



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: Sneha Agrawal

London Met ID: 23049021

College ID: NP01CP4A230168

Assignment Due Date: Thursday, May 15, 2025

Assignment Submission Date: Tuesday, April 15, 2025





Submitted to: Roshan Shrestha

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.




14% Overall Similarity

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

Match Groups

-  **42 Not Cited or Quoted 12%**
Matches with neither in-text citation nor quotation marks
-  **0 Missing Quotations 0%**
Matches that are still very similar to source material
-  **1 Missing Citation 2%**
Matches that have quotation marks, but no in-text citation
-  **0 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 7%  Internet sources
- 1%  Publications
- 14%  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	7
Introduction.....	7
2.Data Preparation	14
1. Import the dataset	15
1. Without low_memory= False.....	15
2.With low_memory= False.....	16
2. Provide your insight on the information and details that the provided dataset carries.....	17
3. Convert the columns "Created Date" and "Closed Date" to datetime datatype and create a new column "Request_Closing_Time" as the time elapsed between request creation and request closing	18
4. Write a python program to drop irrelevant Columns which are listed below.	19
5. Write a python program to remove the NaN missing values from updated dataframe.	21
6. Write a python program to see the unique values from all the columns in the dataframe.	22
2. Data Analysis	23
1. Write a Python program to show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame.	23
2. Write a Python program to calculate and show correlation of all variables.	24
3. Data Exploration.....	25
1. Key Findings from the Data Visualization.....	25
1.1. The Most Common top 10 Complaint Categories	25
1.2. Time Required to Resolve Requests	27
1.3. Complaints by Borough	29
1.4. Analysing the Request Closing time as per Borough.....	30
1.5. Average Resolution Time by Complaint and Borough	34
4. Statistical Testing	39
• Test 1	39
1.1. State the Null Hypothesis (H0) and Alternate Hypothesis (H1).....	39

1.2.	Perform the statistical test and provide the p-value.	39
1.3.	Interpret the results to accept or reject the Null Hypothesis.....	39
•	Test 2	40
2.1.	State the Null Hypothesis (H_0) and Alternate Hypothesis (H_1).	40
2.2.	Perform the statistical test and provide the p-value.	40
2.3.	Interpret the results to accept or reject the Null Hypothesis.....	41
5.	Conclusion	42
	References.....	43

Table of tables

Table 1: Description of Attributes	14
--	----

Table of Figures

Figure 1: Without low memory= false	15
Figure 2: Importing the dataset	16
Figure 3: looking columns information.....	17
Figure 4: Viewing describe	17
Figure 5: Converting to datetime datatype	18
Figure 6: Adding Request_Closing_Time column	19
Figure 7: Dropping Columns	20
Figure 8: Checking is the columns are dropped	20
Figure 9: Removing NaN.....	21
Figure 10: Viewing unique values	22
Figure 11: Summery statistics	23
Figure 12: Correlation.....	24
Figure 13: top 10 Complaint Categories.....	25
Figure 14:Graph of top 10 Complaint Categories.....	26
Figure 15: Finding Request closing time	27
Figure 16: Drawing histogram	28
Figure 17: Complaints of request itime more than 100.....	28
Figure 18: Borough with top 6 complaints	29
Figure 19: Using pie chart for complain given by borough	30
Figure 20: Using scatter plot code for seeing request closing time as per borough	31
Figure 21: Using scatter plot to see the trends of request closing time as per borough.....	32
Figure 22: Using bar plot for seeing request closing time as per borough.....	33
Figure 23: Making pivot table	34
Figure 24: Making overall average variable.....	35
Figure 25: Code to see slowest request closing time as per complaints of each borough	35
Figure 26: line plot to see slowest request closing time as per complaints of each borough.....	36
Figure 27: Group bar Code to see slowest request closing time as per complaints of each borough	37
Figure 28: bar chart to see slowest request closing time as per complaints of each borough.....	38

1.Data Understanding

Introduction

The purpose of this report is to analyze the dataset from New York City's 311 Customer Service Request system. This dataset contains more than 300,000 entries the captures the wide range of the complaints starting from the year 2010 onward. The recorded complaints include only the non-emergency issues that are reported by the citizens, for example: noise disturbances, water leaks, sanitation issues, illegal parking etc. Each

record includes the date and the time of the complaint submission, resolution info, complaint type, descriptors, borough details etc.

Analyzing dataset helps in identifying the patterns regarding the complaint types and how quickly the complaints are solved which helps in understanding the performance of the city services and where the services need attention.

Description and Datatype of some important Attributes:

S.No	Column Name	Description	Data Type
1.	Unique Key	A key that gives unique identity to the each row.	int64
2.	Created Date	The time and date when the service requests were submitted by the customer.	object
3.	Closed Date	The time and date when the service requests were solved.	object
4.	Complaint Type	The complaint's title.	object
5.	Descriptor	Description of the Complaint type	object
6.	Location Type	The location from where the	object

		complaint was submitted.	
7.	Incident Zip	ZIP Code of that place from where the complaint was submitted.	float64
8.	Incident Address	Address where the incident took place.	object
9.	Street Name	Street where the incident took place.	object
10.	Cross Street 1	The First Street that intersects near complaint.	object
11.	Cross Street 2	Second Street that intersects near complaint.	object
12.	Intersection Street 1	The intersection's main street	object
13.	Intersection Street 2	The intersection's secondary street.	object
14.	Address Type	The address type intersection or street.	object
15.	City	The city name where the issue was.	object

16.	Landmark	Recognizable landmark to locate the issue place	object
17.	Facility Type	The facility type related to complaint.	object
18.	Status	The complaint status that was recorded	object
19.	Due Date	The deadline for fixing that issue.	object
20.	Resolution Description	The description of how the problem was fixed.	object
21.	Resolution Action Updated Date	The actions update date	object
22.	Community Board	Local organizations in charge of the certain area.	object
23.	Borough	The Borough from where the complaint was filled	object
24.	X Coordinate (State Plane)	x-axis coordinate to from service centre to the complaint location	float64
25.	Y Coordinate (State Plane)	y-axis coordinate from service	float64

		centre to the complaint location	
26.	Park Facility Name	Name of the Park Facility form where the complaint was filed.	object
27.	Park Borough	The location of the park which is located in the borough	object
28.	School Name	The Name of the school from where the complaint was issued.	object
29.	School Number	The Number of the school from where the complaint was issued.	object
30.	School Region	The Region of the city school from where the complaint of school was issued.	object
31.	School Code	The code of the school given by the authorities.	object

