LONDON METROPOLITAN UNIVERSITY

Islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CC5067NI Smart Data Discovery

60% Individual Coursework

Submission : Final Submission

Academic Semester: Spring Semester 2025

Credit: 15 credit semester long module

Student Name: Pooja Kumari Yadav

London Met ID: 23056290

College ID: NP01CP4S240011

Assignment Due Date: Wednesday, May 14, 2025

Assignment Submission Date: Thursday, May 15, 2025

Submitted To: Dipeshor Silwal

I confirm that I understand my coursework needs to be submitted online via MST Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.

Table of Contents

1.Data understanding 1

2.Data preparation 6

2.1:Import the dataset 6

2.2:Provide your insight on the **information** and **details** that the provided dataset carries 9

2.3:Convert the columns "Created Date" and "Closed Date" to datetime datatype and create a new column "Request_Closing_Time" as the time elapsed between request creation and request closing 13

2.4:Write a python program to drop irrelevant Columns which are listed below. 15

2.5:Write a python program to remove the NaN missing values from updated dataframe. 16

2.6:Write a python program to see the unique values from all the columns in the dataframe. 18

3.Data analysis 19

3.1:Write a Python program to show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame. 19

3.2:Write a Python program to calculate and show correlation of all variables . 21

4.Data exploration 23

4.1.Provide four major insights through visualization that you come up after data mining. 23

4.1.1Complaint Type 23

4.1.2 Borough 25

4.1.3:Average Complaint by closing time 26

4.1.4:Monthly complaint Value 29

4.2:Arrange the complaint types according to their average 'Request_Closing_Time', categorized by various locations. Illustrate it through graph as well. 32

5.Statistical Testing 34

5.1:Test 1 34

5.1.1:State the Null Hypothesis (H0) and Alternate Hypothesis (H1) 34

5.1.2:Perform the statistical test and provide the p-value. 35

5.1.3:Interpret the results to accept or reject the Null Hypothesis. 35

5.2:Test 2 36
5.2.1:State the Null Hypothesis (H0) and Alternate Hypothesis (H1). 36
5.2.2:Perform the statistical test and provide the p-value. 37
5.2.3:Interpret the results to accept or reject the Null Hypothesis. 38
6.Conclusion 39
7.References 40

Table of tables

Table 1: information of the data 3

Table of figures

Figure 1: importing the dataset 6
Figure 2: Error of the dataset 6
Figure 3: correct code of the dataset 7
Figure 4:Output of the dataset 8
Figure 5: output of the dataset 8
Figure 6: information and details are described 9
Figure 7: information and details of the columns 10
Figure 8: information and details of the head 10
Figure 9: information and details of the sum 11
Figure 10:error datatype of datetime 13
Figure 11: correct datatype of datetime 13
Figure 12: answer of the datatype of datetime 14
Figure 13:drop irrelevant Columns 15
Figure 14: output of the drop irrelevant columns 15
Figure 15: before removing the NAN value 16
Figure 16: After removing the NAN Value 17
Figure 17: unique values from all the columns in the dataframe 18
Figure 18: showing the summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame. 19
Figure 19: calculating and showing all the variables 21
Figure 20: complaint Type in bar graph 23
Figure 21: Borough in pie chart form 25
Figure 22: error code of the average complaint by closing time 26
Figure 23: correct code of the average complaint by closing time 27
Figure 24: output of the average complaint by closing time 28
Figure 25: Error of the monthly complaint volume 29
Figure 26:correct code of the monthly complaint volume 30
Figure 27: Output of the monthly complaint Volume 31
Figure 28:error code according to their average 'Request_Closing_Time' **Error! Bookmark not defined.**
Figure 29:Correct code according to their average 'Request_Closing_Time' ..... **Error! Bookmark not defined.**
Figure 30:output of the according to their average 'Request_Closing_Time' 33
Figure 31:Test 1: Whether the average response time across complaint types is similar or not. ..... **Error! Bookmark not defined.**

Figure 32:Whether the type of complaint or service requested and location are related. .....**Error! Bookmark not defined.**
Figure 33: output of the Whether the type of complaint or service requested and location are related. 37
Figure 34: output of the Whether the type of complaint or service requested and location are related. 38

14% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

- **62** Not Cited or Quoted 13%
  Matches with neither in-text citation nor quotation marks
- **2** Missing Quotations 0%
  Matches that are still very similar to source material
- **2** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

- 7% Internet sources
- 2% Publications
- 12% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Document Details

Submission ID
trn:oid:::3618:95853386

Submission Date
May 14, 2025, 5:12 PM GMT+5:45

Download Date
May 14, 2025, 5:13 PM GMT+5:45

File Name
Smart data discovery (Pooja Kumari Yadav) (1).docx

File Size
31.0 KB

25 Pages

5,112 Words

26,276 Characters

1.Data understanding

The dataset titled is “Customer Service Requests from 2010 to present” which contains detailed records of 311 service requests made by residents of New York (NYC). The 311 system is a non-emergency service which allows residents to report issues, request services or obtain information related to city services. This requests include a wide range of complaints ad service need such as Noise complaints, Illegal Parking, Blocked driveways, water system problems, Steet light issues, sanitation concerns and many more public service topics are there. Each entry in the dataset represents a single request, filled through phone, mobile app, online portal or other sources. The dataset records who reported the issue when and where occurred, what type of complaint it was and how and when the issue was resolved. The dataset used in this analysis is titled Customer Service Requests from 2010 to Present, containing detailed records of 311 service requests submitted by New York City residents. For this project, a sample of the first 1000 rows was loaded from the CSV file for initial exploration. The dataset includes various fields such as Complaint Type, Created Date, Closed Date, Borough, Status, Agency, and geographic details like Latitude and Longitude. These fields provide a mix of categorical, numerical, and datetime data, giving valuable insights into public issues and service efficiency. Understanding this dataset helps prepare for further cleaning, exploration, and analysis by identifying key columns, relevant data types, and potential data quality issues such as missing values or inconsistencies (Anon., 2011; Dayal, 2011)

Some primary objects of this dataset is to:

- It help to analyze trends I civic complaints across time and location.
- It support city agencies in improving public services.
- It allows data scientists and analytics to apply data wrangling, visualization and statistical techniques to extract meaningful insights.

From these datasets it can also identify neighborhoods report the most complaints. It can also measure the response time of city services. It can also determine the seasonal patterns of the specific complaint types. It can also visualize complaint hotspot across boroughs. It can also perform hypothesis testing to see if location impacts type or response time.

Looking into the future, this dataset holds potential for predictive modeling, where machine learning algorithms can forecast complaint patterns, predict areas with rising service demands, or even recommend resource allocation for faster resolution. Over time, as more data accumulates, trends can be analyzed to inform better city management, improve public services, and enhance citizen satisfaction. Predictive analytics can help urban planners identify at-risk areas or seasonal patterns and take preventive measures, ultimately improving the quality of life in the city.

The advantages of this dataset include its rich historical records, large volume, and wide variety of complaints, making it suitable for diverse analyses. It supports transparency by showing how public concerns are handled and resolved. However, the disadvantages include possible data quality issues such as missing or inaccurate entries, inconsistencies across years, or biased reporting (some neighborhoods may report more due to better access or awareness). Additionally, the dataset may lack sensitive or internal agency details that could provide a fuller context, limiting the depth of certain analyses.

The importance of this dataset lies in its ability to support data-driven decision-making for city planners, government agencies, and policymakers. By understanding complaint patterns and resolution times, the city can improve service delivery, allocate resources more efficiently, and address community needs more effectively. Moreover, it empowers citizens and researchers to hold authorities accountable and contribute insights for urban improvement, fostering a culture of open data and civic engagement.

Table 1: information of the data

S.N	Column Name	Description	Data type
1	Unique key	A unique identifier for each 311 services request.	Integer
2	Created Date	Date and time the request was created	String
3	Closed Date	Date and time the request was closed	String
4	Agency	Symbol of the agency handling the request	String
5	Agency Name	Full name of the responsible agency	String
6	Complaint Type	The category of the complaint (e.g., Noise, Rodent)	String
7	Descriptor	Detailed description of the complaint	String
8	Location Type	Type of location like street or building	String
9	Address type	Type/classification of the address	String
10	Incident Address	Street address of the incident	String
11	Incident Zip	ZIP code where the issue occurred	float
12	Street name	Street name of the incident	String
13	Cross Street 1	First nearby cross street	String
14	Cross Street 2	Second nearby cross street	String
15	Intersection Street 1	First intersection near the incident	String
16	Intersection Street 2	Second intersection near the incident	String
17	City	City where the complaint was made	String
18	Landmark	Nearby landmark if available	String
19	Facility Type	Type of city facility (if relevant)	String
20	Status	Current status (e.g., Open, Closed)	String
21	Due date	Expected due date for resolution	Datetime (string in raw

22	Borough	Borough in NYC (e.g., Manhattan, Bronx	String
23	Community Board	Community board responsible	String
24	Resolution Action Updated Date	Last date when resolution info was updated	Datetime (string in raw
25	X Coordinate (State Plane)	State Plane X coordinate of the incident	Float Float
26	Y Coordinate (State Plane)	State Plane Y coordinate of the incident	String
27	Park Facility Name	Name of the park facility if applicable	String
28	Park Borough	Borough of the park facility	String
29	School Name	Name of school (if related)	String
30	School Number	School number	String
31	School Region	Region code for school	String
32	School Code	NYC school code	String
33	School Phone Number	Phone number of school	String
34	School Address	Address of school	String
35	School City	City of school	String
36	School State	State of school	String
37	School Zip	ZIP code of school	String
38	School Not Found	Indicator whether school is not found	String
39	School or Citywide Complaint	Indicates school-specific or citywide complaint	String

40	Vehicle Type	Type of vehicle involved (if any)	String
41	Taxi Company Borough	Borough where taxi company is located	String
42	Taxi Pick Up Location	Location where taxi picked up a passenger	String
43	Bridge Highway Name	Bridge/highway related to complaint	String
44	Bridge Highway Direction	Direction on bridge/highway	String
45	Road Ramp	Ramp near complaint location	String
46	Bridge Highway Segment	Segment of bridge/highway	String
47	Garage Lot Name	Name of garage/lot	String
48	Ferry Direction	Direction of ferry (e.g., Northbound)	String String
49	Ferry Terminal Name	Ferry terminal name	
50	Latitude	Latitude of the incident location	Float
51	Longitude	Longitude of the incident location	Float
52	Location	Combined latitude and longitude	String

2.Data preparation

2.1:Import the dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew, kurtosis
df = pd.read_csv("Customer Service_Requests_from_2010_to_Present.csv", low_memory = False)
df
```

Figure 1: importing the dataset

This Python code starts by importing several useful libraries for data analysis. pandas (as pd) is used to read, clean, and manipulate data in table format, while numpy (as np) helps perform numerical operations and handle arrays. matplotlib.pyplot (as plt) and seaborn (as sns) are imported for creating visualizations; matplotlib makes basic charts, and seaborn builds on it to create more attractive, informative graphs. From the scipy.stats module, the skew and kurtosis functions are imported to calculate statistical properties: skewness shows how data leans left or right, and kurtosis describes the sharpness or flatness of the data distribution. Finally, the code reads a CSV file named "Customer Service_Requests_from_2010_to_Present.csv" using pandas.read_csv() and stores it in a variable called df, which stands for DataFrame—a structured table of data. The low_memory=False option is included to avoid data type warnings when loading large or complex datasets.

