



**islington college**  
(इरिलिङ्टन कलेज)



**Module Code & Module Title**

**CC5061NI Applied Data Science**

**60% Individual Coursework**

**Submission : Final Submission**

**Academic Semester: Spring Semester 2025**

**Credit: 15 credit semester long module**

**Student Name: Santoshi Paudel**

**London Met ID: 23048656**

**College ID: np01ai4a230103@islingtoncollege.edu.np**

**Assignment Due Date: Thursday, May 15, 2025**

**Assignment Submission Date: Wednesday, May 14, 2025**

**Submitted To: Dipeshor Silwal / Alish KC sir**

*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.*

## 23048656 Santoshi Paudel(ADS Final).docx

 Islington College, Nepal

### Document Details

Submission ID

trn:oid::3618:95822556

Submission Date

May 14, 2025, 1:58 PM GMT+5:45

Download Date

May 14, 2025, 2:01 PM GMT+5:45

File Name

23048656 Santoshi Paudel(ADS Final).docx

File Size

53.8 KB

61 Pages





8,727 Words

45,010 Characters




## 18% Overall Similarity

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

### Match Groups

-  **149 Not Cited or Quoted 18%**  
Matches with neither in-text citation nor quotation marks
-  **1 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
-  **1 Cited and Quoted 0%**  
Matches with in-text citation present, but no quotation marks

### Top Sources

- 7%  Internet sources
- 3%  Publications
- 16%  Submitted works (Student Papers)

### Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

# Table of Contents

<b>1. Data Understanding .....</b>	<b>1</b>
1.1. About the Data .....	1
1.2. Dataset Column Description Table .....	3
<b>2. Data Preparation.....</b>	<b>11</b>
2.1. Importing the Given Dataset.....	11
2.2. Providing the Insight on the Information and Details that the provided Dataset Carries.....	13
2.3. Converting the column “Created Date” and “Closed Date” to datetime datatype and Creating a new Column “Request_Closing_Time” as the Time Elapsed between Request Creation and Request Closing.....	15
2.4. Write a Python Program to Drop Irrelevant Columns .....	19
2.5. Write a Python Program to Remove the NaN Missing Values from Updated Dataframe. ....	21
2.6. Write a Python Program to See the Unique Values from all the Columns in the Dataframe .....	25
<b>3. Data Analysis.....</b>	<b>26</b>
3.1. Write a Python program to show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame.....	26
3.2. Write a Python Program to Calculate and Show Correlation of all Variables ..	28
<b>4. Data Exploration.....</b>	<b>30</b>
4.1. Provide four major Insights through Visualization that you Come up after Data Mining.....	30
4.1.1. Most Common complain types across New York City .....	30
4.1.2. Number of Request Open and Closed .....	32

4.1.3. Agency handling the most Requests .....	34
4.1.4. Complaint which takes the Longest Time to get Solved .....	36
4.2. Arrange the complaint types according to their average 'Request_Closing_Time', categorized by various locations. Illustrate it through graph as well .....	38
<b>5. Statistical Testing .....</b>	<b>41</b>
5.1. Test 1: Whether the average response time across complaint types is similar or not .....	41
5.2. Test 2: Whether the type of complaint or service requested and location are related .....	43
<b>6. Conclusion .....</b>	<b>45</b>
<b>7. References.....</b>	<b>46</b>

## Table of Figures

Figure 1: Importing the pandas Library as "pd" .....	11
Figure 2: Loading the Dataset from the provided csv file.....	11
Figure 3: Screenshot displaying the loaded Dataset.....	12
Figure 4: Code used to convert the column "Created Date" and "Closed Date" to datetime datatype .....	15
Figure 5: Output that shows the converted datatype of "Created Date" and "Closed Date .....	16
Figure 6: Screenshot of Created Date and Closed Date before converting to datetime datatype .....	16
Figure 7: Screenshot of Created Date and Closed Date before converting to datetime datatype .....	16
Figure 8: Code used to create a new Column "Request_Closing_Time" as the Time Elapsed between Request Creation and Request Closing .....	17
Figure 9: Displaying the new Column "Request_Closing_Time" .....	17
Figure 10: Writing a python program to drop irrelevant Columns .....	19
Figure 11: Output after dropping the irrelevant Columns .....	19
Figure 12: Output before dropping the irrelevant Columns.....	20
Figure 13: Output after dropping the irrelevant Columns.....	20
Figure 14: Python program to remove the NaN missing values from updated dataframe. .....	21
Figure 15: Python Program to check if the Null values still exist in columns.....	21
Figure 16: Result from the Null Value existence check in Column .....	22
Figure 17: Python Program to check if the Null values still exist in rows .....	22
Figure 18: Result from the Null Value existence check in Rows .....	22
Figure 19: Output before removing the missing values.....	23
Figure 20: Output after removing the missing values.....	24
Figure 21: Write a python program to see the unique values from all the columns in the dataframe.....	25
Figure 22: Unique values from all the columns in the dataframe.....	25

Figure 23: Python program to show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame .....	26
Figure 24 :Summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame .....	27
Figure 25: Python program to calculate and show correlation of all variables .....	28
Figure 26: Correlation of all variables .....	28
Figure 27: Code Used to create the Bar Graph of top 10 Complaints in NYC 311 .....	30
Figure 28: Screenshot of the Bar Graph Displaying the Top 10 Complaints in NYC 311 .....	31
Figure 29: Code used to find the number of open and closed requests.....	32
Figure 30: Number of Request Closed and Open .....	32
Figure 31: Bar graph showing the number of open and closed case .....	33
Figure 32: Code used to find the number of requests handled by each agency.....	34
Figure 33: The number of Agency handling the most requests .....	34
Figure 34: Bar graph showing the agency that solve the most cases .....	35
Figure 35: Code used to find the complaint which takes the Longest Time to get Solved .....	36
Figure 36: Details of the Complaint which takes the Longest Time to get Solved .....	36
Figure 37: Bar showing the Complaint which takes the longest time to get Solved.....	37
Figure 38: Code used to arrange the complaint types according to their average 'Request_Closing_Time', categorized by various locations and Illustrating it through graph as well.....	38
Figure 39: Graph showing the complaint types according to their average 'Request_Closing_Time', categorized by various locations .....	39
Figure 40: Code showing ANNOVA Test.....	41
Figure 41: Result from the Test .....	42
Figure 42: Code showing Chi-Square Test .....	43
Figure 43: Result from the Test .....	44

## Table of Tables

Table 1: Dataset column description table .....	10
-------------------------------------------------	----



## 1. Data Understanding

In data science, successfully executing a project begins with clearly defining the problem and exploring the data deeply. Data understanding requires analyzing its core elements to reveal its structure and key details. This phase focus on several objectives. Firstly, it involves examining the dataset's fundamental or basic properties, such as type of variable and their attributes. Additionally, data understanding includes summarizing the dataset by determining essential features such as data type, number of variables it contains and size of the data (3Pillar Global, 2025).

### 1.1. About the Data

This dataset was provided by a New York City government agency and contains a large amount of information about service requests made through NYC311. NYC311 is a platform that people living in New York City use to report non-emergency problems or ask for help with city services. Every day, thousands of people contact NYC311 to report issues in their area. These problems can include things like loud noise, illegal parking, broken streetlights, dirty streets, animal abuse, or people urinating in public. The dataset focuses only on these kinds of service complaints and gives us a closer look at how people use the system and what types of complaints are most common.

The dataset is quite large. It includes 300,698 rows and 53 columns. Each row in the dataset represents a single complaint made by a person, and each column provides a specific detail about that complaint. For example, it shows the exact time and date when the complaint was created and when it was closed. It also shows which city agency or department handled the issue. There is information about the type of complaint, such as whether it was about noise, street conditions, or parking, and a more detailed description of the problem. The dataset also includes location information, such as the street name, ZIP code, full address, and nearby cross streets or intersections. This helps us understand exactly where the problem happened.

In addition to location, the dataset also includes details like which borough the complaint came from. There are also columns that mention community boards, landmarks, and nearby facilities like parks or schools. Some complaints are even connected to taxi services, vehicles, bridges, highways, or ferry terminals. The dataset includes geographic coordinates such as latitude and longitude so that the complaints can be shown on a map during analysis. There are also several columns that help track how the complaint was handled. These include the status of the request (open or closed), the due date for solving it, the actions taken by the agency, and the date when the resolution was last updated.

One important thing to note is that the dataset contains missing or empty values in some columns. Not every complaint includes all the information. These missing entries are found in various fields, particularly in columns related to landmarks, facility types, school-related information, transportation, and infrastructure identifiers. A large number of entries in the fields like Landmark, Facility Type, School Name, School Region, School Code, and School Address are missing from the dataset. This likely means that the details were either not available at the time of submission or were not necessary for most of the complaints filed. Most of the data in columns like Vehicle Type, Taxi Company Borough, Taxi Pick Up Location, Garage Lot Name, Ferry Direction, and Ferry Terminal Name is missing. In the same way, columns like Bridge Highway Name, Road Ramp, and Bridge Highway Segment are also often left blank. These missing values are common in large datasets like this one, but they must be handled carefully during data cleaning so that the analysis remains accurate and meaningful.