32.	School Phone Number	. School's Phone Number	object
33.	School Address	The Address of the school from where the complaint was issued.	object
34.	School City	The city from where that school is.	object
35.	School State	The state from where that school is.	object
36.	School Zip	The school's zip code.	object
37.	School Not Found	The situation where the school was not found in the location.	object
38.	School or City-wide Complaint	The complaint related to school or Complaints that affected multiple area.	float64
39.	Vehicle Type	The vehicle type linked to the complaint.	float64
40.	Taxi Company Borough	The Borough from the taxi company is.	float64

41.	Taxi Pick Up Location	The place from where passengers are picked by taxi.	float64
42.	Bridge Highway Name	The name of the highway and bridge from where the complaint was issued.	object
43.	Bridge Highway Direction	The complaint that was issued from the particular part of the bridge.	object
44.	Road Ramp	The name of the Road Ramp .	object
45.	Bridge Highway Segment	The particular area of the bridge from the complaint was issued	object
46.	Garage Lot Name	Garage Lot's Name	float64
47.	Ferry Direction	The transport ferry's direction.	object
48.	Ferry Terminal Name	Name of the ferry ferry station.	object
49.	Latitude	Latitude from where the complaint was made.	float64

50.	Longitude	Longitude from where the complaint was made.	float64
51.	Location	The precise location from where the complaint was made.	object

Table 1: Description of Attributes

Understanding the structure of the data is very important because it helps in identifying the and having detailed info about delay patterns, frequent complaint zones and the performance of the department. This helps in visualizing, aggregating and predicting the future complaints.

2.Data Preparation

Data preparation includes the operations like filtering, modifying and arranging the data so that the unprocessed data can be suitable for analysing. It contains the series of actions and multiple codes through which we manipulate and refine the data which helps in improving the data integrity. Data preparation is used in both data analysis and machine

learning since it helps in deeper analysis and modelling as well as it helps to build smart experienced and smart model.

1. Import the dataset

1. Without low_memmory= False

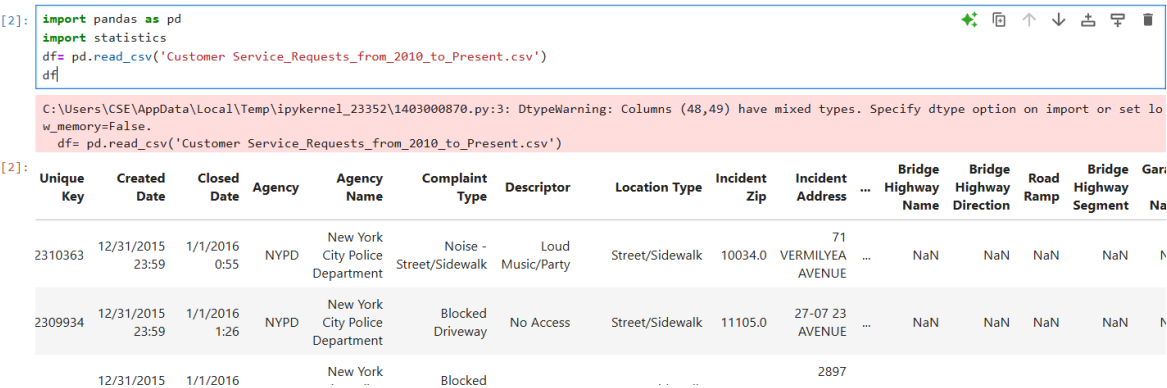


Figure 1: Without low memory= false

2. With low_memory= False

```
[1]:
import pandas as pd
import statistics
df= pd.read_csv('Customer Service Requests from 2010 to Present.csv', low_memory= False)
df
```

[1]:

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location
0	32310363	12/31/2015 23:59	1/1/2016 0:55	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/S
1	32309934	12/31/2015 23:59	1/1/2016 1:26	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/S
2	32309159	12/31/2015 23:59	1/1/2016 4:51	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/S
3	32305098	12/31/2015 23:57	1/1/2016 7:43	NYPD	New York City Police Department	Illegal Parking	Commercial Overnight Parking	Street/S
4	32306529	12/31/2015 23:56	1/1/2016 3:24	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/S
...
300693	30281872	3/29/2015 0:33	NaN	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Re

Figure 2: Importing the dataset

Here, I have imported pandas and statistics which are required to manipulate and analyse the data. And then I have written the code that will read the csv file into the pandas Data Frame called df and store the complete dataset in a df in a structured tabular format.

At first I have written here code without low_memory= false which returned the warning saying that columns have mixed types since, with low-memory = false the pandas will read the large data in chunks, which will cause the datatype to be guessed wrong. So, to solve this issue I have included the code low-memory= false which also removed the warning.

2. Provide your insight on the information and details that the provided dataset carries.

```
df.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
```

Figure 3: looking columns information

The `df.info()` helps in providing the information about column data type and missing values in that column.

```
df.describe()
```

	Unique Key	Created Date	Closed Date	Incident Zip	Latitude	Longitude	Request Closing Time
count	2.911070e+05	291107	291107	291107.000000	291107.000000	291107.000000	291107.000000
mean	3.130158e+07	2015-08-14 11:25:25.319968256	2015-08-14 15:43:57.444651264	10857.977349	40.725681	-73.925035	4.308924
min	3.027948e+07	2015-03-29 00:33:00	2015-03-29 00:57:00	83.000000	40.499135	-74.254937	0.016667
25%	3.079934e+07	2015-06-08 15:38:00	2015-06-08 21:25:00	10314.000000	40.668926	-73.970957	1.266667
50%	3.130675e+07	2015-08-13 22:57:00	2015-08-14 02:50:00	11209.000000	40.717782	-73.930774	2.716667
75%	3.179091e+07	2015-10-19 15:03:00	2015-10-19 20:58:00	11238.000000	40.782973	-73.875788	5.333333
max	3.231065e+07	2015-12-31 23:59:00	2016-01-03 16:22:00	11697.000000	40.912869	-73.700760	592.883333
std	5.753777e+05	NaN	NaN	580.280774	0.082411	0.078654	6.062642

Figure 4: Viewing describe

This code helps us to get the quick overview of the of the basic statics of the number with the help of the columns like count, mean, std, min, 25%, 50%, 75% and max.

