Problem Set 6 - Waze Shiny Dashboard

Peter Ganong, Maggie Shi, and Andre Oviedo

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: MB
2. "I have uploaded the names of anyone I worked with on the problem set **here**" **___** (2 point)
3. Late coins used this pset: **___** Late coins left after submission: **___**
4. Before starting the problem set, make sure to read and agree to the terms of data usage for the Waze data here.
5. Knit your `ps6.qmd` as a pdf document and name it `ps6.pdf`.
6. Push your `ps6.qmd`, `ps6.pdf`, `requirements.txt`, and all created folders (we will create three Shiny apps so you will have at least three additional folders) to your Github repo (5 points). It is fine to use Github Desktop.
7. Submit `ps6.pdf` and also link your Github repo via Gradescope (5 points)
8. Tag your submission in Gradescope. For the Code Style part (10 points) please tag the whole correspondingsection for the code style rubric.

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

IMPORTANT: For the App portion of the PS, in case you can not arrive to the expected functional dashboard we will need to take a look at your `app.py` file. You can use the following

code chunk template to "import" and print the content of that file. Please, don't forget to also tag the corresponding code chunk as part of your submission!

```
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.

```
import zipfile
zip_path =
 ↪  r"C:\\Users\\Mitch\\Documents\\GitHub\student30538\\problem_sets\\ps6\\waze_data.zip"
csv_filename = "waze_data_sample.csv"

with zipfile.ZipFile(zip_path, 'r') as z:
    with z.open(csv_filename) as f:
        waze_sample_df = pd.read_csv(f)

# Display the DataFrame
print(waze_sample_df.head())

waze_sample_df.dtypes
```

```
   Unnamed: 0         city  confidence  nThumbsUp        street  \
0      584358  Chicago, IL           0        NaN           NaN
1      472915  Chicago, IL           0        NaN        I-90 E
2      550891  Chicago, IL           0        NaN        I-90 W
3      770659  Chicago, IL           0        NaN           NaN
4      381054  Chicago, IL           0        NaN  N Pulaski Rd

                                  uuid country         type  \
0  c9b88a12-79e8-44cb-aadd-a75855fc4bcb      US          JAM
1  7c634c0a-099c-4262-b57f-e893bdebce73      US  ROAD_CLOSED
2  7aa3c61a-f8dc-4fe8-bbb0-db6b9e0dc53b      US       HAZARD
3  3b95dd2f-647c-46de-b4e1-8ebc73aa9221      US       HAZARD
4  13a5e230-a28a-4bf4-b928-bc1dd38850e0      US          JAM

                        subtype  roadType  reliability  magvar  \
0                           NaN        17            5     116
1              ROAD_CLOSED_EVENT         3            6     173
2  HAZARD_ON_SHOULDER_CAR_STOPPED         3            5     308
3                HAZARD_ON_ROAD        20            5     155
4              JAM_HEAVY_TRAFFIC         7            5     178

   reportRating                       ts                          geo  \
0             5  2024-07-02 18:27:40 UTC   POINT(-87.64577 41.892743)
1             0  2024-06-16 10:13:19 UTC  POINT(-87.646359 41.886295)
2             5  2024-05-02 19:01:47 UTC   POINT(-87.695982 41.93272)
3             2  2024-03-25 18:53:24 UTC  POINT(-87.669253 41.904497)
4             2  2024-06-03 21:17:33 UTC  POINT(-87.728322 41.978769)

                      geoWKT
0   Point(-87.64577 41.892743)
1  Point(-87.646359 41.886295)
2   Point(-87.695982 41.93272)
3  Point(-87.669253 41.904497)
4  Point(-87.728322 41.978769)

Unnamed: 0        int64
city             object
confidence        int64
nThumbsUp       float64
street           object
uuid             object
```

```
country          object
type             object
subtype          object
roadType          int64
reliability       int64
magvar            int64
reportRating      int64
ts               object
geo              object
geoWKT           object
dtype: object
```

Variable names: Unnamed:0: Quantitative city: Nominal confidence: Ordinal nThumbsUp: Quantititative street: Nominal uuid: Nominal country: Nominal type: Nominal subtype: Nominal roadType: Nominal reliability: Ordinal magvar: Quantitative reportRating: Ordinal

2.

```
csv_filename = "waze_data.csv"

with zipfile.ZipFile(zip_path, 'r') as z:
    with z.open(csv_filename) as f:
        waze_df = pd.read_csv(f)

# Display the DataFrame
print(waze_df.head())

#assign an object that is the column names
variable_name=waze_df.columns

#assign an object the count of the number of each columns nas:
na_count=waze_df.isna().sum()

#assign another object the count of the number of non NAs
non_na_count=waze_df.notna().sum()
#define the df using our created objects:
waze_df_nas=pd.DataFrame({"variable_name":variable_name,
"na_count":na_count,"non_na_count":non_na_count})

#convert to long so we can create a stacked bar chart:
waze_df_nas = waze_df_nas.melt(id_vars="variable_name",
                                    value_vars=["na_count", "non_na_count"],
```

```
                                    var_name="count_type",
                                    value_name="count")

alt.Chart(waze_df_nas).mark_bar().encode(
    alt.X("variable_name:N",title="Variable Name"),
    alt.Y("count",title="Number of Observations"),
    color=alt.Color("count_type",title="Type")
).properties(title="Number of NAs v. Non NAs by Variable in Waze Data")
```

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

```
alt.Chart(...)
```

nThumbsUp, street, subtype all contain NAs. nThumbsUp by far has the greatest number of NAs of all variables.

3.

```
print(waze_df["type"].unique())
print(waze_df["subtype"].unique())

print("Jam type
 ↪  subtypes:",waze_df[waze_df["type"]=="JAM"]["subtype"].unique())
print("Accident type
 ↪  subtypes:",waze_df[waze_df["type"]=="ACCIDENT"]["subtype"].unique())
print("Road closed type
 ↪  subtypes:",waze_df[waze_df["type"]=="ROAD_CLOSED"]["subtype"].unique())
print("Hazard types
 ↪  subtypes:",waze_df[waze_df["type"]=="HAZARD"]["subtype"].unique())
```

```
['JAM' 'ACCIDENT' 'ROAD_CLOSED' 'HAZARD']
[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']
Jam type subtypes: [nan 'JAM_HEAVY_TRAFFIC' 'JAM_MODERATE_TRAFFIC'
'JAM_STAND_STILL_TRAFFIC'
 'JAM_LIGHT_TRAFFIC']
Accident type subtypes: [nan 'ACCIDENT_MAJOR' 'ACCIDENT_MINOR']
Road closed type subtypes: [nan 'ROAD_CLOSED_EVENT'
'ROAD_CLOSED_CONSTRUCTION' 'ROAD_CLOSED_HAZARD']
Hazard types subtypes: [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_HAIL']
```

All of the types have a subtype called NA.

Hazard subtype probably has a further level of subtypes, because there's 9 under the bucket of "On Road", "On Shoulder" has 3, "Weather" has 4. -Jam (type) –Heavy Traffic(subtype) –Moderate Traffic(subtype) –Still Traffic(subtype) –Light Traffic(subtype) -Accident(type) –Major(subtype) –Minor(subtype) -Road Closed(type) –Event (subtype) –Construction(subtype) –Hazard(subtype) -Hazard (type) –On Road(subtype) —Car stopped(sub-subtype) —Construction(sub-subtype) —Emergency Vehicle(sub-subtype) —Ice(sub-subtype) —Object(sub-subtype) —Pot Hole(sub-subtype) —Traffic Light Fault(sub-subtype) —Lane Closed(sub-subtype) —Road Kill(sub-subtype) –On Shoulder (subtype) —Car Stopped(sub-subtype) —Animals(sub-subtype) —Missing Sign(sub-subtype) –Weather(subtype) —Flood(sub-subtype) —Fog(sub-subtype) —Heavy Snow(sub-subtype) —Hail(sub-subtype)

```
#calculate the proportion of the df that has na for a subtype
waze_df["subtype"].isna().size/waze_df.size
```

```
0.06666666666666667
```

I do think we ought to keep the NA subtypes because a substantial amount of the data has NA for subtype, and does not have any association with any particular type

4.