**1.2. Dataset Column Description Table**

S. No	Column Name	Description	Data Type
1.	Unique Key	In the dataset, this serves as a unique identifier of a Service Request (SR).	String
2.	Created Date	The date in which the Service Request was created.	Floating Timestamp
3.	Closed Date	The date in which the Service Request was closed by responding agency.	Floating Timestamp
4.	Agency	It refers to governing body handling the request.	String
5.	Agency Name	The full or official name of the agency handling or responding to the request.	String
6.	Complaint Type	The topic of incident or condition is identified here. The Complaint type may be linked to specific descriptor or may exist independently.	String
7.	Descriptor	This relates to the Complaint type and provides more specific information about the incident or situation. Descriptor value may not always be necessary in Service Request and it vary based on complaint type.	String

<b>8.</b>	Location Type	It specifies or describes what kind of place or location is mentioned in the address information.	String
<b>9.</b>	Incident Zip	It is the zip code of the incident location. This code is provided by geo validation.	String
<b>10.</b>	Incident Address	The addresses where incidents were reported or take place. It includes house/building number of the place where incident take place and is provided by the submitter.	String
<b>11.</b>	Street Name	It includes the name of the street where incidents were reported, basically the street name of incident address.	String
<b>12.</b>	Cross Street 1	The first nearby street that crosses the location where incident occurred, based on map data.	String
<b>13.</b>	Cross Street 2	The second nearby street that crosses the location where incident occurred, based on map data.	String
<b>14.</b>	Intersection Street 1	The first nearby street that intersect the location where incident occurred. It is based on geo validated incident location.	String

<b>15.</b>	Intersection Street 2	The Second nearby street that intersect the location where incident occurred. It is based on geo validated incident location.	String
<b>16.</b>	Address Type	It describes what kind of address detail is provided for the given incident.	String
<b>17.</b>	City	It provides the city of the incident location. It is provided by geo validation.	String
<b>18.</b>	Landmark	It displays the name of the landmark if the incident location is identified as a landmark.	String
<b>19.</b>	Facility Type	It refers to the kind of service facility associated with the incident. This field indicate the type of city facility connected to SR.	String
<b>20.</b>	Status	This field shows the current state of the Service Request (SR). If the status is 'Closed' that it shows that the service request has been completed.	String
<b>21.</b>	Due Date	The date when the SR is expected to get updated by the responding agency. This is based on complaint type.	Floating Timestamp

<b>22.</b>	Resolution Description	It summarizes the recent action performed by the responding agency on SR and may also mention upcoming steps if needed.	String
<b>23.</b>	Resolution Action Updated Date	It is the date when the SR gets last updated by the responding agency.	Date
<b>24.</b>	Community Board	These are local government bodies that facilitates communication between residents and city officials.	String
<b>25.</b>	Borough	It consists of the borough (district or county) information which is provided by the person submitting the request and is confirmed by geo validation.	String
<b>26.</b>	X Coordinate (State Plane)	It presents the east-west position of the incident location and is confirmed by geo validation.	Number
<b>27.</b>	Y Coordinate (State Plane)	It represents the vertical position of the incident location and is confirmed by geo validation.	Number
<b>28.</b>	Park Facility Name	This field indicates the name of the parks facility where the incident occurs if the incident location is a parks dept facility.	String

<b>29.</b>	Park Borough	This field indicates the name of the parks borough where the incident occurs if the incident location is a parks dept facility.	String
<b>30.</b>	School Name	This field indicates the name of the School where or around which the incident occurs if the incident location is a school area.	String
<b>31.</b>	School Number	This field indicates the school number of the school where or around which the incident is reported if the incident location is a school area.	String
<b>32.</b>	School Region	This field indicates the region of the school in or around which the incident is reported if the incident location is a school area.	String
<b>33.</b>	School Code	This field shows the designated school zone code for incidents occurring within the school premises or the immediate surrounding area if the incident location is a school area.	String
<b>34.</b>	School Phone Number	This field shows the school's phone number if the incident	Number

		took place at the school or nearby.	
<b>35.</b>	School Address	This field shows the address of the school if the incident took place or is reported at the school or nearby.	String
<b>36.</b>	School City	This field displays the name of the city where the school is located if the incident had happened at the school or nearby.	String
<b>37.</b>	School State	This field displays the name of that state where the school is located if the incident had happened at the school or nearby.	String
<b>38.</b>	School Zip	It is the zip code of the incident location. This field displays the zip code of the school location if the incident had happened at the school or nearby.	String
<b>39.</b>	School Not Found	This field indicate that no incident relates to any school association.	Boolean
<b>40.</b>	School or Citywide Complaint	It specifies if the SR is associated to school-related issue or a general concern affecting the entire city.	String
<b>41.</b>	Vehicle Type	If the incident is related to vehicle, this field describes	String



		the type of vehicle involved in the incident.	
<b>42.</b>	Taxi Company Borough	If the incident involves a taxi, this field describes the borough where the taxi company is based.	String
<b>43.</b>	Taxi Pick Up Location	If the incident involves a taxi, this field displays the taxi pick up location.	String
<b>44.</b>	Bridge Highway Name	This field displays the name of the Bridge/Highway if the incident is related to Bridge/Highway.	String
<b>45.</b>	Bridge Highway Direction	If the incident is related to Bridge/Highway, the direction where the issue occurred will be displayed here.	String
<b>46.</b>	Road Ramp	If the incident occurred on Bridge/ Highway, this field indicates whether the issue took place on the main road or on a ramp.	String
<b>47.</b>	Bridge Highway Segment	This field provides the additional information on the section of Bridge/ Highway where the incident had occurred.	String
<b>48.</b>	Garage Lot Name	This field displays the name of the Garage or parking lot related to SR, if the issue is	String

		about the parking or vehicle storage issue.	
<b>49.</b>	Ferry Direction	This field describes the direction in which the ferry travels at the time of incident.	String
<b>50.</b>	Ferry Terminal Name	It refers to the name of ferry terminal associated to the service request.	String
<b>51.</b>	Latitude	It represents the north-south position i.e., the geographical latitude coordinate of the incident location.	Number
<b>52.</b>	Longitude	It represents the east or west position i.e., the geographical longitude coordinate of the incident location.	Number
<b>53.</b>	Location	It represents the combination of geographical longitude and latitude representing the exact location the incident.	String

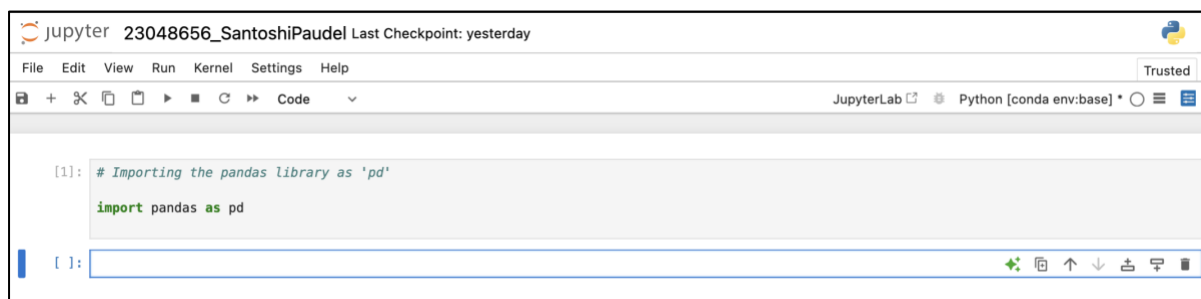
*Table 1: Dataset column description table*

## 2. Data Preparation

Data preparation is a process where data combining, structuring and organizing is performed. Alternatively, data preparation is also known as data wrangling. Commonly, data is often created with missing values, errors or inaccuracies, so data preparation involves improving data quality, correcting data errors, removing missing values and consolidating data sets (Stedman, 2024).

### 2.1. Importing the Given Dataset

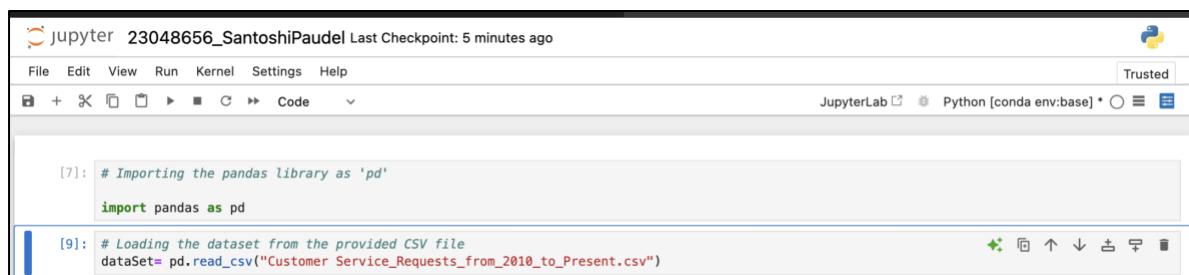
To import the given dataset, firstly library named pandas was imported assigning its aliases as 'pd'.

A screenshot of a JupyterLab interface. The top bar shows the username '23048656\_SantoshiPaudel' and 'Last Checkpoint: yesterday'. The menu bar includes File, Edit, View, Run, Kernel, Settings, and Help. The toolbar has icons for file operations and a 'Code' dropdown. The main area contains a code cell with the following text: 

```
[1]: # Importing the pandas library as 'pd'
import pandas as pd
```

 The status bar at the bottom indicates 'Python [conda env:base]'.

Figure 1: Importing the pandas Library as "pd"

A screenshot of a JupyterLab interface. The top bar shows the username '23048656\_SantoshiPaudel' and 'Last Checkpoint: 5 minutes ago'. The menu bar includes File, Edit, View, Run, Kernel, Settings, and Help. The toolbar has icons for file operations and a 'Code' dropdown. The main area contains two code cells. The first cell has the same import statement as Figure 1. The second cell has the following text: 

```
[9]: # Loading the dataset from the provided CSV file
dataSet= pd.read_csv("Customer Service_Requests_from_2010_to_Present.csv")
```

 The status bar at the bottom indicates 'Python [conda env:base]'.

Figure 2: Loading the Dataset from the provided csv file

The code in the screenshot shows how a CSV (Comma-Separated Values) file named 'Customer Service\_Requests\_from\_2010\_to\_Present.csv' is opened and read using the Pandas library in Python. It uses a function called `pd.read_csv()`, which is commonly used to read csv files. The function `pd.read_csv()` is one of the core functions provided by the Pandas library for reading CSV files. This function reads the contents of the specified CSV file and automatically converts the data into a DataFrame.

In this case, the result of the `pd.read_csv()` function is stored in a variable called `dataSet`, making it easier to access, analyze, and manipulate the data later in the code.

[11]: # Displaying the loaded dataset  
dataSet

[11]:

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	Bridge Highway Name	Bridge Highway Direction
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
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
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
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
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
...	...	...	...	...	...	...	...	...	...	...	...	...
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
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
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
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
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

300698 rows x 53 columns

Figure 3: Screenshot displaying the loaded Dataset

The screenshot above displays the loaded dataset by just writing the name of the variable in which the result from the csv file is stored.

## **2.2. Providing the Insight on the Information and Details that the provided Dataset Carries**

This dataset talks about different problems people in New York City have reported using the 311 service. These are not emergency problems, but common issues like loud noise, cars blocking driveways, animal problems, people using the street as a bathroom, and broken roads. Each row in the dataset shows one complaint from someone, and each column gives a different detail about that complaint. The dataset is very big. It has 300,698 rows and 53 columns. This means it shows over 300,000 complaints and gives 53 types of information about each one. It shows when the complaint was made and when it was closed. It also tells which city department handled the complaint and what the complaint was about, like if it was about noise or parking. It also has a short explanation of what happened. The dataset also gives us a lot of information about the place where each complaint happened. It tells us the name of the street, the ZIP code, and the full address. It even tells us what kind of place it was like, if the problem happened on a street, in a park, near someone's home, or in another type of location. Some complaints are even connected to taxi services, vehicles, bridges, highways, or ferry terminals. The dataset includes geographic coordinates such as latitude and longitude so that the complaints can be shown on a map during analysis. There are also several columns that help track how the complaint was handled. These include the status of the request (open or closed), the due date for solving it, the actions taken by the agency, and the date when the resolution was last updated.

