



slington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CC5067NI Smart Data Discovery

60% Individual Coursework

Submission: Final Submission

Academic Semester: Spring Semester 2025

Credit: 15 credit semester long module

Student Name: Aabriti Pradhan

London Met ID: 23047543

College ID: NP01CP4A230019

Assignment Due Date: Thursday, May 15, 2025

Assignment Submission Date: Thursday, May 15, 2025

Submitted To: Dipeshor Silwal

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

23047543 Aabriti Pradhan8.docx

Islington College,Nepal

Document Details

Submission ID

trn:oid::3618:95951726

Submission Date

May 15, 2025, 7:41 AM GMT+5:45

Download Date

May 15, 2025, 7:43 AM GMT+5:45

File Name

23047543 Aabriti Pradhan8.docx

File Size

30.2 KB

32 Pages

4,765 Words

25,015 Characters







Page 2 of 36 - Integrity Overview

Submission ID trn:oid::3618:95951726




17% Overall Similarity

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

Match Groups

-  **49 Not Cited or Quoted 17%**
Matches with neither in-text citation nor quotation marks
-  **1 Missing Quotations 0%**
Matches that are still very similar to source material
-  **1 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

- 8%  Internet sources
- 2%  Publications
- 15%  Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

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

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

Table Of Contents

1. Data Understanding	1
2. Data Preparation	6
3. Data Analysis	21
4. Data Exploration.....	26
5. Statistical Testing	36
a) Test 1	37
b) Test 2	39
6. Bibliography	41

Table Of Figures

Figure 1: Imported libraries	6
Figure 2: Importing the dataset	7
Figure 3: Information about the columns of provided data set.....	9
Figure 4: Converting created date to date time data type.....	10
Figure 5: Converting closed date to date time data type	11
Figure 6: Creating request_closing_time.....	12
Figure 7: Dropping irrelevant columns.....	13
Figure 8: Saving the updated data frame to the actual csv file	14
Figure 9: Showing NaN values	15
Figure 10: Removing rows that contains NaN values.....	16
Figure 11: Unique values of “unique key”, “created date” and “closed date”	17
Figure 12: Unique values of “agency”, “complaint type” and “descriptor”	18
Figure 13: Unique values of “location type” and “incident zip”	19
Figure 14: Unique values of “city”, “status” and “resolution descriptor”	20
Figure 15: Unique values of “borough”, “latitude”, “longitude” and “request_closing_time”	20
Figure 16: Creating new column with float datatype.....	21
Figure 17: Summary of statistics	24
Figure 18: Correlation of all variables.....	25
Figure 19: Data exploration - maximum number of complaints reported (code).....	26
Figure 20: Data exploration - maximum number of complaints reported (visualization)	27
Figure 21: Data Exploration - borough that receives maximum complaints.....	28
Figure 22: Data Exploration - hours taken for complaints to close down (code).....	29
Figure 23: Data Exploration - hours taken for complaints to close down (visualization)	29
Figure 24: Data Exploration - Maximum reoccurrence of complaint type in months (code)	31
Figure 25: Data Exploration - Maximum reoccurrence of complaint type in months (visualization)	31
Figure 26: Data Exploration - second part (code).....	32

Figure 27: Data exploration – second part (average time taken to solve complaint in New York)	33
Figure 28: Data exploration – second part (average time taken to solve complaint in Astoria).....	33
Figure 29: Data exploration – second part (average time taken to solve complaint in Brooklyn).....	34
Figure 30: Data exploration – second part (average time taken to solve complaint in Bronx).....	34
Figure 31: Data exploration – second part (average time taken to solve complaint in Forest Hills)	35
Figure 32: Test 1 - average response time across complaint types	37
Figure 33: Test 2 - relation of complaint type and location	39

Table Of Tables

Table 1: Data description	5
---------------------------------	---

1. Data Understanding

The data provided to us for our coursework is a data of 311 calls received by New York City Police Department from 2010 to present date. 311 is a non-emergency phone number used by the US government to help their citizens with numerous non-emergency problems like pothole repair, homeless person assistance and others (NYC, 2025).

The data set provided to us contains rows of information about the complaints or requests put in by the civilians of New York City. The data contains request or complaint types, agency to respond to the complaint, geographical location of incident (Mother Duck, 2025), complaint type and complaint description, the city of incident, status of the case, various dates like created dates, due date and closed dates, and many other data collected by New York City Police Department from 2010 to present date.

The detailed information about columns of data in the given data set is listed below.

<u>S.no</u>	<u>Column Name</u>	<u>Description</u>	<u>Data Type</u>
1	Unique Key	Unique Identifier for each 311 calls.	Integer
2	Created Date	Date the call was made.	String
3	Closed Date	Date the case was closed.	String
4	Agency	Code of agency that responded to the complaint or request.	String

5	Agency Name	Name of agency that responded to the complaint or request.	String
6	Complaint Type	Type of complaint or request.	String
7	Descriptor	Details of complaint or request.	String
8	Location Type	Place where the incident occurred.	String
9	Incident Zip	Zip code of where the incident occurred.	Float
10	Incident Address	Location of the incident.	String
11	Street Name	Name of the street where incident occurred.	String
12	Cross Street 1	Closest cross street to the incident.	String
13	Cross Street 2	Second closest cross street to the incident.	String
14	Intersection Street 1	One of the intersections near where the incident occurred.	String
15	Intersection Street 2	Another intersection near where the incident occurred.	String
16	Address Type	Type of address	String
17	City	Name of city where the incident occurred.	String
18	Landmark	Closest landmark to where the incident occurred.	String

19	Facility Type	A type of facility where the incident occurred.	String
20	Status	Current status of the incident.	String
21	Due Date	Expected date of when the issue is to be solved.	String
22	Resolution Description	Description of how the issue was solved by the responder.	String
23	Resolution Action Update Date	Last updated date of when the issue was resolved.	String
24	Community Board	Name of community board where the incident occurred.	String
25	Borough	NYC borough name of where the incident occurred.	String
26	X Coordinate (State Plane)	X-coordinate of the place where incident occurred, according to the state plane.	Float
27	Y Coordinate (State Plane)	Y-coordinate of the place where incident occurred, according to the state plane.	Float
28	Park Facility Name	Name of the park facility where the incident occurred.	String

29	Park Borough	Borough name of the park where the incident occurred.	String
30	School Name	Name of school where the incident occurred.	String
31	School Number	A unique identifier of the school where the incident occurred.	String
32	School Region	Region code of the school where the incident occurred.	String
33	School Code	Code of school where the incident occurred.	String
34	School Phone Number	Phone number of the school.	String
35	School Address	Address of the school.	String
36	School City	City where the school is located.	String
37	School State	State of where the school is.	String
38	School Zip	Zip code of the school.	String
39	School Not Found	States that school could not be found near the incident.	String
40	School or Citywide Complaint	Indicates whether the incident applies to school or city wide.	Float
41	Vehicle Type	The type of vehicle if the incident involves a vehicle.	Float

42	Taxi Company Borough	Borough of where the involved taxi company is located.	Float
43	Taxi Pick Up Location	Location of where the taxi picked up.	Float
44	Bridge Highway Name	Name of bridge or highway if the incident occurred in or near bridge or highway.	String
45	Bridge Highway Direction	Direction of where the bridge or highway is.	String
46	Road Ramp	Name of the road ramp.	String
47	Bridge Highway Segment	Segment of a bridge or a highway.	String
48	Garage Lot Name	Name of a garage or lot.	Float
49	Ferry Direction	Direction of ferry.	String
50	Ferry Terminal Name	Terminal name of the ferry.	String
51	Latitude	Latitude of where the incident occurred.	Float
52	Longitude	Longitude of where the incident occurred.	Float
53	Location	Combination of longitude and latitude of where the incident occurred.	String

Table 1: Data description

2. Data Preparation

For data preparation different libraries are used like numpy and pandas. According to pydata.org the mission of pandas is to be a “building block” for real work data analytics (pydata.org, 2025), and numpy is a numerical computing library that enables us to work with wide range of numerical evaluations (numpy.org, 2025).