```
original_type=waze_df["type"].unique
original_subtype=waze_df["subtype"].unique

updated_waze_df = waze_df[["type",
 ↪  "subtype"]].drop_duplicates().reset_index(drop=True)

print(updated_waze_df)

#now use a function on the existing columns to define
#the new columns

updated_waze_df["updated_type"]=updated_waze_df["type"]

def extract_after_underscore(subtype):
    if pd.isna(subtype):  # Check if the value is NA
        return "Unclassified"
    if isinstance(subtype, str):
        # Check for specific keywords first
        if "ON_ROAD" in subtype:
            return "ON_ROAD"
        elif "ON_SHOULDER" in subtype:
```

```
            return "ON_SHOULDER"
        elif "WEATHER" in subtype:
            return "WEATHER"
        # Otherwise, extract after the first underscore
        if "_" in subtype:
            return subtype.split("_", 1)[1]
    return subtype  # Return as-is for other cases

updated_waze_df["updated_subtype"] =
 ↪  updated_waze_df["subtype"].apply(extract_after_underscore)

#now extract the subsubtype
#conditions: extract everything beyond the subtype's ending
#underscore.
#if its on road or on shoulder, after the 3rd underscore
#if its weather, 2nd underscore. return NA for other cases.
def extractsubsubtype(subtype):
    if isinstance(subtype, str):
        parts = subtype.split("_")
        if "WEATHER" in parts:
            # For cases containing "WEATHER", return everything after the
             ↪  second underscore
            if len(parts) > 2:
                return "_".join(parts[2:])
        elif len(parts) > 3:
            # Otherwise, return everything after the third underscore
            return "_".join(parts[3:])
    return None  # Return None for other cases

updated_waze_df["updated_subsubtype"] = [extractsubsubtype(subtype) for
 ↪  subtype in updated_waze_df["subtype"]]

#now merge the two dfs:

merged_waze_df=pd.merge(updated_waze_df,waze_df,how="outer",on=["type","subtype"])

#count number of rows where type==accident and
#subtype==unclassified

condition_df=merged_waze_df[(merged_waze_df["type"]=="ACCIDENT")&
 ↪  (merged_waze_df["updated_subtype"]=="Unclassified")]
```

```
print("The number of rows with type accident and subtype unclassified
 ↪  is:",condition_df.size)
```

```
           type                              subtype
0           JAM                                  NaN
1      ACCIDENT                                  NaN
2   ROAD_CLOSED                                  NaN
3        HAZARD                                  NaN
4      ACCIDENT                       ACCIDENT_MAJOR
5      ACCIDENT                       ACCIDENT_MINOR
6        HAZARD                       HAZARD_ON_ROAD
7        HAZARD            HAZARD_ON_ROAD_CAR_STOPPED
8        HAZARD           HAZARD_ON_ROAD_CONSTRUCTION
9        HAZARD      HAZARD_ON_ROAD_EMERGENCY_VEHICLE
10       HAZARD                    HAZARD_ON_ROAD_ICE
11       HAZARD                 HAZARD_ON_ROAD_OBJECT
12       HAZARD               HAZARD_ON_ROAD_POT_HOLE
13       HAZARD    HAZARD_ON_ROAD_TRAFFIC_LIGHT_FAULT
14       HAZARD                    HAZARD_ON_SHOULDER
15       HAZARD        HAZARD_ON_SHOULDER_CAR_STOPPED
16       HAZARD                       HAZARD_WEATHER
17       HAZARD                  HAZARD_WEATHER_FLOOD
18          JAM                     JAM_HEAVY_TRAFFIC
19          JAM                  JAM_MODERATE_TRAFFIC
20          JAM               JAM_STAND_STILL_TRAFFIC
21  ROAD_CLOSED                     ROAD_CLOSED_EVENT
22       HAZARD            HAZARD_ON_ROAD_LANE_CLOSED
23       HAZARD                    HAZARD_WEATHER_FOG
24  ROAD_CLOSED              ROAD_CLOSED_CONSTRUCTION
25       HAZARD              HAZARD_ON_ROAD_ROAD_KILL
26       HAZARD            HAZARD_ON_SHOULDER_ANIMALS
27       HAZARD       HAZARD_ON_SHOULDER_MISSING_SIGN
28          JAM                     JAM_LIGHT_TRAFFIC
29       HAZARD             HAZARD_WEATHER_HEAVY_SNOW
30  ROAD_CLOSED                    ROAD_CLOSED_HAZARD
31       HAZARD                   HAZARD_WEATHER_HAIL
The number of rows with type accident and subtype unclassified is: 438462
```

438462 rows with a type of accident and an unclassified subtype.

1.

```
import re
#looked at the documentation and found what I think is most
#important re method and syntax to incorporate into a function.
#ChatGPT prompt: I want to split the longitude and latitude #using
 ↪  re.split(/s, txt), with each txt being the geoWKT output, #assigning text
 ↪  before the blank space to longitude, and the #text after to latitude
# Function to extract longitude and latitude
def split_long_lat(geoWKT):
    # Use regex to extract only the numeric coordinates
    match = re.search(r'\(([-\d.]+)\s+([-\d.]+)\)', geoWKT)
    if match:
        longitude, latitude = match.groups()
        return float(longitude), float(latitude)
    else:
        # Return None for invalid rows
        return None, None

# Apply the function and split into separate columns
merged_waze_df[['longitude', 'latitude']] =
 ↪  merged_waze_df['geoWKT'].apply(split_long_lat).apply(pd.Series)

longitude_binned=round(merged_waze_df["longitude"],2)
latitude_binned=round(merged_waze_df["latitude"],2)

binned_coords=pd.DataFrame({"longitude_binned":longitude_binned,
"latitude_binned":latitude_binned})

#create a summary table that counts the number of
#each combination

grouped_binned_coords=binned_coords.groupby(["longitude_binned","latitude_binned"]).size().re
#longitude and latitude combo with greatest number of
#observations:
print("longitude and latitude combo with greatest number of observations:"
 ↪  ,grouped_binned_coords.head(1))
```

```
longitude and latitude combo with greatest number of observations:
longitude_binned  latitude_binned  count
492            -87.65            41.88  21325
```

```
#take the original df, assign each observation its bin by
#rounding to the nearest hundredth.
```

```
merged_waze_df["latitude"]=merged_waze_df["latitude"].round(2)
merged_waze_df["longitude"]=merged_waze_df["longitude"].round(2)

#groupby type subtype longitude and latitude. this'll give each
#locations number of traffic incidences by type and subtype
agg_type_subtype_df=merged_waze_df.groupby(["type","updated_subtype","longitude","latitude"])

agg_type_subtype_df.to_csv("C:\\Users\\Mitch\\Documents\\GitHub\problem-set-6\\top_alerts_map

agg_type_subtype_df.shape
```

```
<>:11: SyntaxWarning: invalid escape sequence '\p'
<>:11: SyntaxWarning: invalid escape sequence '\p'
C:\Users\Mitch\AppData\Local\Temp\ipykernel_29540\3044782989.py:11:
SyntaxWarning: invalid escape sequence '\p'

  agg_type_subtype_df.to_csv("C:\\Users\\Mitch\\Documents\\GitHub\problem-set-6\\top_alerts_m

(6675, 5)
```

The level of aggregation is looking at the number of type subtype combo at a longitude and latitude combo.

The dataframe has 6675 rows.

2.

```
import altair as alt
min_latitude=agg_type_subtype_df["latitude"].min()
max_latitude=agg_type_subtype_df["latitude"].max()

min_longitude=agg_type_subtype_df["longitude"].min()
max_longitude=agg_type_subtype_df["longitude"].max()

alt.Chart(agg_type_subtype_df).mark_circle().encode(
    alt.X(
        "longitude",
        scale=alt.Scale(domain=[min_longitude, max_longitude])
    ),
    alt.Y(
        "latitude",
        scale=alt.Scale(domain=[min_latitude, max_latitude])
```

```
    ),
    size="count"
).transform_filter(
    (alt.datum.type == "JAM") & (alt.datum.updated_subtype ==
 ↪  "HEAVY_TRAFFIC")
    ).properties(title="Heavy Traffic Jams in Chicago")
```

`alt.Chart(...)`

3. Yes you can download it using requests. I'm using https://www.geeksforgeeks.org/downloading-files-web-using-python/ as a source to guide me in doing so.

b.

4.

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

1.

a.

```
import re
#looked at the documentation and found what I think is most
#important re method and syntax to incorporate into a function.
#ChatGPT prompt: I want to split the longitude and latitude #using
 ↪  re.split(/s, txt), with each txt being the geoWKT output, #assigning text
 ↪  before the blank space to longitude, and the #text after to latitude
# Function to extract longitude and latitude
def split_long_lat(geoWKT):
    # Use regex to extract only the numeric coordinates
    match = re.search(r'\(([-\d.]+)\s+([-\d.]+)\)', geoWKT)
    if match:
        longitude, latitude = match.groups()
        return float(longitude), float(latitude)
    else:
        # Return None for invalid rows
        return None, None

# Apply the function and split into separate columns
merged_waze_df[['longitude', 'latitude']] =
 ↪  merged_waze_df['geoWKT'].apply(split_long_lat).apply(pd.Series)
```

```
longitude_binned=round(merged_waze_df["longitude"],2)
latitude_binned=round(merged_waze_df["latitude"],2)

binned_coords=pd.DataFrame({"longitude_binned":longitude_binned,
"latitude_binned":latitude_binned})
```

b.

