



 slington college  
(इसलिंग्टन कलेज)

**Module Code & Module Title**

**CC5061NI Applied Data Science**

**60% Individual Coursework**

**Submission: Final Submission**

**Academic Semester: Spring Semester 2025**

**Credit: 15 credit semester long module**

**Student Name: Pawan Pokhrel**

**London Met ID: 23048667**

**College ID: NP01AI4A230127**

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

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

**Submitted To: Mr. 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.*

## SIMILARITY INDEX REPORT

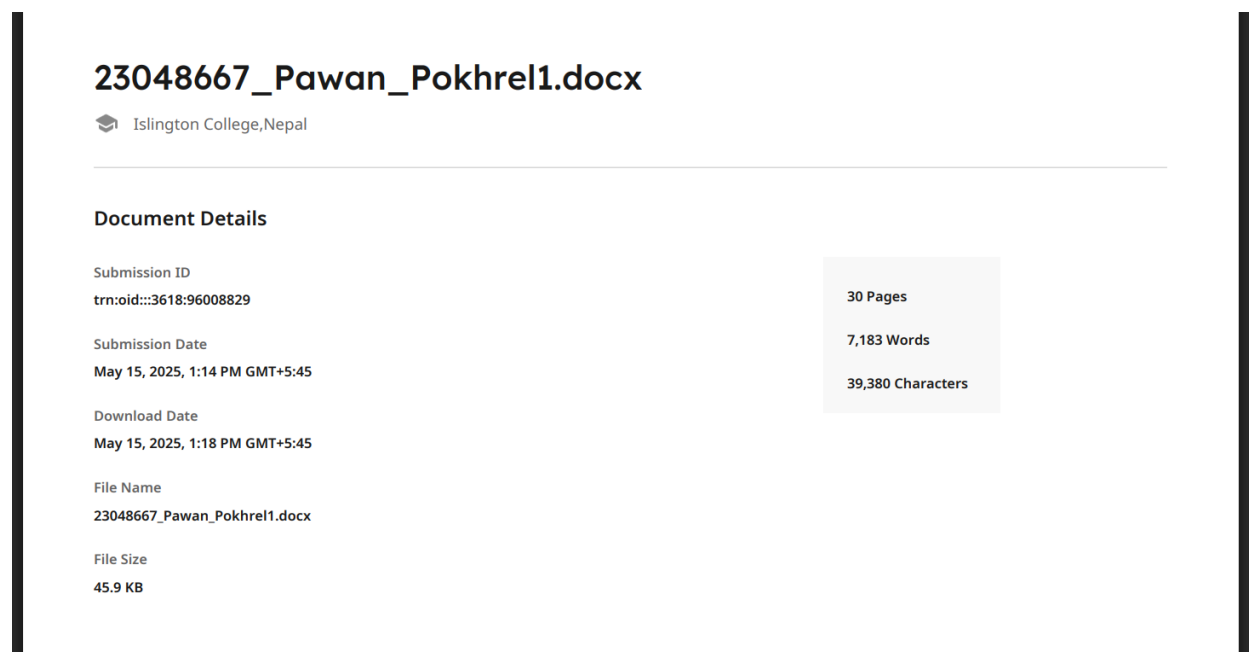


Figure 1: Similarity Report - Page (i)

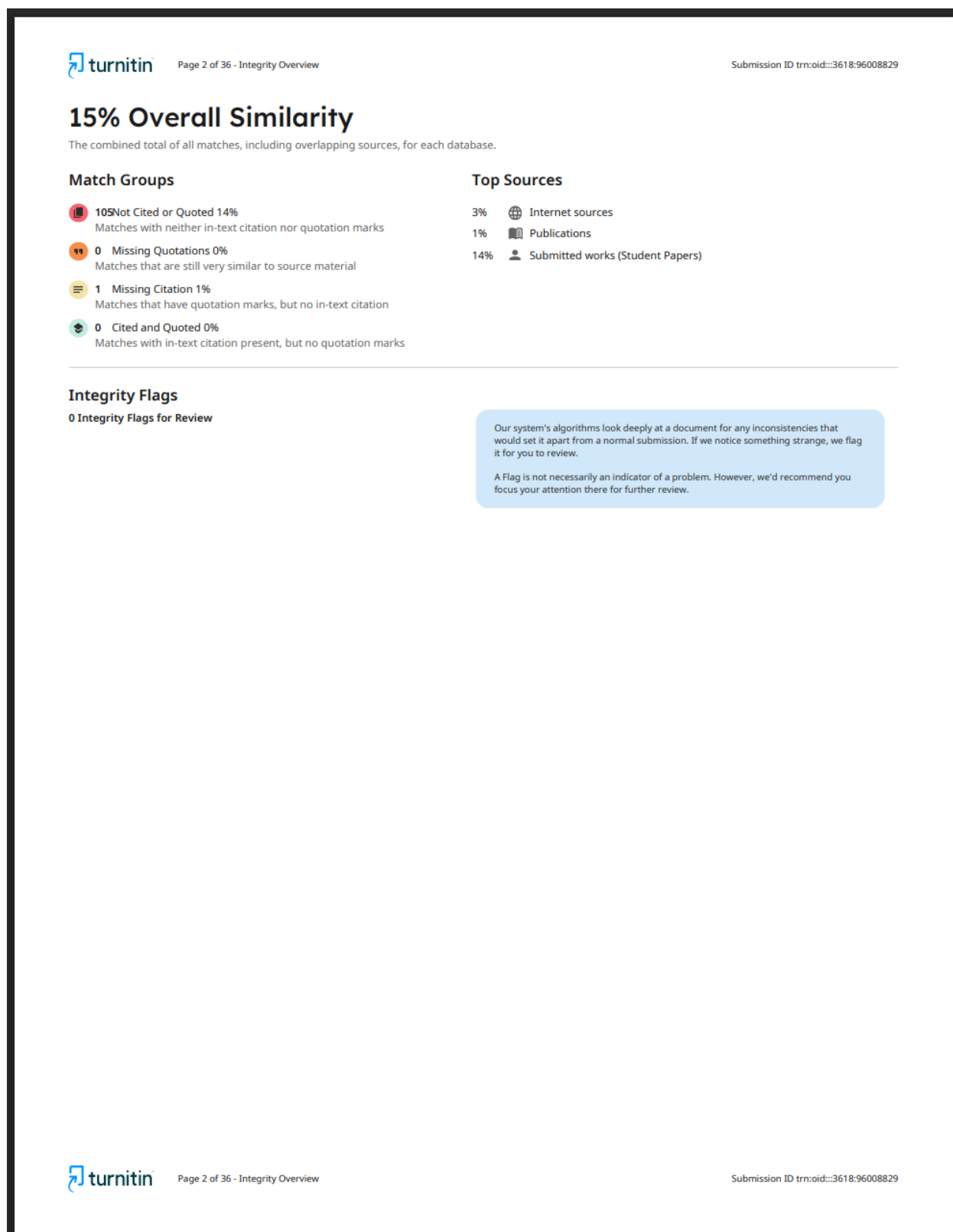


Figure 2: Similarity Report - Page (ii)

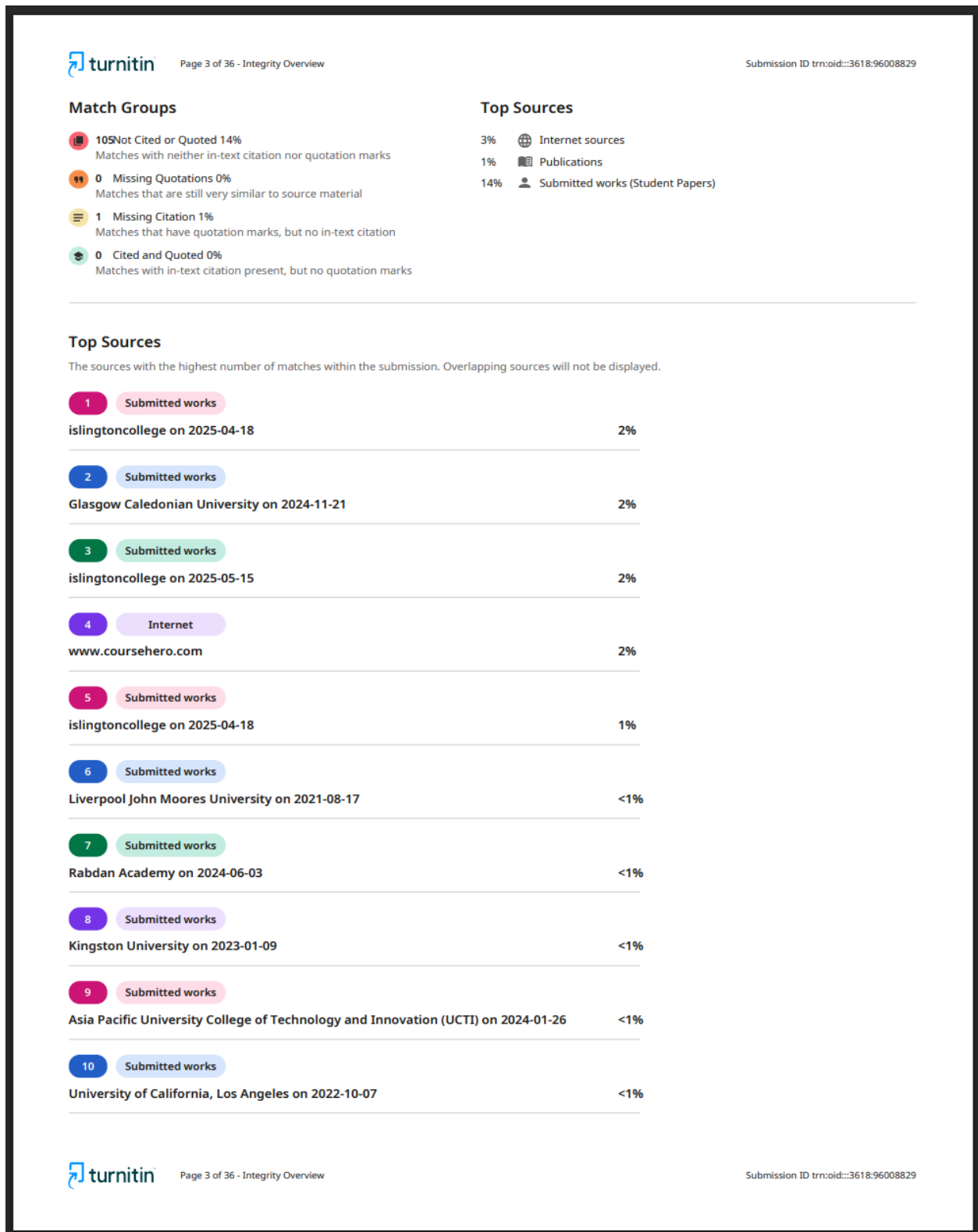


Figure 3: Similarity Report - Page (iii)

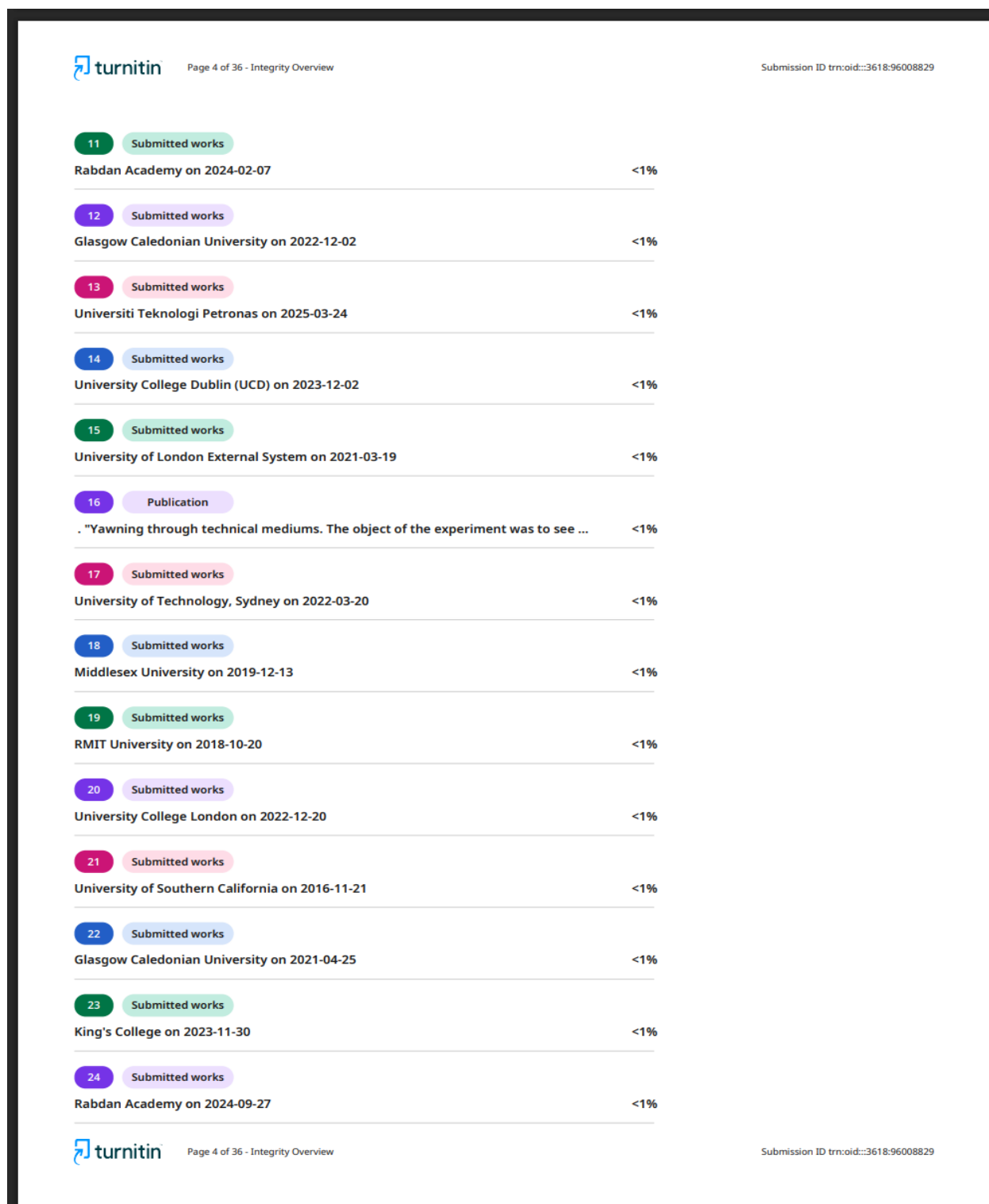


Figure 4: Similarity Report - Page (iv)

