sumana_python_work

November 4, 2023

1 Sumana Chilakamarri: Python Work Sample

This project analyzes how a neighborhood's economic wealth affects access to emergency medical services. Specifically, we used New York City and its characteristic boroughs, Manhattan, Queens, Brooklyn, the Bronx, and Staten Island to hypothesize that lower income boroughs face more barriers to obtain emergency medical response resources than high-income boroughs do on the basis of EMS efficiency.

We chose this region because New York City is a large-scale model for other urbanized areas in the United States that have extreme wealth inequalities (one of the aspects that we analyzed through web scraping), with densely concentrated populations of groups on either end of the spectrum being common indicators of the areas that are wealthy and those that are not.

We also chose to focus on data from the year of 2021, because COVID likely had an effect on all aspects of emergency medical services as well as the demographic data for each borough.

1.1 Data Collection & Cleaning Process

Our data collection process was very long and arduous because most sources from different states privatize EMS data, or have too many holes in the data for the data to be usable. Luckily, we found a huge file documenting every case in NYC for several years, which led us down a rabbithole of finding government published, reliable data that still needed some work to be interpretable.

As a general outline of our process, we first cleaned three data sets, one API related to borough demographics, one json file regarding average emergency response times for each month across several years, and one very very large csv file with approximately 24 million rows of every EMS case in NYC from 2005 to halfway through 2022.

We then proceeded to analyze these data sets in a specific order: first we used the API related to borough demographic data in order to predict 2021's borough demographic data because we couldn't find a single source with that information, then we used the large csv file to analyze the relationship between dispatch response times and incident response times based on various factors, and then we used the json file to just analyze the center and spread of incident response times in more detail.

2 Data Collection and Cleaning

Transfer/update the data collection and cleaning you created for Phase II below. You may include additional cleaning functions if you have extra datasets. If no changes are necessary, simply copy and paste your phase II parsing/cleaning functions.

2.1 Downloaded Dataset (CSV)

[88]:
```
import json
import pandas as pd
import numpy as np
import requests, re
from bs4 import BeautifulSoup
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

[ ]:
```

```

[89]:
```
# Here's where we found this data:
# https://data.cityofnewyork.us/Public-Safety/EMS-Incident-Dispatch-Data/
 ↪76xm-jjuj

def ems_data():
    for i,chunk in enumerate(pd.read_csv("incident_dispatch_data.csv",␣
 ↪chunksize=1000000, dtype=object)):
        chunk.to_csv("C:/Users/diyam/Desktop/CS 2316/Final Project/chunk{}.csv".
 ↪format(i), index=True)
    return chunk

# Because our file was originally 5.22GB with 23.5M rows, we began by splitting␣
 ↪up the file into 23 smaller files, with
# ~1M rows in each "chunk".

# We referenced this website for the above code:
# https://mungingdata.com/python/split-csv-write-chunk-pandas/

def chunked_ems_data():
    df = pd.read_csv("chunk21.csv")
    df1 = pd.read_csv("chunk22.csv")

# We only wanted to work with data from the year of 2021, which lied within two␣
 ↪of our smaller files, hence why we
# created two dataframes for the data in both files.

    df = df.iloc[309675:,:]
    df1 = df1.iloc[:803927,:]
    data = pd.concat([df, df1], axis=0)
    data = data.replace(0, np.nan) # FIXING INCONSISTENCY #1
    data = data.replace(1, np.nan) # FIXING INCONSISTENCY #1

# There was some 2020 data and 2022 data before and after the 2021 data, so we␣
 ↪extracted the 2021 data from both
# files and concatenated the two dataframes together.
```

```
# We also handled extremely small outliers by replacing cells where
# both "DISPATCH_RESPONSE_SECONDS_QY" and "INCIDENT_RESPONSE_SECONDS_QY" are␣
↪equal to 0 or 1 by replacing those cells with
# NaN so that those values don't pull down any averages we may decide to␣
↪calculate with those columns in the future.

    data = data.drop(["Unnamed: 0", "CAD_INCIDENT_ID", "INITIAL_CALL_TYPE",␣
↪"FINAL_CALL_TYPE", "FINAL_SEVERITY_LEVEL_CODE",␣
↪"FIRST_ASSIGNMENT_DATETIME"], axis = 1)
    data = data.drop(["FIRST_ACTIVATION_DATETIME", "FIRST_ON_SCENE_DATETIME",␣
↪"INCIDENT_TRAVEL_TM_SECONDS_QY", "FIRST_TO_HOSP_DATETIME",␣
↪"FIRST_HOSP_ARRIVAL_DATETIME"], axis = 1)
    data = data.drop(["INCIDENT_CLOSE_DATETIME", "HELD_INDICATOR",␣
↪"INCIDENT_DISPATCH_AREA", "ZIPCODE", "POLICEPRECINCT",␣
↪"CITYCOUNCILDISTRICT", "COMMUNITYDISTRICT", "COMMUNITYSCHOOLDISTRICT",␣
↪"CONGRESSIONALDISTRICT", "REOPEN_INDICATOR", "SPECIAL_EVENT_INDICATOR",␣
↪"STANDBY_INDICATOR", "TRANSFER_INDICATOR"], axis = 1)

