Module Code & Module Title

CC5067NI Smart Data Discovery

60% Individual Coursework

Submission: Final Submission

Academic Semester: Spring Semester 2025

Credit: 15 credit semester long module

Student Name: Sajana Karki

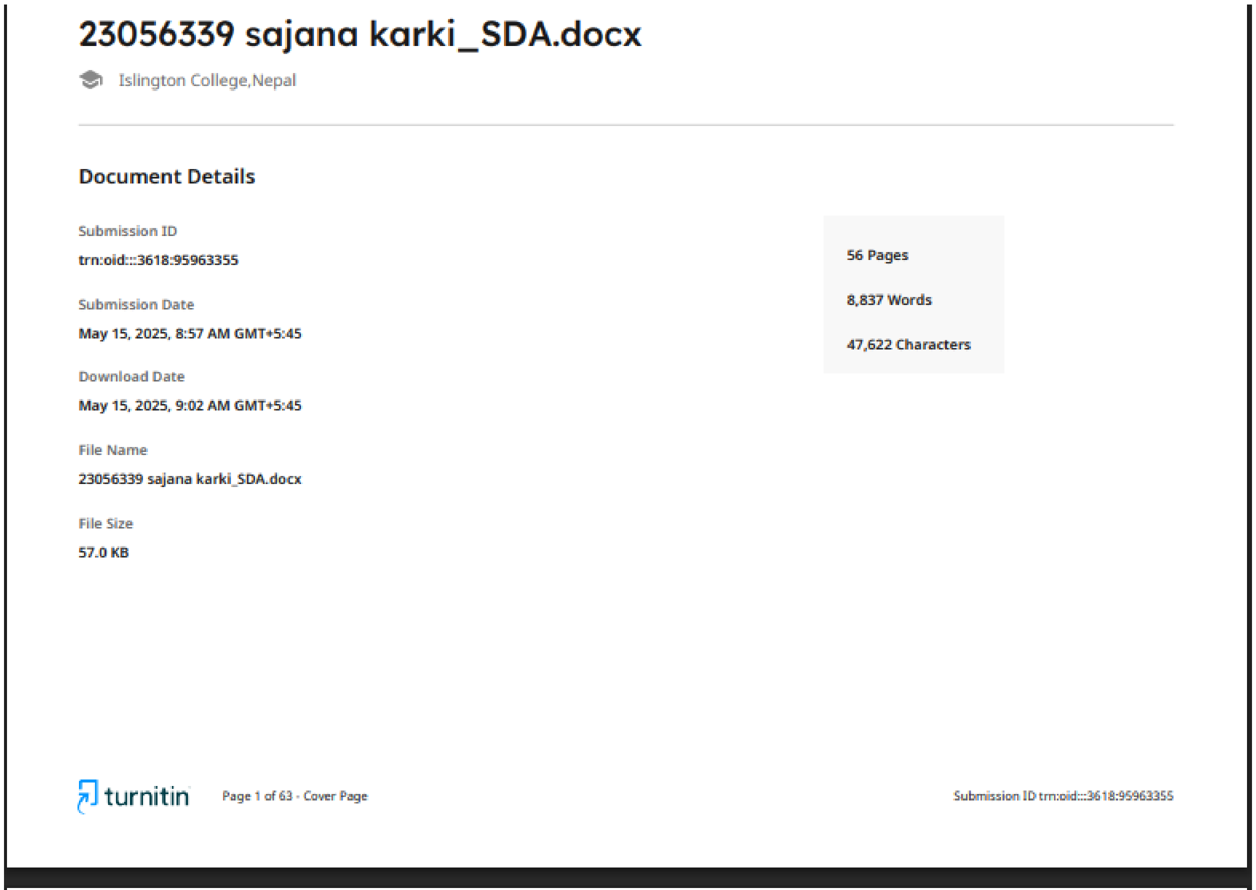
London Met ID: 23056339

College ID: NP01CP4S240034

Assignment Due Date: Thursday, May 15, 2025

Assignment Submission Date: Thursday, May 15, 2025

Submitted To: Dipeshor Silwal


23056339 sajana karki_SDA.docx

Islington College,Nepal

Document Details

Submission ID

trn:oid:::3618:95963355

Submission Date

May 15, 2025, 8:57 AM GMT+5:45

Download Date

May 15, 2025, 9:02 AM GMT+5:45

File Name

23056339 sajana karki_SDA.docx

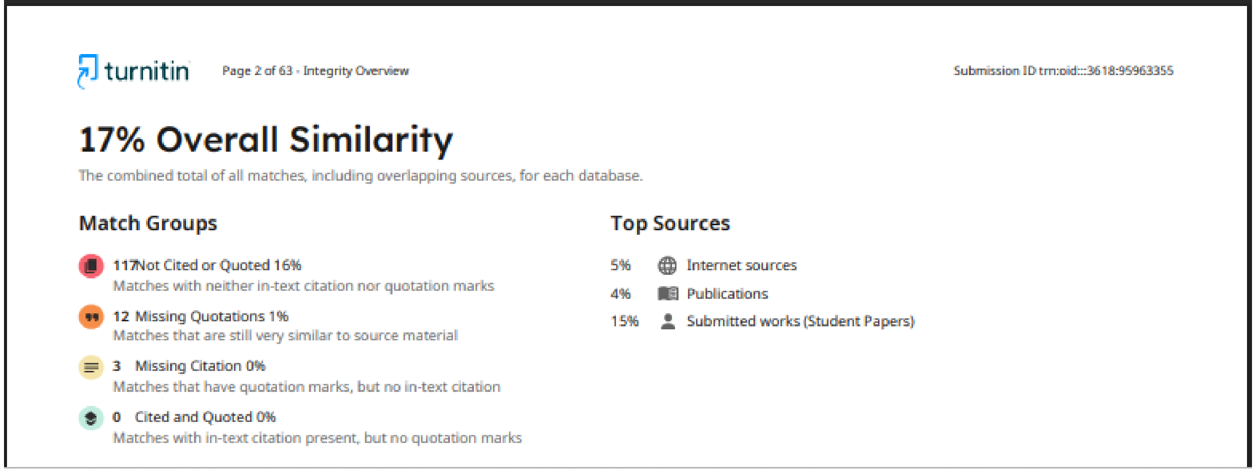
File Size

57.0 KB

56 Pages

8,837 Words

47,622 Characters



17% Overall Similarity

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

Match Groups

- 117 Not Cited or Quoted 16%
  Matches with neither in-text citation nor quotation marks
- 12 Missing Quotations 1%
  Matches that are still very similar to source material
- 3 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

5%	Internet sources
4%	Publications
15%	Submitted works (Student Papers)

Table of Contents

Table of Figures

Table of Tables

1. Data Understanding

The dataset used in this coursework goes by the name 'Customer Service Requests from 2010 to Present,' which host the complaint data registered by the public in New York City to the 311 service. One row represents a single service request in the dataset, which details a complaint's time and nature, location, responsible agency, and how it was resolved.

The dataset consists of more than 300,000 records and contains 53 columns. Created Date, Closed Date, Complaint Type, and Borough are some key columns that will enable understanding when, where, and what some problems are all about at most times, establish what kinds of problems are most common and how long it takes to get them resolved. Latitude and Longitude are columns that can be used to plot case complaints on the map and find a pattern covering more areas in the city.

The data is generally clean but has missing values in some of the columns. These are typically in columns which are only relevant to specific categories of complaints such as taxi, school or highway. Some of these columns are not going to be of use for our analysis and can be eliminated in the data cleaning process.

The data set is in CSV format, which makes it very convenient to work with using tools like Excel. This process of understanding data is very vital. It lets us know what the data contains, how the quality is, and what needs to be fixed or cleaned. Doing this properly ensures the data will be ready for deeper analysis later, like finding patterns or making predictions.

1.1 Data Description

Table 1 table of description

S.No.	Column Name	Description	Data Type
1	Unique Key	The unique identifier for each of the service requests.	Integer
2	Created Date	The date and time when the complaint was recorded.	Datetime
3	Closed Date	The date and time when the complaint was closed.	Datetime
4	Agency	The code of the city agency that is handling the request.	Object
5	Agency Name	The full name of the agency.	Object
6	Complaint Type	The type of complaint recorded.	Object
7	Descriptor	Detailed description of the issue.	Object
8	Location Type	The type of area where the issue occurred.	Object
9	Incident Zip	Zip code of the location where the issue happened.	Float
10	Incident Address	The street address where the incident occurred.	Object

11	Street Name	Name of the street where the incident occurred.	Object
12	Cross Street 1	First cross street near the incident location.	Object
13	Cross Street 2	Second cross street near the incident location.	Object
14	Intersection Street 1	First intersecting street for the incident location.	Object
15	Intersection Street 2	Second intersecting street for the incident location.	Object
16	Address Type	Type of address.	Object
17	City	Name of the city or neighbourhood where the complaint was made.	Object
18	Landmark	Nearby landmark related to the complaint location.	Object
19	Facility Type	Type of facility (e.g., park, school) associated with the complaint.	Object
20	Status	Status of the complaint (e.g., Open, Closed).	Object
21	Due Date	Expected date for the complaint to be resolved.	Datetime

22	Resolution Description	Description of how the complaint was resolved.	Object
23	Resolution Action Updated Date	Date when the resolution action was last updated.	Datetime
24	Community Board	Community board responsible for the location.	Object
25	Borough	NYC borough where the complaint was made.	Object
26	X Coordinate (State Plane)	X coordinate in the NYC State Plane coordinate system.	Float
27	Y Coordinate (State Plane)	Y coordinate in the NYC State Plane coordinate system.	Float
28	Park Facility Name	Name of the park facility, if relevant.	Object
29	Park Borough	Borough where the park facility is located.	String
30	School Name	Name of the school, if the complaint involves a school.	Object
31	School Number	School number related to the complaint.	Object
32	School Region	School region associated with the incident.	Object
33	School Code	Code assigned to the school.	Object

34	School Phone Number	Contact number of the school.	Object
35	School Address	Address of the school.	Object
36	School City	City where the school is located.	Object
37	School State	State where the school is located.	Object
38	School Zip	ZIP code of the school's location.	Object
39	School Not Found	Indicator if the school could not be identified.	Object
40	School or Citywide Complaint	Indicates whether the complaint is related to a school or citywide issue.	Float
41	Vehicle Type	Type of vehicle involved in the complaint.	Object
42	Taxi Company Borough	Borough where the taxi company is registered.	Object
43	Taxi Pick Up Location	Location where the taxi picked up the passenger.	Object
44	Bridge Highway Name	Name of the bridge or highway related to the complaint.	Object
45	Bridge Highway Direction	Direction of travel on the bridge or highway.	Object
46	Road Ramp	Specific ramp involved in the complaint.	Object
47	Bridge Highway Segment	Segment of the bridge or highway where the issue occurred.	Object

48	Garage Lot Name	Name of the garage or lot related to the issue.	Object
49	Ferry Direction	Direction of the ferry in question.	Object
50	Ferry Terminal Name	Name of the ferry terminal.	Object
51	Latitude	Geographic latitude of the complaint location.	Float
52	Longitude	Geographic longitude of the complaint location.	Float
53	Location	Combined latitude and longitude as a string format.	Object

2. Data Preparation

Data preparation is one of the important processes that involve cleaning, transforming, and structuring the raw data to make it ready for analysis. Data preparation is the gathering, consolidation, organizing and structuring of data to be used for business intelligence, analytics and data science. It's done in stages that entail preprocessing, profiling, cleansing, transforming and validating the data.

Data preparation also typically involves bringing together data from an organization's internal systems and external sources. In this section, the steps taken to prepare the NYC 311 service request dataset are discussed. (stedmen, 2021)

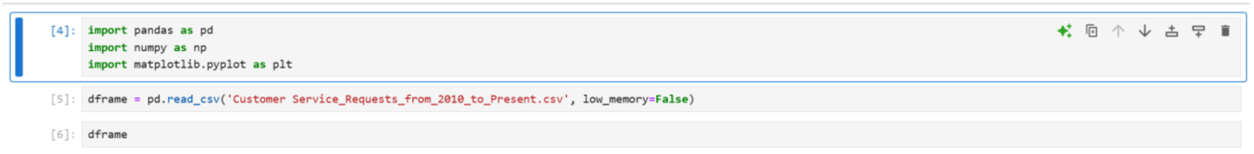
2.1 Question no.1: importing the dataset

Pandas is a capable and available library for data manipulation and analysis. It has capabilities for reading, cleaning, and exploring data in tabular format (i.e., data in rows and columns) using DataFrame objects which are like a spreadsheet or SQL tables.

NumPy (Numerical Python) is a library for numerical data. It contains support for arrays, mathematical operations, and a number of different functions for statistical analysis, which is common when performing data analysis.

Matplotlib is a library that is used to visualize data. pyplot is a module in matplotlib that provides a MATLAB-like interface to create figures and charts like line plots, bar charts, pie charts, etc.

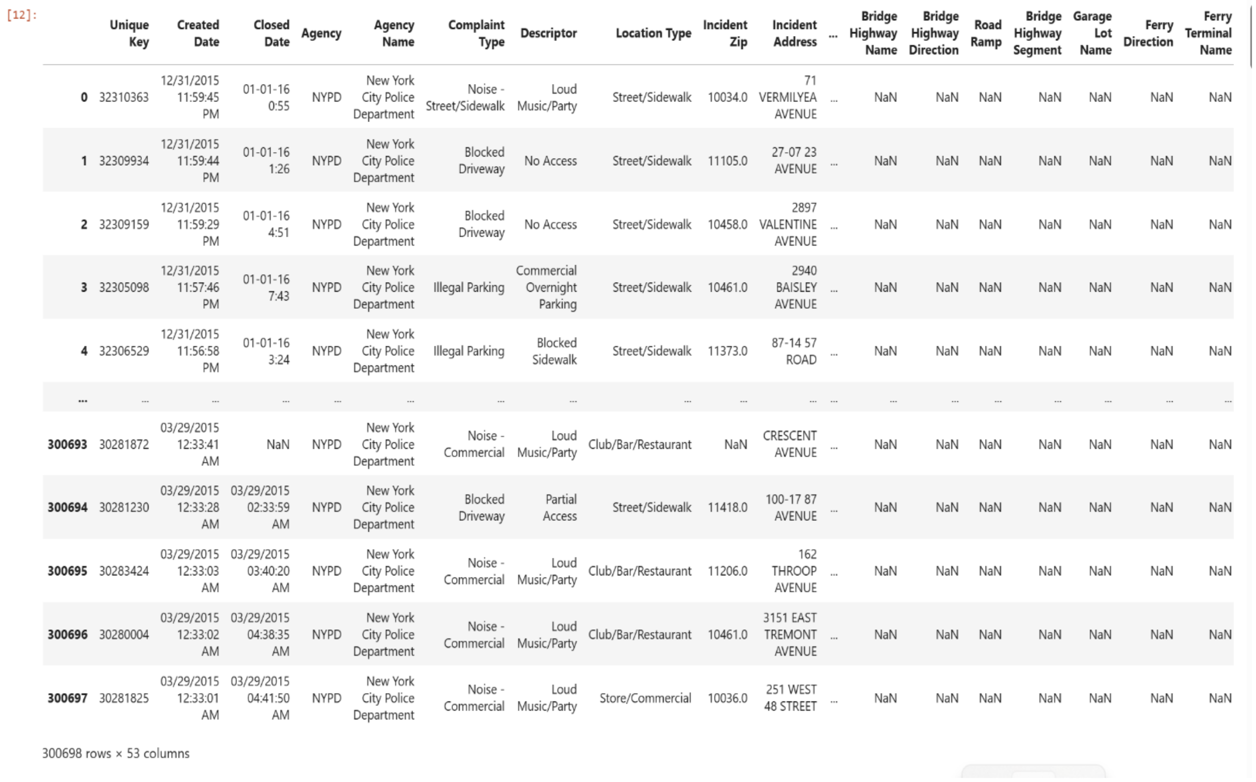
Code:



```
[4]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt

[5]: dframe = pd.read_csv('Customer Service_Requests_from_2010_to_Present.csv', low_memory=False)

[6]: dframe
```

Figure 1 Importing the datasets

Output:



[12]:

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	...	Bridge Highway Name	Bridge Highway Direction	Road Ramp	Bridge Highway Segment	Garage Lot Name	Ferry Direction	Ferry Terminal Name
0	32310363	12/31/2015 11:59:45 PM	01-01-16 0:55	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	10034.0	71 VERMILYEA AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	32309934	12/31/2015 11:59:44 PM	01-01-16 1:26	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	11105.0	27-07 23 AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	32309159	12/31/2015 11:59:29 PM	01-01-16 4:51	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	10458.0	2897 VALENTINE AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	32305098	12/31/2015 11:57:46 PM	01-01-16 7:43	NYPD	New York City Police Department	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk	10461.0	2940 BAISLEY AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	32306529	12/31/2015 11:56:58 PM	01-01-16 3:24	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11373.0	87-14 57 ROAD	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
300693	30281872	03/29/2015 12:33:41 AM	NaN	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	NaN	CRESCENT AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
300694	30281230	03/29/2015 12:33:28 AM	03/29/2015 02:33:59 AM	NYPD	New York City Police Department	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	100-17 87 AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
300695	30283424	03/29/2015 12:33:03 AM	03/29/2015 03:40:20 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	162 THROOP AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
300696	30280004	03/29/2015 12:33:02 AM	03/29/2015 04:38:35 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10461.0	3151 EAST TREMONT AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
300697	30281825	03/29/2015 12:33:01 AM	03/29/2015 04:41:50 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Store/Commercial	10036.0	251 WEST 48 STREET	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN

300698 rows × 53 columns

Figure 2 Displaying the output of datasets

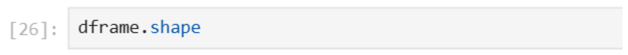
Explanation

The code is that which imports libraries which include pandas for handling data, numpy for performing numerical operations and matplotlib.pyplot for plotting. Next, that code reads the CSV into a DataFrame called dframe. A warning is raised because mixed-type data means that columns 48 and 49 will not work satisfactorily with data processing. Displaying dframe gives a glimpse of the first few rows in the dataset.

2.2 Question no 2: Insight on the information and details that the provided dataset carries

2.2.1 Shape of the Data frame

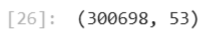
Code:



```
[26]: dframe.shape
```

Figure 3 code for creating shape of the data frame

Output:



```
[26]: (300698, 53)
```

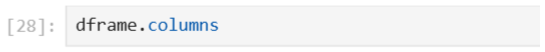
Figure 4 displaying shape of the data frame

Explanation

The dframe.shape attribute is a built-in property in pandas that shows the size of a DataFrame in the form of a tuple (rows, columns). The first value provides the total number of rows (records) in the dataset, and the second value provides the total number of columns (features or variables) in the dataset. The dframe.shape command is most often utilized during the data understanding and preparation phases to rapidly check the size of the dataset (before and after) any data cleaning, exploratory, filtering, or transformation process. This will allow a data engineer to confirm if the number of records and fields are as expected. This will also assist a data engineer in identifying any potential issues with their datasets and reporting systems, such as missing data after making modifications.

2.2.2 Columns of the Data frame

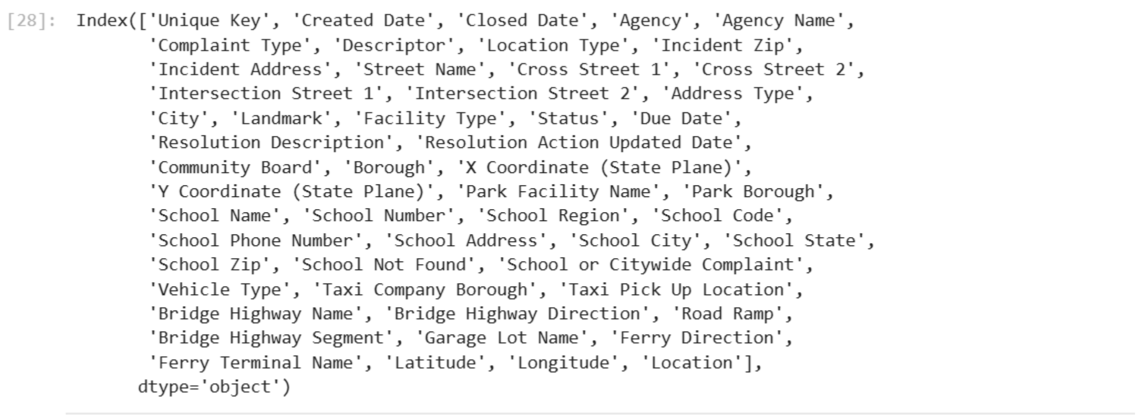
Code:



```
[28]: dframe.columns
```

Figure 5 code for creating columns of data frame

Output:



```
[28]: 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'],
            dtype='object')
```

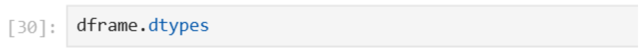
Figure 6 displaying columns of the dataframe

Explanation

In pandas, the dframe.columns attribute will return a list of the names (headers) of all the columns in that DataFrame. When this command is run, it will print the names of each column from the Index object, in the order that they appear in your dataset. This is very helpful in the early portions of the data exploration and preparation process because you get an overview of the features that are available to you from your dataset. You can refer to the column names when beginning the selection of columns to analyze or fields to rename. If there are any out of the ordinary or unwanted columns you may identify them to remove them or clean them.