In this coursework different methods like `.unique()`, `.read_csv()`, `.info()`, `.to_datetime()`, `.isna()` and `.dropna()` are used. The `.unique()` method is a method of pandas which returns the unique values of the given dataset in numpy array, whereas `.read_csv()` is a method used to read a csv file of the name passed as parameter inside the parenthesis. The `.info()` returns information about the given data frame, and `.to_datetime()` converts the datatype to datetime datatype. Lastly the `isna()` method is used to show the null values in a pandas series or in a data frame, whereas `dropna()` is used to delete those very null values rows.

```
[184]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
import scipy.stats as stats
```

Figure 1: Imported libraries

a) Import the dataset.

Here the data set provided to us is imported using the “pd.read_csv(“Customer Service_Requests_from_2010_to_Present.csv”)”. Now, the read variable contains a data frame of that csv file with column name same as the column name given to data in the dataset.

The only function used in here is “read.csv ()”. The read.csv() function is used to read csv files, and is a function of pandas library.

```
[2]: read = pd.read_csv("Customer Service_Requests_from_2010_to_Present.csv")
```

C:\Users\dell\AppData\Local\Temp\ipykernel_3340\2858045245.py:1: DtypeWarning: Columns (48,49) have mixed types. Specify dtype option on import or set low_memory=False.

```
read = pd.read_csv("Customer Service_Requests_from_2010_to_Present.csv")
```

```
[6]: read
```

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

Figure 2: Importing the dataset

- b) Provide your insight on the information and details that the provided dataset carries.

Here with “read.info ()” information about all the columns is listed. Most of the columns are of datatype object or string, while ten of the columns are float and one of the columns is integer. There are about 53 columns out of which some of the columns contain null values. Those values are not needed or should not be kept as while analysing the data it can cause error by misinterpreting the analysis.

By looking at the columns the most important one seems to be dates like created date and closed date, the location like the city or borough. With the help of those columns, we can analyse and find many insights on the data like response time which will give insights as to which complaint type needs more attention, overall or even if some specific city or borough.

```
[10]: read.info()
```

<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

Figure 3: Information about the columns of provided data set

- c) Convert the columns "Created Date" and "Closed Date" to datetime datatype and create a new column "Request_Closing_Time" as the time elapsed between request creation and request closing.

The "Created Date" and "Closed Date" columns were changed to datatype datetime from object datatype using "pd.to_datetime()". The datetime datatype is basically a long number that contains date and time with nanoseconds, which can be really useful for operations to fetch information out of a raw data.

The .to_datetime() method as mentioned above simply converts any datatype to datetime datatype, and this is important because we cannot get much insight into the data with string. When changed to a number, we can perform various calculations, like finding out specific months or years or days, or even time to get into detail about any complaint.

```
[34]: read["Created Date"] = pd.to_datetime(read["Created Date"])

[36]: read["Created Date"]

[36]: 0      2015-12-31 23:59:45
      1      2015-12-31 23:59:44
      2      2015-12-31 23:59:29
      3      2015-12-31 23:57:46
      4      2015-12-31 23:56:58
      ...
      300693  2015-03-29 00:33:41
      300694  2015-03-29 00:33:28
      300695  2015-03-29 00:33:03
      300696  2015-03-29 00:33:02
      300697  2015-03-29 00:33:01
      Name: Created Date, Length: 300698, dtype: datetime64[ns]
```

Figure 4: Converting created date to date time data type


```

[42]: read["Closed Date"] = pd.to_datetime(read["Closed Date"])

[44]: read["Closed Date"]

[44]: 0      2016-01-01 00:55:00
      1      2016-01-01 01:26:00
      2      2016-01-01 04:51:00
      3      2016-01-01 07:43:00
      4      2016-01-01 03:24:00
      ...
      300693      NaT
      300694      2015-03-29 02:33:59
      300695      2015-03-29 03:40:20
      300696      2015-03-29 04:38:35
      300697      2015-03-29 04:41:50
      Name: Closed Date, Length: 300698, dtype: datetime64[ns]

```

Figure 5: Converting closed date to date time data type

A new column “Request_Closing_Time” was made with the values which is the difference between “Closed Date” and “Created Date”. This column is also important for data mining or fetching information out of the data as it gives the time taken to solve any request that is received by the 311 calls.

How it can really be useful is that it can give us an insight into which problem needs to be dealt with faster, or which complaint takes more time to resolve, or other meaningful insight that can improve the overall performance of 311 call handling with logical suggestions.

```
[47]: read["Request_Closing_Time"] = read["Closed Date"] - read["Created Date"]
```

```
[51]: read[["Created Date", "Closed Date", "Request_Closing_Time"]]
```

```
[51]:
```

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

300698 rows × 3 columns

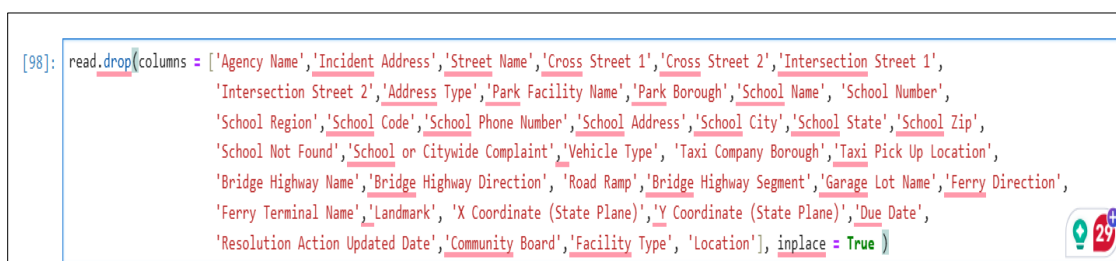
Figure 6: Creating request_closing_time

d) Write a python program to drop irrelevant Columns which are listed below.

['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']

All irrelevant columns were dropped using “.drop()” function and was saved to the actual csv file for further data analysis. The “inplace = True” enables us to make changes to the actual csv file, so that later when we need that change, we don’t have to do it time and again.

The columns that were removed did not contain any valuable information to change the overall performance of the call or even get a meaningful insight into the data.



```
[98]: read_csv(columns = ['Agency Name','Incident Address','Street Name','Cross Street 1','Cross Street 2','Intersection Street 1',
    'Intersection Street 2','Address Type','Park Facility Name','Park Borough','School Name', 'School Number',
    'School Region','School Code','School Phone Number','School Address','School City','School State','School Zip',
    'School Not Found','School or Citywide Complaint','Vehicle Type', 'Taxi Company Borough','Taxi Pick Up Location',
    'Bridge Highway Name','Bridge Highway Direction', 'Road Ramp','Bridge Highway Segment','Garage Lot Name','Ferry Direction',
    'Ferry Terminal Name','Landmark', 'X Coordinate (State Plane)','Y Coordinate (State Plane)','Due Date',
    'Resolution Action Updated Date','Community Board','Facility Type', 'Location'], inplace = True )
```