# We dropped columns we don't need as well as all the holes in the dataset.

    data["VALID_DISPATCH_RSPNS_TIME_INDC"] = np.
↪where(data["VALID_DISPATCH_RSPNS_TIME_INDC"].astype(str)=="Y", True, False)
    data["VALID_INCIDENT_RSPNS_TIME_INDC"] = np.
↪where(data["VALID_INCIDENT_RSPNS_TIME_INDC"].astype(str)=="Y", True, False)
    data = data[(data["VALID_DISPATCH_RSPNS_TIME_INDC"]==True) &␣
↪(data["VALID_INCIDENT_RSPNS_TIME_INDC"]==True)]
    data = data.drop(["VALID_DISPATCH_RSPNS_TIME_INDC",␣
↪"VALID_INCIDENT_RSPNS_TIME_INDC"], axis=1)
    data = data[data["INITIAL_SEVERITY_LEVEL_CODE"] >= 8]
    data = data[(data["INCIDENT_DISPOSITION_CODE"] == 82) |␣
↪(data["INCIDENT_DISPOSITION_CODE"] == 83) |␣
↪(data["INCIDENT_DISPOSITION_CODE"] == 91) |␣
↪(data["INCIDENT_DISPOSITION_CODE"] == 92) |␣
↪(data["INCIDENT_DISPOSITION_CODE"] == 94) |␣
↪(data["INCIDENT_DISPOSITION_CODE"] == 95)␣
↪|(data["INCIDENT_DISPOSITION_CODE"] == 96)]
    data.rename(columns = {"INCIDENT_DATETIME" : "Date",␣
↪"INITIAL_SEVERITY_LEVEL_CODE" : "Severity Level",␣
↪"DISPATCH_RESPONSE_SECONDS_QY" : "Dispatch Response Time",␣
↪"INCIDENT_RESPONSE_SECONDS_QY" : "Incident Response Time",␣
↪"INCIDENT_DISPOSITION_CODE" : "Disposition Code", "BOROUGH" : "Borough"},␣
↪inplace = True)
    data = data.drop(["Severity Level"], axis=1)
```

```
# We used masking to reduce rows from ~2M Rows to ~1776 rows based on various␣
↪criteria (all the dispatch and response
# data had to be valid and categorized as "Y", the case's initial severity code␣
↪had to be at least 8 on a scale of 1-10,
# irrelevent dispositions or statuses of the patient such as "cancelled" or␣
↪"unfounded". We then renamed some columns to be
# more readable.

    data = data.sort_values([ "Borough", "Date"], ascending=[True, True],␣
↪inplace=False)
    data = data.reset_index(drop=True)

# We sorted data based on borough first and then by date, so the dataframe was␣
↪structured in a way where every case in 2021 was
# listed for the Bronx, and below that every case in 2021 was listed for␣
↪Brooklyn, so on and so forth.

# FIXING INCONSISTENCY #2
    data = data.replace("BRONX", "Bronx")
    data = data.replace("BROOKLYN", "Brooklyn")
    data = data.replace("MANHATTAN", "Manhattan")
    data = data.replace("QUEENS", "Queens")
    data = data.replace("RICHMOND / STATEN ISLAND", "Staten Island")

# In order to improve our ability to cross-reference borough data with other␣
↪datasets' borough data, we renamed values in the
# 'Borough' column to be consistent with the other datasets.

    bronx = data[data["Borough"]=="Bronx"]
    brooklyn = data[data["Borough"] =="Brooklyn"]
    manhattan = data[data["Borough"] =="Manhattan"]
    queens = data[data["Borough"]=="Queens"]
    staten_island = data[data["Borough"] == "Staten Island"]

    data.to_csv("2021_ems_data.csv")

#     data.to_excel(summary_file, sheet_name = "Cleaned CSV")
#     summary_file.save()

    return data



############ Function Call ############
```

```
# ems_data()
chunked_ems_data()
```

[89]:

```
                        Date  Dispatch Response Time  Incident Response Time  \
0     01/01/2021 09:22:11 PM                     NaN                     NaN
1     01/02/2021 06:15:14 AM                    13.0                   881.0
2     01/06/2021 04:12:00 AM                    23.0                   439.0
3     01/06/2021 12:38:00 AM                    12.0                   652.0
4     01/07/2021 05:01:00 PM                    16.0                   238.0
…                        …                       …                       …
1794  12/06/2021 08:28:53 PM                    15.0                   405.0
1795  12/10/2021 10:19:34 AM                     NaN                     NaN
1796  12/11/2021 12:07:14 AM                 22781.0                 26133.0
1797  12/12/2021 09:16:11 AM                  2972.0                  8371.0
1798  12/30/2021 08:07:36 AM                    21.0                  1006.0

      Disposition Code        Borough
0                 91.0          Bronx
1                 82.0          Bronx
2                 91.0          Bronx
3                 91.0          Bronx
4                 91.0          Bronx
…                    …              …
1794              91.0  Staten Island
1795              91.0  Staten Island
1796              91.0  Staten Island
1797              91.0  Staten Island
1798              82.0  Staten Island

[1799 rows x 5 columns]
```

2.2 Web Collection (API)

[90]:

```
def borough_data():
    r = requests.get("https://furmancenter.org/stateofthecity/view/
 ↪citywide-and-borough-data").text
    a_list = []
    soup = BeautifulSoup(r, "html.parser")
    tables = soup.find_all("table")

# There were multiple tables per borough on the webpage, so we had to iterate␣
 ↪through each table tag.

    for table in tables:
        trs = table.find_all("tr")[10:12]
```

```
# Rows 10 and 11 were the two demographics that we wanted to extract data from␣
↪for every borough -- median household
# income and poverty rate across several years.

        for tr in trs:
            final = {}
            rows = tr.find_all("td")
            demographic = str(rows[0].text.strip()) # This step extracted the␣
↪name of the demographic being measured
                                                    # (e.g Median household␣
↪income, Poverty rate)


# We iterated through each data value in rows 10 and 11 and created a␣
↪dictionary mapping the name of the borough to a list
# containing two additional dictionaries. The first dictionary mapped the title␣
↪"median household income" to a list of median household
# incomes across the years 2000, 2006, 2010, and 2018, while the second␣
↪dictionary mapped the title "poverty rate" to a list of poverty
# rates for the same years.

            one = rows[1].text.strip()
            two = rows[2].text.strip()
            three = rows[3].text.strip()
            four = rows[4].text.strip()