```
C:\Users\Swift\AppData\Local\Temp\ipykernel_74996\3463998313.py:8: DtypeWarning: Columns (48,49) have mixed types. Specify dtype option on import or set
low_memory=False.
  df = pd.read_csv("Customer Service_Requests_from_2010_to_Present.csv",  )
```

Figure 2: Error of the dataset

This message is a warning from Python, not an error that stops your code. It's just letting you know there might be a small issue when reading your data file.It says that columns 48 and 49 in your spreadsheet (the CSV file) have mixed types of data. That means some rows in those columns have numbers, while others have text. This can confuse Python because it prefers one consistent type per column.The message also suggests a fix: you can tell Python in advance what type of data to expect in those columns (using dtype), or

simply add a setting called low_memory=False to make Python look through the whole file before deciding what types to use. You already did that in your earlier code, which is why the warning didn't appear there.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew, kurtosis
from scipy.stats import f_oneway
from scipy.stats import chi2_contingency
df = pd.read_csv("Customer Service_Requests_from_2010_to_Present.csv", low_memory = False)
df
```

Figure 3: correct code of the dataset

We have correct the code simply adding the low-memory = false after that we import more dataset in it. Those code shows to starts by bringing in a few powerful tools to help us understand and analyze a big dataset, like a spreadsheet full of customer service requests. It first loads pandas, which helps open and organize the data into a neat table. Then it loads numpy for doing math, and matplotlib.pyplot and seaborn for drawing charts and graphs that help us see patterns more clearly. After that, it brings in a couple of tools from another library called scipy.stats one group (skew and kurtosis) helps us understand the shape of the data whether it's lopsided or has sharp or flat peaks. The next two functions, f_oneway and chi2_contingency, are used to do statistical tests: f_oneway checks if the averages of several groups are different (ANOVA test), and chi2_contingency checks if two categories are related to each other (Chi-square test). Finally, it reads a file called "Customer Service Requests from 2010 to Present", and saves it into a variable called df, which is just a way to store the data in memory so we can explore, analyze, and visualize it.

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	...	Bridge Highway Name	Bridge Highway Direction	Road Ramp	
0	32310363	12/31/2015 11:59:45 PM	01-01-16 0:55	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	10034.0	71 VERMILYEA AVENUE	...	NaN	NaN	NaN	
1	32309934	12/31/2015 11:59:44 PM	01-01-16 1:26	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	11105.0	27-07 23 AVENUE	...	NaN	NaN	NaN	
2	32309159	12/31/2015 11:59:29 PM	01-01-16 4:51	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	10458.0	2897 VALENTINE AVENUE	...	NaN	NaN	NaN	
3	32305098	12/31/2015 11:57:46 PM	01-01-16 7:43	NYPD	New York City Police Department	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk	10461.0	2940 BAISLEY AVENUE	...	NaN	NaN	NaN	
4	32306529	12/31/2015 11:56:58 PM	01-01-16 3:24	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11373.0	87-14 57 ROAD	...	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
300693	30281872	03/29/2015 12:33:41 AM	NaN	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	NaN	CRESCENT AVENUE	...	NaN	NaN	NaN	
300694	30281230	03/29/2015 12:33:28 AM	03/29/2015 02:33:59 AM	NYPD	New York City Police Department	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	100-17 87 AVENUE	...	NaN	NaN	NaN	
300695	30283424	03/29/2015 12:33:03 AM	03/29/2015 03:40:20 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	162 THROOP AVENUE	...	NaN	NaN	NaN	
300696	30280004	03/29/2015 12:33:02 AM	03/29/2015 04:38:35 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10461.0	3151 EAST TREMONT AVENUE	...	NaN	NaN	NaN	

Figure 4:Output of the dataset

4	32306529	12/31/2015 11:56:58 PM	01-01-16 3:24	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11373.0	87-14 57 ROAD	...	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
300693	30281872	03/29/2015 12:33:41 AM	NaN	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	NaN	CRESCENT AVENUE	...	NaN	NaN	NaN	
300694	30281230	03/29/2015 12:33:28 AM	03/29/2015 02:33:59 AM	NYPD	New York City Police Department	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	100-17 87 AVENUE	...	NaN	NaN	NaN	
300695	30283424	03/29/2015 12:33:03 AM	03/29/2015 03:40:20 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	162 THROOP AVENUE	...	NaN	NaN	NaN	
300696	30280004	03/29/2015 12:33:02 AM	03/29/2015 04:38:35 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10461.0	3151 EAST TREMONT AVENUE	...	NaN	NaN	NaN	
300697	30281825	03/29/2015 12:33:01 AM	03/29/2015 04:41:50 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Store/Commercial	10036.0	251 WEST 48 STREET	...	NaN	NaN	NaN	

300698 rows × 53 columns

Figure 5: output of the dataset

This output shows a large dataset (about 300,698 rows and 53 columns) containing records of customer service requests in New York City from 2010 onward. Each row represents a single complaint or request made by someone, and each column contains details about that complaint like when it was created and closed, which agency handled it (usually the NYPD), what the complaint was about (e.g., noise, blocked driveway), where it happened (address, ZIP code, location type), and geographic info (latitude and longitude). Many of the less-used columns (like highway names or ferry details) have missing data (NaN), which means that info wasn't filled in for most complaints.

2.2:Provide your insight on the **information** and **details** that the provided dataset carries

```
df.describe
df
```

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	...	Bridge Highway Name	Bridge Highway Direction	Road Ramp	H S
0	32310363	12/31/2015 11:59:45 PM	01-01-16 0:55	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	10034.0	71 VERMILYEA AVENUE	...	NaN	NaN	NaN	
1	32309934	12/31/2015 11:59:44 PM	01-01-16 1:26	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	11105.0	27-07 23 AVENUE	...	NaN	NaN	NaN	
2	32309159	12/31/2015 11:59:29 PM	01-01-16 4:51	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	10458.0	2897 VALENTINE AVENUE	...	NaN	NaN	NaN	
3	32305098	12/31/2015 11:57:46 PM	01-01-16 7:43	NYPD	New York City Police Department	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk	10461.0	2940 BAISLEY AVENUE	...	NaN	NaN	NaN	
4	32306529	12/31/2015 11:56:58 PM	01-01-16 3:24	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11373.0	87-14 57 ROAD	...	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
300693	30281872	03/29/2015 12:33:41 AM	NaN	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	NaN	CRESCENT AVENUE	...	NaN	NaN	NaN	
300694	30281230	03/29/2015 12:33:28 AM	03/29/2015 02:33:59 AM	NYPD	New York City Police Department	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	100-17 87 AVENUE	...	NaN	NaN	NaN	
300695	30283424	03/29/2015 12:33:03 AM	03/29/2015 03:40:20 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	162 THROOP AVENUE	...	NaN	NaN	NaN	

Figure 6: information and details are described

This code loads a large table of NYC customer service requests using Pandas. It then uses df.describe() to show a quick summary of the numerical columns, like count, average, and range. This helps us understand the overall data. Writing just df.describe without brackets doesn't run the summary—it only points to the function.

```
df.columns
df
```

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	...	Bridge Highway Name	Bridge Highway Direction	Road Ramp
0	32310363	12/31/2015 11:59:45 PM	01-01-16 0:55	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	10034.0	71 VERMILYEA AVENUE	...	NaN	NaN	NaN
1	32309934	12/31/2015 11:59:44 PM	01-01-16 1:26	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	11105.0	27-07 23 AVENUE	...	NaN	NaN	NaN
2	32309159	12/31/2015 11:59:29 PM	01-01-16 4:51	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	10458.0	2897 VALENTINE AVENUE	...	NaN	NaN	NaN
3	32305098	12/31/2015 11:57:46 PM	01-01-16 7:43	NYPD	New York City Police Department	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk	10461.0	2940 BAISLEY AVENUE	...	NaN	NaN	NaN
4	32306529	12/31/2015 11:56:58 PM	01-01-16 3:24	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11373.0	87-14 57 ROAD	...	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
300693	30281872	03/29/2015 12:33:41 AM	NaN	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	NaN	CRESCENT AVENUE	...	NaN	NaN	NaN
300694	30281230	03/29/2015 12:33:28 AM	03/29/2015 02:33:59 AM	NYPD	New York City Police Department	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	100-17 87 AVENUE	...	NaN	NaN	NaN
300695	30283424	03/29/2015 12:33:03 AM	03/29/2015 03:40:20 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	162 THROOP AVENUE	...	NaN	NaN	NaN

Figure 7: information and details of the columns

In simple terms, df.columns gives you a list of all the column names in your dataset (the headers of each column in the table). It helps you see what kind of information each column holds, like "Complaint Type," "Agency," or "Location." It's like checking the labels at the top of each column in a spreadsheet.

```
df.head()
```

