LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CC5061NI Applied Data Science

60% Individual Coursework
Submission : Final

Academic Semester: Spring Semester 2025

Credit: 15 credit semester long module

Student Name: Aarya Paudel

London Met ID: 23048670

College ID: np01ai4a230109

Assignment Due Date: Thursday, May 15, 2025

Assignment Submission Date: Wednesday, May 14, 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.

23048670_Aarya_Paudel3.docx

Islington College,Nepal

Document Details

Submission ID
trn:oid:::3618:95844350

Submission Date
May 14, 2025, 4:08 PM GMT+5:45

Download Date
May 14, 2025, 4:09 PM GMT+5:45

File Name
23048670_Aarya_Paudel3.docx

File Size
16.9 KB

21 Pages

2,539 Words

13,807 Characters

17% Overall Similarity

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

Match Groups

- **31** Not Cited or Quoted 17%
  Matches with neither in-text citation nor quotation marks
- **0** Missing Quotations 0%
  Matches that are still very similar to source material
- **0** 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

- 3% Internet sources
- 1% Publications
- 15% Submitted works (Student Papers)

Table of Contents

Table of Figures

Table of Tables

1. Introduction

This course on Applied Data Science goes through working with an actual dataset that involves customer service requests logged in New York's 311 system. This system provides facilities for complaints by citizens of a city on non-emergency issues like noise pollution, sanitation matters, illegal parking, or problems relating to street infrastructure. Once again, this project gives a golden opportunity to an aspiring data scientist to delve deep into and apply core methodologies from importing, cleaning raw data to finding useful patterns and relationships.

The tools that will be required to accomplish this course are Python programming language, Jupyter Notebook, and libraries that, by default, are quite function-rich like pandas, NumPy, matplotlib, and seaborn. These resources are used to solve tasks at each step of the data science workflow. These tasks are divided into several milestones like knowing the structure of the data set, data cleaning, and basic analysis—all very critical steps before any prediction modelling or insight generation can be started.

In particular, the data science lifecycle is as below:

1. Data Collection & Ingestion: Data is collected from the available sources.
2. Data Pre-processing: It involves handling of missing values, inconsistency, and gives the data an appropriate shape.
3. Exploratory Data Analysis: This activity involves visualizations and some descriptive statistics for finding trends, distributions, and outliers.
4. Analysis & Modelling: Statistical or machine learning models are applied to predict or classify.
5. Reporting: It is the process of putting results in a dashboard, reporting form, or through visualizations.

This project is in the very early stages of that lifecycle—where groundwork is laid in understanding the contents of a dataset and ensuring that it is clean and wellprepared. Consequently, this coursework emphasizes data wrangling and quality inspection ahead of data-driven decisions or algorithms in practical environments.

2. Data understanding

This dataset provides users with a deep view of the 311 complaint records in New York City, reflecting a great number of types of service requests from the residents. Basically, the 311 system is a major platform where the citizens register all sorts of non-emergent concerns, which are consequently routed to the responsible agencies in the city, including NYPD, DOT, and DSNY. Therefore, these complaints regard different topics related to noise pollution, problems of sanitation, illegal parking, and street infrastructures maintenance; in other words, the complaints give a microscopic view on the multi-dimensionality of urban life and the requisite efficiency in management of service requests.

Each row in the dataset represents an individual service request that captures all relevant information regarding the complaint and how it was dealt with. Key fields will include the type of complaint, the description of the incident, location, and the timestamps of when it was created and closed. In fact, there are metadata fields, such as borough or responsible agency, which relate to the context of complaints' analysis.

The first step in getting this data in order is the data understanding stage. Its first benefit is to help one understand data structure, content, and quality in an observed set of data. The initial exploration helps to identify the most relevant features that will be useful for further analysis. For example, monitoring service response times will give insight into agency effectiveness, while frequently occurring complaint types would show the problems residents are actually facing. Locating complaint trends across places can also help narrow down areas needing further resources or attention.

At the same time, it is equally important to recognize any data quality issues at this stage. Observing that many columns have missing key values or are of no relevance to the core analysis objectives. For example, school information, ferry terminals, bridge name—these are just a few examples of fields that are mostly missing since they do not apply to a majority of complaint types.

This is an important level of recognition for one to make the right decisions about cleaning and preparing data without which further analysis may go wrong with incomplete data or having wrong datasets.

The dataset itself comes in structured CSV format and contains slightly more than 300,000 records. This scale of data can allow hundreds of data science techniques to be applied against it in the quest to fish out respective mixed insights on the complaints or the city's mechanisms in response to its citizens.

Specified Aspects of Data Understanding

Characteristics of Dataset: This dataset, with 300,698 rows and 53 columns, seems to be one rich, multidimensional source of data, which could give several research opportunities from a number of perspectives. The total number of values, slightly over 15 million, further demonstrates the size and detail of the dataset.

Column Analysis: The attributes within the data-set are complaint details ("Complaint Type," "Descriptor"), timestamps ("Created Date," "Closed Date"), location information ("Incident Address," "Borough"), agency information ("Agency," "Agency Name"), and resolution status ("Status"). Knowledge of the meaning and data type of each column is important in selecting suitable techniques of analysis.

Data Types: Column data types in the dataset will have some with numerical data, such as "Unique Key" and "Incident Zip," while others might still remain categorical, such as "Complaint Type," "Borough," and even some text data like "Resolution Description." The different data types can be noted because they serve different functions when it comes to processing and analyzing data. For example, the date column needs to be in a datetime format for time-based analysis of the data.

Missing Values: Missing values in a few columns, typically to do with schools, landmarks, and taxi information, require due caution while handling in the data preprocessing. These missing values seem to come in from fields supporting information on schools, landmarks, and taxi information that do not apply to the complaints.

No. of Unique Values: This is going to help me understand the data variability by giving me the count of unique values in each of the columns. For instance, it will show the number of categories each of the complaints forms.

Initial Insights: An initial review of the data has posed several questions on which a series of analyses would be based. Identifying the most common complaint types, examining the trend of complaints in the respect of different boroughs, calculating the time taken to resolve complaints, and examining the relationship between complaint types and locations can then go on.

Data Description

S.No	Column Name	Data Type	Description
1	Unique Key	Integer	Unique identifier for each complaint or request.
2	Created Data	Date	Timestamp of when the complaint was filed.
3	Closed Data	Date	Timestamp of when the complaint was closed or resolved.
4	Agency	Object	Code for the agency that handled the complaint.
5	Agency Name	Object	Full name of the agency.
6	Complaint Type	Object	Broad category of the issue.
7	Descriptor	Object	Detailed subcategory or specific nature of the complaint.
8	Location Type	Object	Type of location.
9	Incident Zip	Float	ZIP code of where the issue is occurred.
10	Incident Address	Object	Full street address of the incident.
11	Street Name	Object	Name of the street involved.
12	Cross Street 1	Object	First nearby cross street.
13	Cross Street 2	Object	Second nearby cross street.
14	Intersection Street 1	Object	First street on an intersection.
15	Intersection Street 2	Object	Second street on an intersection.
16	Address Type	Object	Indicates whether the location is an address, intersection, or landmark.
17	City	Object	City where the complaint occurred.

18	Landmark	Object	Landmark reference if available.
19	Facility Type	Object	Type of facility involved.

20	Status	Object	Current status of the request.
21	Due date	Date	Date by which the complaint was expected to be resolved.
22	Resolution description	Object	Textual representation of how the request was resolved.
23	Resolution Action Updated Date	Date	Most recent update to the resolution process.
24	Community Board	Object	NYC community board responsible for handling the request.
25	Borough	Object	Borough in NYC where the incident occurred (e.g., Bronx, Queens).
26	X Coordinate (State Plane)	Float	X-coordinate for mapping on a grid.
27	Y Coordinate (State Plane)	Float	Y-coordinate for mapping on a grid.
28	Park Facility Name	Object	Name of a park facility if the complaint is related to a public park.
29	Park Borough	Object	Borough responsible for managing the associated park facility.
30	School Name	Object	Name of the school associated with the complaint (if applicable).

31	School Number	Object	School ID number for identifying the institution.
32	School Region	Object	Education region associated with the school.
33	School Code	Object	Internal code representing the school.
34	School Phone Number	Object	Official contact number of the school.
35	School Address	Object	Address of the school.
36	School City	Object	City in which the school is located.
37	School State	Object	State in which the school is located (typically NY).
38	School Zip	Object	ZIP/postal code of the school.
39	School Not Found	Object	Indicates if the school could not be matched/found in official records.
40	School or Citywide Complaint	Float	Binary flag to indicate if the complaint affects the whole city or school.
41	Vehicle Type	Object	Type of vehicle (if complaint involves one)
42	Taxi company borough	Object	Borough of the taxi company
43	Taxi pick – up location	Float	Location of taxi pickup
44	Bridge Highway Name	Object	Name of Bridge/ Highway involved
45	Bridge Highway Direction	Object	Direction of Bridge/ Highway

46	Road Ramp	Object	Ramp name associated with bridge/ highway
47	Bridge highway segment	Object	Segment identifier
48	Garage Lot Name	Object	Name of garage/ lot if complaint occurred there
49	Ferry Direction	Object	Ferry direction (to/from)
50	Ferry Terminal Name	Object	Ferry terminal name
51	Latitude	Float	Latitude coordinate
52	Longitude	Float	Longitude coordinate
53	Location	Object	Geospatial coordinates (lat/lon format)

Table 1: Data Description

3. Data preparation

3.1 Import the dataset

```
import pandas as pd

df = pd.read_csv("dataset.csv", low_memory = False)
print(" Dataset Loaded Successfully!")
print("Shape of the dataset:", df.shape)
print(df.head(3))
```