# We stripped the values in each row in order to obtain the median household␣
↪income and poverty rate values for every year. Because
# this segment is within a nested loop iterating through row tags within table␣
↪tags, it did this for every borough.

            final[demographic] = [one, two, three, four]
            a_list.append(final)

# We created a list of the four data values (one, two three, four) and mapped␣
↪it to the corresponding demographic in respective distionaries,
# which we defined in a variable called final that was referenced within the␣
↪loop. We appended all the dictionaries for every
# borough to a list referenced by the variable a_list.

    data = {}
    data["New York City"] = [a_list[0], a_list[1]]
    data["The Bronx"] = [a_list[4], a_list[5]]
    data["Brooklyn"] = [a_list[6], a_list[7]]
    data["Manhattan"] = [a_list[8], a_list[9]]
    data["Queens"] = [a_list[10], a_list[11]]
```

```
    data["Staten Island"] = [a_list[12], a_list[13]]

# We created another dictionary that mapped the borough name to the value in␣
 ↪the list using values from a_list.
# At this point, the data is structured as such:
# {'Brooklyn': [{'Median household income (2019$)': ['$50,500',␣
 ↪'$50,910','$48,670','$62,230']}, {'Poverty rate': ['25.1%', '22.6%', '23.
 ↪0%', '19.0%']}],
# 'Manhattan': [{'Median household income (2019$)': ['$73,910',␣
 ↪'$75,640','$73,720','$86,470']}, {'Poverty rate': ['19.9%', '18.3%', '16.
 ↪4%', '15.5%']}], ...}

    dfs = []
    for key, value in list(data.items()):
        med = value[0]
        pov = value[1]
        median = med["Median household income (2019$)"]
        poverty = pov["Poverty rate"]
        nyc = pd.DataFrame(median, index = ["2000", "2006", "2010", "2018"],␣
 ↪columns = ["Median Household Income"])
        nyc["Poverty Rate"] = poverty
        nyc = nyc.rename(columns = {"Median Household Income": (str(key) + ":␣
 ↪Median Household Income")})
        dfs.append(nyc)

# In order to make the data more readable, we made the data into pandas␣
 ↪DataFrames, but iterating through our complex dictionary
# was challenging, so we appended the Dataframes corresponding to every borough␣
 ↪into a list referenced in the variable dfs.

    writer = pd.ExcelWriter("borough_demographics.xlsx", engine = "xlsxwriter")

    nyc_data = dfs[0]
    nyc_data.to_excel(writer, sheet_name = "NYC Demographics")

    bronx = dfs[1]
    bronx.to_excel(writer, sheet_name = "Bronx Demographics")

    brooklyn = dfs[2]
    brooklyn.to_excel(writer, sheet_name = "Brooklyn Demographics")

    manhattan = dfs[3]
    manhattan.to_excel(writer, sheet_name = "Manhattan Demographics")

    queens = dfs[4]
    queens.to_excel(writer, sheet_name = "Queens Demographics")
```

```
    statenisland = dfs[5]
    statenisland.to_excel(writer, sheet_name = "Staten Island Demographics")

#     writer.save()

#     bronx.to_excel(summary_file, sheet_name = "Cleaned API")
#     brooklyn.to_excel(summary_file, sheet_name = "Cleaned API")
#     manhattan.to_excel(summary_file, sheet_name = "Cleaned API")
#     queens.to_excel(summary_file, sheet_name = "Cleaned API")
#     statenisland.to_excel(summary_file, sheet_name = "Cleaned API")
#     summary_file.save()

# We saved each dataframe to a different sheet corresponding to the borough in␣
↪the same Excel file.
    writer.save()

    return dfs


############ Function Call ############
borough_data()
```

```
[90]: [      New York City: Median Household Income Poverty Rate
 2000                                   $60,180        21.2%
 2006                                   $58,580        19.2%
 2010                                   $56,290        20.1%
 2018                                   $64,850        17.3%,
       The Bronx: Median Household Income Poverty Rate
 2000                               $43,390        30.7%
 2006                               $39,690        29.1%
 2010                               $37,610        30.2%
 2018                               $39,100        27.4%,
       Brooklyn: Median Household Income Poverty Rate
 2000                              $50,500        25.1%
 2006                              $50,910        22.6%
 2010                              $48,670        23.0%
 2018                              $62,230        19.0%,
       Manhattan: Median Household Income Poverty Rate
 2000                               $73,910        19.9%
 2006                               $75,640        18.3%
 2010                               $73,720        16.4%
 2018                               $86,470        15.5%,
       Queens: Median Household Income Poverty Rate
 2000                            $66,690        14.6%
 2006                            $64,520        12.2%
 2010                            $61,270        15.0%
```

```
2018                          $70,470       11.5%,
     Staten Island: Median Household Income Poverty Rate
2000                                 $86,500        10.0%
2006                                 $86,490         9.2%
2010                                 $81,490        11.8%
2018                                 $83,520        11.4%]
```

2.3 Web Collection (.json)

[91]:
```python
# Here's where we found this data
# https://catalog.data.gov/dataset/911-open-data-local-law-119

def response_times():
    with open('nyc_data.json', 'r') as f:
        final_data = []
        jsondict = json.load(f)
        list_of_lists = jsondict["data"]

# We took a json file and loaded it into a python dictionary to parse through.

        for a_list in list_of_lists:
            final_data.append(a_list[8:14])

# We only wanted the 8th to 13th indeces within the list of dictionaries,␣
↪because those were the values that were relevant to us,
# and the surrounding entries in the dictionary were all just metadata. We␣
↪appended those values to a dictionary called final_data
# for further use.

        df = pd.DataFrame(final_data)
        df.columns = ["Month Name", "Agency", "Description", "Borough", "# of␣
↪Incidents", "Response Times"]