2.2.3 Data types of the Data set

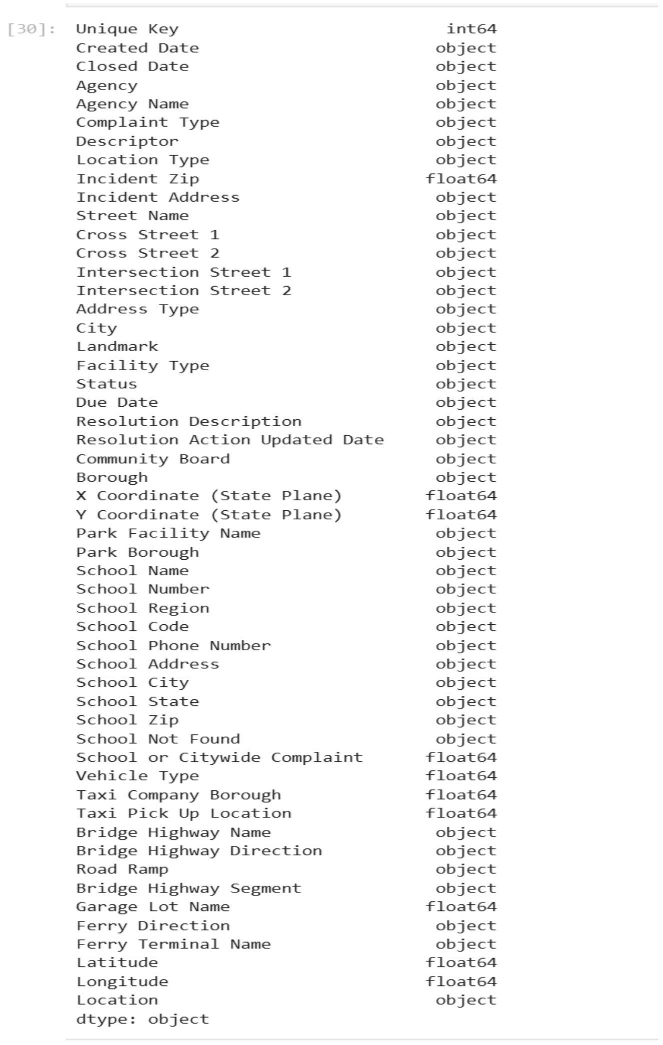
Code:



```
[30]: dframe.dtypes
```

Figure 7 code for data types of the data set

Output:



```
[30]: 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
```

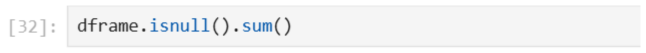
Figure 8 displaying the data types of data sets

Explanation

Pandas provides the dframe.dtypes attribute to show the data type for each column in a DataFrame. When you run this command, it returns a Series, where the index is column names and the value is the datatype, such as int64, float64, object, datetime64[ns], or timedelta64[ns]. This function is particularly valuable during the data preparation process, because it allows analysts to quickly visualize how the data is stored in each column and be able to determine if any of the columns need to be changed to the appropriate type for further analysis. For example, a date may be stored as a string and needs changed to the datetime format or a numeric field was detected as an object(str in pandas) by accident. When computing, visualizing, and doing statistical tests it is good to know your data is being measured with the proper data types.

2.2.4 Checking missing values of dataset

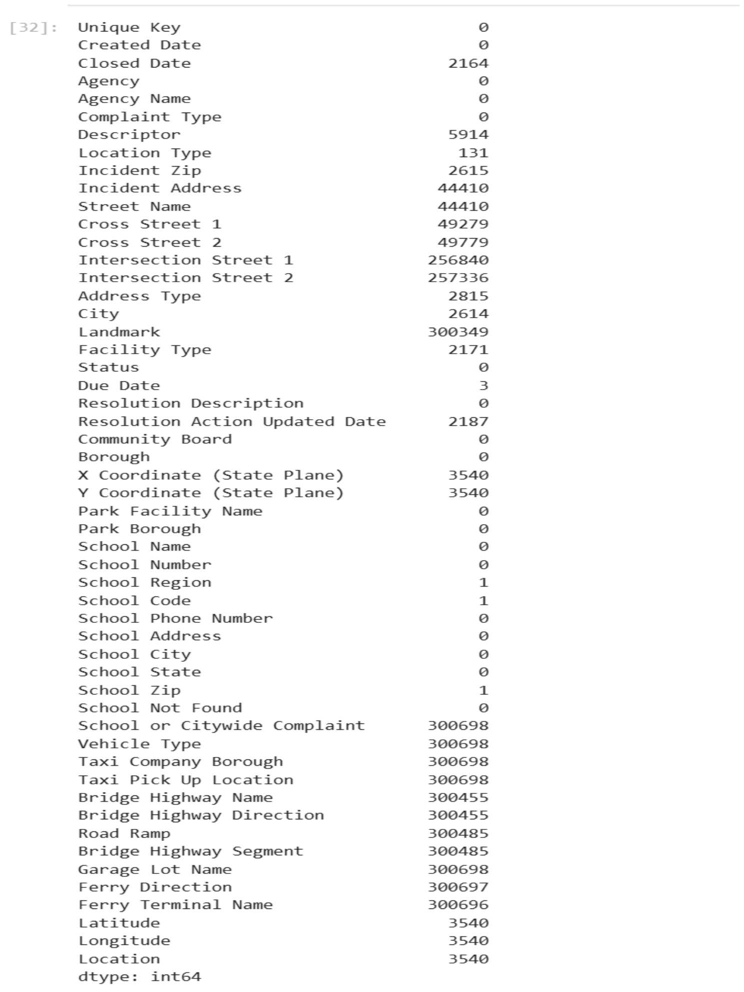
Code:



```
[32]: dframe.isnull().sum()
```

Figure 9 code for checking missing values of dataset

Output:



```
[32]: 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
```

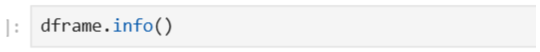
Figure 10 displaying missing values of the data set

Explanation

The dframe.isnull().sum() function in pandas is actually a two part action to locate and count the number of missing (null/NaN) values in each of the columns of a DataFrame. When isnull() is called, it returns a DataFrame of the same shape as the input, but with boolean values - True is there is no data and False is there is data. The chaining method .sum() then just adds up the number of True values for each column, which results in the total number of missing entries in every field. This step is very important to the data preparation / cleaning stage because this really helps the analyst quickly decide which columns are missing data, so that they can then assess overall the number of missing data points and decide whether to delete values, fill them, or ignore them based on how important the missing data is to the RESEARCHER' analysis.

2.2.5 Info of the Data set

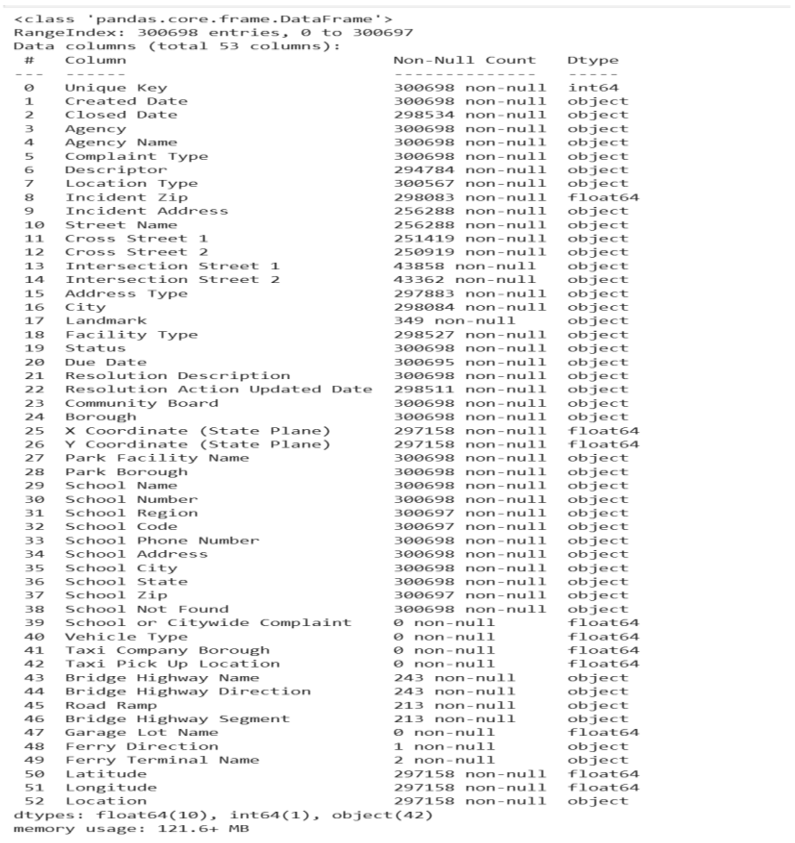
Code:



```
dframe.info()
```

Figure 11 code for info of dataset

Output:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300698 entries, 0 to 300697
Data columns (total 53 columns):
 #   Column                          Non-Null Count   Dtype  
---  ------                          --------------   -----  
 0   Unique Key                      300698 non-null  int64  
 1   Created Date                    300698 non-null  object 
 2   Closed Date                     298534 non-null  object 
 3   Agency                          300698 non-null  object 
 4   Agency Name                     300698 non-null  object 
 5   Complaint Type                  300698 non-null  object 
 6   Descriptor                      294784 non-null  object 
 7   Location Type                   300567 non-null  object 
 8   Incident Zip                    298083 non-null  float64
 9   Incident Address                256288 non-null  object 
 10  Street Name                     256288 non-null  object 
 11  Cross Street 1                  251419 non-null  object 
 12  Cross Street 2                  250919 non-null  object 
 13  Intersection Street 1           43858 non-null   object 
 14  Intersection Street 2           43362 non-null   object 
 15  Address Type                    297883 non-null  object 
 16  City                            298084 non-null  object 
 17  Landmark                        349 non-null     object 
 18  Facility Type                   298527 non-null  object 
 19  Status                          300698 non-null  object 
 20  Due Date                        300695 non-null  object 
 21  Resolution Description          300698 non-null  object 
 22  Resolution Action Updated Date  298511 non-null  object 
 23  Community Board                 300698 non-null  object 
 24  Borough                         300698 non-null  object 
 25  X Coordinate (State Plane)      297158 non-null  float64
 26  Y Coordinate (State Plane)      297158 non-null  float64
 27  Park Facility Name              300698 non-null  object 
 28  Park Borough                    300698 non-null  object 
 29  School Name                     300698 non-null  object 
 30  School Number                   300698 non-null  object 
 31  School Region                   300697 non-null  object 
 32  School Code                     300697 non-null  object 
 33  School Phone Number             300698 non-null  object 
 34  School Address                  300698 non-null  object 
 35  School City                     300698 non-null  object 
 36  School State                    300698 non-null  object 
 37  School Zip                      300697 non-null  object 
 38  School Not Found                300698 non-null  object 
 39  School or Citywide Complaint    0 non-null       float64
 40  Vehicle Type                    0 non-null       float64
 41  Taxi Company Borough            0 non-null       float64
 42  Taxi Pick Up Location           0 non-null       float64
 43  Bridge Highway Name             243 non-null     object 
 44  Bridge Highway Direction        243 non-null     object 
 45  Road Ramp                       213 non-null     object 
 46  Bridge Highway Segment          213 non-null     object 
 47  Garage Lot Name                 0 non-null       float64
 48  Ferry Direction                 1 non-null       object 
 49  Ferry Terminal Name             2 non-null       object 
 50  Latitude                        297158 non-null  float64
 51  Longitude                       297158 non-null  float64
 52  Location                        297158 non-null  object 
dtypes: float64(10), int64(1), object(42)
memory usage: 121.6+ MB
```

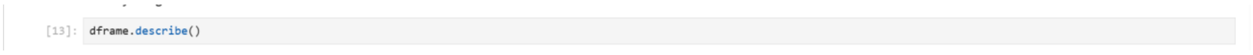
Figure 12 displaying info of dataset

Explanation

The info() function, dframe.info() in pandas, gives a quick summary of the structure of a DataFrame and its important metadata. The command will give the number of observations (rows), the number of variables (columns), and the name of each column with its corresponding data types. It will also show the number of non-null (non-missing) values in each column, which allows you to quickly see if you have missing data. info() also provides the memory usage of the DataFrame which allows analysts to understand the relative size of the dataset and what storage might be required if it were to be physically stored. The info() function is generally used to explore and understand the dataset when in the data understanding and initial explorations planning phase of data analysis to check completeness, detect data types, and give you an idea of what types of data have been stored in the dataset prior to centering in for any analysis or preprocessing.

2.2.6 Describing the dataset

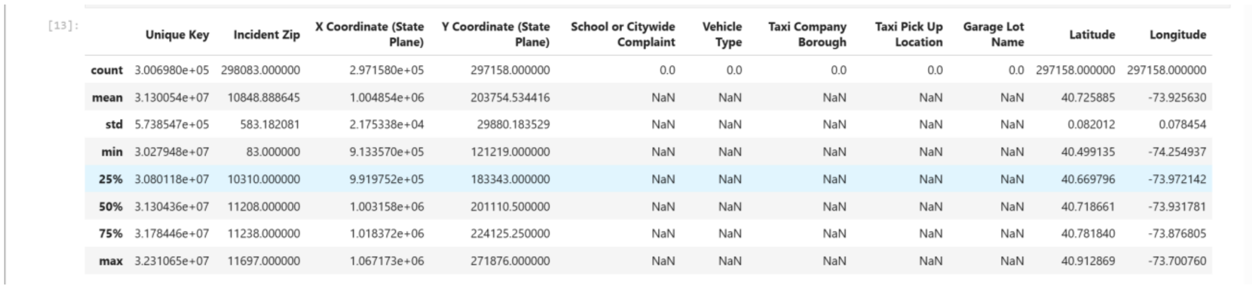
Code:



```
[13]: dframe.describe()
```

Figure 13: describing the dframe

Output:



[13]:

	Unique Key	Incident Zip	X Coordinate (State Plane)	Y Coordinate (State Plane)	School or Citywide Complaint	Vehicle Type	Taxi Company Borough	Taxi Pick Up Location	Garage Lot Name	Latitude	Longitude
count	3.006980e+05	298083.000000	2.971580e+05	297158.000000	0.0	0.0	0.0	0.0	0.0	297158.000000	297158.000000
mean	3.130054e+07	10848.888645	1.004854e+06	203754.534416	NaN	NaN	NaN	NaN	NaN	40.725885	-73.925630
std	5.738547e+05	583.182081	2.175338e+04	29880.183529	NaN	NaN	NaN	NaN	NaN	0.082012	0.078454
min	3.027948e+07	83.000000	9.133570e+05	121219.000000	NaN	NaN	NaN	NaN	NaN	40.499135	-74.254937
25%	3.080118e+07	10310.000000	9.919752e+05	183343.000000	NaN	NaN	NaN	NaN	NaN	40.669796	-73.972142
50%	3.130436e+07	11208.000000	1.003158e+06	201110.500000	NaN	NaN	NaN	NaN	NaN	40.718661	-73.931781
75%	3.178446e+07	11238.000000	1.018372e+06	224125.250000	NaN	NaN	NaN	NaN	NaN	40.781840	-73.876805
max	3.231065e+07	11697.000000	1.067173e+06	271876.000000	NaN	NaN	NaN	NaN	NaN	40.912869	-73.700760

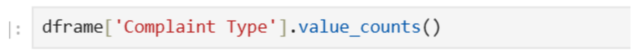
Figure 14: output of dframe.describe()

Explanation

The dframe.describe() function can be used to produce a summary of descriptive statistics for the numerical columns in the dataset. After this command is run, several descriptive statistics are calculated and displayed. Some of the statistics returned include the count, mean, std, min, max, the 25th percentile (Q1), 50th percentile (Q2, which is also the median), and the 75th percentile (Q3). The summary produces a useful and speedy overview of the distribution, spread, and central tendency of the data to identify abnormal values, potential outliers, and general directions of the data (beyond the zero-values) prior to further direct calculations.

2.2.7 Value count of the Data frame

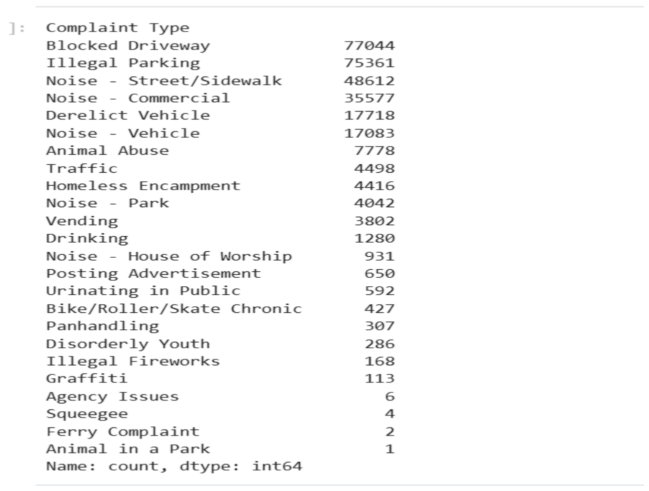
Code:



```
[ ]: dframe['Complaint Type'].value_counts()
```

Figure 15 code for value count of data frame

Output:



```
]: Complaint Type
   Blocked Driveway                77044
   Illegal Parking                 75361
   Noise - Street/Sidewalk         48612
   Noise - Commercial              35577
   Derelict Vehicle                17718
   Noise - Vehicle                 17083
   Animal Abuse                     7778
   Traffic                          4498
   Homeless Encampment              4416
   Noise - Park                     4042
   Vending                          3802
   Drinking                         1280
   Noise - House of Worship          931
   Posting Advertisement             650
   Urinating in Public               592
   Bike/Roller/Skate Chronic         427
   Panhandling                       307
   Disorderly Youth                  286
   Illegal Fireworks                 168
   Graffiti                          113
   Agency Issues                       6
   Squeegee                            4
   Ferry Complaint                     2
   Animal in a Park                    1
   Name: count, dtype: int64
```

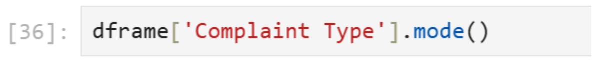
Figure 16 displaying value count of the data frame

Explanation

The pandas function dframe['Complaint Type'].value_counts() counts the number of occurrences of each unique value in the dataset in the ‘Complaint Type’ column. When you run this command, the return value will be a Series object. The index of the Series will correspond to the unique complaint types, and the value will always be how many times each type appears in the dataset, table sorted by default in descending order. This operation is most useful for determining the most frequently reported types of service requests/issues in the dataset I created. It will also provide analysts a basic understanding of the distribution of complaint types by providing the frequency of the complaints types, so analysts identify which types of complaints might need prioritized attention or other resource allocation decisions.

2.2.8 Checking the most appeared value in the column

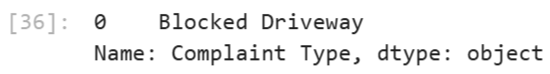
Code:



```
[36]: dframe['Complaint Type'].mode()
```

Figure 17 code for checking the most appeared value in particular column

Output:



```
[36]: 0    Blocked Driveway
      Name: Complaint Type, dtype: object
```

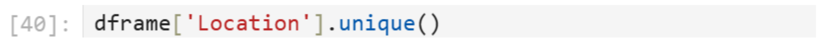
Figure 18 displaying the most appeared value in the particular column

Explanation

The function dframe['Complaint Type'].mode() is used in pandas to find the most common value(s) in the ‘Complaint Type’ column. Upon execution of the function, a Series is outputted containing the mode or a modes if there are multiple values having the maximum frequency. This function is useful for categorical data because it will indicate directly which category or service-needs appears the most frequent from the DataFrame. In the example of NYC 311, it will indicate the type of complaint that is most frequently called in by the residents, which is valuable information for determining the level of priority attention (ex. often need to keep police on certain resources to check against.

2.2.9 Checking unique values of the column

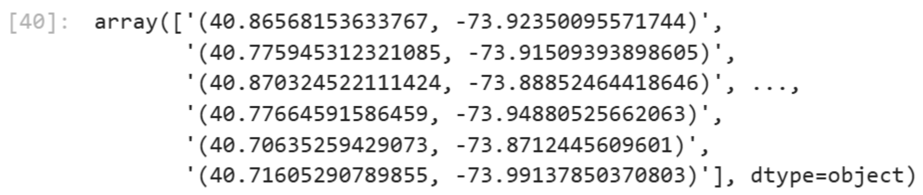
Code:



```
[40]: dframe['Location'].unique()
```

Figure 19checking unique value for particular column

Output:



```
[40]:  array(['(40.86568153633767, -73.92350095571744)',
               '(40.775945312321085, -73.91509393898605)',
               '(40.870324522111424, -73.88852464418646)', ...,
               '(40.77664591586459, -73.94880525662063)',
               '(40.70635259429073, -73.8712445609601)',
               '(40.71605290789855, -73.99137850370803)'], dtype=object)
```

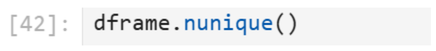
Figure 20 displaying unique value for particular column

Explanation

The function dframe['Location'].unique() outputs a NumPy array of all the unique value(s) in the 'Location' in the DataFrame. Upon execution, it will scan through the entire column to find all unique values with no repetition. This function is valuable during the data understanding/exploration phase because it will help analyst think through the range of unique location entries dataset. It can help analyst find inconsistencies, unexpected values or unusual values that perhaps need cleaning or correction before one gets into the in-depth analysis or to visualize.

2.2.10 Checking the number of unique values in the column

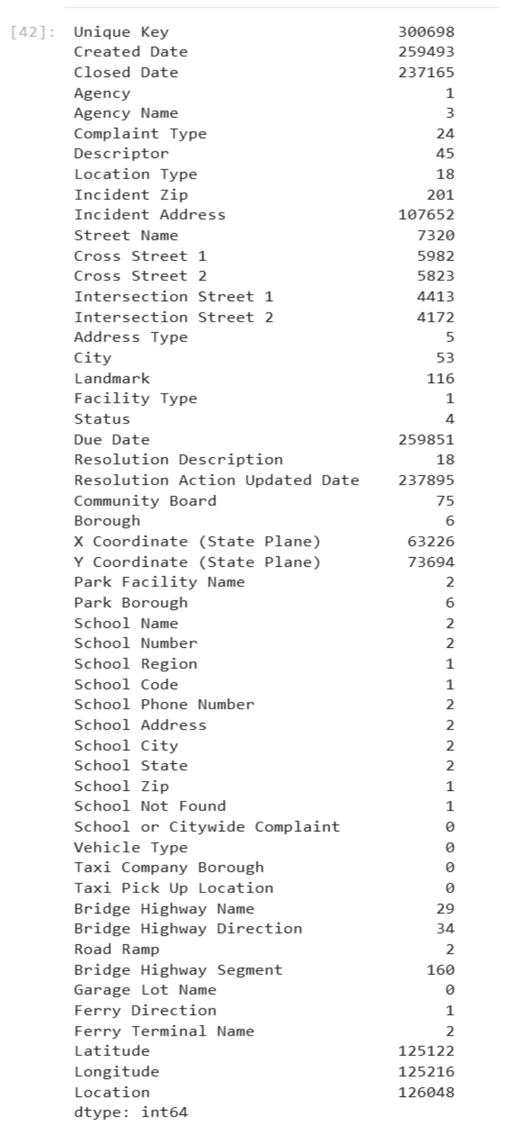
Code:



```
[42]: dframe.nunique()
```

Figure 21 checking the number of unique value in the column

Output:



