EXPLORING NYC BY THE NUMBERS

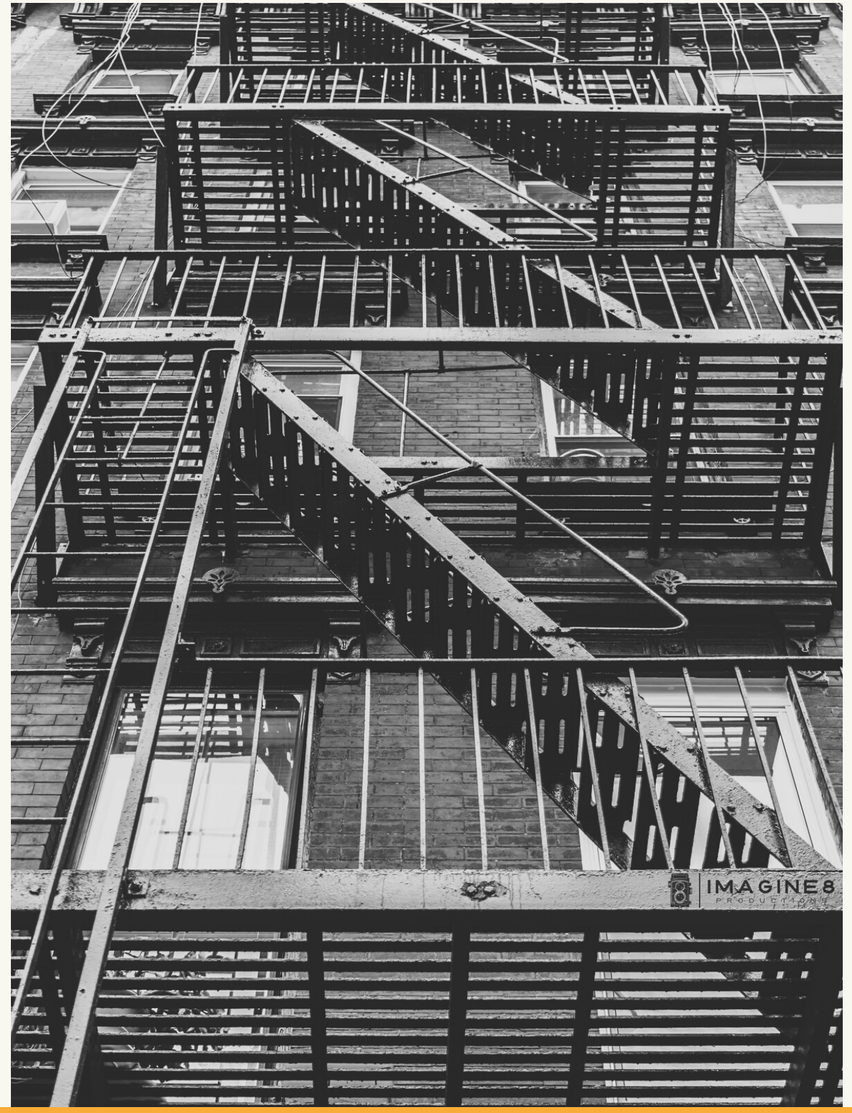
Presented by Jeremy Jang, Stanley Tan, and Elizabeth Vander Vorst

Choosing a Topic

NYC OPEN DATA

When choosing a topic for this project, we first decided to do something related to New York City, since there is an abundance of open data relating to all aspects of the city.

After looking through the available datasets, we decided to focus on those that would be most interesting to someone who had never been there and didn't know much about the city.



What would a first-time visitor be interested in learning?

- Where to stay
- Where to eat
- Closest subway stations
- Where locals spend their time
- Safest neighborhoods
- Points of interest
- What will the weather be like

Selecting Data

HOW DO WE USE DATA TO ANSWER THOSE QUESTIONS?

Before traveling, we could gather information from multiple sources:

Where to stay - Airbnb
Where to eat - Yelp
Points of interest - multiple websites
Weather - weather.com

But how do you gather information on where locals spend their time or which neighborhoods are safest?

Crime data - create a heat map of crime data to determine which neighborhoods are safest and which to avoid.

Demographic data - create a choropleth map from census data showing which neighborhoods are most populous and which ones have the youngest population. Those could give you an idea of where to look for where the locals spend their time.

To make all this information the most useful to a first-time traveler, we need to make it easy to see how everything relates. When the different types of data are combined into one map, it will be easiest to see trends and make decisions about where to go and what to do.