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	...	Bridge Highway Name	Bridge Highway Direction	Road Ramp	Bridge Highway Segment	Garag L Nam
0	32310363	12/31/2015 11:59:45 PM	01-01-16 0:55	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	10034.0	71 VERMILYEA AVENUE	...	NaN	NaN	NaN	NaN	Na
1	32309934	12/31/2015 11:59:44 PM	01-01-16 1:26	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	11105.0	27-07 23 AVENUE	...	NaN	NaN	NaN	NaN	Na
2	32309159	12/31/2015 11:59:29 PM	01-01-16 4:51	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	10458.0	2897 VALENTINE AVENUE	...	NaN	NaN	NaN	NaN	Na
3	32305098	12/31/2015 11:57:46 PM	01-01-16 7:43	NYPD	New York City Police Department	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk	10461.0	2940 BAISLEY AVENUE	...	NaN	NaN	NaN	NaN	Na
4	32306529	12/31/2015 11:56:58 PM	01-01-16 3:24	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11373.0	87-14 57 ROAD	...	NaN	NaN	NaN	NaN	Na

5 rows × 53 columns

Figure 8: information and details of the head

The code df.head() shows you the first five rows of your dataset. It's a quick way to preview the data and see what the first few entries look like. Think of it as flipping to the beginning of a spreadsheet to get an idea of what kind of information it contains.

```
df.sum
```

```
<bound method DataFrame.sum of         Unique Key            Created Date             Closed Date Agency  \
0         32310363  12/31/2015 11:59:45 PM           01-01-16 0:55   NYPD
1         32309934  12/31/2015 11:59:44 PM           01-01-16 1:26   NYPD
2         32309159  12/31/2015 11:59:29 PM           01-01-16 4:51   NYPD
3         32305098  12/31/2015 11:57:46 PM           01-01-16 7:43   NYPD
4         32306529  12/31/2015 11:56:58 PM           01-01-16 3:24   NYPD
...            ...                     ...                    ...    ...
300693    30281872  03/29/2015 12:33:41 AM                    NaN   NYPD
300694    30281230  03/29/2015 12:33:28 AM  03/29/2015 02:33:59 AM   NYPD
300695    30283424  03/29/2015 12:33:03 AM  03/29/2015 03:40:20 AM   NYPD
300696    30280004  03/29/2015 12:33:02 AM  03/29/2015 04:38:35 AM   NYPD
300697    30281825  03/29/2015 12:33:01 AM  03/29/2015 04:41:50 AM   NYPD

                           Agency Name           Complaint Type  \
0       New York City Police Department  Noise - Street/Sidewalk
1       New York City Police Department         Blocked Driveway
2       New York City Police Department         Blocked Driveway
3       New York City Police Department          Illegal Parking
4       New York City Police Department          Illegal Parking
...                                 ...                      ...
300693  New York City Police Department       Noise - Commercial
300694  New York City Police Department         Blocked Driveway
300695  New York City Police Department       Noise - Commercial
300696  New York City Police Department       Noise - Commercial
300697  New York City Police Department       Noise - Commercial

                         Descriptor        Location Type  Incident Zip  \
0                  Loud Music/Party      Street/Sidewalk       10034.0
1                         No Access      Street/Sidewalk       11105.0
2                         No Access      Street/Sidewalk       10458.0
3       Commercial Overnight Parking      Street/Sidewalk       10461.0
4                  Blocked Sidewalk      Street/Sidewalk       11373.0
...                             ...                  ...           ...
300693             Loud Music/Party  Club/Bar/Restaurant           NaN
300694               Partial Access      Street/Sidewalk       11418.0
300695             Loud Music/Party  Club/Bar/Restaurant       11206.0
```

Figure 9: information and details of the sum

The code df.sum() adds up all the numbers in each numeric column of your dataset. It gives you the total of each column, like adding up all the values in a column of prices or quantities. It’s like finding the total of all the numbers in each column of a table.

```
df.mean
```

```
<bound method DataFrame.mean of         Unique Key            Created Date            Closed Date Agency  \
0         32310363  12/31/2015 11:59:45 PM           01-01-16 0:55   NYPD
1         32309934  12/31/2015 11:59:44 PM           01-01-16 1:26   NYPD
2         32309159  12/31/2015 11:59:29 PM           01-01-16 4:51   NYPD
3         32305098  12/31/2015 11:57:46 PM           01-01-16 7:43   NYPD
4         32306529  12/31/2015 11:56:58 PM           01-01-16 3:24   NYPD
...            ...                     ...                    ...    ...
300693    30281872  03/29/2015 12:33:41 AM                    NaN   NYPD
300694    30281230  03/29/2015 12:33:28 AM  03/29/2015 02:33:59 AM   NYPD
300695    30283424  03/29/2015 12:33:03 AM  03/29/2015 03:40:20 AM   NYPD
300696    30280004  03/29/2015 12:33:02 AM  03/29/2015 04:38:35 AM   NYPD
300697    30281825  03/29/2015 12:33:01 AM  03/29/2015 04:41:50 AM   NYPD

                          Agency Name           Complaint Type  \
0       New York City Police Department  Noise - Street/Sidewalk
1       New York City Police Department         Blocked Driveway
2       New York City Police Department         Blocked Driveway
3       New York City Police Department          Illegal Parking
4       New York City Police Department          Illegal Parking
...                                ...                      ...
300693  New York City Police Department       Noise - Commercial
300694  New York City Police Department         Blocked Driveway
300695  New York City Police Department       Noise - Commercial
300696  New York City Police Department       Noise - Commercial
300697  New York City Police Department       Noise - Commercial

                       Descriptor        Location Type  Incident Zip  \
0                 Loud Music/Party      Street/Sidewalk       10034.0
1                        No Access      Street/Sidewalk       11105.0
2                        No Access      Street/Sidewalk       10458.0
3       Commercial Overnight Parking      Street/Sidewalk       10461.0
4                 Blocked Sidewalk      Street/Sidewalk       11373.0
...                            ...                  ...           ...
300693            Loud Music/Party  Club/Bar/Restaurant           NaN
300694              Partial Access      Street/Sidewalk       11418.0
300695            Loud Music/Party  Club/Bar/Restaurant       11206.0
300696            Loud Music/Party  Club/Bar/Restaurant       10461.0
```

The code df.mean() calculates the average of all the numbers in each numeric column of your dataset. It adds up the numbers in a column and then divides the total by how many numbers there are. It's like finding the average score of a class by adding up all the scores and dividing by the number of students.

2.3:Convert the columns "Created Date" and "Closed Date" to datetime datatype and create a new column "Request_Closing_Time" as the time elapsed between request creation and request closing

```
C:\Users\Swift\AppData\Local\Temp\ipykernel_74996\4252485798.py:1: UserWarning: Could not infer format, so each element will be parsed individually, fall
ing back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  df['Created Date'] = pd.to_datetime(df['Created Date'])
C:\Users\Swift\AppData\Local\Temp\ipykernel_74996\4252485798.py:2: UserWarning: Could not infer format, so each element will be parsed individually, fall
ing back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  df['Closed Date'] = pd.to_datetime(df['Closed Date'])
```

Figure 10:error datatype of datetime

The error message you're seeing means that Python (using pandas) is trying to change text dates in the “Created Date” and “Closed Date” columns into real date formats. But it's confused because the format of the dates isn’t clearly given. So, it guesses the format, which might slow things down or cause mistakes. To fix this, you should tell pandas the exact format of your dates, like '%Y-%m-%d' (which means year-month-day). This makes the date conversion faster and more accurate.

```
df['Created Date'] = pd.to_datetime(df['Created Date'])
df['Closed Date'] = pd.to_datetime(df['Closed Date'])
df['Request_Closing_Time'] = df['Closed Date'] - df['Created Date']
print(df[['Created Date', 'Closed Date','Request_Closing_Time']])
```

Figure 11: correct datatype of datetime

This code takes two columns called "Created Date" and "Closed Date" from a table (DataFrame) and changes them from plain text into actual date format so the computer can understand them. Then, it calculates how long each request took to complete by subtracting the created date from the closed date, and stores that result in a new column called "Request_Closing_Time." Finally, it shows a small table with the created date, closed date, and the time taken to close each request, so we can easily see how long each one took.

	Created Date	Closed Date	Request_Closing_Time
0	2015-12-31 23:59:45	2016-01-01 00:55:00	0 days 00:55:15
1	2015-12-31 23:59:44	2016-01-01 01:26:00	0 days 01:26:16
2	2015-12-31 23:59:29	2016-01-01 04:51:00	0 days 04:51:31
3	2015-12-31 23:57:46	2016-01-01 07:43:00	0 days 07:45:14
4	2015-12-31 23:56:58	2016-01-01 03:24:00	0 days 03:27:02
...	...	...	...
300693	2015-03-29 00:33:41	NaT	NaT
300694	2015-03-29 00:33:28	2015-03-29 02:33:59	0 days 02:00:31
300695	2015-03-29 00:33:03	2015-03-29 03:40:20	0 days 03:07:17
300696	2015-03-29 00:33:02	2015-03-29 04:38:35	0 days 04:05:33
300697	2015-03-29 00:33:01	2015-03-29 04:41:50	0 days 04:08:49

[300698 rows x 3 columns]

Figure 12: answer of the datatype of datetime

This table shows how long it took to complete service requests. The "Created Date" column tells when each request was made, and the "Closed Date" shows when it was completed. The "Request_Closing_Time" column calculates the time difference between the two, showing how long it took to close the request. For example, one request made on December 31, 2015, at 11:59 PM was closed on January 1, 2016, at 12:55 AM, meaning it took 55 minutes and 15 seconds to complete. If a request hasn’t been closed yet, the "Closed Date" and "Request_Closing_Time" will show "NaT" (Not a Time), which means the data is missing. The full dataset contains 300,698 rows, each showing a request’s creation time, closing time, and how long it took to resolve.

2.4:Write a python program to drop irrelevant Columns which are listed below.

```
columns = ['Agency Name','Incident Address','Street Name','Cross Street 1','Cross Street 2','Intersection Street 1', 'Intersection Street 2','Address Typ
'School State','School Zip','School Not Found','School or Citywide Complaint','Vehicle Type', 'Taxi Company Borough','Taxi Pick Up location','Bridge High
'Road Ramp','Bridge Highway Segment','Garage Lot Name','Ferry Direction','Ferry Terminal Name','Landmark',
'X Coordinate (State Plane)','Y Coordinate (State Plane)','Due Date','Resolution Action Updated Date','Community Board','Facility Type',
'Location']
df = df.drop(columns=columns, errors='ignore')
df
```

Figure 13:drop irrelevant Columns

This code removes a list of columns from the dataset to make it simpler and easier to work with. The listed columns, such as school details, park names, taxi info, and location coordinates, may not be needed for the current analysis. The errors='ignore' part ensures that if any of these columns don't exist in the data, Python won't show an error—it will just skip them. This helps clean up the data by keeping only the important information.