```
[42]: 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
      dtype: int64
```

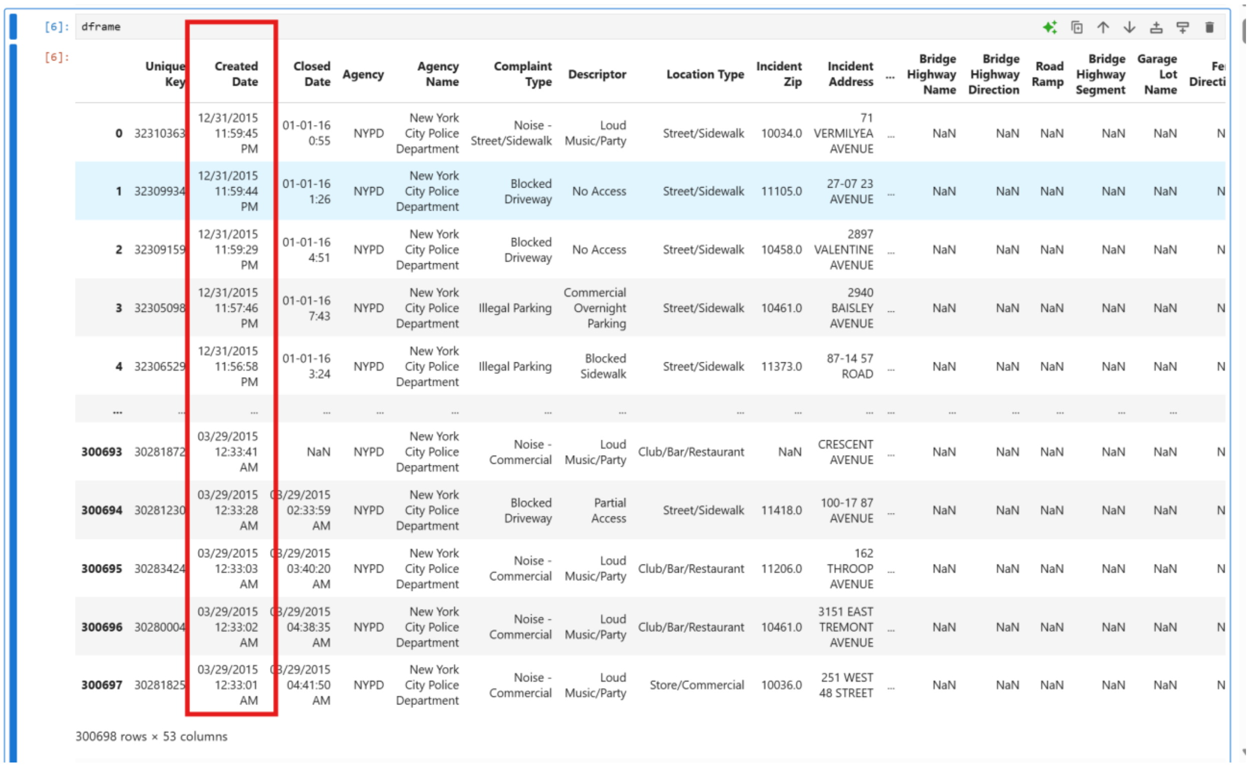
Figure 22 displaying the number of unique values in columns

Explanation

The dframe.nunique() function in pandas helps to get the count of the unique, distinct values in each column of a DataFrame. When we call this function, it returns a Series which has the column names as the index and the total number of unique entries in each column as its values. This is a way to quickly see how much diversity, or variability is in each feature of the dataset. It is a great method to find categorical variables with low cardinality (for example, many unique values), as well as columns that have a high number of unique entries like IDs, timestamps.

2.3 Question no.3: converting created date and closed date to datetime and creating a new column request closing time

Column before converting created date



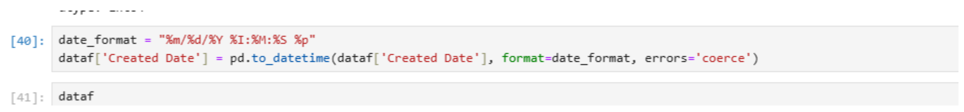
```
[6]: dframe
```

[6]:

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	...	Bridge Highway Name	Bridge Highway Direction	Road Ramp	Bridge Highway Segment	Garage Lot Name	Fe Directi
0	32310363	12/31/2015 11:59:45 PM	01-01-16 0:55	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	10034.0	71 VERMILYEA AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
1	32309934	12/31/2015 11:59:44 PM	01-01-16 1:26	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	11105.0	27-07 23 AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
2	32309159	12/31/2015 11:59:29 PM	01-01-16 4:51	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	10458.0	2897 VALENTINE AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
3	32305098	12/31/2015 11:57:46 PM	01-01-16 7:43	NYPD	New York City Police Department	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk	10461.0	2940 BAISLEY AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
4	32306529	12/31/2015 11:56:58 PM	01-01-16 3:24	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11373.0	87-14 57 ROAD	...	NaN	NaN	NaN	NaN	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
300693	30281872	03/29/2015 12:33:41 AM	NaN	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	NaN	CRESCENT AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
300694	30281230	03/29/2015 12:33:28 AM	03/29/2015 02:33:59 AM	NYPD	New York City Police Department	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	100-17 87 AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
300695	30283424	03/29/2015 12:33:03 AM	03/29/2015 03:40:20 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	162 THROOP AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
300696	30280004	03/29/2015 12:33:02 AM	03/29/2015 04:38:35 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10461.0	3151 EAST TREMONT AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
300697	30281825	03/29/2015 12:33:01 AM	03/29/2015 04:41:50 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Store/Commercial	10036.0	251 WEST 48 STREET	...	NaN	NaN	NaN	NaN	NaN	N

300698 rows × 53 columns

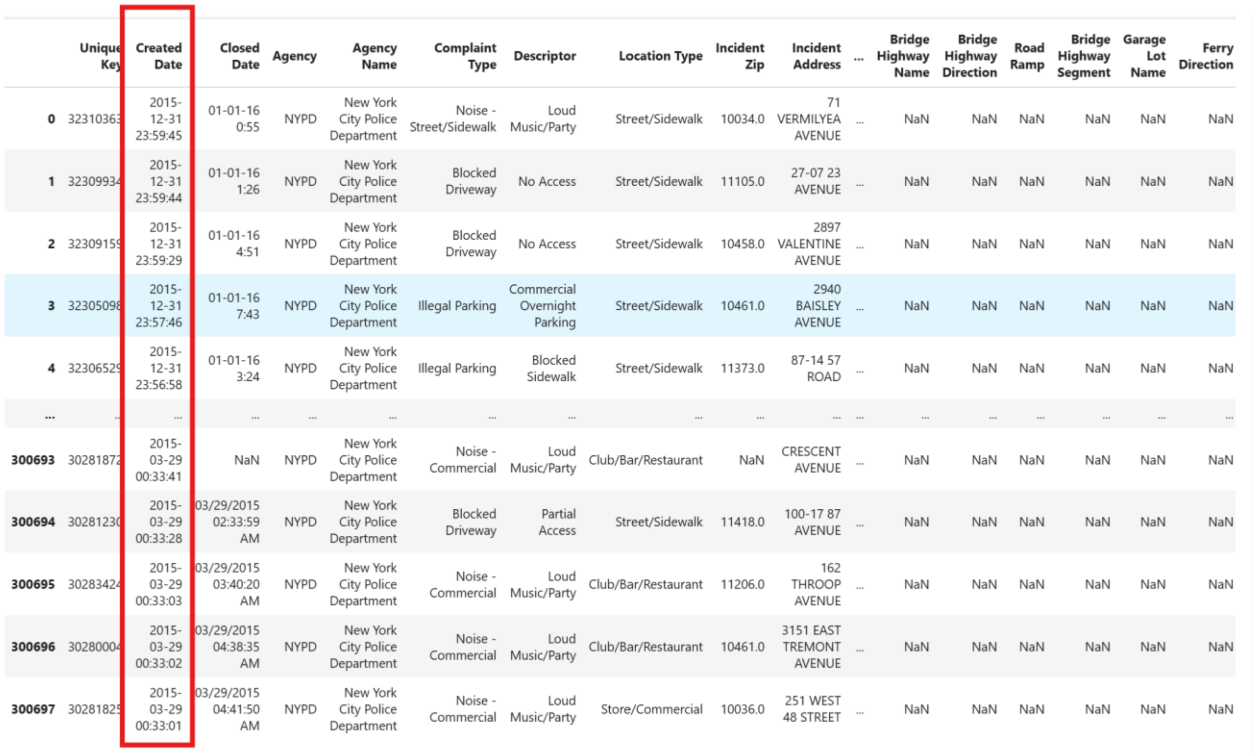
Code:



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

[41]: dataf
```

Figure 23 create date for create date format

Output:



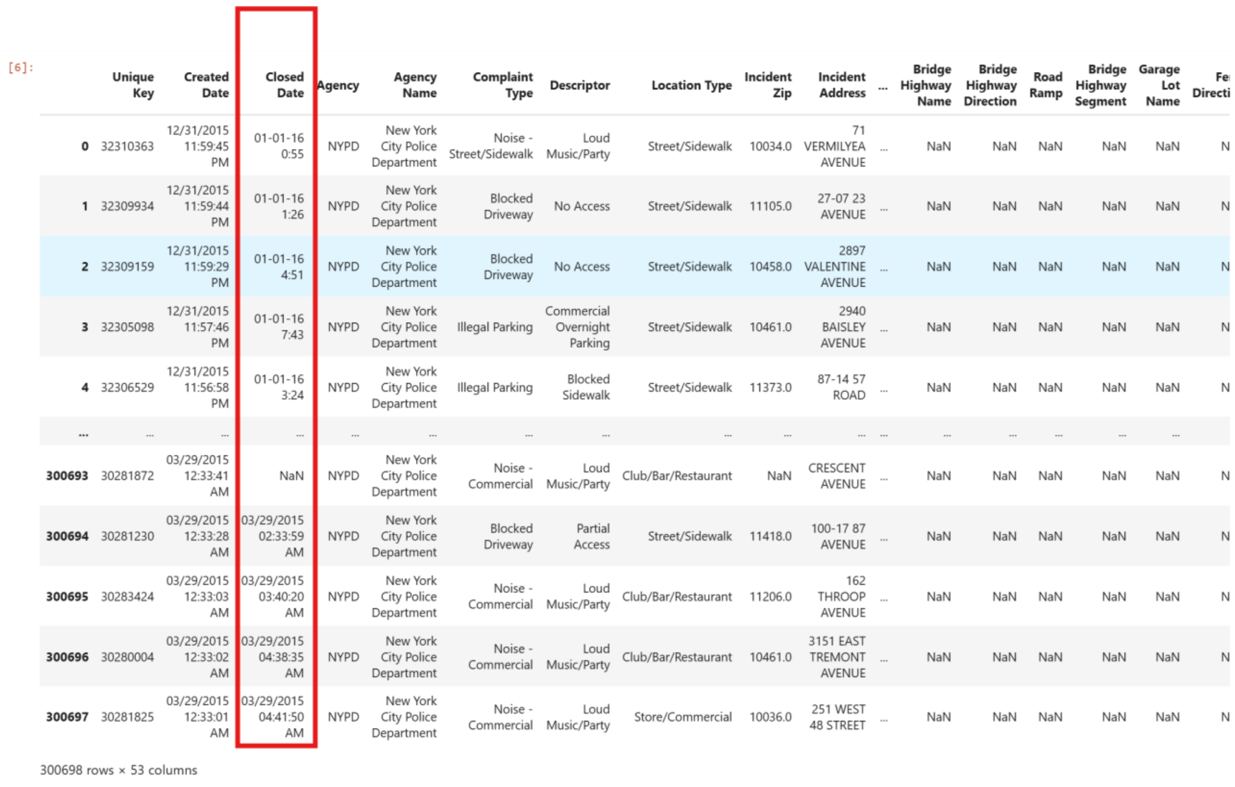
	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	...	Bridge Highway Name	Bridge Highway Direction	Road Ramp	Bridge Highway Segment	Garage Lot Name	Ferry Direction
0	32310363	2015-12-31 23:59:45	01-01-16 0:55	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	10034.0	71 VERMILYEA AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN
1	32309934	2015-12-31 23:59:44	01-01-16 1:26	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	11105.0	27-07 23 AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN
2	32309159	2015-12-31 23:59:29	01-01-16 4:51	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	10458.0	2897 VALENTINE AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN
3	32305098	2015-12-31 23:57:46	01-01-16 7:43	NYPD	New York City Police Department	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk	10461.0	2940 BAISLEY AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN
4	32306529	2015-12-31 23:56:58	01-01-16 3:24	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11373.0	87-14 57 ROAD	...	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
300693	30281872	2015-03-29 00:33:41	NaN	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	NaN	CRESCENT AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN
300694	30281230	2015-03-29 00:33:28	03/29/2015 02:33:59 AM	NYPD	New York City Police Department	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	100-17 87 AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN
300695	30283424	2015-03-29 00:33:03	03/29/2015 03:40:20 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	162 THROOP AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN
300696	30280004	2015-03-29 00:33:02	03/29/2015 04:38:35 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10461.0	3151 EAST TREMONT AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN
300697	30281825	2015-03-29 00:33:01	03/29/2015 04:41:50 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Store/Commercial	10036.0	251 WEST 48 STREET	...	NaN	NaN	NaN	NaN	NaN	NaN

Figure 24 displaying create date format

Explanation:

This code will convert the 'Created Date' columns of a DataFrame into a date as an actual date using the pandas to_datetime() method. The variable date_format specifies the format that the date string is in which represents: "%m/%d/%Y %I:%M:%S %p". So, it expected a date in the format "month/day/year hour:minute:second AM/PM" - (i.e., 04/15/2023 02:45:30 PM). Once the format was passed on to the to_datetime() function, pandas will be able to correctly parse the string into a datetime object. The errors='coerce' argument also ensured that any value that did not conform to the given format would be turned into Not a Time (NaT) so the program would not crash because of a failed formatting. This conversion into datetime is important because it is impossible to perform any calculations or filtering based on time (e.g., calculating durations, ordering records) without converting these values into a datetime format.

Column before converting closed da



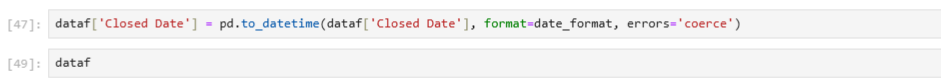
[6]:

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	...	Bridge Highway Name	Bridge Highway Direction	Road Ramp	Bridge Highway Segment	Garage Lot Name	Fe Directi
0	32310363	12/31/2015 11:59:45 PM	01-01-16 0:55	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	10034.0	71 VERMILYEA AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
1	32309934	12/31/2015 11:59:44 PM	01-01-16 1:26	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	11105.0	27-07 23 AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
2	32309159	12/31/2015 11:59:29 PM	01-01-16 4:51	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	10458.0	2897 VALENTINE AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
3	32305098	12/31/2015 11:57:46 PM	01-01-16 7:43	NYPD	New York City Police Department	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk	10461.0	2940 BAISLEY AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
4	32306529	12/31/2015 11:56:58 PM	01-01-16 3:24	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11373.0	87-14 57 ROAD	...	NaN	NaN	NaN	NaN	NaN	N
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
300693	30281872	03/29/2015 12:33:41 AM	NaN	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	NaN	CRESCENT AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
300694	30281230	03/29/2015 12:33:28 AM	03/29/2015 02:33:59 AM	NYPD	New York City Police Department	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	100-17 87 AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
300695	30283424	03/29/2015 12:33:03 AM	03/29/2015 03:40:20 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	162 THROOP AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
300696	30280004	03/29/2015 12:33:02 AM	03/29/2015 04:38:35 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10461.0	3151 EAST TREMONT AVENUE	...	NaN	NaN	NaN	NaN	NaN	N
300697	30281825	03/29/2015 12:33:01 AM	03/29/2015 04:41:50 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Store/Commercial	10036.0	251 WEST 48 STREET	...	NaN	NaN	NaN	NaN	NaN	N

300698 rows × 53 columns

Figure 25 column before converting the closed date

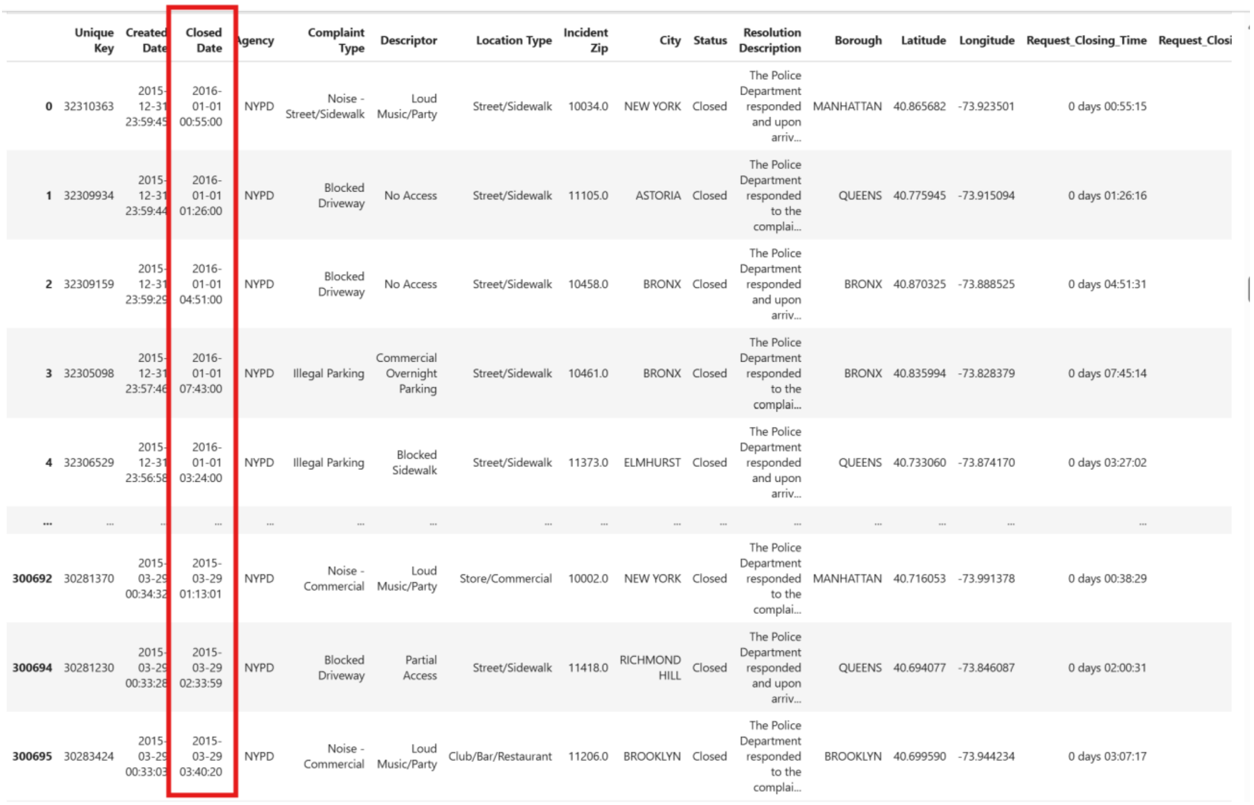
Code:



```
[47]: dataf['Closed Date'] = pd.to_datetime(dataf['Closed Date'], format=date_format, errors='coerce')

[49]: dataf
```

Figure 26 code for closed date

Output:



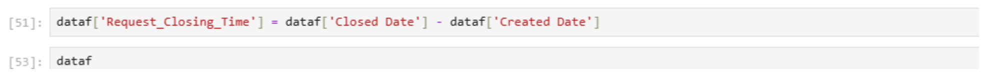
	Unique Key	Created Date	Closed Date	Agency	Complaint Type	Descriptor	Location Type	Incident Zip	City	Status	Resolution Description	Borough	Latitude	Longitude	Request_Closing_Time	Request_Closi
0	32310363	2015-12-31 23:59:45	2016-01-01 00:55:00	NYPD	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk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

Figure 27 Displaying closed date format

Explanation:

This code is intended to change the 'Created Date' column in a DataFrame to a truly datetime format by using the pandas.to_datetime() function. The variable date_format explains the format of date strings it expects if you think about the meaning of the symbols, it would indicate that the date is like "month/day/year hour:minute:second AM or PM" (e.g. "04/15/2023 02:45:30 PM"). By supplying to to_datetime() the proper format as defined by date_format pandas can parse the string correctly to a datetime object. The errors='coerce' argument handles anything other than the datetime format into a NaT value (Not a Timestamp), which prevents the program from crashing when one of the values does not follow the same format. This conversion is required for doing anything with time-based calculations or filtering such as calculating duration or sorting records by date.

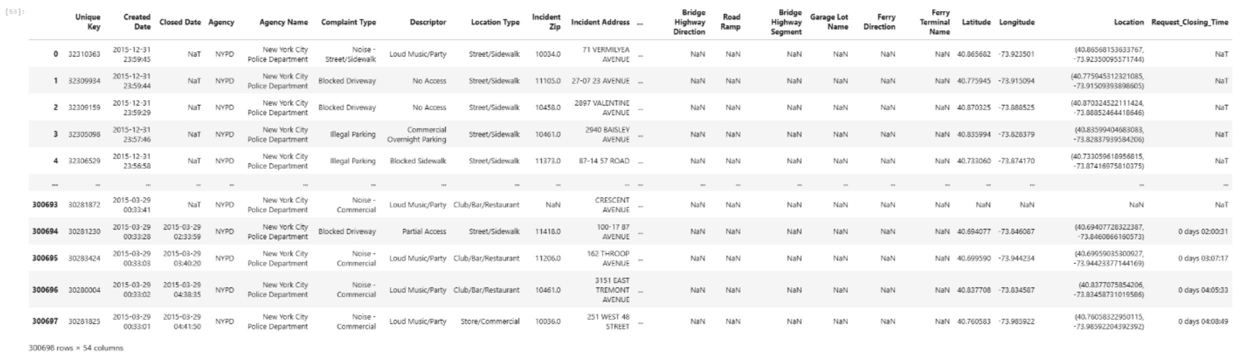
Code:



```
[51]: dataf['Request_Closing_Time'] = dataf['Closed Date'] - dataf['Created Date']

[53]: dataf
```

Figure 28 code for creating request closing time

Output:



[53]:

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	...	Bridge Highway Direction	Road Ramp	Bridge Highway Segment	Garage Lot Name	Ferry Direction	Ferry Terminal Name	Latitude	Longitude	Location	Request_Closing_Time
0	32310363	2015-12-31 23:59:45	NaT	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	10034.0	71 VERMILYEA AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	40.865682	-73.923501	(40.86568153633767, -73.92350095571744)	NaT
1	32309934	2015-12-31 23:59:44	NaT	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	11105.0	27-07 23 AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	40.775945	-73.915094	(40.775945312321085, -73.91509393898605)	NaT
2	32309159	2015-12-31 23:59:29	NaT	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	10458.0	2897 VALENTINE AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	40.870325	-73.888525	(40.870324522111424, -73.88852464418646)	NaT
3	32305098	2015-12-31 23:57:46	NaT	NYPD	New York City Police Department	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk	10461.0	2940 BAISLEY AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	40.835994	-73.828379	(40.83599404683083, -73.82837939584206)	NaT
4	32306529	2015-12-31 23:56:58	NaT	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11373.0	87-14 57 ROAD	...	NaN	NaN	NaN	NaN	NaN	NaN	40.733060	-73.874170	(40.733059618956815, -73.87416975810375)	NaT
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
300693	30281872	2015-03-29 00:33:41	NaT	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	NaN	CRESCENT AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaT
300694	30281230	2015-03-29 00:33:28	2015-03-29 02:33:59	NYPD	New York City Police Department	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	100-17 87 AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	40.694077	-73.846087	(40.69407728322387, -73.8460866160573)	0 days 02:00:31
300695	30283424	2015-03-29 00:33:03	2015-03-29 03:40:20	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	162 THROOP AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	40.699590	-73.944234	(40.69959035300927, -73.94423377144169)	0 days 03:07:17
300696	30280004	2015-03-29 00:33:02	2015-03-29 04:38:35	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10461.0	3151 EAST TREMONT AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	40.837708	-73.834587	(40.8377075854206, -73.83458731019586)	0 days 04:05:33
300697	30281825	2015-03-29 00:33:01	2015-03-29 04:41:50	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Store/Commercial	10036.0	251 WEST 48 STREET	...	NaN	NaN	NaN	NaN	NaN	NaN	40.760583	-73.985922	(40.76058322950115, -73.98592204392392)	0 days 04:08:49

300698 rows × 54 columns

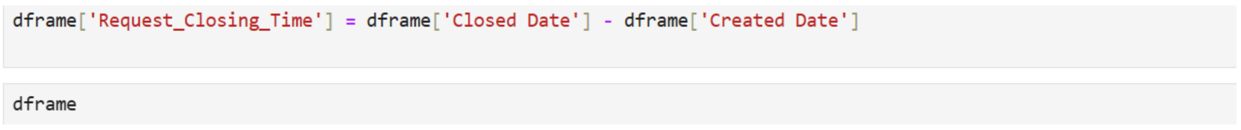
Figure 29 displaying request closing time

Explanation:

Using the method here of, the specified column "Request_Closing_Time" is created by the difference between the closed date and the created date. The associated value gives us the time during which the request was created and closed, indicating the time length taken to resolve the request.