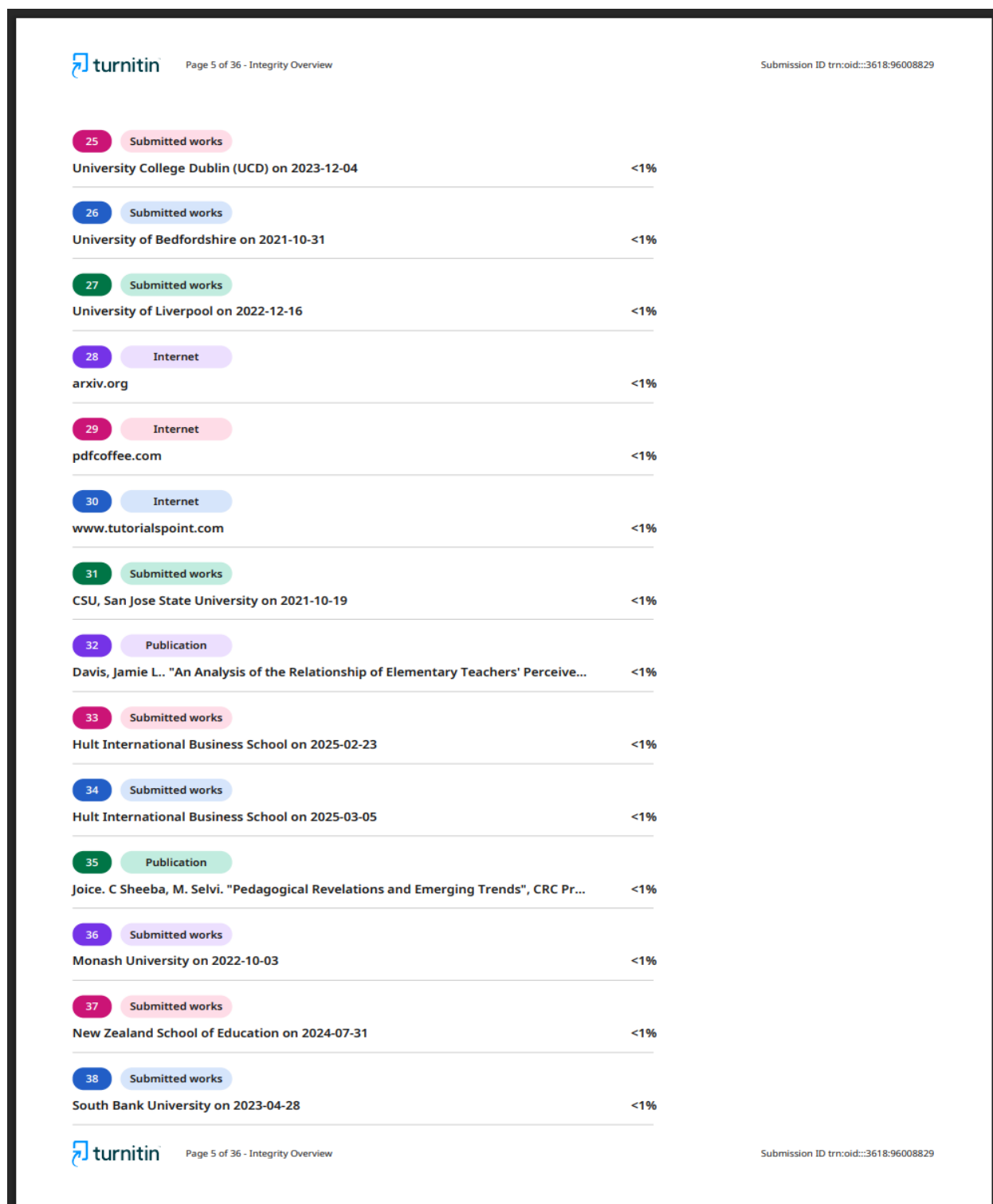


Figure 5: Similarity Report – Page(v)

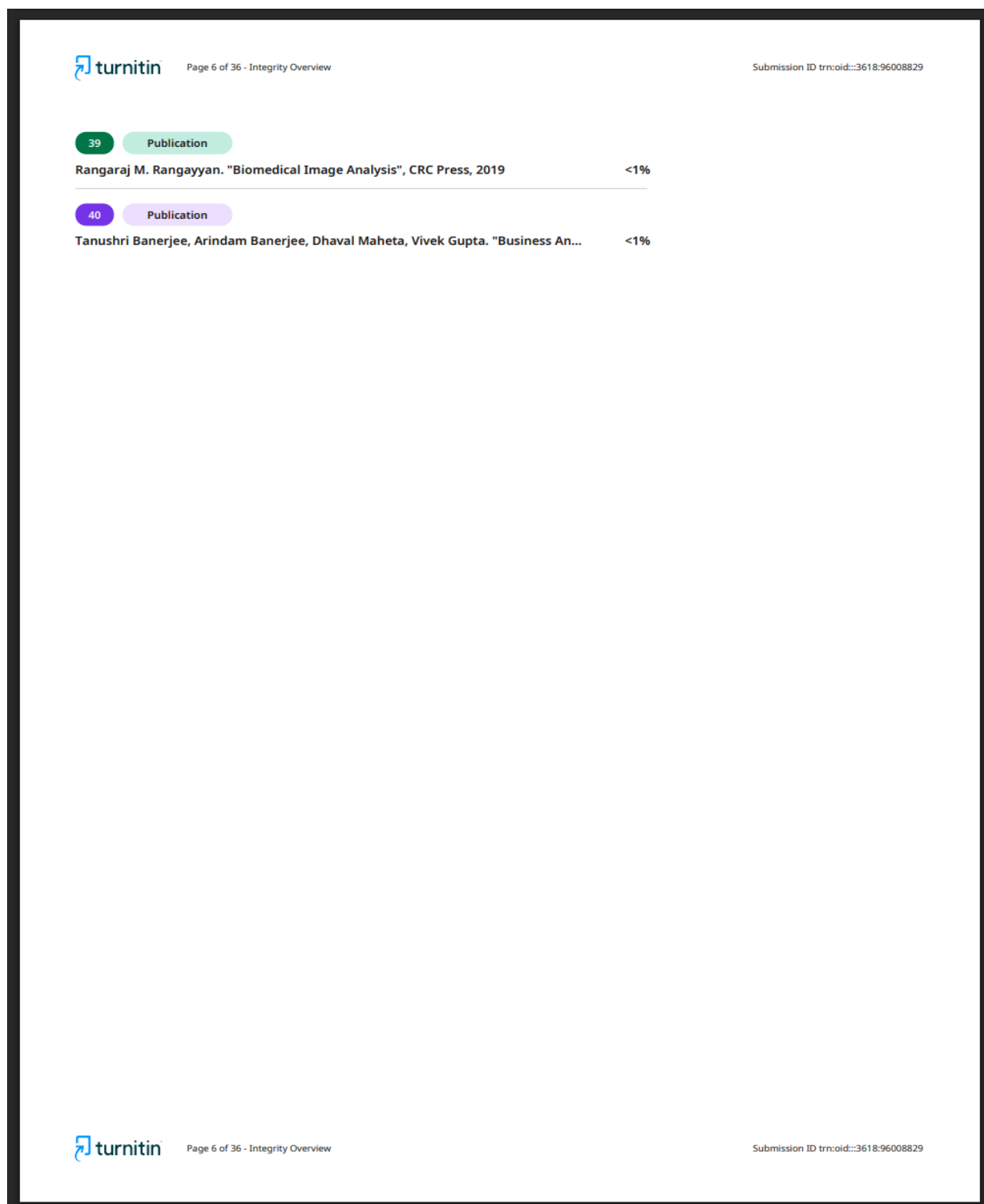


Figure 6: Similarity Report - Page(vi)

## TABLE OF CONTENTS

SIMILARITY INDEX REPORT .....	ii
TABLE OF CONTENTS .....	viii
LIST OF TABLES .....	x
LIST OF FIGURES.....	xi
DATA UNDERSTANDING .....	1
1.1. To understand what your data resources are and the characteristics of those resources. Write down your findings.....	1
DATA PREPARATION .....	6
2.1. Dataset Import .....	6
2.2. Dataset Insights.....	9
2.3. Column datatype conversion.....	12
2.4. Dropping Columns .....	14
2.5. Handling missing / NaN values .....	16
2.6. Unique values from columns.....	17
DATA ANALYSIS .....	19
3.1. Sum, Mean, Standard Deviation, Skewness and Kurtosis .....	19
3.2. Correlation .....	23
DATA EXPLORATION .....	24
4.1. Insights through data visualization .....	24
4.1.1. Most repeated complaint types .....	25
4.1.2. Geographical Concentration of the complaints .....	27
4.1.3. Complaint Trends over the time .....	30
4.1.4. Average Service Response Time .....	34



4.2. Average response time of each complaint types in various locations .....	36
STATISTICAL TESTING .....	42
5.1. Test 1: Similarity of average response time across complaint types .....	42
5.2. Test 2: Whether the complaint types and location types are related .....	44
REFERENCES.....	I

**LIST OF TABLES**

Table 1: Data Understanding .....	1
-----------------------------------	---

## LIST OF FIGURES

Figure 1: Similarity Report - Page (i) .....	ii
Figure 2: Similarity Report - Page (ii) .....	iii
Figure 3: Similarity Report - Page (iii).....	iv
Figure 4: Similarity Report - Page (iv) .....	v
Figure 5: Similarity Report – Page(v) .....	vi
Figure 6: Similarity Report - Page(vi) .....	vii
Figure 7: Screenshot of the dataset importing process through the pandas library with warning.....	6
Figure 8: Screenshot of the dataset importing process without the warning .....	7
Figure 9: Checking if the dataset was imported successfully .....	8
Figure 10: Number of rows and columns in the dataset .....	9
Figure 11: Statistical insights of each column through .describe() .....	10
Figure 12: Number of rows, columns and total individual data in the imported dataset. ....	10
Figure 13: Observing the total number of null values across the columns of dataset....	11
Figure 14: Checking for the duplicate records in the dataset .....	12
Figure 15: Converting the datatype of created_date column and closed_date to datetime from object .....	12
Figure 16: Checking if the datatype was changed to datetime for the date columns.....	13
Figure 17: Creating a new column for response time of each complaint from created and closed date.....	13
Figure 18: Chekcing the number of columns before removing the irrelevant columns ..	15
Figure 19: Dropping the irrelevant columns using .drop() function .....	15
Figure 20: Checking for the possible null values in the columns .....	16
Figure 21: Removing the null values using .dropna() function.....	17
Figure 22: Checking for the possible null values after removing the null values .....	17
Figure 23: Displaying unique values from each column of the dataset .....	18
Figure 24: Creating a new dataframe from the updated dataset with only numeric values .....	19
Figure 25: Changing the value in 'Request_Closing_Time' to integral value in hours ...	20

Figure 26: Calculating sum of each numerical columns of dataset individual.....	20
Figure 27: Calculating the mean value of each numerical columns of the dataset individually.....	21
Figure 28: Calculating the standard deviation of each numerical columns of the dataset individually.....	21
Figure 29: Calculating the value of skewness of each numerical columns of the dataset individually.....	22
Figure 30: Calculating kurtosis of the dataset through .kurt() .....	22
Figure 31: Collectively displaying sum, mean, standard deviation, skewness and kurtosis in a table.....	22
Figure 32: Displaying the correlation between each of the variables .....	23
Figure 33: importing pyplot library from matplotlib.....	24
Figure 34: process involved in bar chart construction .....	26
Figure 35: Insight 1: Most frequent complaint types .....	26
Figure 36: process involved in creating a pie chart for complaints received in each borough.....	28
Figure 37: Insight 2: Complaints in individual borough .....	28
Figure 38: Insight 2i: pie chart with enhanced borough with highest no. of complaints received.....	29
Figure 39: Insight 2ii: pie chart with enhanced borough with least no. of complaints received.....	30
Figure 40: process involved in creating a line chart for complaint trends over months of 2015 .....	32
Figure 41: Insight 3a: Complaint trends in 2015 over months .....	32
Figure 42: process involved in creating a line chart for complaint trends over months of 2015 .....	33
Figure 43: Insight 3b: Complaint trends in 2015 over weeks.....	34
Figure 44: process involved in creating a horizontal bar chart for average response time of each complaint types.....	35
Figure 45: Insight 4: Average response time of each complaint types .....	36

Figure 46: process involved in visualizing average response time by complaint types and borough .....	37
Figure 47: Average response time of each complaint types grouped by borough .....	38
Figure 48: process involved in visualizing average response time by borough and complaint types .....	39
Figure 49: Average response time of each borough grouped by complaint types .....	39
Figure 50: process involved in visualizing average response time by complaint types and location types .....	40
Figure 51: heat map of average response time between complaint types and location types.....	41
Figure 52: Statistical test 1, one way - ANOVA test, for similar mean response time across complaint types .....	43
Figure 53: Statistical Test 2, chi-square contingency test, relation between complaint types and location types.....	45

## DATA UNDERSTANDING

### 1.1. To understand what your data resources are and the characteristics of those resources. Write down your findings

The provided dataset carries detailed records of 311 Customer Service Requests filed in New York City from year 2010 to the present. Each record of the dataset represents an individual customer service request and includes various attributes such as the type of complaint/requests the residents have made, detailed location details, timestamps of the request creation and closure date, responsible agency which is handling the customer service request, resolution status of the request and geospatial co-ordinates. With total of 53 columns and 300698 rows, the dataset consists of combination of categorical, textual, numerical and datetime data types. The structure of this dataset enables in comprehensive understanding of urban helpline service demands, common issues reported by the residents, common issues arising from a certain locality and responsiveness of the agencies. The data is well-suited for geographic and categorical analysis to identify the trends, patterns and potential areas for improvement in public service delivery.