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

```
# checking datatype before changing the dtype of created and closed date
print(df['Created Date'].dtype)
print(df['Closed Date'].dtype)
```

```
object
object
```

```
#converting the columns "Created Date" and "Closed Date" to datetime
df['Created Date'] = pd.to_datetime(df['Created Date'])
df['Closed Date'] = pd.to_datetime(df['Closed Date'])
```

```
#now checking dtype
print(df['Created Date'].dtype)
print(df['Closed Date'].dtype)
```

```
datetime64[ns]
datetime64[ns]
```

Figure 5: Converting to datetime datatype

Here, in the first cell I have used the function `pd.to_datetime` to change the datatype of Created Date and Closed Date attribute from object to date and time. In the second cell I have used `.dtype` function to check whether the data type of both the attributes is changed or not.

```
#creating a new column "Request Closing Time"
df['Request Closing Time'] = df['Closed Date'] - df['Created Date']
df
```

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
Loud Music/Party	Street/Sidewalk	10034.0	71 VERMILYEA AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	40.865682	-73.923501	(40.86568153633767, -73.92350095571744)	0 days 00:56:00
No Access	Street/Sidewalk	11105.0	27-07 23 AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	40.775945	-73.915094	(40.775945312321085, -73.91509393898605)	0 days 01:27:00
No Access	Street/Sidewalk	10458.0	2897 VALENTINE AVENUE	...	NaN	NaN	NaN	NaN	NaN	NaN	40.870325	-73.888525	(40.870324522111424, -73.88852464418646)	0 days 04:52:00

Figure 6: Adding Request_Closing_Time column

Here, I have given a code to created the new column called request closing time by calculating the difference between Closed Date and Created Date

4. 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']

```
#before dropping columns
print(df.columns)
```

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

Here, I have used the `df.columns` to see the columns that exists right now in the dataframe

```
#dropping these columns
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']

df.drop(columns=columns_to_drop, axis=1, inplace=True) #axis 1 means column axis 0 means row axis 0 is taken as default value
```

Figure 7: Dropping Columns

Then, I have first stored the irrelevant columns in the variable and then dropped those columns using the function. `drop ()`.

```
print(df.columns)

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

Figure 8: Checking is the columns are dropped

Then, I have used. column function to check if the columns were dropped or not.

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

```
#before dropna  
#any() function returns boolean value for each column  
df.isna().any()
```

Unique Key	False
Created Date	False
Closed Date	True
Agency	False
Complaint Type	False
Descriptor	True
Location Type	True
Incident Zip	True
City	True
Status	False
Resolution Description	False
Borough	False
Latitude	True
Longitude	True
Request Closing Time	True
dtype: bool	

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

```
#after dropna  
df.isna().any()
```

Unique Key	False
Created Date	False
Closed Date	False
Agency	False
Complaint Type	False
Descriptor	False
Location Type	False
Incident Zip	False
City	False
Status	False
Resolution Description	False

Figure 9: Removing NaN

Here, I have used the .dropna function to remove the NaN missing values from dataframe, and used .shape function to check whether the NaN columns were dropped or not since this function gives the rows and columns as an output.

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

```
for column in df.columns:
    print("Unique values in",column,"is")
    print(df[column].unique())
    print('_____')
```

Unique values in Unique Key is
[32310363 32309934 32309159 ... 30283424 30280004 30281825]

Unique values in Created Date is
<DatetimeArray>
['2015-12-31 23:59:00', '2015-12-31 23:57:00', '2015-12-31 23:56:00',
 '2015-12-31 23:55:00', '2015-12-31 23:54:00', '2015-12-31 23:53:00',
 '2015-12-31 23:52:00', '2015-12-31 23:50:00', '2015-12-31 23:48:00',
 '2015-12-31 23:47:00',
 ...
 '2015-03-29 00:49:00', '2015-03-29 00:48:00', '2015-03-29 00:46:00',
 '2015-03-29 00:44:00', '2015-03-29 00:43:00', '2015-03-29 00:42:00',
 '2015-03-29 00:37:00', '2015-03-29 00:35:00', '2015-03-29 00:34:00',
 '2015-03-29 00:33:00']
Length: 194076, dtype: datetime64[ns]

Unique values in Closed Date is
<DatetimeArray>
['2016-01-01 00:55:00', '2016-01-01 01:26:00', '2016-01-01 04:51:00',
 '2016-01-01 07:43:00', '2016-01-01 03:24:00', '2016-01-01 01:50:00',
 '2016-01-01 01:53:00', '2016-01-01 01:42:00', '2016-01-01 08:27:00',
 '2016-01-01 01:17:00',

Figure 10: Viewing unique values

In this program the for loop travels in each column and prints the unique() values of each column. To print the unique values I have used .unique() function, through this I can quickly see diversity of data.

2. Data Analysis

Data Analysis includes numerical evaluation of the datasets for the prediction to access future trends and characteristics, formulating the conclusions and decision making. It includes using a variety of statistical and computational methods to extract the insights from data (Kelley, 2024).

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

```
num_columns= df.select_dtypes(include= ['float64','int64']) #just like df.describe(include='int')

for column in num_columns:
    print('Summery statics of', column, 'is')
    print('Sum =', df[column].sum())
    print('Mean =', df[column].mean())
    print('Standard deviation =', df[column].std())
    print('Skewness =', df[column].skew())
    print('kurtisis =', df[column].kurtosis())
    print('_'*100)
```

```
Summery statics of Unique Key is
Sum = 9112107955295
Mean = 31301576.242738925
Standard deviation = 575377.7387071877
Skewness = 0.016897722422296077
kurtisis = -1.1765926558306596
```

```
Summery statics of Incident Zip is
Sum = 3160833212.0
Mean = 10857.977348535074
Standard deviation = 580.2807740122854
Skewness = -2.553955898131983
kurtisis = 37.827777247855444
```

```
Summery statics of Latitude is
Sum = 11855530.75877829
Mean = 40.72568079358549
Standard deviation = 0.08241087015112669
Skewness = 0.123114382634822
kurtisis = -0.7348182547719988
```

```
Summery statics of Longitude is
Sum = -21520095.167681944
Mean = -73.92503501352404
Standard deviation = 0.07865355626291953
Skewness = -0.3127386420474719
```

Figure 11: Summery statistics

Here, I have used `.select_dtypes()` function to select only float and integer data type columns from the dataset. Then I have used for loop to go in each column with those data type and then used function.

`.sum()` to calculate sum

`.mean()` to calculate mean

`.std()` to calculate standard deviation

`.skew()` to calculate skewness

`.kurt()` to calculate kurtosis

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

```
#finding the correlation matrix between the numerical columns of dataframe  
df.select_dtypes(include= ['float64','int64']).corr()
```

	Unique Key	Incident Zip	Latitude	Longitude
Unique Key	1.000000	0.025492	-0.032613	-0.008621
Incident Zip	0.025492	1.000000	-0.499081	0.385934
Latitude	-0.032613	-0.499081	1.000000	0.368819
Longitude	-0.008621	0.385934	0.368819	1.000000

Figure 12: Correlation

Here, I have used `.select_dtypes()` function to select only float and integer data type columns from the dataset. Then I have used `.corr()` function to see the correlation between all the columns.

3. Data Exploration

This part presents us the four important findings obtained from the visual analysis of the cleaned 311 data. After that, it analyses the duration taken to resolve different types of complaints and compares them across the boroughs.

1. Key Findings from the Data Visualization

1.1. The Most Common top 10 Complaint Categories

To figure out which 10 complaints are mostly made as per my dataset, we used the function `value_count()` and `head()` in my 'Complaint type dataframe'. It returns the top 10 "Complaints Type" along with the number of counts of the same "Complaint Type" in Data Frame which helps us to know what are the most common problems people there face and is reported frequently.