Figure 1: Importing the dataset (code)

Explanation:

- The pandas library is imported for data handling.
- The dataset dataset.csv is loaded into a DataFrame called df.
- It's better to use low_memory=False if you come across datatype guessing problems.
- It shows a success message, along with the shape of the dataset(rows × columns) and the first 3 rows as a preview.

```
 Dataset Loaded Successfully!
Shape of the dataset: (300698, 53)
   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

                      Agency Name           Complaint Type        Descriptor  \
0  New York City Police Department  Noise - Street/Sidewalk  Loud Music/Party
1  New York City Police Department         Blocked Driveway         No Access
2  New York City Police Department         Blocked Driveway         No Access

     Location Type  Incident Zip       Incident Address  ...  \
0  Street/Sidewalk       10034.0    71 VERMILYEA AVENUE  ...
1  Street/Sidewalk       11105.0        27-07 23 AVENUE  ...
2  Street/Sidewalk       10458.0  2897 VALENTINE AVENUE  ...

  Bridge Highway Name Bridge Highway Direction Road Ramp  \
0                 NaN                      NaN       NaN
1                 NaN                      NaN       NaN
2                 NaN                      NaN       NaN

  Bridge Highway Segment Garage Lot Name Ferry Direction Ferry Terminal Name  \
0                    NaN             NaN             NaN                 NaN
1                    NaN             NaN             NaN                 NaN
2                    NaN             NaN             NaN                 NaN

    Latitude  Longitude                                  Location
0  40.865682 -73.923501   (40.86568153633767, -73.92350095571744)
1  40.775945 -73.915094  (40.775945312321085, -73.91509393898605)
2  40.870325 -73.888525  (40.870324522111424, -73.88852464418646)

[3 rows x 53 columns]
```

Figure 2: Importing the dataset (output)

Output Summary:

- Confirmed and successfully loaded.
- The dataset has 300,698 rows and 53 columns.
- Preview of complaint records with date, agency, complaint type, and location information.

3.2 Dataset information and details

3.2.1 Shape

The dataset contains over 300,000 rows and nearly 50 columns, indicating a rich, multidimensional data source.

```
#  1. Shape of the dataset
print("Shape (rows, columns):", df.shape)
```

```
Shape (rows, columns): (300698, 53)
```

Figure 3: Dataset shape

This line of code will display the shape of the dataset in terms of the total number of rows and columns in the dataframe 'df.' It helps in finding the total number of rows and columns in the dataset in a go, which further allows a quick idea about how large or small the dataset is.

3.2.2 Total Values

There are over 15 million individual data points (cells), which makes the dataset quite large and detailed.

```
# 2. Total number of values (cells)
print("Total values in dataset:", df.size)
```

```
Total values in dataset: 15936994
```

Figure 4: Total values in dataset

This line shows the total number of data values in the whole dataset by estimating the number of rows multiplied by the number of columns, hence providing an overall measure of how large the dataset is.

3.2.3 Columns

```
# 3. Column names
print("Columns:")
print(df.columns.tolist())
```

Figure 5: Dataset columns (code)

Explanation:

- This code prints all the column names from the dataset since .columns.tolist() will convert a DataFrame's column index to a list in Python.
- It allows the user to understand what data are at their disposal, and reference for separate fields in later analysis.

```
Columns:
['Unique Key', 'Created Date', 'Closed Date', 'Agency', 'Agency Name', 'Complaint Type', 'Descriptor', 'Location Type', 'Incident Zip', 'In
cident Address', 'Street Name', 'Cross Street 1', 'Cross Street 2', 'Intersection Street 1', 'Intersection Street 2', 'Address Type', 'Cit
y', 'Landmark', 'Facility Type', 'Status', 'Due Date', 'Resolution Description', 'Resolution Action Updated Date', 'Community Board', 'Boro
ugh', 'X Coordinate (State Plane)', 'Y Coordinate (State Plane)', 'Park Facility Name', 'Park Borough', 'School Name', 'School Number', 'Sc
hool Region', 'School Code', 'School Phone Number', 'School Address', 'School City', 'School State', 'School Zip', 'School Not Found', 'Sch
ool or Citywide Complaint', 'Vehicle Type', 'Taxi Company Borough', 'Taxi Pick Up Location', 'Bridge Highway Name', 'Bridge Highway Directi
on', 'Road Ramp', 'Bridge Highway Segment', 'Garage Lot Name', 'Ferry Direction', 'Ferry Terminal Name', 'Latitude', 'Longitude', 'Locatio
n']
```

Figure 6: Dataset columns (output)

Output Summary:

- The user sees 53 column names which are key fields like 'Created Date', 'Complaint Type', 'Borough', 'Closed Date,' and many others.
- Through this step, you basically get a road map of what features (variables) you can explore, clean, or use for analysis.

3.2.4 Data Types

Displays the type of data stored in each column (e.g., object, int, float) to understand how data is represented.

```
# 4. Data types of each column
print("\nData types:")
print(df.dtypes)
```

Figure 7: Datatype of each column (code)

Explanation:

- It employs df.dtypes, which returns the datatypes of the columns returned in the DataFrame. This includes integer, float, and object.
- Data types are an important part of the data to know because only the correct operations can be performed based on them; for instance, datetime fields require a conversion, and object fields may require encoding while modeling.

```
Data types:
Unique Key                        int64
Created Date                     object
Closed Date                      object
Agency                           object
Agency Name                      object
Complaint Type                   object
Descriptor                       object
Location Type                    object
Incident Zip                    float64
Incident Address                 object
Street Name                      object
Cross Street 1                   object
Cross Street 2                   object
Intersection Street 1            object
Intersection Street 2            object
Address Type                     object
City                             object
Landmark                         object
Facility Type                    object
Status                           object
Due Date                         object
Resolution Description           object
Resolution Action Updated Date   object
Community Board                  object
Borough                          object
X Coordinate (State Plane)      float64
Y Coordinate (State Plane)      float64
Park Facility Name               object
Park Borough                     object
School Name                      object
School Number                    object
School Region                    object
School Code                      object
School Phone Number              object
School Address                   object
School City                      object
School State                     object
School Zip                       object
School Not Found                 object
School or Citywide Complaint    float64
Vehicle Type                    float64
Taxi Company Borough            float64
Taxi Pick Up Location           float64
Bridge Highway Name              object
Bridge Highway Direction         object
Road Ramp                        object
Bridge Highway Segment           object
Garage Lot Name                 float64
Ferry Direction                  object
Ferry Terminal Name              object
Latitude                        float64
Longitude                       float64
Location                         object
dtype: object
```

Figure 8: Dataset type of each column (output)

Output Summary:

- Types such as int64 will be detected for the numeric IDs, float64 for the coordinates, and object for the text.
- 'Created Date' and 'Closed Date' are still in object format; hence, it points towards further conversion to datatime.
- Many fields, including 'School or Citywide Complaints' and 'Taxi Pick Up Location,' are float64; however, they may represent categorical or null-filled data.

3.2.5 First Records

```
# 5. First 5 records
print("\nFirst 5 rows:")
print(df.head())
```

Figure 9: First 5 records (code)

Explanation:

- Visual inspection of a few sample records in the dataset can be done by loading the first five entries of the dataset using df.head().
- The major benefit it offers to users is the ability to verify the structure, content, and possibly available data quality issues such as missing fields.

```
First 5 rows:
   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

                      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

                    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

        Incident Address  ... Bridge Highway Name Bridge Highway Direction  \
0    71 VERMILYEA AVENUE  ...                 NaN                       NaN
1     27-07 23 AVENUE     ...                 NaN                       NaN
2  2897 VALENTINE AVENUE  ...                 NaN                       NaN
3    2940 BAISLEY AVENUE  ...                 NaN                       NaN
4         87-14 57 ROAD   ...                 NaN                       NaN

  Road Ramp Bridge Highway Segment Garage Lot Name Ferry Direction  \
0       NaN                   NaN             NaN             NaN
1       NaN                   NaN             NaN             NaN
2       NaN                   NaN             NaN             NaN
3       NaN                   NaN             NaN             NaN
4       NaN                   NaN             NaN             NaN

  Ferry Terminal Name   Latitude  Longitude  \
0                 NaN  40.865682 -73.923501
1                 NaN  40.775945 -73.915094
2                 NaN  40.870325 -73.888525
3                 NaN  40.835994 -73.828379
4                 NaN  40.733060 -73.874170

                                Location
0   (40.86568153633767, -73.92350095571744)
1  (40.775945312321085, -73.91509393898605)
2  (40.870324522111424, -73.88852464418646)
3   (40.83599404683083, -73.82837939584206)
4  (40.733059618956815, -73.87416975810375)

[5 rows x 53 columns]
```

Figure 10: First 5 records (output)

Output Summary:

- The first five complaints are about the complained type, agency, created date, and partial address.
- Most complaints are from the NYPD. Types include Street/Sidewalk - Noise, Blocked Driveway, Illegal Parking.
- Many columns associated with location have NaN values, probably indicating many complaints inappropriately or inadequately described.

