Mid_report_fix

May 11, 2023

1 PIC16B Traffic Visualization Project

Wonjoon Choi, Kelly Chung, Gabriel Sison

1.0.1 Goal: Create a real-time, interactive traffic visualization of the US displaying incident info, position, and effect on traffic speed.

So far we've created methods to pull incident data from the MapQuest API over a specified area, convert this into a dataframe, and store it in an SQL database. Separately we also have some progress on a functional webapp which we plan to use to contain the project.

[ ]:

```
import plotly.io as pio
pio.renderers.default = "notebook+pdf"
pio.renderers
```

[ ]:

```
Renderers configuration
-----------------------
    Default renderer: 'notebook+pdf'
    Available renderers:
        ['plotly_mimetype', 'jupyterlab', 'nteract', 'vscode',
         'notebook', 'notebook_connected', 'kaggle', 'azure', 'colab',
         'cocalc', 'databricks', 'json', 'png', 'jpeg', 'jpg', 'svg',
         'pdf', 'browser', 'firefox', 'chrome', 'chromium', 'iframe',
         'iframe_connected', 'sphinx_gallery', 'sphinx_gallery_png']
```

2 Old Version of Database Code:

[ ]:

```
import requests
import json
import sqlite3
import time
import folium
import pandas as pd

#create our own database
conn = sqlite3.connect('traffic_data.db')
c = conn.cursor()
#creates a table anmed "incidents" if it doesn't already exist
```

```python
#the table has columns for incdient details such as ID,type, severity, description, latitude, longitude
c.execute("CREATE TABLE IF NOT EXISTS incidents (id INTEGER PRIMARY KEY, type TEXT, severity INTEGER, description TEXT, lat REAL, lng REAL)")
conn.commit()
conn.close()

def insert_data(data):
    #defined to insert incident data inot the "incidents" table
    conn = sqlite3.connect('traffic_data.db')
    c = conn.cursor()
    #takes a list of data tuples as input and insert multiple rows into the table at once
    c.executemany("INSERT INTO incidents (id, type, severity, description, lat, lng) VALUES (?, ?, ?, ?, ?, ?)", data)
    conn.commit()
    conn.close()

def get_traffic_data(bbox):
    #retrieves traffic incident data from MapQuest Traffic API
    key = ""
    response = requests.get(f"https://www.mapquestapi.com/traffic/v2/incidents?key={key}&boundingBox={bbox}&filters=congestion,incidents,construction,event")
    data = response.json()
    return data

def store_traffic_data():
    #store traffic incident data in the database.
    bbox_step = 0.1  # Bbox step size for iterating over the U.S.
    bbox_range = {
        # Starting latitude for bbox
        "lat_start": 24.396308,  #southernmost
        # Ending latitude for bbox
        "lat_end": 49.384358,    #northernmost
        # Starting longitude for bbox
        "lng_start": -125.000000,  #westernmost
        # Ending longitude for bbox
        "lng_end": -66.934570    #easternmost
    }
    create_table()
    bbox = {
        "lat_start": bbox_range["lat_start"],
        "lat_end": bbox_range["lat_start"] + bbox_step,
        "lng_start": bbox_range["lng_start"],
        "lng_end": bbox_range["lng_start"] + bbox_step
    }
    page = 1
    # a loop that continues until the latitude of the current
    #bounding box exceeds the northernmost latitude of the bbox_range.
    while bbox["lat_start"] <= bbox_range["lat_end"]:
        #to fetch traffic incident data for the current bounding box
        response_data =␣
↪get_traffic_data(f"{bbox['lat_start']},{bbox['lng_start']},{bbox['lat_end']},{bbox['lng_end
        incidents = response_data.get("incidents")
        if not incidents:
            break
            #If there are no incidents, it breaks out of the loop, assuming␣
↪that there is no more data to retrieve.
        data = []
        #extracting relevant information from each incident and storing it as a␣
↪tuple in the data list
        for incident in incidents:
            incident_id = incident.get("id")
            incident_type = incident.get("type")
            incident_severity = incident.get("severity")
            incident_description = incident.get("shortDesc")
            incident_lat = incident.get("lat")
            incident_lng = incident.get("lng")
            data.append((incident_id, incident_type, incident_severity,␣
↪incident_description, incident_lat, incident_lng))
        #inserting the incident data into the database.
        insert_data(data)
        print(f"Page {page} processed.")
        page += 1
        #the longitude values of the bounding box are updated by adding the␣
↪bbox_step
        #value to both the starting and ending longitudes
        bbox["lng_start"] += bbox_step
        bbox["lng_end"] += bbox_step
        #If the updated longitude exceeds the easternmost longitude of the␣
↪bbox_range,
        #the latitude values are updated, and the longitude values are reset to␣
↪the starting values
        if bbox["lng_start"] > bbox_range["lng_end"]:
            bbox["lat_start"] += bbox_step
            bbox["lat_end"] += bbox_step
            bbox["lng_start"] = bbox_range["lng_start"]
            bbox["lng_end"] = bbox_range["lng_start"] + bbox_step
        time.sleep(1)  # Sleep for 1 second to avoid hitting API rate limits

def get_incidents_in_area(bbox):
    conn = sqlite3.connect('traffic_data.db')
    c = conn.cursor()
    c.execute("SELECT * FROM incidents WHERE lat BETWEEN ? AND ? AND lng␣
 ↪BETWEEN ? AND ?", bbox)
    incidents = c.fetchall()
    conn.close()
    return incidents

def display_map_with_incidents(incidents):
    # Create an empty map centered around the first incident

    if incidents:
        center_lat, center_lng = incidents[0][4], incidents[0][5]
        #[0]: incident; [4]: latitude; [5]: longitude
    else:
        center_lat, center_lng = 0, 0
    map_traffic = folium.Map(location=[center_lat, center_lng], zoom_start=10)

    # Add markers for each incident
    for incident in incidents:
        #the location parameter set to [incident_lat, incident_lng],
        #representing the coordinates of the incident
        incident_lat, incident_lng = incident[4], incident[5]
        marker = folium.Marker(location=[incident_lat, incident_lng])
        marker.add_to(map_traffic)

    return map_traffic
```