```
#value.count() counts no of time the same complaint occurred.  
# head(10) gives us the top 10 most frequent complaints.  
df['Complaint Type'].value_counts().head(10)
```

[124]:

Complaint Type	
Blocked Driveway	76676
Illegal Parking	74021
Noise - Street/Sidewalk	47747
Noise - Commercial	35144
Derelict Vehicle	17506
Noise - Vehicle	16868
Animal Abuse	7744
Traffic	4466
Noise - Park	3927
Vending	3773

Name: count, dtype: int64

Figure 13: top 10 Complaint Categories

Here, I have used the bar chart to display the top 10 most complaints reported by the people. To access this chart I have used the method `plot(kind="bar")`. It gave me the barchart and the `xticks = 90` helped me to align the name properly in clear format.

```
plt.figure()
df['Complaint Type'].value_counts().head(10).plot(kind='bar')
plt.title('Top 10 Complaints given by people.')
plt.xlabel('Complaint Type')
plt.ylabel('No of Requests')
plt.xticks(rotation=90)
plt.show()
```

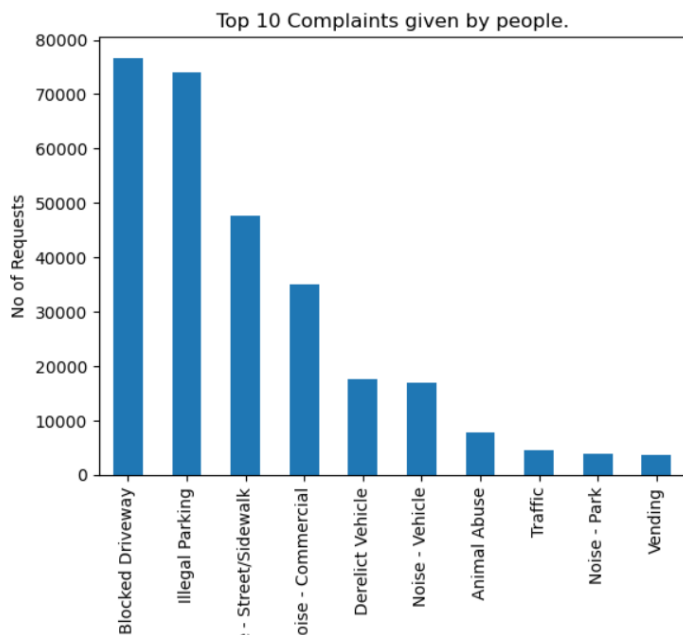


Figure 14: Graph of top 10 Complaint Categories

In this Graph, we can see that the people are facing the blocked Driveways problem a lot and this issue has the highest occurrence among all other in the graph. The second highest “Complaint Type” occurrence is the Illegal Parking with the slight difference with the Blocked driveway, both are above and around 75000 and the other complaints type are below 5000

1.2. Time Required to Resolve Requests

[126]:

```
# .dt.total_seconds() this converts all the hours,min,sec to total secends.  
df['Request Closing Time'] = df['Request Closing Time'].dt.total_seconds() / 3600
```

[128]:

```
print(df['Request Closing Time'].dtype)
```

float64

Figure 15: Finding Request closing time

Here, we have changed the request closing time value from datetime to hours by dividing it with 3600 and I also changed the dtype to the float64 so that I can compare it with multiple conditions.

Here, I have Analysed the trend of request closing time of the complaints using the histogram, to access the histogram function I have used the function `hist(bins=())` and the range in my histogram was defined by the code inside my bins which is `bins= (range(0, 500, 10))`

```
plt.figure(figsize= (10,6))
df["Request Closing Time"].hist(bins= (range(0,500,10)))
plt.title("Trend of Request closing time of Complaints")
plt.xlabel("Request Closing Time")
plt.ylabel("No of complaints")
plt.show()
```

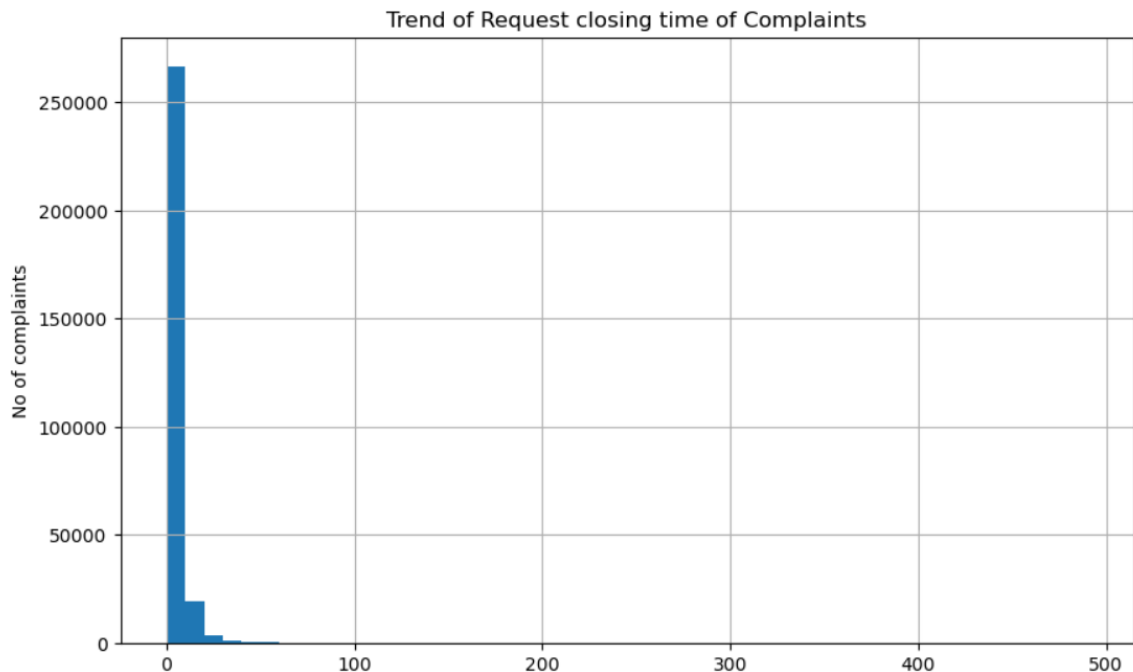


Figure 16: Drawing histogram

```
print('The no of requests that took more than 100 hours to close:',(df['Request Closing Time'] > 100).sum())
```

The no of requests that took more than 100 hours to close: 79

Figure 17: Complaints of request itime more than 100

The above histogram shows that the majority of the Complaints were closed below 40 hours. But, some of the cases do take much longer time, which is indicated by the code `(df['Request Closing Time'] > 100).sum()` it could not be shown on histogram because they were too small to be seen on Histogram.

1.3. Complaints by Borough

To figure out which 5 Borough have most complaints reported, we used the function `value_counts()` and `head()` in my 'Borough dataframe'. It returns the top 5 "Borough" along with the number of counts of the same "Borough" in Data Frame which helps us to know in which Borough most complaints are reported.

