



Project 3

NYC EMS Response Times

Group 1: Holly Cornett,
Alejandra Gomez, Dominic
Marin, Caleb Thornsbury

Project Proposal

- The aim of our project is to uncover patterns in EMS response times in NYC boroughs.
- We'll examine relationships between response time v. income
- Response time before and during the COVID-19 pandemic (Feb 22, 2020 – April 22, 2020)
- Response time v. initial severity level reported.

Data

- EMS Response Times in NYC dataset
 - Source: kaggle.com
 - Data is generated by the EMS Computer Aided Dispatch System
 - Data spans from the time the incident is created in the system to the time the incident is closed in the system
 - Dataset hosted by the City of New York. The city has an open data platform and they update their information according to the amount of data that is brought in.



Data

- Average Incomes in NYC
 - Source: data.cccnewyork.org
 - Data is pulled from U.S. Census Bureau
 - Median household income
 - Dollar amounts for all years adjusted to constant 2021 dollars using the Consumer Price Index Research Series

Median Incomes

Household Type; Dollars; 2021

MAP

TABLE

BAR CHART

SORT  

 

 

Rank / Location

All Households

New York City

\$67,997

BOROUGHES

Bronx

\$43,011

Brooklyn

\$67,567

Manhattan

\$84,435

Queens

\$73,262

Staten Island

\$86,054

Data Analysis

- Imported data into python
- Cleaned data with Pandas
- Created data frames
- Exported csv files

```
ems_data.ipynb x
C: > Users > agomez1 > Downloads > ems_data.ipynb > import pandas as pd
+ Code + Markdown | ▶ Run All | Clear All Outputs | Outline ...

import pandas as pd
[158]

file_one = "ems-incident-dispatch-data.csv"
[159]

file_one_df = pd.read_csv(file_one)
[160]

... c:\Users\Owner\anaconda3\anaconda\envs\PythonData1\lib\site-packages\IPython\core\interactiveshell.py:3457: DtypeWarning: Columns (7,11,17,21,27,28,29,30) have mixed type:
exec(code_obj, self.user_global_ns, self.user_ns)

LOADING IN DATA TO CLEAN

file_one_df
[161]

...
  CAD_INCIDENT_ID  INCIDENT_DATETIME  INITIAL_CALL_TYPE  INITIAL_SEVERITY_LEVEL_CODE  FINAL_CALL_TYPE  FINAL_SEVERITY_LEVEL_CODE  FIRST_ASSIGNMENT_DATETIME
0      202134231      2020-07-31T23:59:51.000      UNC      2      UNC      2      2020-08-01T00:00:19.000
1      202134230      2020-07-31T23:59:50.000      UNC      2      UNC      2      2020-08-01T00:00:15.000
2      202134229      2020-07-31T23:59:27.000      MVAIJ      4      MVAIJ      4      2020-07-31T23:59:38.000
3      202134228      2020-07-31T23:58:54.000      SICK      6      SICK      6      2020-08-01T00:00:30.000
4      202134226      2020-07-31T23:58:43.000      EDP      7      EDP      7      2020-07-31T23:59:17.000
...      ...      ...      ...      ...      ...      ...
20755328      50010005      2005-01-01T00:00:58.000      INJMIN      7      INJMIN      7      2005-01-01T00:02:34.000
```

Database

- Used QuickDBD to create the tables and codes
- Inserted ID column to create an index for every table
- Inserted into PG Admin 4
- Imported csv files into each table
- Selected tables to verify information was correct