```
#create a summary table that counts the number of
#each combination

grouped_binned_coords=binned_coords.groupby(["longitude_binned","latitude_binned"]).size().re
#longitude and latitude combo with greatest number of
#observations:
print("longitude and latitude combo with greatest number of observations:"
 ↪  ,grouped_binned_coords.head(1))
```

```
longitude and latitude combo with greatest number of observations:
longitude_binned  latitude_binned  count
492            -87.65            41.88  21325
```

c.

```
#take the original df, assign each observation its bin by
#rounding to the nearest hundredth.

merged_waze_df["latitude"]=merged_waze_df["latitude"].round(2)
merged_waze_df["longitude"]=merged_waze_df["longitude"].round(2)

#groupby type subtype longitude and latitude. this'll give each
#locations number of traffic incidences by type and subtype
agg_type_subtype_df=merged_waze_df.groupby(["type","updated_subtype","longitude","latitude"])

agg_type_subtype_df.to_csv("C:\\Users\\Mitch\\Documents\\GitHub\problem-set-6\\top_alerts_map

agg_type_subtype_df.shape
```

```
<>:11: SyntaxWarning: invalid escape sequence '\p'
<>:11: SyntaxWarning: invalid escape sequence '\p'
```

```
C:\Users\Mitch\AppData\Local\Temp\ipykernel_29540\3044782989.py:11:
SyntaxWarning: invalid escape sequence '\p'

  agg_type_subtype_df.to_csv("C:\\Users\\Mitch\\Documents\\GitHub\problem-set-6\\top_alerts_
```

```
(6675, 5)
```

The level of aggregation is looking at the number of type subtype combo at a longitude and latitude combo.

The dataframe has 6675 rows.

2.

```
import altair as alt
min_latitude=agg_type_subtype_df["latitude"].min()
max_latitude=agg_type_subtype_df["latitude"].max()

min_longitude=agg_type_subtype_df["longitude"].min()
max_longitude=agg_type_subtype_df["longitude"].max()

alt.Chart(agg_type_subtype_df).mark_circle().encode(
    alt.X(
        "longitude",
        scale=alt.Scale(domain=[min_longitude, max_longitude])
    ),
    alt.Y(
        "latitude",
        scale=alt.Scale(domain=[min_latitude, max_latitude])
    ),
    size="count"
).transform_filter(
    (alt.datum.type == "JAM") & (alt.datum.updated_subtype ==
    "HEAVY_TRAFFIC")
    ).transform_window(
    rank='rank(count)',
    sort=[alt.SortField('count', order='descending')]
).transform_filter(
    alt.datum.rank <= 10
    ).properties(title="Top 10 Locations for Heavy Traffic Jams in Chicago")
```

```
alt.Chart(...)
```

3.

a.

```
import requests

#use the url for the json data in pset
json_url =
 ↪  "https://data.cityofchicago.org/api/geospatial/bbvz-uum9?method=export&format=GeoJSON"

response = requests.get(json_url)
#save the file
file_path =
 ↪  r"C:\\Users\\Mitch\Documents\\GitHub\\problem-set-6\\top_alerts_map\\Boundaries
 ↪  - Neighborhoods.geojson"

with open(file_path, "wb") as file:
    file.write(response.content)

#file confirmed in the folder.
```

b.

```
#file path is already defined above so there's no reason to
#use that part of the template code.
with open(file_path) as f:
    chicago_geojson = json.load(f)
geo_data = alt.Data(values=chicago_geojson["features"])
```

4.

```
jams_chart=alt.Chart(agg_type_subtype_df).mark_circle().encode(
    alt.X(
        "longitude",
        scale=alt.Scale(domain=[min_longitude, max_longitude])
    ),
    alt.Y(
        "latitude",
        scale=alt.Scale(domain=[min_latitude, max_latitude])
    ),
    size="count"
).transform_filter(
    (alt.datum.type == "JAM") & (alt.datum.updated_subtype ==
 ↪  "HEAVY_TRAFFIC")
```

```
    ).transform_window(
    rank='rank(count)',
    sort=[alt.SortField('count', order='descending')]
).transform_filter(
    alt.datum.rank <= 10
    ).properties(title="Top 10 Locations for Heavy Traffic Jams in Chicago")

base_map=alt.Chart(geo_data).mark_geoshape().encode(
    fill=alt.value("grey")
).project(type="identity", reflectY=True)
combined_chart=base_map+jams_chart
combined_chart.show()
```

```
alt.LayerChart(...)
```

5.

a.

There are 16 options in the drop down menu I created.

```
from shiny import App, render, ui

menu_choices = (
    agg_type_subtype_df[["type", "updated_subtype"]]
    .drop_duplicates()
    .apply(lambda row: f"{row['type']} - {row['updated_subtype']}", axis=1)
    .tolist()
)

app_ui = ui.page_fluid(
    ui.panel_title("Traffic Incidents in Chicago"),
    ui.input_select(id="incident",
    label="choose a type",
    choices=menu_choices)
)
def server(input,output,session):
    @render.text
    def txt():
        return f"{input} selected"

#just want the dropdown menu at this stage; there's nothing
```

```
#for the server to run.
app=App(app_ui,server)
```

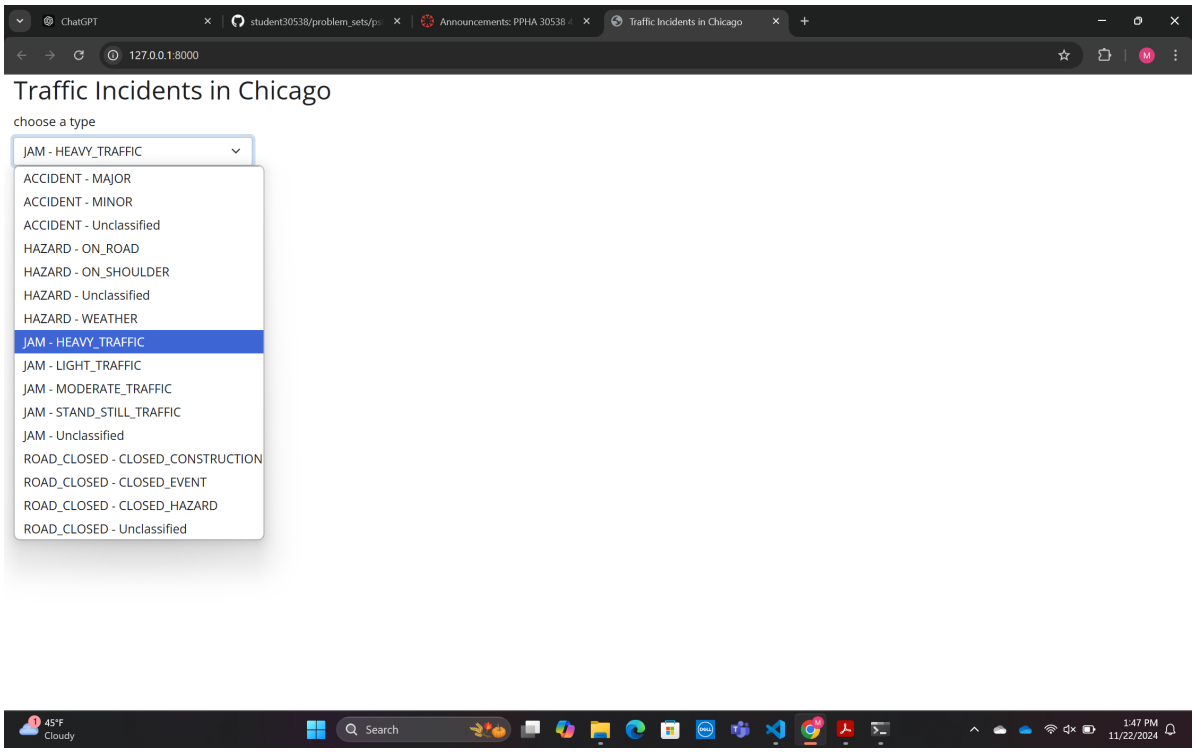


Figure 1: Drop Down Menu UI

b.

Figure 2: Full Dropdown with plot

c. I couldn't get the projection right, but using my knowledge of the city and the fact that it is more concentrated on the eastern part of the city, I'd say most road closures due to events are along the lakeshore, in wrigleyville

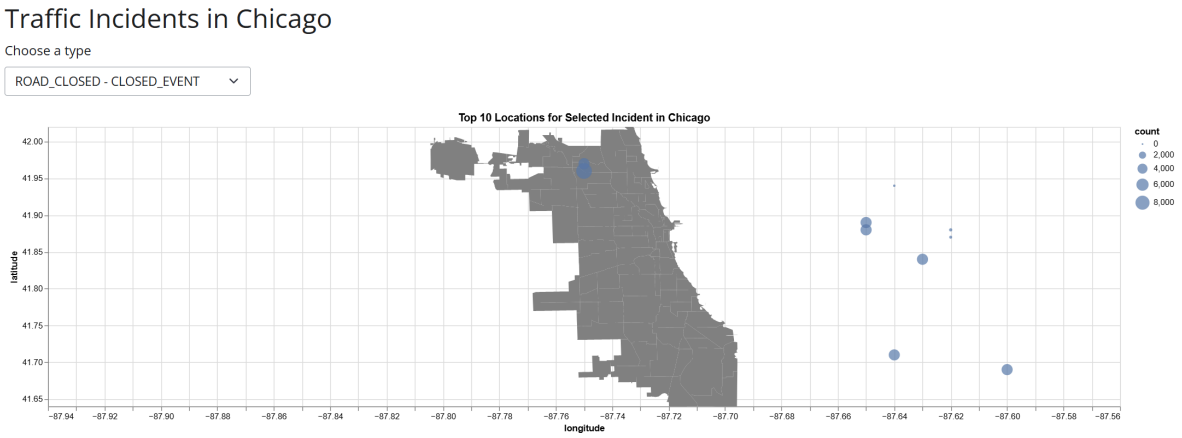


Figure 3: Road Closures Map

d. The dashboard could also be used to identify where most major accidents occur in the city.

From the dashboard, we can tell that most major accidents occur on I90/94.

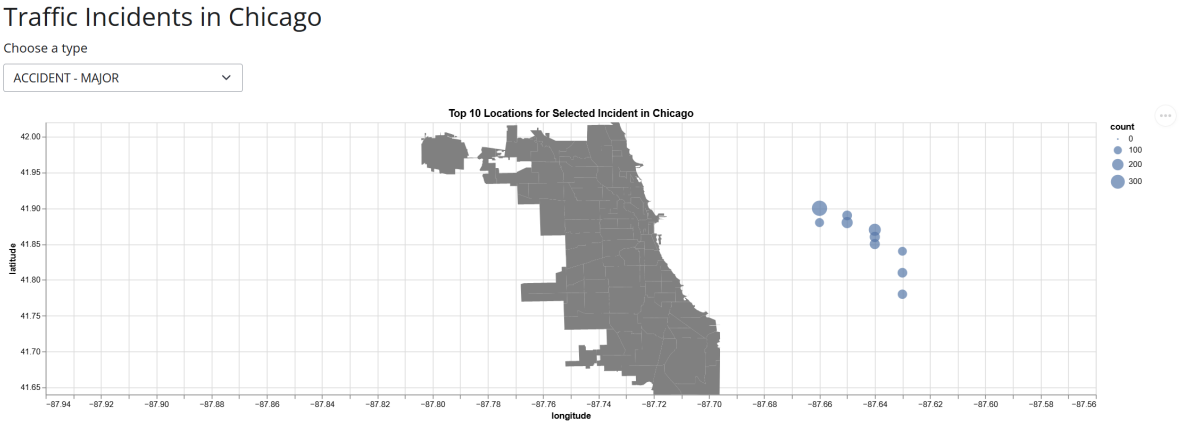


Figure 4: Major Accidents

e.

Adding the subsubtype column would provide some granularity to our analysis. We'd have to aggregate on the type, subtype, subsubtype, longitude, and latitude level to achieve this.

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

1.

a.