```
[194]:
```

```
df['Borough'].value_counts().head(6)
```

```
[194]:
```

```
Borough
BROOKLYN      96858
QUEENS        79790
MANHATTAN     62033
BRONX         40217
STATEN ISLAND 12209
Name: count, dtype: int64
```

Figure 18: Borough with top 6 complaints

Here I have used the pie chart to show the complaints frequency in each borough, to get this pie chart I used the method `plot(kind= "pie", autopct= %1.1f%%)`. `Autopct` function shows the percentage in our piechart which helps us to know which Borough has how much percentage and `bbox_to_anchor= ()` helps to give position to the color index box.

```
#the percentage of complaints through borough
plt.figure(figsize= (6,6))
# autopct= '%1.1f%%' is used to show percentage in piechart
df['Borough'].value_counts().head(6).plot(kind='pie', autopct='%1.1f%' )
plt.title('Complaining given by borough')
plt.ylabel('')
plt.legend(bbox_to_anchor=(1, 1)) # This helps to change the position of the color index box (x-axis, y axis) and 1 means highest 1.1c means Lowest
plt.show()
```

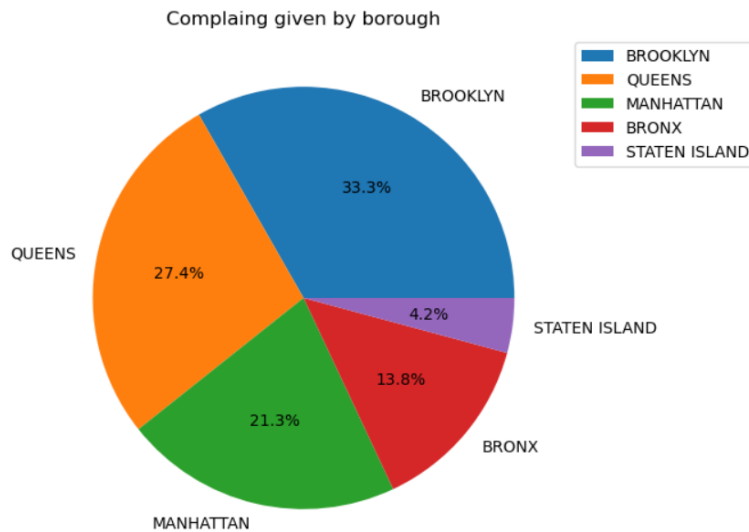


Figure 19: Using pie chart for complain given by borough

This pie chart shows the five boroughs which have the most complaints. It helps in determining the areas which requires the more city services. Here, we can see Brooklyn has the most complaints among the 5 Borough and Manhattan has the least complaints among the 5 Borough.

1.4. Analysing the Request Closing time as per Borough.

1. Using Scatter Plot

Here, to Analyse the trend of "Request closing time" as per borough. I have used here the scatter plot, to generate it at first I have made all the borough values unique and stored it in a variable called unique_borough. I have done this get values of all the borough to run the loop and assign the index of each loop since scatter plot can't recognize string it recognize only the numbers. then I created the empty dictionary called BoroughDic then used the enumerate loop to assign all the Borough index and stored it in the empty dictionary I made. I here assigned the b which is borough the key and i (index) as value

because it takes i as key by default. So, to change this I did that. After I did that I mapped boroughdic with 'Borough dataframe' to assign those ids in the dataframe and gave it x-axis I then gave 'Request closing time' y axis and used plt.scatter(x, y, alpha= 0.5) to make the scatter plot where we used alpha to make the dot transparent.

```
import numpy as np
unique_borough=df["Borough"].unique()

BoroughDic= {}
# enumerate() this function helps to do the loop and will return us the index with the value automatically
# i= index, b=borough, i key by default
for (i,b) in enumerate(unique_borough):
    BoroughDic[b]= i #b key made here. since map matches the df[borough] values with key of
                    #the dictionary and b and df[borough] are same here

x= df['Borough'].map(BoroughDic)# map() matching key with df[] to so assign index(i) value for scatter ploth
y= df["Request Closing Time"]

plt.figure(figsize= (10,6))
plt.scatter(x,y,alpha=0.5) #alpha makes the dot transparent
plt.title("Requist Closing time by Borough")
plt.xlabel("Borough")
plt.ylabel("Request Closing Time")
plt.xticks(ticks= list(BoroughDic.values()), labels= unique_borough, rotation= 90)
plt.show()
```

Figure 20: Using scatter plot code for seeing request closing time as per borough

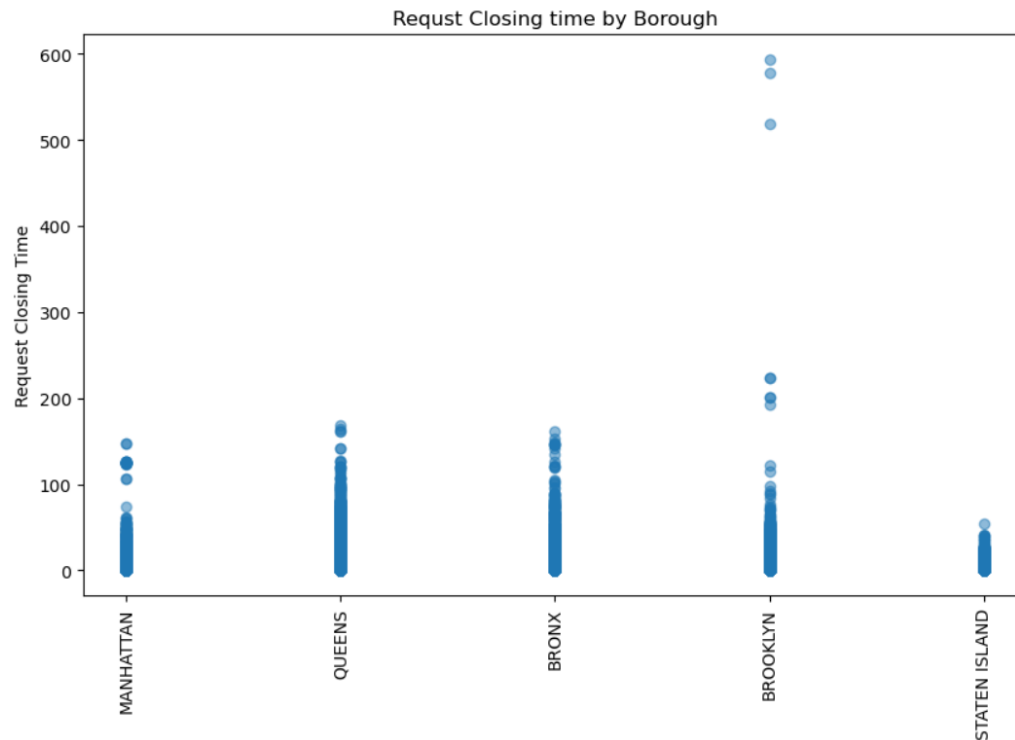


Figure 21: Using scatter plot to see the trends of request closing time as per borough

Here, I have used Scatter plot to Analyse the request closing time as per Borough. And I have founded that. Most of the request closing time in all the Borough is within 200 hours. But we can see that Brooklyn has the most delayed request closing time which is about 600 hours.

2. Using Bar Graph

Here, we have used the Bar graph to analyse the Request Closing Time as per borough. To have bar graph of it, at first I grouped the Borough with Request Closing Time to get Request Closing Time as per Borough. Then took the sum() function to add all the request closing time of particular Borough Then I plotted it in graph.