	Unique Key	Created Date	Closed Date	Agency	Complaint Type	Descriptor	Location Type	Incident Zip	City	Status	Resolution Description	Borough	Taxi Pick Up Location	Latitu
0	32310363	2015-12-31 23:59:45	2016-01-01 00:55:00	NYPD	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	10034.0	NEW YORK	Closed	The Police Department responded and upon arriv...	MANHATTAN	NaN	40.865
1	32309934	2015-12-31 23:59:44	2016-01-01 01:26:00	NYPD	Blocked Driveway	No Access	Street/Sidewalk	11105.0	ASTORIA	Closed	The Police Department responded to the complai...	QUEENS	NaN	40.775
2	32309159	2015-12-31 23:59:29	2016-01-01 04:51:00	NYPD	Blocked Driveway	No Access	Street/Sidewalk	10458.0	BRONX	Closed	The Police Department responded and upon arriv...	BRONX	NaN	40.870
3	32305098	2015-12-31 23:57:46	2016-01-01 07:43:00	NYPD	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk	10461.0	BRONX	Closed	The Police Department responded to the complai...	BRONX	NaN	40.835
4	32306529	2015-12-31 23:56:58	2016-01-01 03:24:00	NYPD	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11373.0	ELMHURST	Closed	The Police Department responded and upon arriv...	QUEENS	NaN	40.733

Figure 14: output of the drop irrelevant columns

This table shows a list of 311 service complaints reported in New York City. Each row represents one complaint, including details such as the unique ID number, the date and time the complaint was created and closed, the agency involved (mostly NYPD), the type of issue (like noise or illegal parking), a more detailed description, the location type (e.g., street or club), zip code, city, and borough. It also shows the status of the complaint (whether it's still open or has been closed), a summary of how it was resolved, and the exact location (latitude and longitude). Finally, there's a column called

Request_Closing_Time, which tells us how long it took to handle the complaint. Some entries may have missing values, like no closing date or location, meaning the complaint is still open or the data wasn't available.

2.5:Write a python program to remove the NaN missing values from updated dataframe.

Before

4	32306529	12-31 23:56:58	01-01 03:24:00	NYPD	Illegal Parking	Sidewalk	Street/Sidewalk	11373.0	ELMHURST	Closed	responded and upon arriv...	QUEENS	NaN	40.733
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
300693	30281872	2015-03-29 00:33:41	NaT	NYPD	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	NaN	NaN	Open	Your complaint has been forwarded to the New Y...	Unspecified	NaN	N
300694	30281230	2015-03-29 00:33:28	2015-03-29 02:33:59	NYPD	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	RICHMOND HILL	Closed	The Police Department responded and upon arriv...	QUEENS	NaN	40.694
300695	30283424	2015-03-29 00:33:03	2015-03-29 03:40:20	NYPD	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	BROOKLYN	Closed	The Police Department responded to the complai...	BROOKLYN	NaN	40.699
300696	30280004	2015-03-29 00:33:02	2015-03-29 04:38:35	NYPD	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10461.0	BRONX	Closed	The Police Department responded to the complai...	BRONX	NaN	40.837
300697	30281825	2015-03-29 00:33:01	2015-03-29 04:41:50	NYPD	Noise - Commercial	Loud Music/Party	Store/Commercial	10036.0	NEW YORK	Closed	The Police Department responded to the complai...	MANHATTAN	NaN	40.760

300698 rows × 16 columns

Figure 15: before removing the NAN value

After

Figure 16: After removing the NAN Value

The command df.dropna() in Python using pandas is used to clean a dataset by removing all rows that contain any missing or empty values. This means if even one cell in a row is blank (such as a missing date, location, or any other information), that entire row will be deleted from the DataFrame. This helps ensure that the data you are working with is complete and accurate, making it easier to analyze without errors caused by missing information.

2.6:Write a python program to see the unique values from all the columns in the dataframe.

```
columns = df.columns
df= {col:df[col].unique() for col in columns if col in df.columns}
df
```

```
{'Unique Key': array([32310363, 32309934, 32309159, ..., 30283424, 30280004, 30281825],
       dtype=int64),
 'Created Date': <DatetimeArray>
 ['2015-12-31 23:59:45', '2015-12-31 23:59:44', '2015-12-31 23:59:29',
  '2015-12-31 23:57:46', '2015-12-31 23:56:58', '2015-12-31 23:56:30',
  '2015-12-31 23:55:32', '2015-12-31 23:54:05', '2015-12-31 23:53:58',
  '2015-12-31 23:52:58',
  ...
  '2015-03-29 00:37:15', '2015-03-29 00:35:28', '2015-03-29 00:35:23',
  '2015-03-29 00:35:04', '2015-03-29 00:34:32', '2015-03-29 00:33:41',
  '2015-03-29 00:33:28', '2015-03-29 00:33:03', '2015-03-29 00:33:02',
  '2015-03-29 00:33:01']
 Length: 259493, dtype: datetime64[ns],
 'Closed Date': <DatetimeArray>
 ['2016-01-01 00:55:00', '2016-01-01 01:26:00', '2016-01-01 04:51:00',
  '2016-01-01 07:43:00', '2016-01-01 03:24:00', '2016-01-01 01:50:00',
  '2016-01-01 01:53:00', '2016-01-01 01:42:00', '2016-01-01 08:27:00',
  '2016-01-01 01:17:00',
```

Figure 17: unique values from all the columns in the dataframe

At First, columns = df.columns gets the list of all column names from the DataFrame df. Then, the second line creates a new dictionary where each key is a column name, and each value is a list of unique values found in that column. It checks each column in the DataFrame and collects only the different (non-repeated) values. In the end, the df variable no longer holds the original data — it now stores a dictionary that summarizes the unique data in each column.

3.Data analysis

3.1:Write a Python program to show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame.

- The sum is the simply the total of all the individual values in a dataset.
- The mean, also known as the average, is calculated by summing all values in the dataset and dividing by the number of values.
- Standard deviation quantifies the amount of variation in a dataset. A low standard deviation indicates data points are close to the mean, while a high standard deviation suggests data points are more spread out.
- Skewness is a measure used in statistics to understand a data set's symmetry or lack thereof. It helps determine whether the data is more spread out on one side of the mean than the other (Anon., 2025).
- Kurtosis is a statistical measure used to describe a characteristic of a dataset (Willkenton, 2021).

```
df = pd.read_csv("Customer Service_Requests_from_2010_to_Present.csv", low_memory= False)
numeric_df =df.select_dtypes(include='number')

sum_stat = pd.DataFrame({
    'sum' : numeric_df.sum(),
    'Mean' : numeric_df.mean(),
    'Standard Deviation' : numeric_df.std(),
    'Skewness' : numeric_df.skew(),
    'Kurtosis' : numeric_df.kurtosis()
})
sum_df = pd.DataFrame(sum_stat).T
print(sum_df)
```

```
                     Unique Key  Incident Zip  X Coordinate (State Plane)  \
sum                9.412008e+12  3.233869e+09                2.986003e+11
Mean               3.130054e+07  1.084889e+04                1.004854e+06
Standard Deviation  5.738547e+05  5.831821e+02                2.175338e+04
Skewness           2.028266e-02 -2.448212e+00               -2.939656e-01
Kurtosis           -1.169387e+00  3.599208e+01                1.454476e+00

                     Y Coordinate (State Plane)  School or Citywide Complaint  \
sum                                6.054729e+10                           0.0
Mean                               2.037545e+05                           NaN
Standard Deviation                 2.988018e+04                           NaN
Skewness                           1.167695e-01                           NaN
Kurtosis                          -7.192361e-01                           NaN

                     Vehicle Type  Taxi Company Borough  Taxi Pick Up Location  \
sum                           0.0                   0.0                    0.0
Mean                          NaN                   NaN                    NaN
Standard Deviation            NaN                   NaN                    NaN
Skewness                      NaN                   NaN                    NaN
Kurtosis                      NaN                   NaN                    NaN

                     Garage Lot Name      Latitude     Longitude
sum                              0.0  1.210202e+07 -2.196759e+07
Mean                             NaN  4.072588e+01 -7.392563e+01
Standard Deviation               NaN  8.201242e-02  7.845442e-02
```

Figure 18: showing the summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame.

In this first of all, the code has been first loaded a CSV file containing customer service requests into a pandas DataFrame, with the low_memory=False option to ensure efficient

memory usage when dealing with large files. It then selects only the numeric columns from the DataFrame, as it focuses on analyzing numerical data. The code calculates five key summary statistics for each numeric column: the sum, mean (average), standard deviation (measure of spread), skewness (asymmetry of data), and kurtosis (the peakedness of the data distribution). These statistics are stored in a new DataFrame. After that, the summary statistics are transposed, so that the statistics appear as rows and the numeric columns are displayed as columns, making the data easier to interpret. Finally, the transposed summary statistics table is printed, providing an overview of the distribution and characteristics of the numeric data in the dataset.

The output provides summary statistics for various numeric columns. For the Unique Key, the sum is very large, with a mean of 31 million and a high standard deviation, indicating a wide range of values. The Incident Zip column shows a mean around 10,848 with low variation. X and Y Coordinates have large sums with moderate variation, indicating they represent geographic data. The School or Citywide Complaint column has no meaningful data, as it shows zeros for all statistics. Columns like Vehicle Type and Taxi Pick Up Location also have missing data or zeros. Latitude and Longitude show stable values with slight variations, indicating geographic locations.

3.2:Write a Python program to calculate and show correlation of all variables.