Creating new column “Request_Closing_time”

Code:

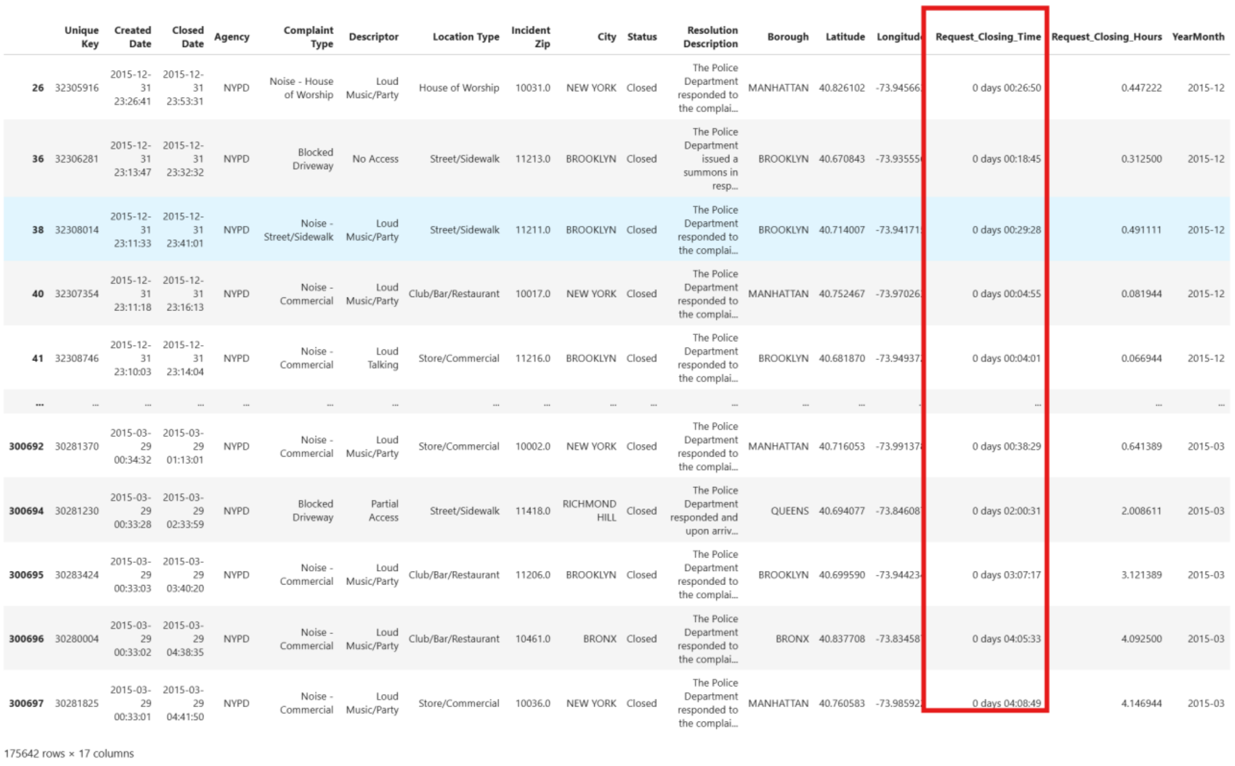


```
dframe['Request_Closing_Time'] = dframe['Closed Date'] - dframe['Created Date']
```

```
dframe
```

Figure 30 code for request closing time table

Output:



	Unique Key	Created Date	Closed Date	Agency	Complaint Type	Descriptor	Location Type	Incident Zip	City	Status	Resolution Description	Borough	Latitude	Longitud	Request_Closing_Time	Request_Closing_Hours	YearMonth
26	32305916	2015-12-31 23:26:41	2015-12-31 23:53:31	NYPD	Noise - House of Worship	Loud Music/Party	House of Worship	10031.0	NEW YORK	Closed	The Police Department responded to the complai...	MANHATTAN	40.826102	-73.94566	0 days 00:26:50	0.447222	2015-12
36	32306281	2015-12-31 23:13:47	2015-12-31 23:32:32	NYPD	Blocked Driveway	No Access	Street/Sidewalk	11213.0	BROOKLYN	Closed	The Police Department issued a summons in resp...	BROOKLYN	40.670843	-73.93555	0 days 00:18:45	0.312500	2015-12
38	32308014	2015-12-31 23:11:33	2015-12-31 23:41:01	NYPD	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	11211.0	BROOKLYN	Closed	The Police Department responded to the complai...	BROOKLYN	40.714007	-73.94171	0 days 00:29:28	0.491111	2015-12
40	32307354	2015-12-31 23:11:18	2015-12-31 23:16:13	NYPD	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10017.0	NEW YORK	Closed	The Police Department responded to the complai...	MANHATTAN	40.752467	-73.97026	0 days 00:04:55	0.081944	2015-12
41	32308746	2015-12-31 23:10:03	2015-12-31 23:14:04	NYPD	Noise - Commercial	Loud Talking	Store/Commercial	11216.0	BROOKLYN	Closed	The Police Department responded to the complai...	BROOKLYN	40.681870	-73.94937	0 days 00:04:01	0.066944	2015-12
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
300692	30281370	2015-03-29 00:34:32	2015-03-29 01:13:01	NYPD	Noise - Commercial	Loud Music/Party	Store/Commercial	10002.0	NEW YORK	Closed	The Police Department responded to the complai...	MANHATTAN	40.716053	-73.99137	0 days 00:38:29	0.641389	2015-03
300694	30281230	2015-03-29 00:33:28	2015-03-29 02:33:59	NYPD	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	RICHMOND HILL	Closed	The Police Department responded and upon arriv...	QUEENS	40.694077	-73.84608	0 days 02:00:31	2.008611	2015-03
300695	30283424	2015-03-29 00:33:03	2015-03-29 03:40:20	NYPD	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	BROOKLYN	Closed	The Police Department responded to the complai...	BROOKLYN	40.699590	-73.94423	0 days 03:07:17	3.121389	2015-03
300696	30280004	2015-03-29 00:33:02	2015-03-29 04:38:35	NYPD	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10461.0	BRONX	Closed	The Police Department responded to the complai...	BRONX	40.837708	-73.83458	0 days 04:05:33	4.092500	2015-03
300697	30281825	2015-03-29 00:33:01	2015-03-29 04:41:50	NYPD	Noise - Commercial	Loud Music/Party	Store/Commercial	10036.0	NEW YORK	Closed	The Police Department responded to the complai...	MANHATTAN	40.760583	-73.98592	0 days 04:08:49	4.146944	2015-03

175642 rows × 17 columns

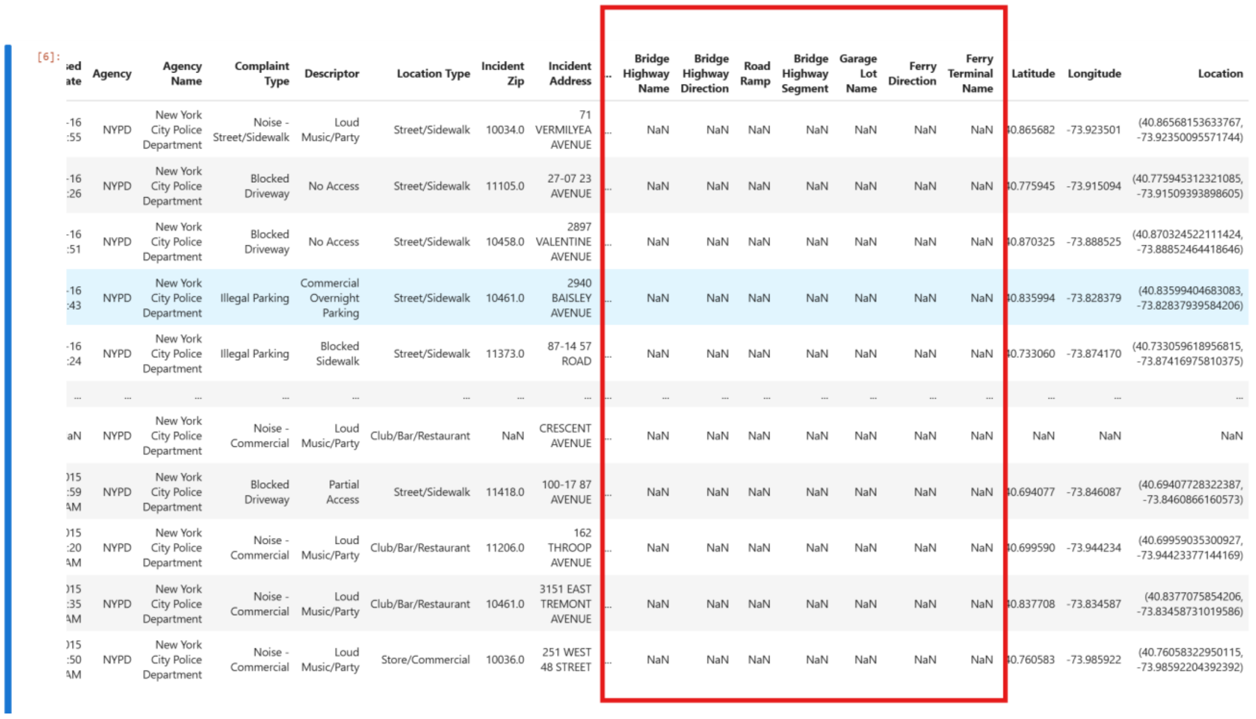
Figure 31 displaying request closing tine in table

Explanation:

The line dframe['Request_Closing_Time'] = dframe['Closed Date'] - dframe['Created Date'] computes the time duration of each complaint by taking the close timestamp and subtracting the creation timestamp. It essentially creates a new column called Request_Closing_Time that contains the duration of each request expressed as a duration in the form of a timedelta object. This new derived column is important to analyze efficiency of service and notice how long certain complaints take to be resolved based on different categories. It allows for other time-based analysis, such as computing averages, or comparing closing times between different types of complaints and boroughs.

2.4 Question no.4: dropping the column

Table before dropping the columns

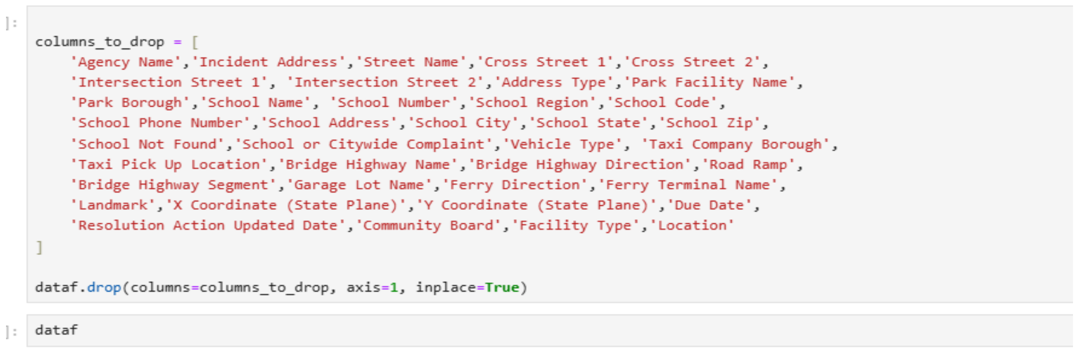


[6]:

ed ate	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	...	Bridge Highway Name	Bridge Highway Direction	Road Ramp	Bridge Highway Segment	Garage Lot Name	Ferry Direction	Ferry Terminal Name	Latitude	Longitude	Location
-16 :55	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	10034.0	71 VERMILYEA AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.865682	-73.923501	(40.86568153633767, -73.92350095571744)
-16 :26	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	11105.0	27-07 23 AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.775945	-73.915094	(40.775945312321085, -73.91509393898605)
-16 :51	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	10458.0	2897 VALENTINE AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.870325	-73.888525	(40.870324522111424, -73.88852464418646)
-16 :43	NYPD	New York City Police Department	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk	10461.0	2940 BAISLEY AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.835994	-73.828379	(40.83599404683083, -73.82837939584206)
-16 :24	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11373.0	87-14 57 ROAD	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.733060	-73.874170	(40.733059618956815, -73.87416975810375)
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
aN	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	NaN	CRESCENT AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
)15 :59 AM	NYPD	New York City Police Department	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	100-17 87 AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.694077	-73.846087	(40.69407728322387, -73.8460866160573)
)15 :20 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	162 THROOP AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.699590	-73.944234	(40.69959035300927, -73.94423377144169)
)15 :35 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10461.0	3151 EAST TREMONT AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.837708	-73.834587	(40.8377075854206, -73.83458731019586)
)15 :50 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Store/Commercial	10036.0	251 WEST 48 STREET	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	40.760583	-73.985922	(40.76058322950115, -73.98592204392392)

Figure 32 before dropping table

Code:

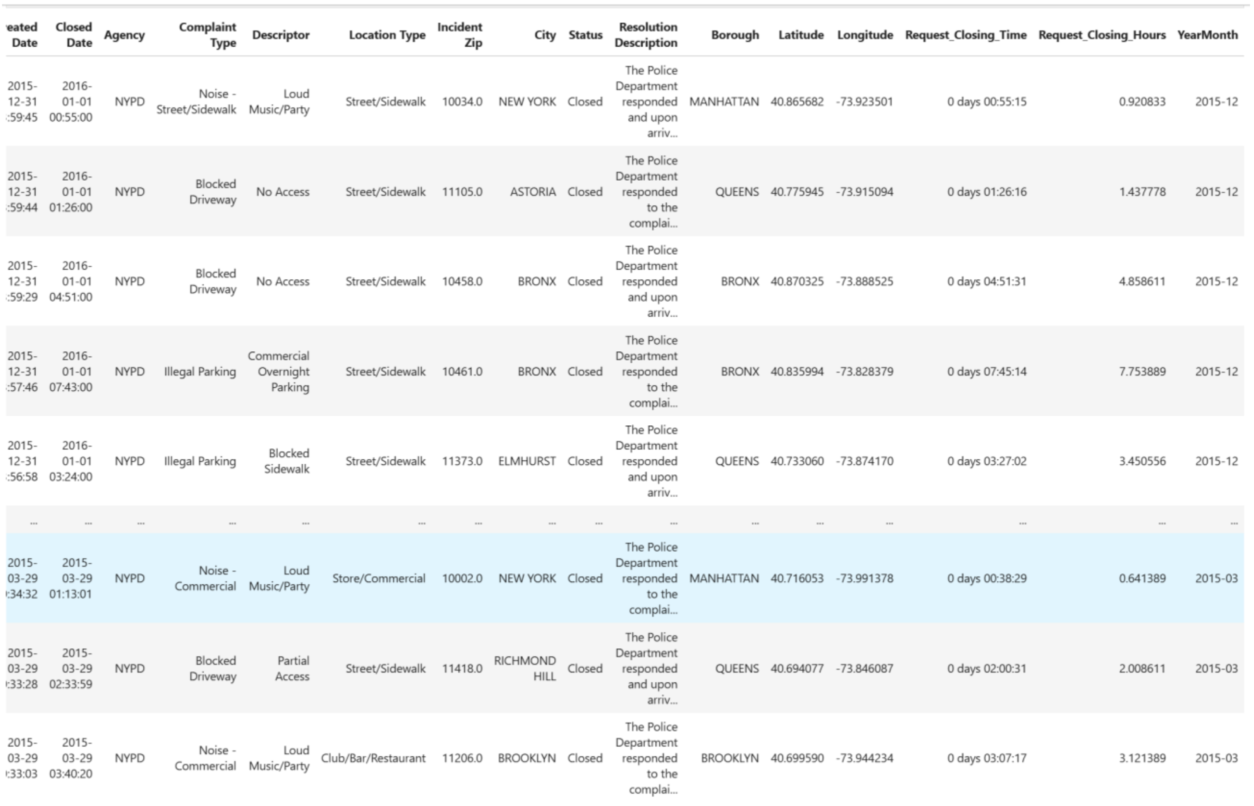


```
]:
columns_to_drop = [
    'Agency Name','Incident Address','Street Name','Cross Street 1','Cross Street 2',
    'Intersection Street 1', 'Intersection Street 2','Address Type','Park Facility Name',
    'Park Borough','School Name', 'School Number','School Region','School Code',
    'School Phone 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'
]

dataf.drop(columns=columns_to_drop, axis=1, inplace=True)
```

```
]: dataf
```

Figure 33 Code for drop column

Output:



eated Date	Closed Date	Agency	Complaint Type	Descriptor	Location Type	Incident Zip	City	Status	Resolution Description	Borough	Latitude	Longitude	Request_Closing_Time	Request_Closing_Hours	YearMonth
2015-12-31 :59:45	2016-01-01 00:55:00	NYPD	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	10034.0	NEW YORK	Closed	The Police Department responded and upon arriv...	MANHATTAN	40.865682	-73.923501	0 days 00:55:15	0.920833	2015-12
2015-12-31 :59:44	2016-01-01 01:26:00	NYPD	Blocked Driveway	No Access	Street/Sidewalk	11105.0	ASTORIA	Closed	The Police Department responded to the complai...	QUEENS	40.775945	-73.915094	0 days 01:26:16	1.437778	2015-12
2015-12-31 :59:29	2016-01-01 04:51:00	NYPD	Blocked Driveway	No Access	Street/Sidewalk	10458.0	BRONX	Closed	The Police Department responded and upon arriv...	BRONX	40.870325	-73.888525	0 days 04:51:31	4.858611	2015-12
2015-12-31 :57:46	2016-01-01 07:43:00	NYPD	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk	10461.0	BRONX	Closed	The Police Department responded to the complai...	BRONX	40.835994	-73.828379	0 days 07:45:14	7.753889	2015-12
2015-12-31 :56:58	2016-01-01 03:24:00	NYPD	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11373.0	ELMHURST	Closed	The Police Department responded and upon arriv...	QUEENS	40.733060	-73.874170	0 days 03:27:02	3.450556	2015-12
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2015-03-29 :34:32	2015-03-29 01:13:01	NYPD	Noise - Commercial	Loud Music/Party	Store/Commercial	10002.0	NEW YORK	Closed	The Police Department responded to the complai...	MANHATTAN	40.716053	-73.991378	0 days 00:38:29	0.641389	2015-03
2015-03-29 :33:28	2015-03-29 02:33:59	NYPD	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	RICHMOND HILL	Closed	The Police Department responded and upon arriv...	QUEENS	40.694077	-73.846087	0 days 02:00:31	2.008611	2015-03
2015-03-29 :33:03	2015-03-29 03:40:20	NYPD	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	BROOKLYN	Closed	The Police Department responded to the complai...	BROOKLYN	40.699590	-73.944234	0 days 03:07:17	3.121389	2015-03

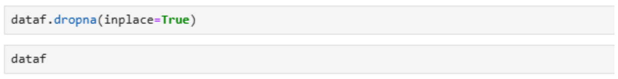
Figure 34 displaying table after drop

Explanation:

This code is used to clean the dataset (i.e., the "dframe" DataFrame) and therefore remove an unnecessary set of or irrelevant columns from the dataset. A list called "columns_to_drop" is defined, which contains names of columns that do not contain any useful data for analysis (i.e., specific address parts, school parts, geographic coordinate parts, spatial metadata, and other types of structures). The drop() function on dframe is invoked to remove these columns from the Dataframe. The axis=1 parents indicate that they are dropping columns (as opposed to rows) and that by including inplace=True, the original dframe is changed before being assigned to a new object. By dropping these columns, it reduces noise in the dataset and performance and readability in the following code by eliminating unnecessary structures or columns. Thus, this performed the standard cleaning that is often required when dealing with datsets.