```
print(waze_df["ts"].nunique)
```

```
<bound method IndexOpsMixin.nunique of 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
                  ...
778089    2024-03-19 21:23:53 UTC
778090    2024-03-19 22:20:02 UTC
778091    2024-03-19 23:32:44 UTC
778092    2024-03-19 16:49:46 UTC
778093    2024-03-19 18:24:46 UTC
Name: ts, Length: 778094, dtype: object>
```

It would be a bad idea to group by the ts column because every single entry is unique; you wouldn't collapse the data at all. We probably will bin by minute, or hour, because each hour minute second combo has a high likeliehood of being unique. b.

```
import time
#make the ts column datetime format

merged_waze_df["ts"]=pd.to_datetime(merged_waze_df["ts"])

#extract the hour, assign it to a new column:

merged_waze_df["hour"]=merged_waze_df["ts"].dt.strftime("%H:00")
```

```
#now group by longitude, latitude, hour, type, and subtype
time_waze_df=merged_waze_df.groupby(

 ↪  ["longitude","latitude","hour","type","updated_subtype"]).size().reset_index(name="count"

print(time_waze_df.shape)

time_waze_df.to_csv("C:\\Users\\Mitch\\Documents\\GitHub\\problem-set-6\\top_alerts_map_byhou
```

```
(62825, 6)
```

62,825 rows in the new df.

c.

```
#create a filtered df, where I'm selecting jam for type
#heavy traffic as subtype, with an additional time element for
#a single snapshot in time. Create 3 plots, per the ed discussion
#thread with professor Shi.
#6am plot:
```

```
base_time_chart=alt.Chart(time_waze_df).mark_circle().encode(
    alt.X(
        "longitude",
        scale=alt.Scale(domain=[min_longitude, max_longitude]),
    ),
    alt.Y(
        "latitude",
        scale=alt.Scale(domain=[min_latitude, max_latitude]),
    ),
    size="count:Q",
    tooltip=["hour", "latitude", "longitude", "count"],
).transform_filter(
    (alt.datum.type == "JAM")
    & (alt.datum.updated_subtype == "HEAVY_TRAFFIC")
    & (alt.datum.hour=="06:00")
).transform_window(
    rank="rank(count)",
    sort=[alt.SortField("count", order="descending")],
    groupby=["hour"],
).transform_filter(
    alt.datum.rank <= 10
).properties(
    title="Top 10 Locations for Heavy Traffic Jams in Chicago at 6 AM"
).project(type="identity", reflectY=True)


#noon plot:
base_map=alt.Chart(geo_data).mark_geoshape().encode(
    fill=alt.value("grey")
).project(type="identity", reflectY=True)
combined_chart=base_map+base_time_chart
combined_chart.show()

base_time_chart=alt.Chart(time_waze_df).mark_circle().encode(
    alt.X(
        "longitude",
        scale=alt.Scale(domain=[min_longitude, max_longitude]),
    ),
    alt.Y(
        "latitude",
        scale=alt.Scale(domain=[min_latitude, max_latitude]),
    ),
```

```
    size="count:Q",
    tooltip=["hour", "latitude", "longitude", "count"],
).transform_filter(
    (alt.datum.type == "JAM")
    & (alt.datum.updated_subtype == "HEAVY_TRAFFIC")
    & (alt.datum.hour=="12:00")
).transform_window(
    rank="rank(count)",
    sort=[alt.SortField("count", order="descending")],
    groupby=["hour"],
).transform_filter(
    alt.datum.rank <= 10
).properties(
    title="Top 10 Locations for Heavy Traffic Jams in Chicago at noon"
).project(type="identity", reflectY=True)



base_map=alt.Chart(geo_data).mark_geoshape().encode(
    fill=alt.value("grey")
).project(type="identity", reflectY=True)
combined_chart=base_map+base_time_chart
combined_chart.show()


#6pm plot:
base_time_chart=alt.Chart(time_waze_df).mark_circle().encode(
    alt.X(
        "longitude",
        scale=alt.Scale(domain=[min_longitude, max_longitude]),
    ),
    alt.Y(
        "latitude",
        scale=alt.Scale(domain=[min_latitude, max_latitude]),
    ),
    size="count:Q",
    tooltip=["hour", "latitude", "longitude", "count"],
).transform_filter(
    (alt.datum.type == "JAM")
    & (alt.datum.updated_subtype == "HEAVY_TRAFFIC")
    & (alt.datum.hour=="18:00")
).transform_window(
```

```
    rank="rank(count)",
    sort=[alt.SortField("count", order="descending")],
    groupby=["hour"],
).transform_filter(
    alt.datum.rank <= 10
).properties(
    title="Top 10 Locations for Heavy Traffic Jams in Chicago at 6 PM"
).project(type="identity", reflectY=True)



base_map=alt.Chart(geo_data).mark_geoshape().encode(
    fill=alt.value("grey")
).project(type="identity", reflectY=True)
combined_chart=base_map+base_time_chart
combined_chart.show()
```