Our Process - data sources

DEMOGRAPHICS

https://www1.nyc.gov/site/planning/planning-level/nyc-population/american-community-survey.page
https://www1.nyc.gov/site/planning/data-maps/open-data/dwn-nynta.page

WHERE TO STAY

https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data

WHERE TO EAT

https://www.yelp.com/developers/documentation/v3/get_started

CRIME

https://dev.socrata.com/foundry/data.cityofnewyork.us/5uac-w243

WHAT TO SEE

https://data.cityofnewyork.us/Business/Filming-Locations-Scenes-from-the-City-/qb3k-n8mm

WEATHER

OPEN WEATHER API https://openweathermap.org/

Creating a choropleth - Preparing the data

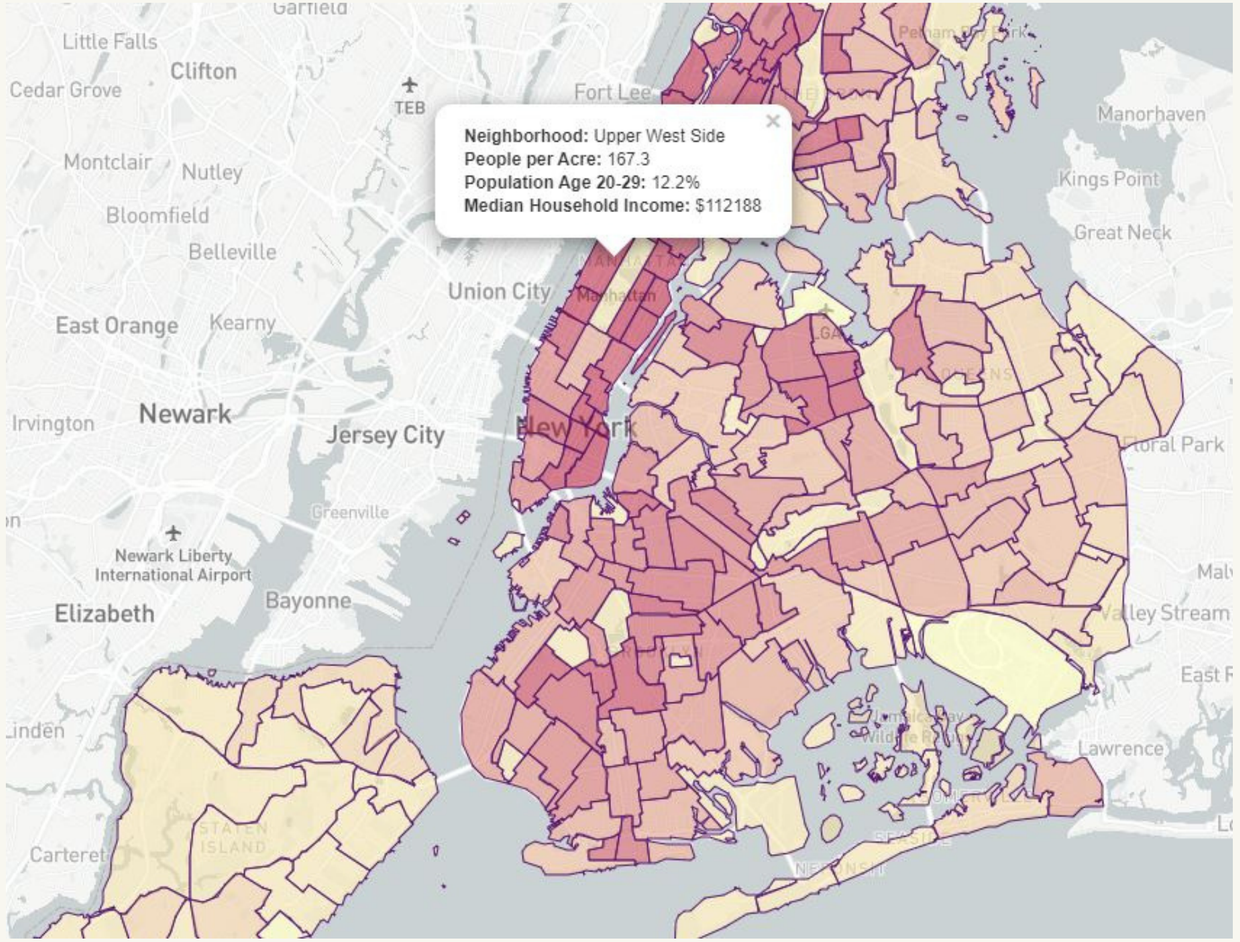
In order to incorporate demographic data into a choropleth map, the information from a CSV needed to be incorporated into an existing GeoJSON file of NYC neighborhood boundaries.

```
In [ ]: gdf = geopandas.read_file('NTAmap.geojson') # geojson file
        pdf = pandas.read_csv('NYCDemo.csv') # CSV file

        joined_gdf = gdf.merge(pdf, on="ntacode")
        joined_gdf.to_file('Demographics.geojson', driver="GeoJSON")
```

Creating a choropleth - the code

```
// Load in geojson data
var geoData = "static/data/Demographics.geojson";

// Grab data with d3
d3.json(geoData, function(data) {
  // Create a new choropleth layer
  geojson = L.choropleth(data, {
    // Define what  property in the features to use
    valueProperty: "PPA2010",
    // Set color scale
    scale: ["#ffffb2", "#b10026"],
    // Number of breaks in step range
    steps: 10,
    // q for quartile, e for equidistant, k for k-means
    mode: "q",
    style: {
      // Border color
      color: "#561771",
      weight: 1,
      fillOpacity: 0.5
    },

    //Binding a pop-up to each layer
     onEachFeature: function(feature, layer) {
       layer.bindPopup("<b>Neighborhood:</b> " + feature.properties.ntaname + "<br><b>People per Acre:</b> " +
       feature.properties.PPA2010 + "<br><b>Population Age 20-29:</b> " + feature.properties.Pop20t29P + "%" +
       "<br><b>Median Household Income:</b> " + "$" + feature.properties.MdHHIncE);
     }
  }).addTo(myMap);
```

Creating a choropleth - the map



Creating a heatmap - Preparing the data

The crime data had millions of rows, so it needed to be reduced to something that would create a usable heatmap that could load in a reasonable amount of time. This was done by using sodapy (Socrata Open Data API).