2.5. Question no.5: removing the NaN values

Code:

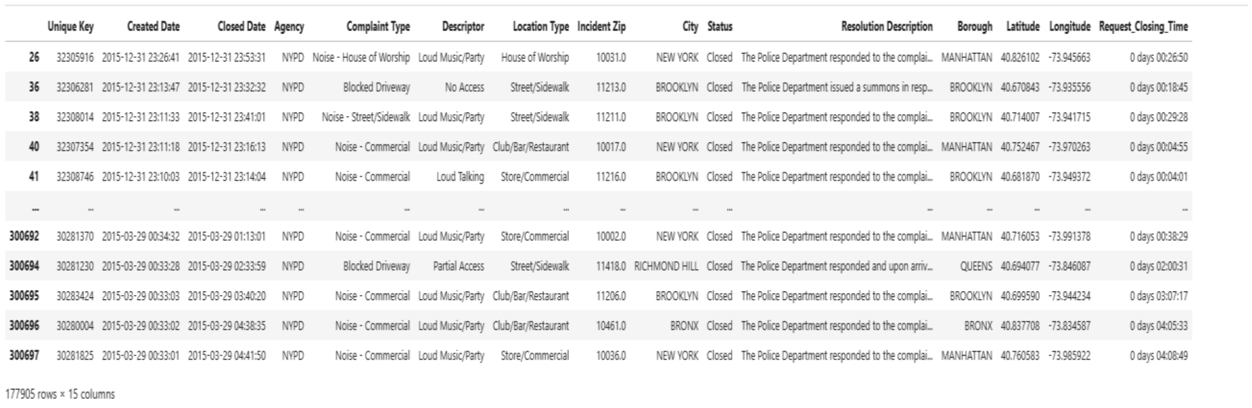


```
dataf.dropna(inplace=True)
```

```
dataf
```

Figure 35 code for missing values

Output:

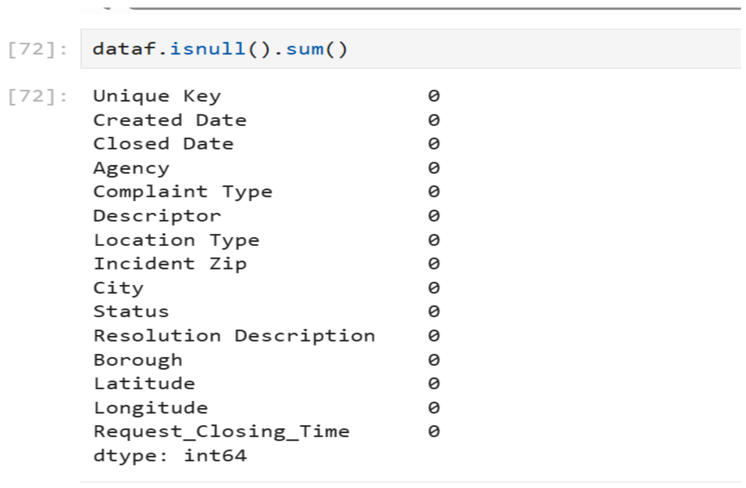


	Unique Key	Created Date	Closed Date	Agency	Complaint Type	Descriptor	Location Type	Incident Zip	City	Status	Resolution Description	Borough	Latitude	Longitude	Request_Closing_Time
26	32305916	2015-12-31 23:26:41	2015-12-31 23:53:31	NYPD	Noise - House of Worship	Loud Music/Party	House of Worship	10031.0	NEW YORK	Closed	The Police Department responded to the complai...	MANHATTAN	40.826102	-73.945663	0 days 00:26:50
36	32306281	2015-12-31 23:13:47	2015-12-31 23:32:32	NYPD	Blocked Driveway	No Access	Street/Sidewalk	11213.0	BROOKLYN	Closed	The Police Department issued a summons in resp...	BROOKLYN	40.670843	-73.935556	0 days 00:18:45
38	32308014	2015-12-31 23:11:33	2015-12-31 23:41:01	NYPD	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	11211.0	BROOKLYN	Closed	The Police Department responded to the complai...	BROOKLYN	40.714007	-73.941715	0 days 00:29:28
40	32307354	2015-12-31 23:11:18	2015-12-31 23:16:13	NYPD	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10017.0	NEW YORK	Closed	The Police Department responded to the complai...	MANHATTAN	40.752467	-73.970263	0 days 00:04:55
41	32308746	2015-12-31 23:10:03	2015-12-31 23:14:04	NYPD	Noise - Commercial	Loud Talking	Store/Commercial	11216.0	BROOKLYN	Closed	The Police Department responded to the complai...	BROOKLYN	40.681870	-73.949372	0 days 00:04:01
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
300692	30281370	2015-03-29 00:34:32	2015-03-29 01:13:01	NYPD	Noise - Commercial	Loud Music/Party	Store/Commercial	10002.0	NEW YORK	Closed	The Police Department responded to the complai...	MANHATTAN	40.716053	-73.991378	0 days 00:38:29
300694	30281230	2015-03-29 00:33:28	2015-03-29 02:33:59	NYPD	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	RICHMOND HILL	Closed	The Police Department responded and upon arriv...	QUEENS	40.694077	-73.846087	0 days 02:00:31
300695	30283424	2015-03-29 00:33:03	2015-03-29 03:40:20	NYPD	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	BROOKLYN	Closed	The Police Department responded to the complai...	BROOKLYN	40.699590	-73.944234	0 days 03:07:17
300696	30280004	2015-03-29 00:33:02	2015-03-29 04:38:35	NYPD	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10461.0	BRONX	Closed	The Police Department responded to the complai...	BRONX	40.837708	-73.834587	0 days 04:05:33
300697	30281825	2015-03-29 00:33:01	2015-03-29 04:41:50	NYPD	Noise - Commercial	Loud Music/Party	Store/Commercial	10036.0	NEW YORK	Closed	The Police Department responded to the complai...	MANHATTAN	40.760583	-73.985922	0 days 04:08:49

177905 rows × 15 columns

Figure 36 displaying missing values

After removing the missing values



```
[72]: dataf.isnull().sum()
```

```
[72]: 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 37 no missing is available

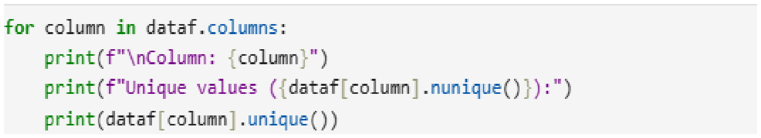
Explanation:

Dataf.dropna(inplace=True) removes any row from the DataFrame dataf that contains any missing (NaN) values. It is a dropna() function which does this by finding at least one column having missing value in a row and dropping this row. By setting inplace=True, the changes are made directly on the original DataFrame instead of creating or assigning a new variable. Thus this helps keep the clean data in dataf by leaving only those rows which are complete and do not have missing data fields.

2.6. Question no.6: displaying unique values from all the column

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

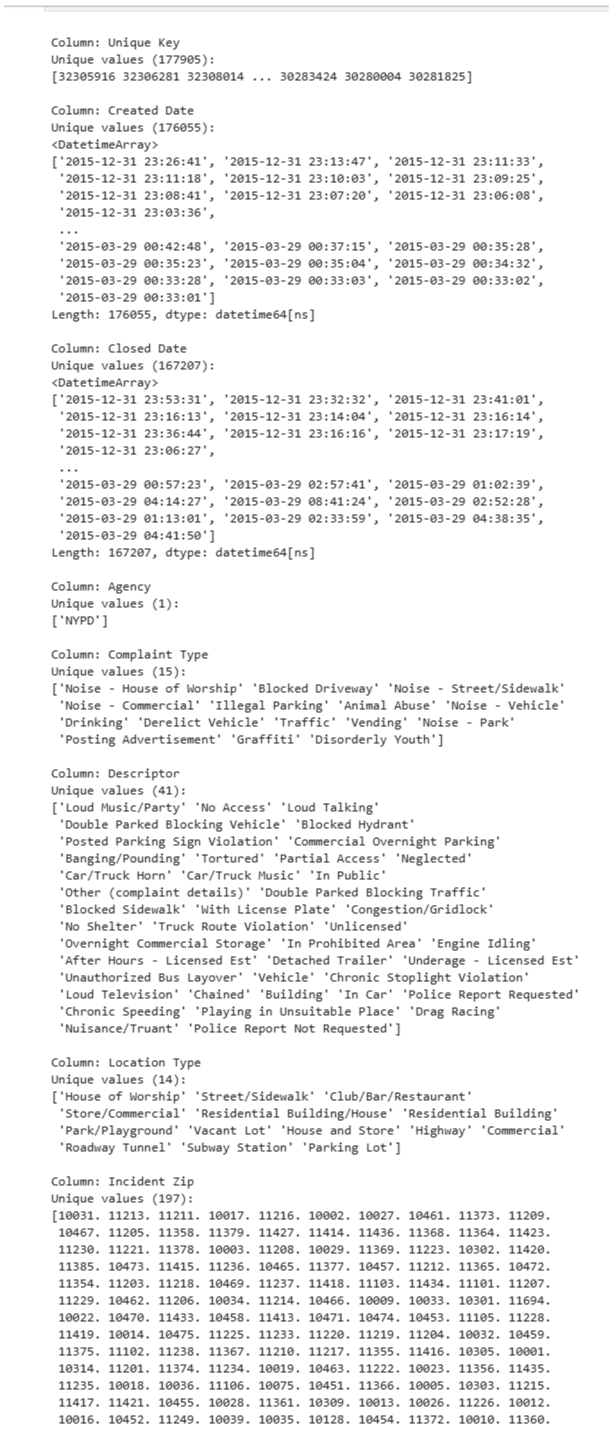
Code:



```
for column in dataf.columns:
    print(f"\nColumn: {column}")
    print(f"Unique values ({dataf[column].nunique()}):")
    print(dataf[column].unique())
```

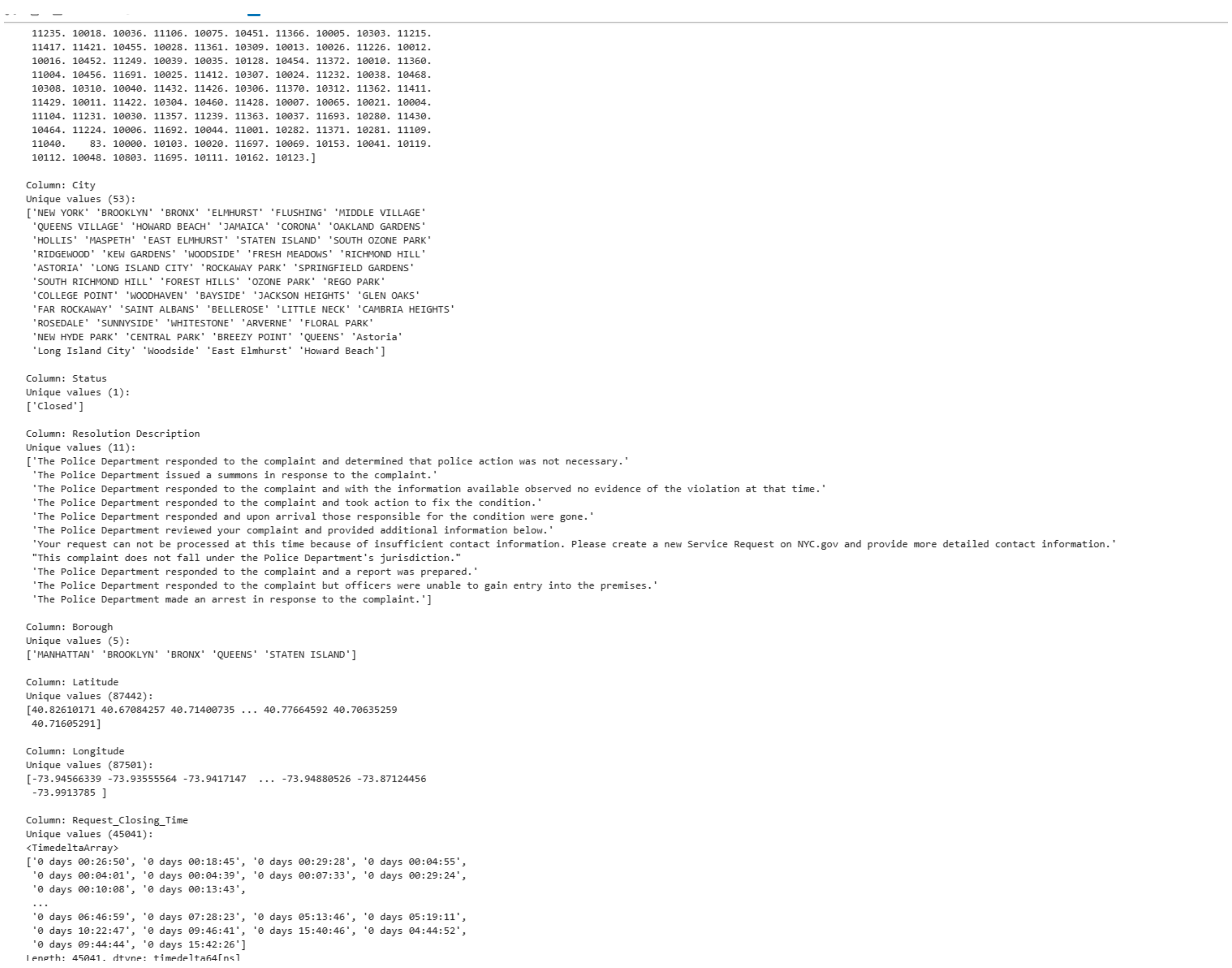
Figure 38 code to see the unique values

Output:



```
Column: Unique Key
Unique values (177905):
[32305916 32306281 32308014 ... 30283424 30280004 30281825]

Column: Created Date
Unique values (176055):
<DatetimeArray>
['2015-12-31 23:26:41', '2015-12-31 23:13:47', '2015-12-31 23:11:33',
 '2015-12-31 23:11:18', '2015-12-31 23:10:03', '2015-12-31 23:09:25',
 '2015-12-31 23:08:41', '2015-12-31 23:07:20', '2015-12-31 23:06:08',
 '2015-12-31 23:03:36',
 ...
 '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: 176055, dtype: datetime64[ns]

Column: Closed Date
Unique values (167207):
<DatetimeArray>
['2015-12-31 23:53:31', '2015-12-31 23:32:32', '2015-12-31 23:41:01',
 '2015-12-31 23:16:13', '2015-12-31 23:14:04', '2015-12-31 23:16:14',
 '2015-12-31 23:36:44', '2015-12-31 23:16:16', '2015-12-31 23:17:19',
 '2015-12-31 23:06:27',
 ...
 '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: 167207, dtype: datetime64[ns]

Column: Agency
Unique values (1):
['NYPD']

Column: Complaint Type
Unique values (15):
['Noise - House of Worship' 'Blocked Driveway' 'Noise - Street/Sidewalk'
 'Noise - Commercial' 'Illegal Parking' 'Animal Abuse' 'Noise - Vehicle'
 'Drinking' 'Derelict Vehicle' 'Traffic' 'Vending' 'Noise - Park'
 'Posting Advertisement' 'Graffiti' 'Disorderly Youth']

Column: Descriptor
Unique values (41):
['Loud Music/Party' 'No Access' 'Loud Talking'
 'Double Parked Blocking Vehicle' 'Blocked Hydrant'
 'Posted Parking Sign Violation' 'Commercial Overnight Parking'
 'Banging/Pounding' 'Tortured' 'Partial Access' 'Neglected'
 'Car/Truck Horn' 'Car/Truck Music' 'In Public'
 'Other (complaint details)' 'Double Parked Blocking Traffic'
 'Blocked Sidewalk' 'With License Plate' 'Congestion/Gridlock'
 'No Shelter' 'Truck Route Violation' 'Unlicensed'
 'Overnight Commercial Storage' 'In Prohibited Area' 'Engine Idling'
 'After Hours - Licensed Est' 'Detached Trailer' 'Underage - Licensed Est'
 'Unauthorized Bus Layover' 'Vehicle' 'Chronic Stoplight Violation'
 'Loud Television' 'Chained' 'Building' 'In Car' 'Police Report Requested'
 'Chronic Speeding' 'Playing in Unsuitable Place' 'Drag Racing'
 'Nuisance/Truant' 'Police Report Not Requested']

Column: Location Type
Unique values (14):
['House of Worship' 'Street/Sidewalk' 'Club/Bar/Restaurant'
 'Store/Commercial' 'Residential Building/House' 'Residential Building'
 'Park/Playground' 'Vacant Lot' 'House and Store' 'Highway' 'Commercial'
 'Roadway Tunnel' 'Subway Station' 'Parking Lot']

Column: Incident Zip
Unique values (197):
[10031. 11213. 11211. 10017. 11216. 10002. 10027. 10461. 11373. 11209.
 10467. 11205. 11358. 11379. 11427. 11414. 11436. 11368. 11364. 11423.
 11230. 11221. 11378. 10003. 11208. 10029. 11369. 11223. 10302. 11420.
 11385. 10473. 11415. 11236. 10465. 11377. 10457. 11212. 11365. 10472.
 11354. 11203. 11218. 10469. 11237. 11418. 11103. 11434. 11101. 11207.
 11229. 10462. 11206. 10034. 11214. 10466. 10009. 10033. 10301. 11694.
 10022. 10470. 11433. 10458. 11413. 10471. 10474. 10453. 11105. 11228.
 11419. 10014. 10475. 11225. 11233. 11220. 11219. 11204. 10032. 10459.
 11375. 11102. 11238. 11367. 11210. 11217. 11355. 11416. 10305. 10001.
 10314. 11201. 11374. 11234. 10019. 10463. 11222. 10023. 11356. 11435.
 11235. 10018. 10036. 11106. 10075. 10451. 11366. 10005. 10303. 11215.
 11417. 11421. 10455. 10028. 11361. 10309. 10013. 10026. 11226. 10012.
 10016. 10452. 11249. 10039. 10035. 10128. 10454. 11372. 10010. 11360.
```

Figure 39 displaying unique values from data frame



```
  11235. 10018. 10036. 11106. 10075. 10451. 11366. 10005. 10303. 11215.
  11417. 11421. 10455. 10028. 11361. 10309. 10013. 10026. 11226. 10012.
  10016. 10452. 11249. 10039. 10035. 10128. 10454. 11372. 10010. 11360.
  11004. 10456. 11691. 10025. 11412. 10307. 10024. 11232. 10038. 10468.
  10308. 10310. 10040. 11432. 11426. 10306. 11370. 10312. 11362. 11411.
  11429. 10011. 11422. 10304. 10460. 11428. 10007. 10065. 10021. 10004.
  11104. 11231. 10030. 11357. 11239. 11363. 10037. 11693. 10280. 11430.
  10464. 11224. 10006. 11692. 10044. 11001. 10282. 11371. 10281. 11109.
  11040.    83. 10000. 10103. 10020. 11697. 10069. 10153. 10041. 10119.
  10112. 10048. 10803. 11695. 10111. 10162. 10123.]

Column: City
Unique values (53):
['NEW YORK' 'BROOKLYN' 'BRONX' 'ELMHURST' 'FLUSHING' 'MIDDLE VILLAGE'
 'QUEENS VILLAGE' 'HOWARD BEACH' 'JAMAICA' 'CORONA' 'OAKLAND GARDENS'
 'HOLLIS' 'MASPETH' 'EAST ELMHURST' 'STATEN ISLAND' 'SOUTH OZONE PARK'
 'RIDGEWOOD' 'KEW GARDENS' 'WOODSIDE' 'FRESH MEADOWS' 'RICHMOND HILL'
 'ASTORIA' 'LONG ISLAND CITY' 'ROCKAWAY PARK' 'SPRINGFIELD GARDENS'
 'SOUTH RICHMOND HILL' 'FOREST HILLS' 'OZONE PARK' 'REGO PARK'
 'COLLEGE POINT' 'WOODHAVEN' 'BAYSIDE' 'JACKSON HEIGHTS' 'GLEN OAKS'
 'FAR ROCKAWAY' 'SAINT ALBANS' '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']

Column: Status
Unique values (1):
['Closed']

Column: Resolution Description
Unique values (11):
['The Police Department responded to the complaint and determined that police action was not necessary.'
 'The Police Department issued a summons in response to the complaint.'
 'The Police Department responded to the complaint and with the information available observed no evidence of the violation at that time.'
 'The Police Department responded to the complaint and took action to fix the condition.'
 'The Police Department responded and upon arrival those responsible for the condition were gone.'
 '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 NYC.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.']

Column: Borough
Unique values (5):
['MANHATTAN' 'BROOKLYN' 'BRONX' 'QUEENS' 'STATEN ISLAND']

Column: Latitude
Unique values (87442):
[40.82610171 40.67084257 40.71400735 ... 40.77664592 40.70635259
 40.71605291]

Column: Longitude
Unique values (87501):
[-73.94566339 -73.93555564 -73.9417147  ... -73.94880526 -73.87124456
 -73.9913785 ]

Column: Request_Closing_Time
Unique values (45041):
<TimedeltaArray>
['0 days 00:26:50', '0 days 00:18:45', '0 days 00:29:28', '0 days 00:04:55',
 '0 days 00:04:01', '0 days 00:04:39', '0 days 00:07:33', '0 days 00:29:24',
 '0 days 00:10:08', '0 days 00:13:43',
 ...
 '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: 45041, dtype: timedelta64[ns]
```

Figure 40 remaining unique values

Explanation:

The code loops through the columns of the DataFrame dataf. For every column, it first prints the column's name. Then the number of unique values in that column is printed using nunique(), then followed by a list of all the unique values themselves with unique(). This is a very fast and handy method to explore data in order to understand the nature of values present in each column.

3 Data Analysis

The process of analysing datasets to better understand the information that they contain is known as data analysis. Organizing, cleaning, and studying the data leads to understanding patterns or trends. It makes involving questions like what is happening or why is it happening.

Organizations then learn from data analysis to make more informed decisions, speed up cycles of operations, and even predict future outcomes. Enterprises extensively use it in their respective segments; it touches various categories like the business world, healthcare, marketing, finance, and even scientific research to find insights and solve. This article is about what data analysis is, types, and tools that can best facilitate effective analysis. Data analysis helps organizations make informed decisions by turning raw data into valuable insights (Anon., 2025)

3.1 Summary statistics of sum, mean, standard deviation, skewness, and kurtosis

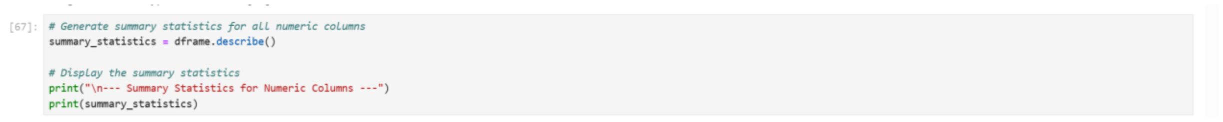
Mean: The mean is an average of all the values in a data set. The mean represents a central value that represents the dataset as a whole.

Standard Deviation: Standard deviation measures the amount of variation or dispersion of a set of values in question. A low standard deviation means that the values tend to be close to the mean or average, while a high standard deviation indicates that the values are spread out over a wider range of values.

Skewness: Skewness measures the asymmetry of the probability distribution of a real-valued random variable. If skewness is negative, you have longer or fatter tails on the left-hand side of the distribution. If skewness is positive, you have longer or fatter tails on the right-hand side.

Kurtosis: Kurtosis is a statistical measure that is used to describe the distribution of observed data around the mean. In other words, how flat or peaked is the data. Higher kurtosis means that there are more outliers and lower kurtosis means that there are fewer extreme values.

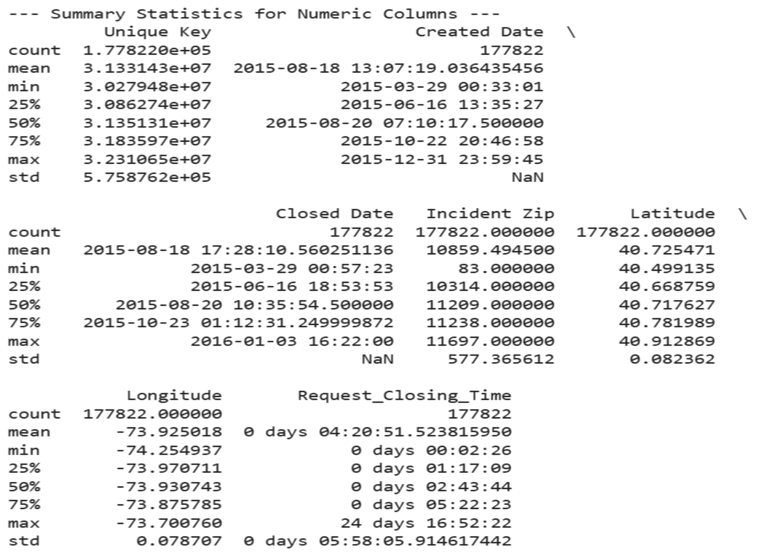
Code for summary statistics of numeric column