*Table 1: Data Understanding*

S. No	Column Name	Description	Data Type
1	Unique Key	This is the primary key of the dataset which provides a unique ID to each 311-customer service request.	int64
2	Created Date	This column contains the date and time when the service request was issued.	Object
3	Closed Date	This column shows when the request was marked as closed.	Object

4	Agency	This column gives an abbreviation representing the agency that handled the request.	Object
5	Agency Name	This column gives the full name of the agency responsible for resolving the complaint.	Object
6	Complaint Type	This column provides information on the general nature of the complaint filed by the residents of the city.	Object
7	Descriptor	This column gives more specific details about the issue within the complaint type.	Object
8	Location Type	This column shows the kind of place where the incident occurred (e.g., street, park, etc.).	Object
9	Incident Zip	This column contains the ZIP code of the area where the incident took place.	int64
10	Incident Address	This column gives the street address where the complaint was reported.	Object
11	Street Name	This column shows the name of the street involved in the incident.	Object
12	Cross Street 1	This column shows the name of the first intersecting street near the complaint location.	Object
13	Cross Street 2	This column shows the second intersecting street near the complaint location.	Object
14	Intersection Street 1	This column provides the name of one of the streets forming an intersection where the issue occurred.	Object
15	Intersection Street 2	This column shows the other street in the intersection related to the complaint.	Object
16	Address Type	This column indicates whether the location is an address, intersection, or other type.	Object

17	City	This column gives information on which city the complaint was reported from.	Object
18	Landmark	This column provides information about nearby landmarks.	Object
19	Facility Type	This column shows what kind of facility is involved like residential, commercial, etc.	Object
20	Status	This column shows whether the service request is still open, closed, or unresolved.	Object
21	Due Date	This column gives the deadline by which the issue was expected to be resolved.	Object
22	Resolution Description	This column provides a brief explanation of how the issue was handled or resolved.	Object
23	Resolution Action Updated Date	This column shows the most recent update regarding the action taken to resolve the issue.	Object
24	Community Board	This column indicates the community board that governs the area from where the complaint was made.	Object
25	Borough	This column provides the borough in which the issue occurred (e.g., Bronx, Brooklyn, etc.).	Object
26	X Coordinate (State Plane)	This column provides the X-coordinates of incident's location for geospatial mapping in the State Plane system.	float64
27	Y Coordinate (State Plane)	This column provides the Y-coordinates of incident's location for geospatial mapping in the State Plane system.	float64
28	Park Facility Name	This column gives the name of the park facility associated with the complaint.	Object



29	Park Borough	This column shows which borough the park facility is located in.	Object
30	School Name	This column provides the name of the school involved in the complaint.	Object
31	School Number	This column contains the assigned school number for identification purposes.	Object
32	School Region	This column shows the administrative region the school belongs to.	Object
33	School Code	This column contains a unique code used to identify the school.	Object
34	School Phone Number	This column gives the contact number of the school involved in the report.	Object
35	School Address	This column gives the address of the school, if connected to the complaint.	Object
36	School City	This column indicates the city in which the related school is located.	Object
37	School State	This column gives the state in which the school is located.	Object
38	School Zip	This column contains the postal code of the school in question.	int64
39	School Not Found	This column flags whether the referenced school was not found in the system.	Object
40	School or Citywide Complaint	This column identifies whether the complaint is about a specific school or a citywide issue.	Object
41	Vehicle Type	This column provides details on the type of vehicle involved in the complaint.	Object
42	Taxi Company Borough	This column indicates the borough the taxi company is based in or operates within.	Object

43	Taxi Pick Up Location	This column shows the area where the taxi picked up the passenger related to the complaint.	Object
44	Bridge Highway Name	This column gives the name of the bridge or highway connected to the report.	Object
45	Bridge Highway Direction	This column shows the direction of travel on the bridge or highway where the issue occurred.	Object
46	Road Ramp	This column gives information about the road ramp involved in the complaint.	Object
47	Bridge Highway Segment	This column identifies a specific segment of the bridge or highway mentioned in the request.	Object
48	Garage Lot Name	This column gives the name of the garage or parking facility associated with the issue.	Object
49	Ferry Direction	This column indicates the direction of travel for a ferry, if part of the complaint.	Object
50	Ferry Terminal Name	This column gives the name of the ferry terminal related to the complaint.	Object
51	Latitude	This column shows the latitude coordinate of the reported incident location.	float64
52	Longitude	This column shows the longitude coordinate of the reported incident location.	float64
53	Location	This column provides a tuple (latitude, longitude) representing the exact location of the complaint.	Object

## DATA PREPARATION

### 2.1. Dataset Import

#### Q. Import the Dataset.

In the starting of Data Preparation, the dataset was successfully imported to the jupyter notebook using Pandas library. This dataset contains information about the 311 Customer Service Requests logged from the year 2010 to the present. Each record represents an individual request made by the residents to the city's service helpline along with various attributes like complaint type, location info, agency involved and resolution status.

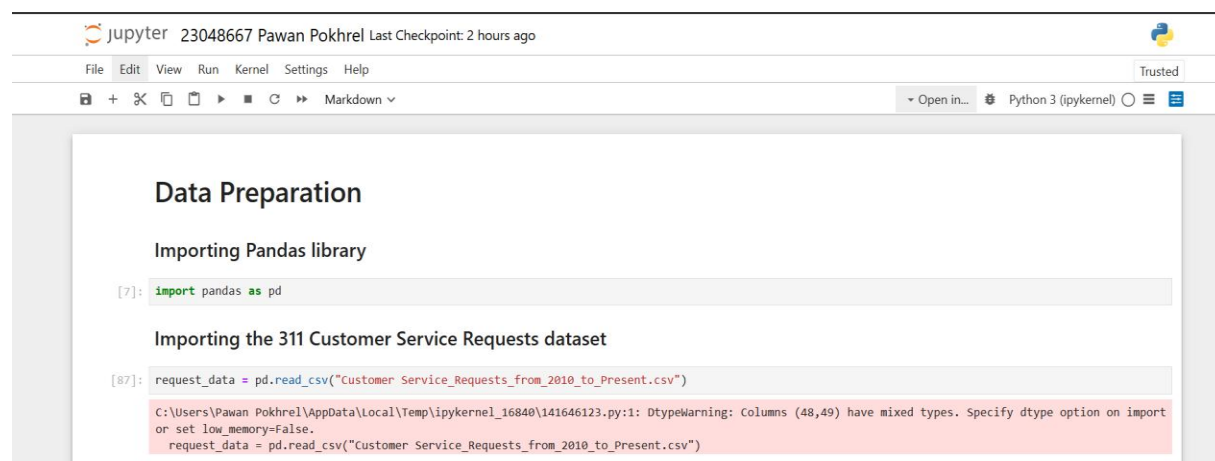


Figure 7: Screenshot of the dataset importing process through the pandas library with warning

The above screenshot shows the process involved while importing the dataset. The csv file containing the dataset was imported to the jupyter notebook file using `.read_csv()` function from the Pandas library. The displayed warning is due to the presence of multiple datatypes in a single column. The warning can be removed by setting `low_memory` to false which tells the Pandas library to process the importing of dataset in one go, ignoring the mixed-datatype warnings.

jupyter 23048667 Pawan Pokhrel Last Checkpoint: 9 days ago

File Edit View Run Kernel Settings Help Trusted

Open in... Python 3 (ipykernel)

## Data Preparation

### Importing Pandas library

```
[3]: import pandas as pd
```

### Importing the 311 Customer Service Requests dataset

```
[*]: request_data = pd.read_csv("Customer_Service_Requests_from_2010_to_Present.csv", low_memory = False)
```

### Checking if the dataset is imported correctly

```
[7]: request_data
```

```
[7]:
```

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...
300693	30281872	03/29/2015 12:33:41 AM	NaN	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	NaN	CRESCENT AVENUE	NaN	NaN	NaN
300694	30281230	03/29/2015 12:33:28 AM	03/29/2015 02:33:59 AM	NYPD	New York City Police Department	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	100-17 87 AVENUE	NaN	NaN	NaN
300695	30283424	03/29/2015 12:33:03 AM	03/29/2015 03:40:20 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	162 THROOP AVENUE	NaN	NaN	NaN
		03/29/2015	03/29/2015		New York	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant		3151 EAST			

Figure 8: Screenshot of the dataset importing process without the warning

The following screenshot displays the imported dataset in the jupyter environment which was done to check if the dataset was correctly imported or not.

Jupyter 23048667 Pawan Pokhrel Last Checkpoint: 2 hours ago

File Edit View Run Kernel Settings Help

Python 3 (ipykernel)

### Checking if the dataset is imported correctly

```
[89]: request_data
```

```
[89]:
```

	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
300693	30281872	03/29/2015 12:33:41 AM	NaN	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	NaN	CRESCENT AVENUE	...	NaN	NaN	NaN
300694	30281230	03/29/2015 12:33:28 AM	03/29/2015 02:33:59 AM	NYPD	New York City Police Department	Blocked Driveway	Partial Access	Street/Sidewalk	11418.0	100-17 87 AVENUE	...	NaN	NaN	NaN
300695	30283424	03/29/2015 12:33:03 AM	03/29/2015 03:40:20 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	11206.0	162 THROOP AVENUE	...	NaN	NaN	NaN
300696	30280004	03/29/2015 12:33:02 AM	03/29/2015 04:38:35 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Bar/Restaurant	10461.0	3151 EAST TREMONT AVENUE	...	NaN	NaN	NaN
300697	30281825	03/29/2015 12:33:01 AM	03/29/2015 04:41:50 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Store/Commercial	10036.0	251 WEST 48 STREET	...	NaN	NaN	NaN

300698 rows x 53 columns

Figure 9: Checking if the dataset was imported successfully

Hence, the dataset of the 311 Customer Service Requests was successfully imported to the jupyter notebook file in request\_data variable as a dataframe.

## 2.2. Dataset Insights

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

The provided dataset, which contains 53 columns and 300698 rows, includes valuable information about the citizen service requests and public service issues within a city. The number of columns and rows can be retrieved from the data frame using `.shape()` function to the data frame. The dataset provides timestamps of the requests being created and closed, type of complaints raised, the agency handling the request and location specific details. From the dataset, we can gain insight of the public service efficiency, common issues faced by the city residents and patterns over time and locations.

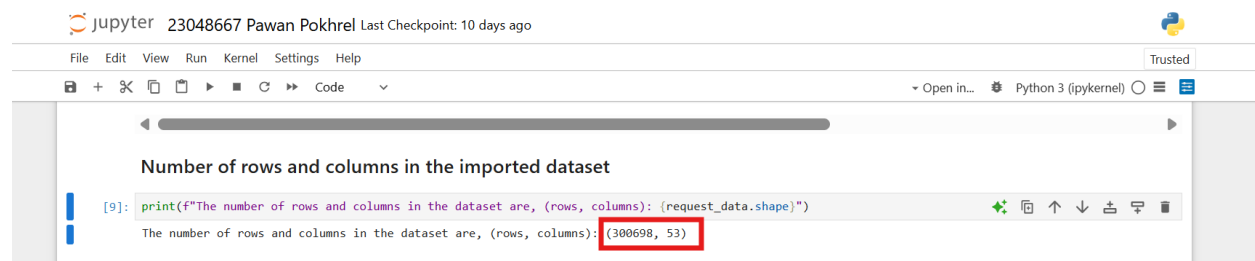


Figure 10: Number of rows and columns in the dataset

The following screenshot shows some of the basic statistical insights from the dataset retrieved from the `.describe()` function for the data frames. This function calculates and displays the number of values in each column, average data of each column, standard deviation, 25<sup>th</sup> percentile, 50<sup>th</sup> percentile, 75<sup>th</sup> percentile and maximum value for each numerical column of the data frame.

Statistical Insights the provided dataset carries

```
[91]: request_data.describe()
```

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

Figure 11: Statistical insights of each column through .describe()

The below screenshot shows the process involved in retrieving the total number of rows, columns and total number of individual data in the imported dataset. This was achieved through the use of .shape and .size attributes of the data frame. The .shape returns a tuple of number of rows and columns as (rows, columns) and the .size returns the number of individual data elements by multiplying rows and columns, i.e. rows \* columns.

Major insights from the dataset

Dataset shape and size

```
[15]: print(f"The total records in the imported dataset: {request_data.shape[0]}")
print(f"The total attributes/columns in the imported dataset: {request_data.shape[1]}")
print(f"The total number of the individual data in the dataset: {request_data.size}")
```

The total records in the imported dataset: 300698  
The total attributes/columns in the imported dataset: 53  
The total number of the individual data in the dataset: 15936994

Figure 12: Number of rows, columns and total individual data in the imported dataset

The next screenshot displays the number of possible numbers of null values in each of the columns using .isnull() which returns a data frame with Boolean values if the data cell is null or not. The received data frame with the Boolean values is then summed up for total number of null values in each columns using .sum() as shown in the screenshot. Ultimately, the .isnull().sum() returns the total number of null values in each columns of the data set.

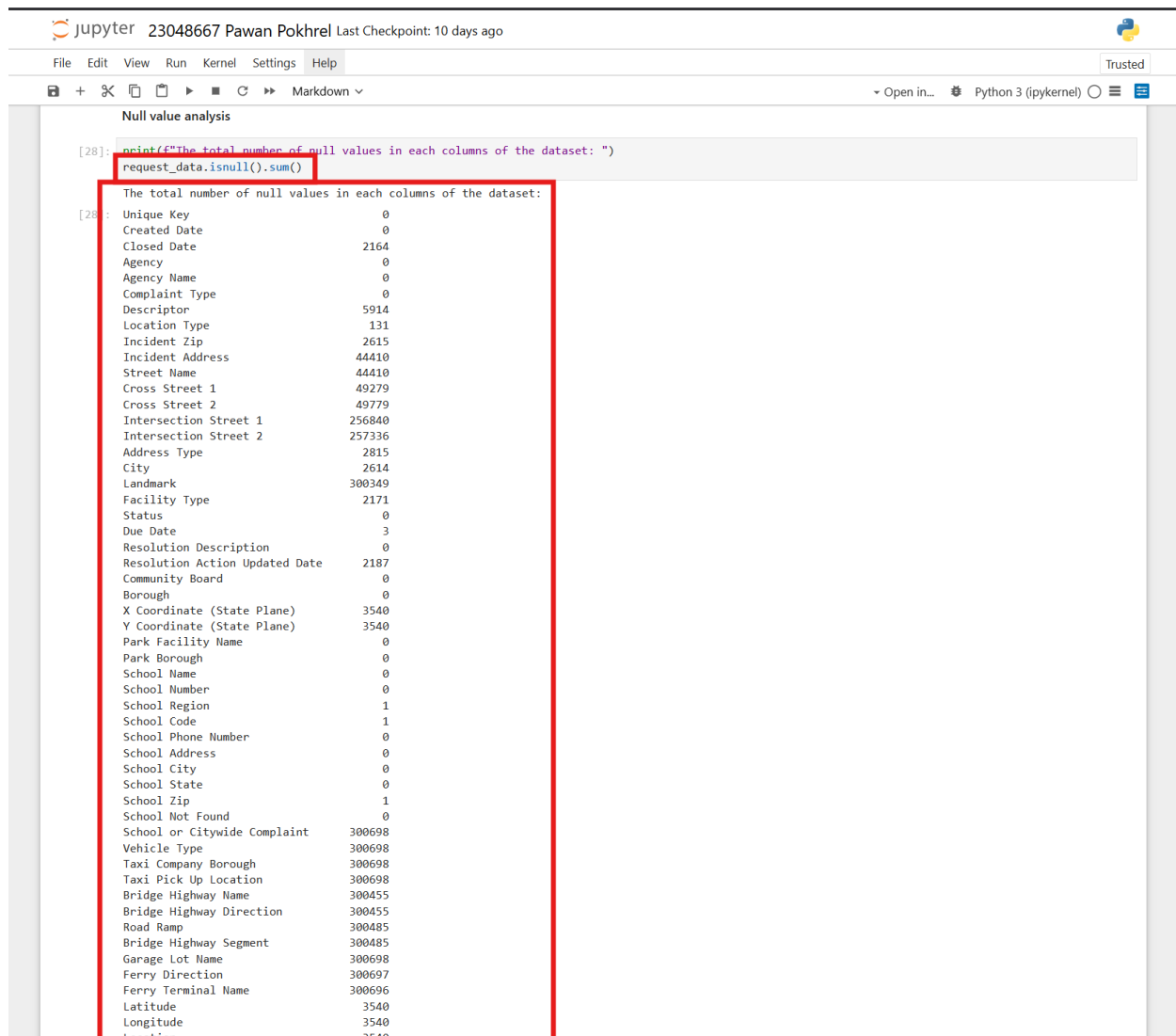
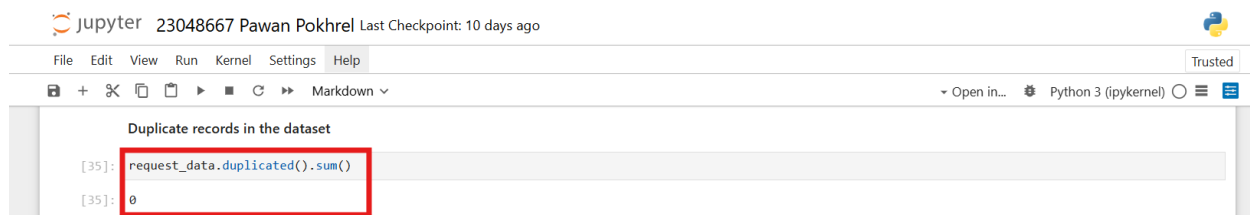