In [ ]:
```
import pandas as pd

from sodapy import Socrata
```

In [ ]:
```
client = Socrata("data.cityofnewyork.us", "w5ANkINfDguTjEbedyzkLA1HW")

results = client.get("5uac-w243", limit=3000)

crime_df = pd.DataFrame.from_records(results)
crime_df.head()
```

In [ ]:
```
crime_df_filtered = crime_df[["rpt_dt", "addr_pct_cd", "ofns_desc", "loc_of_occur_desc", "prem_typ_desc", "latitude", "longitude"
```

In [ ]:
```
crime_df_filtered
```

In [ ]:
```
crime_df_filtered = crime_df_filtered.dropna(subset=["lat_lon"])
crime_df_filtered
```

In [ ]:
```
float(crime_df_filtered["lat_lon"][1]["longitude"])
```

In [ ]:
```
float(crime_df_filtered["lat_lon"][1]["latitude"])
```

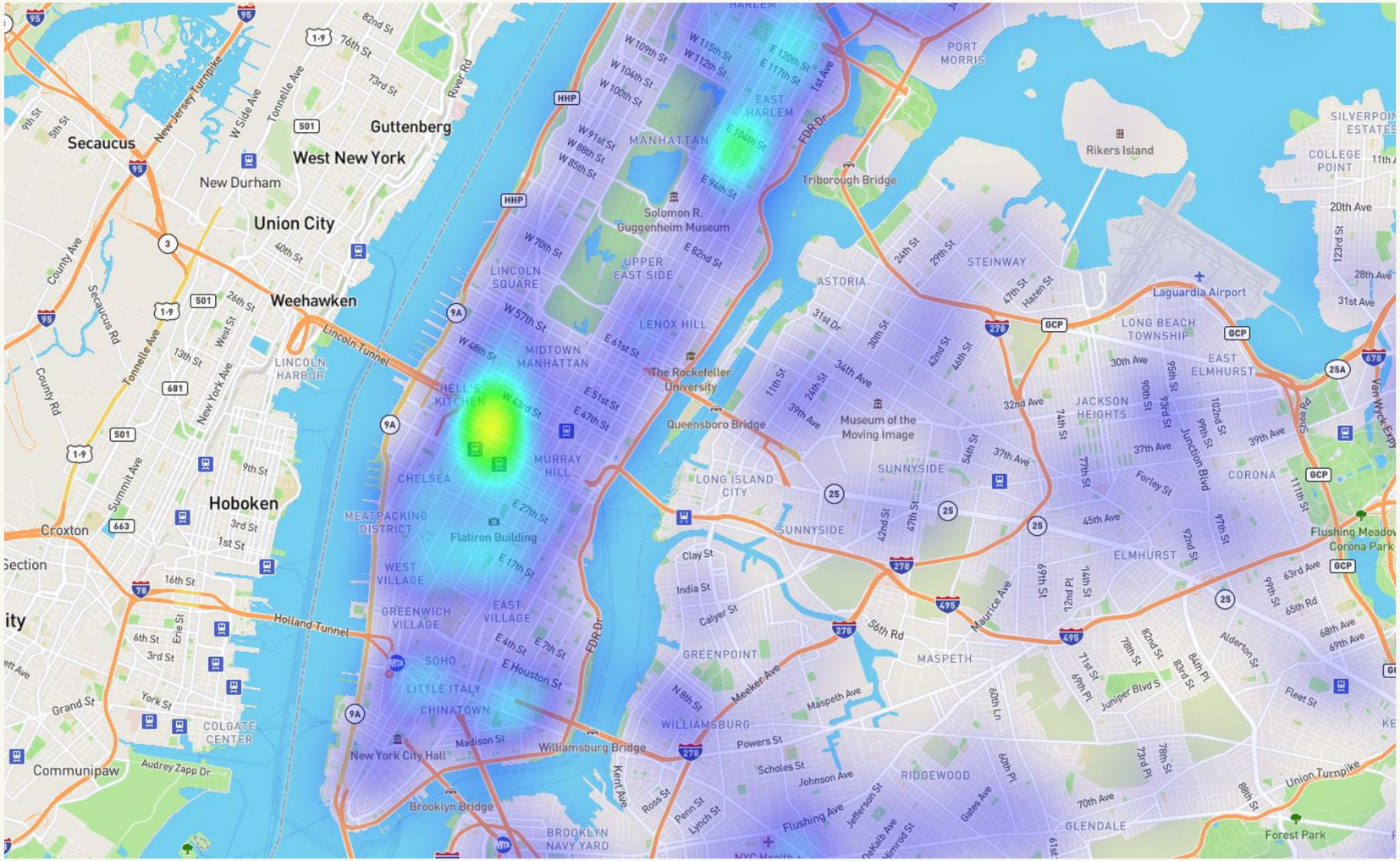
In [ ]:
```
crime_df_filtered["coords_list"] = crime_df_filtered[["latitude", "longitude"]].values.tolist()

crime_df_filtered
```

