Problem Set 6 - Waze Shiny Dashboard

Hiroaki Kurachi

2024-11-23

1. **ps6:** Due Sat 23rd at 5:00PM Central. Worth 100 points (80 points from questions, 10 points for correct submission and 10 points for code style) + 10 extra credit.

We use (`*`) to indicate a problem that we think might be time consuming.

Steps to submit (10 points on PS6)

1. "This submission is my work alone and complies with the 30538 integrity policy." Add your initials to indicate your agreement: **HK**
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **HK** (2 point)
3. Late coins used this pset: **01** Late coins left after submission: **00**
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("```")
```

```
# Import required packages.
import zipfile
import os
import pandas as pd
import altair as alt
import pandas as pd
from datetime import date, time
import numpy as np
import re
import requests
import json
alt.renderers.enable("png")
alt.data_transformers.disable_max_rows()
```

```
DataTransformerRegistry.enable('default')
```

Background

Data Download and Exploration (20 points)

1.

```
# Unzip the datasets
base = (r"C:\Users\hkura\Documents\Uchicago\04 2024
 ↪  Autumn\Python2\problem-set-6-hirokurachi")
path_zip = os.path.join(
    base,
    "waze_data.zip"
)

with zipfile.ZipFile(path_zip, "r") as zip_data:
    zip_data.extractall(base)
```

```
# Load the sample dataset into a DataFrame
path_sample = os.path.join(
    base,
    "waze_data_sample.csv"
)

df_sample = pd.read_csv(path_sample)

# Summarize datatype for each columns
df_sample_dtypes = pd.DataFrame(df_sample.dtypes).reset_index()
df_sample_dtypes.columns = ["columns", "datatypes"]

# Fill the datatype column with the altair datatypes
alt_datatypes = ["Quantitative", "Nominal", "Ordinal", "Quantitative",
 ↪  "Nominal", "Nominal", "Nominal",
             "Nominal", "Nominal", "Nominal", "Ordinal", "Quantitative",
              ↪  "Ordinal", np.nan, np.nan, np.nan]
df_sample_dtypes["datatypes"] = alt_datatypes

print(df_sample_dtypes)
```

```
         columns     datatypes
0     Unnamed: 0  Quantitative
1           city       Nominal
2     confidence       Ordinal
3      nThumbsUp  Quantitative
4         street       Nominal
5           uuid       Nominal
6        country       Nominal
7           type       Nominal
```

```
8         subtype       Nominal
9        roadType       Nominal
10    reliability       Ordinal
11         magvar  Quantitative
12   reportRating       Ordinal
13             ts           NaN
14            geo           NaN
15         geoWKT           NaN
```

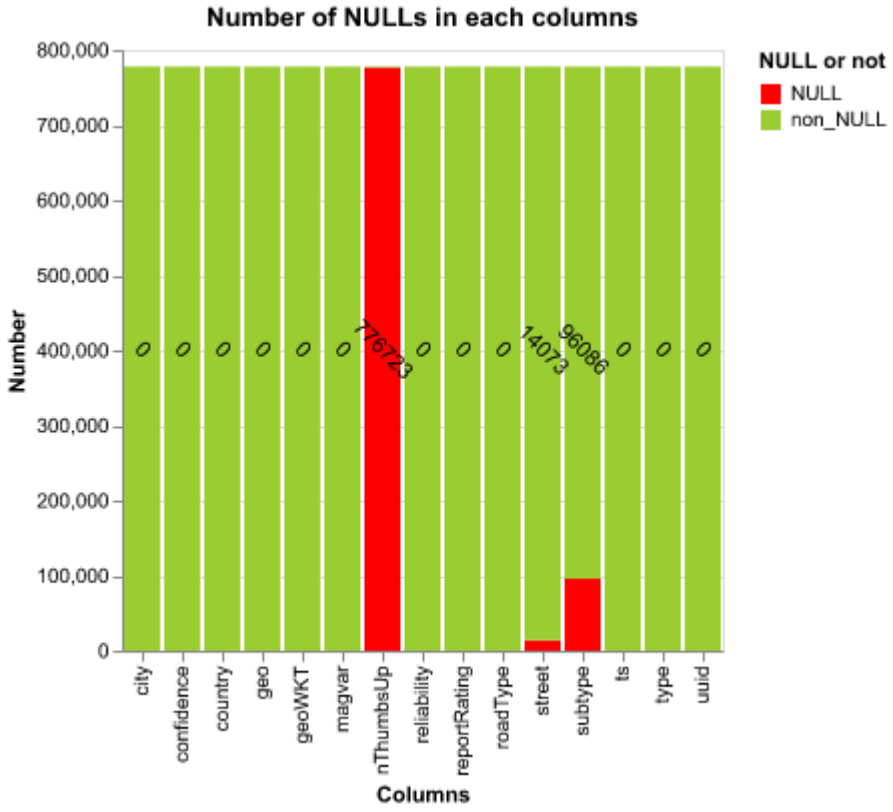
2.

```
# Load the total dataset
path_waze = os.path.join(
    base,
    "waze_data.csv"
)
df_waze = pd.read_csv(path_waze)

# Count the number of Nulls and non-Nulls in each column
null_number = [len(df_waze[df_waze[x].isna()]) for x in df_waze.columns]
non_null_number = [len(df_waze[~df_waze[x].isna()]) for x in df_waze.columns]

# Summarize the number of observations fo each columns, with categories of
 ↪  NULL/missing or not
df_nullshare_waze = pd.DataFrame(
    dict(zip(
        ["Columns", "NULL", "non_NULL"],
        [df_waze.columns, null_number, non_null_number])
    )
)

# Mutate the NULL share
df_nullshare_waze["NULL_share"] = df_nullshare_waze["NULL"] / len(df_waze)

print(df_nullshare_waze)

# Melt the df to specify category (NULL/non-NULL) and whose number for each
 ↪  columns(column names)
df_null_or_not_waze = df_nullshare_waze.melt(
    id_vars="Columns",
    var_name="NULL_or_not",
    value_name="Number"
)
```

```
# Plot a stacked bar of the number of observations for each columns, with
 ↪  categories of NULL/missing or not
df_null_or_not_waze["bool_Null_or_not"] =
 ↪  df_null_or_not_waze["NULL_or_not"].map(
    {"NULL": -1, "non_NULL": 1}
)
chart_null = alt.Chart(df_null_or_not_waze).mark_bar().encode(
    alt.X("Columns:N"),
    alt.Y("Number:Q"),
    alt.Color("NULL_or_not:N",
              legend=alt.Legend(title="NULL or not"),
              scale=alt.Scale(
                  domain=["NULL", "non_NULL"],
                  range=["red", "yellowgreen"]
              )),
    alt.Order("bool_Null_or_not", sort="ascending")
).properties(
    title="Number of NULLs in each columns",
    height=300,
    width=300
)

# Add the numbers of NULL as texts
text_null = alt.Chart(df_null_or_not_waze).mark_text(
    angle=45
).encode(
    x="Columns:N",
    text="Number:N"
).properties(
    height=300,
    width=300
).transform_filter(
    "datum.NULL_or_not == 'NULL'"
)

# Integrate the bar chart and the text
chart_null = chart_null + text_null

chart_null.show()
```

```
         Columns    NULL  non_NULL  NULL_share
```

0	city	0	778094	0.000000
1	confidence	0	778094	0.000000
2	nThumbsUp	776723	1371	0.998238
3	street	14073	764021	0.018087
4	uuid	0	778094	0.000000
5	country	0	778094	0.000000
6	type	0	778094	0.000000
7	subtype	96086	682008	0.123489
8	roadType	0	778094	0.000000
9	reliability	0	778094	0.000000
10	magvar	0	778094	0.000000
11	reportRating	0	778094	0.000000
12	ts	0	778094	0.000000
13	geo	0	778094	0.000000
14	geoWKT	0	778094	0.000000



From the result above, the variables which have the NULL is “nThumbsUp”, “street” and “subtype”, with the highest share of 99.8% for “nThumbsUp”.

3.

a.

```
# Extract Unique types and subtypes
type_unique = df_waze["type"].unique().tolist()
subtype_unique = df_waze["subtype"].unique().tolist()

print(f"type: {type_unique}")
print(f"subtype: {subtype_unique}")
```

```
type: ['JAM', 'ACCIDENT', 'ROAD_CLOSED', 'HAZARD']
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']
```

```
# Find out types which have a subtype that is NA


def subtype_isna(type):
    """Check whether a type include NA subtype"""
    df_waze_bytype = df_waze[df_waze["type"] == type]
    subtype_unique_bytype = df_waze_bytype["subtype"].unique().tolist()
    if np.nan in subtype_unique_bytype:
        return True
    else:
        return False


type_include_nan = [type for type in type_unique if subtype_isna(type)]
print(f"types which includes subtype nan: {type_include_nan}")
```

```
types which includes subtype nan: ['JAM', 'ACCIDENT', 'ROAD_CLOSED',
'HAZARD']
```

All of four types have a subtype that is NA.

Then, we would find out whether the subtypes in each type are informative for their sub-subtypes, by extracting subtype names (including sub-subtype names or not) removing type names from them (This time, we would exclude nan from the checking process).