3 Modified for dataframes (and plotlyexpress)

[ ]:

```
import requests
import json
import sqlite3
import time
import folium
import pandas as pd
from plotly import express as px

#create our own database
conn = sqlite3.connect('traffic_data.db')

def insert_data(conn, data):
    """Inserts new traffic data into incidents table in database"""

    if len(data) > 0:
        data.to_sql("incidents", conn, if_exists="append", index=False)

def get_traffic_data(bbox):
    """retrieves traffic incident data in a given bounding box from MapQuest␣
 ↪Traffic API"""

    # key = ""
    response = requests.get(f"https://www.mapquestapi.com/traffic/v2/incidents?
 ↪key={key}&boundingBox={bbox}&filters=congestion,incidents,construction,event")
    data = pd.DataFrame(response.json()["incidents"])
    if len(data) > 0:
        data = data[['id', 'type', 'severity', 'shortDesc', 'lat', 'lng']]
    return data

def store_traffic_data(conn, lat_start, lat_end, lng_start, lng_end):
    """Iterates over a given area and updates traffic incident database by␣
 ↪MapQuest API calls"""

    bbox_step = 1  # Bbox step size for iterating over the U.S.
    bbox_range = {
        # Starting latitude for bbox
        "lat_start": lat_start, # 24.396308,  #southernmost
        # Ending latitude for bbox
        "lat_end": lat_end, # 49.384358,    #northernmost
        # Starting longitude for bbox
        "lng_start": lng_start, # -125.000000,  #westernmost
        # Ending longitude for bbox
        "lng_end": lng_end # -66.934570    #easternmost
    }
    # create_table()
    bbox = {
        "lat_start": bbox_range["lat_start"],
        "lat_end": bbox_range["lat_start"] + bbox_step,
        "lng_start": bbox_range["lng_start"],
        "lng_end": bbox_range["lng_start"] + bbox_step
    }
    page = 1
    # a loop that continues until the latitude of the current
    #bounding box exceeds the northernmost latitude of the bbox_range.
    while bbox["lat_start"] <= bbox_range["lat_end"]:
        #to fetch traffic incident data for the current bounding box
        data =␣
 ↪get_traffic_data(f"{bbox['lat_start']},{bbox['lng_start']},{bbox['lat_end']},{bbox['lng_end
        insert_data(conn, data)
        print(f"Page {page} processed.")
        page += 1
        #the longitude values of the bounding box are updated by adding the␣
 ↪bbox_step
        #value to both the starting and ending longitudes
        bbox["lng_start"] += bbox_step
        bbox["lng_end"] += bbox_step
        #If the updated longitude exceeds the easternmost longitude of the bbox_range,
        #the latitude values are updated, and the longitude values are reset to the starting values
        if bbox["lng_start"] > bbox_range["lng_end"]:
            bbox["lat_start"] += bbox_step
            bbox["lat_end"] += bbox_step
            bbox["lng_start"] = bbox_range["lng_start"]
            bbox["lng_end"] = bbox_range["lng_start"] + bbox_step
        time.sleep(1)  # Sleep for 1 second to avoid hitting API rate limits

def get_incidents_in_area(conn, bbox):
    """Retrieves incidents within the given bounding box from database"""

    min_lat, max_lat = bbox[0], bbox[1]
    min_lng, max_lng = bbox[2], bbox[3]
    cmd=\
        f"""
            SELECT * FROM incidents
            WHERE lat BETWEEN {min_lat} AND {max_lat}
            AND lng BETWEEN {min_lng} AND {max_lng}
        """
    print(cmd)
    df = pd.read_sql_query(cmd, conn)
    return df

def display_map_with_incidents(incidents, **kwargs):
    """Creates a plotly map of traffic incidents"""

    # try:
    #     center_lat, center_lng = incidents['lat'][0], incidents['lng'][0]
    # except:
    #     print("data frame error?")
    #     center_lat, center_lng = 0, 0

    fig = px.scatter_mapbox(incidents,
                            lat="lat",
                            lon="lng",
                            color="severity",
                            hover_name="id",
                            hover_data=['shortDesc', 'type'],
                            mapbox_style="open-street-map",
                            **kwargs)
    fig.update_layout(margin={"r":0, "l":0,"b":0,"t":0})
    return fig
```