3.2.6 Last Records

```
# 6. Last 5 records
print("\nLast 5 rows:")
print(df.tail())
```

Figure 11: Last 5 records (code)

Explanation:

- This command will display the final five records of a dataset through the df.tail() method and will show a snapshot of the end of the dataset.
- This would be useful for verification of the following: that all rows are consistent and that there are no missing data problems with the most recent entries in the file.

```
Last 5 rows:
        Unique Key            Created Date             Closed Date Agency  \
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        Descriptor  \
300693  New York City Police Department  Noise - Commercial  Loud Music/Party
300694  New York City Police Department    Blocked Driveway    Partial Access
300695  New York City Police Department  Noise - Commercial  Loud Music/Party
300696  New York City Police Department  Noise - Commercial  Loud Music/Party
300697  New York City Police Department  Noise - Commercial  Loud Music/Party

             Location Type  Incident Zip          Incident Address  ...  \
300693  Club/Bar/Restaurant           NaN           CRESCENT AVENUE  ...
300694      Street/Sidewalk       11418.0          100-17 87 AVENUE  ...
300695  Club/Bar/Restaurant       11206.0         162 THROOP AVENUE  ...
300696  Club/Bar/Restaurant       10461.0  3151 EAST TREMONT AVENUE  ...
300697     Store/Commercial       10036.0        251 WEST 48 STREET  ...

       Bridge Highway Name Bridge Highway Direction Road Ramp  \
300693                 NaN                      NaN       NaN
300694                 NaN                      NaN       NaN
300695                 NaN                      NaN       NaN
300696                 NaN                      NaN       NaN
300697                 NaN                      NaN       NaN

       Bridge Highway Segment Garage Lot Name Ferry Direction  \
300693                    NaN             NaN              NaN
300694                    NaN             NaN              NaN
300695                    NaN             NaN              NaN
300696                    NaN             NaN              NaN
300697                    NaN             NaN              NaN

       Ferry Terminal Name   Latitude  Longitude  \
300693                 NaN        NaN        NaN
300694                 NaN  40.694077 -73.846087
300695                 NaN  40.699590 -73.944234
300696                 NaN  40.837708 -73.834587
300697                 NaN  40.760583 -73.985922

                                    Location
300693                                   NaN
300694   (40.69407728322387, -73.8460866160573)
300695  (40.69959035300927, -73.94423377144169)
300696   (40.8377075854206, -73.83458731019586)
300697  (40.76058322950115, -73.98592204392392)

[5 rows x 53 columns]
```

Figure 12: Last 5 records (output)

Output Summary:

- It will have a similar structure to the first 5 rows, with complaint types such as "Noise - Commercial" and "Blocked Driveway."
- Some complaints have missing ZIP codes, NaN values for 'Closed Date,' and almost all location-based fields are missing (NaN), especially in the last record.
- This is an excellent illustration of how, in the dataset, missing data is very much prevalent.

3.2.6 Missing Values

```
# 7. Number of missing values in each column
print("\nMissing values per column:")
print(df.isnull().sum())
```

Figure 13: Missing values (code)

Explanation:

- For every column, this code leverages .isnull().sum() to expose how many missing values each one contains.
- This is one of the most important steps toward understanding and preparing data: one uses this information to decide which column should be cleaned, dropped, or imputed.

Output Summary:

- Thousands of missing values existed in many of the columns,
- notably:
    - ‘School or Citywide Complaint', 'Taxi Pick Up Location', and 'Garage Lot Name' had missing values in all 300,698 rows.
    - 2,164 records did not have 'Closed Date,' which probably means they are unresolved or open complaints.
    - Location-related fields like 'Incident Address', 'Street Name', 'Cross Street 1' are again very heavily missing.
- This output will help us identify which fields are irrelevant and can be dropped and which ones are essential and hence require careful handling.

```
Missing values per column:
Unique Key                          0
Created Date                        0
Closed Date                      2164
Agency                              0
Agency Name                         0
Complaint Type                      0
Descriptor                       5914
Location Type                     131
Incident Zip                     2615
Incident Address                44410
Street Name                     44410
Cross Street 1                  49279
Cross Street 2                  49779
Intersection Street 1          256840
Intersection Street 2          257336
Address Type                     2815
City                             2614
Landmark                       300349
Facility Type                    2171
Status                              0
Due Date                            3
Resolution Description              0
Resolution Action Updated Date   2187
Community Board                     0
Borough                             0
X Coordinate (State Plane)       3540
Y Coordinate (State Plane)       3540
Park Facility Name                  0
Park Borough                        0
School Name                         0
School Number                       0
School Region                       1
School Code                         1
School Phone Number                 0
School Address                      0
School City                         0
School State                        0
School Zip                          1
School Not Found                    0
School or Citywide Complaint   300698
Vehicle Type                   300698
Taxi Company Borough           300698
Taxi Pick Up Location          300698
Bridge Highway Name            300455
Bridge Highway Direction       300455
Road Ramp                      300485
Bridge Highway Segment         300485
Garage Lot Name                300698
Ferry Direction                300697
Ferry Terminal Name            300696
Latitude                         3540
Longitude                        3540
Location                         3540
dtype: int64
```

Figure 14: Missing values (output)

3.2.8 Unique Values

```
# 8. Count of unique values in each column
print("\nUnique value counts:")
print(df.nunique())
```

Figure 15: Unique values (code)

Explanation:

The code above counts how many distinct values are happening in a column. This means that it counts the number of unique values in every column of a dataframe. With `.nunique()`, we calculate: - Which
columns can we consider as categorical, constant, or near zero variance; or those that will affect the way we clean, visualize, or model our data.

Output Summary:

- 'Unique Key' has 300698 unique values, verifying that every complaint is unique.
- 'Complaint Type' has 24 unique categories, which represent diverse types of service requests.
- Some of the columns, like 'School Region', 'School Zip', and 'School Not Found', have only one unique value or zero, which means that they are constant or entirely missing—and thus probable candidates for dropping.
- 'Latitude' and 'Longitude' have more than 125,000 unique entries. This should be expected since complaints emanate from a myriad of locations within NYC.

Output helps in planning further data cleaning by removing uninformative or vacant columns that are to be discarded.

```
Unique value counts:
Unique Key                          300698
Created Date                        259493
Closed Date                         237165
Agency                                   1
Agency Name                              3
Complaint Type                          24
Descriptor                              45
Location Type                           18
Incident Zip                           201
Incident Address                    107652
Street Name                           7320
Cross Street 1                        5982
Cross Street 2                        5823
Intersection Street 1                 4413
Intersection Street 2                 4172
Address Type                             5
City                                    53
Landmark                               116
Facility Type                            1
Status                                   4
Due Date                            259851
Resolution Description                  18
Resolution Action Updated Date      237895
Community Board                         75
Borough                                  6
X Coordinate (State Plane)           63226
Y Coordinate (State Plane)           73694
Park Facility Name                       2
Park Borough                             6
School Name                              2
School Number                            2
School Region                            1
School Code                              1
School Phone Number                      2
School Address                           2
School City                              2
School State                             2
School Zip                               1
School Not Found                         1
School or Citywide Complaint             0
Vehicle Type                             0
Taxi Company Borough                     0
Taxi Pick Up Location                    0
Bridge Highway Name                     29
Bridge Highway Direction                34
Road Ramp                                2
Bridge Highway Segment                 160
Garage Lot Name                          0
Ferry Direction                          1
Ferry Terminal Name                      2
Latitude                            125122
Longitude                           125216
Location                            126048
```

Figure 16: Unique values (output)

3.2.9 Memory Usage

```
# 9. Memory usage
print("\nMemory usage:")
print(df.memory_usage(deep=True))
```

Figure 17: Memory Usage (code)

Explanation:

This code will help you gauge how much memory each column is consuming. It is useful in performance optimization, particularly with very large datasets. The deep=True argument ensures that it captures memory used by objects and not just numerical data.

Output Summary:

- For instance, columns such as 'Resolution Description', 'Agency Name', and 'Complaint Type' have huge memory footprints, primarily influenced by vast long text strings. For example, in isolation, `Resolution Description` consumes well over 46MB.
- Some, like `Taxi Company Borough` or `Garage Lot Name`, still show up in the readout of memory—people probably still have them in the original whole dataset, but they could be dropped in a copy.
- The output is providing a reason for the removal of irrelevant columns, specifically those having high memory usage with no added value.

```
Memory usage:
Index                              132
Unique Key                     2405584
Created Date                  20379557
Closed Date                   20290339
Agency                        15936994
Agency Name                   24055738
Complaint Type                19759386
Descriptor                    19400151
Location Type                 19405927
Incident Zip                   2405584
Incident Address              18415331
Street Name                   17263690
Cross Street 1                17010126
Cross Street 2                16992671
Intersection Street 1         10952654
Intersection Street 2         10911474
Address Type                  17013086
City                          17188572
Landmark                       9633507
Facility Type                 17085511
Status                        16537082
Due Date                      20353161
Resolution Description        46389144
Resolution Action Updated Date    20289360
Community Board               17891729
Borough                       16991966
X Coordinate (State Plane)     2405584
Y Coordinate (State Plane)     2405584
Park Facility Name            18041900
Park Borough                  16991966
School Name                   18041900
School Number                 18041873
School Region                 18041852
School Code                   18041852
School Phone Number           18041879
School Address                18041913
School City                   18041875
School State                  18041871
School Zip                    18041852
School Not Found              15034900
School or Citywide Complaint   2405584
Vehicle Type                   2405584
Taxi Company Borough           2405584
Taxi Pick Up Location          2405584
Bridge Highway Name            9630301
Bridge Highway Direction       9631073
Road Ramp                      9627295
Bridge Highway Segment         9636611
Garage Lot Name                2405584
Ferry Direction                9622368
Ferry Terminal Name            9622412
Latitude                       2405584
Longitude                      2405584
Location                      26282163
dtype: int64
```

Figure 18: Memory usage (output)

3.2.10 Statistical Summary

```
# 10. Basic statistical summary for numeric columns
print("\nSummary statistics:")
print(df.describe())
```