# We wrote the data to a pandas DataFrame and named the columns according to␣
↪the data.

        df = df[(df["Month Name"] == "2021 / 12") | (df["Month Name"] == "2021 /
↪ 11") | (df["Month Name"] == "2021 / 10") | (df["Month Name"] == "2021 /␣
↪09") | (df["Month Name"] == "2021 / 08") | (df["Month Name"] == "2021 / 07")␣
↪| (df["Month Name"] == "2021 / 06") | (df["Month Name"] == "2021 / 05") |␣
↪(df["Month Name"] == "2021 / 04") | (df["Month Name"] == "2021 / 03") |␣
↪(df["Month Name"] == "2021 / 02") | (df["Month Name"] == "2021 / 01")]
        df = df[(df["Agency"] == "EMS") | (df["Agency"] == "Aggregate")]
        df = df[(df["Borough"] != "Unspecified")]
```

```
        df = df[(df["Description"] == "Average response time to life␣
↪threatening medical emergencies by ambulance units") | (df["Description"] ==␣
↪"Combined average response time to life threatening medical emergencies by␣
↪ambulance and fire units")]
        df["# of Incidents"] = df["# of Incidents"].astype(float)
        df["Response Times"] = df["Response Times"].astype(float)

        df = df.rename(columns = {"Response Times": "Incident Response Time"})␣
↪# FIXING INCONSISTENCY #3

# We used masking to cut down ~7600 rows to 120 rows by limiting the data to␣
↪2021 and looking at EMS and Aggregate medical data that disregardes
# FDNY cases. We also got rid of unspecified boroughs and only included average␣
↪response time to severe, life threatening medical emergencies

        df = df.groupby(["Month Name", "Borough"]).agg({"Incident Response␣
↪Time":"mean", "# of Incidents":"mean"})

        df.to_csv("cleaned_response_times.csv", index=True)

# We took the average of the response times and number of incedents per month␣
↪for each borough. We then wrote the
# cleaned dataframe to a csv file.

#     df.to_excel(summary_file, sheet_name = "Cleaned JSON")
#     summary_file.save()

    return(df)

############ Function Call ############
response_times()
```

[91]:

```
                          Incident Response Time  # of Incidents
Month Name Borough
2021 / 01  Bronx                      565.021531          9828.5
           Brooklyn                   534.380453         11056.5
           Manhattan                  548.760145          8646.0
           Queens                     559.172599          8402.5
           Staten Island              547.206999          2086.0
2021 / 02  Bronx                      601.845926          9241.5
           Brooklyn                   563.873652         10193.5
           Manhattan                  564.127419          8244.0
           Queens                     601.199767          7824.0
           Staten Island              568.095534          1863.0
2021 / 03  Bronx                      578.259099          9911.5
           Brooklyn                   536.981045         11576.5
```

	Manhattan	543.618296	9434.5
	Queens	575.567826	8821.5
	Staten Island	566.445252	2102.0
2021 / 04	Bronx	573.192347	10084.5
	Brooklyn	537.680360	11499.0
	Manhattan	551.129183	9813.5
	Queens	568.178527	8524.0
	Staten Island	560.428036	2030.5
2021 / 05	Bronx	588.660982	10286.0
	Brooklyn	557.586797	11884.0
	Manhattan	554.083040	10228.0
	Queens	576.230024	8687.5
	Staten Island	557.663895	2025.5
2021 / 06	Bronx	607.783206	10436.5
	Brooklyn	566.152603	11920.5
	Manhattan	577.013785	11213.5
	Queens	584.934305	8864.5
	Staten Island	568.242382	1979.0
2021 / 07	Bronx	605.654994	10281.0
	Brooklyn	568.380535	12458.5
	Manhattan	562.532338	10896.0
	Queens	581.354606	9007.0
	Staten Island	556.406333	2141.5
2021 / 08	Bronx	661.038258	7350.5
	Brooklyn	612.076541	8863.0
	Manhattan	595.410949	7510.0
	Queens	618.500043	6593.5
	Staten Island	613.692984	1457.5
2021 / 09	Bronx	642.899070	8277.0
	Brooklyn	595.054144	10198.0
	Manhattan	597.011098	8677.0
	Queens	640.371018	7554.0
	Staten Island	611.358401	1760.5
2021 / 10	Bronx	597.244133	9955.0
	Brooklyn	581.288224	12316.0
	Manhattan	575.291818	10586.5
	Queens	595.775050	8587.5
	Staten Island	575.992446	2109.5
2021 / 11	Bronx	605.212516	9374.5
	Brooklyn	572.248901	11410.5
	Manhattan	575.481834	9599.0
	Queens	599.759129	8260.0
	Staten Island	583.084397	2024.5
2021 / 12	Bronx	662.784251	10636.0
	Brooklyn	629.028300	12946.0
	Manhattan	591.632088	10284.5
	Queens	626.568250	9207.5

```
Staten Island              617.715414          2246.0
```

3 Data Analysis

We analyzed these data sets in a specific order: first we used the API related to borough demographic data in order to predict 2021's borough demographic data because we couldn't find a single source with that information, then we used the large csv file to analyze the relationship between dispatch response times and incident response times based on various factors, and then we used the .json file to just analyze the center and spread of incident response times in more detail.

[92]:
```
import statsmodels.api as sm
import warnings
with warnings.catch_warnings():
    warnings.simplefilter(action='ignore', category=FutureWarning)
```

3.1 Insights

[93]:
```
def insight1():
    bronx = pd.read_excel("borough_demographics.xlsx", sheet_name = "Bronx␣
↪Demographics")
    brooklyn = pd.read_excel("borough_demographics.xlsx", sheet_name =␣
↪"Brooklyn Demographics")
    manhattan = pd.read_excel("borough_demographics.xlsx", sheet_name =␣
↪"Manhattan Demographics")
    queens = pd.read_excel("borough_demographics.xlsx", sheet_name = "Queens␣
↪Demographics")
    staten_island = pd.read_excel("borough_demographics.xlsx", sheet_name =␣
↪"Staten Island Demographics")

    final_demographic = pd.DataFrame()
    years = np.array([2000, 2008, 2010, 2018])

    bronx_med = bronx['The Bronx: Median Household Income'].str[1:]
    bronx_med = bronx_med.str.replace(r"\,","")
    bronx_pov = bronx['Poverty Rate'].str[0:]
    bronx_pov = bronx_pov.str.replace(r"\%","")
    bronx_med = pd.to_numeric(bronx_med)
    bronx_pov = pd.to_numeric(bronx_pov)