```
alt.LayerChart(...)
```

```
alt.LayerChart(...)
```

```
alt.LayerChart(...)
```

2.

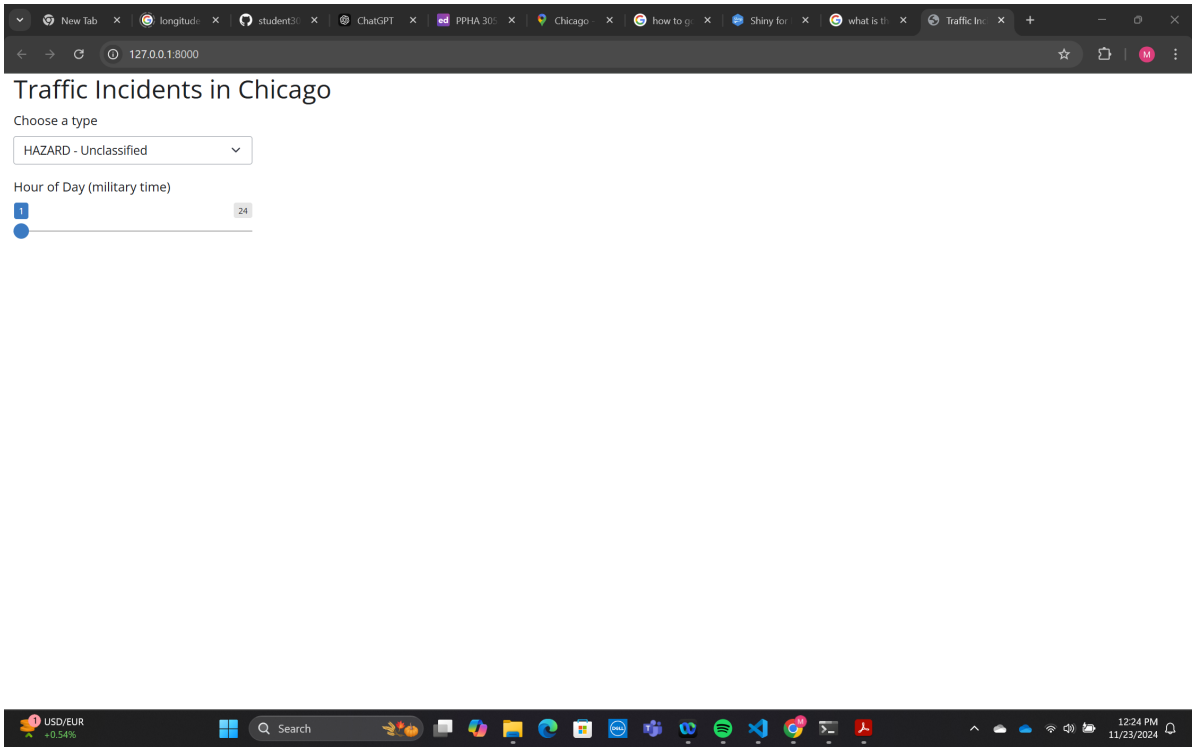
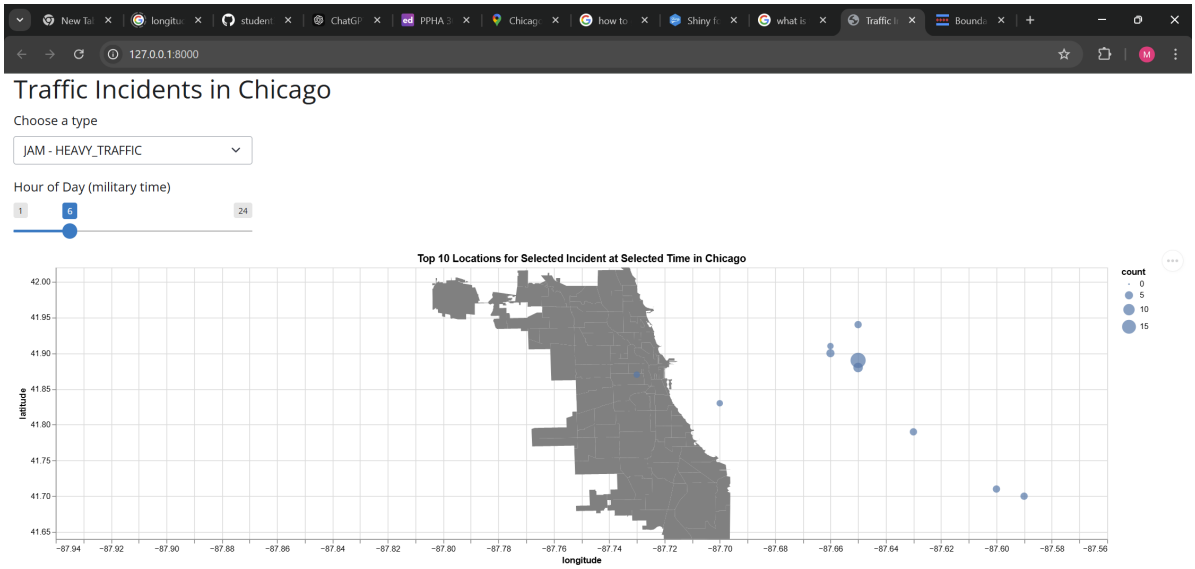
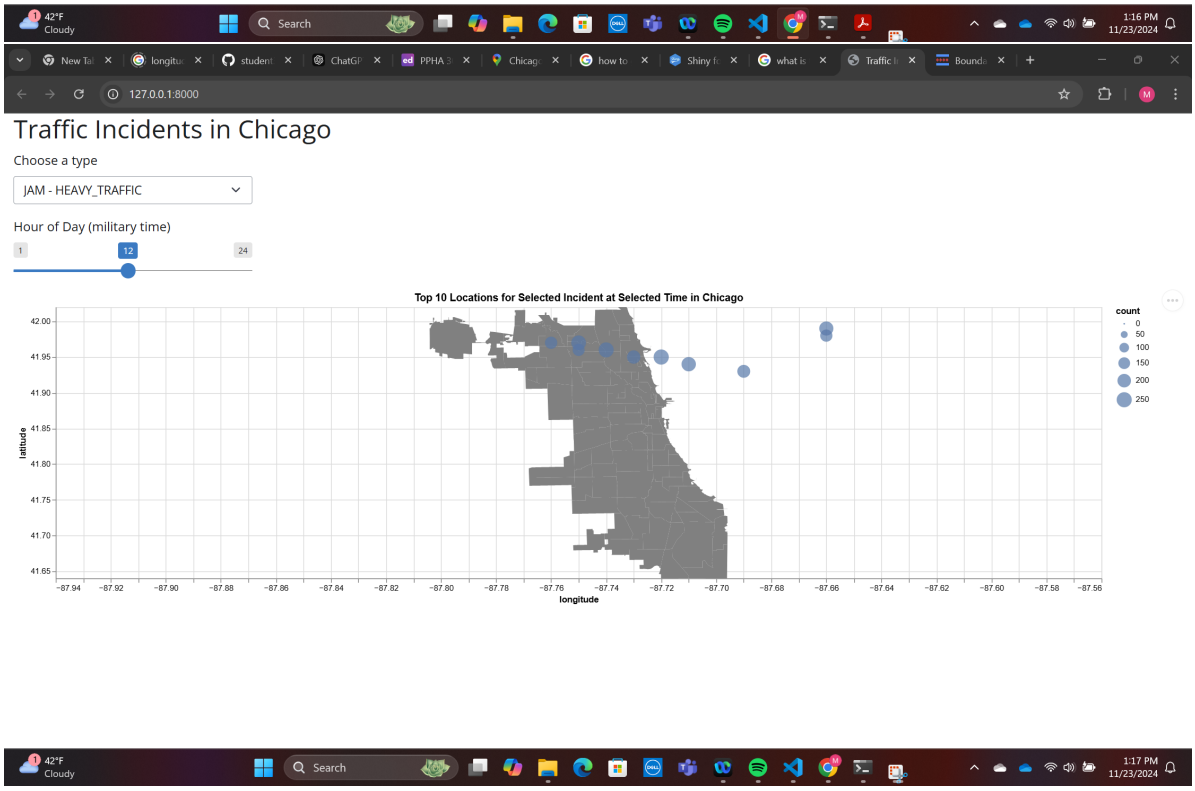


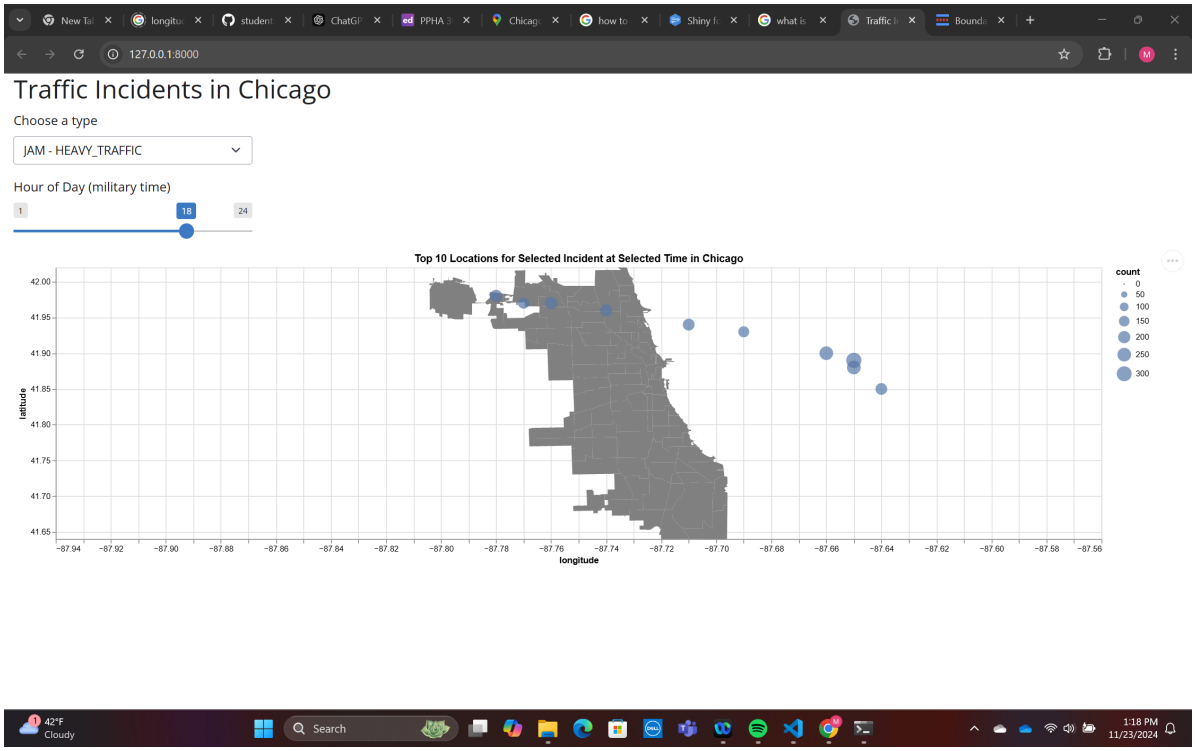
Figure 5: UI for the slider and dropdown

a.

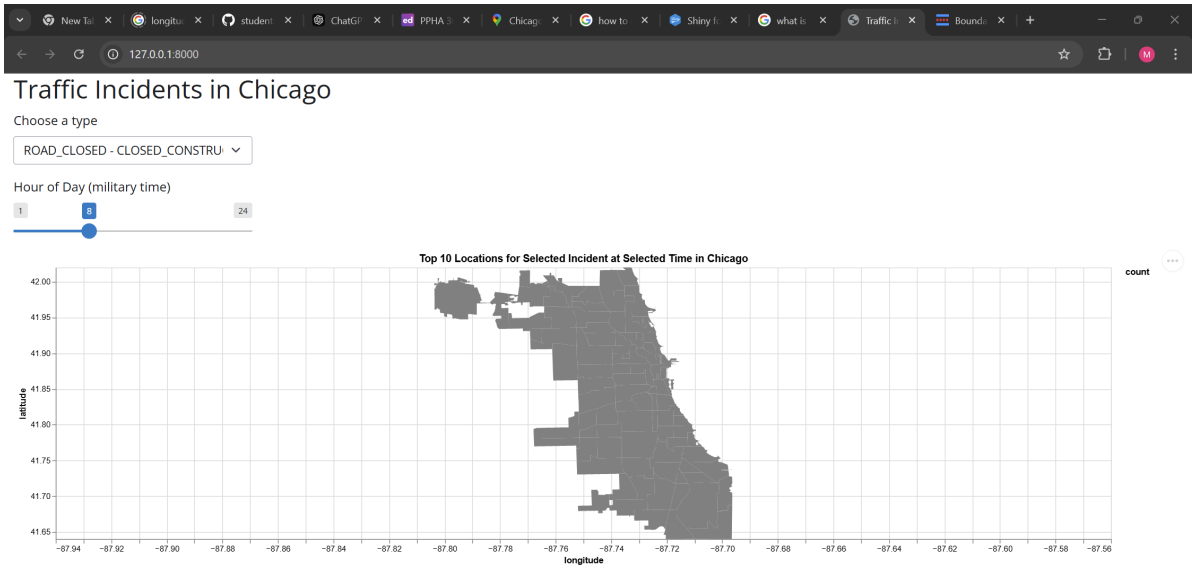
b. I'm assuming you want 3 screenshots;one of each plot created above:

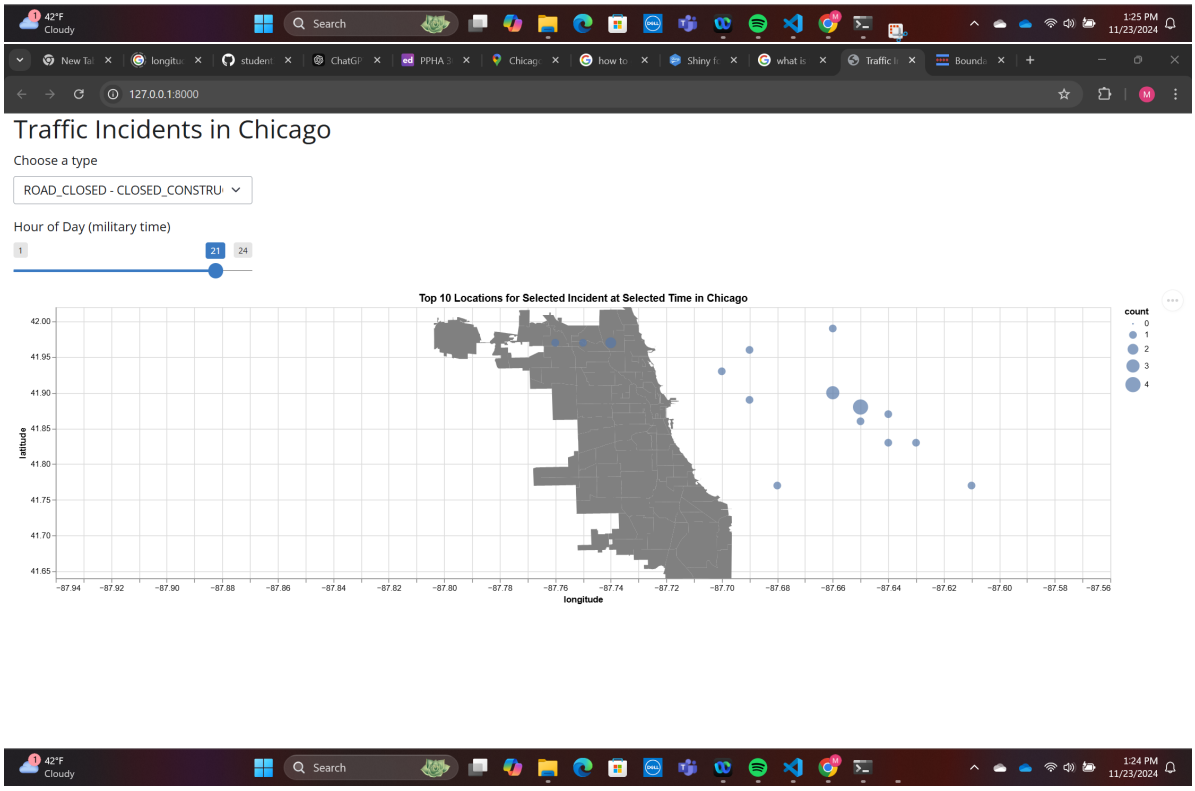

127.0.0.1:8000
Traffic Incidents in Chicago
Choose a type
JAM - HEAVY_TRAFFIC
Hour of Day (military time)
1
6
24
Top 10 Locations for Selected Incident at Selected Time in Chicago
latitude
longitude
count
0
5
10
15


42°F
Cloudy
Search
1:16 PM
11/23/2024
127.0.0.1:8000
Traffic Incidents in Chicago
Choose a type
JAM - HEAVY_TRAFFIC
Hour of Day (military time)
1
12
24
Top 10 Locations for Selected Incident at Selected Time in Chicago
latitude
longitude
count
0
50
100
150
200
250
42°F
Cloudy
Search
1:17 PM
11/23/2024


127.0.0.1:8000
Traffic Incidents in Chicago
Choose a type
JAM - HEAVY_TRAFFIC
Hour of Day (military time)
1
18
24
Top 10 Locations for Selected Incident at Selected Time in Chicago
latitude
longitude
count
0
50
100
150
200
250
300
42°F
Cloudy
Search
1:18 PM
11/23/2024

c.





We'll need to change the type-subtype combo to one that aligns with road construction

Based upon the provided screenshots it appears that construction more often occurs in the evening rather than morning. I chose 8am because that's a popular morning commute time, and 9pm because it seemed like that was the time when most construction events was going on from exploring the dashboard. I would caution however that these dispositions indicate road closures due to construction, and may not reflect all construction, because lots of construction occurs even without a road closure on major throughways such as I90/94, 290, and DLSD.

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

1.

a. If I'm understanding the prompt correctly, I think this would be a bad idea because once we collapse the data down to an aggregation including time range it would restrict the users ability to adjust the range. So we need something that dynamically calculates the number of incidents at each location.

b.