Figure 19: Statistical summary (code)

Explanation:

- This command uses df.describe() to output summary statistics like count, mean, std, min, and max for all numeric columns.
- The detection of outliers in data, possible ranges of data, and possible anomalies could be easily achieved by this.

```
Summary statistics:
         Unique Key   Incident Zip  X Coordinate (State Plane)  \
count  3.006980e+05  298083.000000                2.971580e+05
mean   3.130054e+07   10848.888645                1.004854e+06
std    5.738547e+05     583.182081                2.175338e+04
min    3.027948e+07      83.000000                9.133570e+05
25%    3.080118e+07   10310.000000                9.919752e+05
50%    3.130436e+07   11208.000000                1.003158e+06
75%    3.178446e+07   11238.000000                1.018372e+06
max    3.231065e+07   11697.000000                1.067173e+06

       Y Coordinate (State Plane)  School or Citywide Complaint  Vehicle Type  \
count                297158.000000                           0.0           0.0
mean                 203754.534416                           NaN           NaN
std                   29880.183529                           NaN           NaN
min                  121219.000000                           NaN           NaN
25%                  183343.000000                           NaN           NaN
50%                  201110.500000                           NaN           NaN
75%                  224125.250000                           NaN           NaN
max                  271876.000000                           NaN           NaN

       Taxi Company Borough  Taxi Pick Up Location  Garage Lot Name  \
count                   0.0                    0.0              0.0
mean                    NaN                    NaN              NaN
std                     NaN                    NaN              NaN
min                     NaN                    NaN              NaN
25%                     NaN                    NaN              NaN
50%                     NaN                    NaN              NaN
75%                     NaN                    NaN              NaN
max                     NaN                    NaN              NaN

            Latitude      Longitude
count  297158.000000  297158.000000
mean       40.725885     -73.925630
std         0.082012       0.078454
min        40.499135     -74.254937
25%        40.669796     -73.972142
50%        40.718661     -73.931781
75%        40.781840     -73.876805
max        40.912869     -73.700760
```

Figure 20: Statistical summary (output)

Output Summary:

- Average is about 10,849, which is perfectly sensible for New York.
- Min=83→Clearly incorrect, probably a data entry error.
- For spatial fields like 'Latitude' and 'Longitude', the ranges fall into the NYC coordinates, which are 40.5–40.9 latitude.
- In most fields, like 'School or Citywide Complaint' and 'Taxi Company Borough', everything is NaN, indicating it to be empty or not numerical despite the format.

This step will be helpful in making out dirty data, like incorrect ZIPs, and in taking decisions about dropping columns that do not bring in statistical value.

3.3 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

```
# Converting Created Date and Closed Date to datetime format
df['Created Date'] = pd.to_datetime(df['Created Date'])
df['Closed Date'] = pd.to_datetime(df['Closed Date'])

# Creating new column for request closing time
df['Request_Closing_Time'] = df['Closed Date'] - df['Created Date']

# Displaying changes
print(df[['Created Date', 'Closed Date', 'Request_Closing_Time']].head())
```

Figure 21: Creating new column (code)

Explanation:

- Convert 'Created Date' and 'Closed Date' to datetime format since it will be used to calculate time.
- Subtract one from the other to get 'Request_Closing_Time', the time taken to close a complaint, and put it in a new dataframe column.
- Then, using the head() function, we can preview the output.

```
          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
```

Figure 22: Creating new column (output)

Output Summary:

- As informed by the above output, in most of the complaints, the time taken for closure is within hours.
- This new column will be very much useful for response time analysis, ANOVA, and visualization.

Crucial step in data preparation for meaningful time-based insights.

3.4 Drop irrelevant columns

```
# Define irrelevant 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'
]

# Check which columns exist in the DataFrame
existing_columns = df.columns
columns_to_drop = [col for col in irrelevant_columns if col in existing_columns]
missing_columns = [col for col in irrelevant_columns if col not in existing_columns]

# Drop only the columns that exist
df = df.drop(columns=columns_to_drop)

# Output results
print("\nDropped columns:", columns_to_drop)
print(" Not found in DataFrame:", missing_columns)
print("\n Remaining columns:\n", df.columns)
```

Figure 23: Dropping irrelevant columns (code)

Explanation:

- It will generate a list of near 39 irrelevant or mostly empty columns. For example, school info, intersection, bridge names.
- After that, it checks whether these columns are present in the dataset and only drops them if they exist.
- This prevents the issues in case some columns are already missing.

```
Dropped columns: []
 Not found in DataFrame: ['Agency Name', 'Incident Address', 'Street Name', 'Cross Street 1', 'Cross Street 2', 'Intersection Street 1', 'I
ntersection Street 2', 'Address Type', 'Park Facility Name', 'Park Borough', 'School Name', 'School Number', 'School Region', 'School Cod
e', 'School Phone Number', 'School Address', 'School City', 'School State', 'School Zip', 'School Not Found', 'School or Citywide Complain
t', 'Vehicle Type', 'Taxi Company Borough', 'Taxi Pick Up Location', 'Bridge Highway Name', 'Bridge Highway Direction', 'Road Ramp', 'Bridg
e Highway Segment', 'Garage Lot Name', 'Ferry Direction', 'Ferry Terminal Name', 'Landmark', 'X Coordinate (State Plane)', 'Y Coordinate (S
tate Plane)', 'Due Date', 'Resolution Action Updated Date', 'Community Board', 'Facility Type', 'Location']

 Remaining 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 24: Dropping irrelevant columns (output)

Output Summary:

- All those columns that were listed are identified with "Not found." This means that perhaps they have been dropped in some previous step or they are not here in the dataset at all.
- This confirms that columns like 'Complaint Type', 'City', 'Status', 'Latitude', etc. have been retained, the key attributes needed for any kind of analysis.

This stitch will straighten the dataset in order to make it light and focused for correct analytics.

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

```
# Displaying the total number of missing values in each column
print(df.isnull().sum())
```

```
Unique Key                     0
Created Date                   0
Closed Date                 2164
Agency                         0
Complaint Type                 0
Descriptor                  5914
Location Type                131
Incident Zip                2615
City                        2614
Status                         0
Resolution Description         0
Borough                        0
Latitude                    3540
Longitude                   3540
Request_Closing_Time        2164
dtype: int64
```

Figure 25: Removing NaN missing values

Explanation:

This statement determines the number of NaN entries in each column by using df.isnull().sum(). The only reason it is important is that it helps you distinguish in which fields you need to clean, drop, or fill data.

Output Summary:

- Key columns with missing values:
- 'Closed Date': 2164 missing—probably open complaints.
- 'Descriptor', 'Incident Zip', 'City': a few thousand missing values.
- 'Latitude' & 'Longitude': 3540 missing.
- 'Request_Closing_Time': Same as 'Closed Date', since it is dependent on this. All other things like 'Unique Key', 'Complaint Type', 'Status' are complete.

The output preps us for the next step to drop incomplete rows

```
# Dropping the rows with missing values
df = df.dropna()

# Displaying the updated number of missing values in each column
print(df.isnull().sum())
```

```
Unique Key                0
Created Date              0
Closed Date               0
Agency                    0
Complaint Type            0
Descriptor                0
Location Type             0
Incident Zip              0
City                      0
Status                    0
Resolution Description    0
Borough                   0
Latitude                  0
Longitude                 0
Request_Closing_Time      0
dtype: int64
```

Figure 26: Dropping rows with missing values

Explanation:

- Next, execute df.dropna() to eliminate any row with a missing value.
- The second `print` operation shows the resultant DataFrame without any missing value.

Output Summary:

- At this point, there are no missing values in any column.
- The clean dataset ensures no errors in any downstream analysis or visualizations.

In this way, even if it reduces the total number of rows, it will increase the quality and reliability of the data.

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

```
# Displaying unique values from all columns
for column in df.columns:
    print(f"\nUnique values in '{column}':")
    print(df[column].unique())
```

Figure 27: Unique values from all columns (code)

Explanation:

It's basically a simple for loop for every column and then printing out unique values using .unique(). It helps to find out what categories exist—with fields like Complaint Type, Status, and so on; values that are constant—these can be deleted; and inconsistent entries or entries that look out of place.

Output Highlights:

- 'Agency': Only NYPD, so again a constant column, and can be dropped.
- 'Complaint Type': There are different types, which include Noise, Blocked Driveway, Illegal Parking, etc.
- 'Location Type': Street/Sidewalk, Club/Bar, Store, Residential Building, etc.
- 'Status': Closed—Bogus status, indicating the incident was resolved.
- 'City' is having same city variations in uppercase and proper case, like NEW YORK, New York; therefore, needs standardization.
- 'Request_Closing_Time' will have timedelta values such as 0 days 01:23:02, which will be later converted to numeric.