    # bronx median prediction
    x = bronx_med.to_numpy()
    y = years
    n = np.size(years)
    m_x = np.mean(years)
    m_y = np.mean(bronx_med)
    SS_xy = np.sum(y*x) - n*m_y*m_x
```

```
    SS_xx = np.sum(x*x) - n*m_x*m_x
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x
    bronx_med_2021 = b_1 * 2021 + b_0

# bronx pov predictor
    x = bronx_pov.to_numpy()
    y = years
    n = np.size(years)
    m_x = np.mean(years)
    m_y = np.mean(bronx_pov)
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x
    bronx_pov_2021 = b_1 * 2021 + b_0

    brooklyn_med = brooklyn['Brooklyn: Median Household Income'].str[1:]
    brooklyn_med = brooklyn_med.str.replace(r"\,","")
    brooklyn_pov = brooklyn['Poverty Rate'].str[0:]
    brooklyn_pov = brooklyn_pov.str.replace(r"\%","")
    brooklyn_med = pd.to_numeric(brooklyn_med)
    brooklyn_pov = pd.to_numeric(brooklyn_pov)

    # brooklyn median prediction
    x = brooklyn_med.to_numpy()
    y = years
    n = np.size(years)
    m_x = np.mean(years)
    m_y = np.mean(brooklyn_med)
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x
    brooklyn_med_2021 = b_1 * 2021 + b_0

    # brooklyn pov predictor
    x = brooklyn_pov.to_numpy()
    y = years
    n = np.size(years)
    m_x = np.mean(years)
    m_y = np.mean(brooklyn_pov)
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x
    brooklyn_pov_2021 = b_1 * 2021 + b_0
```

```
manhattan_med = manhattan['Manhattan: Median Household Income'].str[1:]
manhattan_med = manhattan_med.str.replace(r"\,","")
manhattan_pov = manhattan['Poverty Rate'].str[0:]
manhattan_pov = manhattan_pov.str.replace(r"\%","")
manhattan_med = pd.to_numeric(manhattan_med)
manhattan_pov = pd.to_numeric(manhattan_pov)


# manhattan median prediction
x = manhattan_med.to_numpy()
y = years
n = np.size(years)
m_x = np.mean(years)
m_y = np.mean(manhattan_med)
SS_xy = np.sum(y*x) - n*m_y*m_x
SS_xx = np.sum(x*x) - n*m_x*m_x
b_1 = SS_xy / SS_xx
b_0 = m_y - b_1*m_x
manhattan_med_2021 = b_1 * 2021 + b_0

# manhattan pov predictor
x = manhattan_pov.to_numpy()
y = years
n = np.size(years)
m_x = np.mean(years)
m_y = np.mean(manhattan_pov)
SS_xy = np.sum(y*x) - n*m_y*m_x
SS_xx = np.sum(x*x) - n*m_x*m_x
b_1 = SS_xy / SS_xx
b_0 = m_y - b_1*m_x
manhattan_pov_2021 = b_1 * 2021 + b_0

queens_med = queens['Queens: Median Household Income'].str[1:]
queens_med = queens_med.str.replace(r"\,","")
queens_pov = queens['Poverty Rate'].str[0:]
queens_pov = queens_pov.str.replace(r"\%","")
queens_med = pd.to_numeric(queens_med)
queens_pov = pd.to_numeric(queens_pov)


# queens median prediction
x = queens_med.to_numpy()
y = years
n = np.size(years)
m_x = np.mean(years)
m_y = np.mean(queens_med)
```

```
SS_xy = np.sum(y*x) - n*m_y*m_x
SS_xx = np.sum(x*x) - n*m_x*m_x
b_1 = SS_xy / SS_xx
b_0 = m_y - b_1*m_x
queens_med_2021 = b_1 * 2021 + b_0

# queens pov predictor
x = queens_pov.to_numpy()
y = years
n = np.size(years)
m_x = np.mean(years)
m_y = np.mean(queens_pov)
SS_xy = np.sum(y*x) - n*m_y*m_x
SS_xx = np.sum(x*x) - n*m_x*m_x
b_1 = SS_xy / SS_xx
b_0 = m_y - b_1*m_x
queens_pov_2021 = b_1 * 2021 + b_0

staten_med = staten_island['Staten Island: Median Household Income'].str[1:]
staten_med = staten_med.str.replace(r"\,","")
staten_pov = staten_island['Poverty Rate'].str[0:]
staten_pov = staten_pov.str.replace(r"\%","")
staten_med = pd.to_numeric(staten_med)
staten_pov = pd.to_numeric(staten_pov)


# staten median prediction
x = staten_med.to_numpy()
y = years
n = np.size(years)
m_x = np.mean(years)
m_y = np.mean(staten_med)
SS_xy = np.sum(y*x) - n*m_y*m_x
SS_xx = np.sum(x*x) - n*m_x*m_x
b_1 = SS_xy / SS_xx
b_0 = m_y - b_1*m_x
staten_med_2021 = b_1 * 2021 + b_0