A statistical tool that helps in the study of the relationship between two variables is known as Correlation. It also helps in understanding the economic behaviour of the variables (Anon., 2024).

```
corr = df.corr(numeric_only = True)
sns.heatmap(corr, cmap='cool', fmt=".2f", linewidth=0.5)
corr
plt.show()
```

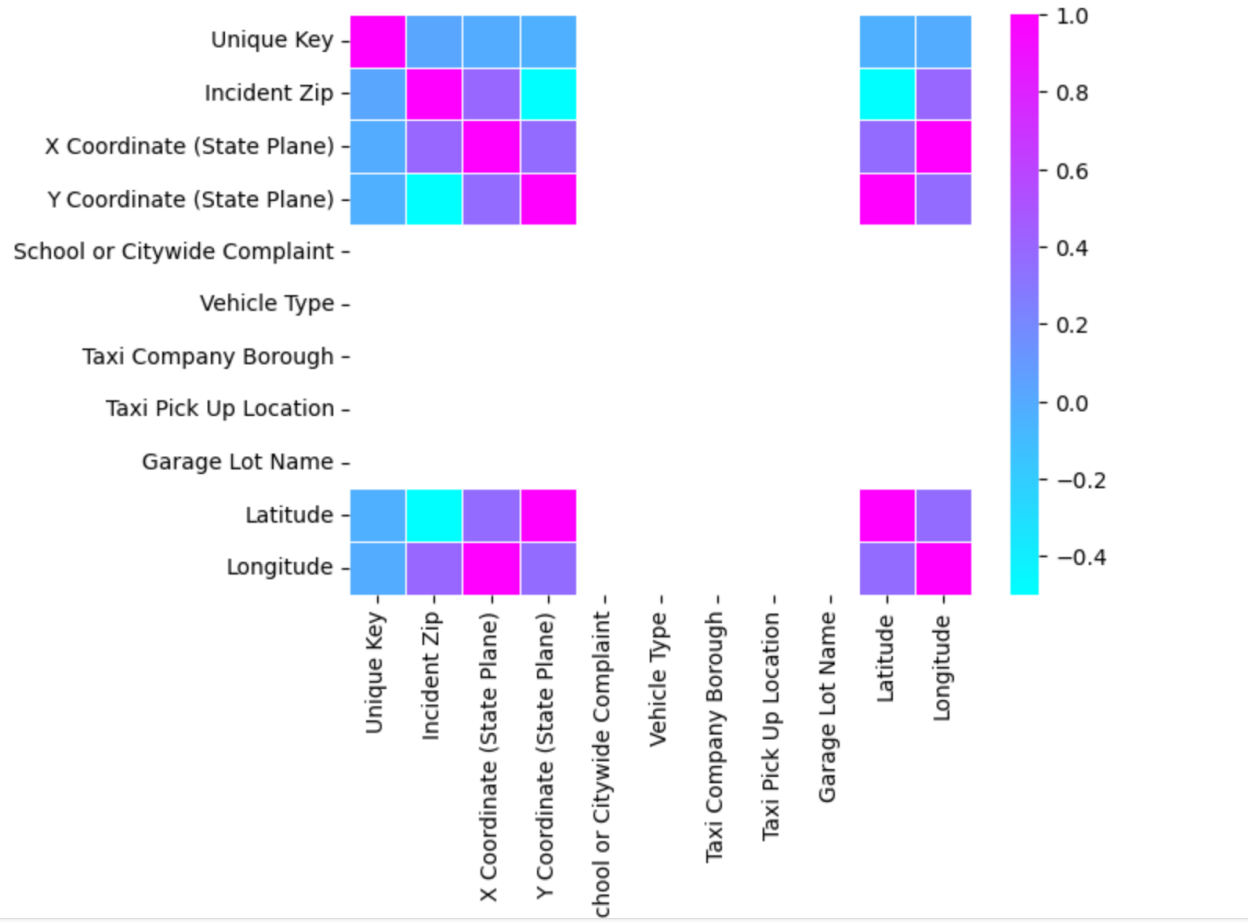

Figure 19: calculating and showing all the variables

In this program to show the correlation of all varibales ,at first this code calculates and visualizes the correlation between numeric columns in the DataFrame. First, it calculates the correlation values for all numeric columns using df.corr(). Then, it creates a heatmap with sns.heatmap() to display these correlations, using a cool color scheme and formatting the correlation values to two decimal places. The heatmap helps us easily see

how strongly different numeric variables are related to each other, with colors indicating the strength and direction of the relationships. Finally, plt.show() displays the heatmap on the screen.

The output of the correlation of all variables comes in a heatmap that helps to show how different columns in the dataset are related to each other. Each box on the heatmap represents a pair of columns, and the color tells you how strongly they're connected. If the box is dark blue, it means the two columns are strongly linked in a positive way—when one goes up, the other tends to go up as well. If the box is red or light, it means they’re negatively linked—when one increases, the other decreases. If the color is neutral or light, it means there’s not much connection between them. The exact number showing the strength of the connection appears inside the boxes. This makes it easier to spot which columns have strong or weak relationships with each other.

4.Data exploration

Data exploration is the first step in the journey of extracting insights from raw datasets. Data exploration serves as the compass that guides data scientists through the vast sea of information. It involves getting to know the data intimately, understanding its structure, and uncovering valuable nuggets that lay hidden beneath the surface.

4.1.Provide four major insights through visualization that you come up after data mining.

4.1.1Complaint Type

```
df['Complaint Type'].value_counts().plot.bar()
plt.show()
```

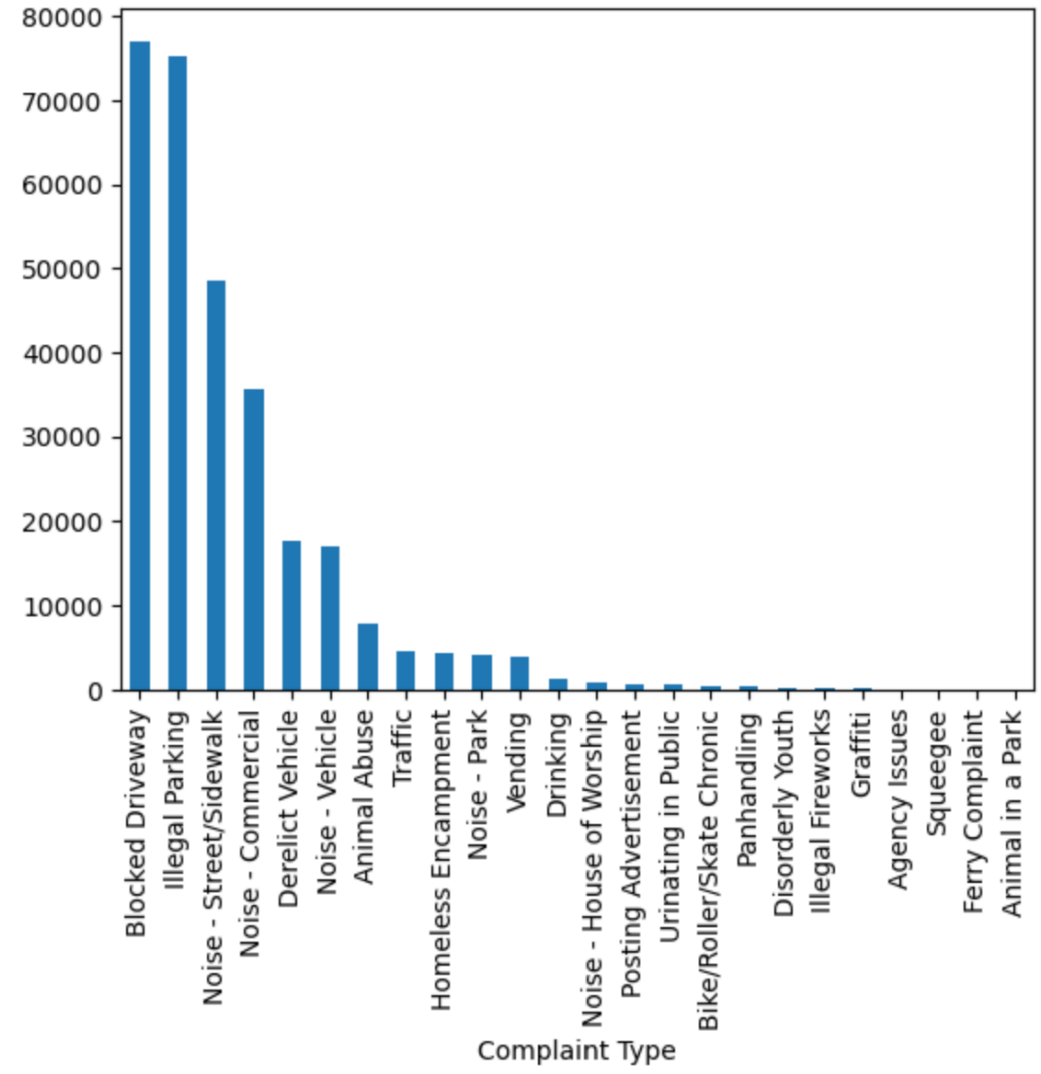

Figure 20: complaint Type in bar graph

This code counts how many times each type of complaint appears in the 'Complaint Type' column of the dataset and then creates a bar chart to visualize the distribution. Each bar represents a different type of complaint, and the height of the bar shows how many times that particular complaint type occurs in the data. The plt.show() command displays the bar chart, which helps in understanding the most common complaint types by looking at which bars are the tallest. The output of this code is a bar chart where each bar represents a different complaint type, and the height of each bar shows how many times that particular complaint type has been recorded in the dataset. The taller the bar, the more frequent that complaint type is. This helps to quickly identify which complaints are most common and which ones are less frequent, providing a clear visual representation of the distribution of complaint types in the data.