```
avg_closing_by_borough = df.groupby("Borough")["Request Closing Time"].sum()

plt.figure(figsize=(10, 6))
plt.bar(avg_closing_by_borough.index, avg_closing_by_borough.values, color="pink")

plt.title("Request Closing Time by Borough")
plt.xlabel("Borough")
plt.ylabel("Request Closing Time (hrs)")
plt.xticks(rotation=45)
```

```
([0, 1, 2, 3, 4],
 [Text(0, 0, 'BRONX'),
  Text(1, 0, 'BROOKLYN'),
  Text(2, 0, 'MANHATTAN'),
  Text(3, 0, 'QUEENS'),
  Text(4, 0, 'STATEN ISLAND')])
```

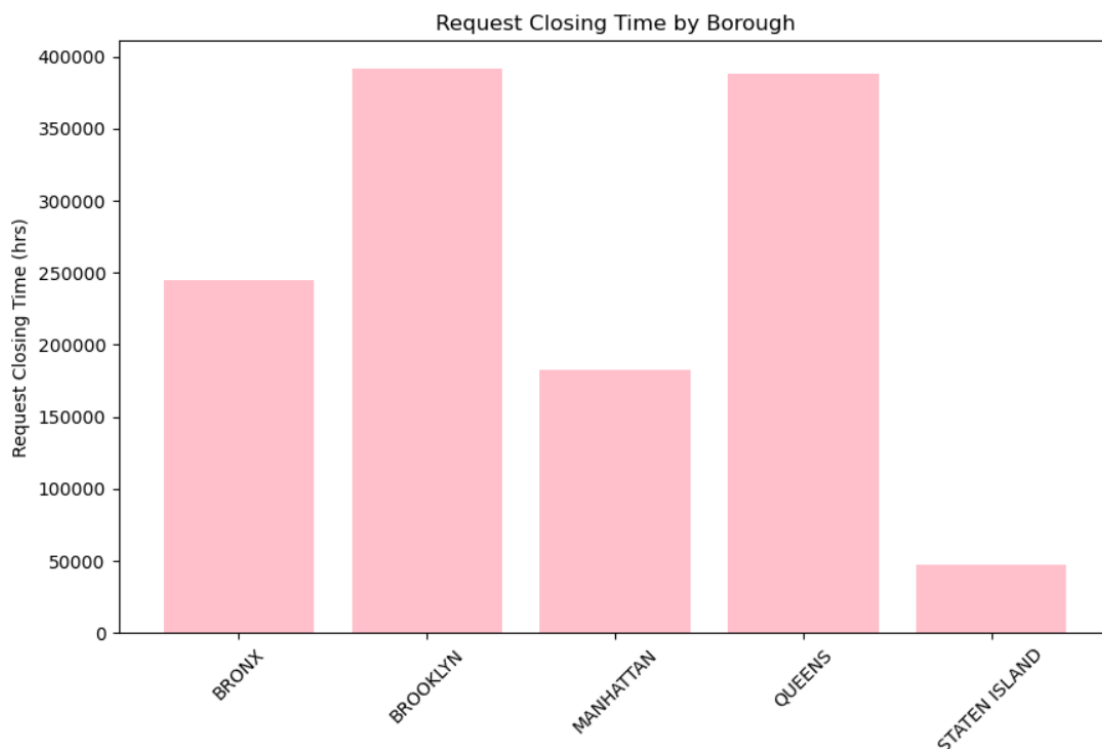


Figure 22: Using bar plot for seeing request closing time as per borough

Here, we can see that the most hour's request service work is done by Brooklyn service since it has the highest bar which is above 350000. The second is queens which is also above 350000. The Borough which has the least service request is Staten Island which is around 5000 hours.

1.5. Average Resolution Time by Complaint and Borough

1. Line plot of the 5 Most Difficult Complaint Types to Resolve

Here, Since we wanted to see trend of slowest request closing time of complaint type on each borough I have presented the mean of "Request Closing Time" and complaint type as per its borough using the pivot table. To get mean for complaint type as per each borough. This has helped us to clearly present and view all the all the mean of Request Closing time and complaint type as per borough in clear and understandable format. I also used here. `.unstack()` method to properly restructure the data that was to be printed, so the data is in table format with rows representing complaint type and column representing the Borough.

[218]:

```
# computing the pivot table
pivot = (
    df
    .groupby(['Borough', 'Complaint Type'])['Request Closing Time']
    .mean()
    .unstack('Borough')
)
print(pivot.head(6))
```

Borough	BRONX	BROOKLYN	MANHATTAN	QUEENS	STATEN ISLAND
Complaint Type					
Animal Abuse	7.336249	4.834937	3.692113	5.416898	4.969150
Blocked Driveway	6.262145	4.410098	3.557656	4.537243	4.070434
Derelect Vehicle	9.211619	5.951346	4.272453	8.491728	5.040692
Disorderly Youth	4.238889	4.149769	2.433333	3.324859	3.894203
Drinking	5.807843	3.540143	3.060317	3.898039	3.493619
Graffiti	8.900000	8.242636	5.065152	6.568018	9.558333

Figure 23: Making pivot table

Here, since we wanted to see trend of slowest request closing time of complaint type on each borough. We stored the top 5 “complaints type” with slowest request closing time using the `df.groupby()` to group the complaint and request closing time for taking out mean, `mean()` to take out mean and `sort_values(5)`. To get top 5 slowest request closing time. Since, we need the top 5 slowest complaint type from here.