# staten pov predictor
x = staten_pov.to_numpy()
y = years
n = np.size(years)
m_x = np.mean(years)
m_y = np.mean(staten_pov)
SS_xy = np.sum(y*x) - n*m_y*m_x
SS_xx = np.sum(x*x) - n*m_x*m_x
b_1 = SS_xy / SS_xx
```

```
    b_0 = m_y - b_1*m_x
    staten_pov_2021 = b_1 * 2021 + b_0


    final_demographic["Bronx"] = bronx_med_2021, bronx_pov_2021
    final_demographic["Brooklyn"] = brooklyn_med_2021, brooklyn_pov_2021
    final_demographic["Queens"] = queens_med_2021, queens_pov_2021
    final_demographic["Manhattan"] = manhattan_med_2021, manhattan_pov_2021
    final_demographic["Staten Island"] = staten_med_2021, staten_pov_2021
    final_demographic = final_demographic.transpose()
    final_demographic = final_demographic.rename(columns = {0: "2021: Median␣
 ↪Household Income", 1: "Poverty Rate"})

#     final_demographic.to_excel(summary_file, sheet_name = "Insight 1")
#     summary_file.save()



    return final_demographic


############ Function Call ############
insight1()
```

```
C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:12:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:14:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:43:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:45:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:74:
FutureWarning:
```

```
The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:76:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:106:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:108:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:138:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:140:
FutureWarning:

The default value of regex will change from True to False in a future version.
```

[93]:

	2021: Median Household Income	Poverty Rate
Bronx	39947.499924	29.350021
Brooklyn	53077.500109	22.425041
Queens	65737.500021	13.325019
Manhattan	77435.000055	17.525031
Staten Island	84499.999987	10.599989

[94]:

```
### Insight 1 Explanation

# The years 2000, 2006, 2010, and 2018 had demographic data for each NYC
↪borough on multiple web sources, however we could
# not find 2021 borough demographic data. Therefore, we used simple linear
↪regression modeling in order to predict median
# household income and poverty rate per borough in 2021. We used this to order
↪the boroughs by socioeconomic wealth, which
# fuels our hypothesis that lower income boroughs (Brooklyn & Bronx) have
↪access to less emergency medical resources than
# higher income boroughs (Staten Island). Also, Queens has a lower poverty rate
↪than Manhattan does but Later
```

```
# on, our grouping of boroughs based on median household income and poverty␣
↪rate are used to substantiate differences in
# EMS dispatch and response times.
```

[95]:
```
def insight2():
    df = pd.read_csv("2021_ems_data.csv")
    data = pd.DataFrame()
    data["Borough"] = df["Borough"]
    data["Disposition Code"] = df["Disposition Code"]

    good = data[(data["Disposition Code"] == 91) | (data["Disposition Code"] ==␣
↪92) | (data["Disposition Code"] == 82) | (data["Disposition Code"] == 94) |␣
↪(data["Disposition Code"] == 95)]
    good = df.groupby("Borough")["Disposition Code"].count()
    good = pd.DataFrame(good)
    good.rename(columns = {"Disposition Code" : "Count of cases Treated/
↪Transported"}, inplace = True)


    bad = data[(data["Disposition Code"] == 83) | (data["Disposition Code"] ==␣
↪96)]
    bad = bad.groupby("Borough")["Disposition Code"].count()
    bad = pd.DataFrame(bad)
    bad.rename(columns = {"Disposition Code" : "Count of Cases Dead"}, inplace␣
↪= True)


    dfs = [good, bad]
    in2 = pd.concat(dfs, axis = 1)
    in2["Total"] = in2["Count of cases Treated/Transported"] + in2["Count of␣
↪Cases Dead"]
    in2["% Treated/Transported"] = (in2["Count of cases Treated/Transported"] /␣
↪in2["Total"]) * 100

#     final.to_excel(summary_file, sheet_name = "Insight 2")
#     summary_file.save()

    return in2

############ Function Call ############
insight2()
```

[95]:
```
                   Count of cases Treated/Transported  Count of Cases Dead  Total  \
Borough
Bronx                                             240                    9    249
Brooklyn                                          434                    9    443
```

```
Manhattan                                  451                4    455
Queens                                     576                5    581
Staten Island                               98                2    100

              % Treated/Transported
Borough
Bronx                     96.385542
Brooklyn                  97.968397
Manhattan                 99.120879
Queens                    99.139415
Staten Island             98.000000
```

[96]:
```
### Insight 2 Explanation

# Within our cleaned dataset titled "2021_ems_data.csv", there were several
 ↪disposition codes indicating whether or not the
# case recieved medical attention on time, or whether they died without getting
 ↪access to EMS services. We grouped the
# dispostion codes into cases treated and trasported and cases dead, and found
 ↪a count of each per borough. Since the total
# number of cased greatly differed accross boroughs, we then found the
 ↪percentage of cases that were treated and transported
# per borough.

# As you can see, the lowest percentage of cases treated/transported were from
 ↪the Bronx, closely followed by Brooklyn, which
# substantiates our claim that lower income boroughs recieve less medical
 ↪attention. The highest percentage of cases
# treated/transported, however, was between Queens and Manhattan. We classified
 ↪Queens as a middle income borough, but
# Queens is home to more middle class families than any other borough in NYC
 ↪from our research on the borough, hence why they
# might have the best EMS treated/transported percentage or access to emergency
 ↪medical resources. Manhattan might have such
# a high EMS treated/transported percentage due to the fact that they have the
 ↪largest population density.