The dataset contains missing or empty values in some columns. Not every complaint includes all the information. These missing entries are found in various fields, particularly in columns related to landmarks, facility types, school-related information, transportation, and infrastructure identifiers. A large number of entries in the fields like Landmark, Facility Type, School Name, School Region, School Code, and School Address are missing from the dataset. This likely means that the details were either not available at the time of submission or were not necessary for most of the complaints filed. Most of the data in columns like Vehicle

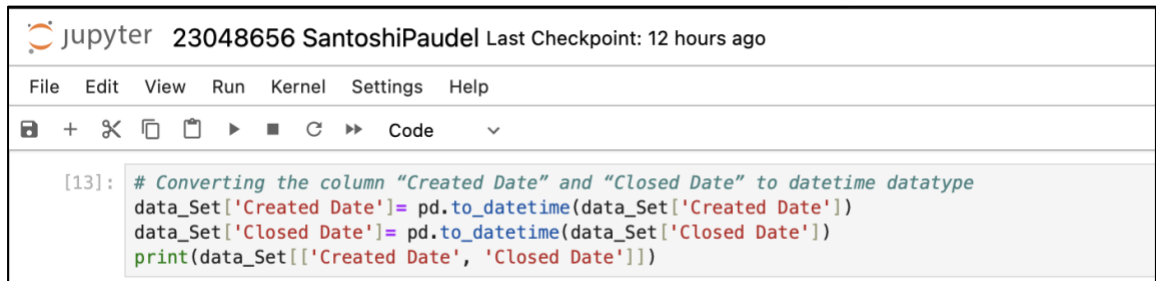
Type, Taxi Company Borough, Taxi Pick Up Location, Garage Lot Name, Ferry Direction, and Ferry Terminal Name is missing.

In general, this dataset is really useful for learning about the types of problems that people in New York City are facing, where these problems are taking place, and how the city is handling them. It includes both tiny details, like the exact time a complaint was made. By studying this data, the city can make its services better and solve common problems more quickly and effectively.

### 2.3. Converting the column “Created Date” and “Closed Date” to datetime datatype and Creating a new Column “Request\_Closing\_Time” as the Time Elapsed between Request Creation and Request Closing

The python program used to convert the column ‘Created Date’ and ‘Closed Date’ to datetime datatype is given below.

#### Code:



```
[13]: # Converting the column "Created Date" and "Closed Date" to datetime datatype
data_Set['Created Date'] = pd.to_datetime(data_Set['Created Date'])
data_Set['Closed Date'] = pd.to_datetime(data_Set['Closed Date'])
print(data_Set[['Created Date', 'Closed Date']])
```

Figure 4: Code used to convert the column “Created Date” and “Closed Date” to datetime datatype

The screenshot above shows a piece of code that is used to change the format of two specific columns in a dataset — one named ‘Created Date’ and the other named ‘Closed Date’. These two columns contain date values that tell us when a service request was made and when it was completed. However, when data is first read from a CSV file in Python using the Pandas library, all values are treated as plain text, even if they look like dates. The computer does not automatically understand them as real dates.

To solve this problem, the code uses a special function called `pd.to_datetime()` from the Pandas library. This function is designed to take text that looks like a date and convert it into a proper datetime format. When the values in ‘Created Date’ and ‘Closed Date’ are converted using this function, they are no longer just strings of text. They become actual date objects, which makes it possible to perform tasks like comparing dates, finding the difference between two dates and much more.

After converting the dates, the `print()` function is used to display only the ‘Created Date’ and ‘Closed Date’ columns from the dataset. This helps to check whether the conversion was done correctly.

**Output:**

```
print(data_Set[['Created Date', 'Closed Date']])
```

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

[300698 rows x 2 columns]

Figure 5: Output that shows the converted datatype of "Created Date" and "Closed Date"

Lastly, after the code runs and the print() function is used, the final output shows the two columns: 'Created Date' and 'Closed Date' with dates that are now displayed in a clear and consistent format after being converted using the pd.to\_datetime() function. The output is helpful because it shows the exact date and time for each service request in a format that is easy to use for further analysis.

**'Created Date' and 'Closed Date' before converting to datetime datatype:**

	Created Date	Closed Date
0	12/31/2015 11:59:45 PM	01-01-16 0:55
1	12/31/2015 11:59:44 PM	01-01-16 1:26
2	12/31/2015 11:59:29 PM	01-01-16 4:51
3	12/31/2015 11:57:46 PM	01-01-16 7:43
4	12/31/2015 11:56:58 PM	01-01-16 3:24
...	...	...
300693	03/29/2015 12:33:41 AM	NaN
300694	03/29/2015 12:33:28 AM	03/29/2015 02:33:59 AM
300695	03/29/2015 12:33:03 AM	03/29/2015 03:40:20 AM
300696	03/29/2015 12:33:02 AM	03/29/2015 04:38:35 AM
300697	03/29/2015 12:33:01 AM	03/29/2015 04:41:50 AM

[300698 rows x 2 columns]

Figure 6: Screenshot of Created Date and Closed Date before converting to datetime datatype

**'Created Date' and 'Closed Date' after converting to datetime datatype:**

```
print(data_Set[['Created Date', 'Closed Date']])
```

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

[300698 rows x 2 columns]

Figure 7: Screenshot of Created Date and Closed Date before converting to datetime datatype



The python program used to create a new column named 'Request\_Closing\_Time' is given below.

### **Code:**

```
[25]: # creating a new column "Request_Closing_Time" as the time elapsed between request creation and request closing
data_Set['Request_Closing_Time'] = data_Set['Closed Date'] - data_Set['Created Date']
print("\n      Request_Closing_Time")
print("-----")
print(data_Set['Request_Closing_Time'])
```

Figure 8: Code used to create a new Column "Request\_Closing\_Time" as the Time Elapsed between Request Creation and Request Closing

The screenshot above shows the code used to create a new column called 'Request\_Closing\_Time', which calculates the time that passed between when a service request was created and when it was closed. The code uses the two existing columns namely, 'Created Date' and 'Closed Date', which contain the dates and times for when the request was opened and closed. The code subtracts the 'Created Date' from the 'Closed Date', and since both columns are in the correct date and time format, this subtraction gives the time difference between them. The time difference is then saved in the new 'Request\_Closing\_Time' column, showing how long it took to close each service request.

Finally, the print() function is used to display only the 'Request\_Closing\_Time' column. This helps to check whether result was calculated correctly or not.

### **Output:**

```
Request_Closing_Time
-----
0      0 days 00:55:15
1      0 days 01:26:16
2      0 days 04:51:31
3      0 days 07:45:14
4      0 days 03:27:02
...
300693      NaT
300694      0 days 02:00:31
300695      0 days 03:07:17
300696      0 days 04:05:33
300697      0 days 04:08:49
Name: Request_Closing_Time, Length: 300698, dtype: timedelta64[ns]
```

Figure 9: Displaying the new Column "Request\_Closing\_Time"

The output shows the new 'Request\_Closing\_Time' column, which shows how long it took to close each service request. It shows the time difference between when the request was created and when it was closed. This time difference is displayed in days, hours, and minutes.

This output allows to easily see how much time was taken to close each service request and this information can be used to find ways to improve the service request process.

## 2.4. Write a Python Program to Drop Irrelevant Columns

### Code:

```
# Droppin the irreleevant columns
irrelevant_columns = [
    'Agency Name', 'Incident Address', 'Street Name', 'Cross Street 1', 'Cross Street 2',
    'Intersection Street 1', 'Intersection Street 2', 'Address Type', 'Park Facility Name',
    'Park Borough', 'School Name', 'School Number', 'School Region', 'School Code',
    'School Phone Number', 'School Address', 'School City', 'School State', 'School Zip',
    'School Not Found', 'School or Citywide Complaint', 'Vehicle Type', 'Taxi Company Borough',
    'Taxi Pick Up Location', 'Bridge Highway Name', 'Bridge Highway Direction', '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'
]

data_Set.drop(columns=irrelevant_columns, inplace=True, errors='ignore')

print(data_Set.columns)
```

Figure 10: Writing a python program to drop irrelevant Columns

This program illustrates the removal of non-essential columns from the dataset to improve its clarity and analytical value. Eliminating these irrelevant fields helps simplify the DataFrame, making it more focused and efficient for subsequent analysis.

In the screenshot provided above, the irrelevant columns are using the drop () function. By using the parameter inplace=True, the original DataFrame is updated directly without needing to assign it again. The updated DataFrame columns is then printed to confirm the removal of the unnecessary columns.

### Output:

```
[12]: print(data_Set.columns)
Index(['Unique Key', 'Created Date', 'Closed Date', 'Agency', 'Complaint Type',
      'Descriptor', 'Location Type', 'Incident Zip', 'City', 'Status',
      'Resolution Description', 'Borough', 'Latitude', 'Longitude',
      'Request_Closing_Time'],
      dtype='object')
```

Figure 11: Output after dropping the irrelevant Columns

The output shows a simpler table after removing the columns that were not needed. This was done using the drop() function. Now, the table only keeps the important columns, like the type of complaint, when it was created and closed, Agency and its current status. Removing the extra columns also helps the program work faster and makes it easier to study the data.