```
# Summarize the hierarchy of types and subtypes (not including nan in
 ↪  subtype)
df_hierarchy = df_waze.groupby(
    ["type", "subtype"]
).size().reset_index().rename(
    columns={0: "count"}
)

# Define general function to remove parent name from children name in the
 ↪  categorical hierarchy



def remove_head(head, base):
    """Remove head strings from base strings"""
    result = base.replace(f"{head}_", "")
    return result


# Remove type name from subtype names
df_hierarchy["subtype"] = [remove_head(h, b) for h, b in zip(
    df_hierarchy["type"], df_hierarchy["subtype"]
)]

print(df_hierarchy)
```

```
          type                       subtype   count
0      ACCIDENT                         MAJOR    6669
1      ACCIDENT                         MINOR    2509
2        HAZARD                       ON_ROAD   34069
3        HAZARD          ON_ROAD_CAR_STOPPED    5482
4        HAZARD         ON_ROAD_CONSTRUCTION   32094
5        HAZARD    ON_ROAD_EMERGENCY_VEHICLE    8360
6        HAZARD                   ON_ROAD_ICE     234
7        HAZARD          ON_ROAD_LANE_CLOSED     541
8        HAZARD               ON_ROAD_OBJECT   16050
9        HAZARD             ON_ROAD_POT_HOLE   28268
```

10	HAZARD	ON_ROAD_ROAD_KILL	65
11	HAZARD	ON_ROAD_TRAFFIC_LIGHT_FAULT	4874
12	HAZARD	ON_SHOULDER	40
13	HAZARD	ON_SHOULDER_ANIMALS	115
14	HAZARD	ON_SHOULDER_CAR_STOPPED	176751
15	HAZARD	ON_SHOULDER_MISSING_SIGN	76
16	HAZARD	WEATHER	2146
17	HAZARD	WEATHER_FLOOD	2844
18	HAZARD	WEATHER_FOG	697
19	HAZARD	WEATHER_HAIL	7
20	HAZARD	WEATHER_HEAVY_SNOW	138
21	JAM	HEAVY_TRAFFIC	170442
22	JAM	LIGHT_TRAFFIC	5
23	JAM	MODERATE_TRAFFIC	4617
24	JAM	STAND_STILL_TRAFFIC	142380
25	ROAD_CLOSED	CONSTRUCTION	129
26	ROAD_CLOSED	EVENT	42393
27	ROAD_CLOSED	HAZARD	13

The result above implies that types "HAZARD" would include sub-subtypes as well as subtypes, because the name of subtypes has some common head strings such as "ON_ROAD" or "WEATHER" - which would be the isolated subtype names, while the remainings are the isolated sub-subtype names.

On the other hand, other 3 types doesn't have such characteristics in their subtype names, implying not having sub-subtypes.

b.

Now, the hierarchy of types are like below, excluding subtype of nan in each type:

- Accident
  - Major
  - Minor
- Hazard
  - On Road
    - (no sub-subtype)
    - Car stopped
    - Construction
    - Emergency vehicle

        * Ice
        * Lane closed
        * Object
        * Pot hole
        * Road kill
        * Traffic light fault
    - On Shoulder
        * (no sub-subtype)
        * Animals
        * Car stopped
        * Missing sign
    - Weather
        * (no sub-subtype)
        * Flood
        * Fog
        * Hail
        * Heavy snow
- Jam
    - Heavy traffic
    - Light traffic
    - Moderate traffic
    - Stand still traffic
- Road Closed
    - Construction
    - Event
    - Hazard

c.

We need to keep the NA subtypes, because they count up to 96086 observations with up to about 12.3% share among whole observations, suggesting that completely dropping these rows would lose significant amount of samples from the population, with risks of affecting our statistical analysis severely and significantly. On contrary, by explicitly saying them "Unclassified", we can recognize them without confusion. And it is still useful to grasp how the dataset successfully includes incomplete data as far as there are no problem in dealing with the data.

4.

a.

```
# Create base df for crosswalk
df_crosswalk = df_waze.copy()

# Summarize the df so that it has rows for each set of type and subtype
df_crosswalk = df_crosswalk.groupby(
    ["type", "subtype"],
    dropna=False
).size().reset_index()

df_crosswalk = df_crosswalk.rename(
    columns={0: "count"}
).drop("count", axis=1)

# Create "updated_" columns with temporal base values
df_crosswalk["updated_type"] = df_crosswalk["type"]
df_crosswalk["updated_subtype"] = df_crosswalk["subtype"]
df_crosswalk["updated_subsubtype"] = df_crosswalk["subtype"]

print(df_crosswalk.head(3))
```

```
       type         subtype updated_type updated_subtype updated_subsubtype
0  ACCIDENT  ACCIDENT_MAJOR     ACCIDENT  ACCIDENT_MAJOR     ACCIDENT_MAJOR
1  ACCIDENT  ACCIDENT_MINOR     ACCIDENT  ACCIDENT_MINOR     ACCIDENT_MINOR
2  ACCIDENT            NaN     ACCIDENT             NaN                NaN
```

b.

```
# Convert nan in "updated_subtype" and "updated_subsubtype" into
 ↪  "Unclassified" to make them recognized as strings in following process
df_crosswalk["updated_subtype"] = df_crosswalk["updated_subtype"].fillna(
    "Unclassified")
```

```
df_crosswalk["updated_subsubtype"] = df_crosswalk["updated_subtype"].fillna(
    "Unclassified")

# Remove type name from "updated_subsubtype" to get subtype + sub-subtype
df_crosswalk["updated_subsubtype"] = [remove_head(h, b) for h, b in zip(
    df_crosswalk["type"], df_crosswalk["updated_subsubtype"])]

# Isolate subtype names in "updated_subtype"


def extract_subtype(type, subtype):
    """Extract isolated subtype name"""
    # For types without sub-subtypes, just remove type name
    if type in ["ACCIDENT", "JAM", "ROAD_CLOSED"]:
        return subtype.replace(f"{type}_", "")
    # For type "Hazard", check which isolated subtype name is included
    elif type == "HAZARD":
        if "ON_ROAD" in subtype:
            return "ON_ROAD"
        elif "ON_SHOULDER" in subtype:
            return "ON_SHOULDER"
        elif "WEATHER" in subtype:
            return "WEATHER"
        elif subtype == "Unclassified":
            return subtype


df_crosswalk["updated_subtype"] = [extract_subtype(t, s) for t, s in zip(
    df_crosswalk["type"], df_crosswalk["updated_subtype"])]

# Remove isolated subtype name from "updated_subsubtype" to get isolated
 ↪  sub-subtype
df_crosswalk["updated_subsubtype"] = [remove_head(h, b) for h, b in zip(
    df_crosswalk["updated_subtype"], df_crosswalk["updated_subsubtype"])]

# Change sub-subtypes to "Unclassified" if they are the same as the
 ↪  "updated_subtype" (this condition include the case where the subtype is
 ↪  "unclassified")
df_crosswalk.loc[df_crosswalk["updated_subsubtype"] ==
                 df_crosswalk["updated_subtype"], "updated_subsubtype"] =
 ↪  "unclassified"
```

```
# Update the "updated_" columns to a readable format
df_crosswalk.iloc[:, 2:] = df_crosswalk.iloc[:, 2:].map(
    lambda x: x.replace("_", " ").capitalize())

print(df_crosswalk)
```

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

       updated_subtype   updated_subsubtype
0                Major         Unclassified
```

```
1                 Minor         Unclassified
2          Unclassified         Unclassified
3               On road         Unclassified
4               On road          Car stopped
5               On road         Construction
6               On road    Emergency vehicle
7               On road                  Ice
8               On road          Lane closed
9               On road               Object
10              On road             Pot hole
11              On road            Road kill
12              On road  Traffic light fault
13           On shoulder         Unclassified
14           On shoulder              Animals
15           On shoulder          Car stopped
16           On shoulder         Missing sign
17              Weather         Unclassified
18              Weather                Flood
19              Weather                  Fog
20              Weather                 Hail
21              Weather           Heavy snow
22         Unclassified         Unclassified
23        Heavy traffic         Unclassified
24        Light traffic         Unclassified
25     Moderate traffic         Unclassified
26  Stand still traffic         Unclassified
27         Unclassified         Unclassified
28         Construction         Unclassified
29                Event         Unclassified
30               Hazard         Unclassified
31         Unclassified         Unclassified
```

c.

```
# Merge the dfs
df_waze_updated = df_waze.merge(
    df_crosswalk,
    how="inner",
    on=["type", "subtype"]
)

# Count the rows for Accident - Unclassified
num_Accdnt_Unclssfd = len(df_waze_updated[
```

```
    (df_waze_updated["updated_type"] == "Accident") & (
        df_waze_updated["updated_subtype"] == "Unclassified")
])
print(
    f"The number of rows for Accident - Unclassified is
     ↪  {num_Accdnt_Unclssfd}."
)
```

```
The number of rows for Accident - Unclassified is 24359.
```

d.

```
# If the crosswalk erroneously contained type or subtype which were different
 ↪  from those in df_waze, the length of df_waze_updated would be longer than
 ↪  that of df_waze
assert(len(df_waze) == len(df_waze_updated)), "There are row(s) with
 ↪  different value(s) between two df"

print(df_waze_updated.head(1))
```

```
          city  confidence  nThumbsUp street  \
0  Chicago, IL           0        NaN    NaN

                                  uuid country type subtype  roadType  \
0  004025a4-5f14-4cb7-9da6-2615daafbf37      US  JAM     NaN        20

   reliability  magvar  reportRating                       ts  \
0            5     139             3  2024-02-04 16:40:41 UTC

                           geo                       geoWKT updated_type  \
0  POINT(-87.676685 41.929692)  Point(-87.676685 41.929692)          Jam

  updated_subtype updated_subsubtype
0    Unclassified       Unclassified
```

As this code runs without error meessage, we can assure that the crosswalk and the new merged dataset have the same values in type and subttype.