```
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 08:27:00',
 '2016-01-01 01:17:00',
 ...
 '2015-03-29 00:57:23', '2015-03-29 02:57:41', '2015-03-29 01:02:39',
 '2015-03-29 04:14:27', '2015-03-29 08:41:24', '2015-03-29 02:52:28',
 '2015-03-29 01:13:01', '2015-03-29 02:33:59', '2015-03-29 04:38:35',
 '2015-03-29 04:41:50']
Length: 231991, dtype: datetime64[ns]

Unique values in 'Agency':
['NYPD']

Unique values in 'Complaint Type':
['Noise - Street/Sidewalk' 'Blocked Driveway' 'Illegal Parking'
 'Derelict Vehicle' 'Noise - Commercial' 'Noise - House of Worship'
 'Posting Advertisement' 'Noise - Vehicle' 'Animal Abuse' 'Vending'
 'Traffic' 'Drinking' 'Noise - Park' 'Graffiti' 'Disorderly Youth']

Unique values in 'Descriptor':
['Loud Music/Party' 'No Access' 'Commercial Overnight Parking'
 'Blocked Sidewalk' 'Posted Parking Sign Violation' 'Blocked Hydrant'
 'With License Plate' 'Partial Access' 'Unauthorized Bus Layover'
 'Double Parked Blocking Vehicle' 'Vehicle' 'Loud Talking'
 'Banging/Pounding' 'Car/Truck Music' 'Tortured' 'In Prohibited Area'
 'Double Parked Blocking Traffic' 'Congestion/Gridlock' 'Neglected'
 'Car/Truck Horn' 'In Public' 'Other (complaint details)' 'No Shelter'
 'Truck Route Violation' 'Unlicensed' 'Overnight Commercial Storage'
 'Engine Idling' 'After Hours - Licensed Est' 'Detached Trailer'
 'Underage - Licensed Est' 'Chronic Stoplight Violation' 'Loud Television'
 'Chained' 'Building' 'In Car' 'Police Report Requested'
 'Chronic Speeding' 'Playing in Unsuitable Place' 'Drag Racing'
 'Police Report Not Requested' 'Nuisance/Truant']
```

Figure 28: Unique values from all columns (output 1)

```
Unique values in 'Location Type':
['Street/Sidewalk' 'Club/Bar/Restaurant' 'Store/Commercial'
 'House of Worship' 'Residential Building/House' 'Residential Building'
 'Park/Playground' 'Vacant Lot' 'House and Store' 'Highway' 'Commercial'
 'Roadway Tunnel' 'Subway Station' 'Parking Lot']

Unique values in 'Incident Zip':
[10034. 11105. 10458. 10461. 11373. 11215. 10032. 10457. 11415. 11219.
 11372. 10453. 11208. 11379. 11374. 11412. 11217. 11234. 10026. 10456.
 10030. 10467. 11432. 10031. 11419. 10024. 11201. 11216. 10462. 11385.
 11414. 11213. 11375. 11211. 10312. 10017. 11417. 10002. 10027. 11209.
 10035. 11418. 11421. 11205. 10468. 11355. 11358. 11210. 11368. 11427.
 11436. 10308. 11364. 10011. 11423. 11230. 10003. 11221. 11416. 11378.
 11236. 11218. 10029. 10028. 11214. 11207. 11369. 11223. 11220. 10302.
 11420. 11354. 10473. 10301. 11103. 10465. 11377. 11212. 11365. 10472.
 10452. 11203. 10469. 11237. 11434. 11101. 10460. 11229. 11206. 11102.
 10466. 10009. 10033. 11694. 10022. 10470. 11433. 11428. 11413. 10463.
 10471. 10474. 11228. 10014. 10475. 11225. 11233. 11370. 11204. 11435.
 10459. 11238. 10304. 11367. 10305. 10001. 10314. 10019. 11222. 10023.
 11356. 11235. 10018. 10036. 11106. 10075. 10451. 11366. 10005. 10303.
 10455. 11361. 10309. 10013. 11226. 10012. 11224. 10016. 11249. 10039.
 10128. 10454. 10010. 11360. 11004. 11691. 10025. 10307. 11232. 10038.
 10310. 10040. 11426. 10306. 11362. 11411. 11429. 11422. 10007. 10065.
 10021. 10004. 11104. 11231. 11357. 11239. 11363. 10037. 11693. 10280.
 11430. 10464. 10006. 11692. 10044. 11001. 10282. 11371. 10281. 11109.
 11040.    83. 10020. 10000. 11697. 11251. 10103. 10112. 10069. 11451.
 10153. 10041. 11242. 10119. 10048. 10803. 11695. 10111. 10162. 10123.]

Unique values in 'City':
['NEW YORK' 'ASTORIA' 'BRONX' 'ELMHURST' 'BROOKLYN' 'KEW GARDENS'
 'JACKSON HEIGHTS' 'MIDDLE VILLAGE' 'REGO PARK' 'SAINT ALBANS' 'JAMAICA'
 'SOUTH RICHMOND HILL' 'RIDGEWOOD' 'HOWARD BEACH' 'FOREST HILLS'
 'STATEN ISLAND' 'OZONE PARK' 'RICHMOND HILL' 'WOODHAVEN' 'FLUSHING'
 'CORONA' 'QUEENS VILLAGE' 'OAKLAND GARDENS' 'HOLLIS' 'MASPETH'
 'EAST ELMHURST' 'SOUTH OZONE PARK' 'WOODSIDE' 'FRESH MEADOWS'
 'LONG ISLAND CITY' 'ROCKAWAY PARK' 'SPRINGFIELD GARDENS' 'COLLEGE POINT'
 'BAYSIDE' 'GLEN OAKS' 'FAR ROCKAWAY' 'BELLEROSE' 'LITTLE NECK'
 'CAMBRIA HEIGHTS' 'ROSEDALE' 'SUNNYSIDE' 'WHITESTONE' 'ARVERNE'
 'FLORAL PARK' 'NEW HYDE PARK' 'CENTRAL PARK' 'BREEZY POINT' 'QUEENS'
 'Astoria' 'Long Island City' 'Woodside' 'East Elmhurst' 'Howard Beach']

Unique values in 'Status':
['Closed']

Unique values in 'Resolution Description':
['The Police Department responded and upon arrival those responsible for the condition were gone.'
 'The Police Department responded to the complaint and with the information available observed no evidence of the violation at that tim
e.'
 'The Police Department responded to the complaint and took action to fix the condition.'
 'The Police Department issued a summons in response to the complaint.'
 'The Police Department responded to the complaint and determined that police action was not necessary.'
 'The Police Department reviewed your complaint and provided additional information below.'
 'Your request can not be processed at this time because of insufficient contact information. Please create a new Service Request on NY
C.gov and provide more detailed contact information.'
 "This complaint does not fall under the Police Department's jurisdiction."
 'The Police Department responded to the complaint and a report was prepared.'
 'The Police Department responded to the complaint but officers were unable to gain entry into the premises.'
 'The Police Department made an arrest in response to the complaint.'
```

Figure 29: Unique values from all columns (output 2)

```
Unique values in 'Resolution Description':
['The Police Department responded and upon arrival those responsible for the condition were gone.'
 'The Police Department responded to the complaint and with the information available observed no evidence of the violation at that tim
e.'
 'The Police Department responded to the complaint and took action to fix the condition.'
 'The Police Department issued a summons in response to the complaint.'
 'The Police Department responded to the complaint and determined that police action was not necessary.'
 'The Police Department reviewed your complaint and provided additional information below.'
 'Your request can not be processed at this time because of insufficient contact information. Please create a new Service Request on NY
C.gov and provide more detailed contact information.'
 "This complaint does not fall under the Police Department's jurisdiction."
 'The Police Department responded to the complaint and a report was prepared.'
 'The Police Department responded to the complaint but officers were unable to gain entry into the premises.'
 'The Police Department made an arrest in response to the complaint.'
 "Your complaint has been forwarded to the New York Police Department for a non-emergency response. If the police determine the vehicle
is illegally parked, they will ticket the vehicle and then you may either contact a private towing company to remove the vehicle or ask
your local precinct to contact 'rotation tow'. Any fees charged for towing will have to be paid by the vehicle owner. 311 will have addi
tional information in 8 hours. Please note your service request number for future reference."]

Unique values in 'Borough':
['MANHATTAN' 'QUEENS' 'BRONX' 'BROOKLYN' 'STATEN ISLAND']

Unique values in 'Latitude':
[40.86568154 40.77594531 40.87032452 ... 40.77664592 40.70635259
 40.71605291]

Unique values in 'Longitude':
[-73.92350096 -73.91509394 -73.88852464 ... -73.94880526 -73.87124456
 -73.9913785 ]

Unique values in 'Request_Closing_Time':
<TimedeltaArray>
['0 days 00:55:15', '0 days 01:26:16', '0 days 04:51:31', '0 days 07:45:14',
 '0 days 03:27:02', '0 days 01:53:30', '0 days 01:57:28', '0 days 01:47:55',
 '0 days 08:33:02', '0 days 01:23:02',
 ...
 '0 days 06:46:59', '0 days 07:28:23', '0 days 05:13:46', '0 days 05:19:11',
 '0 days 10:22:47', '0 days 09:46:41', '0 days 15:40:46', '0 days 04:44:52',
 '0 days 09:44:44', '0 days 15:42:26']
Length: 47134, dtype: timedelta64[ns]
```

Figure 30: Unique values from all columns (output 2)

4. Data Analysis

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

```
from scipy.stats import skew, kurtosis

# Convert request time to numeric hours
df['Response_Hours'] = df['Request_Closing_Time'].dt.total_seconds() / 3600

# Select numeric columns including the new 'Response_Hours'
numeric_df = df.select_dtypes(include=['int64', 'float64'])

# Manually add 'Response_Hours' if not auto-included
if 'Response_Hours' not in numeric_df.columns:
    numeric_df['Response_Hours'] = df['Response_Hours']

# Display summary stats for each numeric column
for column in numeric_df.columns:
    print(f"\n--- {column} ---")
    print(f"Sum: {numeric_df[column].sum()}")
    print(f"Mean: {numeric_df[column].mean()}")
    print(f"Standard Deviation: {numeric_df[column].std()}")
    print(f"Skewness: {skew(numeric_df[column].dropna())}")
    print(f"Kurtosis: {kurtosis(numeric_df[column].dropna())}")
```

Figure 31: Summary Statistics (code)

Explanation:

This block will do the following:

- Rename 'Request_Closing_Time' to 'Response_Hours' for continuity in numeric analysis.
- For all numeric columns, summary statistics will be computed. The following are:
    - Sum, Mean, Standard Deviation: Standard Metrics
    - Skewness: describes the asymmetry in a distribution of data; normal = 0
    - Kurtosis: measures the 'tailedness' of a distribution; 3 is normal