Figure 7: Dropping irrelevant columns

After dropping irrelevant columns only 15 columns remain. Those 15 columns can actually provide a valuable insight. For example, the agency column can give us an insight into the agency that is responsible for response and using other columns like created date or closing date or time column we can identify whether that agency is performing better or not. Similar thing with borough, we can identify whether the borough is performing better in solving the citizens problems with speed or not.

```
[92]: read.to_csv("Customer Service_Requests_from_2010_to_Present.csv", index = False)
```

```
[94]: read.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300698 entries, 0 to 300697
Data columns (total 15 columns):
#   Column                                Non-Null Count  Dtype
---  ---                                -
0   Unique Key                           300698 non-null int64
1   Created Date                         300698 non-null datetime64[ns]
2   Closed Date                         298534 non-null datetime64[ns]
3   Agency                             300698 non-null object
4   Complaint Type                      300698 non-null object
5   Descriptor                         294784 non-null object
6   Location Type                      300567 non-null object
7   Incident Zip                       298083 non-null float64
8   City                               298084 non-null object
9   Status                             300698 non-null object
10  Resolution Description              300698 non-null object
11  Borough                           300698 non-null object
12  Latitude                          297158 non-null float64
13  Longitude                         297158 non-null float64
14  Request_Closing_Time               298534 non-null timedelta64[ns]
dtypes: datetime64[ns](2), float64(3), int64(1), object(8), timedelta64[ns](1)
memory usage: 34.4+ MB
```

Figure 8: Saving the updated data frame to the actual csv file

- e) Write a python program to remove the NaN missing values from updated dataframe.

The `.isna()` method is used to show the null values in a data frame. For data mining or fetching important information from data the null values plays no role as they cannot give any insight into what is required to make the insight fruitful for future complaints registered.

```
read.isna().sum()

[43]:
Unique Key                0
Created Date              0
Closed Date              2164
Agency                  0
Complaint Type            0
Descriptor              5914
Location Type            131
Incident Zip             2615
City                    2614
Status                  0
Resolution Description    0
Borough                 0
Latitude                3540
Longitude               3540
Request_Closing_Time     2164
dtype: int64
```

Figure 9: Showing NaN values

Rows containing NaN values were removed using “`.dropna()`” function. After removing the NaN values only 291107 rows remain. The `.dropna()` function removes the column that contains NaN values, as stated above it does not serve any insight into the data that can be useful to calculate and evaluate many things.

```
[45]:  
  
read.dropna(inplace = True)  
  
[47]:  
  
read.isna().sum()  
  
[47]:  
  
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 10: Removing rows that contains NaN values

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

Unique values of all columns are printed using “. unique()” function. The .unique() function returns a numpy array with each value that is not repeated again in that array. This function is mostly important for loops as in loops we have to use unique data to fetch information out of the raw data provided to us.

```
[38]: read['Unique Key'].unique()

[38]: array([32310363, 32309934, 32309159, ..., 30283424, 30280004, 30281825],
      dtype=int64)

[40]: read['Created Date'].unique()

[40]: <DatetimeArray>
['2015-12-31 23:59:45', '2015-12-31 23:59:44', '2015-12-31 23:59:29',
 '2015-12-31 23:57:46', '2015-12-31 23:56:58', '2015-12-31 23:56:30',
 '2015-12-31 23:55:32', '2015-12-31 23:54:05', '2015-12-31 23:53:58',
 '2015-12-31 23:52:58',
 ...
 '2015-03-29 00:42:48', '2015-03-29 00:37:15', '2015-03-29 00:35:28',
 '2015-03-29 00:35:23', '2015-03-29 00:35:04', '2015-03-29 00:34:32',
 '2015-03-29 00:33:28', '2015-03-29 00:33:03', '2015-03-29 00:33:02',
 '2015-03-29 00:33:01']
Length: 251970, dtype: datetime64[ns]

[42]: read['Closed Date'].unique()

[42]: <DatetimeArray>
['2016-01-01 00:55:00', '2016-01-01 01:26:00', '2016-01-01 04:51:00',
 '2016-01-01 07:43:00', '2016-01-01 03:24:00', '2016-01-01 01:50:00',
 '2016-01-01 01:53:00', '2016-01-01 01:42:00', '2016-01-01 08:27:00',
 '2016-01-01 01:17:00',
 ...
 '2015-03-29 00:57:23', '2015-03-29 02:57:41', '2015-03-29 01:02:39',
 '2015-03-29 04:14:27', '2015-03-29 08:41:24', '2015-03-29 02:52:28',
 '2015-03-29 01:13:01', '2015-03-29 02:33:59', '2015-03-29 04:38:35',
 '2015-03-29 04:41:50']
Length: 231991, dtype: datetime64[ns]
```

Figure 11: Unique values of “unique key”, “created date” and “closed date”

```

[44]: read['Agency'].unique()

[44]: array(['NYPD'], dtype=object)

[46]: read['Complaint Type'].unique()

[46]: array(['Noise - Street/Sidewalk', 'Blocked Driveway', 'Illegal Parking',
          'Derelict Vehicle', 'Noise - Commercial',
          'Noise - House of Worship', 'Posting Advertisement',
          'Noise - Vehicle', 'Animal Abuse', 'Vending', 'Traffic',
          'Drinking', 'Noise - Park', 'Graffiti', 'Disorderly Youth'],
          dtype=object)

[48]: read['Descriptor'].unique()

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

```

Figure 12: Unique values of “agency”, “complaint type” and “descriptor”


```

[50]: read['Location Type'].unique()

[50]: array(['Street/Sidewalk', 'Club/Bar/Restaurant', 'Store/Commercial',
        'House of Worship', 'Residential Building/House',
        'Residential Building', 'Park/Playground', 'Vacant Lot',
        'House and Store', 'Highway', 'Commercial', 'Roadway Tunnel',
        'Subway Station', 'Parking Lot'], dtype=object)

[52]: read['Incident Zip'].unique()

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

```

Figure 13: Unique values of “location type” and “incident zip”

```
[54]: read['City'].unique()

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

[56]: read['Status'].unique()

[56]: array(['Closed'], dtype=object)

[58]: read['Resolution Description'].unique()

[58]: array(['The Police Department responded and upon arrival those responsible for the condition were gone.',
        'The Police Department responded to the complaint and with the information available observed no evidence of the violation at that time.',
        'The Police Department responded to the complaint and took action to fix the condition.',
        'The Police Department issued a summons in response to the complaint.',
        'The Police Department responded to the complaint and determined that police action was not necessary.',
        'The Police Department reviewed your complaint and provided additional information below.',
        'Your request can not be processed at this time because of insufficient contact information. Please create a new Service Request on 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.',
        'Your complaint has been forwarded to the New York Police Department for a non-emergency response. If the police determine the vehicle is illegal, they will take appropriate action.'], dtype=object)
```

Figure 14: Unique values of “city”, “status” and “resolution description”

```
[60]: read['Borough'].unique()

[60]: array(['MANHATTAN', 'QUEENS', 'BRONX', 'BROOKLYN', 'STATEN ISLAND'],
        dtype=object)

[62]: read['Latitude'].unique()

[62]: array([40.86568154, 40.77594531, 40.87032452, ..., 40.77664592,
        40.70635259, 40.71605291])

[64]: read['Longitude'].unique()

[64]: array([-73.92350096, -73.91509394, -73.88852464, ..., -73.94880526,
        -73.87124456, -73.9913785 ])

[66]: read['Request_Closing_Time'].unique()

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

Figure 15: Unique values of “borough”, “latitude”, “longitude” and “request_closing_time”

3. Data Analysis

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

To show the summary statistics of sum, mean, standard deviation, skewness and kurtosis of the data frame, we first need to convert the 'Request_Closing_Time' column to datatype float so that we can evaluate the data according to the time taken to close the case.