**The dataset before dropping the Irrelevant column:**

```
print(data_Set.columns)

Index(['Unique Key', 'Created Date', 'Closed Date', 'Agency', 'Agency Name',
      'Complaint Type', 'Descriptor', 'Location Type', 'Incident Zip',
      'Incident Address', 'Street Name', 'Cross Street 1', 'Cross Street 2',
      'Intersection Street 1', 'Intersection Street 2', 'Address Type',
      'City', 'Landmark', 'Facility Type', 'Status', 'Due Date',
      'Resolution Description', 'Resolution Action Updated Date',
      'Community Board', 'Borough', 'X Coordinate (State Plane)',
      'Y Coordinate (State Plane)', 'Park Facility Name', 'Park Borough',
      'School Name', 'School Number', 'School Region', 'School Code',
      'School Phone Number', 'School Address', 'School City', 'School State',
      'School Zip', 'School Not Found', 'School or Citywide Complaint',
      'Vehicle Type', 'Taxi Company Borough', 'Taxi Pick Up Location',
      'Bridge Highway Name', 'Bridge Highway Direction', 'Road Ramp',
      'Bridge Highway Segment', 'Garage Lot Name', 'Ferry Direction',
      'Ferry Terminal Name', 'Latitude', 'Longitude', 'Location',
      'Request_Closing_Time'],
      dtype='object')
```

*Figure 12: Output before dropping the irrelevant Columns***The dataset after dropping the Irrelevant column:**

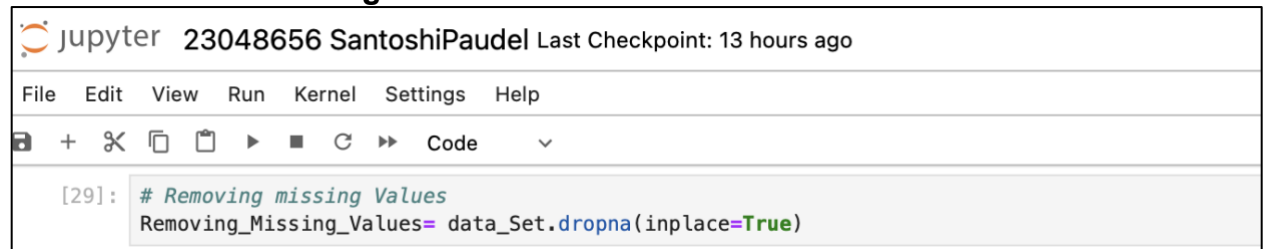
```
[12]: print(data_Set.columns)

Index(['Unique Key', 'Created Date', 'Closed Date', 'Agency', 'Complaint Type',
      'Descriptor', 'Location Type', 'Incident Zip', 'City', 'Status',
      'Resolution Description', 'Borough', 'Latitude', 'Longitude',
      'Request_Closing_Time'],
      dtype='object')
```

*Figure 13: Output after dropping the irrelevant Columns*

## 2.5. Write a Python Program to Remove the NaN Missing Values from Updated Dataframe.

**Code to remove missing values:**



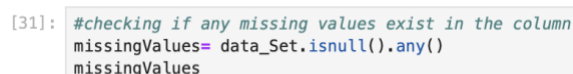
```
[29]: # Removing missing Values
      Removing_Missing_Values = data_Set.dropna(inplace=True)
```

Figure 14: Python program to remove the NaN missing values from updated dataframe.

This code removes missing (NaN) values from the DataFrame using the `dropna()` function. By setting `inplace=True`, the DataFrame is updated directly, so there's no need to assign the result to another variable.

Missing values are removed to keep the data clean and easy to work with. If a row in the dataset has empty or missing parts (called NaN), it can cause problems when doing things like calculations or visualization. It helps the analysis to be correct and also makes sure the program runs smoothly without breaking because of missing data.

**Code to check if the Null values still exist in columns:**



```
[31]: #checking if any missing values exist in the column
      missingValues = data_Set.isnull().any()
      missingValues
```

Figure 15: Python Program to check if the Null values still exist in columns

The code utilizes the `isnull()` function to examine each column for missing values. It generates a Boolean DataFrame where each cell is marked as `True` if it contains a null value, and `False` if it does not. This allows for a clear, column-by-column view of where null entries are present within the dataset.

**Output:**

```
[31]: Unique Key          False
      Created Date       False
      Closed Date        False
      Agency             False
      Complaint Type     False
      Descriptor         False
      Location Type      False
      Incident Zip       False
      City              False
      Status            False
      Resolution Description False
      Borough           False
      Latitude          False
      Longitude         False
      Request_Closing_Time False
      dtype: bool
```

*Figure 16: Result from the Null Value existence check in Column*

The screenshot above shows that, no null values exist in any column of the DataFrame.

**Code to check if the null values exist in any row of the DataFrame:**

```
[17]: missingValues= data_Set.isnull().any(axis=1)
      missingValues
```

*Figure 17: Python Program to check if the Null values still exist in rows*

The program also checks for missing values across each row using the `isnull().any(axis=1)` function. It generates a Boolean DataFrame where each cell is marked as True if it contains a null value, and False if it does not. This allows for a clear, row-by-row view of where null entries are present within the dataset.

**Output:**

```
[17]: 0      False
      1      False
      2      False
      3      False
      4      False
      ...
      300692 False
      300694 False
      300695 False
      300696 False
      300697 False
      Length: 291107, dtype: bool
```

*Figure 18: Result from the Null Value existence check in Rows*

Since the dataset contains no null entries, the screenshot above confirms that every row is free from missing values.

## Dataset before removing the NaN missing values:

```
[10... print(data_Set)
```

	Unique Key	Created Date		Closed Date		Agency	\
0	32310363	2015-12-31	23:59:45	2016-01-01	00:55:00	NYPD	
1	32309934	2015-12-31	23:59:44	2016-01-01	01:26:00	NYPD	
2	32309159	2015-12-31	23:59:29	2016-01-01	04:51:00	NYPD	
3	32305098	2015-12-31	23:57:46	2016-01-01	07:43:00	NYPD	
4	32306529	2015-12-31	23:56:58	2016-01-01	03:24:00	NYPD	
...	...	...	...	...	...	...	...
300693	30281872	2015-03-29	00:33:41		NaT	NYPD	
300694	30281230	2015-03-29	00:33:28	2015-03-29	02:33:59	NYPD	
300695	30283424	2015-03-29	00:33:03	2015-03-29	03:40:20	NYPD	
300696	30280004	2015-03-29	00:33:02	2015-03-29	04:38:35	NYPD	
300697	30281825	2015-03-29	00:33:01	2015-03-29	04:41:50	NYPD	

	Complaint Type		Descriptor		\
0	Noise - Street/Sidewalk		Loud Music/Party		
1	Blocked Driveway		No Access		
2	Blocked Driveway		No Access		
3	Illegal Parking	Commercial	Overnight Parking		
4	Illegal Parking		Blocked Sidewalk		
...	...	...	...	...	...
300693	Noise - Commercial		Loud Music/Party		
300694	Blocked Driveway		Partial Access		
300695	Noise - Commercial		Loud Music/Party		
300696	Noise - Commercial		Loud Music/Party		
300697	Noise - Commercial		Loud Music/Party		

	Location Type	Incident Zip	City	Status	\
0	Street/Sidewalk	10034.0	NEW YORK	Closed	
1	Street/Sidewalk	11105.0	ASTORIA	Closed	
2	Street/Sidewalk	10458.0	BRONX	Closed	
3	Street/Sidewalk	10461.0	BRONX	Closed	
4	Street/Sidewalk	11373.0	ELMHURST	Closed	
...	...	...	...	...	...
300693	Club/Bar/Restaurant	NaN	NaN	Open	
300694	Street/Sidewalk	11418.0	RICHMOND HILL	Closed	
300695	Club/Bar/Restaurant	11206.0	BROOKLYN	Closed	
300696	Club/Bar/Restaurant	10461.0	BRONX	Closed	
300697	Store/Commercial	10036.0	NEW YORK	Closed	

	Resolution Description		Borough	\
0	The Police Department responded and upon arriv...		MANHATTAN	
1	The Police Department responded to the complai...		QUEENS	
2	The Police Department responded and upon arriv...		BRONX	
3	The Police Department responded to the complai...		BRONX	
4	The Police Department responded and upon arriv...		QUEENS	
...	...	...	...	...
300693	Your complaint has been forwarded to the New Y...		Unspecified	
300694	The Police Department responded and upon arriv...		QUEENS	
300695	The Police Department responded to the complai...		BROOKLYN	
300696	The Police Department responded to the complai...		BRONX	
300697	The Police Department responded to the complai...		MANHATTAN	

	Latitude	Longitude	Request_Closing_Time
0	40.865682	-73.923501	0 days 00:55:15
1	40.775945	-73.915094	0 days 01:26:16
2	40.870325	-73.888525	0 days 04:51:31
3	40.835994	-73.828379	0 days 07:45:14
4	40.733060	-73.874170	0 days 03:27:02
...	...	...	...
300693	NaN	NaN	NaT
300694	40.694077	-73.846087	0 days 02:00:31
300695	40.699590	-73.944234	0 days 03:07:17
300696	40.837708	-73.834587	0 days 04:05:33
300697	40.760583	-73.985922	0 days 04:08:49

[300698 rows x 15 columns]

Figure 19: Output before removing the missing values

## Dataset after removing the NaN missing values:

[45... print(data\_Set)]

	Unique Key	Created Date	Closed Date	Agency	\
0	32310363	2015-12-31 23:59:45	2016-01-01 00:55:00	NYPD	
1	32309934	2015-12-31 23:59:44	2016-01-01 01:26:00	NYPD	
2	32309159	2015-12-31 23:59:29	2016-01-01 04:51:00	NYPD	
3	32305098	2015-12-31 23:57:46	2016-01-01 07:43:00	NYPD	
4	32306529	2015-12-31 23:56:58	2016-01-01 03:24:00	NYPD	
...	...	...	...	...	...
300692	30281370	2015-03-29 00:34:32	2015-03-29 01:13:01	NYPD	
300694	30281230	2015-03-29 00:33:28	2015-03-29 02:33:59	NYPD	
300695	30283424	2015-03-29 00:33:03	2015-03-29 03:40:20	NYPD	
300696	30280004	2015-03-29 00:33:02	2015-03-29 04:38:35	NYPD	
300697	30281825	2015-03-29 00:33:01	2015-03-29 04:41:50	NYPD	
	Complaint Type	Descriptor	\		
0	Noise - Street/Sidewalk	Loud Music/Party			
1	Blocked Driveway	No Access			
2	Blocked Driveway	No Access			
3	Illegal Parking	Commercial Overnight Parking			
4	Illegal Parking	Blocked Sidewalk			
...	...	...	...		
300692	Noise - Commercial	Loud Music/Party			
300694	Blocked Driveway	Partial Access			
300695	Noise - Commercial	Loud Music/Party			
300696	Noise - Commercial	Loud Music/Party			
300697	Noise - Commercial	Loud Music/Party			
	Location Type	Incident Zip	City	Status	\
0	Street/Sidewalk	10034.0	NEW YORK	Closed	
1	Street/Sidewalk	11105.0	ASTORIA	Closed	
2	Street/Sidewalk	10458.0	BRONX	Closed	
3	Street/Sidewalk	10461.0	BRONX	Closed	
4	Street/Sidewalk	11373.0	ELMHURST	Closed	
...	...	...	...	...	...
300692	Store/Commercial	10002.0	NEW YORK	Closed	
300694	Street/Sidewalk	11418.0	RICHMOND HILL	Closed	
300695	Club/Bar/Restaurant	11206.0	BROOKLYN	Closed	
300696	Club/Bar/Restaurant	10461.0	BRONX	Closed	
300697	Store/Commercial	10036.0	NEW YORK	Closed	
	Resolution Description	Borough	\		
0	The Police Department responded and upon arriv...	MANHATTAN			
1	The Police Department responded to the complai...	QUEENS			
2	The Police Department responded and upon arriv...	BRONX			
3	The Police Department responded to the complai...	BRONX			
4	The Police Department responded and upon arriv...	QUEENS			
...	...	...	...		
300692	The Police Department responded to the complai...	MANHATTAN			
300694	The Police Department responded and upon arriv...	QUEENS			
300695	The Police Department responded to the complai...	BROOKLYN			
300696	The Police Department responded to the complai...	BRONX			
300697	The Police Department responded to the complai...	MANHATTAN			
	Latitude	Longitude	Request_Closing_Time		
0	40.865682	-73.923501	0 days 00:55:15		
1	40.775945	-73.915094	0 days 01:26:16		
2	40.870325	-73.888525	0 days 04:51:31		
3	40.835994	-73.828379	0 days 07:45:14		
4	40.733060	-73.874170	0 days 03:27:02		
...	...	...	...		
300692	40.716053	-73.991378	0 days 00:38:29		
300694	40.694077	-73.846087	0 days 02:00:31		
300695	40.699590	-73.944234	0 days 03:07:17		
300696	40.837708	-73.834587	0 days 04:05:33		
300697	40.760583	-73.985922	0 days 04:08:49		

