Title: Module 5 Capstone Project

Course: ALY 6140 Analytics System Technology

Academic Year: 2022-2023

Instructor: Professor Daya Rudramoorthi

Completion Date: 25th June 2023

Group Number: 9

Group Members: Shreyas Sadhale, Swathi Nadimpilli

Introduction:

This dataset offers details on a variety of infractions, such as disorderly premises, careless driving, trespassing, drinking while driving, smoking in stores, and equipment violations. Along with topographical information like borough, precinct and coordinates, it also provides demographic characteristics like age group, sex, and race.

a. Summons_Key: A distinguishing code for every summons file.
b. Summons_Date: The date summon was issued
c. Offense_Description: Information about the crime that was committed.
d. Law_Section_Number: The offense-specific law's section number
e. Law_Description: Law details for the offense are provided
f. Summon_Category_Type: The summon's category type
g. Age_Group: The age of the perpetrator of the incident
h. Sex: The gende of the person who committed the offense
i. Race: The perpetrator of the offense's race and ethnicity.
j. Jurisdiction_Code: Code identifying the offense's jurisdiction
k. Boro: The locality in which the offense took place
l. Precinct_Of_Occur: The police station where the crime was committed.
m. X_Coordinate_CD: The exact coordinates of the scene of the crime
n. Y_Coordinate_CD: The Y-coordinate of the scene of the offense
o. Latitude: The offense's location's latitude coordinate
p. Longitude: The location of the offense's longitude coordinate
q. New Georeferenced Column: A georeferenced point that represents the scene of the offense
r. Zip Codes: Zip code connected to the scene of the offense
s. Community Districts: The neighbourhood where the offense was committed
t. Borough Boundaries: The borough boundary that surrounds the scene of the offense
u. City Council Districts: The district in which the offense was committed
v. Police Precincts: The precinct connected to the scene of the crime.

The project's objectives are to examine a dataset of summons records and learn more about the patterns and trends associated with various infractions. The analysis seeks to comprehend how violations are distributed among distinct racial and geographic groups. Additionally the study seeks to find any noteworthy connections or correlations between jurisdictions, demographic and offense kinds.

Methods of Analysis Employed:
a. Descriptive Statistics: Calculate the descriptive statistics like counts, frequencies and percentages to get a general sense of the dataset and the distribution of the various variables.

b. Data Visualization: Create maps, graphs and charts to illustrate the links and patterns in the data to make is easier to analyze and spot trends.

c. Statistical Analysis: To ascertain the importance of relationships between the variables and gauge the potency of associations, use statistical tests like correlation analysis

d. Temporal Analysis: Analyze the distribution of the offenses across time and look for any seasonal patterns or temporal trends.

Rationale:

a. Availability: The dataset is easily obtainable and accessible, enabling us to work with the data for our study without difficulty.
b. Relevance To Law Enforcement: The dataset comprises data on a variety of offenses and the related information, including the description of the offense, the relevant statute section number, and the jurisdiction codes. Insights into the types of offenses that law enforcement agencies deal with can be gained from the analysis of this data, which can also help pinpoint problem areas
c. Geographical and Demographic Factors: The dataset contains geographic information like borough, precinct, and coordinates as well as demographic information like age group, sex and race. This enables us to investigate the connections between crimes and racial or regional characteristics, potentially exposing trends or discrepancies in legal procedures.
d. Range of Infractions: The dataset includes a variety of infractions such as trespassing, equipment violations, reckless driving and disorderly property. We can undertake a thorough examination of the many crime kinds and their features thanks to this diversity.
e. Potential Societal Impact: Examining the dataset can reveal information about the distribution of offenses across various racial and geographic groups. Policymakers, law enforcement groups, and community organizations may find these findings useful in addressing issues of social justice, crime prevention and resource allocation.

In light of these factors, our team is of the opinion that this dataset presents many possibilities for insightful research and can reveal insightful information regarding the criminal justice practices, crime patterns and potential equities.

Questions to Investigate

a. Distribution of Summons Categories Across Different Demographics:
To analyze the distribution of summons categories across different age groups, races and sexes, we would need a dataset that includes information on these variables along with the summons category types. By examining the frequencies or proportions of each summons category within each demographic group, you can identify any variations or patterns.

b. Monthly trend of Summons Issuance:
To understand the monthly trend of summons issuance, we would need a dataset that includes the date or month of each summons issuance. By aggregating the number of summons issued per month and plotting them over time, you can observe any patterns or changes in the issuance rate.

c. Distribution of Summons Category Types Across Differnet Boroughs:
   To explore the distribution of summons category types across different boroughs, we would need a dataset that includes information on both the summons category types and the borough in which they were issued. By examining the frequencies or proportions of each summons category within each borough, we can identify any geographical patterns or variations.

Exploratory Data Analysis:

Firstly, the python libraries which we are using in our project are:

a. Pandas: Pandas is a robust Python framework for handling and analyzing data. It offers data structures like Data Frames and series that make it possible to manage and analyze structured data effectively. Data cleansing, combining, filtering and aggregation are just a few of the activities that Pandas different functions and methods help with.
b. NumPY: The core python package for scientific computing is called NumPy. Large, multidimensional arrays and matrices are supported, and a number of mathematical operations can be performed n these arrays are also provided. For numerical operations, data manipulation and scientific computations , people frequently utilize NumPy.
c. Seaborn: A data visualization library built on top of Matplotlib is called Seaborn. It offers a sophisticated user interface for producing interesting and useful statistical visuals. The process of making plots like scatter plots, line plots, bar plots, histograms, and heatmaps is made easier by Seaborn. Additionally, it provides tools to improve the aesthetics of plots visually and facilitates statistical analysis and pattern recognition.
d. Requests: The Python package Requests is used to send HTTP requests. When getting data from APIs, scraping web pages, or dealing with web services, it makes the process of sending HTTP queries and managing the results simpler. An easy-to-use interface for interacting with web resources, handling headers, parameters, cookies, and other HTTP-related features is provided by the Requests library.
e. Matplotlib: Matplotlib is a well-liked Python data visualization library. It offers a complete set of classes and functions for building various kinds of interactive, animated, and static plots. You can create line graphs, scatter plots, bar plots, histograms, pie charts, and more with Matplotlib. In order to build visualizations that are both aesthetically pleasing and educational, it provides comprehensive customization choices for labels, titles, axes, colors, and styles.
f. Scikit-Learn (SKLearn): Scikit-learn (sklearn) is a well-known Python machine learning library. For tasks including classification, regression, clustering, dimensionality reduction, and model evaluation, it offers a wide range of tools and algorithms. A consistent and user-friendly interface is provided by Scikit-learn for creating and analyzing machine learning models. Additionally, it has tools for selecting features, preparing data, and choosing models.
g. Matplotlib: Matplotlib is a Matplotlib sub-library that offers a range of functions to make different kinds of plots. By offering a MATLAB-like interface, it makes the process of using Matplotlib to create visualizations simpler. For making line graphs, scatter plots, bar plots, histograms, and more, people frequently utilize Matplotlib.

The data which we have chosen is New York Criminal Summon Data. It is a real-time data which has a total of 21.7K rows and 22 columns. Now, we imported the data through an API.

The initial task was to directly import the code into our Python environment directly using the API link. So we have used the pandas library and requests library to import the data. To import the whole dataset, we used three different functions.

```
def get_data_from_api(url):
    response = requests.get(url)
    data = response.json()
    return data
def create_dataframe(data):
    df = pd.DataFrame(data)
    return df
def get_complete_dataframe(api_url):
    url = api_url
    complete_data = []
    offset = 0
    limit = 1000
    while True:
        url = api_url + f"?$offset={offset}&$limit={limit}"
        data = get_data_from_api(url)
        complete_data.extend(data)
        if len(data) < limit:
            break
        offset += limit
```

```
        offset += limit

    dataframe = create_dataframe(complete_data)
    return dataframe
# API URL
api_url = "https://data.cityofnewyork.us/resource/mv4k-y93f.json"
# Get the complete DataFrame
complete_dataframe = get_complete_dataframe(api_url)
```

The functions we defined are get_data_from_api , create_dataframe, get_complete_dataframe. So the data is then stored in ‘complete_dataframe’.

After importing the dataset, we then checked the head and tail of the dataset using the pandas. The head(10) provides the first 10 entries of the dataset while the tail(10) provides the last 10 entries of the dataset. As mentioned in the introduction, there are total 22 columns and the description of the columns has been provided above. Using the .shape function, we then check the total dimension of the dataset which comes as 21724 rows and 22 columns.

```
nyc_summon.head(10)
```

	summons_key	summons_date	offense_description	law_section_number	law_description	summons_category_type	age_group	sex	race	jurisdiction_code	...	x_coordinate_cd	y_coordinate_c
0	262694309	2023-01-28T00:00:00.000	DISORDERLY PREMISE	106-6	ABC	ABC	65+	F	WHITE HISPANIC	0	...	1006499	23356
1	262490655	2023-01-25T00:00:00.000	SMOKING IN A STORE	27-4274	(null)	(null)	18-24	M	WHITE HISPANIC	0	...	989330	18909
2	262604725	2023-01-27T00:00:00.000	RECKLESS DRIVING	1212	VTL	VTL	25-44	M	BLACK	0	...	995876	17482
3	264134676	2023-02-23T00:00:00.000	RECKLESS DRIVING	1212	VTL	VTL	25-44	M	WHITE HISPANIC	0	...	1019953	23898
4	265775874	2023-03-28T00:00:00.000	ALCOHOLIC BEVERAGE IN PUBLIC	10-125(2B)	(null)	(null)	25-44	M	WHITE	0	...	982661	17528
5	266023035	2023-03-31T00:00:00.000	OTHER ABC	9999	ABC	OTHER ABC	25-44	M	BLACK	0	...	1008434	21862
6	262692730	2023-01-29T00:00:00.000	RECKLESS DRIVING	1212	VTL	VTL	25-44	M	WHITE	0	...	995387	17814
7	265389517	2023-03-20T00:00:00.000	TRESPASS	140.05	Penal Law	TRESPASS	<18	M	WHITE	0	...	990954	21317
8	263040207	2023-02-05T00:00:00.000	RECKLESS DRIVING	1212	VTL	VTL	18-24	M	BLACK	0	...	1027947	26300
9	262046956	2023-01-17T00:00:00.000	RECKLESS DRIVING	1212	VTL	VTL	25-44	M	WHITE HISPANIC	0	...	1002655	23023