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

1.

a.

```
# Split the strings column "geo" at every white-space character
series_geo_lists = df_waze_updated["geo"].map(lambda x: re.split(r"\s", x))

# Convert the series of lists into df
df_coordinate = pd.DataFrame(series_geo_lists.tolist(), columns=[
                      "longitude", "latitude"])

# Remove redundant strings from the columns
df_coordinate["longitude"] = df_coordinate["longitude"].map(
    lambda x: float(x.replace("POINT(", "")))
df_coordinate["latitude"] = df_coordinate["latitude"].map(
    lambda x: float(x.replace(")", "")))

# Join the df into the base df
df_waze_updated = df_waze_updated.join(df_coordinate)

print(df_waze_updated[["longitude", "latitude"]].head(3))
```

```
   longitude   latitude
0 -87.676685  41.929692
1 -87.624816  41.753358
2 -87.614122  41.889821
```

b.

```
# Bin the longitude and latitude variables
df_waze_updated[["longitude", "latitude"]] = df_waze_updated[[
    "longitude", "latitude"]].map(lambda x: round(x, 2))

# Count each group by latitude and longitude
df_coordinate_count = df_waze_updated.groupby(
    ["longitude", "latitude"]
).size().reset_index().rename(
    columns={0: "count"}
).sort_values(
    "count",
    ascending=False
)

print(df_coordinate_count.head(1))
```

```
     longitude  latitude  count
492     -87.65     41.88  21325
```

The set of binned latitude-longitude combination above has the greatest number of observations in the overall dataset.

c.

```
# Summarize the df by type, subtype, latitude and longitude
df_top_alerts_map = df_waze_updated.groupby(
    ["updated_type", "updated_subtype", "longitude", "latitude"]
).size().reset_index().rename(
    columns={0: "count"}
)

# Group and filter top 10 from the sorted df by type and subtype
df_top_alerts_map = df_top_alerts_map.sort_values(
    "count",
    ascending=False
).groupby(
    ["updated_type", "updated_subtype"]
).head(10).reset_index(drop=True)

df_top_alerts_map.to_csv(r"top_alerts_map\top_alerts_map.csv")
```

The level of aggregation is (sets of) type and subtype.

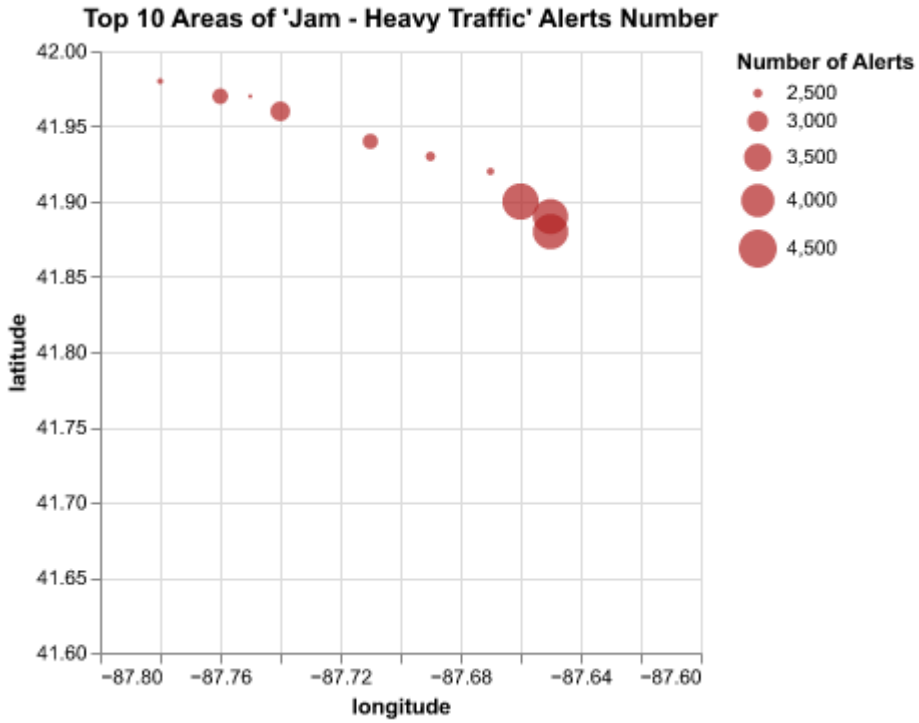
```
print(f"And the DataFrame has {len(df_top_alerts_map)} rows.")
```

```
And the DataFrame has 155 rows.
```

2.

```
# Plot the alerts by latitude and longitude for "Jam-Heavy Traffic" cases
chart_alerts = alt.Chart(df_top_alerts_map).mark_point(
    color="firebrick",
    filled=True
).encode(
    alt.X(
        "longitude:Q",
        scale=alt.Scale(
            domain=[-87.80, -87.60]
```

```
        )
    ),
    alt.Y(
        "latitude:Q",
        scale=alt.Scale(
            domain=[41.60, 42.00]
        )
    ),
    alt.Size(
        "count:Q",
        scale=alt.Scale(
            domain=[2400, 4500]
        ),
        legend=alt.Legend(
            title="Number of Alerts"
        )
    )
).properties(
    title="Top 10 Areas of 'Jam - Heavy Traffic' Alerts Number",
    height=300,
    width=300
).transform_filter(
    "datum.updated_type == 'Jam' & datum.updated_subtype == 'Heavy traffic'"
)

chart_alerts.show()
```



3.

a.

```
# Download and save from the url
url_geojson =
 ↪  "https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON"

response = requests.get(url_geojson)
data = response.json()

# Save as a geojson file
path_json = os.path.join(
    base,
    r"top_alerts_map\chicago-boundaries.geojson"
)

with open(path_json, "w") as f:
    json.dump(data, f)
```

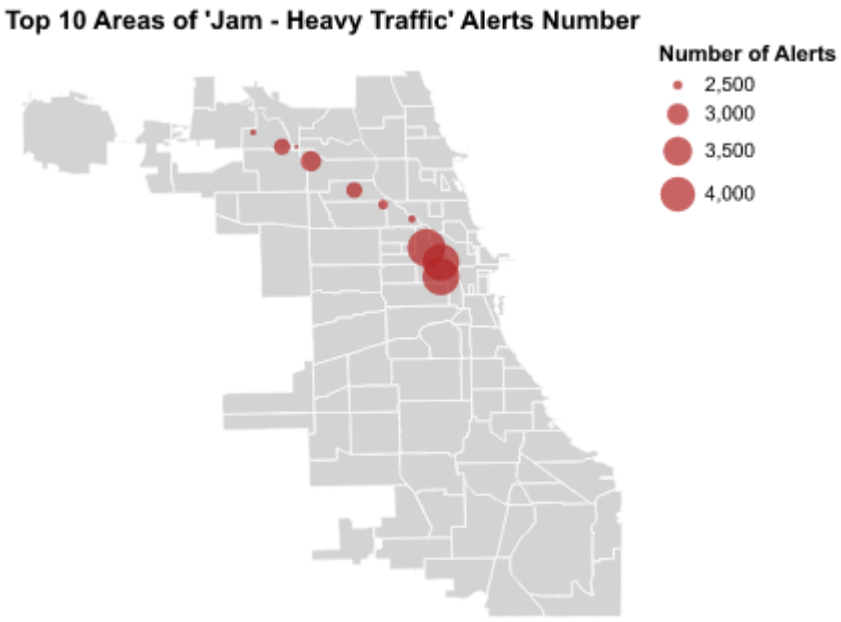
b.

```
# Load the geojson file
with open(path_json) as f:
    chicago_geojson = json.load(f)

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

4.

```
# Create subset
df_top_alerts_map_chosen = df_top_alerts_map[(
    df_top_alerts_map["updated_type"] == "Jam"
) & (
    df_top_alerts_map["updated_subtype"] == "Heavy traffic"
)]

# Set appropriate domain for chosen type and subtype
domain_1 = [
    df_top_alerts_map_chosen["count"].min(),
    df_top_alerts_map_chosen["count"].max()
]

# Redefine the scatterplot
chart_alerts = alt.Chart(df_top_alerts_map_chosen).mark_point(
    color="firebrick",
    filled=True
).encode(
    longitude="longitude:Q",
    latitude="latitude:Q",
    size=alt.Size(
        "count:Q",
        scale=alt.Scale(
            domain=domain_1
        ),
        legend=alt.Legend(
            title="Number of Alerts"
        )
    )
).properties(
    title="Top 10 Areas of 'Jam - Heavy Traffic' Alerts Number",
    height=300,
    width=300
).transform_filter(
```

```
    "datum.updated_type == 'Jam' & datum.updated_subtype == 'Heavy traffic'"
)

# Plot map
chart_map = alt.Chart(geo_data).mark_geoshape(
    fill="lightgray",
    stroke="white"
).project(
    type="equirectangular"
)


# Overlay the plots
chart_alerts_map = chart_map + chart_alerts

chart_alerts_map.show()
```



5.

a.