```
#create plot with top 10 incidents by range of hour from 6-9am

range_chart=alt.Chart(time_waze_df).mark_circle().encode(
    alt.X(
        "longitude",
        scale=alt.Scale(domain=[min_longitude, max_longitude])
    ),
    alt.Y(
        "latitude",
        scale=alt.Scale(domain=[min_latitude, max_latitude])
    ),
    size="count",
    tooltip=["hour", "latitude", "longitude", "count"]
).transform_filter(
    (alt.datum.type == "JAM") &
    (alt.datum.updated_subtype == "HEAVY_TRAFFIC") &
    ((alt.datum.hour == "06:00") |
     (alt.datum.hour == "07:00") |
     (alt.datum.hour == "08:00") |
     (alt.datum.hour == "09:00"))  # Filter for the specific hours
).transform_window(
    rank='rank(count)',  # Rank by count across all the hours
```

```
    sort=[alt.SortField('count', order='descending')]  # Sort by count in
 ↪  descending order
).transform_filter(
    alt.datum.rank <= 10  # Only show the top 10 locations based on count
).properties(title="Top 10 Locations for Heavy Traffic Jams in Chicago,
 ↪  6am-9am")

base_map=alt.Chart(geo_data).mark_geoshape().encode(
    fill=alt.value("grey")
).project(type="identity", reflectY=True)
combined_chart=base_map+range_chart
combined_chart.show()
```

`alt.LayerChart(...)`

2.

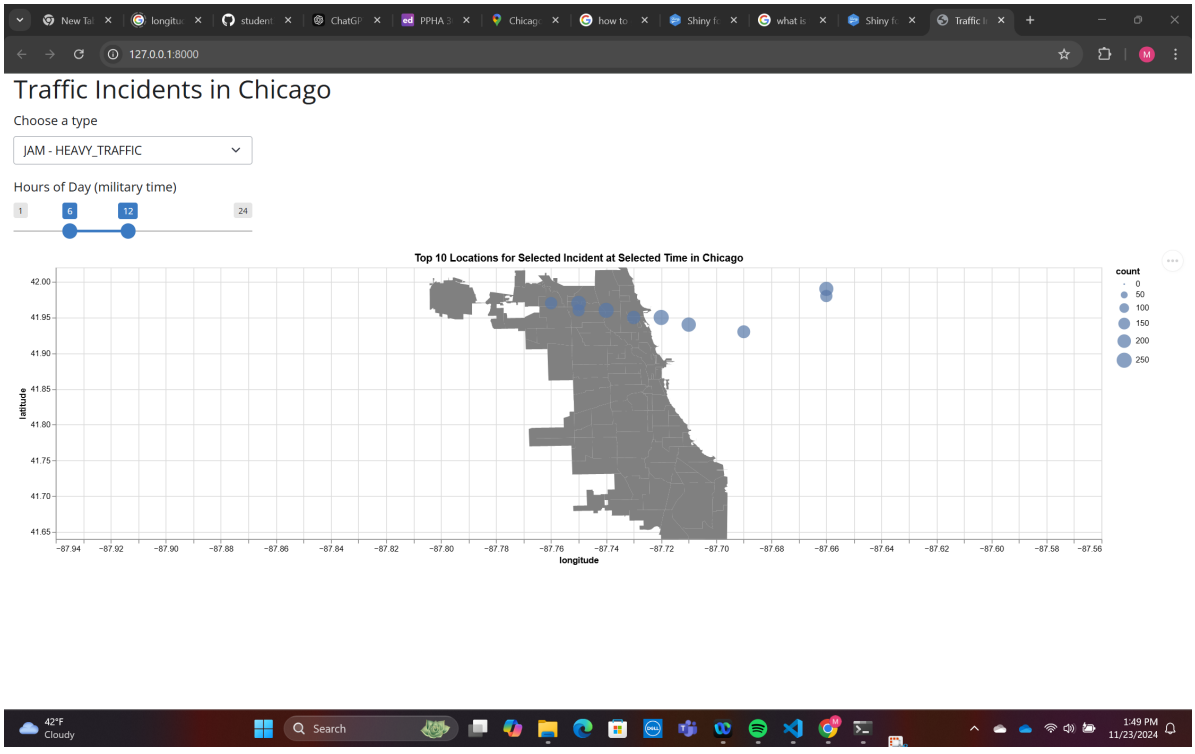