Using the '.dt.total_seconds()' function we convert the timedelta datatype to float data type where the time is shown in seconds, for the time to show in hours, we divide the number by 3600, and store the changed time data in new column, 'closing_time_in_seconds'.

```
[59]: read['closing_time_in_seconds'] = read['Request_Closing_Time'].dt.total_seconds() / 3600

[67]: read.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300698 entries, 0 to 300697
Data columns (total 16 columns):
 #   Column                Non-Null Count  Dtype
---  -
 0   Unique Key            300698 non-null int64
 1   Created Date           300698 non-null datetime64[ns]
 2   Closed Date            298534 non-null datetime64[ns]
 3   Agency                 300698 non-null object
 4   Complaint Type         300698 non-null object
 5   Descriptor             294784 non-null object
 6   Location Type          300567 non-null object
 7   Incident Zip           298083 non-null float64
 8   City                   298084 non-null object
 9   Status                 300698 non-null object
10  Resolution Description  300698 non-null object
11  Borough                300698 non-null object
12  Latitude                297158 non-null float64
13  Longitude               297158 non-null float64
14  Request_Closing_Time    298534 non-null timedelta64[ns]
15  closing_time_in_seconds 298534 non-null float64
dtypes: datetime64[ns](2), float64(4), int64(1), object(8), timedelta64[ns](1)
memory usage: 36.7+ MB
```

Figure 16: Creating new column with float datatype

Summary of statistics of sum, mean, standard deviation, skewness and kurtosis of the dataframe is shown below.

The mean of 'closing_time_in_seconds' shows the average time taken to solve any issue that is recorded by 311 calls, and the standard deviation shows that the data is highly scattered as the standard deviation is higher than the mean of the data, and the numbers are far from the average.

For finding out the mean and standard deviation we have used the ".describe" function of pandas. The ".describe" function gives the count, mean, standard deviation, min, 25% or 25 percentile, 50% or 50 percentile or median, 75% or 75 percentile, and max of the numerical columns. Here using ".loc" we select the specific rows that are specified via indexes in square brackets, which is "[['mean', 'std']]". The "include = float" is responsible for only showing mean and standard deviation of columns that has datatype float.

The sum of 'closing_time_in_seconds' shows the total time taken to solve the issue. For finding out the sum ".sum" method was used; here however the sum of "incident zip", "latitude" and "longitude" does not give much insight into analysis, the last sum which is "closing_time_in_seconds" can give somewhat information about how much time the NYPD has spent in 311 calls.

The other functions that are used while performing the sum are "numeric_only = True" which helps filter out the columns that has numbers in it, as strings can not be added using ".sum()". Similar to finding out mean and standard deviation, ".loc()" function was used to only show sum of "incident zip", "latitude", "longitude" and "closing_time_in_seconds".

The skewness of data is positive in 'closing_time_in_seconds', which shows that the graph is tilted towards the right side, or the tail is more in the right side of the graph. This gives the information that some cases are taking longer to solve.

For finding out the skewness the ".skew()" function was used, it is a built in function that calculates the skewness of different columns of a data frame. The "numeric_only = True" and ".loc()" function is also used here to only get skewness of numeric columns, specifically of "incident_zip", "latitude", "longitude" and "closing_time_in_seconds".

The kurtosis of the 'closing_time_in_seconds' is positive, indicating that the graph has heavier tails. This indicates that most of the cases recorded by 311 calls take around the same time to close, however some cases are extreme and can take much more time to close than average closing time.

The functions used here are ".kurt" function and ".loc()" function which helps to find the kurtosis of the given columns and specifies exactly which columns respectively. "numeric_only = True" is also used here which specifies the ".kurt()" function that only columns with numeric values are to be taken to find out the kurtosis.

```
[81]: read.describe(include = float).loc[['mean', 'std']]
```

	Incident Zip	Latitude	Longitude	closing_time_in_seconds
mean	10857.977349	40.725681	-73.925035	4.308926
std	580.280774	0.082411	0.078654	6.062641

```
[83]: read.sum(numeric_only = True).loc[['Incident Zip', 'Latitude', 'Longitude', 'closing_time_in_seconds']]
```

```
[83]: Incident Zip          3.160833e+09
Latitude          1.185553e+07
Longitude         -2.152010e+07
closing_time_in_seconds  1.254358e+06
dtype: float64
```

```
[85]: read.skew(numeric_only = True).loc[['Incident Zip', 'Latitude', 'Longitude', 'closing_time_in_seconds']]
```

```
[85]: Incident Zip          -2.553956
Latitude           0.123114
Longitude          -0.312739
closing_time_in_seconds  14.299525
dtype: float64
```

```
[87]: read.kurt(numeric_only = True).loc[['Incident Zip', 'Latitude', 'Longitude', 'closing_time_in_seconds']]
```

```
[87]: Incident Zip          37.827777
Latitude          -0.734818
Longitude           1.455600
closing_time_in_seconds  849.777081
dtype: float64
```

Figure 17: Summary of statistics

- b) Write a Python program to calculate and show correlation of all variables.

The correlation between two columns is given by “.corr()” function. In the image below there is a data showcasing correlation between different columns of the data set provided to us.

The correlation between “Incident Zip” and ‘closing_time_in_seconds’ is weak as the correlation number of “Incident Zip” is closer to zero.

The correlation of “Latitude”/ “Longitude” and ‘closing_time_in_seconds’ is almost nonexistent as the correlation number of those columns is very close to zero.

The functions used are “.corr()” which helps find the correlation of different columns, with the specified column, the “.loc()” is also used to specify which columns are supposed to be correlated with the given column which is ‘closing_time_in_seconds’. The “numeric_only = True” tells the corr function that only numeric columns are to be considered for correlating.

```
[89]: read.corr(numeric_only=True)['closing_time_in_seconds'].loc[['Incident Zip', 'Latitude', 'Longitude', 'closing_time_in_seconds']]

[89]: Incident Zip          0.057182
      Latitude             0.024497
      Longitude            0.109724
      closing_time_in_seconds  1.000000
      Name: closing_time_in_seconds, dtype: float64
```

Figure 18: Correlation of all variables

4. Data Exploration

For data exploration part of this coursework, matplotlib has been used. Matplotlib is a built-in python library which was developed by neurobiologist John Hunter. Matplotlib is a library that is capable of producing high quality graphs that are suitable for publication. Another mission of creating this library was to make plotting easier, even the complex ones (matplotlib.org, 2012).

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

- For the first data mining, the graph chosen to represent the data is a bar graph. Here the different functions of matplotlib library were used to showcase the bar graph. The functions used are, “figure” function which helps set the height and width of the bar graph, “.bar” function is used to determine which values are to be taken into account to make a bar of the graph, “.xlabel” and “.ylabel” is used to give names to the x-axis and y-axis respectively, the “.xticks” and “.yticks” are used to set the values of x-axis and y-axis. The “.title” is used to give title to the graph, and “.legend()” is used to give the information about what the bar represents. Lastly the “.show()” is responsible for showing the graph.

```
[96]: a = read["Complaint Type"].head(100).value_counts()
plt.figure(figsize=(23,6))
plt.bar(a.index, a.values, label="No. of Complaints")
plt.xlabel("Complaint Type", fontsize=20)
plt.ylabel("Counts", fontsize=20)
plt.yticks(np.arange(0,a.max()+5,5))
plt.xticks(rotation = 90, fontsize=20)
plt.title("Number of Complaints per Complaint", fontsize=20)
plt.legend()
plt.show()
```

Figure 19: Data exploration - maximum number of complaints reported (code)

The bar graph below shows which complaint type is most repeated in the first one hundred complaints. “Blocked Driveway” was reported the most with about 28-29 complaints out of 100 complaints, followed by noise-commercial and illegal parking.