Figure 13: Observing the total number of null values across the columns of dataset

The presence of duplicated records was checked by using the `.duplicated().sum()` functions. The `.duplicated()` returns a data dictionary with a Boolean value for each record if the record is duplicated or not. The `.sum()` adds all the Boolean values and returns a number if there is a presence of duplicated records. In the below screenshot, the duplicated records is found to be 0 in the 311 customer service requests dataset.





```
request_data.duplicated().sum()
```

```
0
```

Figure 14: Checking for the duplicate records in the dataset

## 2.3. Column datatype conversion

Q. 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 datatype of 'Created Date' and 'Closed Date' columns of the 311 customer service request dataset was changed to datetime from object using the `.to_datetime()` function of the Pandas library. The `errors = 'coerce'` parameter was applied to the function in a bid to ensure that any irregular entries when it comes to the date entry gets set at NaT (Not a Time). It prevents the unexpected behaviours of the data frame, errors and inconsistent data in the dataset. The datatype conversion was performed to enable easier date-based calculations such as determining the response times of the complaints by calculating the time difference between the 'Created Date' and 'Closed Date' of the complaint in the 311 customer service requests dataset.

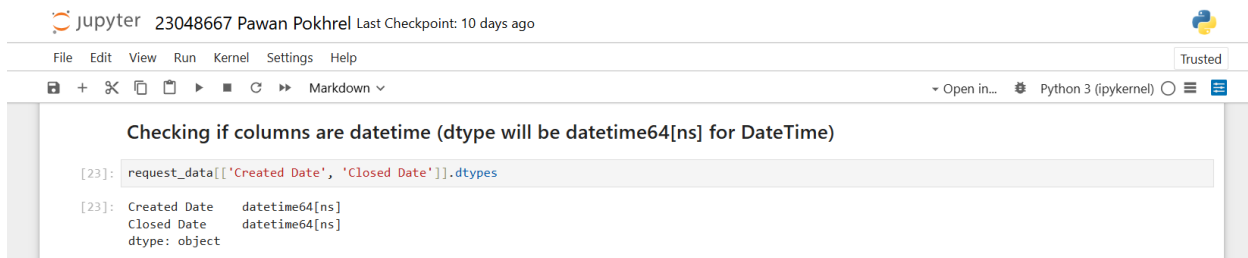


```
request_data[['Created Date', 'Closed Date']].dtypes
```

Figure 15: Converting the datatype of created\_date column and closed\_date to datetime from object

The 'Created Date' and 'Closed Date' columns were checked for their types using the `request_data[['Created Date', 'Closed Date']].dtypes` command. The result showed that

both columns were successfully converted to the `datetime64[ns]` type, which confirmed the success of the earlier step of conversion. This confirmation was done to ensure that the data were in the right format for the following analyses, e.g., calculating time differences.



The screenshot shows a Jupyter Notebook interface with the title '23048667 Pawan Pokhrel Last Checkpoint: 10 days ago'. The menu bar includes File, Edit, View, Run, Kernel, Settings, and Help. The toolbar shows various icons for file operations and execution. The code cell contains the following text:

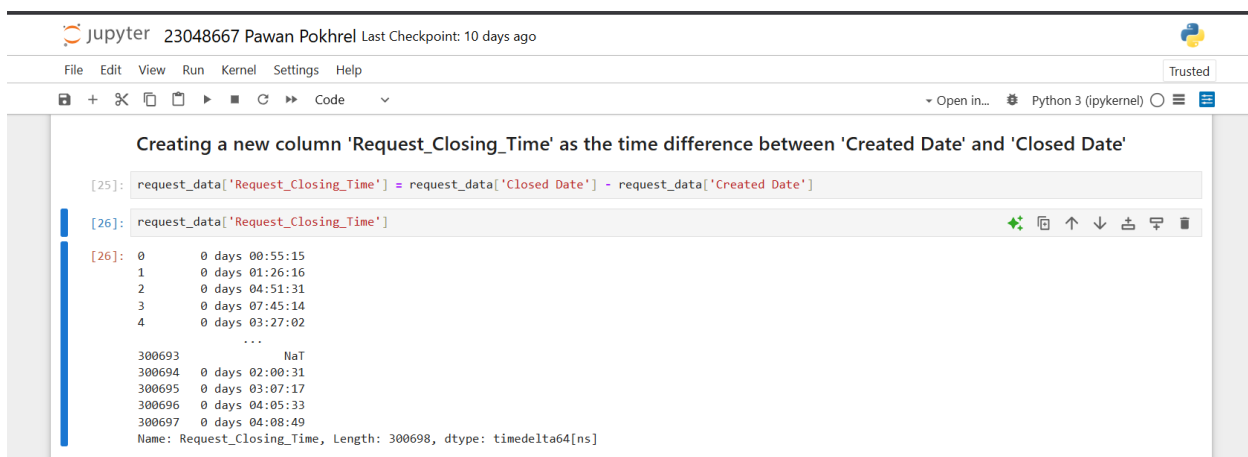
```
Checking if columns are datetime (dtype will be datetime64[ns] for DateTime)
```

```
[23]: request_data[['Created Date', 'Closed Date']].dtypes
```

```
[23]: Created Date    datetime64[ns]
      Closed Date    datetime64[ns]
      dtype: object
```

Figure 16: Checking if the datatype was changed to datetime for the date columns

A new column named 'Request\_Closing\_Time' was created by subtracting the 'Created Date' column from the 'Closed Date' column using the operation `request_data['Closed Date'] - request_data['Created Date']`. The resulting time differences were stored in this new column, with a sample of the data showing durations like 0 days 00:56:15 and 0 days 04:03:29. The data type of this column was identified as `timedelta64[ns]`, representing time durations, and the dataset was found to contain 300,698 rows. This step was carried out to facilitate the analysis of how long it took to resolve customer service requests.



The screenshot shows a Jupyter Notebook interface with the title '23048667 Pawan Pokhrel Last Checkpoint: 10 days ago'. The menu bar includes File, Edit, View, Run, Kernel, Settings, and Help. The toolbar shows various icons for file operations and execution. The code cell contains the following text:

```
Creating a new column 'Request_Closing_Time' as the time difference between 'Created Date' and 'Closed Date'
```

```
[25]: request_data['Request_Closing_Time'] = request_data['Closed Date'] - request_data['Created Date']
```

```
[26]: request_data['Request_Closing_Time']
```

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

Figure 17: Creating a new column for response time of each complaint from created and closed date

## 2.4. Dropping Columns

Q. 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']
```

The unnecessary or irrelevant data columns were removed to simplify the data analysis process. The above listed columns which were mostly related to detailed location or school data was dropped from the dataset as these columns do not contribute meaningfully to the analysis of complaint patterns or resolution efficiency. Hence, the columns were excluded from dataset to streamline the analysis operations.

In the first screenshot, all the columns were listed using the `.columns.to_list()` functions in the data frame. The `.columns` attribute provides all the columns of the data frame in an array and the `.to_list()` function converts the array to list for better readability. The number of columns before dropping the irrelevant columns was 54 which was found using the `request_data.shape[1]` command which returns the number of columns from the data shape of `request_data`.

Dropping the irrelevant table columns from the dataset (inplace = True for permanently changing in the dataset)

Columns before dropping the irrelevant columns

```
[19]: print(request_data.columns.tolist())
```

```
[19]: ['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']
```

```
[37]: print(f"The number of columns before dropping the irrelevant columns: {request_data.shape[1]}")
```

The number of columns before dropping the irrelevant columns: 53

Figure 18: Checking the number of columns before removing the irrelevant columns

In the following screenshot, the irrelevant columns from the dataset were listed in a variable `irrelevant_table_columns`. These columns were dropped from the dataset using `.drop()` function in the data frame by passing columns parameter and `inplace = True` to ensure the columns are dropped permanently from the data frame itself. If the `inplace = True` is not used it displays the columns without the dropped columns but the data frame would be intact without the drop action happening in it.

Dropping the irrelevant columns

```
[41]: irrelevant_table_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']
```

```
request_data.drop(columns = irrelevant_table_columns, inplace = True)
```

Figure 19: Dropping the irrelevant columns using `.drop()` function

The remaining columns were listed after dropping the irrelevant columns using the same `.columns.to_list()` function and the number of columns was found to be 14 stating that the drop action was successful.

After dropping the irrelevant columns

```
[43]: print(request_data.columns.tolist())
```

```
[43]: ['Unique Key', 'Created Date', 'Closed Date', 'Agency', 'Complaint Type', 'Descriptor', 'Location Type', 'Incident Zip', 'City', 'Status', 'Resolution Description', 'Borough', 'Latitude', 'Longitude']
```

```
[45]: print(f"The number of columns after dropping the irrelevant columns: {request_data.shape[1]}")
```

The number of columns after dropping the irrelevant columns: 14

## 2.5. Handling missing / NaN values

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

The rows which contained missing values were eliminated from the dataset to maintain the integrity of our analysis. The removal of missing (NaN) values ensured that the data visualizations or analysis are based on a complete, reliable and consistent data.

In the first screenshot, the columns with missing values were checked using the `.isnull().sum()` functions in the data frame. The `.isnull()` method identifies missing values in each column by returning a Boolean value (True for missing, False otherwise), and the `.sum()` function totals the number of missing values per column. In the screenshot, the number of missing values was found to be 2164 for 'Created Date' and 'Closed Date', 5914 for 'Complaint Type', 113 for 'Location Type', 2615 for 'Incident Zip', 2014 for 'City', 3540 for 'Latitude', 3540 for 'Longitude', and 2164 for 'Request\_Closing\_Time', with other columns having 0 missing values.

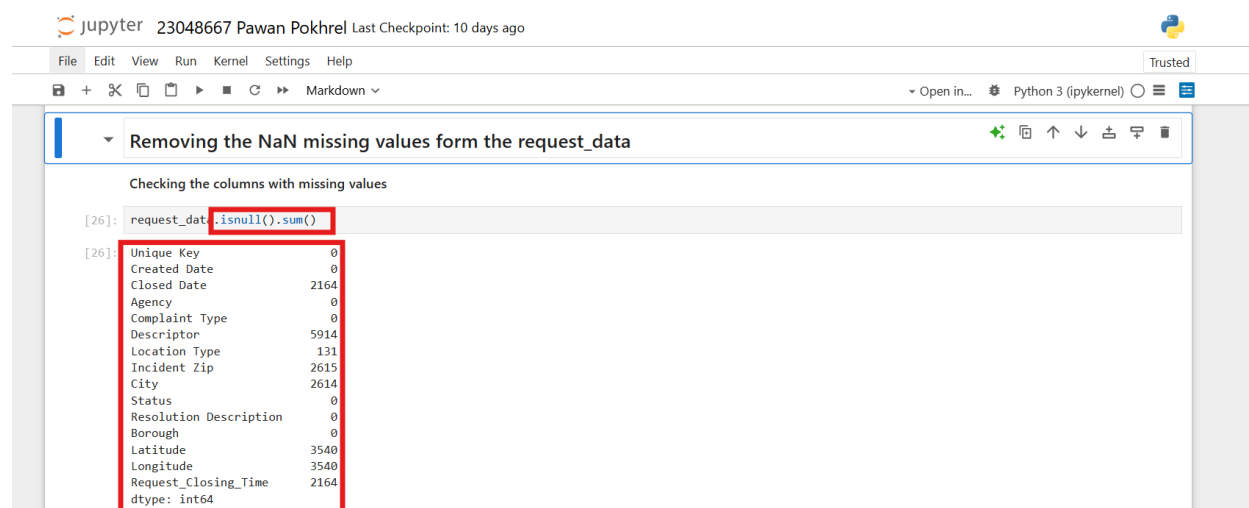
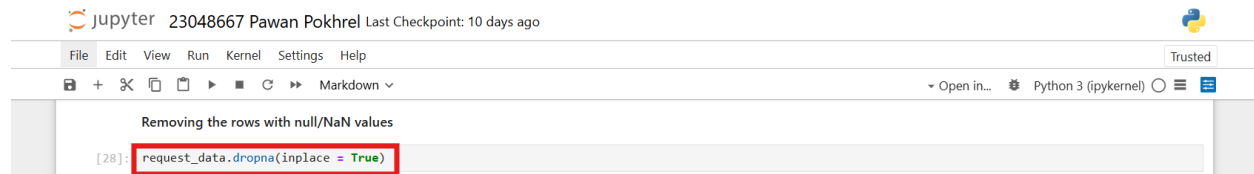


Figure 20: Checking for the possible null values in the columns

These null/NaN rows have been removed from the dataset using the `request_data.dropna(inplace=True)` command. The `dropna()` method removes any row that contains at least one missing value, and by using the `inplace=True` argument, the

changes are applied permanently to the dataset. This was carried out for preprocessing the 311 customer service requests dataset to eliminate incomplete records.



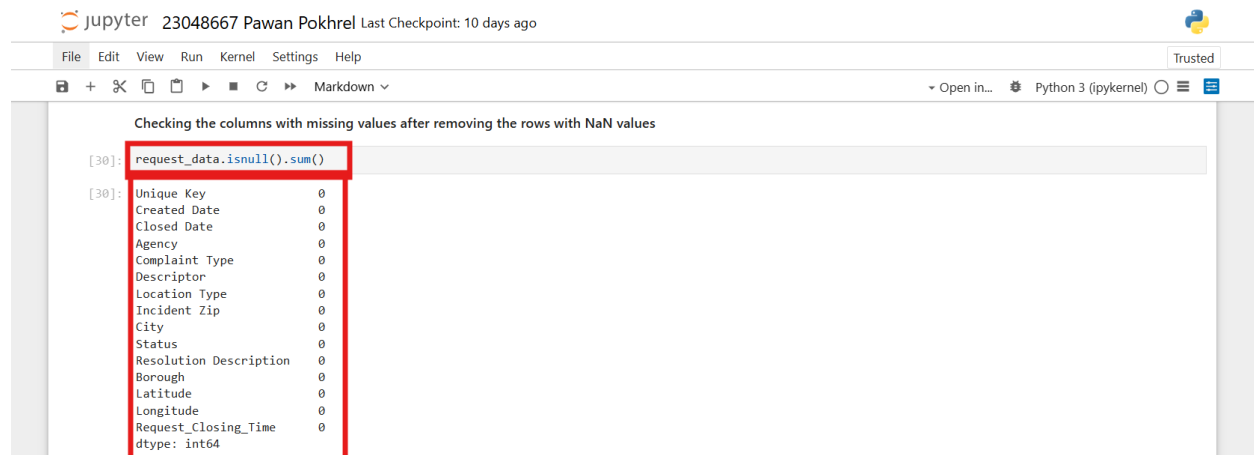
The screenshot shows a Jupyter Notebook interface with the following code cell:

```
[28]: request_data.dropna(inplace = True)
```

The code cell is titled "Removing the rows with null/NaN values".

Figure 21: Removing the null values using .dropna() function

The columns were checked for remaining missing values after the rows with NaN values were removed using the .isnull().sum() functions in the data frame. The .isnull() method identifies any remaining missing values in each column by returning a Boolean value (True for missing, False otherwise), and the .sum() function totals the number of missing values per column. In the screenshot, the number of missing values was found to be 0 for all columns, confirming that all rows with missing values were successfully removed from the 311 customer service requests dataset.



The screenshot shows a Jupyter Notebook interface with the following code cell:

```
[30]: request_data.isnull().sum()
```

The code cell is titled "Checking the columns with missing values after removing the rows with NaN values". The output of the code is displayed below the code cell:

```
[30]: 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 22: Checking for the possible null values after removing the null values

## 2.6. Unique values from columns

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

The unique values were displayed from each column of the dataset using the .unique() function. The print(request\_data.columns) command was used to list all column names, and the print(request\_data[column].unique()) command was applied to each column to

retrieve its unique values. In the screenshot, the unique values for the 'Unique Key' column were shown as [739913735, 739913734, 739913733, ..., 739480826, 739712456], and for the 'Request\_Closing\_Time' column, the unique values were displayed as ['0 days 00:56:15', '0 days 04:51:31', '0 days 01:51:21', ..., '0 days 15:40:46', '0 days 04:14:52'], with a length of 47154 and a data type of timedelta64[ns]. This step was carried out to explore the distinct values present in the 311 customer service requests dataset which helps in understanding of the data distribution and helps identify possible anomalies.

The screenshot shows a Jupyter Notebook window titled 'jupyter 23048667 Pawan Pokhrel Last Checkpoint: 2 hours ago'. The interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar. The main area displays a code cell with the following Python code:

```
[151]: for columns in request_data.columns:
        print(f'Columns: {columns}')
        print(request_data[columns].unique())
        print('-' * 100)
```

The output of the code is displayed below the code cell. It shows the unique values for two columns: 'Longitude' and 'Request\_Closing\_Time'.

```
Columns: Longitude
[-73.92350096 -73.91509394 -73.88852464 ... -73.94880526 -73.87124456
-73.9913785 ]

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

Figure 23: Displaying unique values from each column of the dataset

## DATA ANALYSIS

### 3.1. Sum, Mean, Standard Deviation, Skewness and Kurtosis

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

With the help of the `request_num_data = request_data.select_dtypes(include=['number'])` command, a new Pandas DataFrame with only numeric values was formed for the statistical data analysis. In order to prepare the dataset for reporting summary statistics, `select_dtypes()` method was used in filtering the columns for numeric data types. In the screenshot, a portion of numeric data has been shown in the given columns i.e. Unique Key, Incident Zip, Latitude, Longitude and Request\_Closing\_Time, with the values such as 32301063, 10034.0, 40.865682, etc.

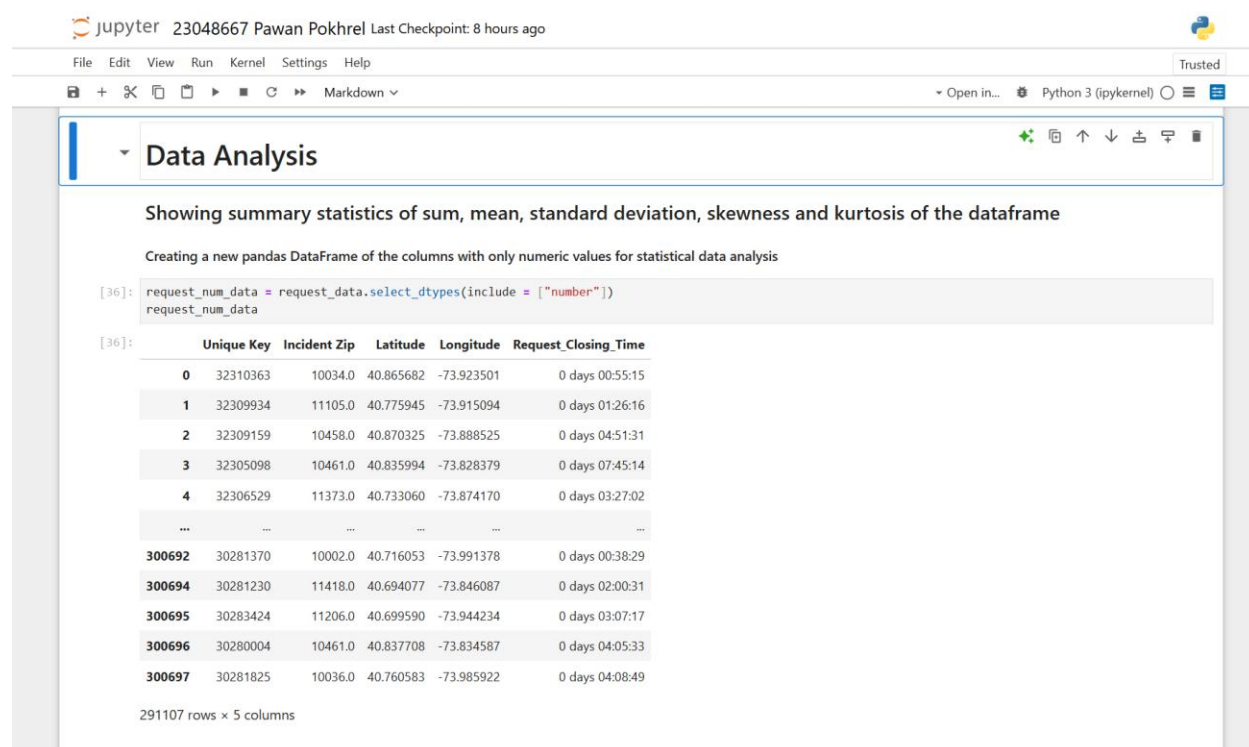


Figure 24: Creating a new dataframe from the updated dataset with only numeric values



The data in 'Request\_Closing\_Time' column was formatted to numerical value by using `.dt.total_seconds() / (60 * 60)`. The conversion of time intervals to seconds was made easier with the `dt.total_seconds()` command followed by the division by `(60 * 60)` to hours. The below screenshot depicted the converted 'Request\_Closing\_Time' values like 0.9375 hours, 1.437778 hours, 4.856611 hours for initial rows.

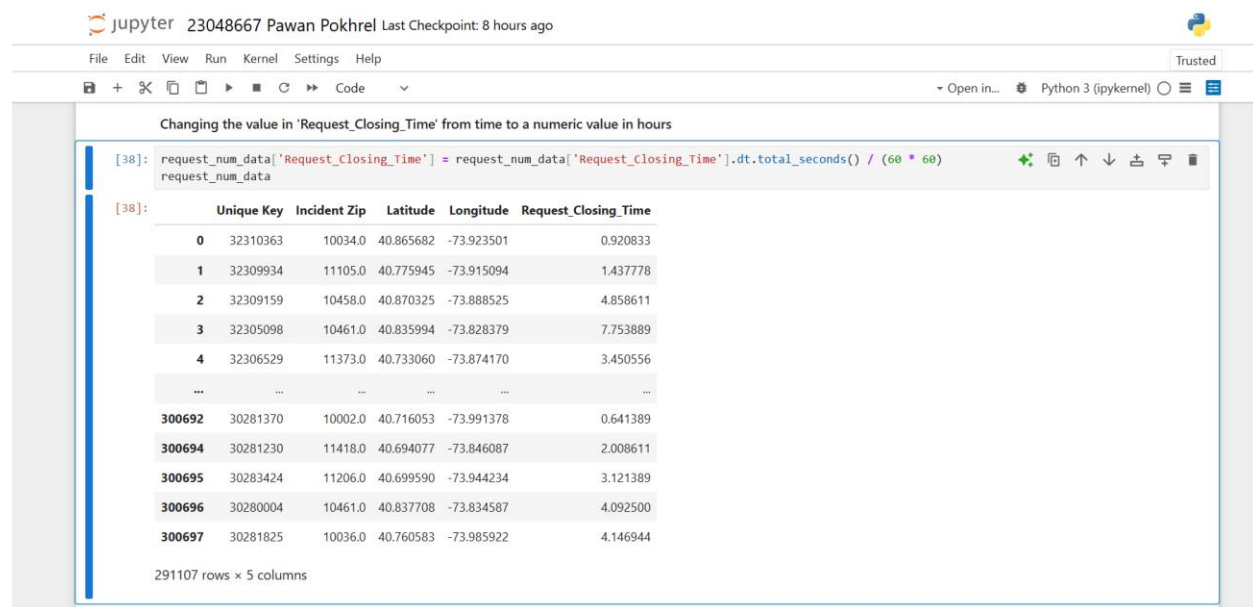


Figure 25: Changing the value in 'Request\_Closing\_Time' to integral value in hours

The sum of the numeric columns was calculated using the `request_num_data.sum()` function. The `sum()` method was applied to add up all values in each numeric column of the dataset. The only useful analysis from the `sum()` function is the `request_closing_time` as it gives the overall request closing time for every single complaint.

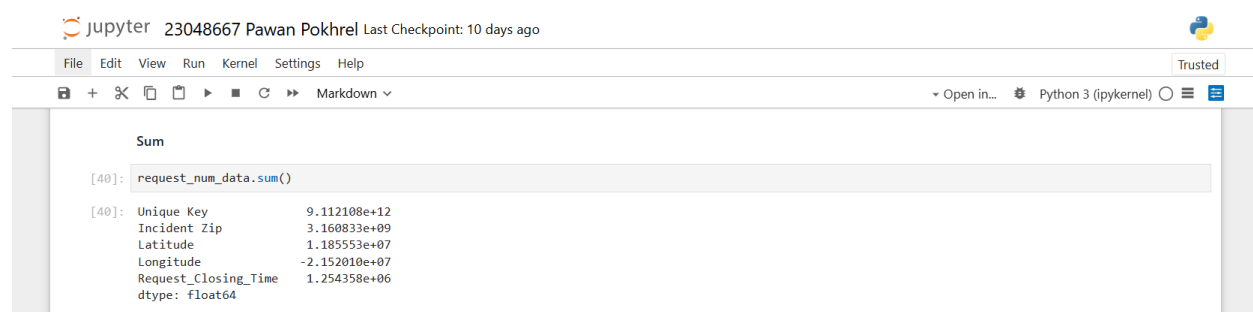


Figure 26: Calculating sum of each numerical columns of dataset individual

The mean of the numeric columns was calculated using the `request_num_data.mean()` function. The `mean()` method was applied to compute the average value for each numeric column in the dataset. From the obtained data, the average value for `request_closing_time` can be learnt. It states that the average response time for the complaints is 4.3 hours.

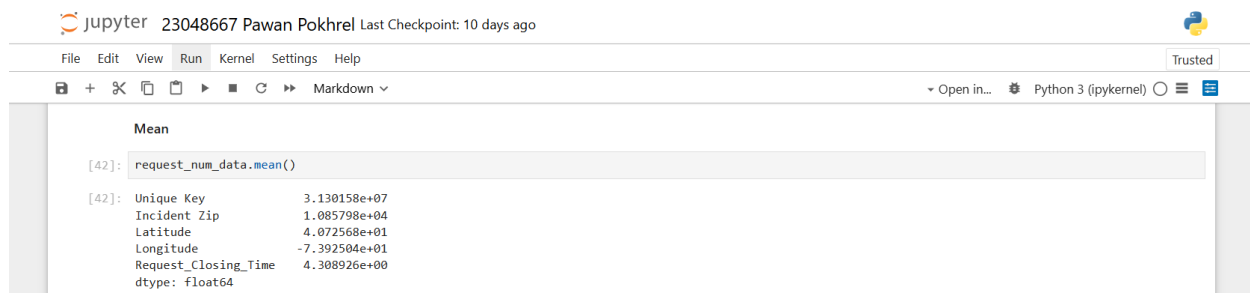


Figure 27: Calculating the mean value of each numerical columns of the dataset individually

The standard deviation of the numeric columns was calculated using the `request_num_data.std()` function. The `std()` method was applied to measure the dispersion of values in each numeric column around the mean. The standard deviation between the values of response time was found to be 6.06 hours.

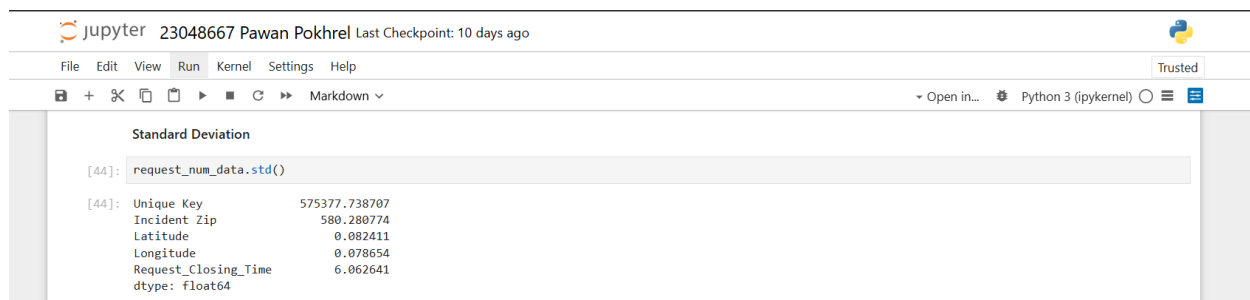


Figure 28: Calculating the standard deviation of each numerical columns of the dataset individually

The skewness of the numeric columns was calculated using the `request_num_data.skew()` function. The `skew()` method was applied to assess the asymmetry of the distribution of values in each numeric column. The request closing time has the highest positive skew value which states the distribution of request closing time is not normal and the distribution's mean lies in the left side of the overall distribution.

Jupyter Notebook interface showing the calculation of skewness for the 'request\_num\_data' dataset. The code cell [46] contains the command `request_num_data.skew()`. The output displays the skewness values for each numerical column: Unique Key (0.016898), Incident Zip (-2.553956), Latitude (0.123114), Longitude (-0.312739), and Request\_Closing\_Time (14.299525). The dtype is float64.