In [ ]:
```
crime_df_filtered.to_csv("NYC_crime.csv")
```

Creating a heatmap - the code

```
d3.csv("static/data/NYC_crime.csv", function(data){
  var heatArray = [];
  data.forEach(function(d){

    //coordinates = d.Lat_Lon;
    latitude = d.latitude;
    longitude = d.longitude;

    if(latitude && longitude){
      heatArray.push([latitude, longitude]);
    }
  });

  var heatMap = L.heatLayer(heatArray, {
    radius: 20,
    blur: 35
  }).addTo(myMap);

});
```

Creating a heatmap - the map



Creating a marker map - Preparing the data

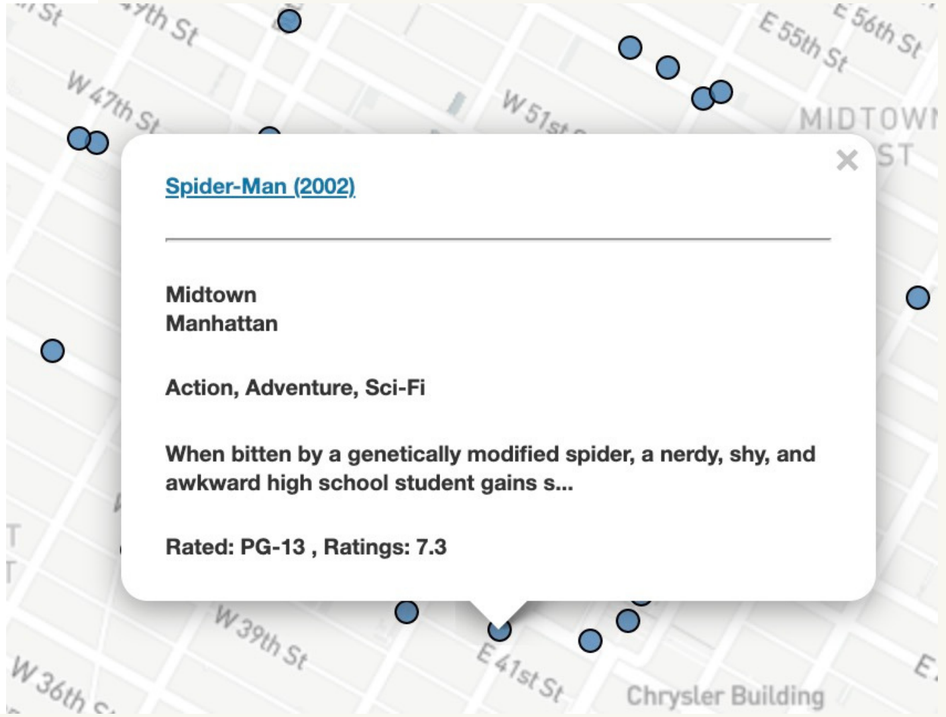
```
from time import sleep

api_key = api_key
headers = {'Authorization':'Bearer %s' %api_key}
url = 'https://api.yelp.com/v3/businesses/search'
for zip_code in manhattan_zip_codes:
    params = {'term':'food', 'location':zip_code}
    response=requests.get(url, params=params, headers=headers)
    business_data = response.json()
    print('The Status Code is {}'.format(response.status_code))
    print(zip_code)
    cols = list(business_data['businesses'][0].keys())
    print(cols)
    df = pd.DataFrame(columns=cols)
    print(df)
    for biz in business_data['businesses']:
        df = df.append(biz, ignore_index=True)
    df2 = df['coordinates'].apply(pd.Series)
    df = pd.concat([df, df2], axis=1)
```

For Yelp Data, the API was accessed and returned data in JSON format. Using Pandas, the data was transformed to fit into CSV. From Stackoverflow, got code to convert csv to geojson. Other alternative was to use pandas to convert a dataframe to a geojson (choropleth demographic data).