[291107 rows x 15 columns]

Figure 20: Output after removing the missing values



## 2.6. Write a Python Program to See the Unique Values from all the Columns in the Dataframe

**Code:**

```
[51]: # Displaying the unique values from all the columns in the DataFrame
      for column in data_Set.columns:
          unique_values = data_Set[column].unique()
          print(f"Unique Values in '{column}': {unique_values}\n")
```

Figure 21: Write a python program to see the unique values from all the columns in the dataframe

This code loops through every column in the DataFrame and displays the distinct values found in each one. It uses a for loop to go through the list of column names in `data_Set.columns`. For each column, it extracts the unique values using `data_Set[column].unique()` and assigns them to the variable `unique_values`. It then prints a message showing the column name and its corresponding unique values.

**Output:**

```
Unique Values in 'Unique Key': [32310363 32309934 32309159 ... 30283424 30280004 30281825]

Unique Values in '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 00:42:48', '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 00:33:28', '2015-03-29 00:33:03', '2015-03-29 00:33:02',
 '2015-03-29 00:33:01']
Length: 251970, dtype: datetime64[ns]

Unique Values in '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 00:27:00']
```

Figure 22: Unique values from all the columns in the dataframe

This output displays the unique values from three columns of the dataset: 'Unique Key', 'Created Date', and 'Closed Date'. The 'Unique Key' column consists of distinct numbers, each representing a unique complaint or service request, ensuring that each complaint is separately identifiable.

### 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 code to show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame is given below.

**Code to show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame:**

The image shows a JupyterLab window with a single code cell. The code cell contains a Python script that calculates and prints summary statistics for a dataset named 'data\_Set'. The statistics include the sum, mean, standard deviation, skewness, and kurtosis, all calculated for numeric columns only. The code is as follows:

```
[19]: # showing the summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame.
print("Summary Statistics")
print("-----")
Sum= data_Set.sum(numeric_only=True)
print(f"Sum :{Sum}\n")

Mean= data_Set.mean(numeric_only=True)
print(f"Mean :{Mean}\n")

Standard_Deviation= data_Set.std(numeric_only=True)
print(f"Standard Deviation :{Standard_Deviation}\n")

Skewness= data_Set.skew(numeric_only=True)
print(f"Skewness :{Skewness}\n")

Kurtosis= data_Set.kurtosis(numeric_only=True)
print(f"Kurtosis :{Kurtosis}\n")
```

Figure 23: Python program to show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame

This python program generates a statistical summary of the given dataset by computing several metrics. The `sum()` function is used to calculate the total of all the numeric values in each columns, giving an cumulative view of the dataset's value. The `mean()` function is used to determine the central value i.e., the average value for each column. The `std()` function is used to measure the amount of dispersion from the mean, indication how far the data is from its central value. The `skew()` function is used to determine the asymmetry of data distribution. Lastly the `kurtosis()` function is used to understand the flatness or sharpness of the data curve.

Finally, the print statement prints the calculated statics summary providing the overview of the given dataset data.

**Output:**

```
Summary Statistics
-----
Sum :
Unique Key      9.112108e+12
Incident Zip    3.160833e+09
Latitude        1.185553e+07
Longitude       -2.152010e+07
dtype: float64

Mean :
Unique Key      3.130158e+07
Incident Zip    1.085798e+04
Latitude        4.072568e+01
Longitude       -7.392504e+01
dtype: float64

Standard Deviation :
Unique Key      575377.738707
Incident Zip    580.280774
Latitude        0.082411
Longitude       0.078654
dtype: float64

Skewness :
Unique Key      0.016898
Incident Zip    -2.553956
Latitude        0.123114
Longitude       -0.312739
dtype: float64

Kurtosis :
Unique Key     -1.176593
Incident Zip   37.827777
Latitude      -0.734818
Longitude      1.455600
dtype: float64
```

Figure 24 :Summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame

This output shows summary statistics for different columns in the dataset.

1. **Sum:** This represents the total sum of all values in each column. Here, the sum of 'Unique Key' is 9.11 trillion, and for 'Incident Zip', it's 3.16 billion. These numbers represent the total sum of all values in these columns.
2. **Mean:** It gives the average value of each column. the 'Unique Key' has an average of 31.3 million, and the 'Incident Zip' averages about 10,858. This gives us a sense of the central value in each column.
3. **Standard Deviation:** It is used to measure the amount of dispersion from the mean, indication how far the data is from its central value. Here, 'Latitude' and 'Longitude' have a low standard deviation, meaning their values are close to the average.
4. **Skewness:** It is used to determine the asymmetry of data distribution. Here, Latitude and Longitude have small skewness, meaning they are more evenly distributed.
5. **Kurtosis:** It shows how peaked or flat the distribution is compared to a normal distribution. Here, Incident Zip has a very high kurtosis (37.83), meaning it has a sharp peak.

### 3.2. Write a Python Program to Calculate and Show Correlation of all Variables

The python program to calculate correlation is given below.

#### Code to calculate the Correlation of all variables:

```
[41]: #calculating and showing correlation of all variables.
Correlation= data_Set.select_dtypes(include="number").corr()
print('      \n                      Correlation ')
print("-----")
Correlation
```

Figure 25: Python program to calculate and show correlation of all variables

This program calculates and displays correlations between numerical variables in a dataset to identify dependencies. It first filters numerical columns using `select_dtypes()` with `include='number'`, ensuring only numeric data (like integers and floats) are considered. The program then calculates the correlation matrix using the `corr()` function, which computes the pairwise correlations between all selected columns. Correlation coefficients range from -1 to 1. A value close to 1 indicates a strong positive correlation, -1 represents a strong negative correlation, and 0 signifies no correlation. This helps to identify relation between each column.

#### Output:

Correlation						
	Unique Key	Incident Zip	Latitude	Longitude	Request_Closing_Time	
Unique Key	1.000000	0.025492	-0.032613	-0.008621	0.053126	
Incident Zip	0.025492	1.000000	-0.499081	0.385934	0.057182	
Latitude	-0.032613	-0.499081	1.000000	0.368819	0.024497	
Longitude	-0.008621	0.385934	0.368819	1.000000	0.109724	
Request_Closing_Time	0.053126	0.057182	0.024497	0.109724	1.000000	

Figure 26: Correlation of all variables

Explaining the correlation value:

- 1. Unique Key and Incident Zip:** The value of correlation is 0.025492. This means very small positive correlation or almost no relationship between Unique Key and Incident Zip.

2. **Unique Key and Latitude:** The value of correlation is -0.032613. There is very small negative correlation between them.
3. **Unique Key and Longitude :** The value of correlation is -0.008621. There is very small negative correlation between them.
4. **Unique Key and Request\_Closing\_Time:** The value of correlation is 0.053126. There is very small positive correlation between them.
5. **Incident Zip and Latitude:** The value of correlation is -0.499081. There is moderate negative correlation. As Incident Zip increases, Latitude tends to decrease, and vice versa.
6. **Incident Zip and Longitude:** The value of correlation is 0.385934. There is Moderate positive correlation. As Incident Zip increases, Longitude tends to increase as well.
7. **Incident Zip and Request\_Closing\_Time:** The value of correlation is 0.057182. There is very small positive correlation between them.
8. **Latitude and Longitude:** The value of correlation is 0.368819. There is moderate positive correlation. As Latitude increases, Longitude tends to increase as well.
9. **Latitude and Request\_Closing\_Time:** The value of correlation is 0.024497. There is very small positive correlation between them.
10. **Longitude and Request\_Closing\_Time:** The value of correlation is 0.109724. There is small positive correlation. Longitude and Request\_Closing\_Time have a slight positive relationship.

## 4. Data Exploration

The initial step in data analysis where data analyst uses statistical techniques and data visualization to describe the data character, such as quantity, accuracy and size which helps to better understand the nature of the data is refer to as Data Exploration. It is done through both manual analysis and automated data exploration techniques (HEAVY.AI, 2025).

### 4.1. Provide four major Insights through Visualization that you Come up after Data Mining

The four major insights that comes up after Data Mining includes the following.

#### 4.1.1. Most Common complain types across New York City

After analyzing the NYC 311 Service Request dataset, it was found that “Blocked Driveway” was one of the most common complain made by the residents of New York City. This means that many people in New York City are facing problems where vehicles are blocking their driveways. There were more than 70000 complaints related to “Blocked Driveway. Other frequent complaint includes “Illegal Parking”.

To understand the analysis better, a bar graph was created showing the top 10 most common complaint types. This visual representation clearly shows which issues happen the most across the city. It indicates the need for better enforcement or parking regulation in certain areas.