Figure 6: Initial slider with plot

a.

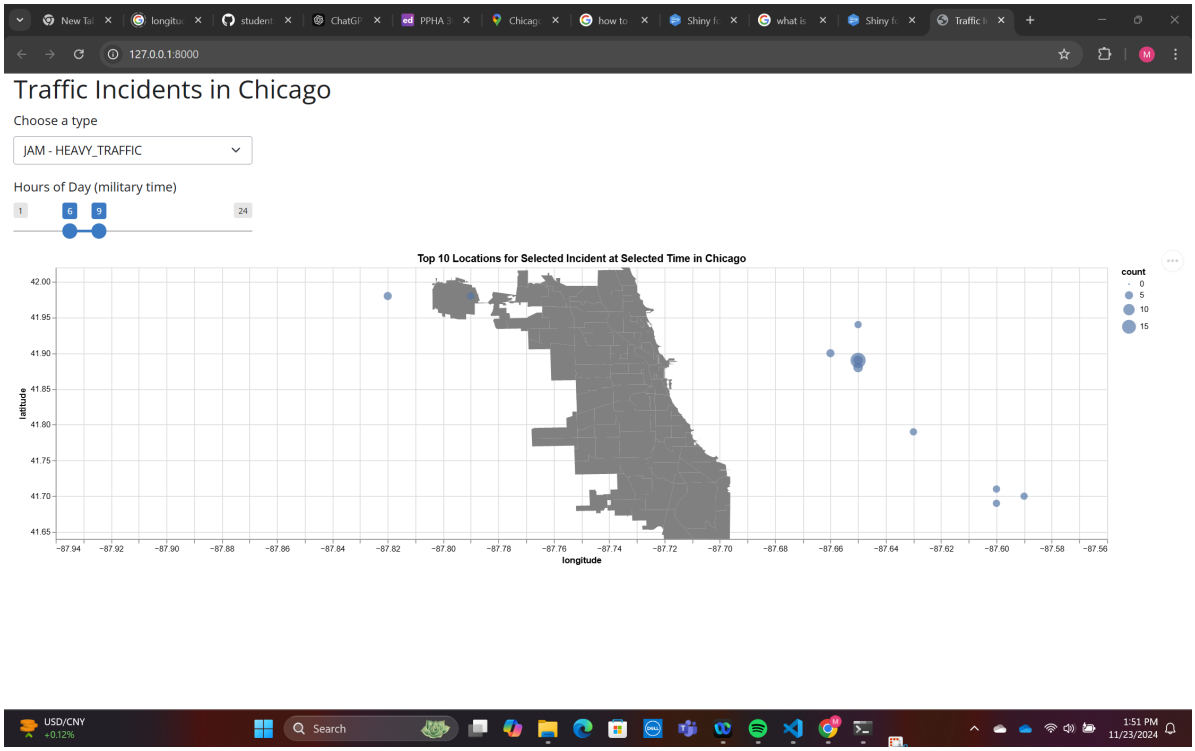


Figure 7: 6-9am heavy traffic jams

b.

3.

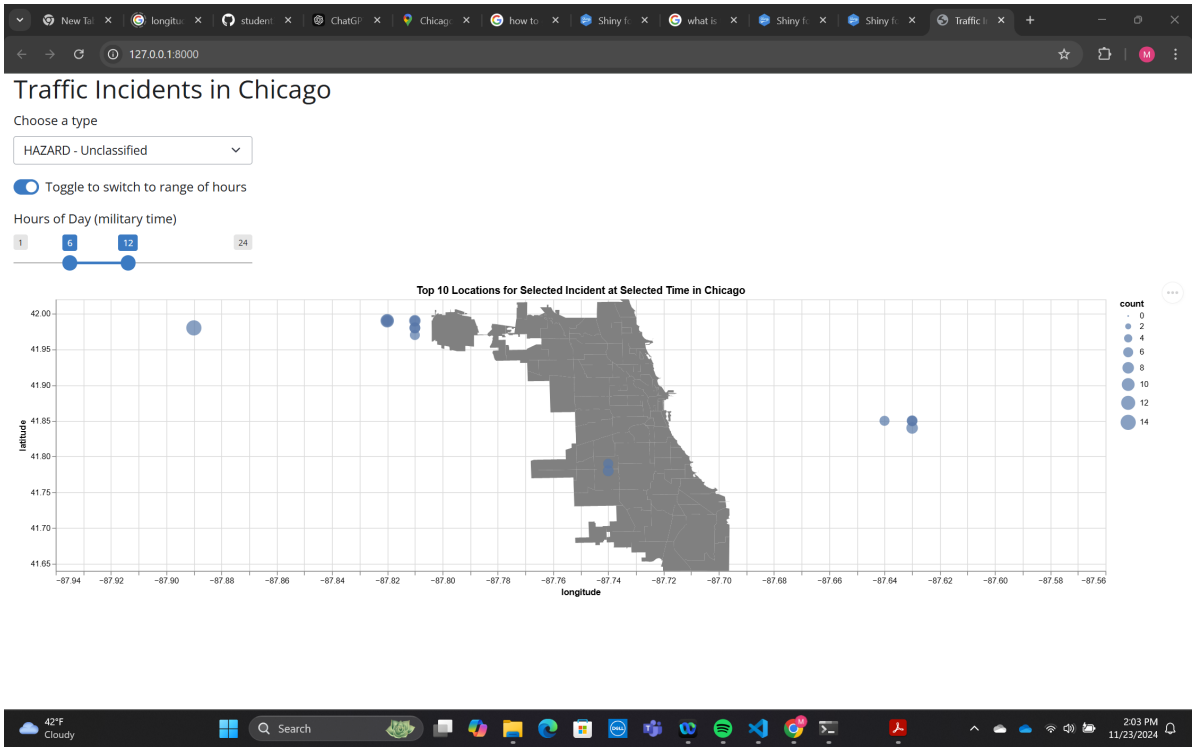
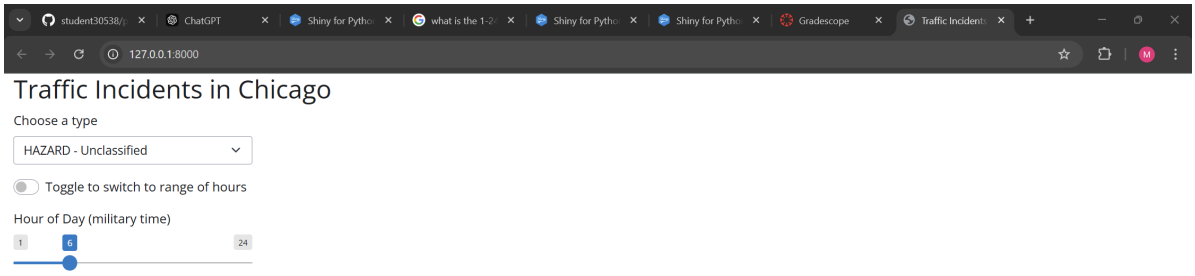


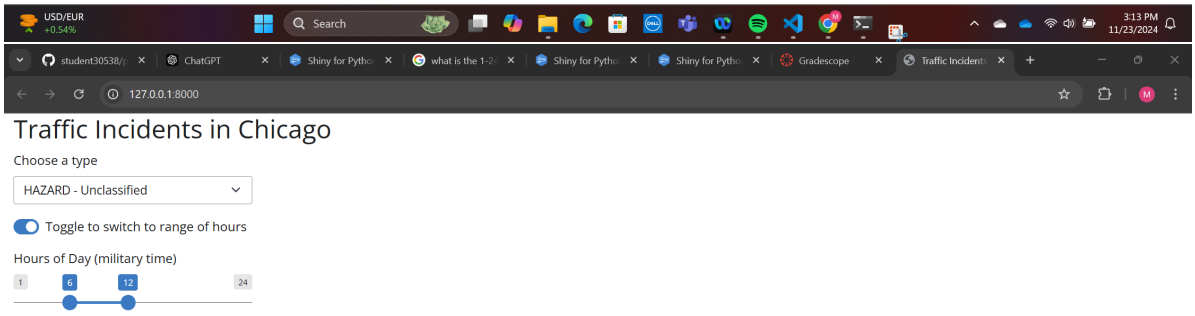
Figure 8: nonfunction toggle button

a.

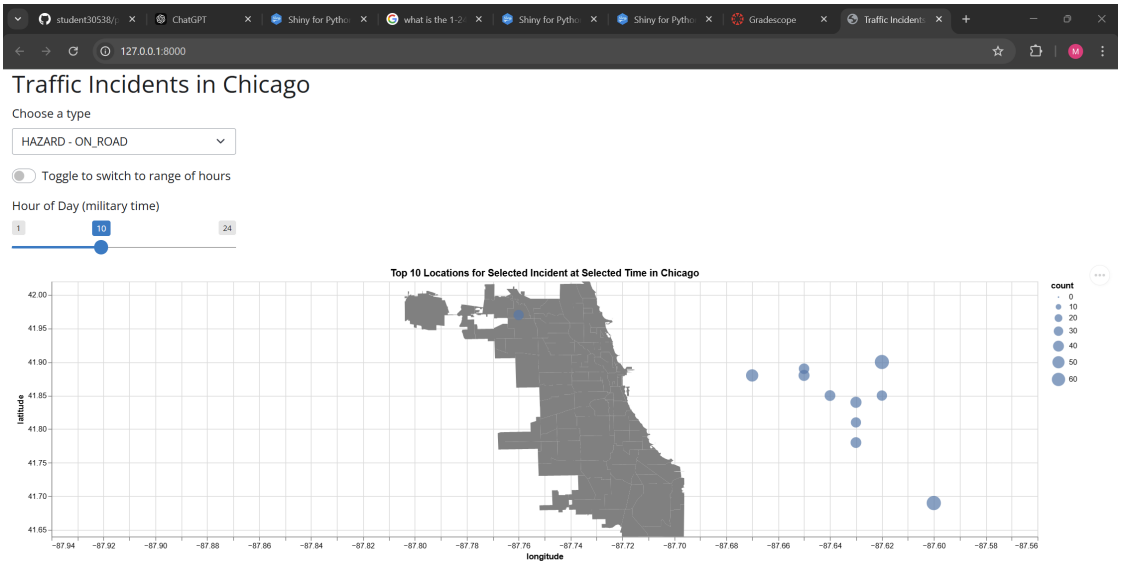
The possible values for the input_switch are True or False according to the documentation shared.

b.

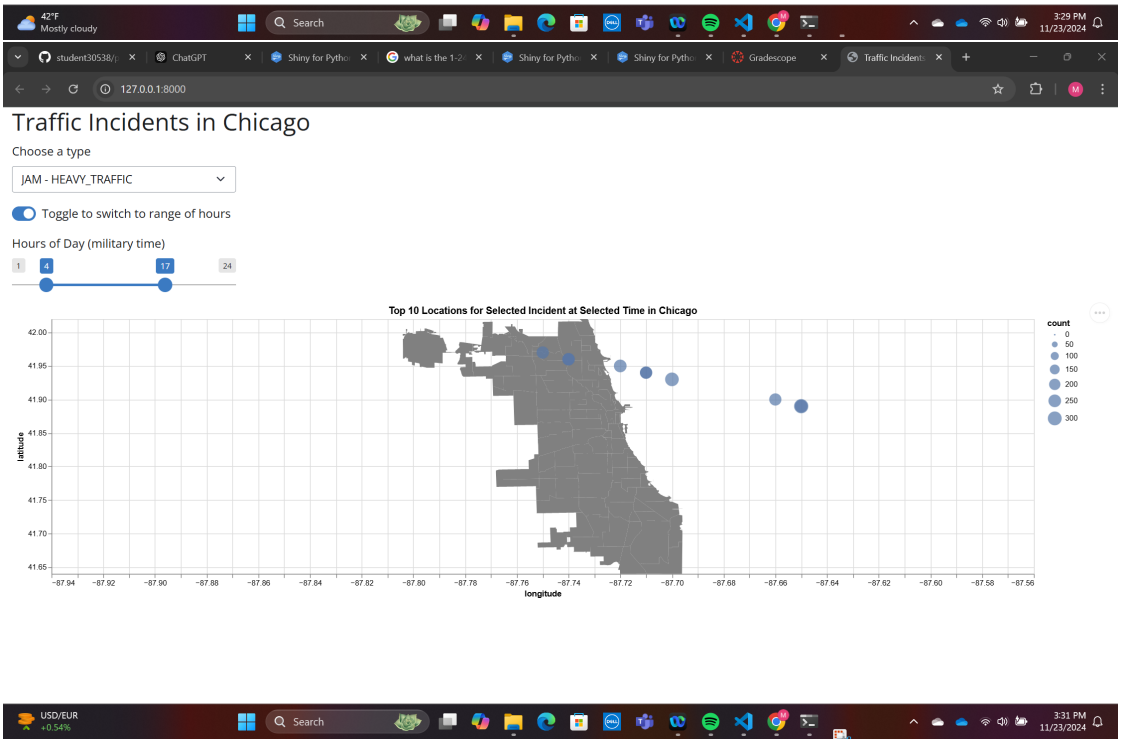

127.0.0.1:8000
Traffic Incidents in Chicago
Choose a type
HAZARD - Unclassified
Toggle to switch to range of hours
Hour of Day (military time)
1
6
24


USD/EUR
+0.54%
Search
3:13 PM
11/23/2024
127.0.0.1:8000
Traffic Incidents in Chicago
Choose a type
HAZARD - Unclassified
Toggle to switch to range of hours
Hours of Day (military time)
1
6
12
24


USD/EUR
+0.54%
Search
3:13 PM
11/23/2024



c.



d. I would set the plot to contain all the times of day, and then use altair to create a transformation that categorizes all morning hours as "morning" and all evening hours as "evening". I'd need to filter all other hours that don't fit those two categories. Then in

the color layer I'd give morning red, evening blue. Then I'd need to change the fill to be none on the markcircle layer.