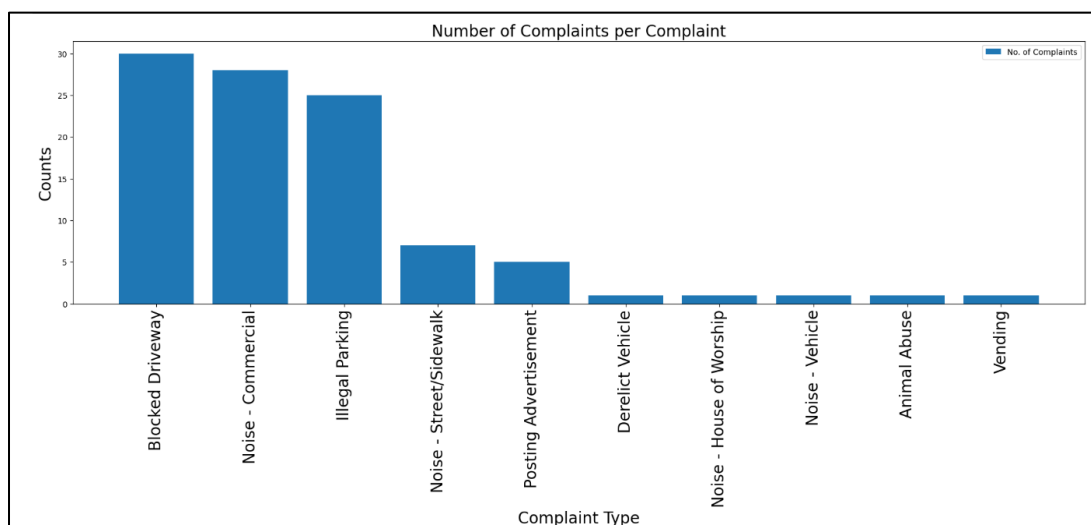


Figure 20: Data exploration - maximum number of complaints reported (visualization)

From this data visualization we can conclude that blocked driveway seems to be the biggest problem of the city. Similarly, noise-commercial and illegal parking is also a problem that needs to be solved. The other problems like noise-street/sidewalk, posting advertisement, derelict vehicle, noise-house of worship, noise-vehicle, animal abuse and vending are not reported as much as the top three problems, but still needs attention in some parts of it.

- For the second data mining, the same bar chart was used, but this time the bar chart represents which borough gets the most complaints out of all the boroughs. From the graph representation below we can say that Queens has the most complaints reported, followed by Brooklyn, which has the second highest number of complaints.

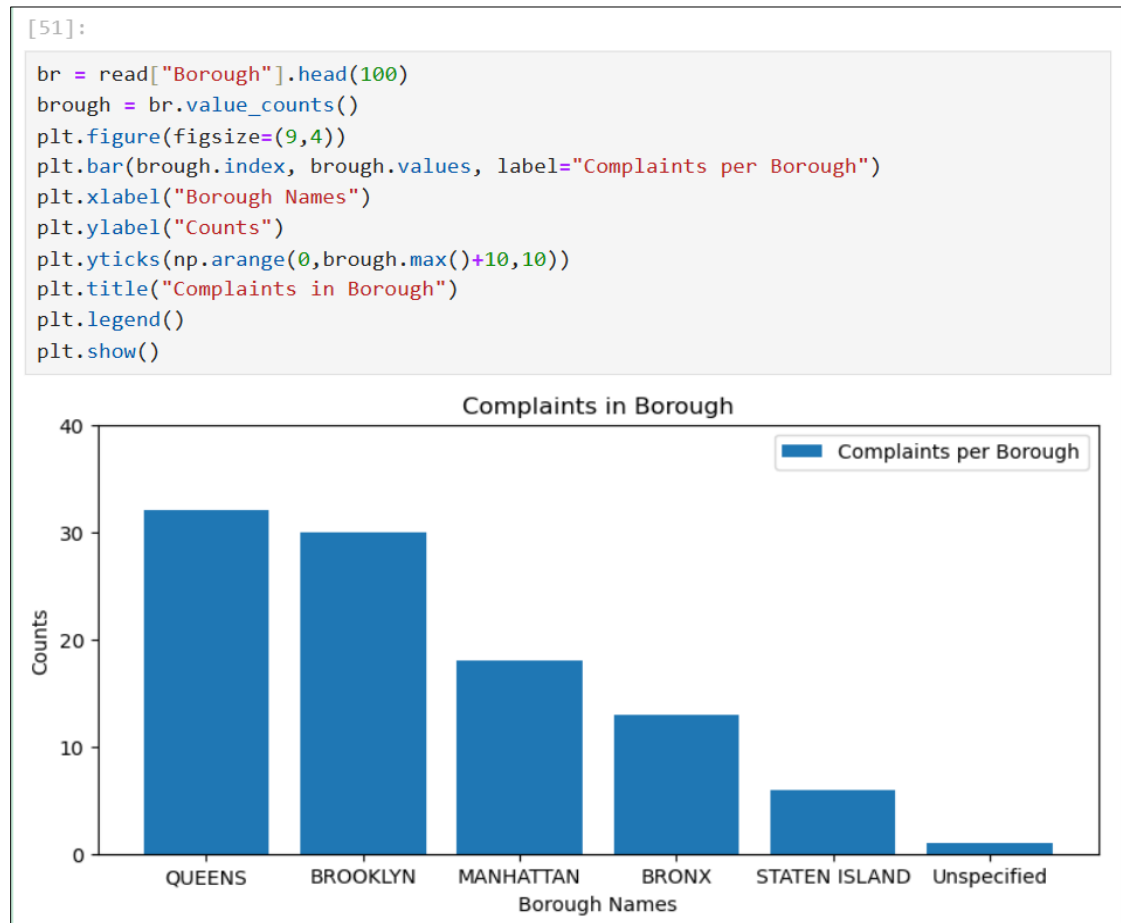


Figure 21: Data Exploration - borough that receives maximum complaints

- For the third data mining, scatter chart was taken into consideration. Similar to bar graph, different functions of matplotlib were used; most of the functions are the same as used to display bar graph, however for scatter plot, instead of “.bar” “.scatter()” function was used, which helps display the scatter graph.

```
[122]: complaint = read["Complaint Type"].head(100)
closing_time = read["closing_time_in_seconds"].head(100)
plt.figure(figsize=(10,3))
plt.scatter(complaint, closing_time, label="Closed Time")
plt.xlabel("Complaint Type")
plt.ylabel("Time in Hours")
plt.yticks(np.arange(0,closing_time.max()+10,10))
plt.xticks(rotation = 90, fontsize=10)
plt.title("Number of Complaints")
plt.legend()
plt.show()
```

Figure 22: Data Exploration - hours taken for complaints to close down (code)

The scatter chart shown below represents the hours it took for each complaint to resolve.