```
print_file_contents("./top_alerts_map/app.py")
```

```python
from shiny import App, render, ui, reactive
from shinywidgets import render_altair, output_widget
import pandas as pd
import altair as alt
import json

app_ui = ui.page_fluid(
    ui.input_select(
        id="type_subtype",
        label="Select Type and Subtype",
        choices=[]
    ),
    output_widget("chart_alerts_map")
)


def server(input, output, session):
    # Load and store waze data
    @reactive.calc
    def df_top_alerts_map():
        """Create base df"""
        df = pd.read_csv("top_alerts_map.csv")
        return df

    # Create choices for selector
    @reactive.calc
    def df_choices():
        """Summarize sets of type and subtype"""
        df = df_top_alerts_map().groupby(
            ["updated_type", "updated_subtype"]
        ).size().reset_index()
        return df

    @reactive.effect
    def _():
        """Define type-subtype choices"""
        choices = [f"{t} - {s}" for t, s in zip(
            df_choices()["updated_type"],
            df_choices()["updated_subtype"]
        )]
        ui.update_select("type_subtype", choices=choices)
```

```
# Save input from the selecter
@reactive.calc
def type_chosen():
    """Extract chosen type from selecter input"""
    if input.type_subtype() == None:
        result = "Accident"  # Set default for loading time
    # Might be categorized as "Hazard" type erroneously by the criterion
    below
    elif input.type_subtype() == "Road closed - Hazard":
        result = "Road closed"
    else:
        for type in df_choices()["updated_type"].unique():
            if type in input.type_subtype():
                result = type
                break
    return result

@reactive.calc
def subtype_chosen():
    """Extract chosen subtype from selecter input"""
    if input.type_subtype() == None:
        result = "Major"  # Set default for loading time
    else:
        result = input.type_subtype().replace(
            " - ", ""
        ).replace(
            type_chosen(), ""
        )
    return result

# Create output from the input
@reactive.calc
def df_chosen():
    """Create subset of waze df"""
    df = df_top_alerts_map()[(
        df_top_alerts_map()["updated_type"] == type_chosen()
    ) & (
        df_top_alerts_map()["updated_subtype"] == subtype_chosen()
    )]
    return df

@reactive.calc
def domain():
```

```
    """Set appropriate domain for chosen type and subtype"""
    domain = [min(df_chosen()["count"]), max(df_chosen()["count"])]
    return domain

@reactive.calc
def chart_alerts():
    """Create scatter plot for number of alert"""
    chart = alt.Chart(df_chosen()).mark_point(
        color="firebrick",
        filled=True
    ).encode(
        longitude="longitude:Q",
        latitude="latitude:Q",
        size=alt.Size(
            "count:Q",
            scale=alt.Scale(
                domain=domain()
            ),
            legend=alt.Legend(
                title="Number of Alerts"
            )
        )
    ).properties(
        title=f"Top 10 Areas of '{input.type_subtype()}' Alerts Number",
        height=300,
        width=300
    )
    return chart

# Create map
@reactive.calc
def geo_data():
    """Load and store geojson"""
    with open("chicago-boundaries.geojson") as f:
        chicago_geojson = json.load(f)
    geo_data = alt.Data(values=chicago_geojson["features"])
    return geo_data

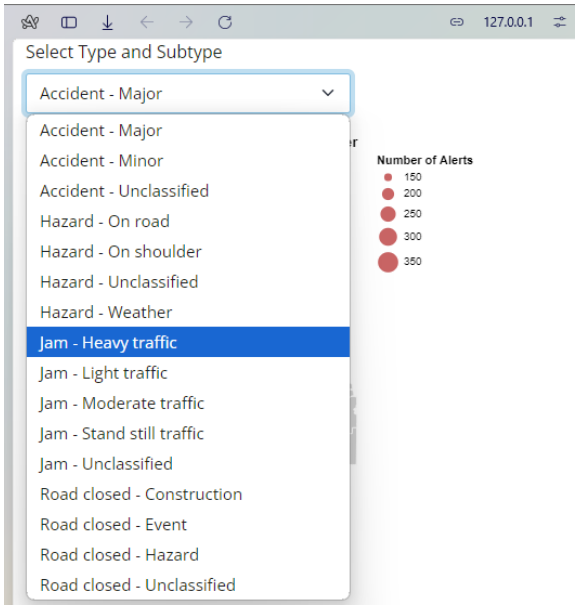
@reactive.calc
def chart_map():
    """Create the map"""
    chart = alt.Chart(geo_data()).mark_geoshape(
        fill="lightgray",
```

```
            stroke="white"
        ).project(
            type="equirectangular"
        )
        return chart

    # Create plot for output_widget
    @render_altair
    def chart_alerts_map():
        """Overlay the plots"""
        return chart_map() + chart_alerts()

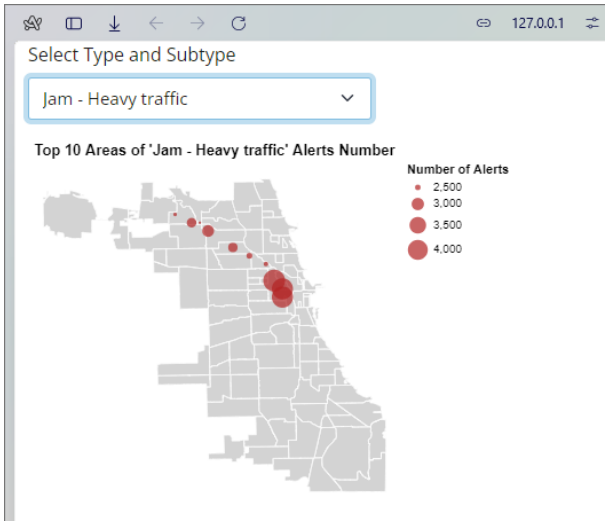

app = App(app_ui, server)

```

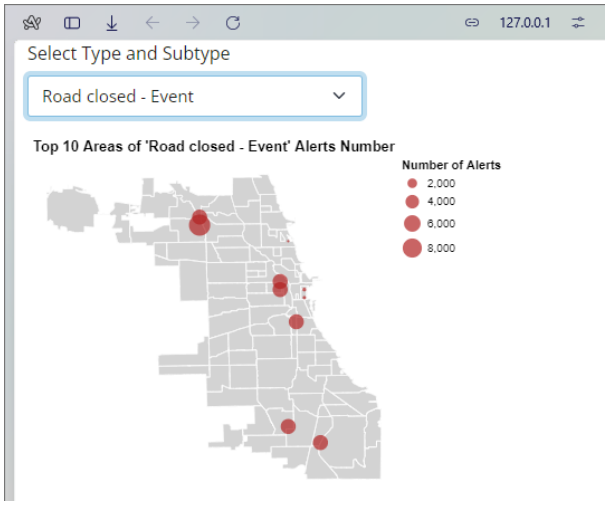


There are 16 total combinations.

b.



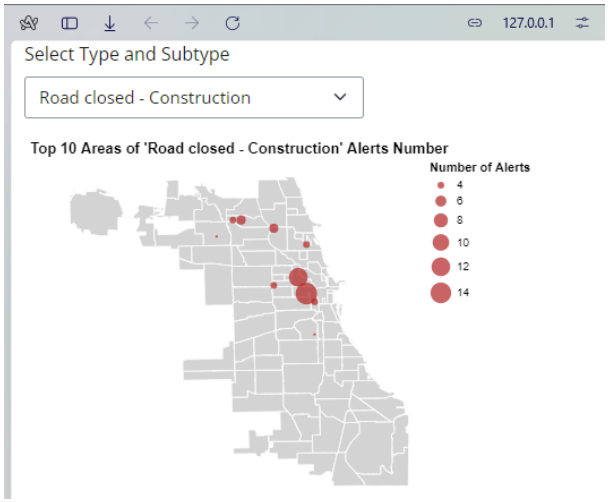
c.



North east part of Portage Park neighborhood, which locate on north west part of Chicago.

d.

Question: in which area road closure due to construction most commonly observed?



Answer: West from loop, where large interstates (e.g. I-90/94) is running, which might be under maintenance or rehabilitation.

e.

One idea is adding "roadType" column to the dashboard, as tooltip for each point. It would suggest in which road in the area the alerts happen, whether primary street or secondary/(standard) street etc, which is useful in answering the previous question I formulated above. The dataset dictionary suggests that the road types include even railroads or pedestrian, so it would be better to include this information to clarify the ratio of road types for each point in the dashboard.

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

1.

a.

Collapsing the dataset by the column *ts* would be not a good idea. We of course need to subset the dataset based on the information in *ts*, but we need to extract only hours from each value in the column, removing redundant information, such as year-month-date(irrelevant in our analysis), or minute and seconds (too detailed for input in UI, not requested by users), or "UTC" (All data in the column are UTC in this case).

b.