3.0.1 Issues:

The main two problems we have so far are the API call limit for MapQuest and (related to that) the area limit for requesting traffic data within a given set of latitude/longitude boundaries. This leads to the problem that to update the traffic incident data for a large area (such as the whole state of Nevada), we are forced to perform multiple api calls limited to square areas of 1 degree latitude by 1 degree longitude. In the case of Nevada, this required 49 separate API calls, which can add up quickly and might make the 15000 monthly API call limit a problem.

3.0.2 Workaround(s):

To address this issue we can limit the scope of the project from the entire US to California (or another state) specifically. This can massively reduced the amount of API calls necessary as well as the time required to update the database. Moreover, we can try using other API's (like TomTom) in conjunction with MapQuest to raise our call limit.

3.0.3 Creating a dataframe of state coordinate bounds:

```
[ ]: state_bounds = pd.read_csv("https://gist.githubusercontent.com/a8dx/
      ↪2340f9527af64f8ef8439366de981168/raw/
      ↪81d876daea10eab5c2675811c39bcd18a79a9212/US_State_Bounding_Boxes.csv")
     state_bounds = state_bounds[['NAME', 'STUSPS', 'xmin', 'ymin', 'xmax', 'ymax']]
     state_bounds = state_bounds.rename(columns={"xmin": "min_lng", "xmax":␣
      ↪"max_lng", "ymin": "min_lat", "ymax": "max_lat"})
     state_bounds.head(5)
```

```
[ ]:              NAME STUSPS     min_lng    min_lat     max_lng    max_lat
     0          Alabama     AL  -88.473227  30.223334  -84.889080  35.008028
     1           Alaska     AK -179.148909  51.214183  179.778470  71.365162
     2   American Samoa     AS -171.089874 -14.548699 -168.143300 -11.046934
     3          Arizona     AZ -114.816510  31.332177 -109.045223  37.004260
     4         Arkansas     AR  -94.617919  33.004106  -89.644395  36.499600
```

3.0.4 Extracting the coordinate limits for Massachusetts:

```
[ ]: ma_minlat = state_bounds.loc[state_bounds['STUSPS']=="MA"]['min_lat'].values[0]
     ma_maxlat = state_bounds.loc[state_bounds['STUSPS']=="MA"]['max_lat'].values[0]
     ma_minlng = state_bounds.loc[state_bounds['STUSPS']=="MA"]['min_lng'].values[0]
     ma_maxlng = state_bounds.loc[state_bounds['STUSPS']=="MA"]['max_lng'].values[0]
     ma_minlat, ma_maxlat, ma_minlng, ma_maxlng
```

```
[ ]: (41.237964, 42.886589, -73.508142, -69.928393)
```

3.0.5 Updating the traffic incident database:

```
[ ]: import credentials as cred
     conn = sqlite3.connect('traffic_data.db')
     key = cred.mapquest_api_key
```

```
store_traffic_data(conn=conn, lat_start=ma_minlat, lat_end=ma_maxlat,␣
 ↪lng_start=ma_minlng, lng_end=ma_maxlng)
```

```
Page 1 processed.
Page 2 processed.
Page 3 processed.
Page 4 processed.
Page 5 processed.
Page 6 processed.
Page 7 processed.
Page 8 processed.
```

3.0.6 Retrieving the incidents in Massachusetts from our database:

[ ]:
```
# bbox = (39, 40, -122, -121)
bbox = (ma_minlat, ma_maxlat, ma_minlng, ma_maxlng)

conn = sqlite3.connect('traffic_data.db')

incidents = get_incidents_in_area(conn, bbox)
incidents.head(5)
```