```
[67]: # Generate summary statistics for all numeric columns
summary_statistics = dframe.describe()

# Display the summary statistics
print("\n--- Summary Statistics for Numeric Columns ---")
print(summary_statistics)
```

Figure 41 code for summary statistics of numeric column

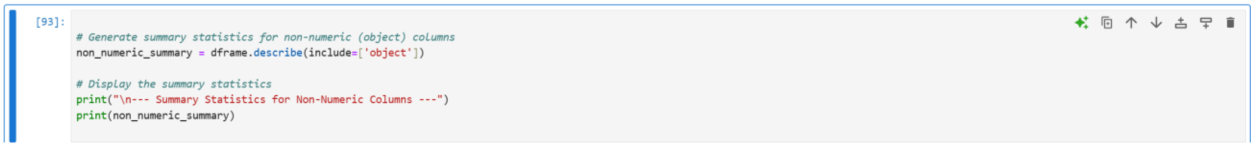
Output:



```
--- Summary Statistics for Numeric Columns ---
         Unique Key                   Created Date  \
count  1.778220e+05                         177822
mean   3.133143e+07  2015-08-18 13:07:19.036435456
min    3.027948e+07            2015-03-29 00:33:01
25%    3.086274e+07            2015-06-16 13:35:27
50%    3.135131e+07     2015-08-20 07:10:17.500000
75%    3.183597e+07            2015-10-22 20:46:58
max    3.231065e+07            2015-12-31 23:59:45
std    5.758762e+05                            NaN

                         Closed Date   Incident Zip       Latitude  \
count                         177822  177822.000000  177822.000000
mean   2015-08-18 17:28:10.560251136   10859.494500      40.725471
min              2015-03-29 00:57:23      83.000000      40.499135
25%              2015-06-16 18:53:53   10314.000000      40.668759
50%       2015-08-20 10:35:54.500000   11209.000000      40.717627
75%    2015-10-23 01:12:31.249999872   11238.000000      40.781989
max              2016-01-03 16:22:00   11697.000000      40.912869
std                              NaN     577.365612       0.082362

           Longitude       Request_Closing_Time
count  177822.000000                     177822
mean      -73.925018  0 days 04:20:51.523815950
min       -74.254937            0 days 00:02:26
25%       -73.970711            0 days 01:17:09
50%       -73.930743            0 days 02:43:44
75%       -73.875785            0 days 05:22:23
max       -73.700760           24 days 16:52:22
std         0.078707  0 days 05:58:05.914617442
```

Figure 42 output for summary statistics for numeric column

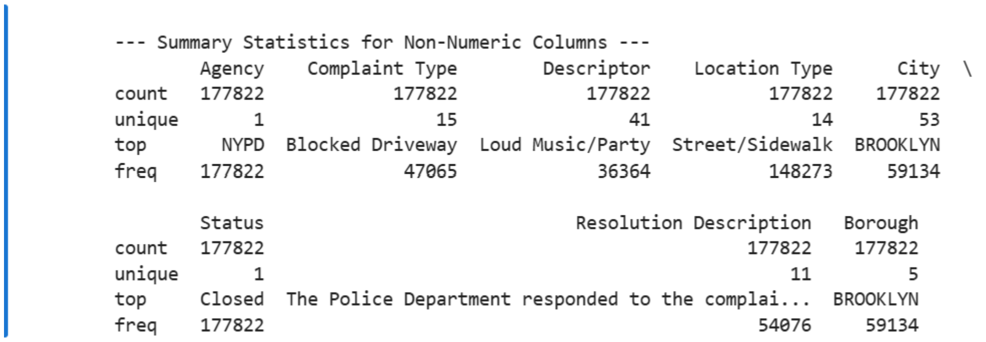
Code for summary statistics of non-numeric column



```
[93]:
# Generate summary statistics for non-numeric (object) columns
non_numeric_summary = dframe.describe(include=['object'])

# Display the summary statistics
print("\n--- Summary Statistics for Non-Numeric Columns ---")
print(non_numeric_summary)
```

Figure 43 code for summary statistics for non-numeric columns

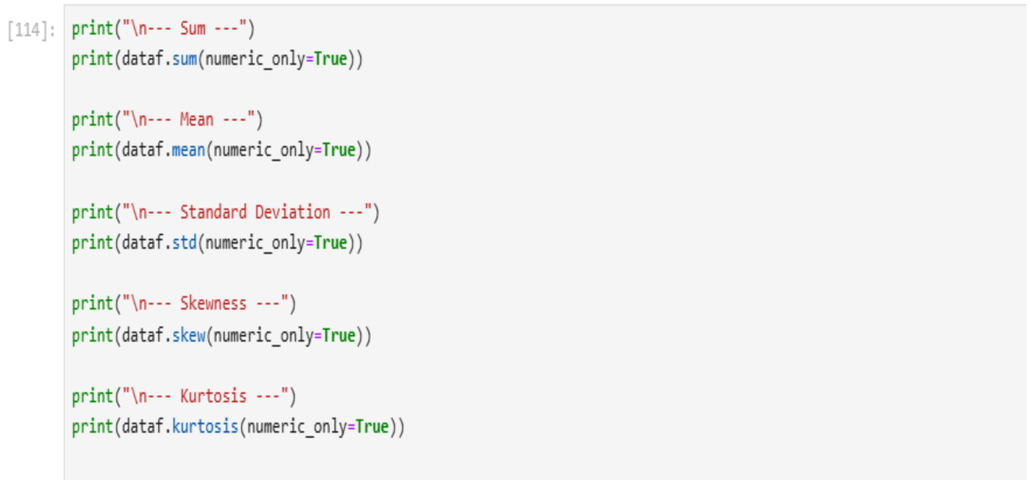
Output:



```
--- Summary Statistics for Non-Numeric Columns ---
        Agency    Complaint Type        Descriptor    Location Type     City  \
count   177822            177822            177822           177822   177822
unique       1                15                41               14       53
top       NYPD  Blocked Driveway  Loud Music/Party  Street/Sidewalk  BROOKLYN
freq    177822             47065             36364           148273    59134

        Status                             Resolution Description   Borough
count   177822                                             177822    177822
unique       1                                                 11         5
top     Closed  The Police Department responded to the complai...  BROOKLYN
freq    177822                                              54076     59134
```

Figure 44 output for summary statistics for non-numeric columns

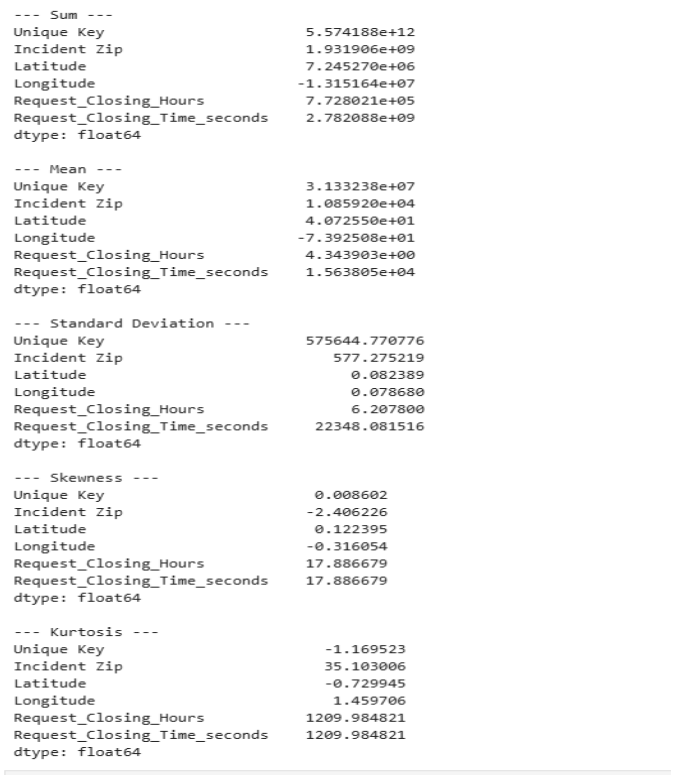
Code for sum, mean, standard deviation, skewness, and kurtosis of the data frame.



```
[114]: print("\n--- Sum ---")
print(dataf.sum(numeric_only=True))

print("\n--- Mean ---")
print(dataf.mean(numeric_only=True))

print("\n--- Standard Deviation ---")
print(dataf.std(numeric_only=True))

print("\n--- Skewness ---")
print(dataf.skew(numeric_only=True))

print("\n--- Kurtosis ---")
print(dataf.kurtosis(numeric_only=True))
```

Figure 45 code to show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame

Output:



```
--- Sum ---
Unique Key                        5.574188e+12
Incident Zip                      1.931906e+09
Latitude                          7.245270e+06
Longitude                        -1.315164e+07
Request_Closing_Hours             7.728021e+05
Request_Closing_Time_seconds      2.782088e+09
dtype: float64

--- Mean ---
Unique Key                        3.133238e+07
Incident Zip                      1.085920e+04
Latitude                          4.072550e+01
Longitude                        -7.392508e+01
Request_Closing_Hours             4.343903e+00
Request_Closing_Time_seconds      1.563805e+04
dtype: float64

--- Standard Deviation ---
Unique Key                        575644.770776
Incident Zip                         577.275219
Latitude                               0.082389
Longitude                              0.078680
Request_Closing_Hours                  6.207800
Request_Closing_Time_seconds       22348.081516
dtype: float64

--- Skewness ---
Unique Key                         0.008602
Incident Zip                      -2.406226
Latitude                           0.122395
Longitude                         -0.316054
Request_Closing_Hours             17.886679
Request_Closing_Time_seconds      17.886679
dtype: float64

--- Kurtosis ---
Unique Key                          -1.169523
Incident Zip                        35.103006
Latitude                            -0.729945
Longitude                            1.459706
Request_Closing_Hours             1209.984821
Request_Closing_Time_seconds      1209.984821
dtype: float64
```

Figure 46 displaying statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame

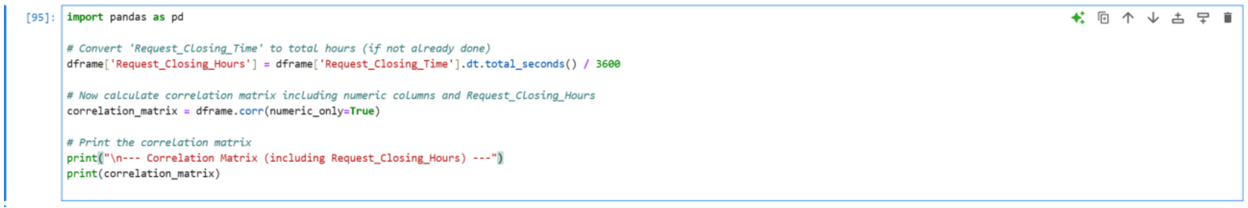
Explanation:

In this code, statistical summaries are calculated and printed for a number of variables with numeric types in DataFrame dataf. It first prints the sum of each numeric column using the sum() function; next, it uses the mean() function, which shows the average value, then on to the standard deviation, which measures how spread out data are, using std(). Next, it calculates the skewness of each column using the skew() function that tells how asymmetric the data distribution is. Finally, it computes and displays the kurtosis from the kurtosis() function and is a measure of "tailedness" or extremeness of the data distribution. The numeric_only=True argument ensures that the operations are performed on columns populated with specific data types.

3.2 calculating the correlation

Correlation is a statistical measure that indicates the strength and direction of a relationship between two variables. A positive correlation indicates both variables increase together (the first variable rises as the second variable rises), a negative correlation indicates one increases while the other decreases, and a value close to zero indicates a small.

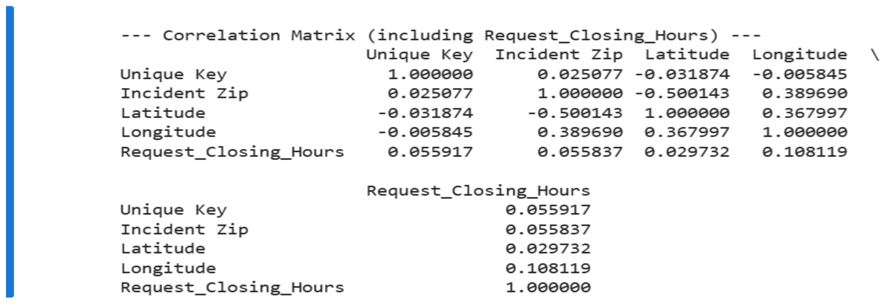
Code:



```
[95]: import pandas as pd

      # Convert 'Request_Closing_Time' to total hours (if not already done)
      dframe['Request_Closing_Hours'] = dframe['Request_Closing_Time'].dt.total_seconds() / 3600

      # Now calculate correlation matrix including numeric columns and Request_Closing_Hours
      correlation_matrix = dframe.corr(numeric_only=True)

      # Print the correlation matrix
      print("\n--- Correlation Matrix (including Request_Closing_Hours) ---")
      print(correlation_matrix)
```

Figure 47 code to calculate and show correlation of all variables

Output:



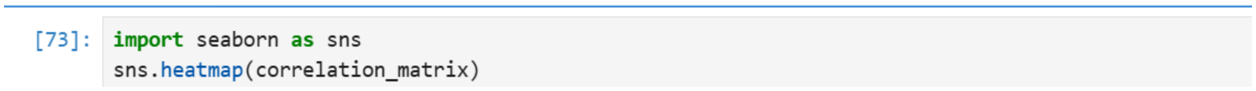
```
--- Correlation Matrix (including Request_Closing_Hours) ---
                       Unique Key  Incident Zip  Latitude  Longitude  \
Unique Key               1.000000      0.025077 -0.031874  -0.005845
Incident Zip             0.025077      1.000000 -0.500143   0.389690
Latitude                -0.031874     -0.500143  1.000000   0.367997
Longitude               -0.005845      0.389690  0.367997   1.000000
Request_Closing_Hours    0.055917      0.055837  0.029732   0.108119

                       Request_Closing_Hours
Unique Key                          0.055917
Incident Zip                        0.055837
Latitude                            0.029732
Longitude                           0.108119
Request_Closing_Hours               1.000000
```

Figure 48 displaying calculate and show correlation of all variables

Explanation:

The function calculates the correlation matrix for the Data Frame dataf using the corr() method with numeric_only=True, applying the calculations only to columns that are numeric in type. The correlation matrix indicates the strength of a pair of columns being correlated with one another, that is, with values ranging from -1 to 1. A value close to 1 indicates a strong positive correlation, a value close to -1 indicates a strong negative correlation, while a value around 0 indicates that there is little to no correlation. The correlation matrix is printed out after being computed in the code, giving a reader a clearer insight into the relationships between multiple numeric features in the dataset.

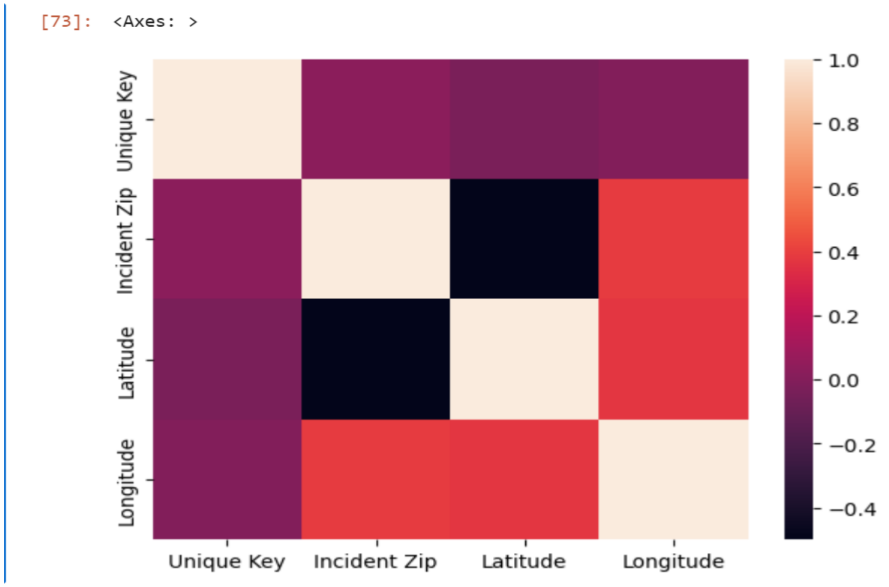
Code for heatmap of correlation



```
[73]: import seaborn as sns
      sns.heatmap(correlation_matrix)
```

Figure 49 code for heatmap of correlation

Output:



```
[73]: <Axes: >
```

Figure 50 displaying a proper heatmap of correlation

Explanation:

The sns.heatmap(correlation_matrix) function is utilized to display a visual of the correlation matrix of numerical variables withinthe dataset. The heatmap shows the strength and direction of the linear relationships between each pair of numerical columns. The colors represent either positive relationship (usually warmer colors) to negative relationship (usually cooler colors), and makes it apparent quickly which are related (positively or negatively). Thus, it highlights relationships, patterns, redundancies, or susceptibility to multicollinearity in the data. In exploratory data analysis this would be considered a low-cost access to additional information for decision-making purposes before higher-cost analyses or predictive modeling occur.

4 Data Exploration:

Data Exploration which is also known as Exploratory Data Analysis (EDA), is a critical component of the data science workflow; it involves the detailed examination of datasets for pattern discovery, anomaly, assumptions, and data quality before any major modelling techniques are concerned.

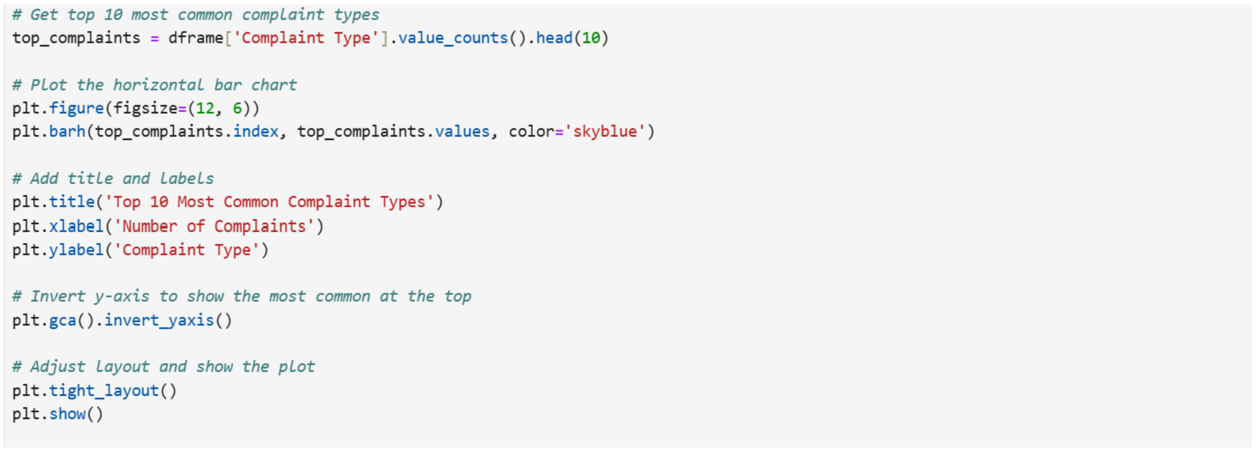
EDA combines descriptive statistics with data-visualization techniques, instrumental in giving the analyst a feel for the nature and structure of their data. The latter involves computing the summary statistics - mean, median, and maximum; minimum, standard deviation, and interquartile range - that pertain to the concepts of distribution, central tendency, and variability in the context of numerical variables. Data visualization techniques such as histograms, box plots, scatter plots, and density maps provide an alternate way to retrace unexplained data trends, clusters, outliers, and irregularities.

Furthermore, bivariate and multivariate analyses serve to reveal existing relationships among the variables; these could be cross tabbing for categorical data or correlation matrices for continuous variables. The other key element that tends to characterize this stage is a thoroughgoing examination of anomalies in the dataset: that is, looking for missing values, duplicate records, and inconsistent data types, etc.; anything that will hinder the accuracy and reliability of the subsequent analysis. An analyst, then, begins with a full-fledged understanding of the data, which helps him choose the statistical tests appropriate for his analysis, validate the assumptions concerning his modelling, and mitigate the confusion arising from it. (Staff, 2024)

4.2 Question no 1: four major insights through visualization

4.2.4 1st insight: Top 10 most common complaint types

Code:



```
# Get top 10 most common complaint types
top_complaints = dframe['Complaint Type'].value_counts().head(10)

# Plot the horizontal bar chart
plt.figure(figsize=(12, 6))
plt.barh(top_complaints.index, top_complaints.values, color='skyblue')

# Add title and labels
plt.title('Top 10 Most Common Complaint Types')
plt.xlabel('Number of Complaints')
plt.ylabel('Complaint Type')

# Invert y-axis to show the most common at the top
plt.gca().invert_yaxis()

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```

Figure 51 code for top 10 most common complaints types

Output:

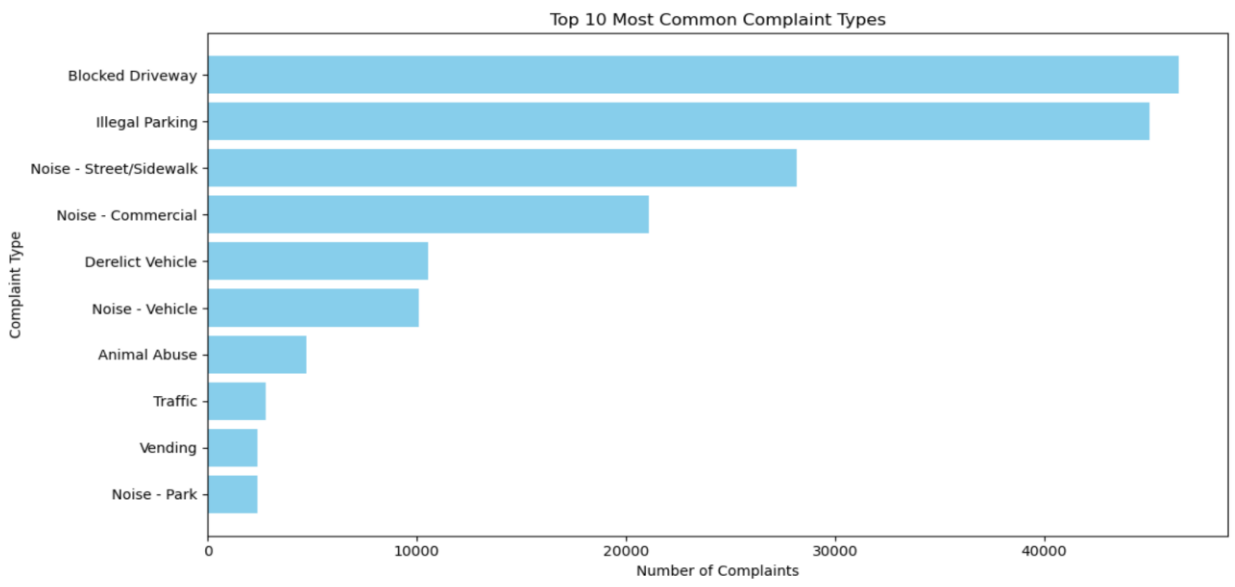


Figure 52 horizontal bar graph for top 10 most common complaints types

How the insight can help us:

The presented insight will help us by allowing us to identify the most frequently reported complaints in New York City through the 311 service request system, enabling the respective city service departments to assign and prioritize their assets effectively and enlist staff to provide policies or interventions for the most concerning issues for the community. Understanding the most widely reported complaints will allow the city to streamline public service, improve resident satisfaction, and allow the city to respond to troubling issues that reinstate complaints and cause difficulty in everyday living.