Head 10 values

```
nyc_summon.tail(10)
```

	summons_key	summons_date	offense_description	law_section_number	law_description	summons_category_type	age_group	sex	race	jurisdiction_code	...	x_coordinate_cd	y_coordina
21714	262466730	2023-01-25T00:00:00.000	FEDERAL MOTOR VEH. SAFETY REG	CFR 49	NYS Transportation	NYS TRANS	UNKNOWN	(null)	(null)	0	...	1008070	
21715	265444781	2023-03-19T00:00:00.000	CONSUMPTION OF ALCOHOL IN VEHICLE	1227	VTL	VTL	25-44	M	WHITE HISPANIC	0	...	1041743	
21716	264868706	2023-02-24T00:00:00.000	PICK UP FROM BUS STOP	145-6	Tax Law	TLC	45-64	M	BLACK	0	...	984945	
21717	265766844	2023-03-28T00:00:00.000	FEDERAL MOTOR VEH. SAFETY REG	CFR 49	NYS Transportation	NYS TRANS	UNKNOWN	(null)	(null)	0	...	989639	
21718	261286002	2023-01-03T00:00:00.000	ALCOHOLIC BEVERAGE IN PUBLIC	10-125(2B)	(null)	(null)	25-44	F	WHITE HISPANIC	0	...	1009865	
21719	264026696	2023-02-22T00:00:00.000	FEDERAL MOTOR VEH. SAFETY REG	CFR 49	NYS Transportation	NYS TRANS	UNKNOWN	(null)	(null)	0	...	982611	
21720	262033003	2023-01-17T00:00:00.000	UNLICENSED GENERAL VENDOR	20-453	Administrative Code	VENDING	45-64	M	BLACK	0	...	988912	
21721	264671196	2023-03-05T00:00:00.000	ALCOHOLIC BEVERAGE IN PUBLIC	10-125(2B)	(null)	(null)	25-44	M	BLACK	0	...	1014313	
21722	265612207	2023-03-24T00:00:00.000	NON PAYMENT OF FARE (OTHER)	1050.4(A)	(null)	(null)	25-44	M	BLACK	1	...	1052169	
21723	263577618	2023-02-14T00:00:00.000	FEDERAL MOTOR VEH. SAFETY REG	CFR 49	NYS Transportation	NYS TRANS	UNKNOWN	(null)	(null)	0	...	984166	

Tail 10 Values.

There are certain columns in the dataset which are randomly named and supposed to be renamed accordingly. The columns which we renamed are “Zip Codes”, “Community Districts”, “Borough Boundaries”, “City Council Districts”, “Police Precincts”. These columns are located at the very end of the dataset.

```
[18] nyc_summon = nyc_summon.rename(columns = {':@computed_region_efsh_h5xi' : 'Zip_Codes', ':@computed_region_f5dn_yrer' : 'Community Districts', ':@computed_region_yeji_bk3q' : 'Borough Bound
```

After renaming the required columns, we just took a look how the data is distributed in different data types. After running the command with .dtypes function, we got the following results.

```
nyc_summon.dtypes
```

```
summons_key                 int64
summons_date               object
offense_description        object
law_section_number         object
law_description            object
summons_category_type      object
age_group                  object
sex                        object
race                       object
jurisdiction_code           int64
boro                       object
precinct_of_occur           int64
x_coordinate_cd             int64
y_coordinate_cd             int64
latitude                  float64
longitude                 float64
geocoded_column            object
Zip_Codes                 float64
Community Districts       float64
Borough Boundaries        float64
City Council Districts    float64
Police Precincts          float64
dtype: object
```

Here as you can see there are almost equal number of categorical variables and equal number of numeric variables.

Categoric Variables: Summon Date, Offense Description, Law Section Number, Law Description, Summons Category Type, Age Group, Sex, Race, Boro, Geocoded Column.

Numeric Variable: Summons Key, Jurisdiction Key, Precinct of Occur, X Coordinate, Y Coordinate, Latitude, Longitude, Zip Codes, Community Districts, Borough Boundaries, City Council Districts, Police Precincts

Through this information we can further go ahead with our analysis.

```
nyc_summon.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21724 entries, 0 to 21723
Data columns (total 22 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   summons_key             21724 non-null  int64
 1   summons_date            21724 non-null  object
 2   offense_description     21724 non-null  object
 3   law_section_number      21724 non-null  object
 4   law_description         21724 non-null  object
 5   summons_category_type   21724 non-null  object
 6   age_group               21724 non-null  object
 7   sex                     21724 non-null  object
 8   race                    21724 non-null  object
 9   jurisdiction_code       21724 non-null  int64
 10  boro                    21724 non-null  object
 11  precinct_of_occur       21724 non-null  int64
 12  x_coordinate_cd         21724 non-null  int64
 13  y_coordinate_cd         21724 non-null  int64
 14  latitude                21724 non-null  float64
 15  longitude               21724 non-null  float64
 16  geocoded_column         21724 non-null  object
 17  Zip_Codes               21631 non-null  float64
 18  Community Districts     21723 non-null  float64
 19  Borough Boundaries      21723 non-null  float64
 20  City Council Districts  21723 non-null  float64
 21  Police Precincts        21723 non-null  float64
dtypes: float64(7), int64(5), object(10)
memory usage: 3.6+ MB
```

Going ahead to look how many null values we have in column, we can see we have only null values in Zip Codes with total 93 nulls and Community Districts, Borough Boundaries, City Council Districts, Police Precincts with one null value each. But there are certain columns like Sex, Race, Law Description wherein they have printed (null) for showing it as a null value which is the reason why we cannot see them in the Null Counts.