```
# Create "hour" column
df_waze_updated["hour"] = pd.to_datetime(
    df_waze_updated["ts"].replace("UTC", "")).dt.strftime("%H:00")

# Summarize the df by type, subtype, latitude, longitude and hour
df_top_alerts_map_byhour = df_waze_updated.groupby(
    ["updated_type", "updated_subtype", "longitude", "latitude", "hour"]
).size().reset_index().rename(
    columns={0: "count"}
)

# Group and filter top 10 from the sorted df by hour
df_top_alerts_map_byhour = df_top_alerts_map_byhour.sort_values(
    "count",
    ascending=False
).groupby(
    ["updated_type", "updated_subtype", "hour"]
).head(10).reset_index(drop=True)

df_top_alerts_map_byhour.to_csv(
    r"top_alerts_map_byhour\top_alerts_map_byhour.csv")
```

The level of aggregation is sets of type and subtype, and additionally, hour in this case.

```
print(f"And the DataFrame has {len(df_top_alerts_map_byhour)} rows.")
```

```
And the DataFrame has 3202 rows.
```

c.

```
# Create subset
df_top_alerts_map_byhour_chosen = df_top_alerts_map_byhour[(
    df_top_alerts_map_byhour["updated_type"] == "Jam"
) & (
    df_top_alerts_map_byhour["updated_subtype"] == "Heavy traffic"
) & (
    df_top_alerts_map_byhour["hour"].isin(["10:00", "16:00", "23:00"])
)]

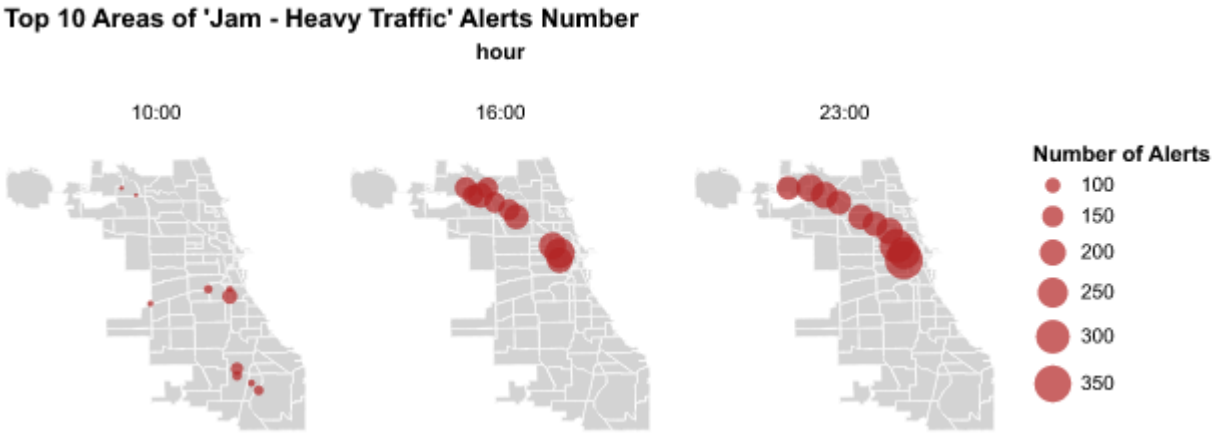
# Set appropriate domain for chosen type and subtype
domain_2 = [
```

```
    df_top_alerts_map_byhour_chosen["count"].min(),
    df_top_alerts_map_byhour_chosen["count"].max()
]

# Plot the alerts by longitude and latitude for "Jam-Heavy Traffic" cases,
 ↪  for 3 timings
chart_alerts = alt.Chart().mark_point(
    color="firebrick",
    filled=True
).encode(
    longitude="longitude:Q",
    latitude="latitude:Q",
    size=alt.Size(
        "count:Q",
        scale=alt.Scale(
            domain=domain_2
        ),
        legend=alt.Legend(
            title="Number of Alerts"
        ))
).properties(
    height=150,
    width=150
)

chart_alerts_map_byhour = alt.layer(chart_map, chart_alerts).facet(
    data=df_top_alerts_map_byhour_chosen,
    column="hour:N"
).properties(
    title="Top 10 Areas of 'Jam - Heavy Traffic' Alerts Number"
)

chart_alerts_map_byhour.show()
```



2.

a.

```
print_file_contents("./top_alerts_map_byhour/app.py")
```

```
```python
from shiny import App, render, ui, reactive
from shinywidgets import render_altair, output_widget
import pandas as pd
import altair as alt
import json

app_ui = ui.page_fluid(
    ui.input_select(
        id="type_subtype",
        label="Select Type and Subtype",
        choices=[]
    ),
    ui.input_slider("hour_chosen", "Pick hour", 0, 23, 0),
    output_widget("chart_alerts_map_byhour")
)


def server(input, output, session):
    # Load and store waze data
    @reactive.calc
    def df_top_alerts_map_byhour():
        """Create base df"""
```
```

```
    df = pd.read_csv("top_alerts_map_byhour.csv")
    return df

# Create choices for selector
@reactive.calc
def df_choices():
    """Summarize sets of type and subtype"""
    df = df_top_alerts_map_byhour().groupby(
        ["updated_type", "updated_subtype"]
    ).size().reset_index()
    return df

@reactive.effect
def _():
    """Define type-subtype choices"""
    choices = [f"{t} - {s}" for t, s in zip(
        df_choices()["updated_type"],
        df_choices()["updated_subtype"]
    )]
    ui.update_select("type_subtype", choices=choices)

# Save inputs from UI side
@reactive.calc
def type_chosen():
    """Extract chosen type from selecter input"""
    if input.type_subtype() == None:
        result = "Accident"  # Set default for loading time
    # Might be categorized as "Hazard" type erroneously by the criterion
    below
    elif input.type_subtype() == "Road closed - Hazard":
        result = "Road closed"
    else:
        for type in df_choices()["updated_type"].unique():
            if type in input.type_subtype():
                result = type
                break
    return result

@reactive.calc
def subtype_chosen():
    """Extract chosen subtype from selecter input"""
    if input.type_subtype() == None:
        result = "Major"  # Set default for loading time
```

```
    else:
        result = input.type_subtype().replace(
            " - ", ""
        ).replace(
            type_chosen(), ""
        )
    return result

@reactive.calc
def hour_chosen():
    """Extract chosen hour from slider input"""
    result = f"{input.hour_chosen()}:00"
    return result

# Create output from the inputs
@reactive.calc
def df_chosen():
    """Create subset of waze df"""
    df = df_top_alerts_map_byhour()[(
        df_top_alerts_map_byhour()["updated_type"] == type_chosen()
    ) & (
        df_top_alerts_map_byhour()["updated_subtype"] == subtype_chosen()
    ) & (
        df_top_alerts_map_byhour()["hour"] == hour_chosen()
    )]
    return df

@reactive.calc
def domain():
    """Set appropriate domain for chosen type and subtype"""
    domain = [min(df_chosen()["count"]), max(df_chosen()["count"])]
    return domain