**Code used to create the bar graph:**

```
[21]: import pandas as pd
import matplotlib.pyplot as plt

# Strip column names to remove any leading/trailing spaces
data_Set.columns = data_Set.columns.str.strip()

# Getting the top 10 most common complaint types
top_complaints = data_Set['Complaint Type'].value_counts().head(10)

# Plotting
plt.figure(figsize=(12, 6))
top_complaints.plot(kind='bar', color='coral')
plt.title('Top 10 Most Common Complaint Types in NYC 311 Dataset')
plt.xlabel('Complaint Type')
plt.ylabel('Number of Complaints')
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

Figure 27: Code Used to create the Bar Graph of top 10 Complaints in NYC 311

The screenshot provides the python code that is used understand what kinds of complaints people in New York City report the most through the 311 system. Firstly, the columns name was clean by removing the extra spaces. Then, it looks at the column called 'Complaint Type' and counts how many times each type of complaint was made. After that, it picks the top 10 complaint types that were reported the most.

It then uses the matplotlib library to create a bar chart that shows these top 10 complaints. the chart includes a title, labels, and tilted names at the bottom to make them easier to read. The `tight_layout()` makes sure nothing is cut off in the chart, and `show()` displays the chart.

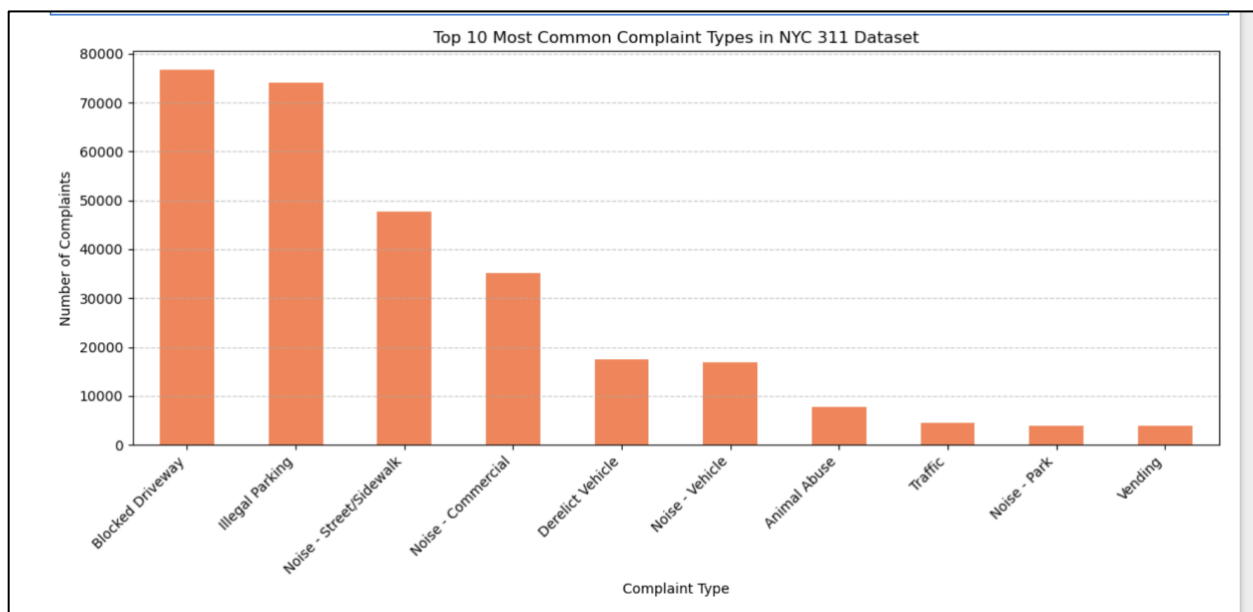


Figure 28: Screenshot of the Bar Graph Displaying the Top 10 Complaints in NYC 311

The above provided bar graph gives a clear and easy-to-understand picture of the most common issues people in NYC are facing. This can help the city respond better to people's problems. It is clearly seen that Blocked Driveways have the most number of complaints which is more than 70,000, followed by Illegal parking, Noise- street, Noise-commercial, Derelict Vehicle, Noise-Vehicle, Animal Abuse, Traffic, Noise- Park, Vending.

### 4.1.2. Number of Request Open and Closed

**Code used to find the number of open and closed requests:**

```
import pandas as pd

# Load the dataset
data = pd.read_csv("Customer Service_Requests_from_2010_to_Present.csv")

# Clean up column names
data.columns = data.columns.str.strip()

# Count how many complaints are Closed vs Open
status_counts = data['Status'].value_counts()
status_counts
```

*Figure 29: Code used to find the number of open and closed requests*

The provided screenshot above shows Python code which is used to understand the current status of service requests made by people in New York City. First, it loads the dataset containing all the complaints using the pandas library. After loading the data, it cleans the column names by removing any extra spaces. Then, it looks at the 'Status' column of the dataset, which tells whether each complaint is closed, open, assigned, or still in draft. Using the `value_counts()` function, it counts how many times each of these statuses appears in the dataset. This gives a clear idea of how many complaints have been resolved and how many are still waiting to get resolved.

#### Output

```
[41]: Status
      Closed    298471
      Open      1439
      Assigned    786
      Draft        2
      Name: count, dtype: int64
```

*Figure 30: Number of Request Closed and Open*

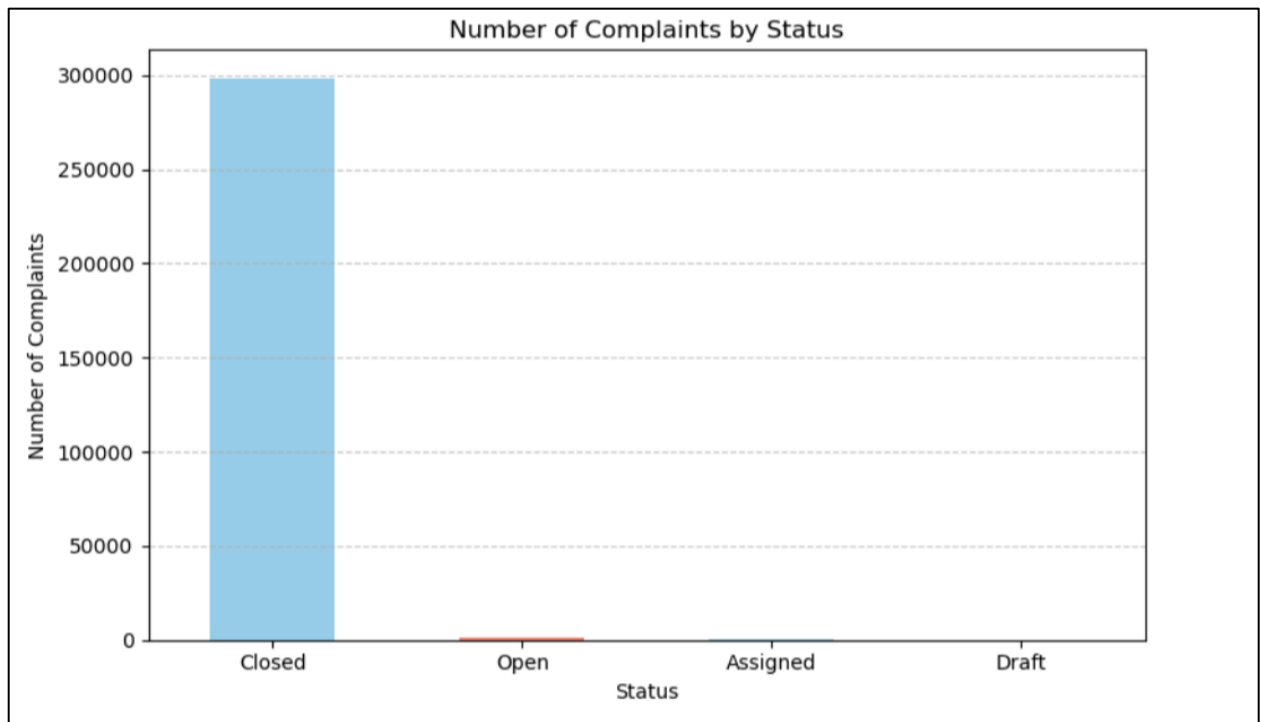
The data on the status of NYC 311 service requests gives us a clear picture of how complaints are handled. The majority of the complaints i.e., about 298,471, are marked as Closed, which means that the city has taken action and addressed these issues. This shows the 311 system is responding to most of the problems reported by the public.

On the other hand, there are 1,439 complaints still marked as Open, indicating that these issues have not yet been resolved. It shows that there are still some requests



waiting for attention. There are also 786 complaints listed as Assigned, meaning that they are in progress. Lastly, only 2 complaints are in Draft status. This likely means that those reports were started but not fully submitted.

**Bar graph showing the number of open and closed case.**



*Figure 31: Bar graph showing the number of open and closed case*

This bar graph shown above shows how many customer complaints are in each status. Most of the complaints are in the 'Closed' group, which means they have been solved. The 'Closed' bar is very tall, showing that almost all complaints are finished. Only a small number of complaints are still 'Open', meaning they are not solved yet. There are also very few complaints marked as 'Assigned' or 'Draft'. This means the complaint system is working well because most problems are being fixed, and only a few are left.

### 4.1.3. Agency handling the most Requests

**Code used to find the number of Agency handling the most requests:**



```
[43]: import pandas as pd

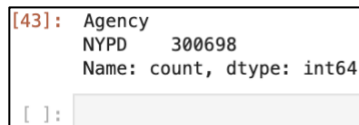
# Load the dataset
data = pd.read_csv("Customer Service Requests_from_2010_to_Present.csv")

# Count the number of requests handled by each agency
agency_counts = data['Agency'].value_counts()
agency_counts
```

*Figure 32: Code used to find the number of requests handled by each agency*

The provided screenshot above shows Python code which is used to find the number of requests handled by each agency. After loading the data, the code focuses on the 'Agency' column. It uses a function called `value_counts()` to count how many times each agency name shows up. This tells us how many complaints each agency got. The results are saved in a new variable called `agency_counts`, which stores the number of requests each agency handled. This helps us easily see which agencies were the busiest or had the most complaints.

#### Output:



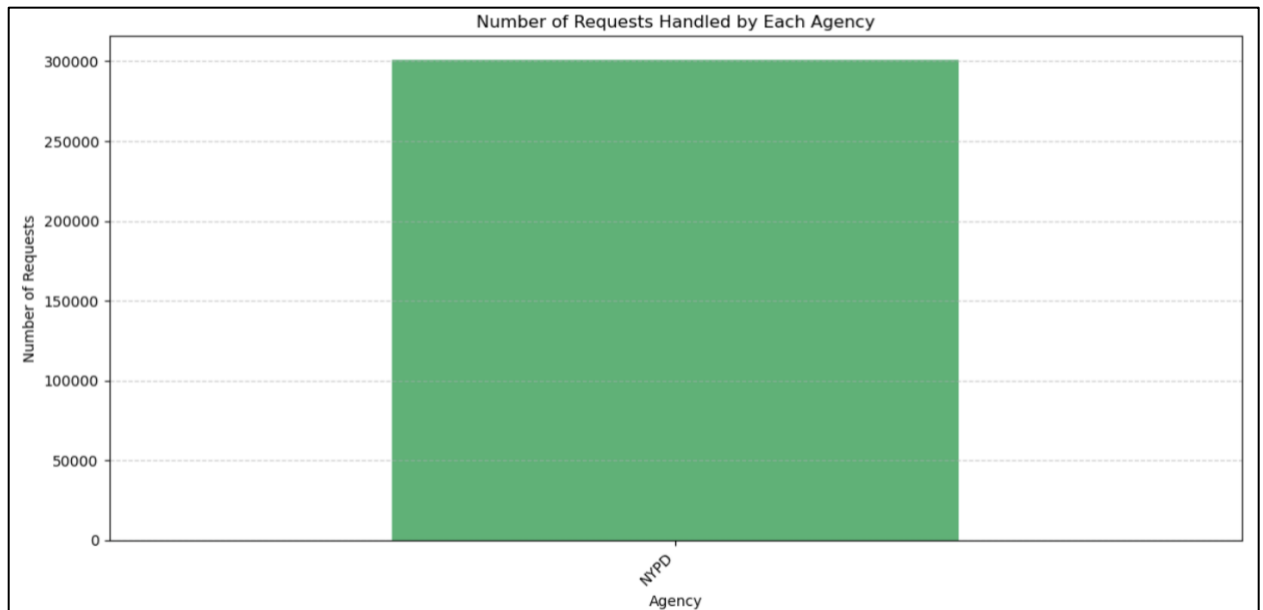
```
[43]: Agency
      NYPD    300698
      Name: count, dtype: int64