Data Cleaning:

- The dataset contains several null values that were identified during the data cleaning process. Specifically, the "Zip Codes" variable contained 93 null values, while the variables "Community Districts," "Borough Boundaries," "City Council Districts," and "Police Precincts" each had one null value.
- To handle these null values, we have to drop the corresponding rows from the dataset. When the number of missing values is low in comparison to the size of the dataset, dropping the rows with null values is a usual approach since it ensures accurate analysis while maintaining the integrity of the data.

```
Missing values:
 summons_key                0
summons_date               0
offense_description        0
law_section_number         0
law_description            0
summons_category_type      0
age_group                  0
sex                        0
race                       0
jurisdiction_code          0
boro                       0
precinct_of_occur          0
x_coordinate_cd            0
y_coordinate_cd            0
latitude                   0
longitude                  0
geocoded_column            0
Zip Codes                 93
Community Districts        1
Borough Boundaries         1
City Council Districts     1
Police Precincts           1
dtype: int64
```

```
Missing values after dropping:
 summons_key                0
summons_date               0
offense_description        0
law_section_number         0
law_description            0
summons_category_type      0
age_group                  0
sex                        0
race                       0
jurisdiction_code          0
boro                       0
precinct_of_occur          0
x_coordinate_cd            0
y_coordinate_cd            0
latitude                   0
longitude                  0
geocoded_column            0
Zip Codes                  0
Community Districts        0
Borough Boundaries         0
City Council Districts     0
Police Precincts           0
dtype: int64
```

After getting known this basic information about the data, we then proceed to calculate the total count of the unique entities in each column.

```
[28] nyc_summon['boro'].value_counts().sort_index()

BRONX            5730
BROOKLYN         8144
MANHATTAN        3867
NEW YORK           40
QUEENS           3603
STATEN ISLAND     340
Name: boro, dtype: int64
```

Checking for boro which represents the areas where the crime occurred, we have a total of 6 unique entries and as we can see Brooklyn (8144) is more prone towards the crimes where as New York (40) is the least.

```
[29] nyc_summon['race'].value_counts().sort_index()

(null)                            5597
AMERICAN INDIAN/ALASKAN NATIVE     143
ASIAN / PACIFIC ISLANDER          1112
BLACK                             7460
BLACK HISPANIC                    1481
OTHER                               66
UNKNOWN                            225
WHITE                             1308
WHITE HISPANIC                    4332
Name: race, dtype: int64
```

We have overall 8 different races involved in crimes with total of 5597 nulls which means they are not listed in the data. Among the listed ones we have Black Race with maximum crimes and others race with least with just 66 crimes.

```
[32] nyc_summon['sex'].value_counts().sort_index()

(null)     5597
F          1751
M         14295
U            81
Name: sex, dtype: int64
```

Male, Female and Unknown are the three different types of sex involved in crimes. In this scenario too, we have maximum null values of 5597, then the majority of the crimes are committed by men 14,295, the count for female is 1751 and the unknown genders are 81.

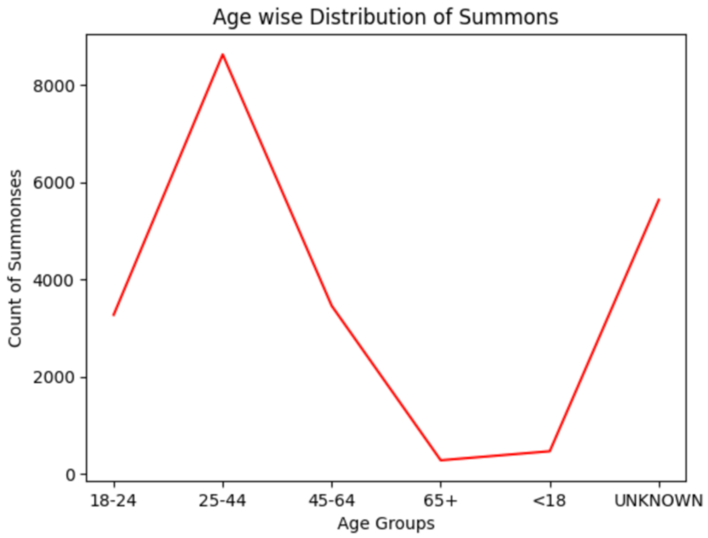
```
[33] nyc_summon['jurisdiction_code'].value_counts().sort_index()

0    17955
1      766
2     3003
Name: jurisdiction_code, dtype: int64
```

Now looking at the offense jurisdiction code, there are majorly type 0 offenses committed with 17,955 cases and least are type 1 offenses with 766 cases.

Moving ahead with the data visualization part:

1. The first graph which we plotted is a line graph based on different age groups involved in crime.

```
[6] data1 = nyc_summon['age_group']
    counts = data1.value_counts().sort_index()
    plt.plot(counts.index, counts.values, color = 'red')
    plt.xlabel('Age Groups')
    plt.ylabel('Count of Summonses')
    plt.title('Age wise Distribution of Summons')
    plt.show()
```

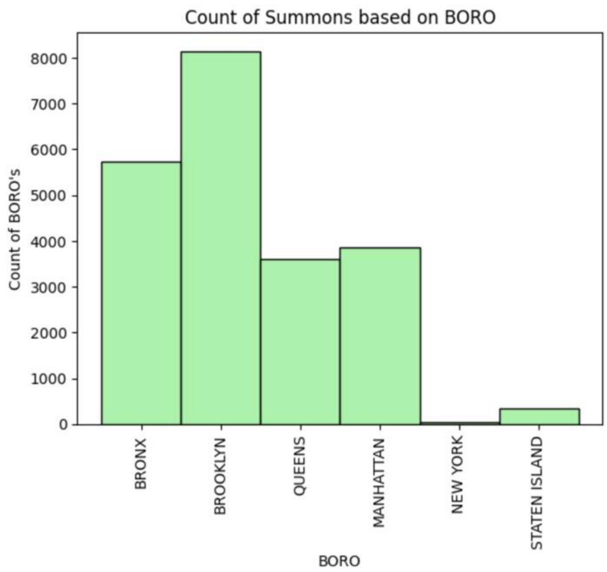


So here we uniquely set the age_group column in data1 dataframe then we took the counts of each unique entity and by using matplotlib, we plotted the line graph of ages with their values and indexes.