##### Grouped dispatch codes for cases treated/transported #####
# 82 => transporting patient
# 91 => condition corrected
# 92 => treated not transported
# 94 => treated and transported
# 95 => triaged at scene no transport

##### Grouped dispatch codes for cases dead #####
# 83 => patient pronounced dead
```

```
# 96 => patient gone on arrival
```

```
[97]: def insight3():
    data = pd.read_csv("2021_ems_data.csv")
    data = data.dropna()
    X = data['Dispatch Response Time'].values.reshape(-1,1)
    Y = data['Incident Response Time'].values.reshape(-1,1)
    times = np.array([X, Y])

    linear_regressor = LinearRegression()
    linear_regressor.fit(X, Y)
    Y_pred = linear_regressor.predict(X)


    data["Predicted"] = Y_pred

    data = data[(data["Borough"] == "Bronx") | (data["Borough"] == "Staten␣
↪Island") | (data["Borough"] == "Manhattan")]

    data["Residual"] = data["Incident Response Time"] - data["Predicted"]


    greater = data[data['Residual'] > 0]
    greater = greater.groupby("Borough")["Residual"].count()
    greater = pd.DataFrame(greater)
    greater.rename(columns = {"Residual" : "actual response time > predicted␣
↪response time"}, inplace = True)


    less = data[data['Residual'] <= 0]
    less = less.groupby("Borough")["Residual"].count()
    less = pd.DataFrame(less)
    less.rename(columns = {"Residual" : "actual response time < predicted␣
↪response time"}, inplace = True)


    dfs = [less, greater]
    final = pd.concat(dfs, axis = 1)

    final["Total"] = final["actual response time < predicted response time"] +␣
↪final["actual response time > predicted response time"]
    final["Percentage actual response time < predicted response time"] =␣
↪(final["actual response time < predicted response time"] / final["Total"]) *␣
↪100

    return final
```

```python

############ Function Call ############
insight3()
```

[97]:

```
                 actual response time < predicted response time  \
Borough
Bronx                                                      166
Manhattan                                                  210
Staten Island                                               41

                 actual response time > predicted response time  Total  \
Borough
Bronx                                                       30    196
Manhattan                                                  130    340
Staten Island                                               30     71

                 Percentage actual response time < predicted response time
Borough
Bronx                                                    84.693878
Manhattan                                                61.764706
Staten Island                                            57.746479
```

[98]:

```python
### Insight 3 Explanation

# Within our cleaned dataset titled "2021_ems_data.csv", there were two columns␣
 ↪relating to dispatch response time and
# incident response time for each case in 2021. The two are different because␣
 ↪the former is defined by how long it took for
# EMS services to respond to an emergency call, while the latter is defined by␣
 ↪how long it took for EMS services to
# arrive and provide their services.

# We hypothesized that there is a positive linear correlation between dispatch␣
 ↪response times and incident response times,
# so we ran an ordinary linear regression between the two variables. We then␣
 ↪found the RESIDUALS for incident response times,
# which is defined as the actual incident response time minus the predicted␣
 ↪incident response time based on our linear
# regression. For each borough, we counted the positive residuals and the␣
 ↪negetive residuals.

# Because the total number of cases differed by borough, we found the␣
 ↪percentage of cases for each borough where the
# actual response time was less than predicted.
```

```
# Conclusion: As you can tell, the Bronx, a lower income borough, had the␣
↪greatest percentage of their cases where  actual response
# time was less than predicted, as opposed to Manhattan and Staten Island. This␣
↪actually disproves our hypothesis, since we
# would expect a lower percentage of cases where actual response time is less␣
↪than predicted response time for
# lower income neighborhoods, based on our hypothesis.
```

[99]:
```
def insight4():
    df = pd.read_csv("cleaned_response_times.csv")
    in4=pd.DataFrame()
    in4["Mean"] = df.groupby("Borough")["Incident Response Time"].mean()
    in4["Median"] = df.groupby("Borough")["Incident Response Time"].median()
    in4=in4.sort_values(by="Mean", ascending = False)
    in4=in4.transpose()

    return in4

############ Function Call ############
insight4()
```

[99]:

Borough	Bronx	Queens	Staten Island	Brooklyn	Manhattan
Mean	607.466359	593.967595	577.194339	571.227630	569.674333
Median	603.529221	590.354677	568.168958	567.266569	569.709618

[100]:
```
### Insight 4 Explanation

# Conclusion: Lower income boroughs such as the Bronx have a higher mean and␣
↪median incident response time as compared to
# higher income boroughs such as Manhattan.

# We used aggregate methods within our pandas Data Frame in order to group each␣
↪borough by its mean and median Incident Response
# Times. We then sorted the values to compare the average response time in each␣
↪borough. We found that the Bronx had the
# highest mean and median incident response time and that Manhattan had the␣
↪lowest. However, some boroughs such as Brooklyn,
# which have a lower median income/poverty rate, had a lower average and median␣
↪response time.
```

[101]:
```
def insight5():
    df = pd.read_csv("cleaned_response_times.csv")
    response_stats=pd.DataFrame()
    response_stats["Standard Deviation"] = df.groupby("Borough")["Incident␣
↪Response Time"].std()
```

```
    response_stats["Max"] = df.groupby("Borough")["Incident Response Time"].
↪max()
    response_stats["Min"] = df.groupby("Borough")["Incident Response Time"].
↪min()
    response_stats=response_stats.sort_values(by="Standard Deviation",␣
↪ascending = False)
    response_stats=response_stats.transpose()

    return response_stats

############ Function Call ############

insight5()
```

[101]:
```
Borough                   Bronx    Brooklyn      Queens  Staten Island  \
Standard Deviation    32.351783   29.556556   24.636748      24.214621
Max                  662.784251  629.028300  640.371018     617.715414
Min                  565.021531  534.380453  559.172599     547.206999

Borough               Manhattan
Standard Deviation    18.576803
Max                  597.011098
Min                  543.618296
```

[102]:
```
### Insight 5 Explanation

# Conclusion: Lower income boroughs, including the Bronx and Brooklyn, have a␣
↪greater spread of incident response times
# than higher income boroughs, including Manhattan.