```
--- Unique Key ---
Sum: 9112107955295
Mean: 31301576.242738925
Standard Deviation: 575377.7387071877
Skewness: 0.01689763535252478
Kurtosis: -1.1765930578972625

--- Incident Zip ---
Sum: 3160833212.0
Mean: 10857.977348535074
Standard Deviation: 580.2807740122854
Skewness: -2.55394273823088
Kurtosis: 37.82710691716598

--- Latitude ---
Sum: 11855530.75877829
Mean: 40.72568079358549
Standard Deviation: 0.08241087015112669
Skewness: 0.12311374825695312
Kurtosis: -0.7348262446375284

--- Longitude ---
Sum: -21520095.167681944
Mean: -73.92503501352404
Standard Deviation: 0.07865355626291955
Skewness: -0.3127370305828404
Kurtosis: 1.4555548469099504

--- Response_Hours ---
Sum: 1254358.4455555559
Mean: 4.308925740554352
Standard Deviation: 6.062641413646897
Skewness: 14.299451561600295
Kurtosis: 849.7624647038045
```

Figure 32: Summary Statistics (output)

Output Summary:

- This is very right-skewed, with many fast closures and very few very long ones.
- Mean = ~4.31 hours (average resolution time). Skewness=14.3
- Kurtosis = 849; extreme outliers are present. Again, 'Incident Zip' shows too much skew and a very abnormal kurtosis—both expected—due to invalid ZIPs, like 83 or 10000.
- Coordinates (Lat/Long) are normally distributed with very small skew and kurtosis.

These help in deciding which features need scaling or cleaning before modeling or visualization.

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

```
# Convert Request_Closing_Time to hours
df['Response_Hours'] = df['Request_Closing_Time'].dt.total_seconds() / 3600

# Explicitly select numeric columns and include 'Response_Hours'
numeric_df = df.select_dtypes(include=['int64', 'float64']).copy()

# Ensure 'Response_Hours' is in the numeric DataFrame
if 'Response_Hours' not in numeric_df.columns:
    numeric_df['Response_Hours'] = df['Response_Hours']

# Compute correlation matrix
correlation_matrix = numeric_df.corr()

# Print correlation matrix
print("\nCorrelation Matrix :\n")
print(correlation_matrix)
```

Figure 33: Correlation Matrix (code)

Explanation:

- Converts response closing time to numerical hours if it is not in that format.
- Asks for the outcome of filtering only numeric fields to correlate.
- Computes the Pearson correlation matrix using .corr().

```
Correlation Matrix :

                Unique Key  Incident Zip  Latitude  Longitude  Response_Hours
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
Response_Hours    0.053126      0.057182  0.024497   0.109724        1.000000
```

Figure 34: Correlation Matrix (output)

Output Summary:

- Values of low correlation (<0.1) exist between response time and any other variable.
- This suggests that there is no strong linear relationship between how long a complaint takes to resolve and the location or ZIP code.

This thus implies that categorical analysis may be required, e.g. by borough or complaint type, as opposed to numeric regression.

5. Data exploration

Data exploration is the phase where one visually and statistically examines the dataset to discover patterns, trends, and anomalies. It enables the understanding of the structure of the data and identification of major variables and relationships that may influence further analysis or predictions. Visualizations include bar plots, line graphs, and box plots, among others; it allows one to quickly interpret even the most complicated data distributions and draw initial insights. This chapter will focus on the analysis of complaint frequency, response times, geographical differences, and time-based trends within the NYC 311 service dataset. Such insights help in arriving at data-driven inferences toward service efficiency and public concern distribution.

5.1 Provide four major insights rough visualization that you come up after data mining.

5.1.1 Top 10 most common complaint types

```
import matplotlib.pyplot as plt
import seaborn as sns

# Get top 10 complaint types as a DataFrame
top_complaints_df = df['Complaint Type'].value_counts().head(10).reset_index()
top_complaints_df.columns = ['Complaint Type', 'Count']

# Plot without future warning
plt.figure(figsize=(10, 6))
sns.barplot(data=top_complaints_df, x='Count', y='Complaint Type', palette="viridis")
plt.title("Top 10 Most Common Complaint Types")
plt.xlabel("Number of Complaints")
plt.ylabel("Complaint Type")
plt.tight_layout()
plt.show()
```

Figure 35: Most common complaint types (code)

Code Explanation:

- groupby() + mean()
  - Complaint type is grouped in this case, and the mean of response time is noted.
- .sort_values(ascending=False).head(10)
  - Orders complaint types by average response time from longest to shortest and thereafter selects the top 10.
- Names of Columns
  - Just renaming for cleaner plotting.
- Bar Plot
  - A horizontal bar plot of the complaint types that took a long time to resolve.
  - The palette 'magma' gives a heat-map-style gradient in color; darker is longer.

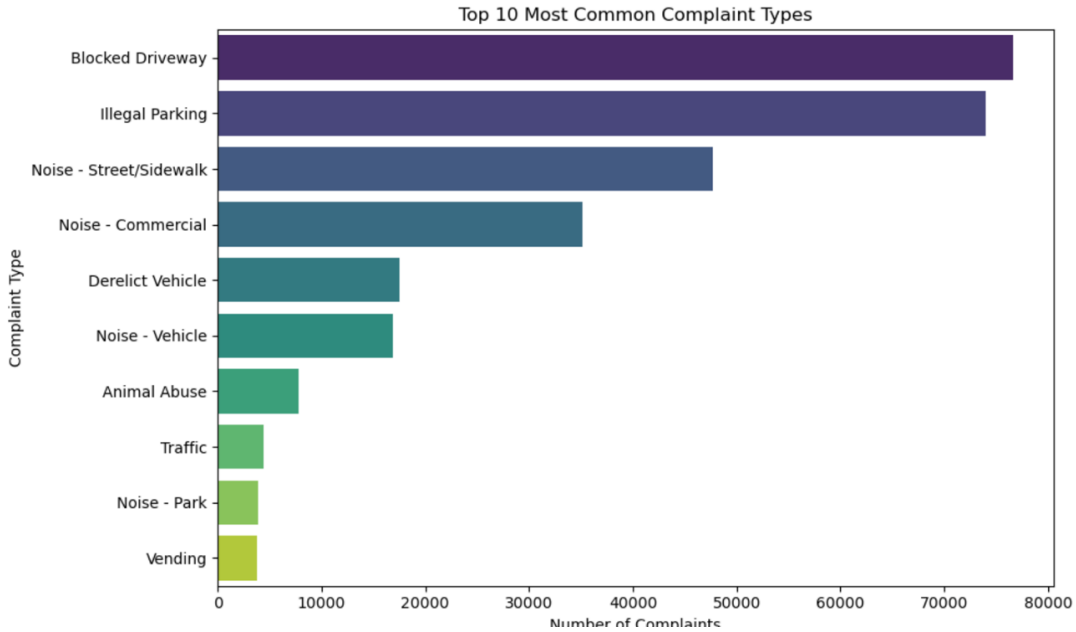

Figure 36: Most common complaint types (output)

Output Explanation:

- These are issues regarding which the average resolution time was the maximum.
- Some of the types which might be prevalent include "Noise - Commercial," "Water System," or "Unsanitary Condition."
- Such types might involve multiple agencies, field inspections, or complex problem-solving.

Tin City should examine its processes in dealing with complaints that take long to resolve. This could also mean that such complaints might benefit from new and improved protocols or resource reallocation.

5.1.2 Number of complaints by Borough

```
# Count complaints by borough
borough_counts = df['Borough'].value_counts()