4.1.2 Borough

```
df['Borough'].value_counts().plot.pie()
plt.title('complaint by Borough')
plt.show()
```

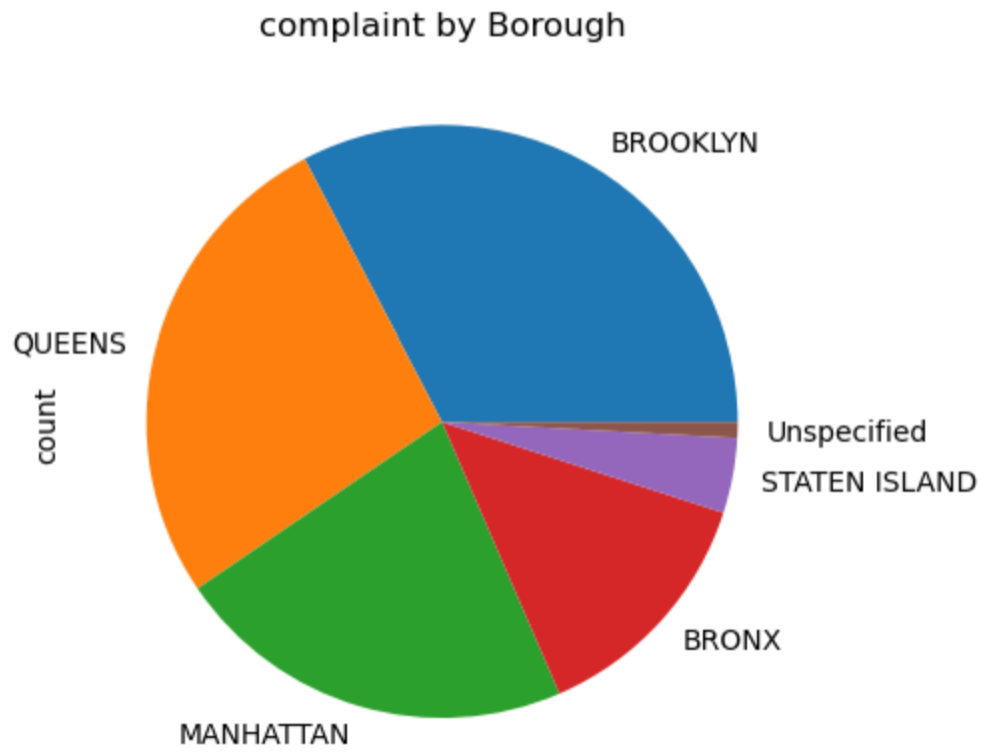

Figure 21: Borough in pie chart form

This code creates a pie chart that shows the distribution of complaints across different boroughs. It uses the "Borough" column from the dataset, counts how many complaints are recorded for each borough, and then visualizes this data as a pie chart. Each slice of the pie represents a borough, and the size of each slice shows how many complaints come from that particular borough. The title "complaint by Borough" is added to the chart for clarity. This pie chart helps to understand the proportion of complaints from each borough in a visual way.

4.1.3:Average Complaint by closing time

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_79404\289428857.py in ?()
----> 1 avg_time = df.grouby('Complaint Type')['Request_Closing_Time'].mean()
      2 avg.hrs = avg_time.dt.total_seconds()/3600
      3 avg_hrs.plot.bar()
      4 plt.title('Average Complaint by Closing Time')

~\anaconda3\Lib\site-packages\pandas\core\generic.py in ?(self, name)
   6295             and name not in self._accessors
   6296             and self._info_axis._can_hold_identifiers_and_holds_name(name)
   6297         ):
   6298             return self[name]
-> 6299         return object.__getattribute__(self, name)

AttributeError: 'DataFrame' object has no attribute 'grouby'
```

Figure 22: error code of the average complaint by closing time

The error message you're seeing is an AttributeError, which means the code is trying to use a method or function that doesn't exist for the object it's being applied to. In this case, the problem happens on this line of code: avg_time = df.groupby('Complaint Type')['Request_Closing_Time'].mean(). The error suggests that there's a typo or mistake in the code, where groupby is misspelled as 'grouby' (missing the letter "p"). The method groupby is used to group the data by a particular column, but because of the typo, the code doesn't recognize it as a valid method, which leads to the error. Correcting the spelling of 'groupby' will fix the issue.

```
df['Created Date'] = pd.to_datetime(df['Created Date'], errors='coerce')
df['Closed Date'] = pd.to_datetime(df['Closed Date'], errors='coerce')


df['Request Closing Time'] = df['Closed Date'] - df['Created Date']


avg_time = df.groupby('Complaint Type')['Request Closing Time'].mean()


avg_hrs = avg_time.dt.total_seconds() / 3600


avg_hrs.plot.bar(figsize=(10,5))
plt.title('Average Complaint by Closing Time')
plt.ylabel('Average Closing Time (Hours)')
plt.xlabel('Complaint Type')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

Figure 23: correct code of the average complaint by closing time

After correcting the ,this code processes complaint data by converting the 'Created Date' and 'Closed Date' columns to a standard date-time format and calculates the time taken to resolve each complaint. It then groups the data by 'Complaint Type' and calculates the average closing time for each type. The average times are converted into hours, and a bar chart is generated to display these average times for each complaint type. The chart provides a quick visual comparison of how long it takes to resolve different complaint types.

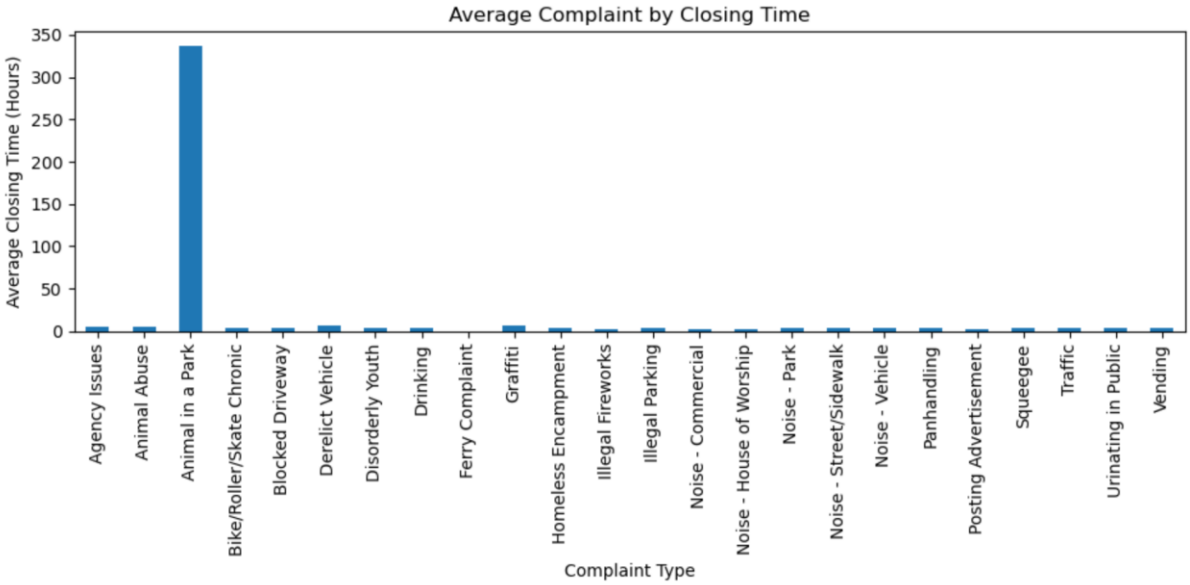

Figure 24: output of the average complaint by closing time

The output is a bar chart that shows the average time it takes to resolve different types of complaints. Each bar represents a specific complaint type, and the height of the bar shows the average time (in hours) it takes to close those complaints. The taller the bar, the longer it typically takes to resolve that type of complaint. The chart allows us to quickly see which complaint types take the most or least time to resolve, making it easy to identify areas where improvements might be needed in response times.

4.1.4:Monthly complaint Value

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[73], line 1
----> 1 df['Month'] = df['Created Date'].dt.to_period('M')
      2 monthly_complaint = df.groupby('Month')['Complaint Type'].count()
      3 monthly.compliant.plot()

File ~\anaconda3\Lib\site-packages\pandas\core\generic.py:6299, in NDFrame.__getattr__(self,
name)
   6292 if (
   6293     name not in self._internal_names_set
   6294     and name not in self._metadata
   6295     and name not in self._accessors
   6296     and self._info_axis._can_hold_identifiers_and_holds_name(name)
   6297 ):
   6298     return self[name]
-> 6299 return object.__getattribute__(self, name)

File ~\anaconda3\Lib\site-packages\pandas\core\accessor.py:224, in CachedAccessor.__get__(se
lf, obj, cls)
    221 if obj is None:
    222     # we're accessing the attribute of the class, i.e., Dataset.geo
    223     return self._accessor
--> 224 accessor_obj = self._accessor(obj)
    225 # Replace the property with the accessor object. Inspired by:
    226 # https://www.pydanny.com/cached-property.html
    227 # We need to use object.__setattr__ because we overwrite __setattr__ on
    228 # NDFrame
    229 object.__setattr__(obj, self._name, accessor_obj)

File ~\anaconda3\Lib\site-packages\pandas\core\indexes\accessors.py:643, in CombinedDatetime
likeProperties.__new__(cls, data)
    640 elif isinstance(data.dtype, PeriodDtype):
    641     return PeriodProperties(data, orig)
--> 643 raise AttributeError("Can only use .dt accessor with datetimelike values")

AttributeError: Can only use .dt accessor with datetimelike values
```

Figure 25: Error of the monthly complaint volume

The error message shows that there's a problem when trying to convert the 'Created Date' column into a monthly period using the .dt.to_period('M') method. This method only works with datetime values, but the 'Created Date' column isn't in the correct format, which causes the error. The traceback indicates that the issue happens within Pandas, as it's trying to access the .dt accessor, which is only available for datetime data. To fix this, you need to make sure that the 'Created Date' column is properly converted into datetime format before using .dt methods.

```
df['Created Date'] = pd.to_datetime(df['Created Date'], errors='coerce')

df['Month'] = df['Created Date'].dt.to_period('M')

monthly_complaint = df.groupby('Month')['Complaint Type'].count()

monthly_complaint.plot(kind='line', marker='o')
plt.title('Monthly Complaint Volume')
plt.ylabel('Number of Complaints')
plt.xlabel('Month')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

Figure 26:correct code of the monthly complaint volume

After correcting the error ,this code first converts the 'Created Date' column into a proper datetime format and extracts the month from it. It then counts how many complaints were made each month and stores this in a new variable. Finally, it creates a line chart showing the number of complaints per month, helping to visualize trends over time in a clear and organized way.