The screenshot displays the pgAdmin 4 interface. On the left, the 'Browser' pane shows the 'public' schema with various objects. The 'Tables (13)' folder is expanded, showing tables like 'avg_response_per_borough', 'bronx', 'brooklyn', 'initial_severity_br', 'initial_severity_bx', 'initial_severity_m', 'initial_severity_nyc', 'initial_severity_q', 'initial_severity_si', 'manhattan', 'nyc', 'queens', 'staten_island', and 'trigger_functions'. The 'staten_island' table is selected. The main pane shows the 'Query' editor with the following SQL code:

```
130 BOROUGH VARCHAR(15) NOT NULL,
131 CONSTRAINT pk_Queens PRIMARY KEY (
132 ID
133 );
134
135
136 CREATE TABLE Staten_Island (
137 ID INT NOT NULL,
138 INCIDENT_DATETIME TIMESTAMP NOT NULL,
139 INITIAL_SEVERITY_LEVEL_CODE INT NOT NULL,
140 FINAL_SEVERITY_LEVEL_CODE INT NOT NULL,
141 FIRST_ASSIGNMENT_DATETIME TIMESTAMP NOT NULL,
142 INCIDENT_RESPONSE_SECONDS_QY INT NOT NULL,
143 BOROUGH VARCHAR(25) NOT NULL,
144 CONSTRAINT pk_Staten_Island PRIMARY KEY (
145 ID
146 );
147
148
149 SELECT * FROM nyc;
```

The 'Data Output' pane at the bottom shows the results of the query, displaying a table with 13 columns and 13 rows of data. The columns are: id, incident_datetime, initial_severity_level_code, final_severity_level_code, first_assignment_datetime, incident_response_seconds_qy, and borough. The data is as follows:

	id	incident_datetime	initial_severity_level_code	final_severity_level_code	first_assignment_datetime	incident_response_seconds_qy	borough
1	357061	2020-04-21 23:59:00		6	2020-04-22 00:00:10	352	MANHATTAN
2	357063	2020-04-21 23:58:00		4	2020-04-21 23:58:44	396	BROOKLYN
3	357065	2020-04-21 23:58:00		6	2020-04-21 23:59:19	195	BROOKLYN
4	357066	2020-04-21 23:57:00		7	2020-04-21 23:58:14	576	BRONX
5	357067	2020-04-21 23:57:00		6	2020-04-21 23:58:09	256	BROOKLYN
6	357068	2020-04-21 23:55:00		5	2020-04-21 23:55:48	504	MANHATTAN
7	357069	2020-04-21 23:55:00		4	2020-04-21 23:55:29	515	BROOKLYN
8	357071	2020-04-21 23:54:00		4	2020-04-21 23:54:47	536	BRONX
9	357072	2020-04-21 23:53:00		4	2020-04-21 23:54:06	311	BRONX
10	357076	2020-04-21 23:51:00		4	2020-04-21 23:51:33	435	QUEENS
11	357077	2020-04-21 23:51:00		6	2020-04-21 23:51:29	663	BRONX
12	357079	2020-04-21 23:50:00		4	2020-04-21 23:51:12	230	MANHATTAN
13	357080	2020-04-21 23:50:00		6	2020-04-21 23:50:55	194	MANHATTAN

Flask API

- Created an app.py file to hold our Flask API
- Used SQL Alchemy to create the flask to our Postgres SQL server
- Created different app routes to pull information from selected database tables
- Returned the information as json to use d3 in javascript

```
23 # Create our session from Python to the DB.
24
25
26 app = Flask(__name__)
27
28 CORS(app)
29 @app.route("/Manhattan1")
30 def manhattan1():
31     query = "select * from avg_response_after_covid"
32     df = pd.read_sql_query(query, engine)
33     json_df = df.to_json()
34
35     return json_df
36
37 @app.route("/Manhattan2")
38 def manhattan2():
39     query2 = "select * from avg_response_before_covid"
40     df2 = pd.read_sql_query(query2, engine)
41     json_df2 = df2.to_json()
42
43     return json_df2
44
45 @app.route("/Manhattan3")
46 def manhattan3():
47     query3 = "select * from initial_severity_m"
48     df3 = pd.read_sql_query(query3, engine)
49     json_df3 = df3.to_json()
50
51     return json_df3
52 @app.route("/Staten Island1")
53 def staten_island1():
54     query = "select * from avg_response_after_covid"
55     df = pd.read_sql_query(query, engine)
56     json_df = df.to_json()
57
58     return json_df
59
```