# The spread of the data varies significantly from borough to borough; the␣
↪Bronx has a greater range of incident response
# times as compared to Manhattan.
# This spread could be due to a variety of factors including population␣
↪density, the
# proximity of EMS services, etc. Lower income boroughs (Bronx and Brooklyn)␣
↪have a greater standard deviation than higher
# income boroughs (Staten Island and Manhattan), further supporting our initial␣
↪hypothesis.
```

3.2 Data Visualizations

We created data visualizations and included images of them in a seperate .pdf file to visually represent our insights (attached).

[103]:
```
## Import Modules: Do not Change this cell###
import plotly.express as px
import plotly.graph_objects as go
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

[104]:
```
def visual1(final):
    final = final.reset_index()
#     return final
#     df = pd.read_csv("2021_ems_data.csv")
#     data = pd.DataFrame()
#     data["Borough"] = df["Borough"]
#     data["Disposition Code"] = df["Disposition Code"]
#     data["Incident Status"] = (data["Disposition Code"] == "Bad").
 ↪where((data["Disposition Code"] == 83) | (data["Disposition Code"] == 96),␣
 ↪"Alive")


    fig = px.bar(final, x="Borough", y="% Treated/Transported")
    fig.show()
############ Function Call ############
x = insight2()
visual1(x)
```

[105]:
```
### Visualization 1 Explanation

# This visualization substantiates insight #2. The % treated/transported for␣
 ↪lower income boroughs (Bronx & Brooklyn) is
# lower than the percentage treated/transported for higher income boroughs␣
 ↪(Manhattan & Staten Island). Although Queens has
# a lower median household income and poverty rate than Staten Island does, it␣
 ↪has a higher percentage of treated/transported
# cases. We provided a potential explanation for why this may be in insight #2.
```

[106]:
```
def visual2():
    data = pd.read_csv("2021_ems_data.csv")
    bronx = data[data["Borough"]=="Bronx"]
#     brooklyn = data[data["Borough"] =="Brooklyn"]
    manhattan = data[data["Borough"] =="Manhattan"]
#     queens = data[data["Borough"]=="Queens"]
    staten_island = data[data["Borough"] == "Staten Island"]

    combined = pd.concat([bronx, manhattan, staten_island], ignore_index = True)
    fig = px.scatter(combined, x = 'Dispatch Response Time', y = 'Incident␣
 ↪Response Time', color = 'Borough', title = "Dispatch vs. Response Times Per␣
 ↪Case")
```

```
    fig.show()


############ Function Call ############
visual2()
```

[107]:
```
### Visualization 2 Explanation

# This visualization substantiates the regression in insight 3. As we can tell,␣
↪dispatch response time and incident
# response time have a strong, positive linear correlation. Response times by␣
↪boroughs are concentrated along this linear
# scatter plot, with Staten Island being most clustered at the bottom,␣
↪Manhattan being spread throughout, and the Bronx
# ahving the most sporadic spread.
```

[108]:
```
def visual3():
    df = pd.read_csv("cleaned_response_times.csv")
    fig = px.box(df, x = "Borough", y = "Incident Response Time", color =␣
↪"Borough", labels = {"Borough": "Borough","Incident Response Time":␣
↪"Incident Response Time (seconds)"})
    return fig.show()

############ Function Call ############
visual3()
```

[109]:
```
### Visualization 3 Explanation

# This visualization substantiates insight 4 and insight 5, where we␣
↪respectively calculated measures of center and measures
# of spread for each borough's incident response times. As you can tell the␣
↪median response time is higher in the Bronx
# and lower in Staten Island by the middle line in each box plot. However,␣
↪Brooklyn's median incident response time is close
# to that of Staten Island despite being significantly lower income with high␣
↪poverty rates. This might be because of denser
# population. The variability for lower income boroughs like the Bronx and␣
↪Brooklymn is much higher than that of higher income
# boroughs like Manhattan and Staten Island, however, as indicated by the␣
↪spread of the box plots.
```

[110]:
```
## Summary Files
```

[87]:
```
def summary1(final_demographic, in2, final, in4, response_stats, data, dfs, df):
    summary_file = pd.ExcelWriter("summary.xlsx", engine = "xlsxwriter")
    final_demographic.to_excel(summary_file, sheet_name = "Insight 1")
```

```
    in2.to_excel(summary_file, sheet_name = "Insight 2")
    final.to_excel(summary_file, sheet_name = "Insight 3")
    in4.to_excel(summary_file, sheet_name = "Insight 4")
    response_stats.to_excel(summary_file, sheet_name = "Insight 5")
    data.to_excel(summary_file, sheet_name = "Cleaned CSV")
    df.to_excel(summary_file, sheet_name = "Cleaned JSON")
    summary_file.save()



# Due to the nature of the cleaned API being in the format of a list of␣
 ↪dataframes, we were unable to add it to the summary
# file.

############ Function Call ############
a = insight1()
b = insight2()
c = insight3()
d = insight4()
e = insight5()
f = chunked_ems_data()
g = borough_data()
h = response_times()
summary1(a, b, c, d, e, f, g, h)
```

```
C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:12:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:14:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:43:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:45:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:74:
FutureWarning:
```

```
The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:76:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:106:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:108:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:138:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\AppData\Local\Temp\ipykernel_25844\1911405007.py:140:
FutureWarning:

The default value of regex will change from True to False in a future version.

C:\Users\diyam\Documents\miniconda\lib\site-packages\xlsxwriter\workbook.py:339:
UserWarning:

Calling close() on already closed file.
```

[111]:
```
# Cited Sources

# If you used any additional sources to complete your Data Analysis section,␣
↪list them here:

# Insight #1: https://www.geeksforgeeks.org/
↪linear-regression-python-implementation/
# Insight #3: https://realpython.com/linear-regression-in-python/
```