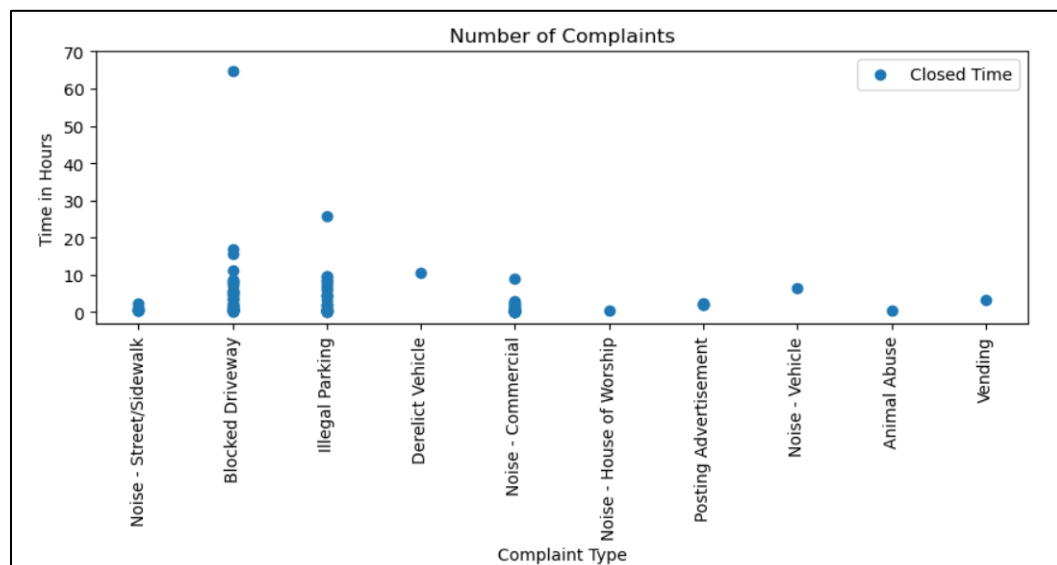


Figure 23: Data Exploration - hours taken for complaints to close down (visualization)

From the scatter chart we can see that blocked driveway's data is scattered the most, so we can conclude that for the blocked driveway complaint, there is no set time of how much it takes to resolve a problem, some might take less time than others. Similarly, illegal parking and noise-commercial data is also scattered, showing that they also do not have a set time as to how much it would take for one specific complaint of that complaint type to resolve.

- For the fourth data mining, the bar graph was taken into consideration. Here firstly “.dt.month” is used to extract only the month of the “created date”, so that we can use it as values of y-axis, to represent the month of the year. Then “.groupby(‘Complaint Type’)” function is used which groups the data frame according to the complaint types, the “[‘months’]” allows only the months column to be grouped by complaint types. In the same line “.agg()” function along with lambda function, “.mode()” function, “.iloc[0]” and “.reset_index()” functions are used.

The “.agg()” function allows different aggregate functions to be applied to the grouped column, the lambda function basically is inline function for quick manipulation, here it receives the series of months and helps apply the “.mode” function is used to extract the one number or one month that is repeated the most in that list. The “.iloc[0]” is used to choose the first number if there is more than one mode in a list. Lastly the “.reset_index()” function is applied to the whole of data frame and is used to reset index to “0,1, 2...”, as previously the index was complaint types.

Similar to the previous bar graphs, the same functions are repeated here to display the bar graph.

```
[183]: read['months'] = read['Created Date'].dt.month
frequent = pd.DataFrame(read.groupby('Complaint Type')['months'].agg(lambda x: x.mode().iloc[0])).reset_index()
plt.figure(figsize=(60,20))
plt.bar(frequent['Complaint Type'], frequent['months'], label = "Month the complaint occurs the most")
plt.xlabel("Complaint Type", fontsize=40)
plt.ylabel("Months", fontsize=40)
plt.xticks(np.arange(0,13,1), fontsize=40)
plt.xticks(fontsize=40, rotation = 90)
plt.title("Maximum Occurance of Complaints in different month", fontsize=40)
plt.legend(fontsize=40)
plt.show()
```

Figure 24: Data Exploration - Maximum reoccurrence of complaint type in months (code)

The bar graph shown below shows which complaint is hiked in which month of the year. This will help us find the trend of complaints.

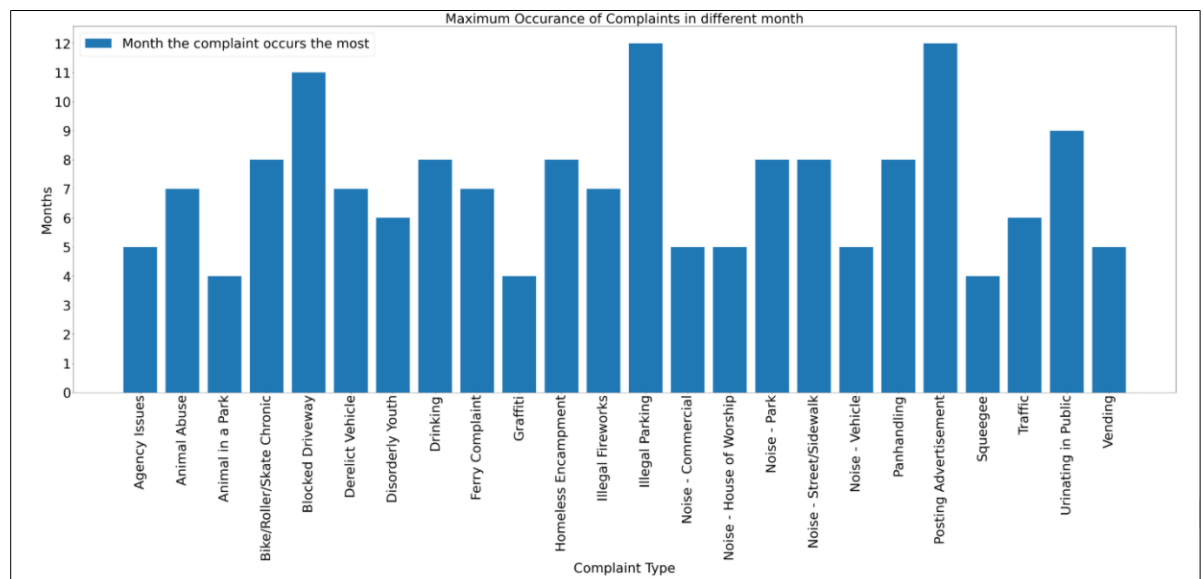


Figure 25: Data Exploration - Maximum reoccurrence of complaint type in months (visualization)

From the representation of data shown above we can say that noise-street/sidewalk peaks in the eighth month, while blocked driveway peaks in the eleventh month, illegal parking peaks in the twelfth month, and so on.

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

For the second question of data exploration, we have to find the average time of all the complaint types and check the average time taken for closing all those complaint in that city.

For this we have first group the 'closing_time_in_seconds' according to city and complaint type, then mean of that column ('closing_time_in_seconds') is extracted and indexes are reset to "0,1,2..." from indexes city and complaint type.

The variable city can change according to the graph to be represented, the line "data[data['City'] == city]" stores only the data frame of city, complaint type and closing time of the city that is written in "city" variable.

The graph represented here is bar graph, similar to previous graphs here too matplotlib functions are used.

```
[50]: df = read[['Complaint Type', 'City', 'closing time in seconds']].head(100)
      data = df.groupby(['City', 'Complaint Type'])['closing time in seconds'].mean().reset_index()
      city = 'NEW YORK'
      g = data[data['City'] == city]
      plt.figure(figsize=(14,4))
      plt.bar(g['Complaint Type'], g['closing time in seconds'], label = "Average time taken to solve a complaint")
      plt.xlabel("Complaint Type")
      plt.ylabel("Average Time")
      plt.xticks(np.arange(0,g['closing time in seconds'].max()+5,5))
      plt.title("Complaint in NEW YORK and average time taken to solve them")
      plt.legend()
      plt.show()
```

Figure 26: Data Exploration - second part (code)