NYC OpenData + OMDB Movie API

```
for index, row in df.iterrows():
    try:
        movie = df['Film'][index]
        response = requests.get(f"http://www.omdbapi.com/?apikey=72
        data = response.json()
        try:
            df['Box Office'][index] = data['BoxOffice']
        except:
            print(f"none Box Office {index}")
        try:
            df['Director'][index] = data['Director']
        except:
            print(f"none Director {index}")
        try:
            df['Genre'][index] = data["Genre"]
        except:
            print(f"none Genre {index}")
        try:
            df['Plot'][index] = data["Plot"]
        except:
            print(f"none Plot {index}")
        try:
            df['Rated'][index] = data['Rated']
        except:
            print(f"none Rated {index}")
        try:
            df['Ratings'][index] = data['Ratings']
        except:
            print(f"none Runtime {index}")
```

NYC OpenData



OMDb API

The Open Movie Database

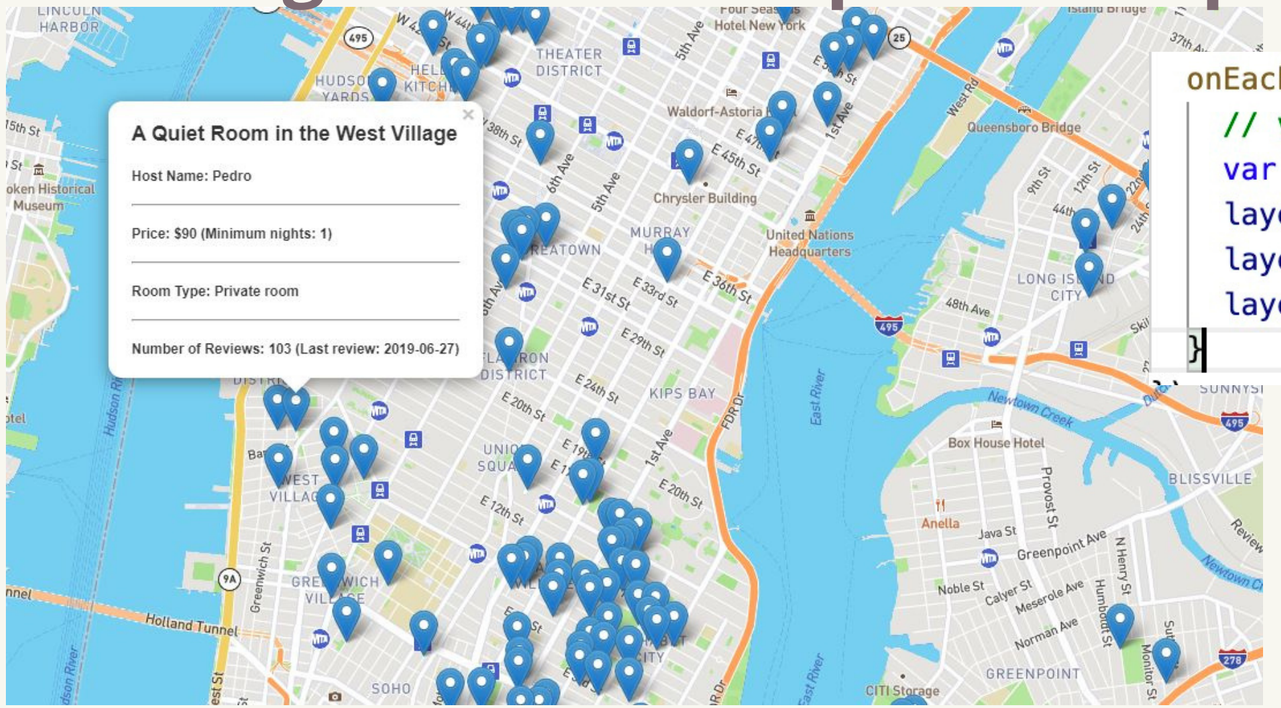
Downloaded XML file from NYC, then called OMDB movie API to combine other information so the popups on the map would provide information. And, like Yelp Data made a geojson for plotting.

Creating a marker map - the code

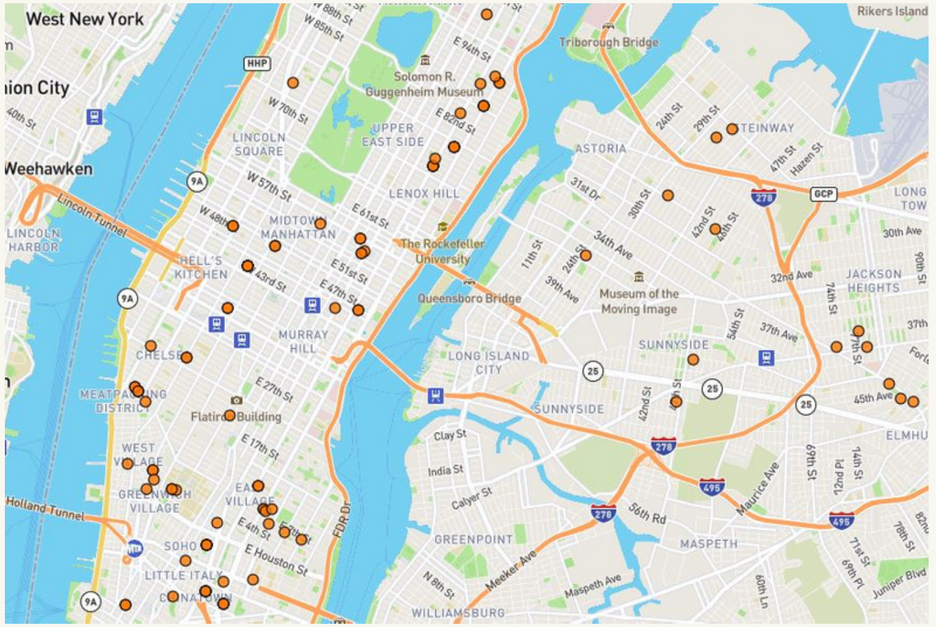
```
d3.csv("static/data/filtered_air_bnb.csv", function(data){
  data.forEach(function(d){
    coordinates = [d.latitude, d.longitude];
    L.marker(coordinates).bindPopup("<h2>" + d.name + "</h2> <h4>Host Name: "
      + d.host_name + "</h4>  <hr> <h4>Price: $"
      + d.price + " (Minimum nights: "
      + d.minimum_nights + ")</h4>  <hr> <h4>Room Type: "
      + d.room_type + "</h4>  <hr> <h4>Number of Reviews: "
      + d.number_of_reviews +" (Last review: "
      + d.last_review + ")</h4>").addTo(myMap);
  });
});
```