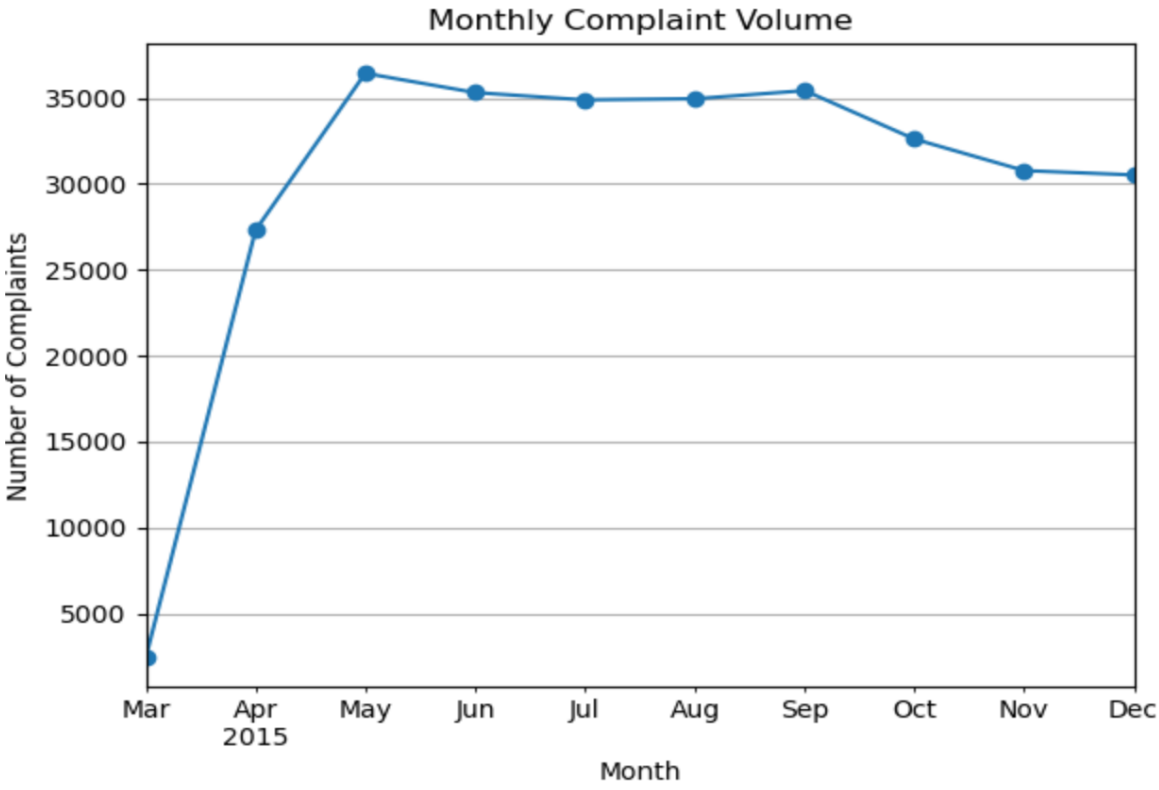

Figure 27: Output of the monthly complaint Volume

The output of this code is a line chart that shows how the number of complaints changes from month to month. Each point on the line represents a specific month, and the height of the point shows how many complaints were made in that month. If the line goes up, it means more complaints were made; if it goes down, it means fewer complaints. This chart helps you easily see which months had the most or least complaints and spot any trends over time.

4.2:Arrange the complaint types according to their average 'Request_Closing_Time', categorized by various locations. Illustrate it through graph as well.

```
df['Created Date'] = pd.to_datetime(df['Created Date'], format='%m/%d/%Y %I:%M:%S %p', errors='coerce')
df['Closed Date'] = pd.to_datetime(df['Closed Date'], format='%m/%d/%Y %I:%M:%S %p', errors='coerce')

df_clean = df.dropna(subset=['Created Date', 'Closed Date']).copy()

df_clean['Request_Closing_Time'] = (df_clean['Closed Date'] - df_clean['Created Date']).dt.total_seconds() / 3600

avg_time = df_clean.groupby(['Complaint Type', 'Borough'])['Request_Closing_Time'].mean()

pivot_table = avg_time.unstack()


plt.figure(figsize=(12, 10))
sns.heatmap(pivot_table, cmap='coolwarm', fmt=".2f", linewidths=0.5, annot=True)
plt.title('Average Request Closing Time (Hours) by Complaint Type and Borough', fontsize=14)
plt.xlabel('Borough')
plt.ylabel('Complaint Type')
plt.tight_layout()
plt.show()
```

Figure 28:according to their average 'Request_Closing_Time'

This code converts the 'Created Date' and 'Closed Date' columns into datetime format, cleans the data by removing rows with missing dates, and calculates the time taken to close a request in hours. It then groups the data by 'Complaint Type' and 'Borough' to calculate the average closing time for each group. The data is transformed into a pivot table, which is visualized using a heatmap to show the average request closing times. The heatmap uses color gradients to indicate shorter or longer closing times, with values annotated in each cell for clarity. This analysis helps to understand how complaint types and boroughs affect the time it takes to close requests.

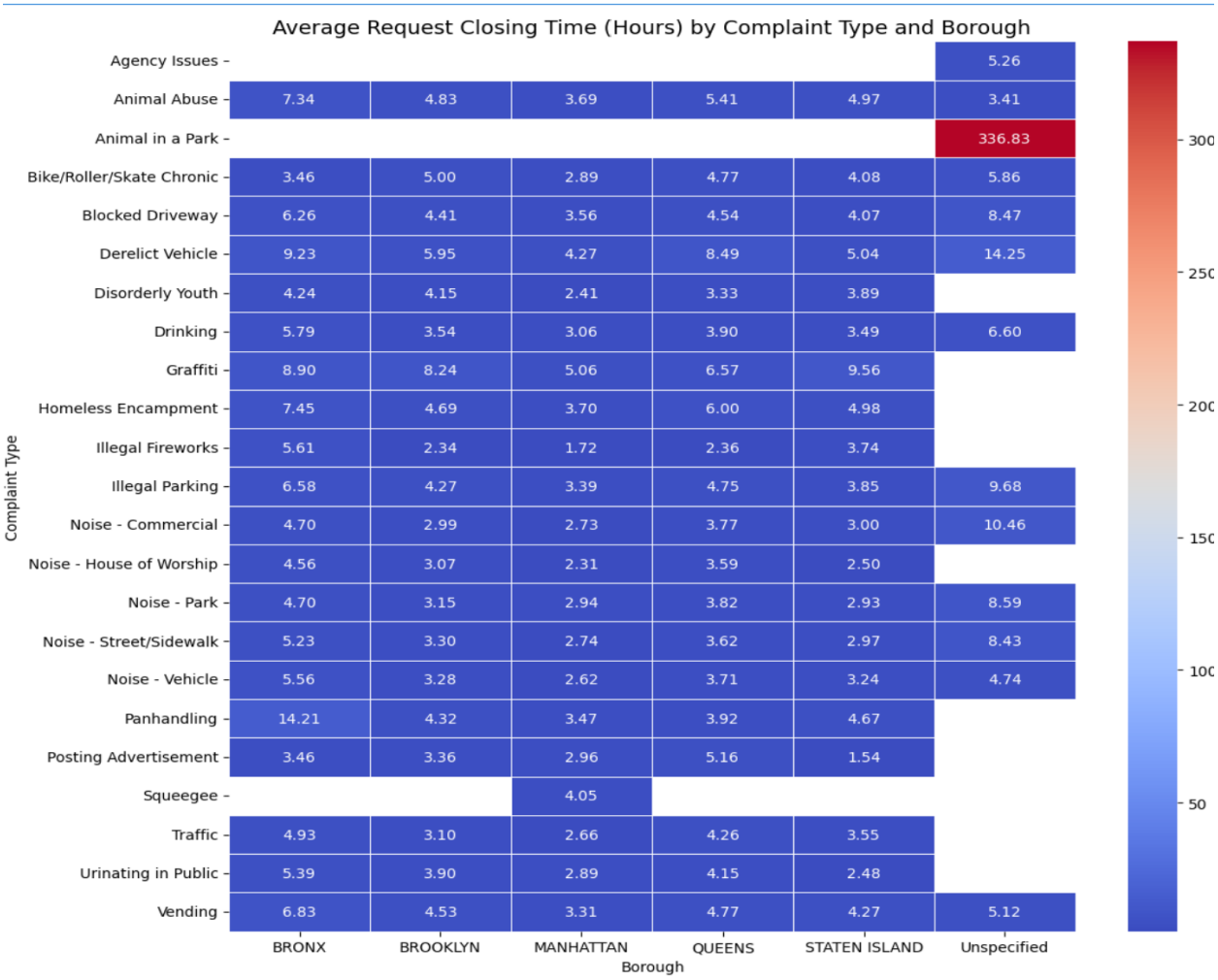

Figure 29:output of the according to their average 'Request_Closing_Time'

The output of the code is a heatmap that visually represents how long, on average, it takes to resolve different types of complaints across various boroughs in New York City. Each row in the heatmap corresponds to a specific complaint type, while each column represents a borough. The colored cells display the average request closing time in hours, calculated based on the time difference between the complaint's creation and closure. Warmer colors (like red) indicate longer response times, whereas cooler colors (like blue) signify faster responses. This visual makes it easy to identify which complaint types or boroughs have slower or faster average response times, helping in performance assessment and decision-making.

5.Statistical Testing

In statistics, statistical testing means that the result that was produced has a reason behind it, it was not produced randomly, or by chance (Anon., 2025).

5.1:Test 1: Whether the average response time across complaint types is similar or not.

5.1.1:State the Null Hypothesis (H0) and Alternate Hypothesis (H1)

When doing statistical testing, we begin with the null hypothesis (H_0), which assumes there is no effect, change, or difference — it's our default assumption. For example, if comparing response times for complaint types, H_0 assumes all have the same average time. The alternative hypothesis (H_1) is what we aim to support — it suggests there is a real difference or effect. A statistical test helps us decide whether the data provides strong enough evidence to reject H_0 and accept H_1. If the p-value is small (typically below 0.05), we reject H_0 and conclude that the observed differences are likely real, not due to random chance (Anon., 2025).

```
#test1
df['Created Date'] = pd.to_datetime(df['Created Date'], errors='coerce')
df['Closed Date'] = pd.to_datetime(df['Closed Date'], errors='coerce')
df_clean = df.dropna(subset=['Created Date', 'Closed Date']).copy()
df_clean['Request_Closing_Time'] = (df_clean['Closed Date'] - df_clean['Created Date']).dt.total_seconds() / 3600
top_complaints = df_clean['Complaint Type'].value_counts().nlargest(5).index
filtered_df = df_clean[df_clean['Complaint Type'].isin(top_complaints)]
groups = [group['Request_Closing_Time'].dropna() for name, group in filtered_df.groupby('Complaint Type')]
t_stat, p_value = stats.f_oneway(*groups)
print("F-statistic:", t_stat)
print("P-value:", p_value)
print("Average Response Time (Hours):", filtered_df['Request_Closing_Time'].mean())
if p_value < 0.05:
    print("Reject the null hypothesis: Average response time differs significantly across complaint types.")
else:
    print("Fail to reject the null hypothesis: No significant difference in average response time across complaint types.")
```

Figure 30:: Whether the average response time across complaint types is similar or not

This code checks if different types of complaints take different amounts of time to resolve. It first cleans the date columns and calculates how long each complaint took in hours. Then, it focuses on the top 5 most common complaint types and compares their average response times using an ANOVA test. If the p-value is below 0.05, it means the average response times are significantly different, so the null hypothesis is rejected. If the p-value is higher, we conclude that response times are similar across complaint types.