2. The second graph is the histogram which shows the comparison of crimes committed in different types of boros.

```
bar1 = sns.histplot(nyc_summon['boro'],
                    color = 'lightgreen')
bar1.set_xticklabels(bar1.get_xticklabels(), rotation = 90);
plt.xlabel("BORO")
plt.ylabel("Count of BORO's")
plt.title("Count of Summons based on BORO")
```

```
<ipython-input-35-b4ba552145a6>:3: UserWarning: FixedFormatter should only be used together with FixedLocator
  bar1.set_xticklabels(bar1.get_xticklabels(), rotation = 90);
Text(0.5, 1.0, 'Count of Summons based on BORO')
```

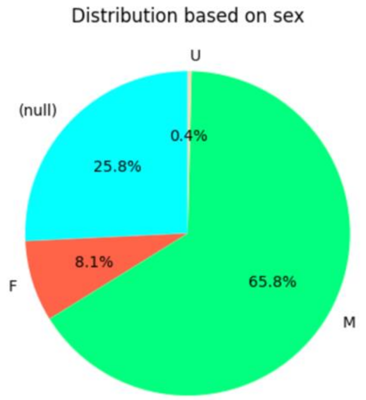


Here we used the seaborn library to plot the histogram. This gives an idea about the BORO in which maximum crimes took place which is the Brooklyn and the least crimes as seen in New York.

3. The next one is a Pie Chart depicting the percentage of Genders involved in Crime.

```
colors = ['cyan', 'tomato', 'springgreen', 'peachpuff']
plt.pie(u_sex, autopct = '%.1f%%', labels = u_sex.index, startangle= 90, colors = colors)
plt.title('Distribution based on sex')
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



So firstly we assigned the colors which we will be filling in our pie chart. Then using the matplotlib library, we plotted the pie chart with the percentage values round to 1 decimal place.
As we can see, more than 50% crimes are committed by Men, 8.1% by Women, 0.4% by unknown genders and almost $1/4^{th}$ of the data has null values contributing 25.8% which we will need to clean for our further analysis.

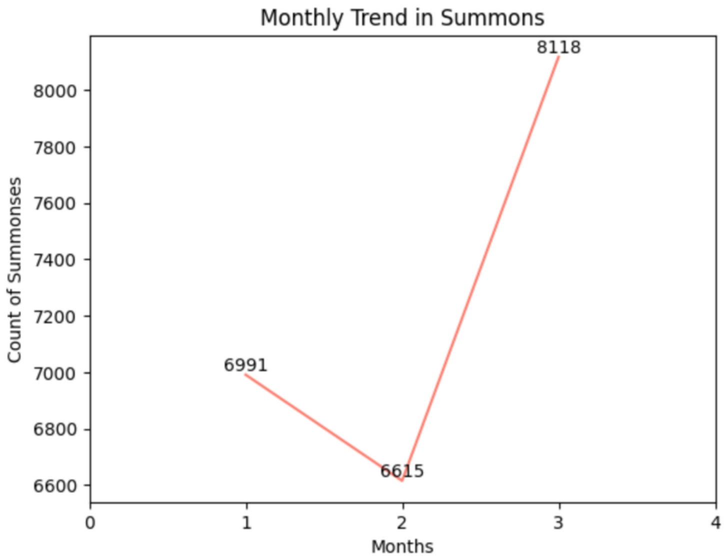
4. Next comes the Line graph again but here, we have shown the monthly analysis of crimes occurring. We have the data recorded for the year 2023 March.

```
[38] nyc_summon1 = nyc_summon
     nyc_summon1['summons_date'] = pd.to_datetime(nyc_summon1['summons_date'])
     nyc_summon1['month'] = nyc_summon1['summons_date'].dt.month
     month_counts = nyc_summon1.groupby('month').size()
     month_counts
```

```
month
1    6991
2    6615
3    8118
dtype: int64
```

Here, we firstly used the to_datetime function from pandas library to separate the date and month and year from the summons_date column in our dataset. Then by using the groupby() function, we grouped the months and given the output of total counts by using the size() for each month.

```
[11] plt.plot(month_counts.index, month_counts.values, marker = '', color = 'salmon')
     plt.xticks(range(len(month_counts.index)+2))
     for x,y in zip(month_counts.index, month_counts.values):
       plt.text(x,y, str(y), ha = 'center', va = 'bottom')
     plt.title('Monthly Trend in Summons')
     plt.xlabel('Months')
     plt.ylabel('Count of Summonses')
     plt.show()
```



By making use of the matplotlib library, we then plotted the line graph with labels indicating the count for each month.

5. The correlation matrix is an important factor to be considered as it shows the inter-relatability among the different variables in the dataset.

