New York Uber data collected by the City Taxi and Limousine Commission (TLC)

Dataset

From https://github.com/fivethirtyeight/uber-tlc-foil-response/tree/master/uber-trip-data. I was specifically told to focus on the files uber-raw-data-<month>14.csv, which track anonymized pickup locations for drivers in NYC.

For this project, I was instructed to transform the data such that half of the data become pickup locations, and the other half drop off. Then develop a website that displays information about the dataset.

Project Code

The webpage can be accessed at http://www.redootv.com, and my project code is located at https://github.com/jried31/google-maps-uber-data.

Client Requirements

Required

- Show a map of relevant data **(COMPLETED)**
- Be able to draw an arbitrary shape or geofence on the map **(COMPLETED)**
- filter data that is fully contained (begins and ends) within the shape **(COMPLETED)**
- Be able to answer where the top pickups are for a given shape **(COMPLETED)**
- Be hosted on Heroku, EC2, or somewhere we can access it by visiting a URL. **(COMPLETED)**

Extra Credit

- Create a heatmap showing trip density for any shape **(COMPLETED)**
- Filters for time of day, or day/night **(NOT COMPLETED)**
- A page of interesting statistics about the current shape **(UNCLEAR REQUIREMENT)**
- Show popular routes **(UNCLEAR REQUIREMENT)**
- Show areas of high and low speeds **(UNCLEAR REQUIREMENT)**

Assumptions & Limitations about the Requirements

1. A page of interesting statistics about the current shape
    a. Statement is too broad, making it hard for me to clearly interpret.
2. Show popular routes
    a. Actual route information has not been provided in the dataset. Therefore, I'm unable to show this information.
    b. In lieu of this, I created clickable markers that compute the route from the pickup to the corresponding drop off location. The routes are shown in green.

Completed Requirements

1. Show map of relevant data
    a. Webpage displays relevant data (limit of the 1st 1000 rows)
2. Draw an arbitrary shape and filter data fully contained w/in the shape
    a. Lines drawable via the Google Maps palette.
    b. **Any shape that is 4 or more sides is permissible. Triangles will not work, and I did not have enough time to investigate.**
3. Answer where the top pickups are for a given shape
    a. Heatmap and cluster markers show popular pickup locations

Design Decisions and Tradeoffs

1. **When the http://redootv.com page first loads, an initial query is performed limiting only to 1000 rows in order to minimize liklihood of memory issues on the server and browser.**
2. **The webpage allows for multiple polygons to be drawn on the view. However, data will be shown only for the most recent polygon drawn on the webpage. Next steps will involve extending functionality for multiple polygons.**
    a. **Polygons are removed from the map by selecting the polygon you want to delete and pressing the "Remove Shape" button.**
3. Used a hashmap to quickly reference pickup and dropoff locations when a pickup marker is clicked in order to speed up the calculation references
4. Used the haversine formula to calculate the straight line distance between the pickup/dropoff points. I added Google Map travel routes later in the project and didn't have time to update with the route distance provided by Google
5. Plugins such as "clock", "datetimepicker" and other boostrap view plugins are loaded in a the file pluginConfig.js. I chose this to separate the main logic from initializers
6. The Node.js file system is divided in a way that separates the model, service and API layers using separate directories. Advantage is modularized code.

Implementation Details

Database: Postgresql with PostGIS extension
Backend: Node.js, Express
Front End: HTML, Javascript, JQuery

Assumptions

The requirements state that the final data (ie: table) should have the following columns:

- Date/Time
- Lat
- Lon
- DropoffLat (taken from another row's 'Lat')
- DropoffLon (taken from another row's 'Lon')

I added two extra columns to represent the dropoff/pickup locations as a GEOMETRY point. PostGIS requires this in order to perform geospatial queries.

Step 1: Created a table named "Rides"

```
CREATE TABLE public.Rides
(
  id integer NOT NULL DEFAULT nextval('ride_data_id_seq'::regclass),
  datetime timestamp without time zone,
  lat numeric,
  lon numeric,
  base text,
  geog geometry(Point,4326)
  CONSTRAINT ride_data_pkey PRIMARY KEY (id)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE public.Rides
  OWNER TO rhbqgfvvastkht;
```