```
[46]: request_num_data.skew()

[46]: Unique Key      0.016898
      Incident Zip   -2.553956
      Latitude       0.123114
      Longitude     -0.312739
      Request_Closing_Time 14.299525
      dtype: float64
```

Figure 29: Calculating the value of skewness of each numerical columns of the dataset individually

The kurtosis of the numeric columns was calculated using the `request_num_data.kurt()` function. The `kurt()` method was applied to measure the tailedness of the distribution of values in each numeric column.

Jupyter Notebook interface showing the calculation of kurtosis for the 'request\_num\_data' dataset. The code cell [181] contains the command `request_num_data.kurt()`. The output displays the kurtosis values for each numerical column: Unique Key (-1.176593), Incident Zip (37.827777), Latitude (-0.734818), Longitude (1.455600), and Request\_Closing\_Time (849.777081). The dtype is float64.

```
[181]: request_num_data.kurt()

[181]: Unique Key      -1.176593
      Incident Zip    37.827777
      Latitude       -0.734818
      Longitude      1.455600
      Request_Closing_Time 849.777081
      dtype: float64
```

Figure 30: Calculating kurtosis of the dataset through `.kurt()`

The sum, mean, standard deviation, skewness and kurtosis of each column were collectively displayed in a pandas dataframe.

Jupyter Notebook interface showing the collective display of sum, mean, standard deviation, skewness, and kurtosis for the 'request\_data\_statistics' dataset. The code cell [192] contains the command `request_data_statistics = pd.DataFrame([request_num_data.sum(), request_num_data.mean(), request_num_data.std(), request_num_data.skew(), request_num_data.kurt()], index = ['Sum', 'Mean', 'Standard Deviation', 'Skewness', 'Kurtosis'])`. The output displays a pandas DataFrame with the following statistics:

```
[192]: request_data_statistics = pd.DataFrame([
      request_num_data.sum(),
      request_num_data.mean(),
      request_num_data.std(),
      request_num_data.skew(),
      request_num_data.kurt()
    ], index = ['Sum', 'Mean', 'Standard Deviation', 'Skewness', 'Kurtosis'])
      request_data_statistics
```

	Unique Key	Incident Zip	Latitude	Longitude	Request_Closing_Time
<b>Sum</b>	9.112108e+12	3.160833e+09	1.185553e+07	-2.152010e+07	1.254358e+06
<b>Mean</b>	3.130158e+07	1.085798e+04	4.072568e+01	-7.392504e+01	4.308926e+00
<b>Standard Deviation</b>	5.753777e+05	5.802808e+02	8.241087e-02	7.865356e-02	6.062641e+00
<b>Skewness</b>	1.689772e-02	-2.553956e+00	1.231144e-01	-3.127386e-01	1.429953e+01
<b>Kurtosis</b>	-1.176593e+00	3.782778e+01	-7.348183e-01	1.455600e+00	8.497771e+02

Figure 31: Collectively displaying sum, mean, standard deviation, skewness and kurtosis in a table

### 3.2. Correlation

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

With the help of `.corr()` function the correlations between each variable were displayed in a tabular format.

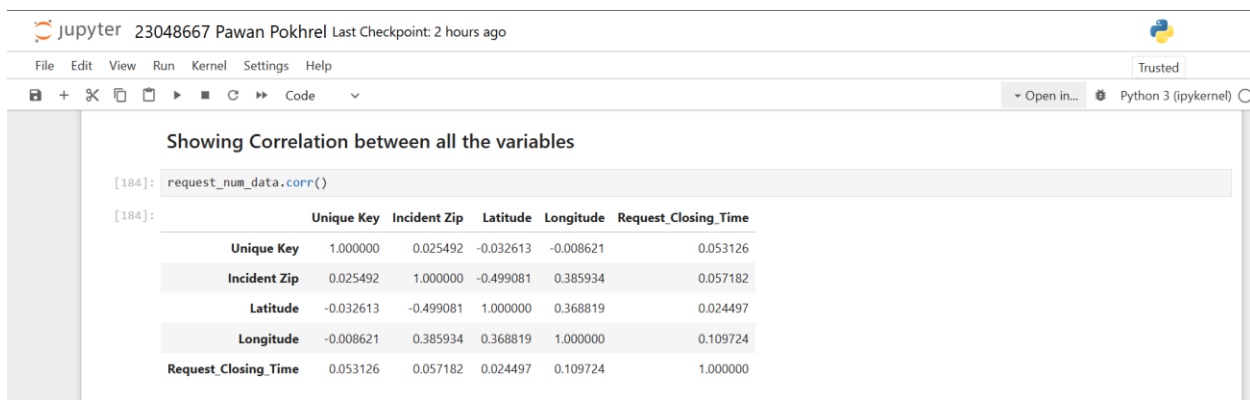


Figure 32: Displaying the correlation between each of the variables

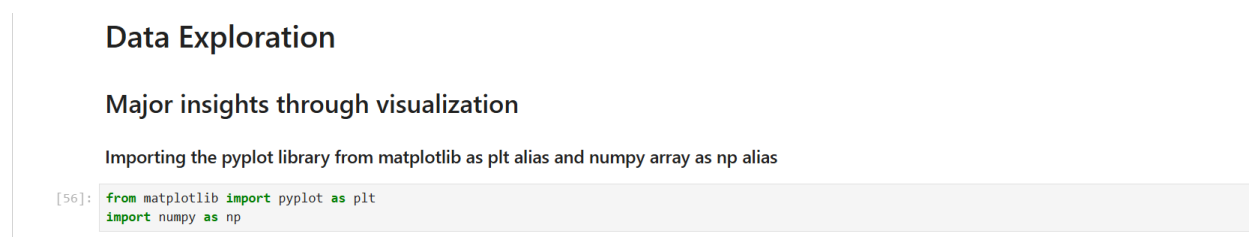
## DATA EXPLORATION

### 4.1. Insights through data visualization

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

The 311-customer service requests dataset was thoroughly analysed using multiple data visualisation techniques which provided meaningful insights and uncovered important patterns and trends of the customer service requests. The dataset was explored from multiple perspectives through the application of visual analytics which helped for the identification of most prevalent complaint types, distribution of the customer requests across different boroughs and temporal fluctuations in service demands. The data visualization techniques proved instrumental during the recognition of patterns which could not have been highlighted through raw data alone. Additionally, the areas with consistently high numbers of complaints were also mapped out, which could indicate the localized issues that must be addressed in a focused way. From this visual data exploration, four major insights were derived each yielding valuable information for comprehending urban service problems and informing data-driven decision-making for public service improvement.

Below is the screenshot for the initial process involved during the Data Exploration process which is importing the necessary libraries required for Data Visualization. Pyplot library from matplotlib was imported for the visualization processes and numpy array was imported to support numerical operations.



*Figure 33: importing pyplot library from matplotlib*

The major insights that were gained from the dataset are listed as follows:

#### 4.1.1. Most repeated complaint types

The first insight from the dataset was the frequency of repetition of complaint types. The most frequent complaint types represent the issues most reported by the NYC residents which occurs repeatedly in the city. For example, the complaints like “Blocked Driveway”, “Illegal Parking” and “Noise – Street/Sidewalk” appears to be the most frequent ones which indicates the widespread concerns related to quality of life and traffic enforcements. These frequent complaints insight could help city officials on prioritizing resources and addressing the most pressing public issues.

Below is the screenshot and description of the process involved during the data visualization of the top complaints or the highest frequent complain types from the NYC 311-customer service requests dataset.

- The `request_data['Complaint Types'].value_counts()` returns a Pandas Series with the number of complaints for each unique Complaint Type, in which the Complaint Types being the index and their frequencies being the value which was stored to `complaint_counts`.
- `plt`, i.e. `pyplot` (alias `plt`), is used for the entire data visualization process. `plt.figure()` helps in changing the size of the diagram according to our requirement.
- `plt.bar()` creates a bar chart with the unique complaint types (the categories for the bars) in the x-axis which is the `complaint_counts.index` and the value counts (the height of the bars) is on the y-axis which is the `complaint_counts.values`. The `width` attribute defines how wide the individual bar in the bar chart should be.
- `plt.xticks()` defines the bar labels in the x-axis. The `rotation` attribute defines the rotation of the label texts in degrees, i.e. `rotation = 90` defines the label to be vertical which helps in avoiding overlapping labels.
- `plt.title()` sets the title of the pyplot charts making it clear to the viewer what the chart represents.
- `plt.xlabel()` labels the x-axis which informs what the horizontal or x-axis in the chart represents. Here, `plt.xlabel("Complaint Types")` names the x-axis as Complaint Types.

- `plt.ylabel()` labels the y-axis which informs what the vertical or y-axis in the chart represents. Here, `plt.ylabel("No. of Complaints")` names the y-axis as No. of Complaints.
- `plt.show()` renders the plot and displays the chart to the user.

#### Top Complaint Types

```
[58]: complaint_counts = request_data['Complaint Type'].value_counts()
plt.figure(figsize = (14, 8))
plt.bar(complaint_counts.index, complaint_counts.values, width = 0.8)
plt.xticks(rotation=90)
plt.title("Top Complaint types received")
plt.xlabel('Complaint Types')
plt.ylabel('No. of complaints')
plt.show()
```

Figure 34: process involved in bar chart construction

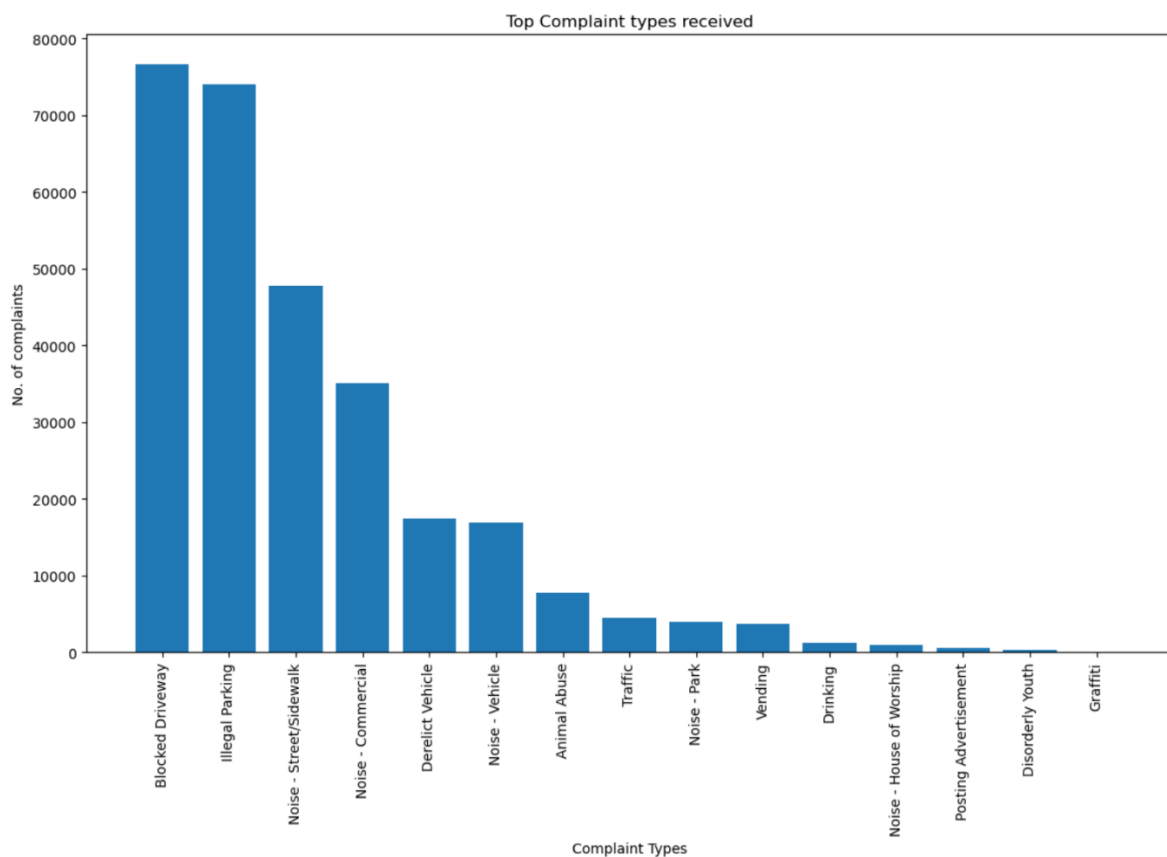


Figure 35: Insight 1: Most frequent complaint types

From the insight 1, we can observe that the most common complaint types were, Blocked Driveway and Illegal Parking whereas the least common complaint type was Graffiti.

#### 4.1.2. Geographical Concentration of the complaints

The second insight derived from the dataset was the distribution of complaint across the different boroughs of NYC. The complaints classified by boroughs helps in understanding in which boroughs have the highest number of complaints which indicates the areas that may require more attention and resources. This insight represents wither a higher population density in the area, more active reporting behaviour or repeatedly occurring issues in specific areas. For example, Brooklyn and Queens have been seen the boroughs with the highest number of complaints over the time which indicates more urban challenges or greater public awareness. Hence, the insight can support local officers/authorities in geographically targeting the responses and improving their service delivery.

Below is the screenshot and description of the code and process involved in visualizing the number of complaints received from each borough in the NYC 311-customer service requests dataset.

- The `request_data['Borough'].value_counts()` function was used to obtain the number of complaints made from each borough. The Pandas Series result was stored in the variable `complaint_borough`, where each borough represents an index and the number of complaints in its corresponding value.
- `plt.figure()` helps in changing the size of the diagram according to our requirement.
- `plt.pie()` creates a pie chart for the percentages of complaints across the boroughs of NYC. The `autopct = '%1.1f%%'` attribute set the percentage labels inside each slice of the pie chart which gave viewers a clearer view of the proportion of each borough that contributed to the total and the `colour` attribute set the colours of each individual slice for better distinction between the boroughs.
- `plt.title()` sets the title of the chart as “No. of Complaints received in each Borough” which informs the viewer about what the pie chart is illustrating.
- `plt.show()` rendered the final chart and displayed it to the user.



## Complaints by Borough