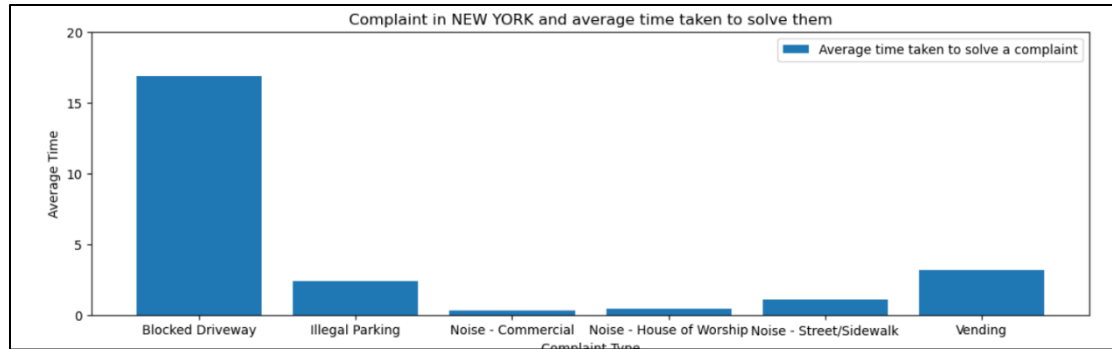


Figure 27: Data exploration – second part (average time taken to solve complaint in New York)

The graph above shows the average time taken for different complaint types to get resolved in New York City. The complaint type “Blocked Driveway” takes the maximum average of sixteen hours to get resolved, whereas complaint type “vending” takes about four hours average to get resolved. The least average time taken to solve a complaint type is about one hour for complaint type “noise – Commercial”.

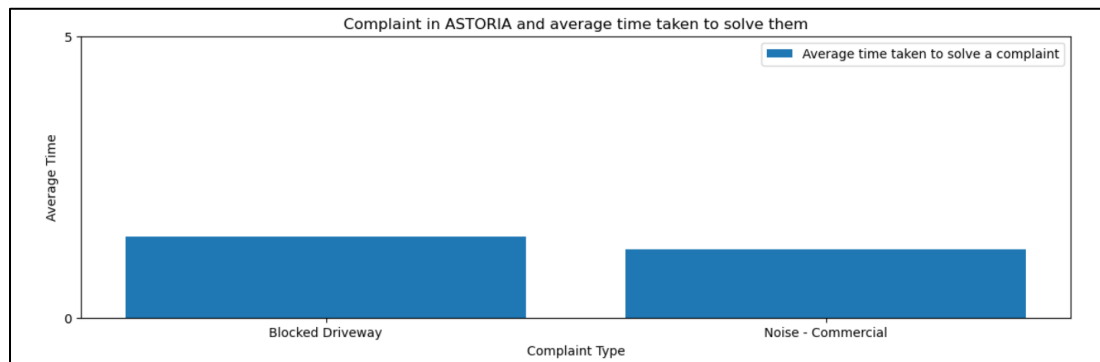


Figure 28: Data exploration – second part (average time taken to solve complaint in Astoria)

The graph above shows the average time taken for different complaint types to get resolved in Astoria. The complaint type “Blocked Driveway” takes the maximum average of two hours to get resolved, whereas the least average time taken to solve a complaint type is about one hour for complaint type “noise – Commercial”.

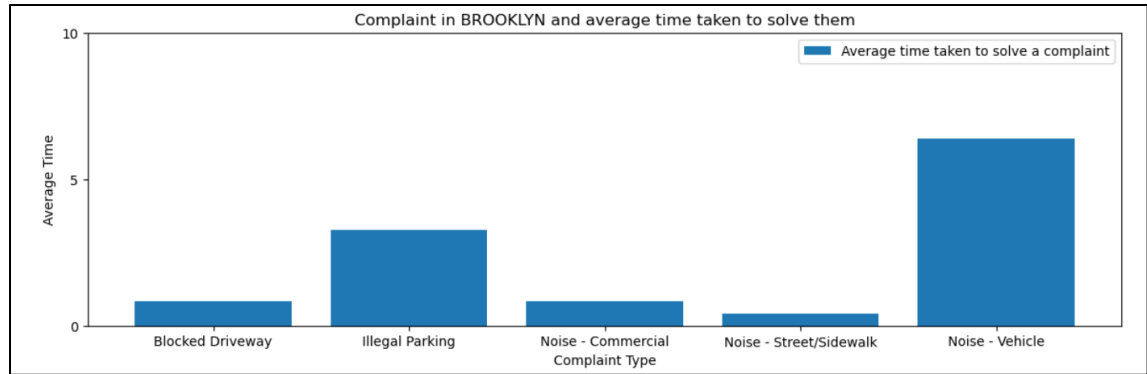


Figure 29: Data exploration – second part (average time taken to solve complaint in Brooklyn)

The graph above shows the average time taken for different complaint types to get resolved in Brooklyn. The complaint type “Noise – Vehicle” takes the maximum average of seven hours to get resolved, whereas the least average time taken to solve a complaint type is about one hour for complaint type “noise – Street/Sidewalk”.

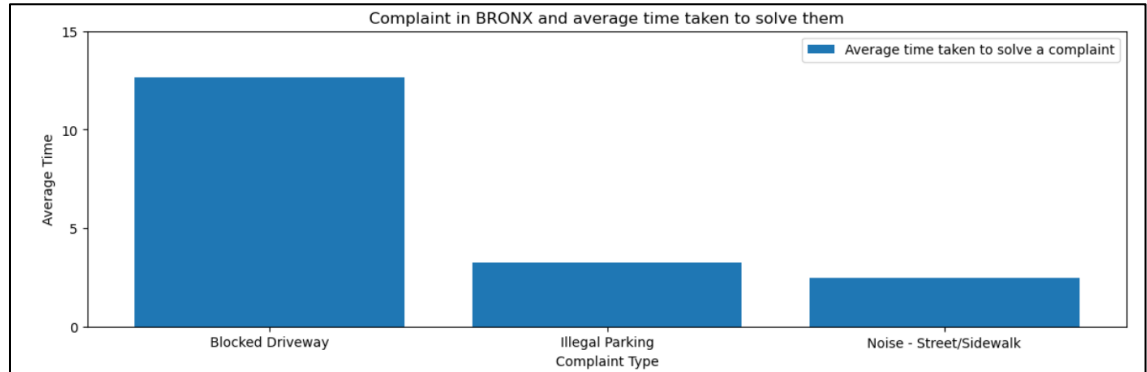


Figure 30: Data exploration – second part (average time taken to solve complaint in Bronx)

The graph above shows the average time taken for different complaint types to get resolved in Bronx. The complaint type “Blocked Driveway” takes the maximum average of thirteen hours to get resolved, whereas the least average time taken to solve a complaint type is about two hours for complaint type “noise – Street/Sidewalk”.

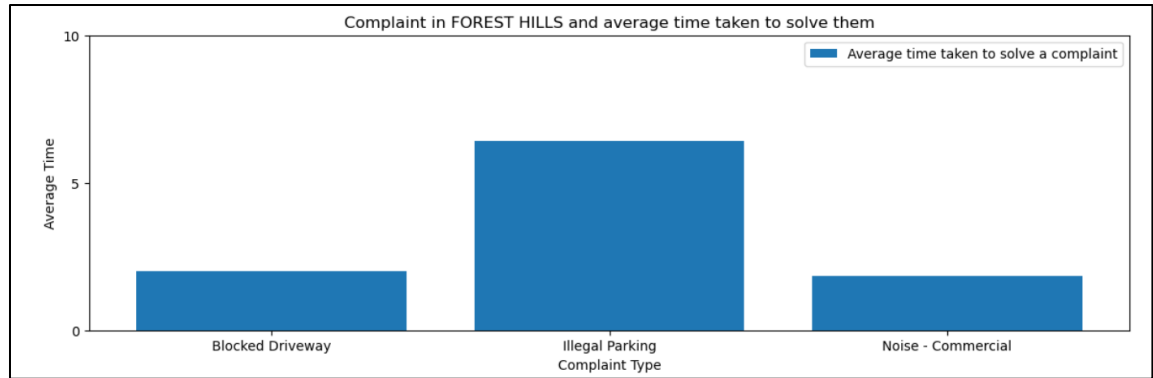


Figure 31: Data exploration – second part (average time taken to solve complaint in Forest Hills)

The graph above shows the average time taken for different complaint types to get resolved in Forest Hills. The complaint type “Illegal Parking” takes the maximum average of seven hours to get resolved, whereas the least average time taken to solve a complaint type is about two hours for complaint type “noise – Commercial” and “Blocked Driveway”.