[ ]:
```

*Figure 33: The number of Agency handling the most requests*

This means that every one of the 300,698 complaints in the dataset was handled by the NYPD (New York Police Department). So, all the problems people reported like loud noise, blocked driveways, or other neighborhood issues were sent to the police. It also shows that this dataset only includes complaints that needed help from the police and does not have complaints managed by other city departments.

**Bar graph showing the agency that solve the most cases.**



*Figure 34: Bar graph showing the agency that solve the most cases*

This bar graph given above shows how many customer requests were handled by each agency. From the graph, it can be seen that only one agency, the NYPD (New York Police Department), handled all the requests. The tall green bar shows that NYPD took care of about 300,000 requests. There are no bars for other agencies, which means they either didn't get any requests or got very few. This tells us that NYPD was the main agency solving the complaints in this data.

#### 4.1.4. Complaint which takes the Longest Time to get Solved

##### Code:

```
# Find the row with the maximum time taken
longest_request = data_Set.loc[data_Set['Request_Closing_Time'].idxmax()]

# Display the full complaint details
print("Complaint that took the longest to resolve:\n")
print(longest_request)
```

Figure 35: Code used to find the complaint which takes the Longest Time to get Solved

The screenshot above shows the code that is used to find the complaint that took the longest time to resolve in a dataset. It identifies the row (i.e., the complaint) where this time is the longest by using the `idxmax()` function, which returns the index of the maximum value in the "Request\_Closing\_Time" column. Using this index, it retrieves the full details of that specific complaint using `loc`. Finally, it prints a message followed by the complete information of the complaint that had the longest resolution time.

##### Output

```
Complaint that took the longest to resolve:

Unique Key                30684975
Created Date              2015-05-23 23:51:59
Closed Date               2015-06-17 16:44:21
Agency                   NYPD
Complaint Type            Noise - Street/Sidewalk
Descriptor                Loud Music/Party
Location Type             Street/Sidewalk
Incident Zip              11238.0
City                     BROOKLYN
Status                    Closed
Resolution Description    The Police Department responded to the complai...
Borough                   BROOKLYN
Latitude                  40.68595
Longitude                 -73.959422
Request_Closing_Time      24 days 16:52:22
Name: 244488, dtype: object
```

Figure 36: Details of the Complaint which takes the Longest Time to get Solved

This output provides the complete details of the complaint that took the longest time to resolve in the dataset. The complaint, identified by the unique key 30684975, was created on May 23, 2015, at 11:51 PM and was closed on June 17, 2015, at 4:44 PM, taking a total of 24 days, 16 hours, 52 minutes, and 22 seconds to resolve. It was handled by the NYPD and categorized under "Noise - Street/Sidewalk" with a specific description of "Loud Music/Party." The incident

occurred on the street or sidewalk in Brooklyn, ZIP code 11238. The status of the complaint is marked as closed, and the resolution description indicates that the police responded to the complaint.

### Bar showing the Complaint which takes the longest time to get Solved

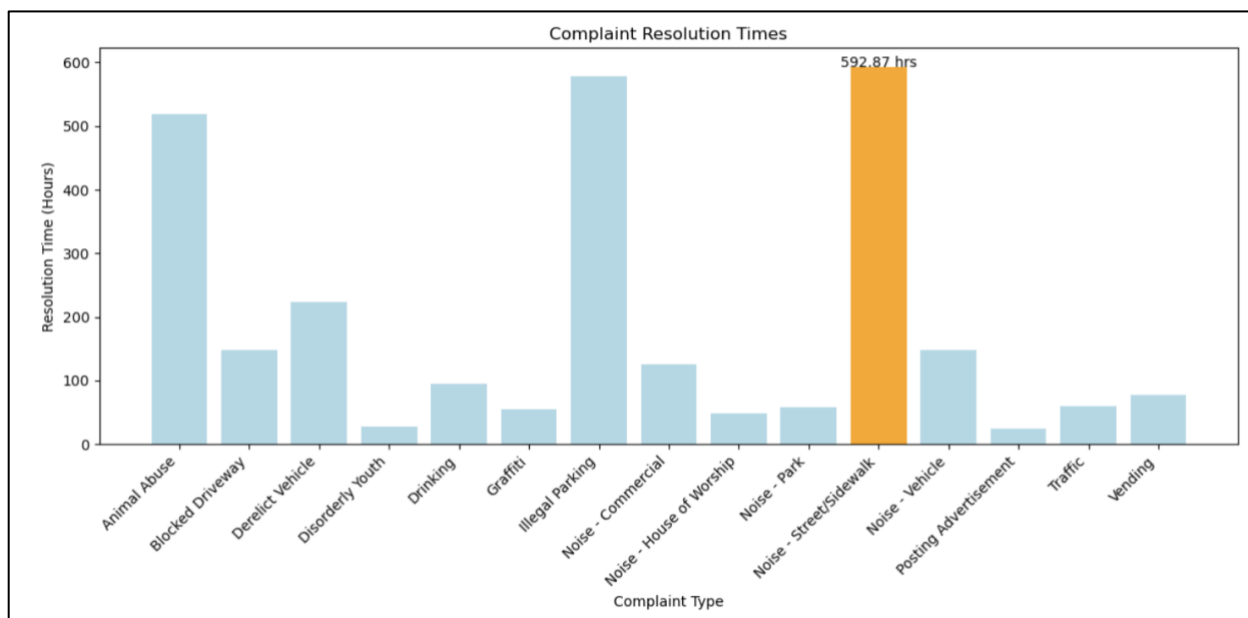


Figure 37: Bar showing the Complaint which takes the longest time to get Solved

This bar chart given above shows how long it takes to solve different types of complaints, using hours to measure the time. Each bar stands for one type of complaint, like animal abuse, parking problems, or loud noise. The taller the bar, the longer it takes to fix the problem. Most complaints are fixed in less than 300 hours. But one type of complaint takes much longer than all the others is the noise on the street or sidewalk. This one takes 592.87 hours which is about 593 hours to solve, which is the slowest of all. This means people have to wait a very long time for street noise problems to be fixed. Other complaints, like traffic or drinking issues, are fixed much faster and have shorter bars. So, this chart clearly shows that noise from the street takes the most time to fix.

## 4.2. Arrange the complaint types according to their average 'Request\_Closing\_Time', categorized by various locations. Illustrate it through graph as well

Code:

```
[57]: import matplotlib.pyplot as plt
import seaborn as sns

# Converting the 'Request_Closing_Time' to total hours and store in a new column
data_Set['Resolution_Hours'] = data_Set['Request_Closing_Time'].dt.total_seconds() / 3600

# Group by complaint type and location type to find average resolution time
average_resolution = data_Set.groupby(['Complaint Type', 'Location Type'])['Resolution_Hours'].mean().reset_index()

# Sorting the values by resolution time in descending order
average_resolution = average_resolution.sort_values(by='Resolution_Hours', ascending=False)

# Plot bar chart with Complaint Type on x-axis
plt.figure(figsize=(18, 8))
sns.barplot(
    data=average_resolution,
    x='Complaint Type',
    y='Resolution_Hours',
    hue='Location Type'
)

plt.title('Average Time Taken to Resolve Complaints\nby Complaint Type and Location')
plt.xlabel('Type of Complaint')
plt.ylabel('Average Resolution Time (in Hours)')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Location Category', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

Figure 38: Code used to arrange the complaint types according to their average 'Request\_Closing\_Time', categorized by various locations and illustrating it through graph as well

The screenshot of the code provided is used to to arrange the complaint types according to their average 'Request\_Closing\_Time', categorized by various locations and make a bar chart that shows how long different types of complaints take to get solved, based on where they happened.

Firstly, The code first changes the Request\_Closing\_Time into hours by calculating the total time in seconds and then dividing by 3600 (the number of seconds in one hour). After that, it groups the data based on the type of complaint and the location it happened. Then, it calculates the average time it took to solve each type of complaint in each location.

The data is then sorted to show which complaints took the longest time to resolve first. The reset\_index() function is used to tidy up the data and make it easier to read. The code then creates a bar chart using sns.barplot(). This chart shows

each complaint type on the x-axis and the average resolution time in hours on the y-axis. Different colors are used for different locations to make the chart clearer.

Finally, the chart is displayed with a title, labels for the axes, and a legend to explain the colors. The `plt.show()` function shows the chart on the screen so it can be seen and proper analysis of the data is performed.