```
[60]: complaint_borough = request_data['Borough'].value_counts()
plt.figure(figsize = (14, 8))
plt.pie(complaint_borough.values, labels=complaint_borough.index, autopct='%1.1f%%', colors=['#2F6787', '#4F7FB7', '#6F97C7', '#8FADF7', '#A7C7E7'])
plt.title("No. of Complaints received in each Borough")
plt.show()
```

Figure 36: process involved in creating a pie chart for complaints received in each borough

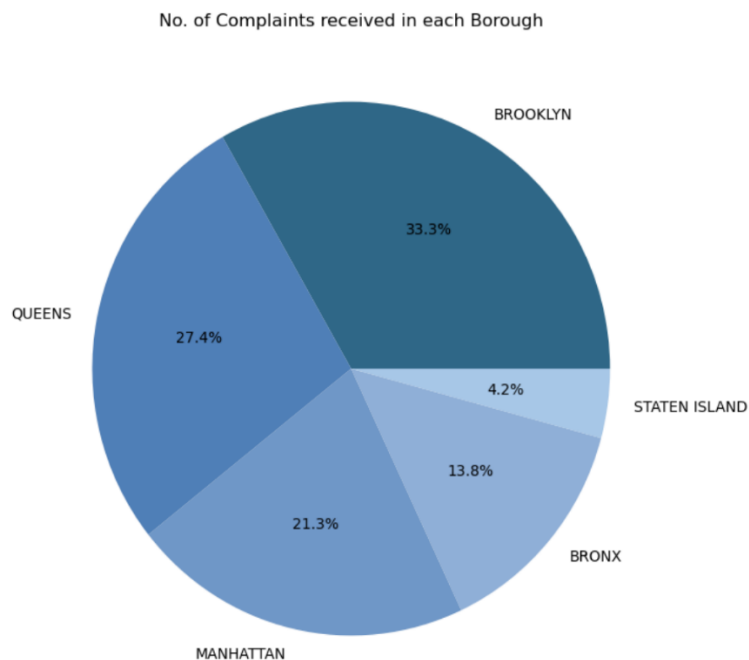


Figure 37: Insight 2: Complaints in individual borough

From the pie chart of the distribution of complaints across the Boroughs, we can learn that Brooklyn had the greatest number of complaints over the time and Staten Island had the least number of complaints.

This can also be illustrated in the chart with a light emphasis as follows:

- The explode = [0.1, 0, 0, 0, 0] ensured to explode the first slice slightly outward than the pie chart which was the slice with the greatest proportionality.

Borough with the highest no. of complaints

```
plt.figure(figsize = (14, 8))
plt.pie(complaint_borough.values, labels=complaint_borough.index, autopct='%1.1f%%', colors=['#2F6787', '#4F7FB7', '#6F97C7', '#8FADF7', '#A7C7E7'],
        explode = [0.1, 0, 0, 0, 0])
plt.title("Borough with the highest no. of complaints received")
plt.show()
```

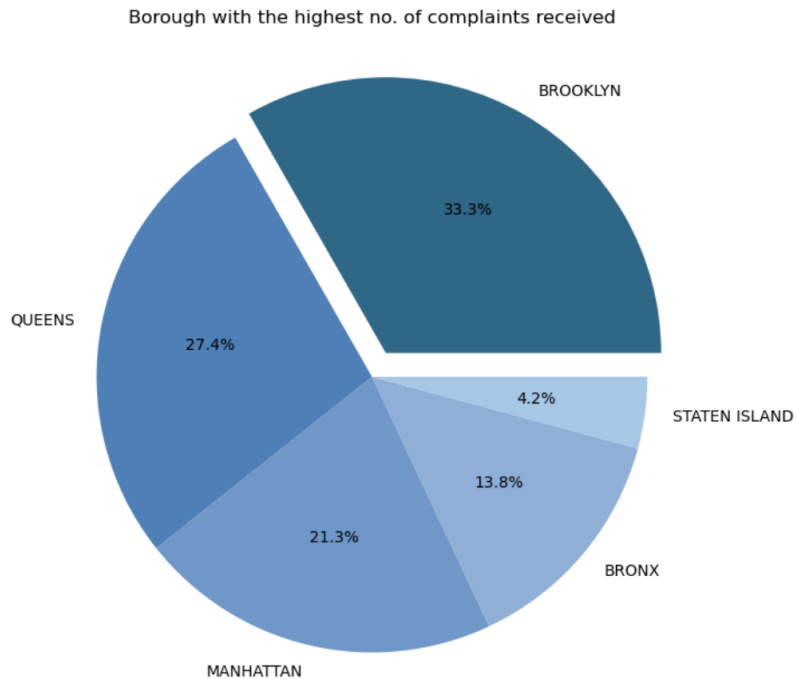


Figure 38: Insight 2i: pie chart with enhanced borough with highest no. of complaints received

- The explode = [0, 0, 0, 0, 0.3] ensured to explode the last slice slightly outward than the pie chart which was the slice with the least percentage in the pie chart.

Borough with the lowest no. of complaints

```
plt.figure(figsize = (14, 8))
plt.pie(complaint_borough.values, labels=complaint_borough.index, autopct='%1.1f%%', colors=['#2F6787', '#4F7FB7', '#6F97C7', '#8FAFD7', '#A7C7E7'],
        explode = [0, 0, 0, 0, 0.3])
plt.title("Borough with the least no. of complaints received")
plt.show()
```

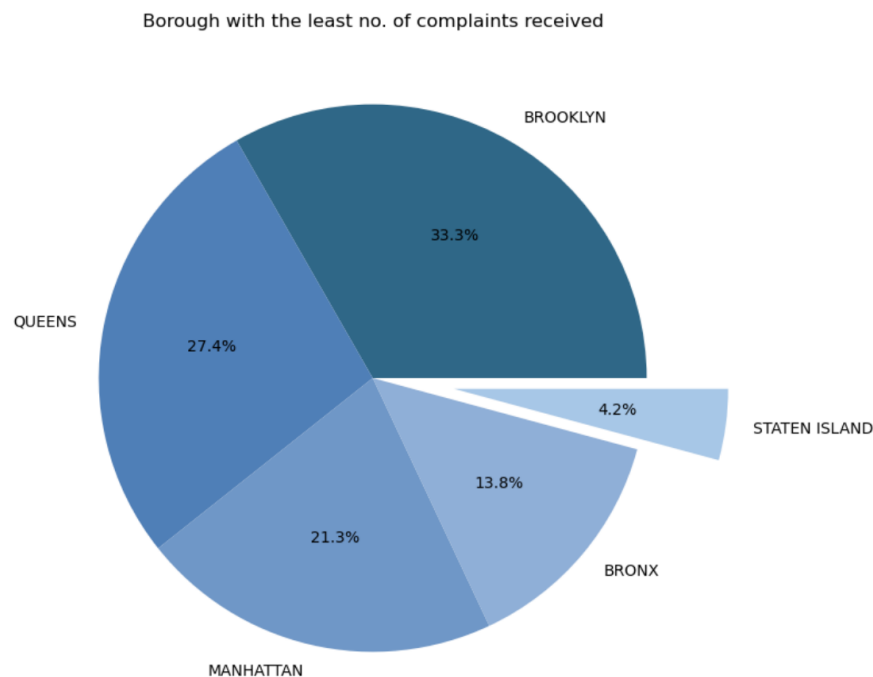


Figure 39: Insight 2ii: pie chart with enhanced borough with least no. of complaints received

#### 4.1.3. Complaint Trends over the time

The third insight from the dataset was the complaints trend throughout the year 2015 which helps in understanding how the number of service requests changes cross the months or weeks of the year 2015. This analysis helps in recognition of seasonal or cyclic patterns, peak periods and possible anomalies in public service demands. It was found that some months experience noticeably higher complaint numbers which could be associated with the weather-related issues, holidays or seasonal public behaviour. For example, the warmer months are likely to sow a rise in noise or heat-related complaint which the colder months may reflect concerns to heating or road conditions. The recognition of such trends and patterns over a certain period can help the local agencies prepare for complaint surges and optimize resource plannings across the various seasons.

Below is the screenshot and the explanation of the process used to visualize the monthly distribution of complaints for the year 2015 from 311-customer service request dataset:

- The `request_data['Created Date'].dt.month.value_counts().sort_index()` returned a Pandas Series with the number of complaints in each month sorted in chronological order, i.e. January to December.
- `plt.figure()` helped in adjusting the size of the line chart to make it clearly visible and well-spaced for all 12 months.
- `plt.plot()` plotted the data as a line chart with a sky blue filled circular marker by using `marker = 'o'` and `mfc (marker face color) = 'skyblue'` with a blue border using `mec (marker edge color) = 'blue'`. The `linewidth` attribute changes the thickness of the line in the chart.
- `plt.title()` sets the title of the chart as “No. of Complaints in 2015 AD over months” which informs the viewer about what the pie chart is illustrating.
- `plt.xlabel()` labels the x-axis which informs what the horizontal or x-axis in the chart represents. Here, `plt.xlabel("Month", fontsize = 14)` names the x-axis as Months with a larger font size than the default.
- `plt.ylabel()` labels the y-axis which informs what the vertical or y-axis in the chart represents. Here, `plt.ylabel("Number of Complaints", fontsize = 14)` names the y-axis as Number of Complaints with a larger font size than the default.
- `plt.xticks()` customizes the x-axis data labels using standard month abbreviations from January (Jan) to December (Dec) for improved readability.
- `plt.grid(True)` set the gridlines in the plot helping in following the data more accurately.
- `plt.show()` rendered and displayed the final line chart to the user.

## Complaint Trends in 2015 AD over months and weeks

## Over Months

```
[144]: complaint_dates_month = request_data['Created Date'].dt.month.value_counts().sort_index()

plt.figure(figsize=(14, 8))
plt.plot(complaint_dates_month.index, complaint_dates_month.values, marker = 'o', mfc = 'skyblue', mec = 'blue', color = 'skyblue', linewidth=2)

plt.title('Number of Complaints in 2015 AD over months', fontsize = 16)
plt.xlabel('Month', fontsize = 14)
plt.ylabel('Number of Complaints', fontsize = 14)
plt.xticks(ticks=range(1, 13), labels=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.grid(True)
plt.show()
```

Figure 40: process involved in creating a line chart for complaint trends over months of 2015

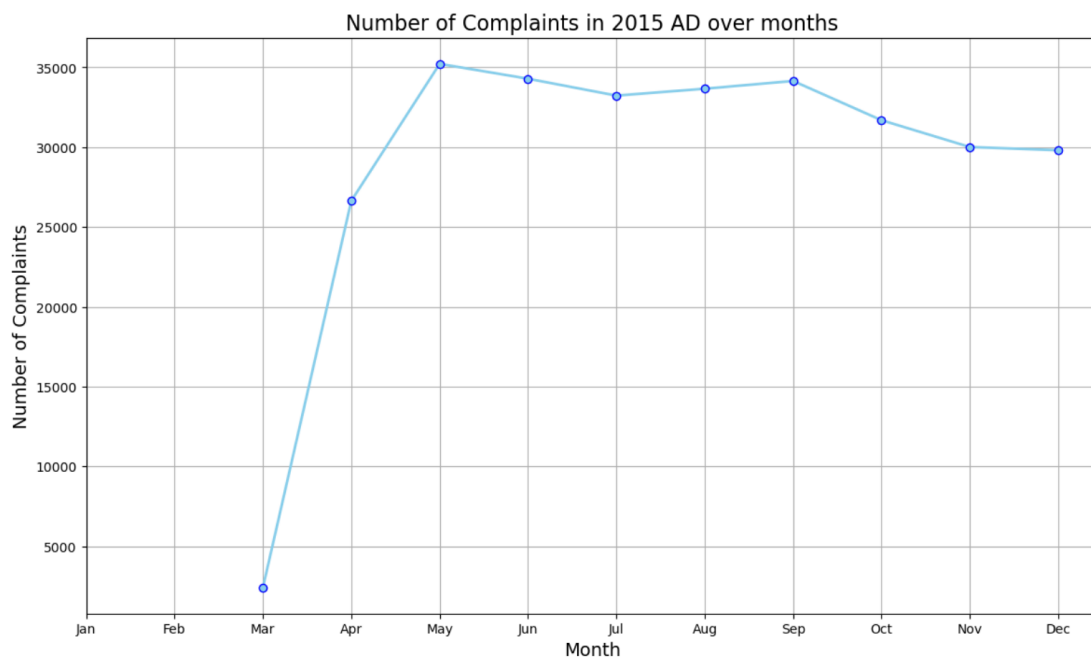


Figure 41: Insight 3a: Complaint trends in 2015 over months

From the line chart of monthly trends of complaint volumes over 2015, we can observe that the volume of complaints was higher during May 2015 to September 2015 in comparison to the overall months. May had the highest number of complaints in a month with touching over 35000 complaints and March had the least number of complaints as per the data with not even 5000 complaints.

Below is the screenshot and the explanation of the process used to visualize the weekly distribution of complaints for the year 2015 from 311-customer service request dataset. The process in monthly trends line chart and the weekly trend of complaint volumes

histogram is almost identical except the data is grouped according to the weeks and instead of a line chart histogram is being used:

- `request_data.loc[request_data['Created Date'].notna(), 'Created Date'].dt.isocalendar().week` extracted the ISO week number (1 to 53) from each non-null 'Created Date' value using `.dt.isocalendar().week`. This created a Pandas Series containing the week of the year when each complaint was made and stored to weeks.
- `plt.hist()` plotted a histogram to show the frequency of the complaints in each week. The `bins` attribute was used to define the number of bar that is to be used to represent the histogram, `edgecolor` attribute to define the border color of each bin, `color` attribute to define the filling color of each bin and `align` to define the bar alignment with respect to bin range.
- `plt.xticks(range(1, 53, 2))` customized the tick marks on the x-axis to show every second week for better readability and less clutter.
- `plt.grid(axis='y', linestyle=':', alpha=0.7)` enabled the horizontal gridlines with a dotted style defined by the `linestyle` attribute and semi-transparent or (70% opaque) appearance to help track bar heights visually.
- `plt.tight_layout()` prevents the overlapping of plot elements and ensures proper layout.