Step 2: Import the uber pickups dataset from the CSV file to Rides table

```
COPY Rides(datetime,lat,lon) FROM '/tmp/uber/uber-raw-data-apr14.csv' DELIMITER ',' CSV
HEADER;
```

```
COPY Rides(datetime,lat,lon) FROM '/tmp/uber/uber-raw-data-may14.csv' DELIMITER ',' CSV HEADER;
COPY Rides(datetime,lat,lon) FROM '/tmp/uber/uber-raw-data-jun14.csv' DELIMITER ',' CSV HEADER;
COPY Rides(datetime,lat,lon) FROM '/tmp/uber/uber-raw-data-jul14.csv' DELIMITER ',' CSV HEADER;
COPY Rides(datetime,lat,lon) FROM '/tmp/uber/uber-raw-data-aug14.csv' DELIMITER ',' CSV HEADER;
COPY Rides(datetime,lat,lon) FROM '/tmp/uber/uber-raw-data-sep14.csv' DELIMITER ',' CSV HEADER;
```

Step 3: Fill in the geog column representing the “GEOMETRY” represnetation of the pickup latitude/longitude points (as required by PostGIS for geospatial queries).

```
UPDATE "Rides" SET geog = ST_SetSRID(ST_MakePoint(lon, lat), 4326)::geometry;
```

Step 4: Transform the dataset by segmenting half of the rows to represent pickups and drop offs.

This stored procedure assigns even #'s to pickups and odd #'s to dropoff's. In the case where the total number of recorded pickups is an odd number, I select the MIN(pickups, dropoffs) and partition accordingly resulting in one row not assigned to a ride.

Since there are over 500k recorded pickups, a SELECT … RANDOM() query will not be performant to partition the dataset. Therefore, I use GENERATE_SERIES to generate the id’s of the rows to be segmented accordingly and insert into the Taxi table.

```
CREATE OR REPLACE FUNCTION public.process_rides()
  RETURNS void AS
$BODY$
DECLARE
        dataSize INTEGER;
        debugVariable INTEGER := 0;
        pickups INTEGER[];
        dropoffs INTEGER[];
BEGIN
        -- Get last (largest) row number from the rides_table
        SELECT MAX(id) FROM "Rides" INTO dataSize;
        --raise notice 'Max Rows: %', dataSize;

        -- Assignment of pickup/dropoffs (pickups = Odd rows | dropoffs = even rows)
```

```
	pickups := array(SELECT num FROM GENERATE_SERIES(2, dataSize,2) AS s(num)
ORDER BY RANDOM());
	--raise notice 'Pickups: %', pickups;
	dropoffs := array(SELECT num FROM GENERATE_SERIES(1, dataSize,2) AS s(num)
ORDER BY RANDOM());
	--raise notice 'Dropoffs: %', dropoffs;

	SELECT array_length(pickups,1) INTO debugVariable;
	--raise notice 'Pickups: %', debugVariable;
	SELECT array_length(dropoffs,1) INTO debugVariable;
	--raise notice 'Dropoffs: %', debugVariable;

	-- Build the new table
	DELETE FROM "Taxi";
	INSERT INTO "Taxi"(pickup_time,lat_pickup,lon_pickup,lat_dropoff,lon_dropoff,
dropoff_time)
	SELECT A.pickup_time as pickup_time,A.lat_pickup as lat_pickup,A.lon_pickup as
lon_pickup,B.lat_dropoff as lat_dropoff,B.lon_dropoff as lon_dropoff, B.dropoff_time as
dropoff_time FROM
	(
		SELECT row_number() OVER() AS colId, datetime AS pickup_time, lat AS
lat_pickup, lon AS lon_pickup FROM "Rides" WHERE id = ANY(pickups)
	) as A
	INNER JOIN
	(
		SELECT row_number() OVER() AS colId, datetime AS dropoff_time, lat AS
lat_dropoff, lon AS lon_dropoff FROM "Rides" WHERE id = ANY(dropoffs)
	) AS B ON A.colId = B.colId;

	-- Assign the geometry columns for geospatial queries
	UPDATE "Taxi" SET
	pickup_point = ST_SetSRID(ST_MakePoint(lon_pickup, lat_pickup), 4326)::geography,
	dropoff_point = ST_SetSRID(ST_MakePoint(lon_dropoff, lat_dropoff), 4326)::geography;
END;
$BODY$
  LANGUAGE plpgsql;
  -- Invoke the stored proceedure
select * from process_rides();
```

Step 5: Build the backend in Node.js.

The service layer was implemented in Node.js using the Express framework. The file structure is segmented in a way that separates the routes in a folder named “routes”, the database access layer in the “apis” folder, and the data model object representations in the “models” folder.

Since we perform raw geospatial queries in this project and there’s only one end-point, the “models” folder is omitted for this project