```
corr = nyc_summon.corr().round(2)
corr
```

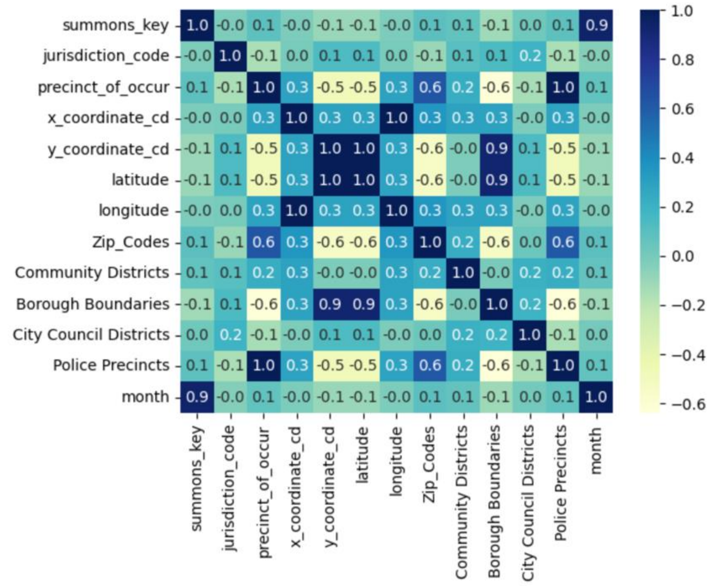
```
<ipython-input-44-09bc74d25d10>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns
  corr = nyc_summon.corr().round(2)
```

	summons_key	jurisdiction_code	precinct_of_occur	x_coordinate_cd	y_coordinate_cd	latitude	longitude	Zip_Codes	Community Districts	Borough Boundaries	City Council Districts	Police Precincts	month
summons_key	1.00	-0.02	0.08	-0.02	-0.10	-0.10	-0.02	0.09	0.07	-0.12	0.02	0.08	0.94
jurisdiction_code	-0.02	1.00	-0.15	0.00	0.13	0.13	0.00	-0.10	0.10	0.14	0.19	-0.15	-0.04
precinct_of_occur	0.08	-0.15	1.00	0.29	-0.49	-0.49	0.29	0.63	0.22	-0.63	-0.14	1.00	0.07
x_coordinate_cd	-0.02	0.00	0.29	1.00	0.29	0.29	1.00	0.35	0.29	0.29	-0.02	0.28	-0.02
y_coordinate_cd	-0.10	0.13	-0.49	0.29	1.00	1.00	0.29	-0.55	-0.02	0.92	0.13	-0.48	-0.10
latitude	-0.10	0.13	-0.49	0.29	1.00	1.00	0.29	-0.55	-0.02	0.92	0.13	-0.48	-0.10
longitude	-0.02	0.00	0.29	1.00	0.29	0.29	1.00	0.35	0.29	0.29	-0.02	0.28	-0.02
Zip_Codes	0.09	-0.10	0.63	0.35	-0.55	-0.55	0.35	1.00	0.23	-0.55	0.03	0.61	0.09
Community Districts	0.07	0.10	0.22	0.29	-0.02	-0.02	0.29	0.23	1.00	-0.01	0.21	0.22	0.06
Borough Boundaries	-0.12	0.14	-0.63	0.29	0.92	0.92	0.29	-0.55	-0.01	1.00	0.19	-0.64	-0.11
City Council Districts	0.02	0.19	-0.14	-0.02	0.13	0.13	-0.02	0.03	0.21	0.19	1.00	-0.15	0.01
Police Precincts	0.08	-0.15	1.00	0.28	-0.48	-0.48	0.28	0.61	0.22	-0.64	-0.15	1.00	0.07
month	0.94	-0.04	0.07	-0.02	-0.10	-0.10	-0.02	0.09	0.06	-0.11	0.01	0.07	1.00

By using the corr() function, we plot the correlation matrix with rounded values upto two decimal places.

```python
corrplot = sns.heatmap(corr, annot = True, cmap = 'YlGnBu', fmt = "0.1f")
corrplot
```

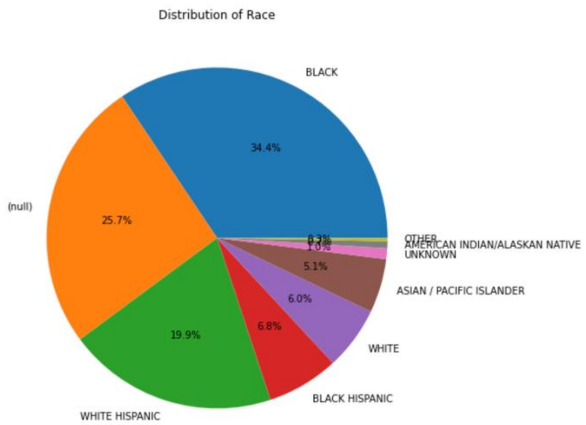
<Axes: >



This is the heatmap depicting the inter-relatability in detail. We used the seaborn library to plot the heatmap.

6. The pie-chart based on percentages of crimes committed by different listed races.

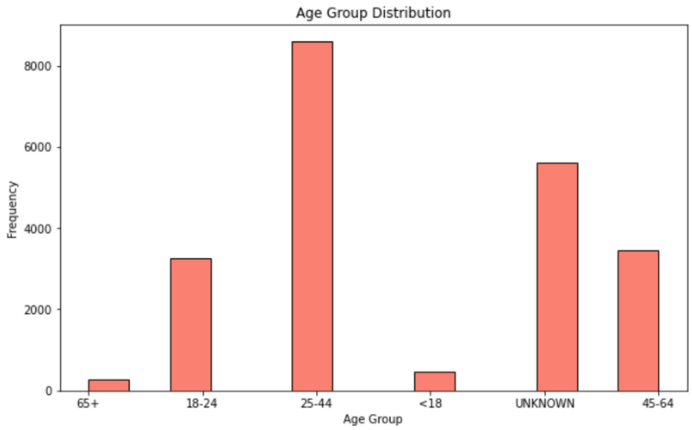
```python
# Count the occurrences of each race
race_counts = nyc_summon['race'].value_counts()

# Plot a pie chart
plt.figure(figsize=(8, 8))
plt.pie(race_counts.values, labels=race_counts.index, autopct='%1.1f%%')
plt.title('Distribution of Race')
plt.show()
```



From the following pie-chart we can conclude that Black race is majorly involved in crimes contributing a total of 34.4% followed by White Hispanic with almost 20%. Here too, almost 1/4th of the data is filled by Null values contributing to almost 25.7%.

7. A bar graph representing the crimes by different age groups.