Creating the airbnb markers

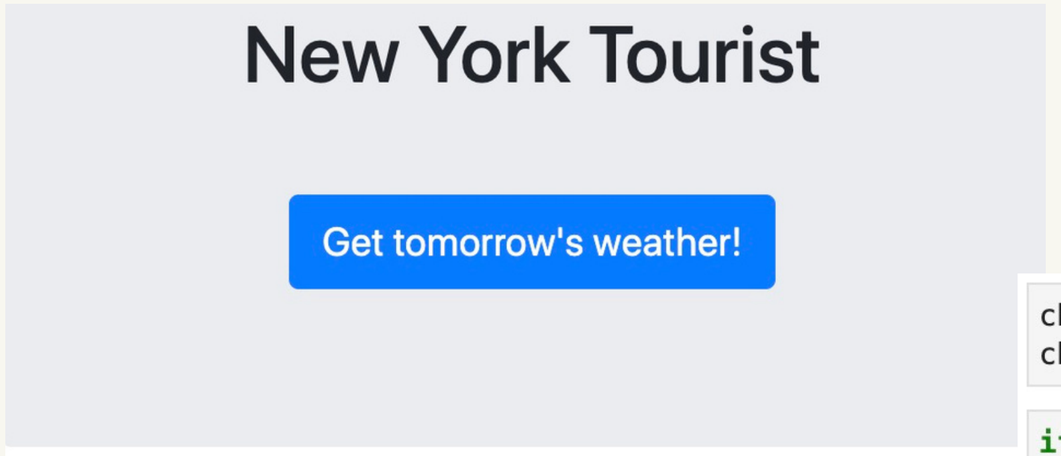
Creating a marker map - the map




```
onEachFeature: function (feature, layer) {
  // var popupText = "<h3>"+"<a href=\""+ feature.properties
  var popupText = "<h3>"+ feature.properties.name + "</h3> <
  layer.bindPopup(popupText);
  layer.on('mouseover', function() {layer.openPopup();});
  layer.on('mouseout', function() {layer.closePopup();});
}
```



Preparing for Flask to render Weather Data



pandas

OpenWeather

```
check_for_snow_list = [s for s in list_of_descriptions if "snow" in s]
check_for_rain_list = [s for s in list_of_descriptions if "rain" in s]
```

```
if len(check_for_snow_list) > 0:
    print('snow possibility')
    snow_variable = 'Yes'
else:
    print('no snow possibility')
    snow_variable = 'No'
```

```
no snow possibility
```

```
if len(check_for_rain_list) > 0:
    print('rain possibility')
    rain_variable = 'Yes'
else:
    print('no rain possibility')
    rain_variable = 'No'
```

Temperature for New York (2021-01-07): Min: 33°F , Max: 43°F

Will it rain over the next few days? No. How about snow? No.

Running Flask

Required to set up virtual environment to load packages:

```
from flask import Flask, request, jsonify, render_template, redirect
from flask_pymongo import PyMongo
import pandas as pd
import json
from datetime import datetime
import requests
```

Convert JSON to pandas dataframe in Flask. With virtual environment can import a lot of different libraries. Then, like as if you are in a Jupyter Notebook, where I prepared the code to load into app.py

```
app.route('/call')
ef search_city():
    query_url = "http://api.openweathermap.org/data/2.5/forecast?id=" + "5128581" + "&units=imperial"+"&a
    weather_response = requests.get(query_url)
    weather_json = weather_response.json()
    # return f" Ny Temp Current Temp: {weather_response['list'][0]['main']['temp']}"
    df = pd.json_normalize(weather_json, 'list')
```