5. Statistical Testing

For statistical testing, Anova and chi-test are used. Anova is a statistical test that compares mean value of three or more groups to determine whether those groups have statical significant differences or not (GeeksforGeeks, 2025).

On the other hand, chi-test is used to check the relationship between categories. It states that the difference between outcomes of two categories is large not because of coincidence, suggesting that there is a chance of relation between the outcome and the category (GeeksforGeeks, 2025).

For the tests we will be using SciPy.Stats library which is a sub package of SciPy library. This library is usually used for “probabilistic distribution” and “statical operations” (GeeksforGeeks, 2025). In the two tastings below, chi2_contingency and f_oneway functions are used.

The chi2_contingency function is used to perform the chi squared test on the contingency table to find the relation between two categorical values i.e. complaint type and city or location, whereas f_oneway function is used to perform one-way Anova test on the given groups of average time of each complaint type, to check whether the response time across all complaint type is similar or not.

a) Test 1

Whether the average response time across complaint types is similar or not.

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

Null Hypothesis (H0) – the average response time across complaint type is similar, i.e., complaint type does not affect the average response time.

Alternate Hypothesis (H1) - the average response time across complaint type is not similar, i.e., the complaint type affects the average response time.

```
[27]:  
  
res = read[['Complaint Type', 'closing_time_in_seconds']].dropna()  
top_complaints = res['Complaint Type'].value_counts().head(5).index  
fil = res[res['Complaint Type'].isin(top_complaints)]  
  
groups = [group['closing_time_in_seconds'].values  
           for name, group in fil.groupby('Complaint Type')]  
  
f_stat, p_val = stats.f_oneway(*groups)  
  
print("F-statistic:", f_stat)  
print("P-value:", p_val)  
  
if p_val <= 0.05:  
    print('We can reject the null hypothesis')  
else:  
    print('We can accept the null hypothesis')  
  
F-statistic: 1799.600524153762  
P-value: 0.0  
We can reject the null hypothesis
```

Figure 32: Test 1 - average response time across complaint types

The p-value is 0.0 which is less than 0.05, so we can reject the null hypothesis.

Rejecting the null hypothesis means that the average response time across complaint type is not similar, and the complaint type does affect the average response time.

To conclude to this result different functions were used, like “.dropna()” function that drops Na values if there is any in the data frame, “.value_counts()” which counts the number of times a specific complaint type is repeated in that data frame, whereas “.head(5)” extracts only the top five complaints out of all the unique complaints, and “.index” extracts the names of the complaint types as index.

The “df[df['Complaint Type'].isin(top_complaints)]” creates a data frame that contains rows of only the top five complaint types, whereas “[group['closing_time_in_seconds'].values for name, group in fil.groupby('Complaint Type')]” creates a list of arrays of the closing time in respect to the complaint types, later “(*groups)” is used to unpack the list and pass each array as different group to f_oneway function.

Here, f_stat and p_val is also used, f_stat or f statistics is a value that is calculated and returned by f_oneway, and it numerically describes the variance between the numbers of an array and its mean, and also the mean of all arrays with the grand total mean (GeeksforGeeks, 2023), whereas p_val or p value is a value that is calculated after the f_stat is calculated, and it helps accept or reject the null hypothesis. If p value is smaller than 0.05 the null

hypothesis is rejected, whereas if the p value is greater than 0.05 the null hypothesis is accepted.

b) Test 2

Whether the type of complaint or service requested, and location are related.

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

Null Hypothesis (H0) – the complaint type and location are not related.

Alternate Hypothesis (H1) – the complaint type and location are related.

```
[266]:  
dat = pd.crosstab(read['Complaint Type'], read['City'])  
  
test, p_val, dof, expected_val = chi2_contingency(dat)  
alpha = 0.05  
print("The p-value of our test is " + str(p_val))  
  
if p_val <= alpha:  
    print('We can reject the null hypothesis')  
else:  
    print('We can accept the null hypothesis')  
  
The p-value of our test is 0.0  
We can reject the null hypothesis
```

Figure 33: Test 2 - relation of complaint type and location

The p-value of the chi test is 0.0 which is less than 0.05, so we can reject the null hypothesis and say that the two variables, complaint type and city or location is related to one another.

For this conclusion, different functions were used, the first one is “.crosstab()” function which creates a contingency table or a frequency table which compares two categorical data, here in this test, the contingency table or frequency table consists of comparison between complaint type and city.

The “chi2_contingency” function returns four values, first one is the test value, which is a numerical value that describes the difference between observed value and expected value, the higher the difference between them the more dependent the two categories are. The second returned value is p value, which is calculated after the test value is calculated, and it helps accept or reject the null hypothesis. The third value returned is degrees of freedom or dof,, and the fourth value returned is expected value, which is hypothetical value presented if there is no relationship between the two categorical columns.

6. Bibliography

- GeeksforGeeks, 2023. *How to Perform an F-Test in Python*. [Online]
Available at: <https://www.geeksforgeeks.org/how-to-perform-an-f-test-in-python/>
[Accessed 15 May 2025].
- GeeksforGeeks, 2025. *How to Perform a One-Way ANOVA in Python*. [Online]
Available at: <https://www.geeksforgeeks.org/how-to-perform-a-one-way-anova-in-python/>
[Accessed 14 May 2025].
- GeeksforGeeks, 2025. *Python - Pearson's Chi-Square Test*. [Online]
Available at: <https://www.geeksforgeeks.org/python-pearsons-chi-square-test/>
[Accessed 14 May 2025].
- GeeksforGeeks, 2025. *SciPy - Stats*. [Online]
Available at: <https://www.geeksforgeeks.org/scipy-stats/>
[Accessed 14 May 2025].
- matplotlib.org, 2012. *Mission Statement*. [Online]
Available at: <https://matplotlib.org/stable/project/mission.html>
[Accessed 4 May 2025].
- Mother Duck, 2025. *NYC 311 Complaint Data*. [Online]
Available at: <https://motherduck.com/docs/getting-started/sample-data-queries/nyc-311-data/#:~:text=Each%20row%20of%20data%20contains,customer%20who%20made%20the%20request>
[Accessed 9 April 2025].
- numpy.org, 2025. *About Us*. [Online]
Available at: <https://numpy.org/about/>
[Accessed 4 May 2025].
- NYC, 2025. *NYC311*. [Online]
Available at: <https://portal.311.nyc.gov/article/?kanumber=KA-02498#:~:text=NYC311%20can%20help%20with%20a,and%20365%20days%20a%20year>
[Accessed 9 April 2025].
- pydata.org, 2025. *About pandas*. [Online]
Available at: <https://pandas.pydata.org/about/index.html>
[Accessed 4 May 2025].
- SciPy, 2025. *chi2_contingency*. [Online]
Available at:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.chi2_contingency.html
[Accessed 14 May 2025].

SciPy, 2025. *f_oneway*. [Online]

Available at:

https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.f_oneway.html

[Accessed 14 May 2025].