What we have learned from this:

From these analyses, we gained a clear ranking of the most reported complaint types in the data set. It gives a simple but handy glance at public discontent, and urban service issues in New York City as they proceed over time. This frequency distribution provides analysts with an easy way to see which issues matter most to the community often, and what might need situating for analysis purposes as to why an issue is more common. It also provides a great start for a solid analysis framework, such as whether complaint types vary by borough or over time.

Explanation of the code:

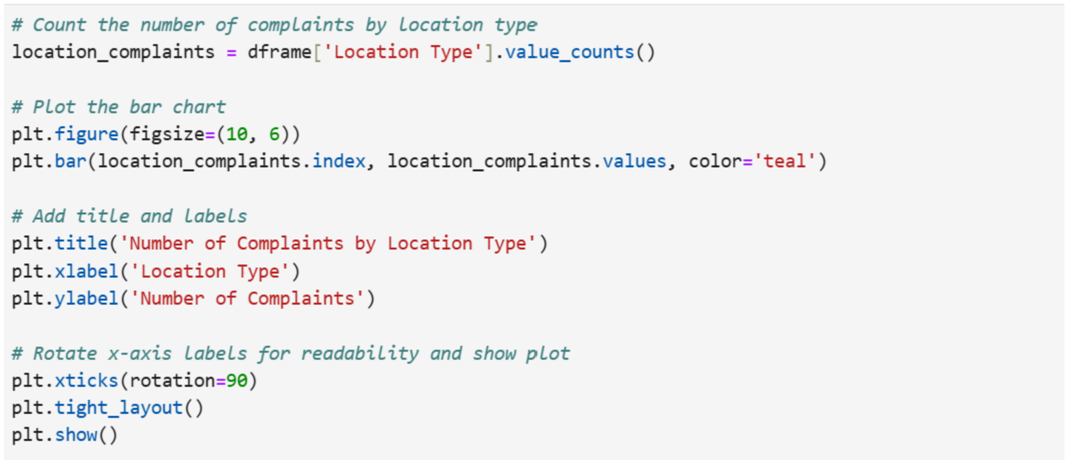
This code begins with counting each unique complaint type in the data using the value_counts() function on the Complaint Type column, then it extracts the top 10 most frequently reported complaint types with the head(10) function before saving that subset into a variable called top_complaints. Then, using the matplotlib.pyplot library, it creates a horizontal bar plot with complaint type on the y-axis and frequencies of complaints on the x-axis. The bar plot is created with a figure size of (12,6) because it seemed too easy to read and interpret - the cleaner the plot the better use of the cleaned dataset. The default color for bar plots showing complaint frequencies, is coloring in 'skyblue' so the result has a clean and easy to read diagram. To make reading the plot easier for the viewer, the y-axis was inverted so highest complaint frequency would appear at the top. Narrative titles for the graph and axis labels were added to better plane the dataset for the visualization and to assist with how interpret-able the bar graph was. Finally, the code has a tight_layout() method to better try and fit the plot in the figure.

Explanation of the output:

The output of this code is the horizontal bar plot of the top 10 complaint types associated with service requests. The horizontal bar compound denotes that the type of complaint represented on the bar graph will be represented as bars where the length of the bars is reflective of the number of times this type of complaint is present in the data. Further, since the most frequent type of complaint is represented first on the top of the bar graph due to the inversion of the y-axis denoting the direction of the plot, the visual comparison provides a quick and easy means of comparing the number of complaints present. Therefore, the resultant chart makes interpreting which areas of service likely require the attention of city officials much easier, which will ultimately lead to a better-informed decision-making process regarding the operation and improvement of local public service delivery.

4.2.5 2nd insight: Number of complaints type by location

Code:



```
# Count the number of complaints by location type
location_complaints = dframe['Location Type'].value_counts()

# Plot the bar chart
plt.figure(figsize=(10, 6))
plt.bar(location_complaints.index, location_complaints.values, color='teal')

# Add title and labels
plt.title('Number of Complaints by Location Type')
plt.xlabel('Location Type')
plt.ylabel('Number of Complaints')

# Rotate x-axis labels for readability and show plot
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```

Figure 53 code for number of complaints by location

Output:

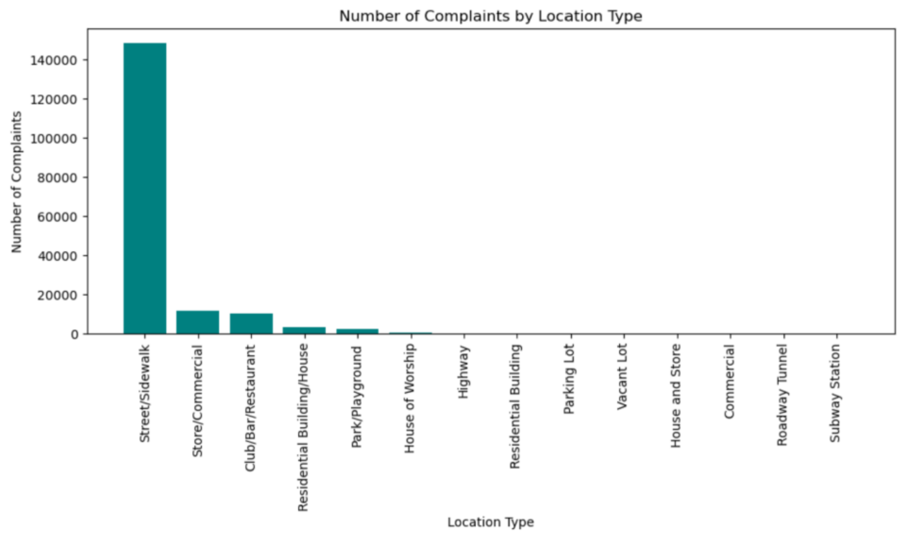


Figure 54 bar graph for number of complaints by location

How the insight can help us:

This insight will help us understand the types of locations in New York City that have the largest amount of 311 service complaints. By knowing where the complaints are reported most frequently whether in residential buildings, on sidewalks or commercial properties, city agencies and service managers will better manage where resources are allocated and how they formulate response strategies on trending location types. It allows the city to identify issues that arise in specific areas so that they can take preventative action or mitigate issues in relation to service levels. This process will improve the overall quality of public serviced and improve the day-to-day experience of residents.

What we have learned from this:

From the analysis we were able to determine precisely where complaints are located over time, which geographic areas are least and most likely to create convenience problems for the public (keeping agencies aware of where the public has experience service issues), and which geospatial areas require real-time or near real-time municipal action. This information is useful for operational plans, asset maintenance plans, and safety concerns; it gives agencies locations for distributing patrols, performing inspections, and executing preventive maintenance programs; it also gives planners and decision-makers from all areas of the city a means of assessing whether certain locations types, such as residential lands or public streets, are consistently presenting service problems for the city and due to urban management issues.

Explanation of the code:

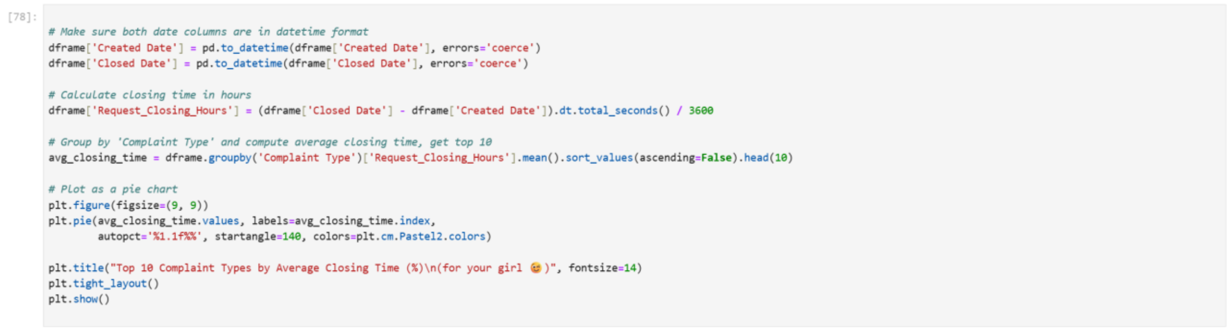
The code first utilizes the value_counts() function on the Location Type column of the Dataset to find the total amount of complaints that fall under each unique location type in the dataset. These results are stored in a variable entitled location_complaints. A vertical bar chart is created with matplotlib.pyplot, and as the intent is to improve the visibility of the chart for the reader, the figure is created with a size of (10, 6). The x-axis represents the various location types, and the y-axis counts the amount of complaints of each respective location type. The bars are colored 'teal' for comparison with future charts as well as for distinct coloration. The x-axis labels were too long and longstanding so they were rotated 90 degrees using xticks(rotation=90) to ensure they did not overlap and provided visibility of the labeling. Titles and axis tick labels are added for clarity and tight_layout() is used to ensure that everything fits nicely in the figure space.

Explanation of the output:

When the above code block is executed, the output will be a vertical bar chart displaying the count of service complaints by location type. Each bar represents a particular setting in which complaints have been reported, and the height of each bar reflects the total number of complaints by this category. The bar chart visually ranks the location types by the number of complaints, so users can instantly see whether residential buildings, streets, or commercial buildings had the highest number of complaints. This also provide a visual summary of the locations that may need some additional intervention or improvement by a public service provider, which can be helpful in regard to better service planning and coordination throughout the city.

4.2.6 3rd insight: top 10 complaints with longest average closing time

Code:



```
[78]:
# Make sure both date columns are in datetime format
dframe['Created Date'] = pd.to_datetime(dframe['Created Date'], errors='coerce')
dframe['Closed Date'] = pd.to_datetime(dframe['Closed Date'], errors='coerce')

# Calculate closing time in hours
dframe['Request_Closing_Hours'] = (dframe['Closed Date'] - dframe['Created Date']).dt.total_seconds() / 3600

# Group by 'Complaint Type' and compute average closing time, get top 10
avg_closing_time = dframe.groupby('Complaint Type')['Request_Closing_Hours'].mean().sort_values(ascending=False).head(10)

# Plot as a pie chart
plt.figure(figsize=(9, 9))
plt.pie(avg_closing_time.values, labels=avg_closing_time.index,
        autopct='%1.1f%%', startangle=140, colors=plt.cm.Pastel2.colors)

plt.title("Top 10 Complaint Types by Average Closing Time (%)\n(for your girl 😉)", fontsize=14)
plt.tight_layout()
plt.show()
```

Figure 55 code for complaints with longest average closing time

Output:

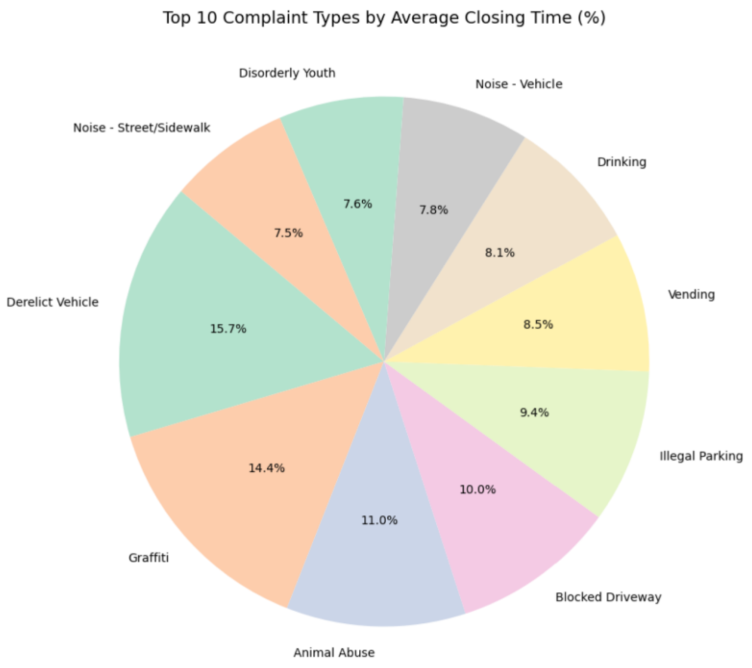


Figure 56 pie chart for longest average closing time

How the insight can help us:

This information helps, in several ways, to highlight which of the complaint types takes the average longest to resolve and give an immediate understanding of service efficiencies and operational blockages happening in the cities operational 311 Service Request System. By having an understanding of the complaint types that consistently are taking more time to close, the city agencies can examine why, and whether there are procedural reviews, or whether it may require walkthroughs or specialized teams to close where certain complaint type delays are occurring. As decision makers are able to focus on problem resolution to address the issues causing extending response times of 12, 18 and greater periods of time, they are directly impacting public performance service to the citizen by providing better service, and which in turn increases resident satisfaction with the services provided directly.

What we have learned from this:

From the analysis, we obtained some useful understanding about differences in response times by complaint type. The observation identifies the average times to closure as a percentage, and the ten complaints with the longest average closure times, in pie chart form. In this instance, it allows us to make a quick visual comparison of time expenditure across complaint types. Detecting these categories is useful for operational purposes in the City, as they provide some awareness to which categories may be in need of additional resources, changes to processes, or changes in policy, where there are excess delays in progression and service delivery.

Explanation of code:

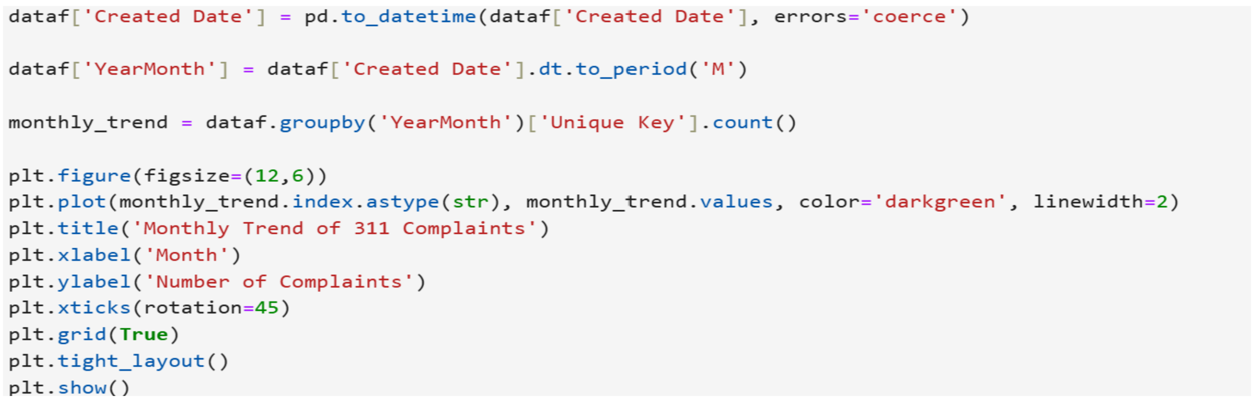
The code starts there converting the Created Date and Closed Date into datetime objects using the pd.to_datetime() method, also coercively dealing with invalid parsing with errors='coerce'. Then, the code calculates Each service request's total time to close in Hours through Created Date – Closed Date, while converting them into seconds into hours using the .dt.total_seconds() / 3600 method. This was accomplished after which the dataset was grouped by Complaint Type while calculating the mean hours to close for each Complaint Type using groupby() and mean () methods. The averages were then sorted in descending order using sort_values(ascending=False), and with head (10), the top 10 highest average closing Complaint Types were returned. The code then creates a pie chart using matplotlib.pyplot to visualize the percentage of total average closing time being taken up by the top 10 Complaint Types. The average closing times are the values while each slice is confirmed as that Complaint Type.

Explanation of the output:

The result of this code is a pie chart showing the percentage contributions of the top 10 complaint types with the longest average times to close. Each slice represents a complaint type, and the size of the slice conveys the amount of total average closing time that the complaint type represents. This visualization is valuable for understanding which complaint types are exceptionally time-consuming in relation to other complaint types. Additionally, it offers operational managers some tangible data to consider where staff may want to investigate, add resources to, or shorten procedures for resolving public service requests more efficiently.

4.2.7 4th insight: monthly trend of 311 complaints

Code:



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

dataf['YearMonth'] = dataf['Created Date'].dt.to_period('M')

monthly_trend = dataf.groupby('YearMonth')['Unique Key'].count()

plt.figure(figsize=(12,6))
plt.plot(monthly_trend.index.astype(str), monthly_trend.values, color='darkgreen', linewidth=2)
plt.title('Monthly Trend of 311 Complaints')
plt.xlabel('Month')
plt.ylabel('Number of Complaints')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

Figure 57 code for monthly trend of 311 Complaints

Output:

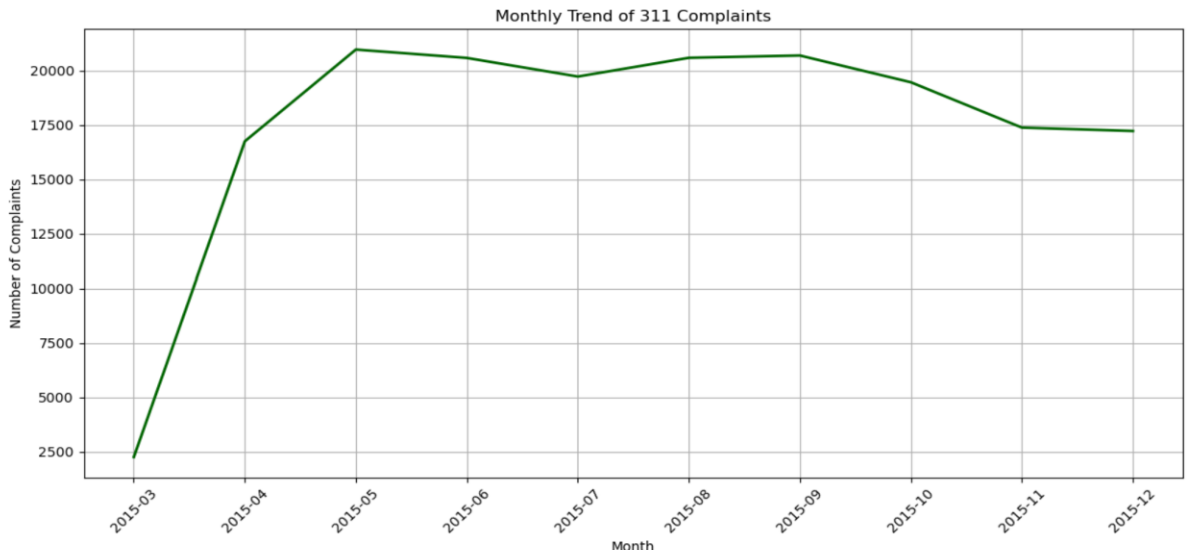


Figure 58 line graph for monthly trend of 311 complaints

How the insight can help us:

This knowledge is beneficial in that it illustrates how 311 service complaints may evolve temporally in New York City. By measuring time series levels of each type of complaint monthly, even distinguishing between seasonal variations, new issues, and irregular spikes in demand for services, public service departments may better manage their resources, anticipate complaint activity, and prepare for re-emerging service barriers while demand is increasing. By examining requests for municipal services temporally, the city agencies can evaluate the influence of the municipal government’s policy or public actions from time to time based on the number of complaint changes.

What we have learned from this:

This analysis provided a illustration of how the volume of complaints fluctuates over the months represented in the dataset. The line chart clearly identifies the months with high or low complaint activity, which contributes to understanding community service usage patterns, as well as public engagement behaviour. Trends can be identified, like spikes in complaints in certain months or flat lined months, and leaders will be able to understand how the community behaved around their needs for public service that led them to submit complaints, and use that to develop better operational planning, workforce staffing, and preventative maintenance scheduling.

Explanation of code:

The code begins by converting the Created Date column to a proper datetime using pd.to_datetime(), so any bad/odddates can be handled with errors='coerce' (we can convert invalid or inconsistent date values to NaT). A new column YearMonth will be created that extracts the year and month portion of each created date using .dt.to_period('M') to reduce it to a monthly level. Once a new YearMonth column has been created, we can group the dataframe using the YearMonth column and then count the number of complaints that were made each month by counting, the Unique Key values with .count().

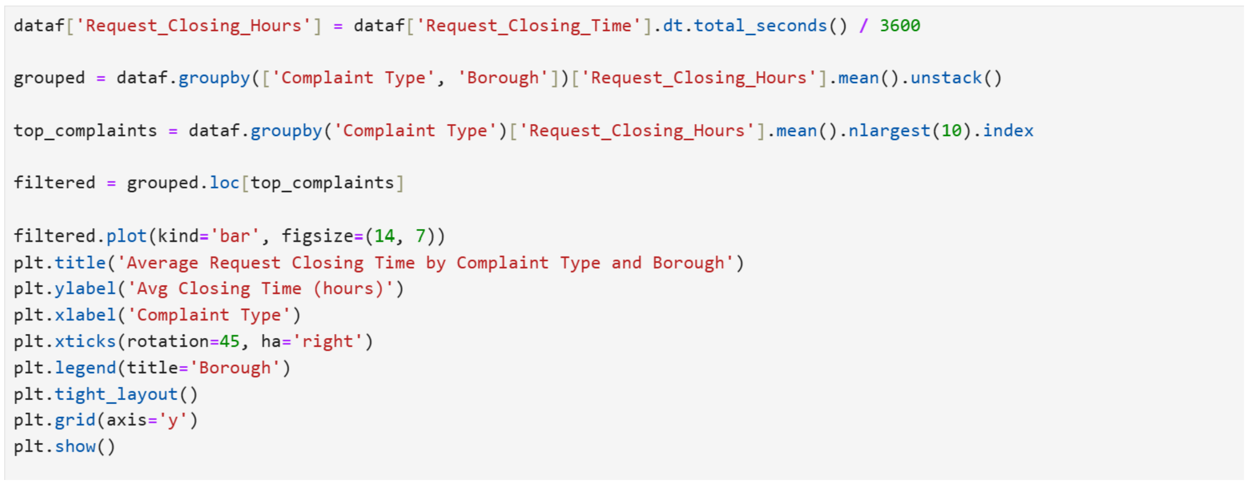
Next, we will create a line plot using matplotlib.pyplot, using str() to display the months and count of complaints on the y-axis. The line will have a 'darkgreen' style and a linewidth of 2 which should help show the trend. The plot will include a title, axis texts and will rotate the x-axis labels using xticks(rotation=45) to avoid text overlapping. A grid will be implemented to help visualize the trends and. tight_layout() will arrange the plot so nothing overlaps.

Explanation of the output:

The output of this code is a line chart that depicts the monthly trend of 311 service complaints over the time range of the dataset. The total number of complaints submitted in one month was graphed for each month, with a continuous line connecting the points to show trends and fluctuations over time. The chart provides a quick assessment of the months that had extreme values up and down, while also being able to reflect upon long-term trends or seasonal demands on 311 service. This provides the municipal staff with visual clarity of the data, which is valuable for operational planning in anticipation of a spike in volume (e.g. winter is here, or front lawn summons to appear in court) or when evaluating the effectiveness of new municipal programs over time on reducing the number of complaints.

4.3 Question no 2: avg request closing by complaint type and borough

Code:



```
dataf['Request_Closing_Hours'] = dataf['Request_Closing_Time'].dt.total_seconds() / 3600

grouped = dataf.groupby(['Complaint Type', 'Borough'])['Request_Closing_Hours'].mean().unstack()

top_complaints = dataf.groupby('Complaint Type')['Request_Closing_Hours'].mean().nlargest(10).index

filtered = grouped.loc[top_complaints]

filtered.plot(kind='bar', figsize=(14, 7))
plt.title('Average Request Closing Time by Complaint Type and Borough')
plt.ylabel('Avg Closing Time (hours)')
plt.xlabel('Complaint Type')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Borough')
plt.tight_layout()
plt.grid(axis='y')
plt.show()
```

Figure 59 code for average 'Request_Closing_Time' by complaint type and borough

Output:

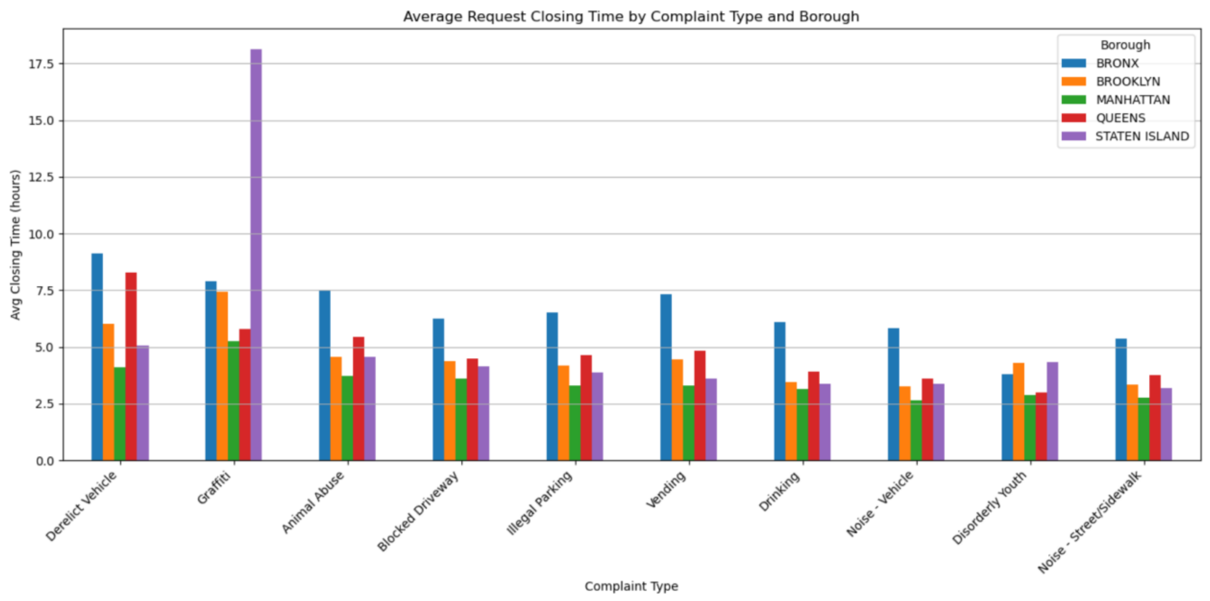


Figure 60 multiple bar graph for average 'Request_Closing_Time' by complaint type and borough

How the insight can help us:

This information is useful in that it highlights the differences in average time taken to resolve complaints by borough and by complaint type. It allows authorities and administrators to see which boroughs are faster or slower in resolving certain complaints. Perhaps authorities can optimize resource allocation, work on inefficiencies, and produce more reasonable above-target expectations by comparing boroughs with longer times and those with shorter ones. This allows the latter boroughs to consider reviewing and refining the internal processes that caused delays in resolution times. In the end, the boroughs can provide better service delivery to their citizens and provide citizens with a better complaint residential record by enabling a more efficient 311 service system.

What we gained from this:

The analysis will show which type of complaints have the longest resolution times and how they vary spatially. It not only allows for the comparison of differences between boroughs regarding which complaint types may be causing greater delays; it also provides documentation that may ultimately permit the prioritization of those complaint types that warrant immediate administrative action. This produces clear actionable data that provide data-based insights as to the next steps of improving operational efficiencies on complaint systems that minimize delays, and guarantee that the most serious public complaints from citizens are addressed in a reasonable time frame regardless of which area the complaint was submitted.

Explanation of code:

The code starts by checking that 'Created Date' and 'Closed Date' are both in datetime format, so that we can conduct calculations based on time. The code then calculates the complaint resolution time in hours and instantiates a new column named 'Request_Closing_Hours' to keep track of this duration. The data then get grouped by 'Complaint Type' and 'Borough' and averages for resolution time were calculated based on these two groupings. With this, the 10 complaint types with the highest overall average resolution time were pulled. The groupings were then filtered to only these complaint types. Finally, a grouped bar chart plot was ran, to easily compare how well each borough resolving the complaint type from the top 10.

Output of code:

The takeaway from this chart is that is clearly shows how average resolution times relate to each borough when comparing the most time-consuming complaint types. For each complaint type seen on the x-axis, the plotted bars for each borough allows a straightforward and comprehensive assessment of efficiency in providing service. Any inconsistencies can be easily identified (for example, one borough having consistently higher resolution times might be flagged for a breakdown in the process or lack of resources), and with the comparison to all borough will help qualified individuals in decisions to balance workloads, and reduce complaint resolution times. Overall, this analysis assists in visualizing differences in time resolution while also uncovering trends in service across boroughs.

5 Statistical Testing

A statistical testing method is a hypothesis testing method that aims at finding strong evidence against the null hypothesis concerning the observation data compared to the predictions made by the assumption. Thus, the method is crucially important in the field of data science, as it allows researchers to extrapolate population information from sample sets and to formulate probabilities surrounding such extrapolations. Statistical tests aim at comparing two contradicting hypotheses: the null hypothesis (H_0) and the alternative hypothesis (H_1). The statistical test provides the laboratory with a p-value to calculate the probability of observing results as extreme or more extreme than those obtained, given that the null hypothesis is true. H_0 is rejected when a predetermined significance level has been set at, and the computed p-value is below that level. Statistical tests can be classified into one of two types:

- Parametric tests, which include t-tests and ANOVA. They operate under the assumption that a certain statistical distribution is applicable. Normality is one such example of a distribution.

- Nonparametric tests, such as chi-square tests and the Mann-Whitney U tests, are examples of statistical procedures that make the least assumptions regarding the distribution of data.

Statistical testing is applied in data analysis and coding to preserve the patterns found and prove that they originate in a natural way from the data set. (Bevans, 2023)

T-test:

A t-test is a parametric inferential test that is widely renowned for its usefulness in comparing the means of two distinct groups or populations. This technique has been in existence since the early twentieth century, having been researched and published by William Sealy Gosset, who wrote under the pseudonym of "Student" as part of the statistical methodological development (Student, 1908). The test works by testing whether there is a statistically significant difference between the populations' means when it is expected that they are sampled from normal populations with equal variances. There are different types of t-tests for different scenarios. Their differences, according to their categories, are: one-sample t-tests; independent t-test for two sample tests; and paired t-test. To have a valid t-test, it should be noncontinuous data which are normally distributed and independent observations with equal variance between groups otherwise the results could be misleading. Therefore, it is very important to assess and ensure these conditions are met in the exploratory phase of data before performing the t-test. (Chugh, 2021)

Chi-Square Test

Chi-square tests are sort of a non-parametric statistical tool, which means one is used to determine whether there is a significant relationship between categorical variables. It is applied especially to frequency distributions in contingency tables, where it subjects the observed distribution of counts to what would have been expected under the assumption of independence (McHugh, 2013). The Chi-square test has primarily two types: Chi-square independence test is used to determine the relationship between the two categorical variables (such as gender vs. selection of a product) while the Chi-square goodness of fit test is used to assess whether the distribution of one categorical variable concurs with a predicted distribution. The test is often used in survey research, marketing, and the social sciences, where its simplicity and versatility are well-appreciated. It has a caveat, though: a minimum of five expected frequencies from either or each experience is typically expected for the test results to be considered reliable. (CodeRivers, 2021)

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

- State the Null Hypothesis (H0) and Alternate Hypothesis (H1).

Null hypothesis (H_0)

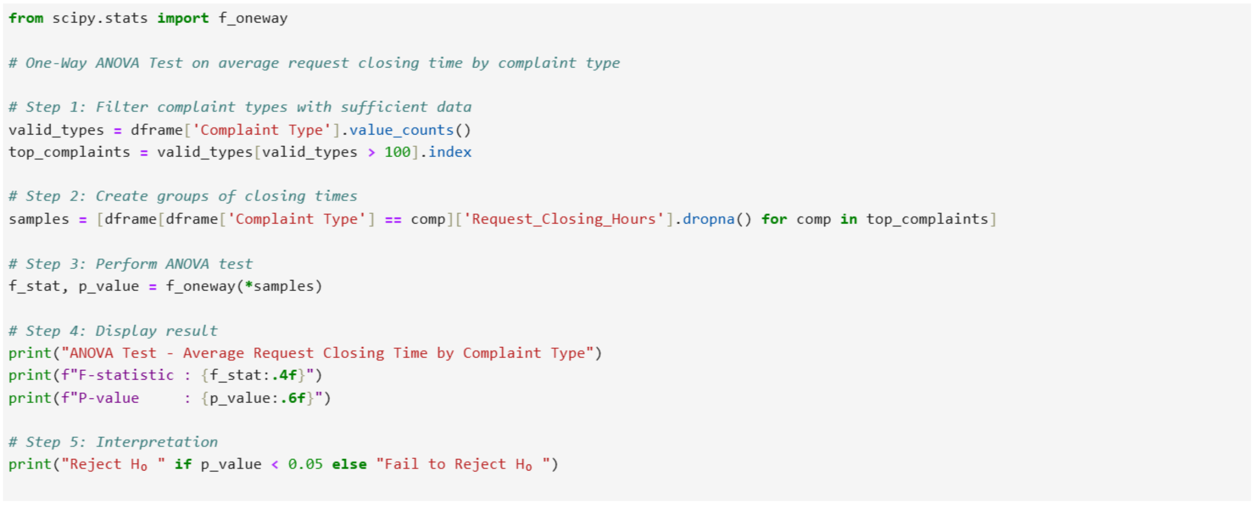
The null hypothesis (H_0), which in a sense represents the antithesis of all that has preceded in inferential statistics, is the assertion that no effect, no difference, and no relation between variables exist. It serves as an initial baseline, putting forth the idea that any observed outcome results from random chance or sampling error. (Frost, 2022)

Alternative hypothesis(H_1)

H_1 calls for an effect, a difference, or an association among variables. It specifies a statement that the researcher tries to support with evidence. The alternative hypothesis can be either one-tailed, predicting the direction of an effect (e.g., one group will do better than another), or two-tailed, predicting a difference without indicating direction. The decision of whether to consider a particular hypothesis as one-tailed or two-tailed will give sensitivity and will impact the interpretation of the test. One-tailed tests are more powerful in the one direction but cannot detect effects in the opposite direction. (Turney, 2025)

- Perform the statistical test and provide the p-value.
- Interpret the results to accept or reject the Null Hypothesis.

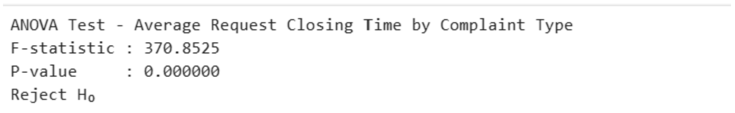
Code:



```
from scipy.stats import f_oneway

# One-Way ANOVA Test on average request closing time by complaint type

# Step 1: Filter complaint types with sufficient data
valid_types = dframe['Complaint Type'].value_counts()
top_complaints = valid_types[valid_types > 100].index

# Step 2: Create groups of closing times
samples = [dframe[dframe['Complaint Type'] == comp]['Request_Closing_Hours'].dropna() for comp in top_complaints]

# Step 3: Perform ANOVA test
f_stat, p_value = f_oneway(*samples)

# Step 4: Display result
print("ANOVA Test - Average Request Closing Time by Complaint Type")
print(f"F-statistic : {f_stat:.4f}")
print(f"P-value     : {p_value:.6f}")

# Step 5: Interpretation
print("Reject H₀ " if p_value < 0.05 else "Fail to Reject H₀ ")
```

Figure 61 code for Null Hypothesis (H0) and Alternate Hypothesis (H1).

Output:



```
ANOVA Test - Average Request Closing Time by Complaint Type
F-statistic : 370.8525
P-value     : 0.000000
Reject H₀
```

Figure 62 output for Null Hypothesis (H0) and Alternate Hypothesis (H1).

Explanation of code:

The code executes a one-way ANOVA test, which tests whether there is a statistically significant difference in average request closing times across different complaint types. One-way ANOVA is ideal because it compares the means of a continuous dependent variable, Request_Closing_Hours, across independent groups; this is distinctly different from the t-test (only two groups) or Chi-Square test (categorical variables). The code filters for complaint types with over 100 records to allow for reliable results as well as to filter for the closing times within those selected complaint types into groups. The code then implements SciPy's f_oneway() function to compute both the F-statistic and associated p-value to assess the relationship. The F-statistic compares the variance both between and within groups, while the p-value indicates the relative probability for obtaining a result like the one we obtained assuming that the null hypothesis is true. We will use a significance level of 0.05 for our initial loss of the null hypothesis. Therefore, if the p-value is below 0.05, this would indicate that average closing times are significantly different between different complaint types, and is a useful operational insight when thinking about the efficiency of service response time improvements, etc.

Explanation of output:

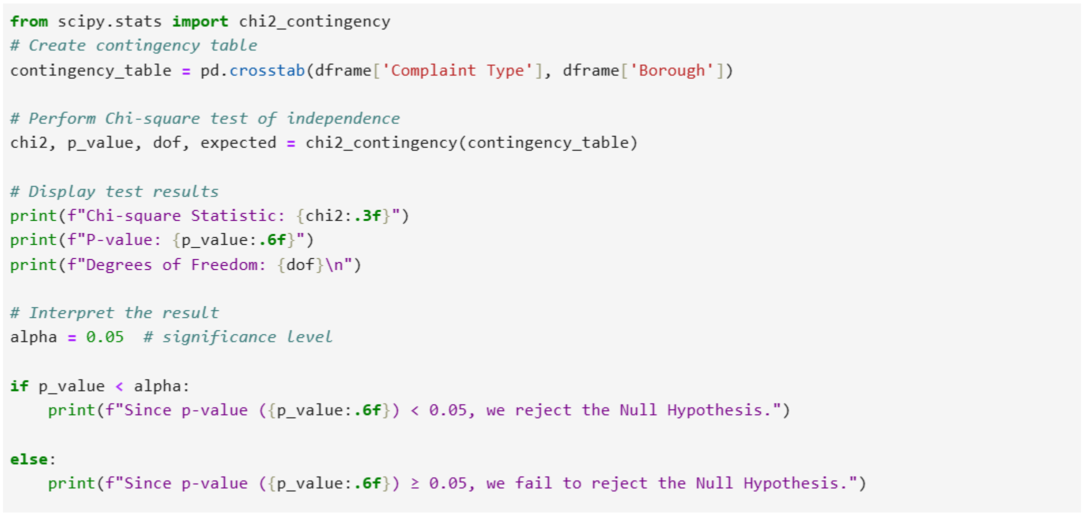
The F-statistic of 370.853 measures the variability of mean closing times between complaint types relative to variability within complaint types. A difference of this magnitude would indicate that group averages differ greatly. The p-value (0.000000) tells us the difference is statistically significant as it is well below the acceptable level of 0.05. We will therefore reject the null hypothesis and conclude that mean closing times differ significantly between complaint types.

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

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.

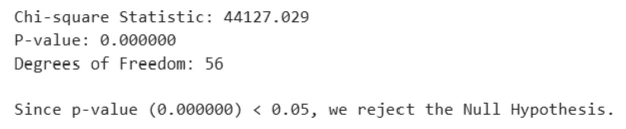
Code:



```
from scipy.stats import chi2_contingency
# Create contingency table
contingency_table = pd.crosstab(dframe['Complaint Type'], dframe['Borough'])

# Perform Chi-square test of independence
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

# Display test results
print(f"Chi-square Statistic: {chi2:.3f}")
print(f"P-value: {p_value:.6f}")
print(f"Degrees of Freedom: {dof}\n")

# Interpret the result
alpha = 0.05  # significance level

if p_value < alpha:
    print(f"Since p-value ({p_value:.6f}) < 0.05, we reject the Null Hypothesis.")

else:
    print(f"Since p-value ({p_value:.6f}) ≥ 0.05, we fail to reject the Null Hypothesis.")
```

Figure 63 code for test 2 of statistical testing

Output:



```
Chi-square Statistic: 44127.029
P-value: 0.000000
Degrees of Freedom: 56

Since p-value (0.000000) < 0.05, we reject the Null Hypothesis.
```

Figure 64 output for test 2 of statistical testing

Explanation of the code:

For this part of the analysis, we want to see if the type of complaint a person makes is associated with the location (borough) the complaint was made. To do this we use the Chi-Square Test of Independence. The Chi-square test is useful when you want to analyse whether two categorical variables (e.g. "Complaint Type" & "borough") are related. We will create a contingency table using the pd.crosstab() function which counts how many times each complaint type occurs for each borough. Then we will use the chi2_contingency() function from the scipy.stats library to perform the chi-square test. This function returns the Chi-square statistic, the p-value, the degrees of freedom, and the expected values if the two variables are independent.

Explanation of the output:

Based on the test results, the Chi-square value was 44127.029 and the p-value was 0.000000, with degrees of freedom equal to 56. The null hypothesis (H_0) states that complaint type and borough are independent (i.e., the type of complaint does not depend on where it was reported). Since the p-value is less than .05, we reject the null hypothesis. In other words, selective patterns exist between complaint type and borough. In other words, certain complaint types are selective to certain boroughs where others are not, which can possibly help the city with resource allocation to the specific issues at location.

6 Conclusion

This coursework involved the exploratory analysis of the New York City 311 Customer Service Requests dataset using Python and associated data analysis packages (Pandas, Matplotlib, Seaborn and Scipy). With respect to Python, an initial phase was an understanding of the data where we explored its structure, column descriptions and key attributes to understand the type and purpose of the data. The data preparation phase involved the importing of the dataset, deal with the missing values, convert the date columns into a datetime format, calculate request close times, and drop irrelevant, and redundant columns from the dataset to create a clean dataset suitable for analysis. We also counted unique values and checked the data types in the dataset to ensure that the data was still appropriate.

In the data analysis section, we provided a statistical summary of the numerical and categorical columns while identifying central tendencies, distributions, and completeness. Finally, we used correlation matrices and heat maps to investigate relationships amongst the numerical features. The initially considered analysis included producing visualizations utilizing bar charts, pie charts, line plots, and grouped bar charts, to identify patterns in complaint frequencies, complaint types by borough, monthly trends, and average closing times. The use of visualizations will provide insight into patterns based on complaint type, complaint frequencies, and response times for specific complaint types across geographic distance, in addition to any relevant year to year shifts in response times pertaining to all service requests. Statistical testing was completed to test two hypotheses; one to examine if, on average, the response time variance from complaint type was statistically significant (one-way ANOVA test) and to examine if complaint type and location were correlated (chi-square test of independence). Each of these hypothesis tests provided worthwhile evidence about operational differences, and geographic relations in the service request process.

In conclusion, this data analytics coursework demonstrated practical application of data wrangling, exploration, analysis, and statistical testing techniques providing a practical purpose by offering actionable insights to make improvements operationally and at a basic planning resource level for public services process.

7. References

Anon., 2025. *What is Data Analysis?.* [Online]
Available at: https://www.geeksforgeeks.org/what-is-data-analysis/
[Accessed 11 4 2025].

Bevans, R., 2023. *Test statistics | Definition, Interpretation, and Examples.* [Online]
Available at: https://www.scribbr.com/statistics/statistical-tests/
[Accessed 4 5 2025].

Chugh, V., 2021. *An Introduction to Python T-Tests.* [Online]
Available at: https://www.datacamp.com/tutorial/an-introduction-to-python-t-tests
[Accessed 3 5 2025].

CodeRivers, 2021. *Python Chi-Square Test: A Comprehensive Guide.* [Online]
Available at: Chi-square tests are sort of a non-parametric statistical tool, which means one is used to determine whether there is a significant relationship between categorical variables. It is applied especially to frequency distributions in contingency tables, wher
[Accessed 4 5 2025].

Frost, J., 2022. *Null Hypothesis: Definition, Rejecting & Examples.* [Online]
Available at: https://statisticsbyjim.com/hypothesis-testing/null-hypothesis/
[Accessed 4 5 2025].

Staff, C., 2024. *What Is Data Exploration? Definition, Types, Uses, and More.* [Online]
Available at: https://www.coursera.org/articles/data-exploration?msockid=1e60427ec5c56c7824e15673c4776d44
[Accessed 4 5 2025].

stedmen, C., 2021. *What is data preparation? An in-depth guide.* [Online]
Available at: https://www.techtarget.com/searchbusinessanalytics/definition/data-preparation
[Accessed 13 4 2025].

Turney, S., 2025. *Null & Alternative Hypotheses | Definitions, Templates & Examples.* [Online]
Available at: https://www.scribbr.com/statistics/null-and-alternative-hypotheses/
[Accessed 4 5 2025].