Javascript

- Used the geojson data for NYC from class activity along with json from Flask API to create different plotly charts

```
1 // Creating the map object
2
3
4 let myMap = L.map("map", {
5   center: [40.7128, -74.0059],
6   zoom: 11
7 });
8
9 // Adding the tile layer
10 L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
11   attribution: '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
12 }).addTo(myMap);
13
14 // Use this link to get the GeoJSON data.
15 let link = "https://2u-data-curriculum-team.s3.amazonaws.com/dataviz-classroom/v1.1/15-Mapping-Web/nyc.geojson";
16
17 // The function that will determine the color of a neighborhood based on the borough that it belongs to
18 function chooseColor(borough) {
19   if (borough == "Brooklyn") return "yellow";
20   else if (borough == "Bronx") return "red";
21   else if (borough == "Manhattan") return "green";
22   else if (borough == "Queens") return "orange";
23   else if (borough == "Staten Island") return "blue";
24   else return "black";
25 }
26 function getUrlForBorough1(borough) {
27   return `http://127.0.0.1:5000/${borough}1`;
28 }
29 function getUrlForBorough2(borough) {
30   return `http://127.0.0.1:5000/${borough}2`;
31 }
32 function getUrlForBorough3(borough) {
33   return `http://127.0.0.1:5000/${borough}3`;
34 }
35
36 //let selector = d3.select('#selDataset').value
37 //console.log(selector);
38 function BuildCharts(value){
39   //let selector = document.getElementById('selDataset').value
40   let selector = value
41   // Getting our GeoJSON data
42   d3.json(link).then(function(data) {
43     // Creating a GeoJSON layer with the retrieved data
44     L.geoJson(data, {
```


HTML & CSS

```
1 <!DOCTYPE html>
2 <html lang="en-us">
3   <head>
4     <meta charset="UTF-8">
5     <title>NYC Boroughs</title>
6
7     <!-- Leaflet CSS -->
8     <link rel="stylesheet" href="https://unpkg.com/leaflet@1.3.3/dist/leaflet.css"
9       integrity="sha512-Rkscm5RenBEKSKFjgI3a41vrjkw4EVPlJ3+OiI65vTjIdo9br1AacEuK0iQ50Fh7c0I1bkDwLqdlw3Zg0cRJAQ=="
10     crossorigin="" />
11
12     <!-- Leaflet JavaScript code -->
13     <script src="https://unpkg.com/leaflet@1.3.3/dist/leaflet.js"
14       integrity="sha512-tAG6CfR4Sc5ZP5Z0Vz0quoZDYX5aCtEm/eu1KhSLj2c9eFrylXZknQYmxUssFaVJKvvc0dJQixhGjG2yXwiV9Q=="
15     crossorigin=""></script>
16
17     <!-- D3 library -->
18     <script src="https://d3js.org/d3.v5.min.js"></script>
19     <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
20
21     <!-- Our CSS -->
22     <link rel="stylesheet" type="text/css" href="./style.css">
23
24   </head>
25   <body>
26     <h1 align="center">Ambulance Response Time for NYC and its 5 Boroughs</h1>
27     <select onchange="BuildCharts(this.value)" id="selDataset">
28       <option value="Before_Covid">Before Covid</option>
29       <option value="During_Covid">During Covid</option>
30     </select>
31
32     <div id="map"></div>
33     <h2 align="center">Average Response Time per Borough</h2>
34     <div id="plot"></div>
35     <h3>Initial Severity Level of Calls in Borough</h3>
36     <div id="pie"></div>
37
38
39     <script type="text/javascript" src="./plots.js"></script>
40
41   </body>
42 </html>
```

```
1 body {
2   padding: 0;
3   margin: 0;
4 }
5
6 #map,
7 body,
8 html {
9   height: 73%;
10 }
11
12 #plot,
13 body,
14 html {
15   height: 65%;
16 }
17
18 #pie,
19 body,
20 html {
21   height: -10%;
22 }
```

Web Page

Thank you.