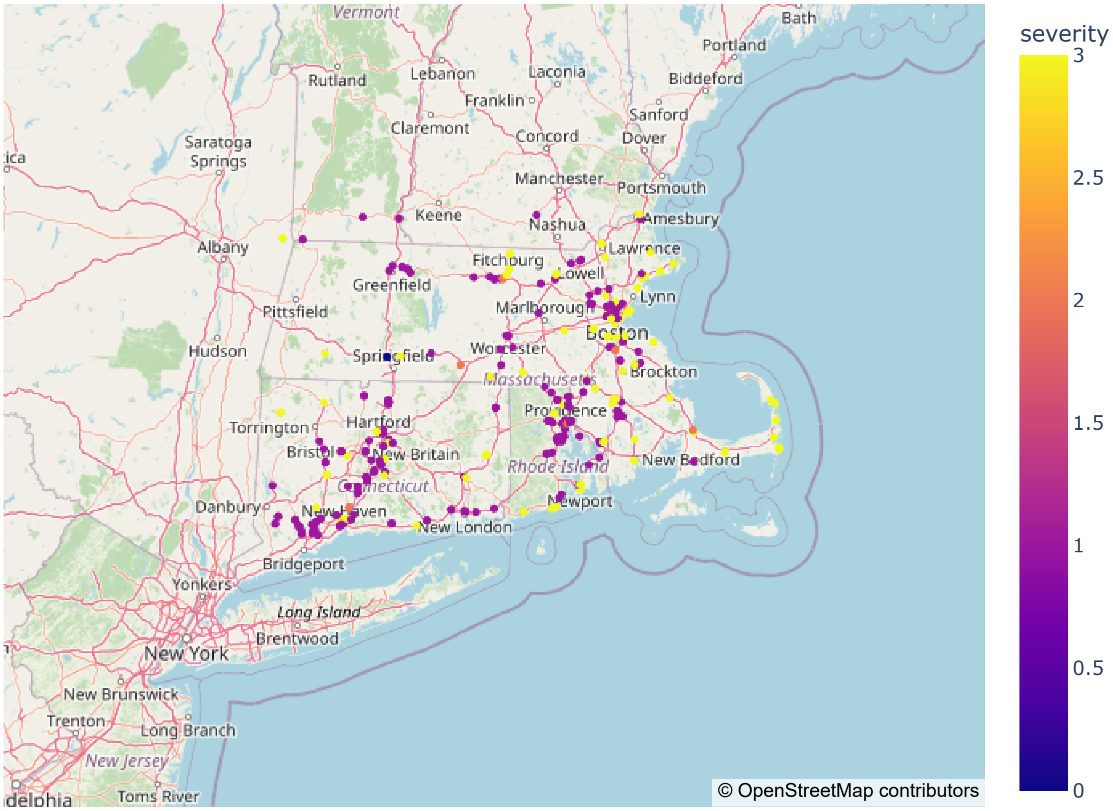
```
            SELECT * FROM incidents
            WHERE lat BETWEEN 41.237964 AND 42.886589
            AND lng BETWEEN -73.508142 AND -69.928393
```

[ ]:
```
                    id  type  severity                      shortDesc
0  4405418776464707010     1         1              Construction work  \
1   533119548599105528     1         1  Bridge maintenance operations
2   505275340163207417     1         1              Road construction
3  4453299463648210620     1         1              Construction work
4  2509738909818101436     1         1              Road construction

        lat       lng
0  41.53979 -72.77216
1  41.31785 -72.90188
2  41.25740 -73.21953
3  41.56791 -72.65059
4  41.30297 -72.60386
```

3.0.7 Displaying the traffic incidents as a plotly scatter mapbox:

[ ]:
```
map_with_incidents = display_map_with_incidents(incidents, zoom=6)
map_with_incidents.show()
```


severity
3
2.5
2
1.5
1
0.5
0
Vermont
Bath
Portland
Rutland
Lebanon
Laconia
Biddeford
Franklin
Sanford
Claremont
Concord
Dover
Saratoga Springs
Manchester
Portsmouth
Keene
Amesbury
Nashua
Albany
Lawrence
Fitchburg
Greenfield
Lowell
Lynn
Pittsfield
Marlborough
Boston
Hudson
Springfield
Worcester
Brockton
Massachusetts
Providence
Torrington
Hartford
Bristol
New Britain
New Bedford
Rhode Island
Connecticut
Danbury
New Haven
Newport
New London
Bridgeport
Yonkers
Long Island
Brentwood
New York
New Brunswick
Trenton
Long Branch
New Jersey
Toms River
© OpenStreetMap contributors

```
[ ]:
```