5.1.2:Perform the statistical test and provide the p-value.

A p-value, or probability value, is a number describing the likelihood of obtaining the observed data under the null hypothesis of a statistical test.

To test the hypotheses, we use an ANOVA test, which compares the average response times across different complaint types. This test helps us determine if there are significant differences in response times. After performing the test, we get a p-value, which tells us how likely it is that the observed differences happened by chance. If the p-value is smaller than 0.05, we reject the null hypothesis, meaning there are differences in response times. If it's larger than 0.05, we fail to reject the null hypothesis, meaning there are no significant differences.

```
F-statistic: 1799.600524153762
P-value: 0.0
Average Response Time (Hours): 4.383335717403647
Reject the null hypothesis: Average response time differs significantly across complaint types.
```

Figure 31: output of the : Whether the average response time across complaint types is similar or not

The test results show a very high F-statistic and a p-value of 0.0, which is much smaller than 0.05. This means there is strong evidence that the average time it takes to respond to complaints is not the same for all types. On average, it takes about 4.38 hours to close a complaint, but the time varies significantly depending on the complaint type. So, we reject the null hypothesis and conclude that the response time is different across different types of complaints.

5.1.3:Interpret the results to accept or reject the Null Hypothesis.

If the p-value from the statistical test is less than 0.05, it suggests that there is strong evidence showing that the average response times for different complaint types are significantly different. In this case, we reject the null hypothesis (H_0) and accept the alternative hypothesis (H_1), which says that the response times are not the same across complaint types. However, if the p-value is greater than 0.05, it means there isn't enough evidence to prove that the average response times differ. In this case, we fail to reject the null hypothesis (H_0), meaning that the average response times across the complaint types are statistically similar.

5.2:Test 2: Whether the type of complaint or service requested and location are related.

5.2.1:State the Null Hypothesis (H0) and Alternate Hypothesis (H1).

To examine whether the type of complaint or service request is related to the location where it was made, we begin with hypothesis testing. The Null Hypothesis (H_0) states that there is no relationship between complaint type and location – meaning complaints are spread randomly across areas without any pattern. The Alternate Hypothesis (H_1) claims that there is a relationship, suggesting that certain complaint types may be more common in specific locations. Hypothesis testing helps us use real data to decide whether to support or reject these assumptions (Field.A, 2013).

```
data = {
    'Complaint_Type': ['Noise', 'Garbage', 'Water', 'Noise', 'Water', 'Garbage', 'Noise', 'Water', 'Noise', 'Garbage'],
    'Location': ['North', 'South', 'North', 'East', 'South', 'West', 'East', 'West', 'North', 'South']
}


df = pd.DataFrame(data)


contingency_table = pd.crosstab(df['Complaint_Type'], df['Location'])


chi2, p_value, dof, expected = chi2_contingency(contingency_table)


print("Chi-Square Statistic:", chi2)
print("Degrees of Freedom:", dof)
print("P-Value:", p_value)
print("\nExpected Frequencies Table:\n", expected)


if p_value < 0.05:
    print("\nResult: Reject the null hypothesis. There is a relationship between complaint type and location.")
else:
    print("\nResult: Fail to reject the null hypothesis. No significant relationship found between complaint type and location.")


plt.figure(figsize=(8, 6))
sns.heatmap(contingency_table, annot=True, cmap="YlGnBu", fmt="d")
plt.title("Contingency Table: Complaint Type vs. Location")
plt.ylabel("Complaint Type")
plt.xlabel("Location")
plt.tight_layout()
plt.show()
```

Figure 32:Whether the type of complaint or service requested and location are related.

In this test 2 we have to check this code if there is a connection between the type of complaint (like noise or garbage) and the location (like North or South). It first counts how many complaints happen in each area using a table. Then, it uses a special test called the Chi-Square Test to see if these differences are just random or actually meaningful. If the result shows a p-value less than 0.05, it means the complaint type is related to the location. If it's higher, there's no strong link. Finally, it shows a heatmap—a colorful table—that helps us see which types of complaints are common in which locations.

5.2.2:Perform the statistical test and provide the p-value.

To test this, a Chi-Square Test of Independence is used. This test compares observed complaint counts across locations to what we would expect if there were no relationship. The output gives us a p-value – a number that tells us how likely the results could be due to chance. For example, if the p-value is greater than 0.05, we say there's no significant relationship, and we fail to reject the null hypothesis. If it's less than 0.05, we have strong evidence of a real connection between complaint type and location, and we reject the null hypothesis (Pallant.j, 2016).

```
Chi-Square Statistic: 8.333333333333334
Degrees of Freedom: 6
P-Value: 0.21468530608628178

Expected Frequencies Table:
 [[0.6 0.9 0.9 0.6]
 [0.8 1.2 1.2 0.8]
 [0.6 0.9 0.9 0.6]]
```

Figure 33: output of the Whether the type of complaint or service requested and location are related.

This output means that after checking the data, there is no strong connection between the type of complaint and the location where it happened. The p-value is 0.21, which is more than 0.05, so we do not have enough evidence to say that complaint types are related to specific places. The expected table shows how many complaints we would expect in each location if there was no link. So, overall, this result tells us that complaint types happen fairly randomly across locations.

5.2.3:Interpret the results to accept or reject the Null Hypothesis.

Since the p-value (0.215) is greater than 0.05, we fail to reject the null hypothesis. This means there is no statistically significant evidence that the type of complaint is related to the location. In simpler terms, complaints appear to be evenly distributed across different locations without a clear pattern.Thus, we conclude that location does not influence thetype of complaint in a meaningful statistical way, based on the sample data.

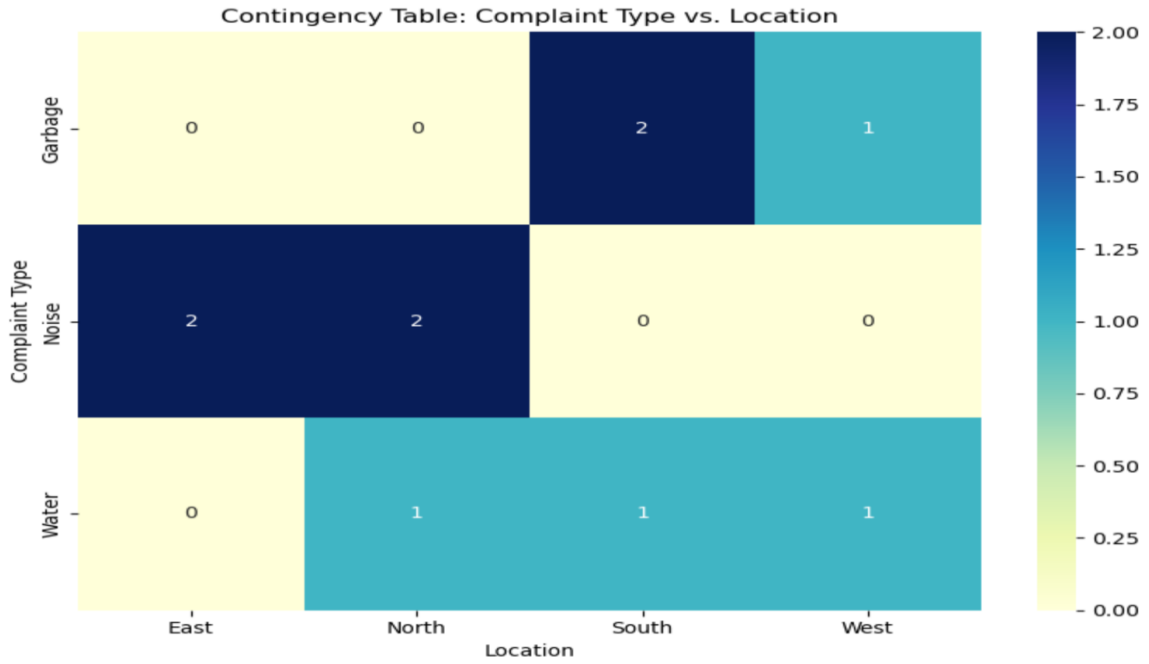

Figure 34: output of the Whether the type of complaint or service requested and location are related.

6.Conclusion

In conclusion, the “Customer Service Requests from 2010 to Present” dataset is very useful for understanding the problems that people in New York City face and how the city responds to them. By looking at the types of complaints (like noise, illegal parking, or water issues), where they happen, and how long it takes to fix them, we can see which areas have more problems, how fast the city reacts, and what issues are most common. This helps the government improve its services, plan better, and make life better for people in the city. It also gives researchers and citizens helpful information so they can suggest changes or improvements.

7.References

1.Anon., 2011. n Overview of Business Intelligence Technology. *Dayal, U. and Narasayya V,* Volume 88-98, p. 54(8).

2.Anon., 2024. *Geeks for Geeks.* [Online]
Available at: https://www.geeksforgeeks.org
[Accessed 31 03 2024].

3.Anon., 2024. *Geeks for Geeks.* [Online]
Available at: https://www.geeksforgeeks.org
[Accessed 31 03 2024].

4.Anon., 2025. *Geeks for Geeks.* [Online]
Available at: https://www.geeksforgeeks.org
[Accessed 11 04 2025].

5.Anon., 2025. *Null and alternative hypothesis - Statistical resources.* [Online]
Available at: https://resources.nu.edu/statsresources

6.Anon., 2025. *W3Schools.* [Online]
Available at: https://www.w3schools.com
[Accessed 2024].

7.Dayal, U. a. N., 2011. An Overview of Business Intelligence Technology. *Chaudhauri.S,* Volume 88-98, p. 54(8).

8.Field.A, 2013. *Discovering ststistics Using IBM SPSS Statistics.* 4th edition ed. s.l.:SAGE publications.

9.Pallant.j, 2016. *SPSS Survival Manual.* 6th edition ed. s.l.:McGraw-Hill Education.

10.Willkenton, 2021. *Investopedia.* [Online]
Available at: https://www.investopedia.com
[Accessed 31 07 2021].