```
overall_avg = (
    df.groupby('Complaint Type')['Request Closing Time']
      .mean()
      .sort_values(ascending=False)
      .head(5)
)
print(overall_avg)
```

```
Complaint Type
Derelict Vehicle    7.347370
Graffiti           7.151327
Animal Abuse        5.218511
Blocked Driveway    4.737926
Illegal Parking     4.483444
Name: Request Closing Time, dtype: float64
```

Figure 24: Making overall average variable

Then, I ran the loop on complaint type(index) of overall_average and plotted the mean of the borough which was stored in pivot table only if the complaint type inside pivot table equals the complaint type of overall average. where ever this condition was fulfilled the marker which means dot would come and line graph would come

```
plt.figure()
for complaint in overall_avg.index: # index= complaint and value= mean, taking index only
    plt.plot(pivot.columns, pivot.loc[complaint], marker='o', label=complaint) #plot(pivot.columns, pivot.loc[complaint]): says by loc function plot
                                     #pivot columns(borough) when complaint["complaint type stored in each loop"]

plt.title('Avg Closing Time by Borough for Top 5 Slowest Complaint Types')
plt.xlabel('Borough')
plt.legend()
plt.ylabel('Average Closing Time (hours)')
plt.xticks(rotation=45)
plt.show()
```

Figure 25: Code to see slowest request closing time as per complaints of each borough

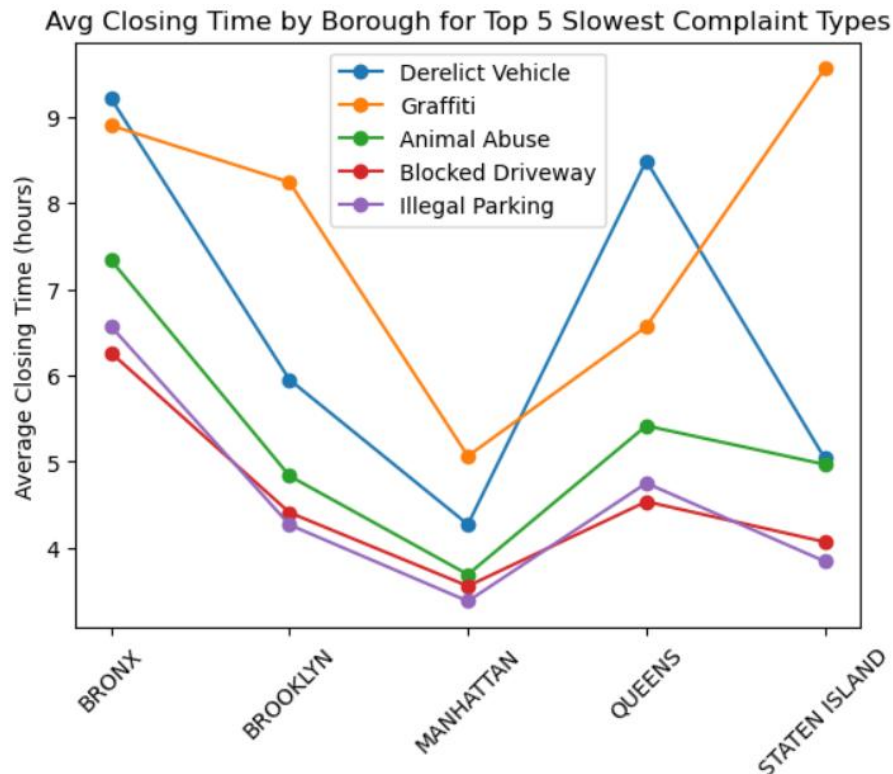


Figure 26: line plot to see slowest request closing time as per complaints of each borough

The line graph shows the five complaints with takes the highest average resolution time and also shows how they perform in each borough. Here we can see that the Graffiti complaint type from Staten Island had the slowest average “Request closing time”.

The complaint type derelict vehicle, animal abuse, illegal parking and Blocked driveway. All had ‘Slowest request closing time’ in Bronx.

2. Bar Plot Graph of the 5 Most Difficult Complaint Types to Resolve

Here, since we wanted to see trend of slowest request closing time of complaint type on each borough, we used the Group bar graph method here at first I collected the top 5 slowest average Request Closing Time as per each Borough with the help of the method `pivot.loc[overall_avg.index]` which stored the average Request Closing Time from pivot table only if the complaint type inside pivot table equals the complaint type of overall average then I gave the `.T.plot()` to show group chart in it and gave `kind='bar'` to say it a bar plot, `figsize=(10, 6)` to give figure size and added `colormap='Set3'` to give suitable colors in each bar.

```
slowest_pivot = pivot.loc[overall_avg.index]

slowest_pivot.T.plot(kind='bar', figsize=(10, 6), colormap='Set3')