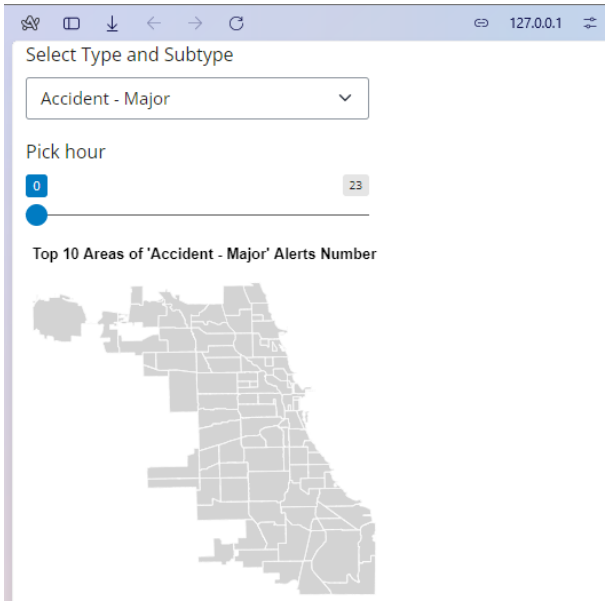
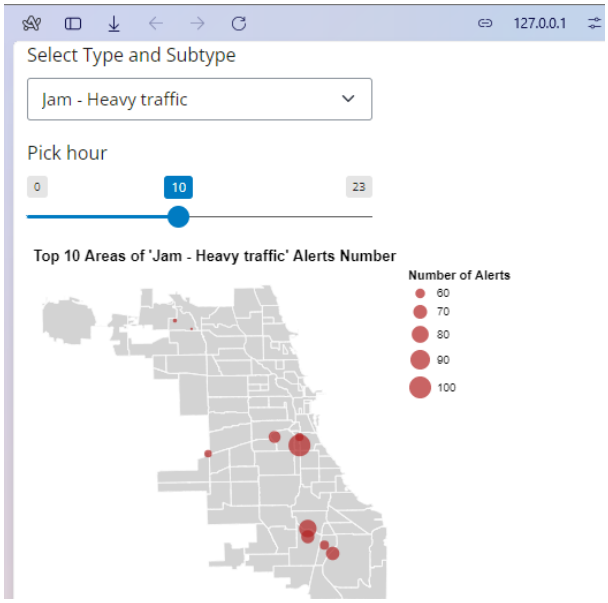
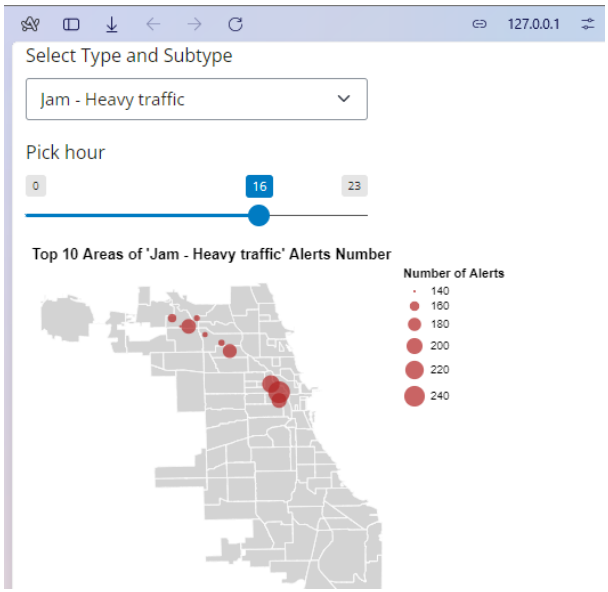
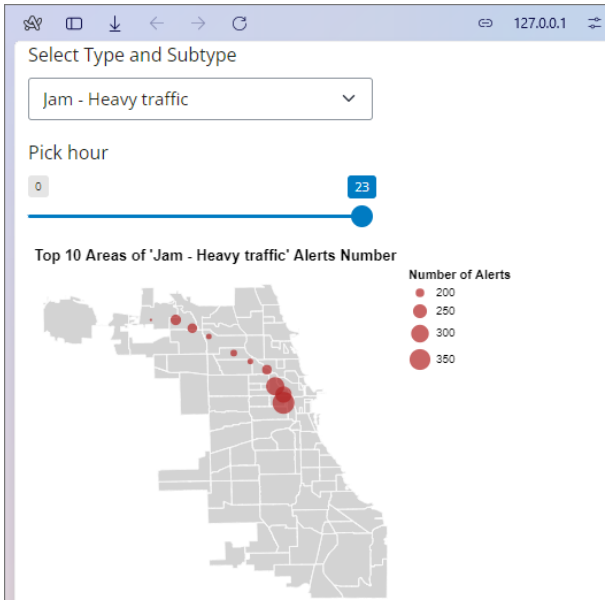
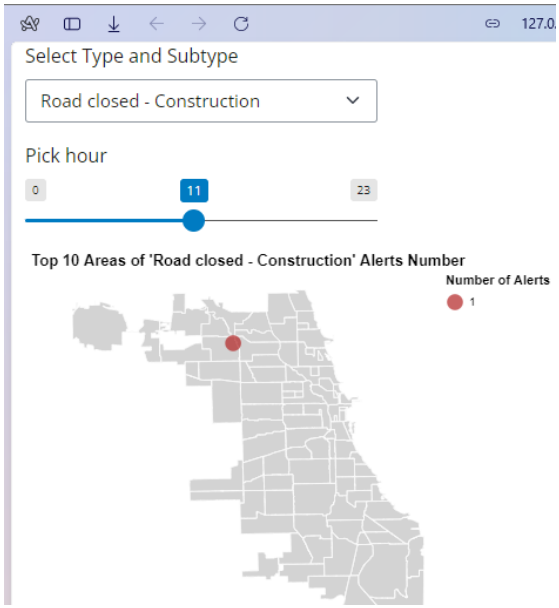
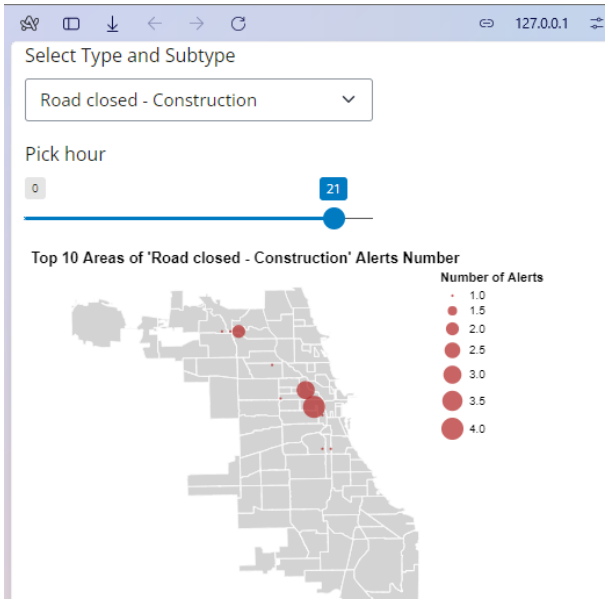
@reactive.calc
def chart_alerts_byhour():
    """Create scatter plot for number of alert"""
    chart = alt.Chart(df_chosen()).mark_point(
        color="firebrick",
        filled=True
    ).encode(
        longitude="longitude:Q",
        latitude="latitude:Q",
        size=alt.Size(
```

```
            "count:Q",
            scale=alt.Scale(
                domain=domain()
            ),
            legend=alt.Legend(
                title="Number of Alerts"
            )
        )
    ).properties(
        title=f"Top 10 Areas of '{input.type_subtype()}' Alerts Number",
        height=300,
        width=300
    )
    return chart

# Create map
@reactive.calc
def geo_data():
    """Load and store geojson"""
    with open("chicago-boundaries.geojson") as f:
        chicago_geojson = json.load(f)
    geo_data = alt.Data(values=chicago_geojson["features"])
    return geo_data

@reactive.calc
def chart_map():
    """Create the map"""
    chart = alt.Chart(geo_data()).mark_geoshape(
        fill="lightgray",
        stroke="white"
    ).project(
        type="equirectangular"
    ).properties(
        title=f"Top 10 Areas of '{input.type_subtype()}' Alerts Number",
        height=300,
        width=300
    )
    return chart

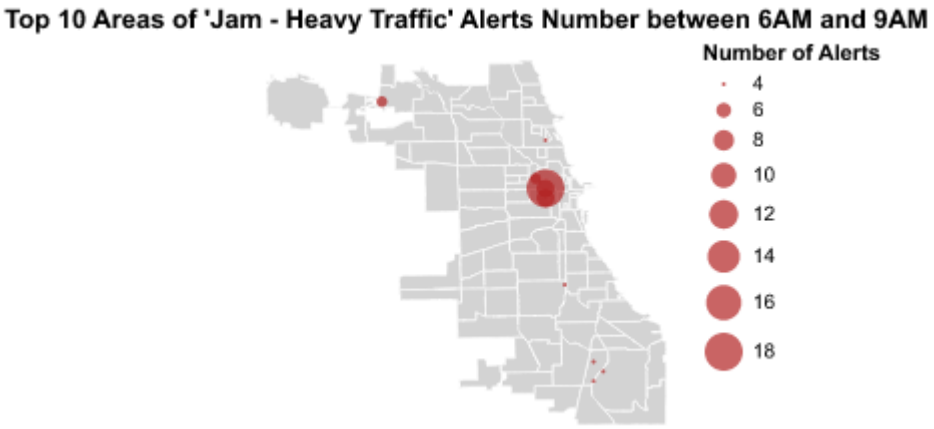
# Create plot for output_widget
@render_altair
def chart_alerts_map_byhour():
```

```
        """Overlay the plots, if there are observations which satisfy
        conditions"""
        if len(df_chosen()) == 0:
            return chart_map()
        else:
            return chart_map() + chart_alerts_byhour()


app = App(app_ui, server)

```



b.


127.0.0.1
Select Type and Subtype
Jam - Heavy traffic
Pick hour
0
10
23
Top 10 Areas of 'Jam - Heavy traffic' Alerts Number
Number of Alerts
60
70
80
90
100


127.0.0.1
Select Type and Subtype
Jam - Heavy traffic
Pick hour
0
16
23
Top 10 Areas of 'Jam - Heavy traffic' Alerts Number
Number of Alerts
140
160
180
200
220
240


127.0.0.1
Select Type and Subtype
Jam - Heavy traffic
Pick hour
0
23
Top 10 Areas of 'Jam - Heavy traffic' Alerts Number
Number of Alerts
200
250
300
350

c.


127.0.
Select Type and Subtype
Road closed - Construction
Pick hour
0
11
23
Top 10 Areas of 'Road closed - Construction' Alerts Number
Number of Alerts
1



More during night hours. There seems no alerts before 11am, while there are frequent alerts at each night time. Even if there are other factor such as the difference in the amount of traffic between morning and night, the observation above would be strong enough to support my assumption.

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

1.

a.

Collapsing the dataset by range of hours would not be a good idea. If do so, the app need to aggregate the dataset internally for each time the slider was moved by the user, which is resource intensive.

b.