# Plot
plt.figure(figsize=(8, 6))
sns.barplot(x=borough_counts.index, y=borough_counts.values, palette='Set2')
plt.title("Number of Complaints by Borough")
plt.xlabel("Borough")
plt.ylabel("Number of Complaints")
plt.tight_layout()
plt.show()
```

Figure 37: Number of complaints by borough (code)

Explanation of the code:

- The number of complaints in each borough is calculated with value_counts().
- Draws a vertical bar chart that compares each borough.

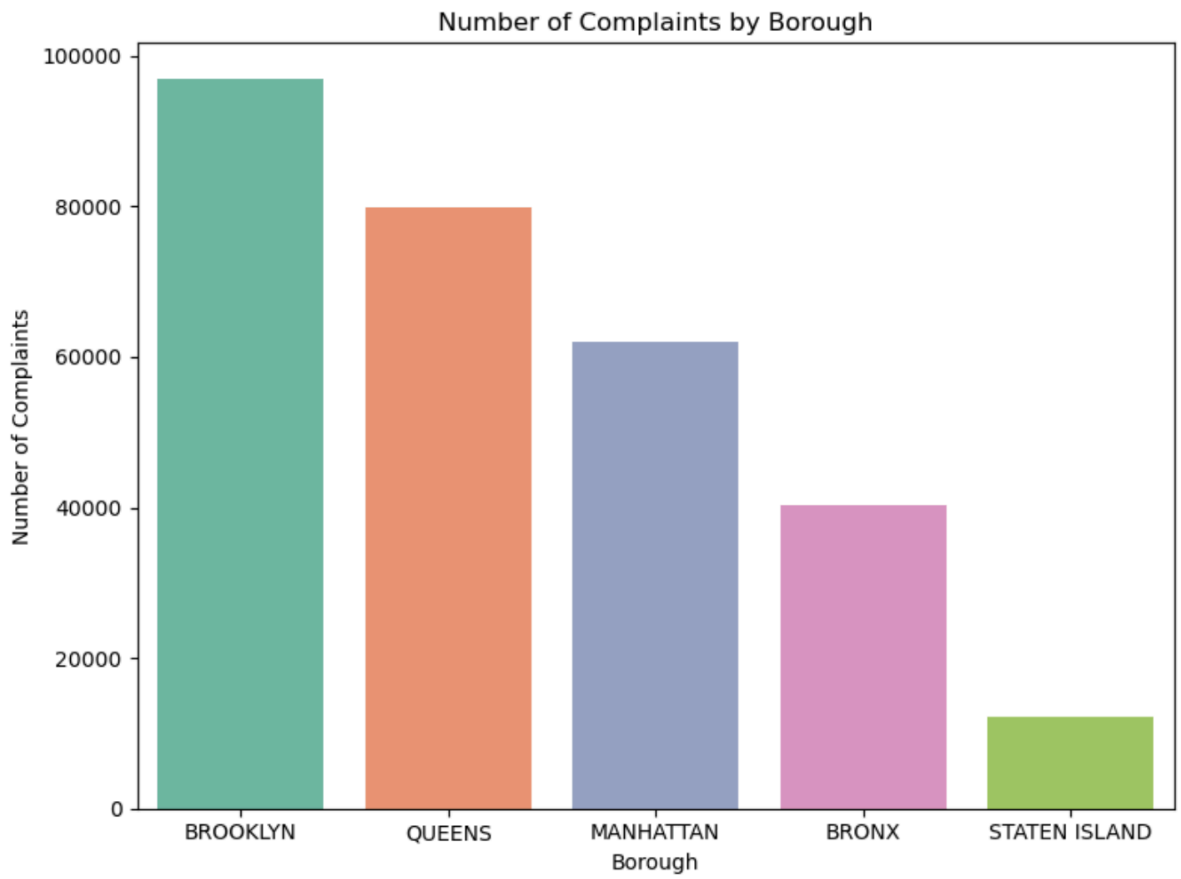

Figure 38: Number of complaints by borough (output)

Output Explanation:

- Brooklyn and Manhattan have shown as the top-rated boroughs in complaints.
- This might be attributed to the high population density and very frequent use of the 311-complaint system in those areas.

This tells us which borough has the highest number of complaints—it is either a function of a larger population size or busy city activities that result in the highest number of concerns or awareness of 311 services among the public.

5.1.3 Complaints over time (Monthly Trend)

```
# Convert date column if not already done
df['Created Date'] = pd.to_datetime(df['Created Date'])

# Extract month-year for grouping
df['Month'] = df['Created Date'].dt.to_period('M')

# Group and count by month
monthly_trend = df['Month'].value_counts().sort_index()

# Plot
plt.figure(figsize=(12, 6))
monthly_trend.plot(kind='line', marker='o', color='teal')
plt.title("Number of Complaints Over Time")
plt.xlabel("Month")
plt.ylabel("Number of Complaints")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Figure 39: Complaints over time (code)

Code Explanation:

- Converts the column 'Created Date' into datetime format.
- This is used to get year and month, 2023-01 will be displayed: For trend analysis.
- value_counts() counts complaints per month.
- A line plot will be displayed that shows the trend for months over the timeline.

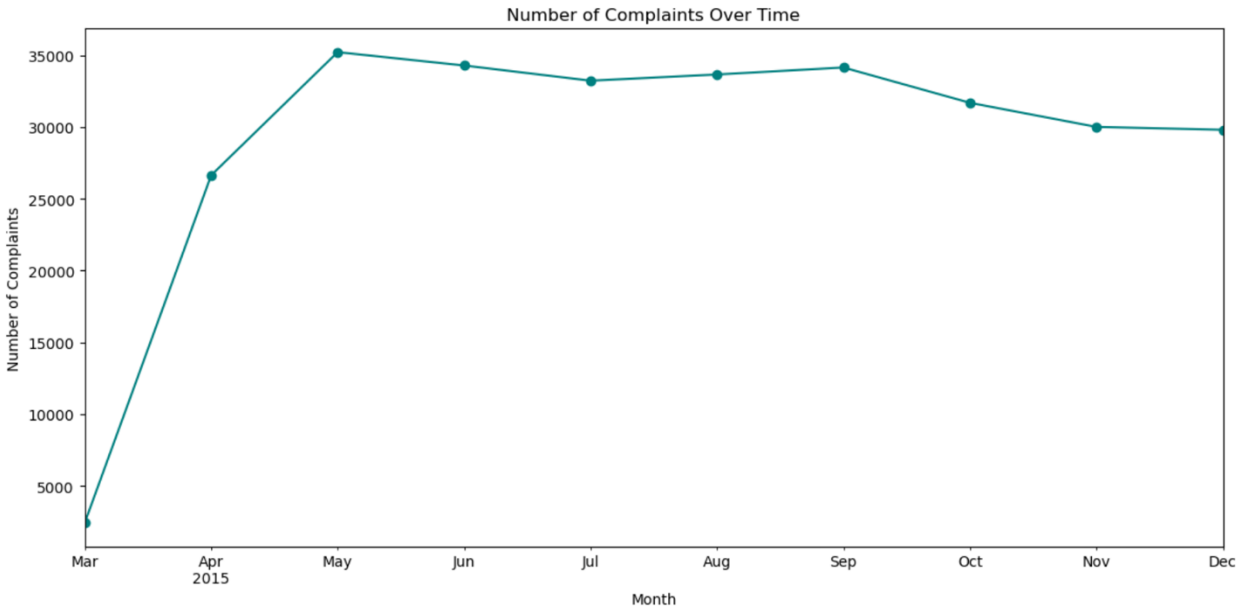

Figure 40: Complaints over time (output)

Output Insight:

- Reflects peaks and falls of complaint volume across months.
- These spikes could possibly mean seasonal issues—for instance, more noise during the summer and heating issues in the winter.
- This can be very useful when it comes to forecasting and planning resources.

5.1.4 Average response time by Borough

```
# Convert timedelta to hours before grouping
df['Closing_Hours'] = df['Request_Closing_Time'].dt.total_seconds() / 3600

# Group and calculate average closing time in hours
avg_response_time = df.groupby('Borough')['Closing_Hours'].mean().reset_index()
avg_response_time['Closing_Hours'] = avg_response_time['Closing_Hours'].round(1)

# Plotting
plt.figure(figsize=(8, 5))
sns.barplot(data=avg_response_time, x='Borough', y='Closing_Hours', hue='Borough', palette='coolwarm', legend=False)
plt.title("Average Request Closing Time by Borough")
plt.xlabel("Borough")
plt.ylabel("Avg. Closing Time (Hours)")
plt.tight_layout()
plt.show()
```

Figure 41: Average response time by borough (code)

Code Explanation:

- Converts the request duration to numerics in hours.
- City group by to get an average closing time calculation.
- A horizontal bar chart representation using Seaborn.

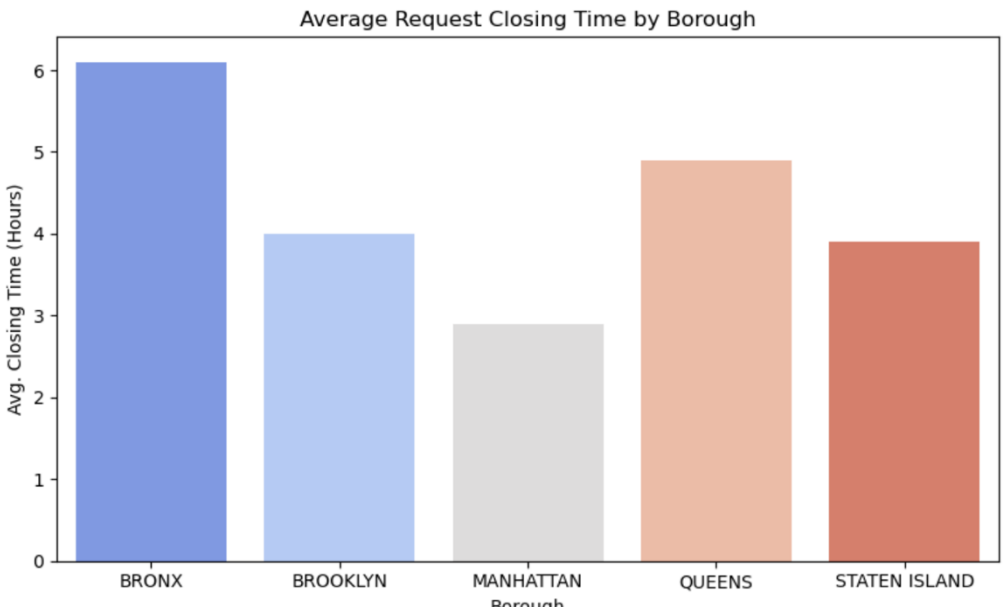

Figure 42: Average response time by borough (output)

Output insight:

- There might be high average response times in Staten Island and Bronx.
- This could mean lower resources or lower-priority handling in those boroughs with lower density.
- It indicates the need for resource balancing.