```python
# Plot a histogram of ages
plt.figure(figsize=(10, 6))
plt.hist(nyc_summon['age_group'], edgecolor='black', bins = 14, color = 'salmon')
plt.xlabel('Age Group')
plt.ylabel('Frequency')
plt.title('Age Group Distribution')
plt.show()
```

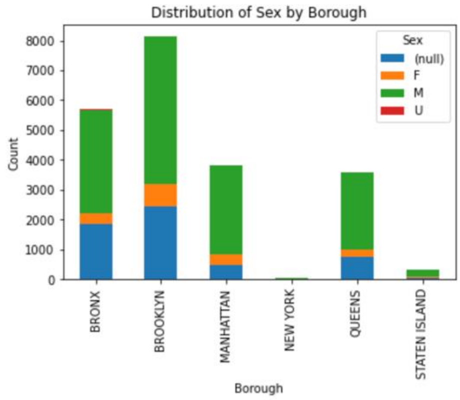


We have the age-groups 25-44 highly involved in crimes of more than 8000 cases, followed by the Unknown age groups (Unlisted ages), age group 18-24 & 45-64 share a moderate count of crimes, the least are minor children (<18) and people with age 65+.

8. A stacked bar graph based on different sex involved in crime at different BOROs.

```python
borough_sex_counts = nyc_summon.groupby(['boro', 'sex']).size().unstack()

plt.figure(figsize=(10, 6))
borough_sex_counts.plot(kind='bar', stacked=True)
plt.xlabel('Borough')
plt.ylabel('Count')
plt.title('Distribution of Sex by Borough')
plt.legend(title='Sex')
plt.show()
```

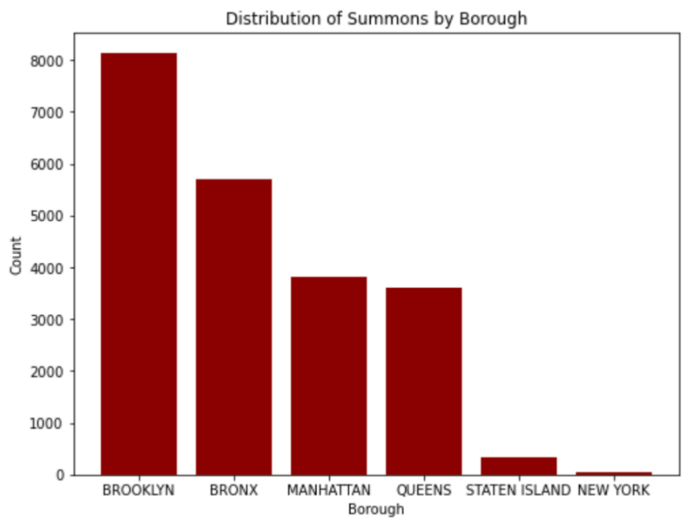
```
<Figure size 720x432 with 0 Axes>
```



This stacked bar graph has green depicting Male, Orange depicting Female, Red depicting Unknown and Blue depicting Null. This gives good insight on how the crimes are occurred by the people with different types of sex at different places.

9. The boro count of total crimes.

```
plt.figure(figsize=(8, 6))
borough_counts = nyc_summon['boro'].value_counts()
plt.bar(borough_counts.index, borough_counts.values, color = 'darkred')
plt.xlabel('Borough')
plt.ylabel('Count')
plt.title('Distribution of Summons by Borough')
plt.show()
```

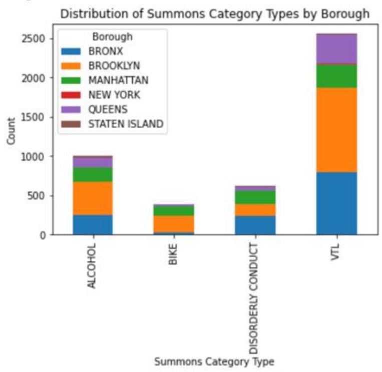


This is a bar graph and is arranged in the descending order of crimes at boros.

10. Crimes of different categories at different Boros.

```
# Select specific types from summons_category_type variable
selected_types = ['BIKE', 'ALCOHOL', 'DISORDERLY CONDUCT', 'VTL']

# Filter the DataFrame based on selected types
filtered_data = nyc_summon[nyc_summon['summons_category_type'].isin(selected_types)]

# Create a cross-tabulation of summons_category_type and boro
cross_tab = pd.crosstab(filtered_data['summons_category_type'], filtered_data['boro'])

# Plot the stacked bar chart
plt.figure(figsize=(10, 6))
cross_tab.plot(kind='bar', stacked=True)
plt.xlabel('Summons Category Type')
plt.ylabel('Count')
plt.title('Distribution of Summons Category Types by Borough')
plt.legend(title='Borough')
plt.show()
```