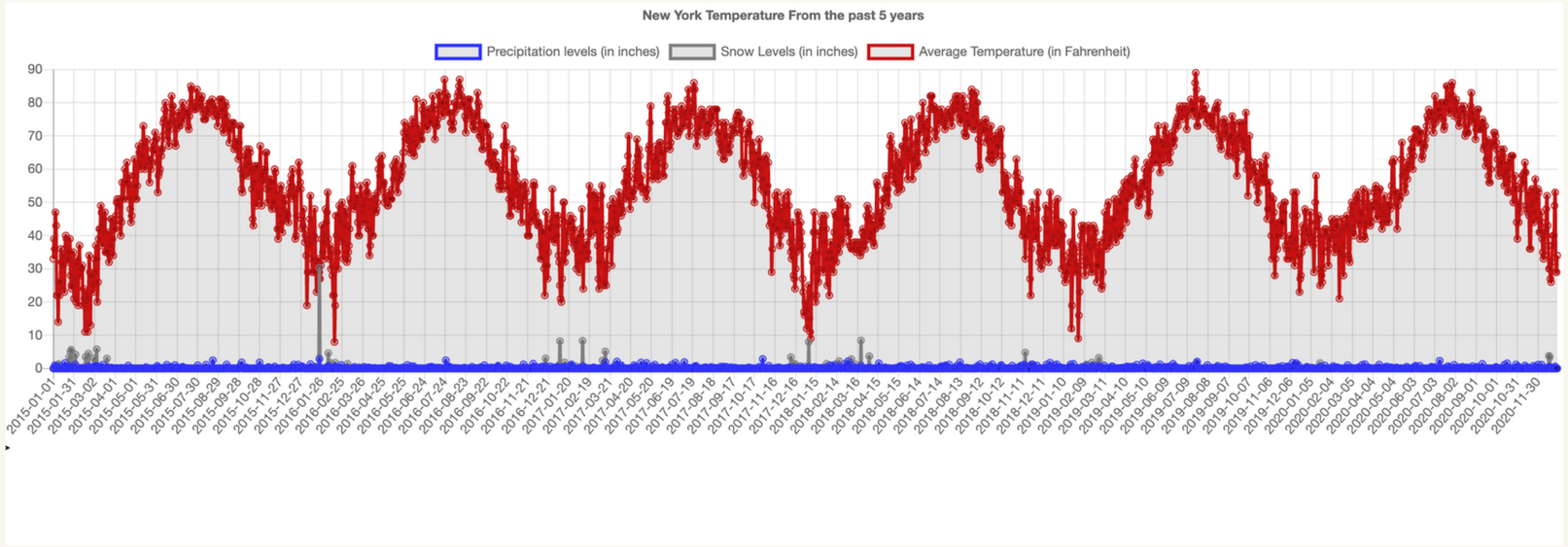
MongoDB : dictionary to server

The goal is to display the date, min and max temp for tomorrow, and whether or not there will be rain or snow in the forecast for the API call.

```
new_york_max_temp = round(filtered_df['main.temp'].max())

new_york_min_temp = round(filtered_df['main.temp'].min())

tomorrows_date = filtered_df['date'].tolist()[0].strftime('%Y-%m-%d')

new_york_weather_data = {
  "snow_variable": snow_variable,
  "rain_variable": rain_variable,
  "date": tomorrows_date,
  "min_temp": new_york_min_temp,
  "max_temp": new_york_max_temp
}

mongo.db.collection.update({}, new_york_weather_data, upsert=True)

return redirect("/")
```

Adding Weather Data (Using Chart.js)

```
function makeChart(weather) {
  console.log(weather)
  test_data = weather

  var date = weather.map(function(d){return d.DATE;});
  var precipitation = weather.map(function(d){return d.PRCP;});
  var snow = weather.map(function(d){return d.SNOW;});
  var avgTemp = weather.map(function(d){return d.TAVG;});


var chart = new Chart('lineChart', {
  type: 'line',
  data: {
    labels: date,
    datasets:[{
      borderColor: "#3232FF",
      data: precipitation,
      label: "Precipitation levels (in inches)"
    },
    {
      borderColor: "#808080",
      data: snow,
      label: "Snow Levels (in inches)"
    },
    {
      borderColor: "#C81414",
      data: avgTemp,
      label: "Average Temperature (in Fahrenheit)"
    }
    ]
  },
  options: {
    responsive: true,
    maintainAspectRatio:true,
    title: {
      display: true,
      text: 'New York Temperature From the past 5 years'
    }
  }
});


}
```



Data for weather was downloaded from for the last 5 years.

NOAA NATIONAL CENTERS FOR ENVIRONMENTAL INFORMATION
NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION

Putting it all together

Queuing all jsons used for the map

```
d3.queue()
  .defer(d3.json, link1)
  .defer(d3.json, link2)
  .defer(d3.json, link3)
  .defer(d3.json, link4)
  .defer(d3.json, link5)
  .await(makeMyMap);
```

Function used to create all the layers

```
function makeMyMap(error, movies, nyc_yelp, demographics, nyc_crime, airbnb)  {

  console.log(airbnb);
  test_json = nyc_crime;

  var heatArray = [];

  for (var i = 0; i < nyc_crime['features'].length; i++) {
    var location1 = nyc_crime['features'][i]['geometry']['coordinates'];

    if (location1) {
      heatArray.push([location1[1], location1[0]]);
    }
  }
  console.log(heatArray);

  var heat = L.heatLayer(heatArray, {
    radius: 44,
    blur: 35
  });


  var airbnbGeoJsonMap = L.geoJson(airbnb, {
    pointToLayer: function (feature, latlng) {
    return L.circleMarker(latlng, geojsonMarkerOptions3);
    },
    onEachFeature: function (feature, layer) {
      var popupText = "<h5>" + feature.properties.name + "/<h5> <hr> <h5>" + featu
      layer.bindPopup(popupText);
    }

  });
```