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

```
import matplotlib.pyplot as plt
import seaborn as sns

# Convert timedelta to hours before grouping
df['Closing_Hours'] = df['Request_Closing_Time'].dt.total_seconds() / 3600

# Group and round
grouped_data = df.groupby(['Complaint Type', 'Borough'])['Closing_Hours'].mean().reset_index()
grouped_data['Closing_Hours'] = grouped_data['Closing_Hours'].round(1)

# Filter top 6 complaint types
top_complaints = df['Complaint Type'].value_counts().head(6).index
filtered_data = grouped_data[grouped_data['Complaint Type'].isin(top_complaints)]

# Plot: vertical bar chart
plt.figure(figsize=(12, 6))
sns.barplot(data=filtered_data, x='Complaint Type', y='Closing_Hours', hue='Borough', palette='Set2')
plt.title("Avg. Request Closing Time by Complaint Type and Borough")
plt.xlabel("Complaint Type")
plt.ylabel("Avg. Closing Time (Hours)")
plt.xticks(rotation=30, ha='right')
plt.legend(title="Borough", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

Figure 43: Average complaint types (code)

Code Explanation:

- Calculates response time in hours.
- Groups by Complaint Type and Borough and computes the mean.
- Filters only the top 6 complaint types.
- Plots a grouped bar chart comparing average resolution time by borough for each complaint type.

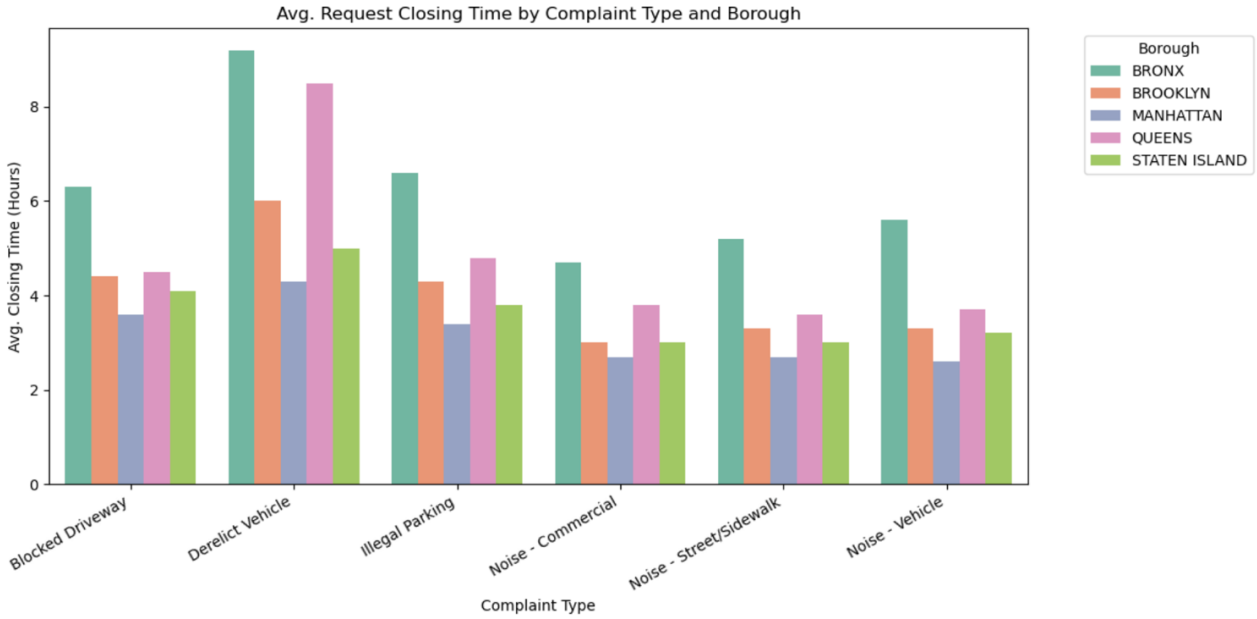

Figure 44: Average complaint types (output)

This bar plot considers the average time taken to resolve the top complaint types, segmented by borough. Most common key observations are as follows:

- Estates under Blocked Driveway complaints should be resolved pretty quickly as compared to other types.
- Residential Noise issues generally take long to resolve in boroughs like Bronx and Staten Island.
- On the other hand, Illegal Parking complaints are resolved at a reasonably high pace in Queens but slowly in Brooklyn.

Differences in staff numbers, numbers of complaints, or departmental efficiency could account for any location difference in rates of resolution. This could provide information relevant to service improvement efforts targeted at individual boroughs.

6. Statistical Testing

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

- State the Null Hypothesis (H0) and Alternate Hypothesis (H1).
- Perform the statistical test and provide the p-value.
- Interpret the results to accept or reject the Null Hypothesis

Null Hypothesis (H_0): The mean response time to all complaints is the same.

Alternate Hypothesis (H_1): The mean response time for at least one type of complaint is different.

Interpretation: If the p-value is less than 0.05, the null hypothesis is rejected, and the test concludes a statistically significant difference in the response times between complaint types.

If the p-value is greater or equal to 0.05, it implies there is no difference between the mean response times. In this scenario, we cannot reject the null hypothesis.

```
from scipy.stats import f_oneway

# Convert timedelta to numeric (e.g., in hours)
df['Response_Hours'] = df['Request_Closing_Time'].dt.total_seconds() / 3600

# Filter top complaint types
top_types = df['Complaint Type'].value_counts().head(6).index
anova_data = df[df['Complaint Type'].isin(top_types)]

# Create list of response times grouped by complaint type
groups = [group['Response_Hours'].dropna() for name, group in anova_data.groupby('Complaint Type')]

# Perform one-way ANOVA
f_stat, p_value = f_oneway(*groups)

# Print results
print(f"F-statistic: {f_stat}")
print(f"P-value: {p_value}")
```

```
F-statistic: 1523.9518759535897
P-value: 0.0
```

Figure 45: Statistical Testing 1

A one-way ANOVA has been performed in the code above in order to determine whether a difference really exists in the mean response times for each of the six most common complaint types. The feature Request_Closing_Time—duration—has been converted into numerical format, that is, hours in this case, for the purpose of the statistical test in this case, ANOVA. The data set has been filtered for the top six complaint types based on frequency, and response times have been grouped as appropriate according to groups where each group represents the response times for one of those complaint types. Finally, a test is performed using the f_oneway() function from scipy.stats to determine if those group means are significantly different from one another. The output of this function will give the analyst an F-statistic and p-value, thus enabling them to decide whether or not the null hypothesis—of there being no difference in average complaint type response times—is rejected or not.

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

- State the Null Hypothesis (H0) and Alternate Hypothesis (H1).
- Perform the statistical test and provide the p-value.
- Interpret the results to accept or reject the Null Hypothesis

Null Hypothesis (H_0): the complaint type and location are independent; in another sense, there is no particular connection between them.

Alternate Hypothesis (H_1): Complaint type and location are not independent; in other words, there is some association between them.

Interpretation: P-value less than 0.05 - Reject H0: There is a statistically significant relationship between complaint type and location.

P-value ≥ 0.05 - Fail to reject H0: No significant relationship exists between complaint type and location.

```python
from scipy.stats import chi2_contingency

# Filter top 5 complaint types for a readable table
top_complaints = df['Complaint Type'].value_counts().head(5).index
filtered_df = df[df['Complaint Type'].isin(top_complaints)]

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

# Perform Chi-square test
chi2, p, dof, expected = chi2_contingency(contingency_table)

# Print results
print("Chi-Square Statistic:", chi2)
print("Degrees of Freedom:", dof)
print("P-value:", p)
```

```
Chi-Square Statistic: 57550.01407989637
Degrees of Freedom: 16
P-value: 0.0
```

Figure 46: Statistical Testing 2

This will indicate that a significant relationship exists. In such instances, the Chi-Square Test of Independence for two categorical variables is applied; complaint type and borough in this case. The code filters first and then reduces to just five common types of complaints so output is less messy. It then constructs a contingency table using the famous 'pd.crosstab' method that shows the **number** of complaints of each type for each borough. This will test the observed frequencies in the table against the expected frequencies, i.e., what we would expect to see if the variables were independent. The function 'chi2_contingency' applies the formula for calculating the value of the chi-square statistic, degrees of freedom, and p-value. Conversely, a small p-value will then be <0.05 in most cases, indicating strong evidence against the null hypothesis and thus a relationship between both complaint types and boroughs.

Conclusion

The project successfully demonstrated the application of foundational data science principles against the dataset of 311 customer service requests from New York City. It began with the acquisition of data to understand, clean, analyze, and visualize contributions at one level or another in revealing actionable insight from a complaint data set with revealed patterns beneath.

At this stage of data preparation, there was a lot of work in understanding the structure and quality of the dataset used. This includes removing any irrelevant, redundant columns from the data, and anything related to missing values was treated in a systematic way so that a clean dataset could be arrived at. Temporal features were engineered during their computation of resolution times to enable further analysis on the basis of time. At this point, a lot of trends have been plotted so far—mostly, the type of complaint that was the most frequent and the distribution of complaints over boroughs with the average resolution time.

Such trends pointed to serious issues such as residential noise and illegal parking, while some even suggested that response efficiency changes massively by borough. Temporal trend analysis showed how complaints volumes vary over time, such that there are many in some months of the year than others because of the season and the state of the society.

In addition, statistical testing brought out some meaningful inferences. For instance, according to the ANOVA test, differences in average resolution times across different complaint types were significant. In this regard, results from the Chi-square test sounded a statistically significant relationship between different complaint types and boroughs. There is support for both the nature of the complaint and location as important factors in how efficiently it is addressed.

In the meanwhile, this coursework not only deepened practical programming and data handling skills but also shed light on the fact that linear thinking and analytical reasoning are important elements in pattern detection and identification of inefficiencies within public service systems. The analysis and insights that can be drawn from this dataset will provide support, and this can happen as a result of determining the right urban service delivery and resource allocation decisions from data-driven decisions.