```
# Create base column for calculating range of hours
df_top_alerts_map_byhour["hour_dt"] = pd.to_datetime(
    df_top_alerts_map_byhour["hour"], format="%H:%M")
df_top_alerts_map_byhour["hour_dt"] = [x.time()
                                       for x in
                                        ↪ df_top_alerts_map_byhour["hour_dt"]]
```

```
# Create subset
df_top_alerts_map_byhour_range = df_top_alerts_map_byhour[(
    df_top_alerts_map_byhour["updated_type"] == "Jam"
) & (
    df_top_alerts_map_byhour["updated_subtype"] == "Heavy traffic"
) & (
    df_top_alerts_map_byhour["hour_dt"].between(time(6, 0), time(9, 0))
)].drop("hour_dt", axis=1).head(10)

# Set appropriate domain for chosen type and subtype
domain_3 = [
    df_top_alerts_map_byhour_range["count"].min(),
    df_top_alerts_map_byhour_range["count"].max()
]

# Plot the alerts by longitude and latitude for "Jam-Heavy Traffic" cases,
 ↪  for 3 timings
chart_alerts = alt.Chart(df_top_alerts_map_byhour_range).mark_point(
    color="firebrick",
    filled=True
).encode(
    longitude="longitude:Q",
    latitude="latitude:Q",
    size=alt.Size(
        "count:Q",
        scale=alt.Scale(
            domain=domain_3
        ),
        legend=alt.Legend(
            title="Number of Alerts"
        ))
).properties(
    height=200,
    width=200
)

chart_alerts_map_byhour_range = alt.layer(chart_map,
 ↪  chart_alerts).properties(
    title="Top 10 Areas of 'Jam - Heavy Traffic' Alerts Number between 6AM
 ↪  and 9AM"
)
```

```
chart_alerts_map_byhour_range.show()
```



2.

a.

```
print_file_contents("./top_alerts_map_byhour_sliderrange/app.py")
```

```
```python
from shiny import App, render, ui, reactive
from shinywidgets import render_altair, output_widget
import pandas as pd
import altair as alt
import json
from datetime import date, time

app_ui = ui.page_fluid(
    ui.input_select(
        id="type_subtype",
        label="Select Type and Subtype",
        choices=[]
    ),
    ui.input_slider("hours_range", "Set range of hours",
                    0, 23, (0, 23), drag_range=True),
    output_widget("chart_alerts_map_byhour")
)
```
```

```
# Attribution: parameter "drag_range" refering to Shiny document
(https://shiny.posit.co/py/api/core/ui.input_slider.html)


def server(input, output, session):
    # Load and store waze data
    @reactive.calc
    def df_top_alerts_map_byhour():
        """Create base df"""
        df = pd.read_csv("top_alerts_map_byhour.csv")
        df["hour_dt"] = pd.to_datetime(df["hour"], format="%H:%M")
        df["hour_dt"] = [x.time() for x in df["hour_dt"]]
        return df
        # Attribution: parameter "format" refering to Perplexity
        (https://www.perplexity.ai/search/from-shiny-import-app-render-u-Tt7cqT5WTmeh0z2CLZtU

    # Create choices for selector
    @reactive.calc
    def df_choices():
        """Summarize sets of type and subtype"""
        df = df_top_alerts_map_byhour().groupby(
            ["updated_type", "updated_subtype"]
        ).size().reset_index()
        return df

    @reactive.effect
    def _():
        """Define type-subtype choices"""
        choices = [f"{t} - {s}" for t, s in zip(
            df_choices()["updated_type"],
            df_choices()["updated_subtype"]
        )]
        ui.update_select("type_subtype", choices=choices)

    # Save inputs from UI side
    @reactive.calc
    def type_chosen():
        """Extract chosen type from selecter input"""
        if input.type_subtype() == None:
            result = "Accident"  # Set default for loading time
        # Might be categorized as "Hazard" type erroneously by the criterion
        below
        elif input.type_subtype() == "Road closed - Hazard":
```

```
        result = "Road closed"
    else:
        for type in df_choices()["updated_type"].unique():
            if type in input.type_subtype():
                result = type
                break
    return result

@reactive.calc
def subtype_chosen():
    """Extract chosen subtype from selecter input"""
    if input.type_subtype() == None:
        result = "Major"  # Set default for loading time
    else:
        result = input.type_subtype().replace(
            " - ", ""
        ).replace(
            type_chosen(), ""
        )
    return result

@reactive.calc
def hours_range():
    """Extract chosen range of hours from slider"""
    result = list(input.hours_range())
    return result

# Create output from the inputs
@reactive.calc
def df_chosen():
    """Create subset of waze df"""
    df = df_top_alerts_map_byhour()[(
        df_top_alerts_map_byhour()["updated_type"] == type_chosen()
    ) & (
        df_top_alerts_map_byhour()["updated_subtype"] == subtype_chosen()
    ) & (
        df_top_alerts_map_byhour()["hour_dt"].between(
            time(hours_range()[0], 0), time(hours_range()[1], 0))
    )].drop("hour_dt", axis=1).head(10)
    return df

@reactive.calc
def domain():
```

```
    """Set appropriate domain for chosen type and subtype"""
    domain = [min(df_chosen()["count"]), max(df_chosen()["count"])]
    return domain

@reactive.calc
def chart_alerts_byhour():
    """Create scatter plot for number of alert"""
    chart = alt.Chart(df_chosen()).mark_point(
        color="firebrick",
        filled=True
    ).encode(
        longitude="longitude:Q",
        latitude="latitude:Q",
        size=alt.Size(
            "count:Q",
            scale=alt.Scale(
                domain=domain()
            ),
            legend=alt.Legend(
                title="Number of Alerts"
            )
        )
    ).properties(
        title=f"Top 10 Areas of '{input.type_subtype()}' Alerts Number",
        height=300,
        width=300
    )
    return chart

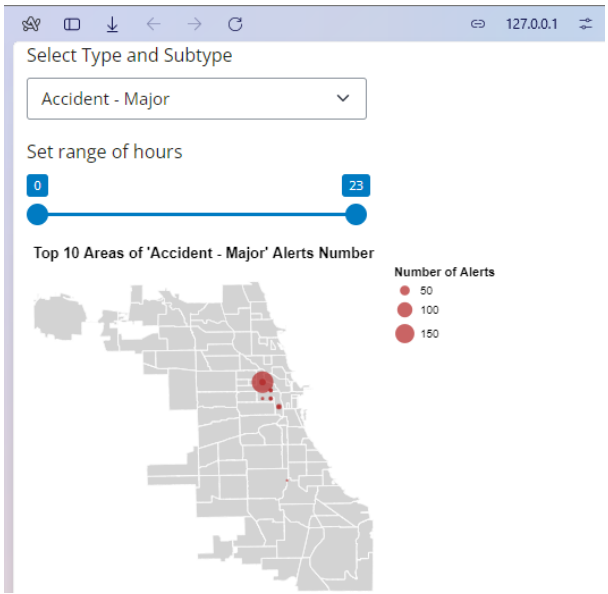
# Create map
@reactive.calc
def geo_data():
    """Load and store geojson"""
    with open("chicago-boundaries.geojson") as f:
        chicago_geojson = json.load(f)
    geo_data = alt.Data(values=chicago_geojson["features"])
    return geo_data

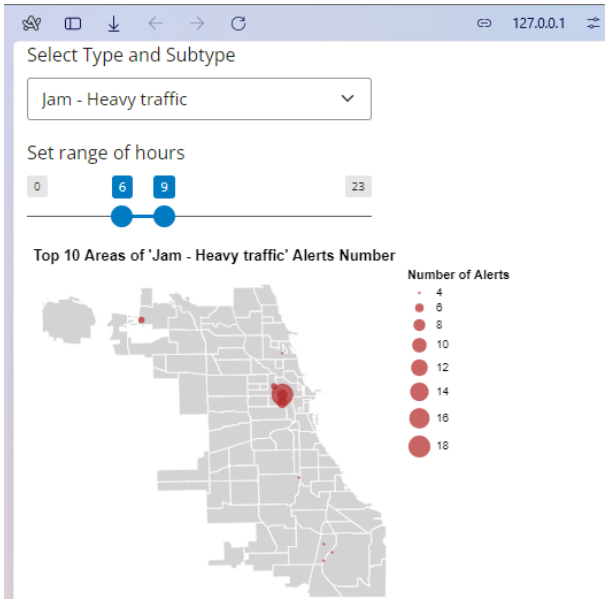
@reactive.calc
def chart_map():
    """Create the map"""
    chart = alt.Chart(geo_data()).mark_geoshape(
        fill="lightgray",
```

```
            stroke="white"
        ).project(
            type="equirectangular"
        ).properties(
            title=f"Top 10 Areas of '{input.type_subtype()}' Alerts Number",
            height=300,
            width=300
        )
        return chart

    # Create plot for output_widget
    @render_altair
    def chart_alerts_map_byhour():
        """Overlay the plots, if there are observations which satisfy
        conditions"""
        if len(df_chosen()) == 0:
            return chart_map()
        else:
            return chart_map() + chart_alerts_byhour()


app = App(app_ui, server)

```
```



b.
```



3.

a.

