

Dashboard showing plot with switch on (single hour):



App_Switch_Button_Off

BingChat Citation: How do I add a UI Conditional Panel, when ui-conditional is yielding an error that says it is not recognized?

d. I would add a column called “Time Period” that specifies that alerts that occur between 00:00 and 11:59 are in the “Morning” and 12:00 and 23:59 are in the “Afternoon”. I would also specify that the color of the dots vary on “Time Period”.