### Output:

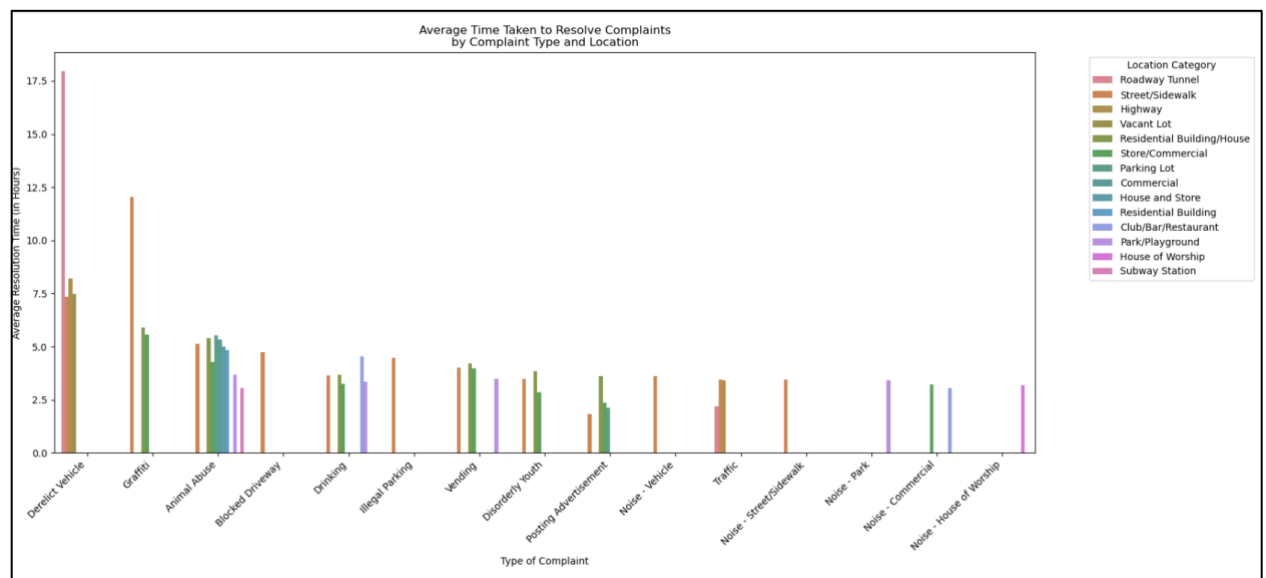


Figure 39: Graph showing the complaint types according to their average 'Request\_Closing\_Time', categorized by various locations

This output shows how long it takes to fix different types of complaints in different places, measured in hours. Complaints related to Derelict Vehicles tend to take the longest to resolve, particularly in locations like Roadway Tunnel and Street/Sidewalk. It took nearly 18 hours to remove, while graffiti on streets or sidewalks took about 12 hours that is, Graffiti are resolved more quickly, especially in Residential Building/House. Animal abuse cases had different times depending on the location. In parking lots or houses, it took over 5 hours, while cases in subway stations were fixed quicker, taking around 3 hours.

Complaints like blocked driveways, public drinking, and illegal parking on streets generally took about 4 to 5 hours to resolve. On the other hand, street vendors were handled more quickly, usually in about 4 hours or less. Noise-related complaints, such as loud music, cars, or parties, were resolved in about 3 to 3.5 hours. Traffic problems on highways took around 3.4 hours to fix, but in tunnels, they were cleared even faster, taking only about 2 hours. The fastest complaints to fix were those about illegal posters and ads. If someone put up ads on a street or sidewalk, it took less than 2 hours to remove them.

In summary, larger problems like complaints related to Derelict Vehicles take much longer to fix than smaller ones, like noise or ads. The location of the problem also plays a role, busy areas like streets tend to take longer to resolve than quieter spots.



## 5. Statistical Testing

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

This test is performed to test whether the average response time across different complaint types is similar or not. A test called One-Way ANOVA is used.

#### Step 1: Setting up the Hypotheses

**Null Hypothesis (H0):** The average response time is the same for all complaint types.

**Alternative Hypothesis(H1):** Some complaint types take more or less time than others to get response.

#### Step 2: Calculate the Group means

Calculating the average response time (Resolution\_Hours) for each complaint type.

#### Step 3: Run ANNOVA Test

The main idea behind ANOVA is to compare the variation in response times across different complaint types.

#### Code:

```
import pandas as pd
import scipy.stats as stats

# Calculate resolution time in hours (assuming Request_Closing_Time already exists)
data_Set['Resolution_Hours'] = data_Set['Request_Closing_Time'].dt.total_seconds() / 3600

# Get unique complaint types
complaint_types = data_Set['Complaint Type'].unique()

# Group Resolution_Hours by each complaint type
grouped_data = [data_Set[data_Set['Complaint Type'] == complaint]['Resolution_Hours'] for complaint in complaint_types]

# Perform One-Way ANOVA
f_statistic, p_value = stats.f_oneway(*grouped_data)

# Print the results
print(f"F-statistic: {f_statistic:.4f}")
print(f"P-value: {p_value:.4f}")

# Interpretation
alpha = 0.05
if p_value < alpha:
    print("Result: Different complaint types have significantly different average resolution times.")
else:
    print("Result: No significant difference in average resolution times across complaint types.")
```

Figure 40: Code showing ANNOVA Test

This code checks if different types of complaints take different amounts of time to fix. First, it calculates how long each complaint took to resolve (Resolution\_Hours) by converting the closing time into hours. Then, it lists all the complaint types in the data and puts the resolution times into groups based on each type. After that, it uses a statistical method called One-Way ANOVA. Using a statistical method called ANOVA, the code compares the average resolution times across all complaint categories. The test produces two important results: an F-statistic that measures how much the groups differ from each other, and a p-value that indicates whether these differences are statistically significant.

The p-value tells, if the difference in average resolution times is real or just random. If the p-value is less than 0.05, we say the difference is meaningful and reject the null hypothesis, meaning not all complaint types take the same amount of time. If the p-value is more than 0.05, the difference is likely just random, so we say all complaint types take about the same time to fix.

### **Output**

```
F-statistic: 578.9120  
P-value: 0.0000  
Result: Different complaint types have significantly different average resolution times.
```

*Figure 41: Result from the Test*

This output shows that the ANOVA test found a clear difference in how long it takes to fix different types of complaints. The F-statistic (578.9120) is a high number, which means there are big differences between the complaint types. The p-value (0.0000) is much smaller than 0.05, which tells us that these differences are real and not just due to chance. So, the null hypothesis was rejected and it was concluded that different complaint types take significantly different amounts of time to resolve.

### **Result:**

Hence, there are significant differences in average response times between complaint types.

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

This test is performed to test whether the type of complaint or service requested and location are related. A test called Chi-square test is used.

### Step 1: Setting up the Hypotheses

**Null Hypothesis (H0):** There is no relationship between the location and type of complaint.

**Alternative Hypothesis(H1):** There is relation between the location and the type of complaint.

### Step 2: Statistical Test Used: Chi-Square Test

The main idea behind Chi-Square is to compare and check the relation between the type of complaint and the location.

#### Code:

```
import pandas as pd
import scipy.stats as stats

# Create contingency table
contingency_table = pd.crosstab(data['Complaint Type'], data_Set['Borough'])

# Performing Chi-Square Test of Independence
chi2_stat, p_val, dof, expected = stats.chi2_contingency(contingency_table)

# Displaying the result
print(f"Chi-Square Statistic: {chi2_stat:.4f}")
print(f"P-value: {p_val:.4f}")
print(f"Degrees of Freedom: {dof}")

# Interpretation
alpha = 0.05
if p_val < alpha:
    print("Result: Complaint type and location are related.")
else:
    print("Result: Complaint type and location are not significantly related.")
```

Figure 42: Code showing Chi-Square Test

This code checks whether the type of complaint people make is connected to the location they are in. It first creates a contingency table, which is a count of how many complaints of each type were made in each borough. This table helps to compare the number of complaints for every combination of complaint type and borough.

Then, the code runs a Chi-Square Test of Independence, which is a statistical test used to determine if two categories, i.e., complaint type and location, are related or not. The test provides three important outputs: the Chi-Square statistic, which tells how different the observed data is from the expected data, the p-value which helps decide if the result is significant, and the degrees of freedom related to the size of the table. Finally, the code checks the p-value.

If the p-value is less than 0.05, it means the result is statistically significant, so the null hypothesis is rejected and is concluded that the type of complaint and location are related. If the p-value is greater than 0.05, null hypothesis was fail to reject, meaning there's no strong evidence that the type of complaint is related to the location. In simple terms, this helps us understand if the patterns of complaints differ across different areas.

**Output:**

```
Chi-Square Statistic: 73264.6216  
P-value: 0.0000  
Degrees of Freedom: 56  
Result: Complaint type and location are related.
```

*Figure 43: Result from the Test*

The output shows a strong relation between the type of complaint and the location (borough). The Chi-Square Statistic value of 73264.6216 tells us that the actual data, i.e., how complaints are spread across boroughs is very different from the expected data if there were no relationship between complaint type and location. The p-value of 0.0000 is much smaller than 0.05, which means this difference is significant and not just by chance. The Degrees of Freedom (56) is part of the calculation but doesn't affect the result. So, it was concluded the complaints are related to the location.

**Result:**

Hence, the type of complaint and the location are clearly connected.

## 6. Conclusion

This project was all about studying the NYC 311 service request data, and it was a great learning experience. It focused on analyzing the NYC 311 service request data, which contains information about the different problems reported by people living in New York City. The main purpose of the project was to understand what kinds of complaints residents usually make and how the city responds to those complaints. Python tools like pandas and matplotlib was used to work with the data step by step.

In the beginning, the data was needed to be cleaned. This included checking for missing information, removing repeated entries, and making sure all the data was in the correct format. Cleaning the data was important because it helped make sure that all the analysis done later would be accurate and reliable. After the cleaning process, the data was explored to find useful information. The most common types of complaints were identified, such as blocked driveways and noise problems, which were reported very frequently. It was also found that every complaint in the dataset was handled by the NYPD, meaning the police department was responsible for solving these issues. The project also looked at how many complaints had been resolved and how many were still open. Most complaints were already closed, which shows that the city responds to issues quickly and efficiently. Graphs like bar charts and pie charts were used to show all these findings in a simple and clear way. These visual tools made it easier to see patterns and understand the data better.

In conclusion, this project teaches how to use data to understand and solve real-life problems. It helps students learn how to collect data, clean it, and find useful information from it. This project focuses on using tools like Python to look at data in a smart way. Students also learn how to make charts and graphs to show their findings clearly. The goal is to use data to help people make better decisions, whether it's for a city, a business, or any other place. It's a very practical project that shows how data can be used to make things work better in the real world.

## 7. References

3Pillar Global, I., 2025. *Data Analytics & Insights*. [Online]

Available at: <https://www.3pillarglobal.com/services/data-analytics/data-analytics-services/>

[Accessed 10 April 2025].

Stedman, C., 2024. *What is data preparation? An in-depth guide*. [Online]

Available at: <https://www.techtarget.com/searchbusinessanalytics/definition/data-preparation>

[Accessed 11 April 2025].

HEAVY.AI, 2025. *What is Data Exploration?*. [Online]

Available at: <https://www.heavy.ai/learn/data-exploration#:~:text=What%20is%20Data%20Exploration%3F,the%20nature%20of%20the%20data.>

[Accessed 3 May 2025].

Salatino, F., 2024. Data Exploration: aDefinition, Importance & How It Differs from Data Analytics (CData Software). [Online] Available at: <https://www.cdata.com/blog/what-is-data-exploration> [Accessed 7 May 2025].

S, S., 2020. What Is Pandas in Python? Everything You Need to Know - ActiveState. [Online]

Available at: <https://www.activestate.com/resources/quick-reads/what-is-pandas-in-python-everything-you-need-to-know/>

[Accessed 9 May 2025].

Suer, M., 2023. What Is Data Quality and Why Is It Important? | Alation. [Online]

Available at: <https://www.alation.com/blog/what-is-data-quality-why-is-it-important/>

[Accessed 9 May 2025].