```
print_file_contents("./top_alerts_map_byhour_sliderrange/app_switch.py")
```

```
```python
from shiny import App, render, ui, reactive
from shinywidgets import render_altair, output_widget
import pandas as pd
import altair as alt
import json
from datetime import date, time

app_ui = ui.page_fluid(
    ui.input_select(
        id="type_subtype",
        label="Select Type and Subtype",
        choices=[]
    ),
    ui.input_switch("switch_to_range",
                    "Toggle to switch to range of hours", value=False),
    ui.panel_conditional(
        "!input.switch_to_range",
        ui.input_slider("hour_chosen", "Pick hour", 0, 23, 0)
```
```

```
    ),
    ui.panel_conditional(
        "input.switch_to_range",
        ui.input_slider("hours_range", "Set range of hours",
                        0, 23, (0, 23), drag_range=True)
    ),
    output_widget("chart_alerts_map_byhour")
)
# Attribution: parameter "drag_range" refering to Shiny document
(https://shiny.posit.co/py/api/core/ui.input_slider.html)


def server(input, output, session):
    # Load and store waze data
    @reactive.calc
    def df_top_alerts_map_byhour():
        """Create base df"""
        df = pd.read_csv("top_alerts_map_byhour.csv")
        df["hour_dt"] = pd.to_datetime(df["hour"], format="%H:%M")
        df["hour_dt"] = [x.time() for x in df["hour_dt"]]
        return df
        # Attribution: parameter "format" refering to Perplexity
        (https://www.perplexity.ai/search/from-shiny-import-app-render-u-Tt7cqT5WTmeh0z2CLZtU

    # Create choices for selector
    @reactive.calc
    def df_choices():
        """Summarize sets of type and subtype"""
        df = df_top_alerts_map_byhour().groupby(
            ["updated_type", "updated_subtype"]
        ).size().reset_index()
        return df

    @reactive.effect
    def _():
        """Define type-subtype choices"""
        choices = [f"{t} - {s}" for t, s in zip(
            df_choices()["updated_type"],
            df_choices()["updated_subtype"]
        )]
        ui.update_select("type_subtype", choices=choices)

    # Save inputs from UI side
```

```
@reactive.calc
def type_chosen():
    """Extract chosen type from selecter input"""
    if input.type_subtype() == None:
        result = "Accident"  # Set default for loading time
    # Might be categorized as "Hazard" type erroneously by the criterion
    below
    elif input.type_subtype() == "Road closed - Hazard":
        result = "Road closed"
    else:
        for type in df_choices()["updated_type"].unique():
            if type in input.type_subtype():
                result = type
                break
    return result

@reactive.calc
def subtype_chosen():
    """Extract chosen subtype from selecter input"""
    if input.type_subtype() == None:
        result = "Major"  # Set default for loading time
    else:
        result = input.type_subtype().replace(
            " - ", ""
        ).replace(
            type_chosen(), ""
        )
    return result

# range on/off from switch
@reactive.calc
def switch_to_range():
    return input.switch_to_range()

@reactive.calc
def hour_chosen():
    """Extract chosen hour from slider input"""
    result = f"{input.hour_chosen()}:00"
    return result

@reactive.calc
def hours_range():
    """Extract chosen range of hours from slider"""
```

```
    result = list(input.hours_range())
    return result

# Create output from the inputs
@reactive.calc
def df_chosen():
    """Create subset of waze df"""
    if switch_to_range():
        df = df_top_alerts_map_byhour()[(
            df_top_alerts_map_byhour()["updated_type"] == type_chosen()
        ) & (
            df_top_alerts_map_byhour(
            )["updated_subtype"] == subtype_chosen()
        ) & (
            df_top_alerts_map_byhour()["hour_dt"].between(
                time(hours_range()[0], 0), time(hours_range()[1], 0))
        )].drop("hour_dt", axis=1).head(10)
    else:
        df = df_top_alerts_map_byhour()[(
            df_top_alerts_map_byhour()["updated_type"] == type_chosen()
        ) & (
            df_top_alerts_map_byhour(
            )["updated_subtype"] == subtype_chosen()
        ) & (
            df_top_alerts_map_byhour()["hour"] == hour_chosen()
        )].drop("hour_dt", axis=1)
    return df

@reactive.calc
def domain():
    """Set appropriate domain for chosen type and subtype"""
    domain = [min(df_chosen()["count"]), max(df_chosen()["count"])]
    return domain

@reactive.calc
def chart_alerts_byhour():
    """Create scatter plot for number of alert"""
    chart = alt.Chart(df_chosen()).mark_point(
        color="firebrick",
        filled=True
    ).encode(
        longitude="longitude:Q",
        latitude="latitude:Q",
```

```
        size=alt.Size(
            "count:Q",
            scale=alt.Scale(
                domain=domain()
            ),
            legend=alt.Legend(
                title="Number of Alerts"
            )
        )
    ).properties(
        title=f"Top 10 Areas of '{input.type_subtype()}' Alerts Number",
        height=300,
        width=300
    )
    return chart

# Create map
@reactive.calc
def geo_data():
    """Load and store geojson"""
    with open("chicago-boundaries.geojson") as f:
        chicago_geojson = json.load(f)
    geo_data = alt.Data(values=chicago_geojson["features"])
    return geo_data

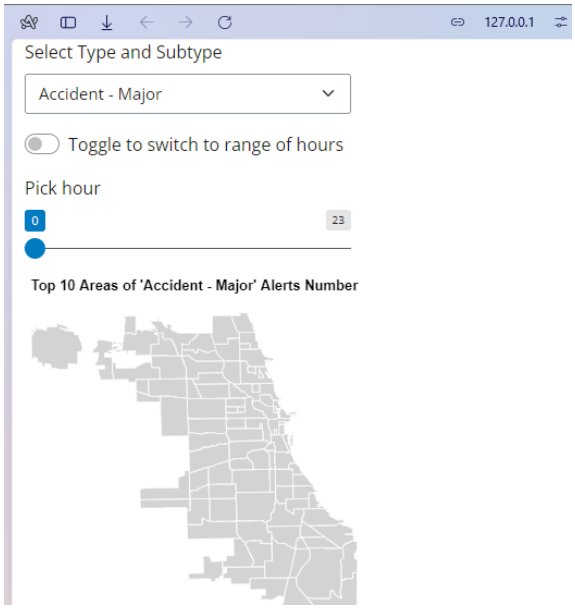
@reactive.calc
def chart_map():
    """Create the map"""
    chart = alt.Chart(geo_data()).mark_geoshape(
        fill="lightgray",
        stroke="white"
    ).project(
        type="equirectangular"
    ).properties(
        title=f"Top 10 Areas of '{input.type_subtype()}' Alerts Number",
        height=300,
        width=300
    )
    return chart

# Create plot for output_widget
@render_altair
def chart_alerts_map_byhour():
```

```
        """Overlay the plots, if there are observations which satisfy
        conditions"""
        if len(df_chosen()) == 0:
            return chart_map()
        else:
            return chart_map() + chart_alerts_byhour()

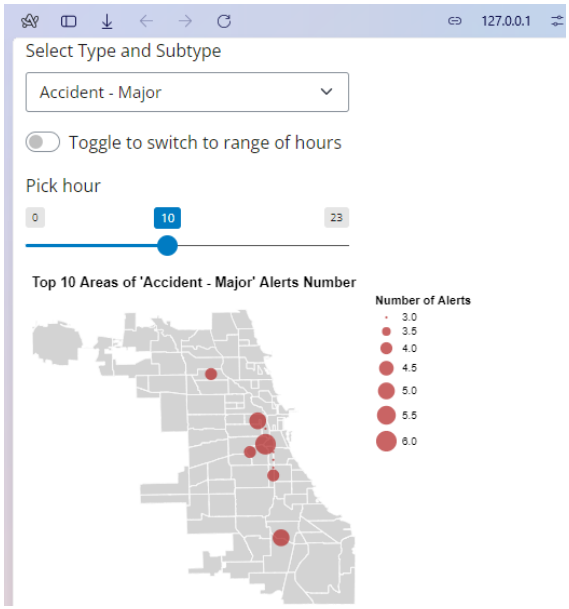

app = App(app_ui, server)

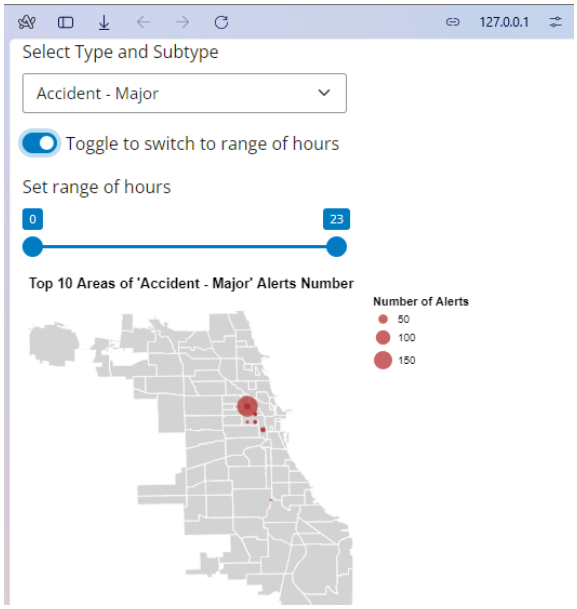
```
```



possible values: True, False

b.
```





c.

The same screenshots as in b.

d.

After creating subset “df_chosen()” (, no difference wheather switch is toggled), create sub-subsets for morning and afternoon observations, by checking if each data in “hour” column are included in [“00:00”, “01:00”, … “12:00”] or in [“13:00”, “14:00”, … “23:00”].

Then we need to define two distinct plot functions instead of “chart_alerts_byhour()” for corresponding two sub-subsets, with different color parameter in “mark_point()”. We need to overlay both of these plots as well as the map plot to render final output.

Also, we need some other necessary fixes on the server side for our app, not complehensive but such as

- fix domain();

  Change the minimum value of domain into zero.

- fix chart_map();

  Change to “fill=None” in mark_geoshape()

, etc.