Putting it all together

Event handling for legend

```
myMap.on('overlayadd', function (eventLayer) {
  // Switch to the Population legend...
  if (eventLayer.name === 'demographics') {
      this.removeControl(legend);
      legend.addTo(this);
  } else { // Or switch to the Population Change legend...
      this.removeControl(legend);
  }
});
```

https://ehvandervorst.github.io/project-2/

Adding all layers created into the final map

```
var baseMaps = {
  StreetMap: streetmap,
  Light: light
};

var overlayMaps = {
  movies: moviesGeoJsonMap,
  movies_popup: moviesGeoJsonMap2,
  yelp_popup: yelpGeoJsonMap,
  demographics: demographicsGeoJsonMap,
  crime: heat,
  airbnb: airbnbGeoJsonMap
};

// Creating map object
var myMap = L.map("map", {
  center: [40.757507, -73.987772],
  zoom: 12,
  layers: [streetmap]
});

L.control.layers(baseMaps, overlayMaps).addTo(myMap);
```

Questions?

Appendix

Convert CSV TO GEOJSON: https://stackoverflow.com/questions/48586647/python-script-to-convert-csv-to-geojson

```
import pandas as pd
import geojson

def data2geojson(df):
    features = []
    insert_features = lambda X: features.append(
            geojson.Feature(geometry=geojson.Point((X["LONGITUDE"],
                                                    X["LATITUDE"])),
                            properties=dict(film=X["Film"],
                                            year=X["Year"],
                                            rating=X["ratings"],
                                            rated=X['Rated'],
                                            genre=X['Genre'],
                                            location=X["Location Display Text"],
                                            plot=X["Plot"],
                                            film_year=X["Film_Year"],
                                            url=X["Director/Filmmaker IMDB Link"]
                                            )))
    df.apply(insert_features, axis=1)
    with open('movies.geojson', 'w', encoding='utf8') as fp:
        geojson.dump(geojson.FeatureCollection(features), fp, sort_keys=True, ensure_ascii=False)
```

```
data2geojson(second_pdf)
```

Appendix - Flask app.py - timestamps

```
df = pd.json_normalize(weather_json, 'list')
df['date'] = pd.to_datetime(df['dt'],unit='s')
df['day'] = 0

for index, row in df.iterrows():
  df['day'][index] = df['date'][index].strftime('%d')

df['description'] = "0"

for index, row in df.iterrows():
  df['description'] = df['weather'][index][0]['description']

list_of_descriptions = df['description'].tolist()
```

Appendix - Flask app endpoints

```
# create route that renders index.html template
@app.route("/")
def home():
    new_york_info = mongo.db.collection.find_one()
    return render_template("index.html", news_info=new_york_info)


@app.route('/call')
def search_city():
... modifying the dataframe
mongo.db.collection.update({}, new_york_weather_data, upsert=True)


 <h5>Temperature for New York ({{ news_info.date}}): Min: {{ news_info.min_temp}}
&amp;#8457; , Max: {{ news_info.max_temp}}&amp;#8457;</h5>
      <p>Will it rain over the next few days? {{ news_info.rain_variable }}. How about
snow? {{ news_info.snow_variable }}.
        <br>See below for more info on historical weather data, or <a href="https://
firedynasty.github.io/second_project/tenth_step_plus_airbnb/" target="_blank">click
here</a> to view things to do in NY.
      </p>
    return redirect("/")
```

Appendix - Flask virtual environment

Create an environment ¶

Create a project folder and a `venv` folder within:

```
$ mkdir myproject
$ cd myproject
$ python3 -m venv venv
```

On Windows:

```
$ py -3 -m venv venv
```

If you needed to install virtualenv because you are using Python 2, use the following c

```
$ python2 -m virtualenv venv
```

On Windows:

```
> \Python27\Scripts\virtualenv.exe venv
```

Activate the environment

Before you work on your project, activate the corresponding environment:

```
$ . venv/bin/activate
```

Appendix - logic.js - Legend Setup

```
// Set up the legend
  var legend = L.control({ position: "bottomright" });
  legend.onAdd = function() {
  var div = L.DomUtil.create("div", "info legend");
  var limits = demographicsGeoJsonMap.options.limits;
  var colors = demographicsGeoJsonMap.options.colors;
  var labels = [];

  // Add min & max
  var legendInfo = "<h1>Population Density - Per Acre</h1>" +
    "<div class=\"labels\">" +
      "<div class=\"min\">" + limits[0] + "</div>" +
      "<div class=\"max\">" + limits[limits.length - 1] + "</div>" +
    "</div>";

  div.innerHTML = legendInfo;

  limits.forEach(function(limit, index) {
    labels.push("<li style=\"background-color: " + colors[index] + "\"></li>");
  });

  div.innerHTML += "<ul>" + labels.join("") + "</ul>";
  return div;
  };
```