```
<Figure size 720x432 with 0 Axes>
```



So there are a total of types of crimes mainly Bike, Alcohol, Disorderly Conduct and VTL. The following bar graph shows their occurrences at different boros depicted in different colors. As we can see the major type of crime committed is the VTL. Brooklyn and Bronx are the two prone areas where it has occurred the most.

Predictive Models:

We are yet to build the models so we are just mentioning which models we chose and how we are going to proceed.

Before moving forward with the predictive modelling, we still have our dataset unclean. So,we will have to remove the null values present in the columns of the dataset. After that as seen in the above correlation heatmap, we have certain variables which are highly correlated with each other which brings up multicollinearity. So we will also have to remove the multicollinearity either by making use of VIF which is Variance Inflation Factor or PCA which stands for Principal Component Analysis.

Model1:

So for the first model, we are gonna predict the summons category type. For the following prediction, we are going to make use of Classification method of KNN (K-Nearest Neighbour).

Steps involved:

a. Choosing the target variable and the features: The features and the target variable are first taken from the nyc_summon DataFrame. The features include columns for 'age_group', 'race','sex', 'offense_description', and 'jurisdiction_code'. 'summons_category_type' is the desired variable.
b. Performing One-Hot Encoding: The OneHotEncoder from Scikit-Learn is used to one-hot encode the category features. In order to prevent multicollinearity, the parameter drop='first' is initialized in the encoder, causing it to ignore the first category of each feature. The category features are transformed using the fit_transform function into binary columns. The encoded_features DataFrame contains the final encoded features.
c. Dividing the dataset: The train_test_split function of the scikit-learn library is used to divide the encoded features (X) and the target variable (y) into training and testing sets. 30% of the data is used for testing (X_test, y_test) and 70% of the data is used for training (X_train, y_train). The percentage of the data to be set aside for testing is indicated by the test_size=0.3 parameter.
d. Converting the feature column names to strings: After one-hot encoding, the feature column names are encoded as numbers; therefore, they must be converted to strings in order to work with the KNN classifier. The.astype(str) function is used on the columns attribute of the X_train and X_test DataFrames to accomplish this.
e. The KNN Classifier: KNeighborsClassifier() is used to initialize the KNN classifier. The model is then adjusted to the training set by invoking the classifier object's.fit() function with the arguments X_train and y_train.
f. Predicting and Determining the Accuracy: By using the classifier object's.predict() method and giving X_test as a parameter, the trained model is utilized to predict the target variable for the test data. Using the accuracy_score function of scikit-learn, the model's accuracy is determined by contrasting the predicted values with the actual values from the y_test.
g. Preparing the results: Scikit-Learn's classification_report function, which offers precision, recall, an F1-score, and support for each class in the target variable, is used to create the classification report.

Model2:

- Random Forest is capable of capturing non-linear relationships between variables, It can provide more accurate predictions by considering the interactions and dependencies among the variables. We have taken Summon_category_Type as target variable and 'age_group', 'race','sex', 'offense_description', and 'jurisdiction_code'. 'summons_category_type' as feature variables.
- The OneHotEncoder from Scikit-Learn is used to one-hot encode the category features. In order to prevent multicollinearity, the parameter drop='first' is initialized in the encoder, causing it to ignore the first category of each feature. The category features are transformed using the fit_transform function into binary columns. The encoded_features DataFrame contains the final encoded features.
- The target variable and the encoded features are split into training and testing sets using the train_test_split function of the scikit-learn module. 70% of the data is utilised for training (X_train, y_train), while 30% is used for testing (X_test, y_test). The test_size=0.3 option indicates how much of the data should be reserved for testing.
- We have used Random Forest classifier function to initialize the classifier and then we used model.fit function with the training data.
- Further we have used model.predict and accuracy_score functions to predict the accuracy and generated a classification report using classification_report function.
- The summons category type was predicted using a random forest model that took in features such as age group, race, sex, offense description, and jurisdiction code.
- The model's accuracy of 97.1% was impressive. This high level of accuracy shows that the features chosen have a significant impact on predicting the type of summons issued.
- The model can effectively categorize the summonses into their appropriate categories based on the specified features, as shown by the accuracy of 97.1%. The nature of the various summonses issued in NYC can be better understood and analyzed using this information.

Interpretive & Conclusions:

- The data offers useful insights into the NYPD's enforcement activities and gives the chance to look at patterns and trends in the issue of criminal summonses. The application of Random Forest and KNN models on the provided dataset yielded promising results, with accuracies of 97% and 91%, respectively. These models aid in a better understanding of police enforcement operations in NYC by offering insightful predictions into the classification of criminal summonses.
- The Random Forest model achieved an impressive accuracy of 97%. This high accuracy suggests that random forest is the best model for the selected features and has significant predictive power in determining the category of the summons issued.

References

1. *NYC Open Data*. NYC Criminal Summons Data
   URL: https://data.cityofnewyork.us/Public-Safety/NYPD-YTD-Criminal-Summons-Summary-Dashboard/6c4k-4mp6

2. *SciKit Learn*. Linear Regression.
   URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

3. *Adam Hayes*, April 29, 2023. Multiple Linear Regression Definition, Formula and Example.
   URL: https://www.investopedia.com/terms/m/mlr.asp

4. *NeuralNine*, August 28, 2021. Linear Regression From Scratch in Python (Mathematical).
   URL: https://www.youtube.com/watch?v=VmbA0pi2cRQ

5. *CS Dojo*, June 11, 2018. Intro to Data Analysis/Visualization with Python, Matplotlib and Pandas | Matplotlib Tutorial.
   URL: https://www.youtube.com/watch?v=a9UrKTVEeZA&t=2s

6. *Matt Macarty*, November 4, 2019. Introduction to Line Plot Graphs with matplotlib Python.
   URL: https://www.youtube.com/watch?v=AYorFcI1MTU

7. *Jie Jenn*, October 25, 2021. How to Plot a Bar Graph with matplotlib for Beginners
   URL: https://www.youtube.com/watch?v=zwSJeIcRFuQ

8. *Ishaan Sharma*, March 20, 2021. Pie and Donut Chart in matplotlib Python.
   URL: https://www.youtube.com/watch?v=X9v5iTW3YA8

9. Jie Jenn, December 26, 2022. Plot a Stacked Bar Graph Using Matplotlib for Beginners.
   URL: https://www.youtube.com/watch?v=KiB1c8oWxJc