plt.title('Avg Closing Time by Borough for Top 5 Slowest Complaint Types')
plt.xlabel('Borough')
plt.ylabel('Average Closing Time (hours)')
plt.xticks(rotation=90)
plt.legend(title='Complaint Type')
plt.show()
```

Figure 27: Group bar Code to see slowest request closing time as per complaints of each borough

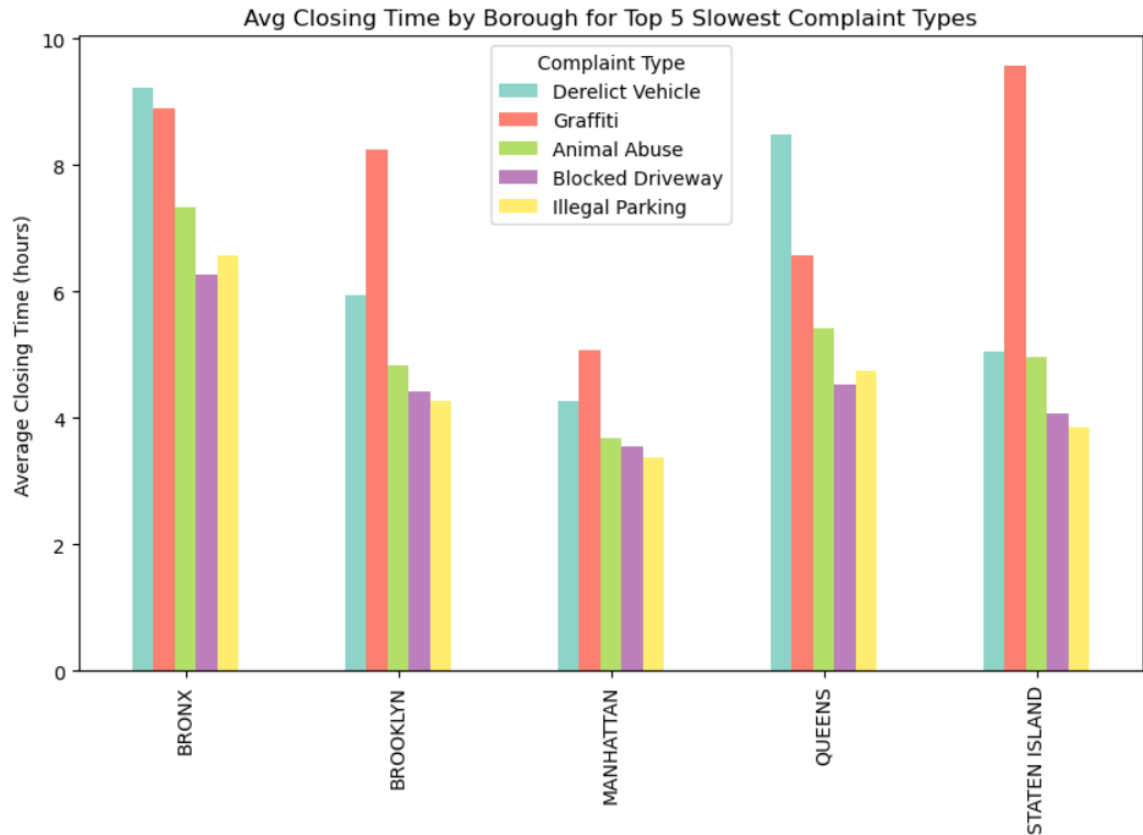


Figure 28: bar chart to see slowest request closing time as per complaints of each borough

The bar graph shows the five complaints with takes the highest average resolution time and also shows how they perform in each borough. Here we can see that the Graffiti complaint type from Staten Island had the slowest average “Request closing time”.

The complaint type derelict vehicle, animal abuse, illegal parking and Blocked driveway. All had ‘Slowest request closing time’ in Bronx.

4. Statistical Testing

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

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

- Null hypothesis (H0): Is the state in which the average closing time of each complaint type has huge difference which concludes that. The Complaint type affects the average closing time.
To be it the $p > 0.05$
- Alternative hypothesis (H1): Is the state in which the average closing time is not that much difference which concludes that. The Complaint type doesn't the average closing time.

1.2. Perform the statistical test and provide the p-value.

```
from scipy.stats import f_oneway

top5 = df['Complaint Type'].value_counts().head(5).index # we need index since value counts stores complaint type(index) and its counts(value)
complaints = df[df['Complaint Type'].isin(top5)] # filters out and keeps only the info regarding top 5

groups = []
for ctype in top5:
    values = complaints.loc[complaints['Complaint Type']==ctype, 'Request Closing Time'].values
    groups.append(values)

af_stat, p_val = f_oneway(*groups)
print("ANOVA p-value:", p_val)
if p_val < 0.05:
    print("The closing time is affected by the Complaint Type")
else:
    print("The closing time is not affected by the Complaint Type")

ANOVA p-value: 0.0
The closing time is affected by the Complaint Type
```

Figure 29 Test 1

1.3. Interpret the results to accept or reject the Null Hypothesis.

- Here, Since the Result is less than 0.05 it is 0.0 it gives the accurate result that request closing time is affected by the Complaint Type

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

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

- Null hypothesis (H0): Is the state in which the average Borough and each complaint type has huge difference which concludes that. The Complaint type is affected the borough. To be it the $p > 0.05$
- Alternative hypothesis (H1): Is the state in which the average Borough and each complaint type has not huge difference which concludes that. The Complaint type is affected the borough.

2.2. Perform the statistical test and provide the p-value.


```

from scipy.stats import chi2_contingency

table = pd.crosstab(df['Complaint Type'], df['Borough'])
print("Contingency Table:")
print(table)

chi2, p_val, dof, expected = chi2_contingency(table)
print("Chi-square p-value:", p_val)
if p_val < 0.05:
    print("The Borough is affected Request closing timee")
else:
    print("The Borough is not affected Request closing timee")

```

Contingency Table:

Borough	BRONX	BROOKLYN	MANHATTAN	QUEENS	STATEN ISLAND
Animal Abuse	1412	2390	1511	1874	557
Blocked Driveway	12740	28119	2055	31621	2141
Derelict Vehicle	1948	5164	530	8102	1762
Disorderly Youth	63	72	68	59	23
Drinking	187	257	294	357	175
Graffiti	9	43	22	37	2
Illegal Parking	7829	27386	11981	21944	4881
Noise - Commercial	2431	11451	14528	6057	677
Noise - House of Worship	79	338	189	297	17
Noise - Park	522	1537	1167	634	67
Noise - Street/Sidewalk	8864	13315	20362	4391	815
Noise - Vehicle	3385	5145	5374	2608	356
Posting Advertisement	16	45	41	30	515
Traffic	355	1082	1531	1302	196
Vending	377	514	2380	477	25

Chi-square p-value: 0.0
The Borough is affected Request closing timee

Figure 30 Test 2

2.3. Interpret the results to accept or reject the Null Hypothesis

- Here, Since the Result is less than 0.05 it is 0.0 it gives the accurate result that complaint type is affected by borough.

5.Conclusion

This Coursework was about Analysing and Filtering the Data set of the Customer Service Request of the New York. Here, we used the Python programming on the Anaconda Notebook of the Jupiter lab which helped us to easily analyse and filter the data by importing the Matplot.lib and the panda's library. The dataset of customer service request contained more than 3 hundred thousand complaints from multiple borough.

Important tasks which includes the date conversion of the columns, figuring the request closing time, deleting the missing values in the data frame and deleting non-important columns, we did all of this while filtering the data. We also did here the statistical testing and also visualized the to compare the request closing time and the complaint type and the borough with each other. I also investigated their distributions which helped us in identifying the trends and variance efficiently. After considering all the things, I would like to conclude by saying that, this analysis helped me by providing me the insightful information about hoe the city services addresses to the public issues. Which will enhance the resource management and the future decision making

References

Kelley, K., 2024. *Simplilearn*. [Online]

Available at: <https://www.simplilearn.com/data-analysis-methods-process-types-article>

Also I took references from the lecture slide provided from the college and contents provided in onedrive file.