```
Over Weeks
[130]: weeks = request_data.loc[request_data['Created Date'].notna(), 'Created Date'].dt.isocalendar().week

plt.figure(figsize=(14, 8))
plt.hist(weeks, bins=range(1, 54), edgecolor='black', color='teal', align='left')

plt.title('Histogram of Complaints per Week (2015)', fontsize=16)
plt.xlabel('Week Number', fontsize=14)
plt.ylabel('Number of Complaints', fontsize=14)
plt.xticks(range(1, 53, 2))
plt.grid(axis='y', linestyle=':', alpha=0.7)

plt.tight_layout()
plt.show()
```

Figure 42: process involved in creating a line chart for complaint trends over months of 2015

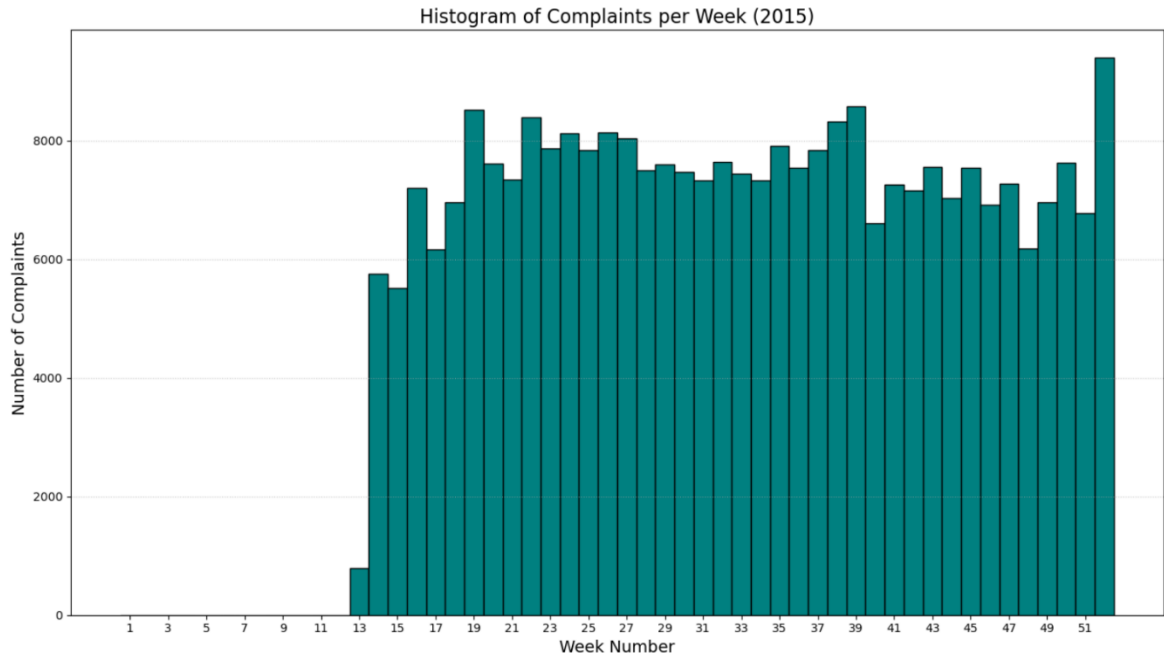


Figure 43: Insight 3b: Complaint trends in 2015 over weeks

From the histogram of weekly trend of the complaints, we can observe that the last week of the year had the most complaints due to some reason and the thirteenth week had the least complaints as per the dataset we are provided with.

#### 4.1.4. Average Service Response Time

The fourth insight from the dataset was the average time taken to resolve the complaints for each complaint types. This analysis provides valuable information about the efficiency and responsiveness of the city services for the various type of issues. The complaint types with longer resolve time suggests the complex issues or potential inefficiencies in the responsible departments. The complaints resolved quickly may reflect well-established workflows and better resource allocation for the issues. This insight can be used by the city officials in evaluation of performance, identification of bottlenecks in service delivery and improve the overall response times across the complaint types to public concerns.

Below is the screenshot and explanation of the process involved to visualize the average closing time for complains by the types from NYC 311-customer service request dataset:

- `request_data['Request_Closing_Time'].dt.total_seconds() / 3600` converted the date time data to a numeric data in total hours by extracting the total seconds using the `.dt.total_seconds()` and dividing it by  $(60 * 60)$ , i.e. 3600 second = 1 hour
- the average request closing time was extracted from the dataframe `request_data` which was grouped by the complaint types and was stored to `avg_closing_time`
- `plt.barh()` created a horizontal bar chart with complaint types on y-axis and the corresponding average closing time on the x-axis

Average Response Time for each complaint types

Converting Request\_Closing\_Time into a numeric data in the dataframe

```
[69]: request_data['Request_Closing_Time'] = request_data['Request_Closing_Time'].dt.total_seconds() / 3600

[70]: avg_closing_time = request_data.groupby('Complaint Type')['Request_Closing_Time'].mean().sort_values(ascending=True)

plt.figure(figsize=(14, 10))
plt.barh(avg_closing_time.index, avg_closing_time.values, color='steelblue')
plt.title("Average Request Closing Time by Complaint Type", fontsize=16)
plt.xlabel('Avg Closing Time (Hours)', fontsize=14)
plt.ylabel('Complaint Types', fontsize=14)
plt.grid(True, axis='x')
plt.tight_layout()
plt.show()
```

Figure 44: process involved in creating a horizontal bar chart for average response time of each complaint types



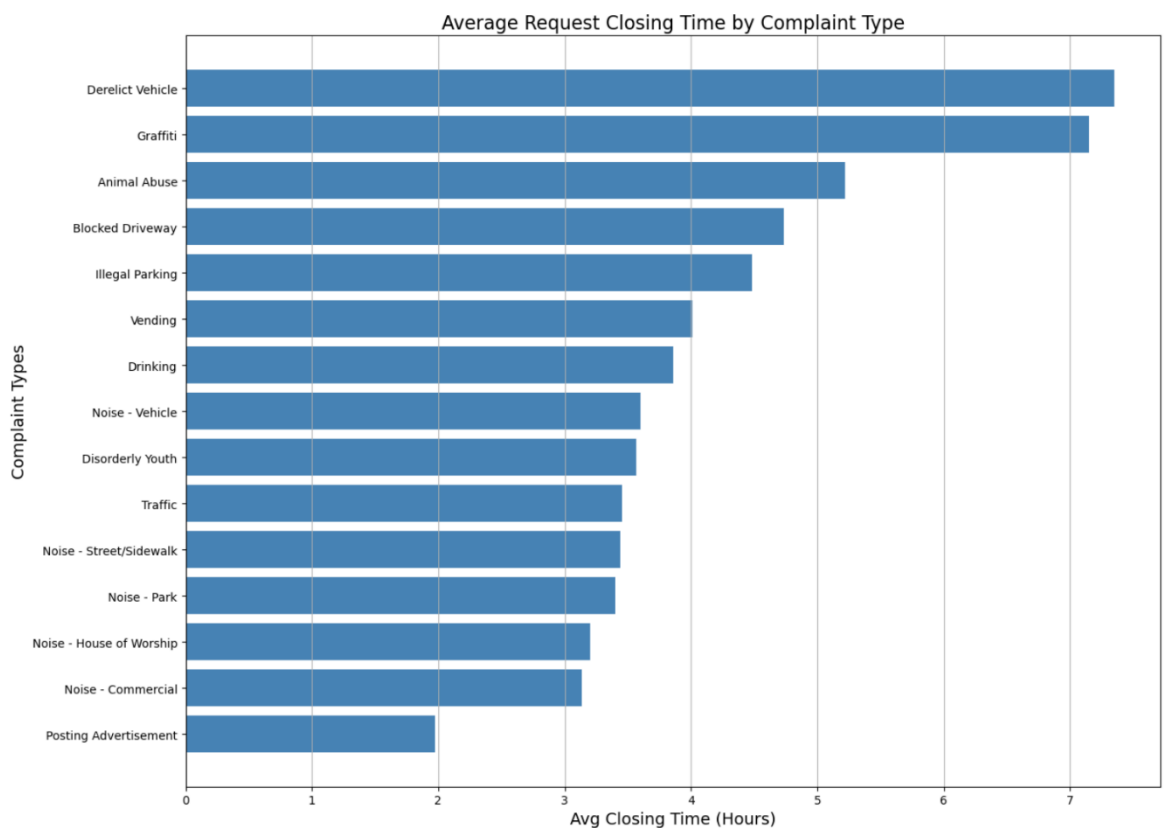


Figure 45: Insight 4: Average response time of each complaint types

From the above horizontal bar chart, we can observe the average response time of each of the complaint types. The fastest response time was for complaint related to Posting Advertisement whereas the slowest response time was for Derelict Vehicle complaints as per the 311 customer service requests dataset.

#### 4.2. Average response time of each complaint types in various locations

The following visualization compares the average request closing time for each complaint type across all NYC boroughs and the average request closing time in each NYC boroughs for all complaint types. These comparisons allow for a in-depth analysis at how quickly the various complaint types are addressed in different borough and how quickly the various boroughs address each complaint types respectively. The differences in response times may be affected due to the borough specific infrastructure, resources, staffing levels or complaint volumes. By analysing this data, city departments can identify the places for efficiency improvements or the best practises from the other boroughs can be implemented.

Below is the processed involved during the comparative visualization process for average response time of each complaint types in borough:

- The average response time was extracted from the dataframe request\_data grouped by borough and complaint type and was stored to avg\_closing.
- The avg\_closing data was transformed into a pivot table with complaint types index and borough as values which led to the chart visualization more focused on the complaint types.
- For loop was initiated to plot each complaint types bar separately.

▼ Average Response Time of different complaint types in each Borough and Location Types

```
[71]: avg_closing = request_data.groupby(['Borough', 'Complaint Type'])['Request_Closing_Time'].mean().reset_index()
```

Average Response time in boroughs for each complaint type

```
[72]: pivot_complaint_type = avg_closing.pivot(index='Complaint Type', columns='Borough', values='Request_Closing_Time').fillna(0)

x = np.arange(len(pivot_complaint_type.index))
colors = ['steelblue', 'skyblue', 'dodgerblue', 'deepskyblue', 'powderblue']

plt.figure(figsize=(18, 10))
for index, borough in enumerate(pivot_complaint_type.columns):
    plt.bar(x + index * 0.14, pivot_complaint_type[borough], width = 0.14, label = borough, color = colors[index % len(colors)])

plt.xlabel('Complaint Type', fontsize = 14)
plt.ylabel('Average Request Closing Time (in Hours)', fontsize = 14)
plt.title('Average Request Closing Time by Complaint Type and Borough', fontsize = 16)
plt.xticks(ticks = range(0, 15), labels = pivot_complaint_type.index, rotation = 90)
plt.legend(title = 'Borough')
plt.grid(True, axis = 'y', linestyle = ':', alpha = 0.7)
plt.tight_layout()
plt.show()
```

Figure 46: process involved in visualizing average response time by complaint types and borough

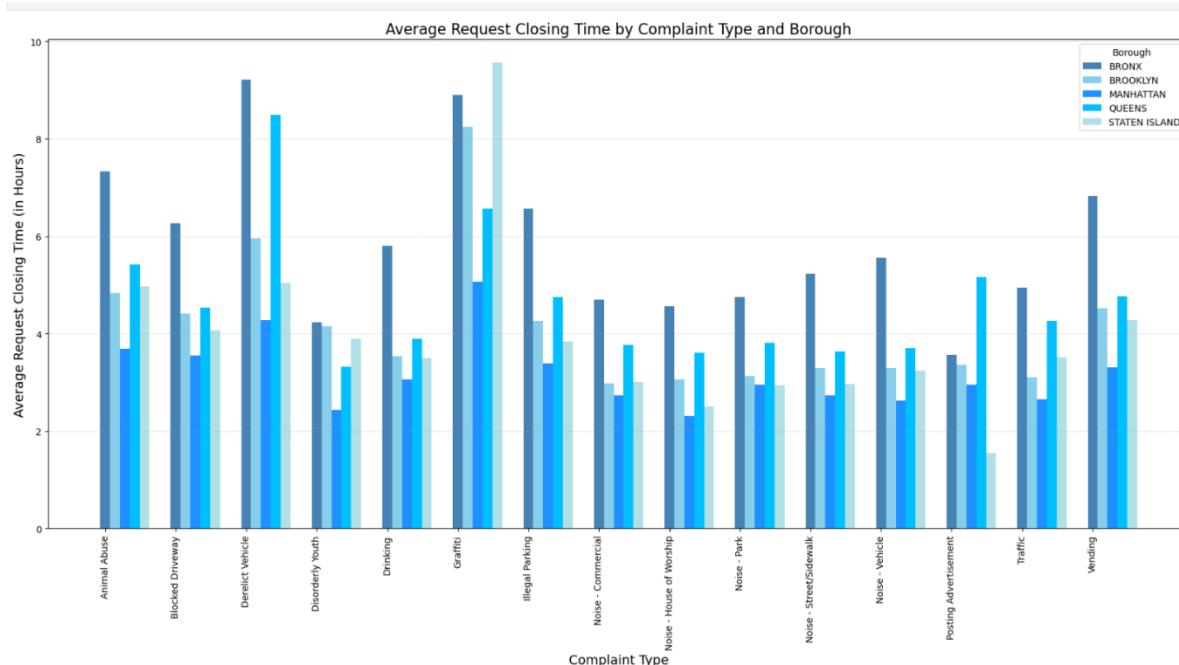


Figure 47: Average response time of each complaint types grouped by borough

From the above stacked bar chart of the average response time of each complaint types further grouped by boroughs, we can observe the average response time taken by the officers of each borough for each complaint types.

Below is the processed involved during the comparative visualization process for average response time of complaint types in each borough:

- The avg\_closing data was transformed into a pivot table with borough index and complaint types as values which led to the chart visualization more focused on the borough.
- For loop was initiated to plot each borough bar separately.

▼ Average Response time of complaint types in each borough

```
[73]: pivot_borough = avg_closing.pivot(index='Borough', columns='Complaint Type', values='Request_Closing_Time').fillna(0)

boroughs = pivot_borough.index.tolist()
complaint_types = pivot_borough.columns.tolist()
x = np.arange(len(boroughs))
colors = ['steelblue', 'skyblue', 'dodgerblue', 'deepskyblue', 'powderblue', 'cornflowerblue', 'lightskyblue']

plt.figure(figsize=(18, 10))

for index, complaint in enumerate(complaint_types):
    plt.bar(x + index * 0.06, pivot_borough[complaint], width = 0.06, label = complaint, color = colors[index % len(colors)])

plt.xlabel('Borough', fontsize = 14)
plt.ylabel('Average Request Closing Time (in Hours)', fontsize = 14)
plt.title('Average Request Closing Time by Borough and Complaint Type', fontsize = 16)
plt.xticks(ticks = x + 0.06 * (len(complaint_types) - 1) / 2, labels = pivot_borough.index)
plt.legend(title = 'Complaint Type', loc='upper right')
plt.grid(True, axis = 'y', linestyle = ':', alpha = 0.7)
plt.tight_layout()
plt.show()
```

Figure 48: process involved in visualizing average response time by borough and complaint types

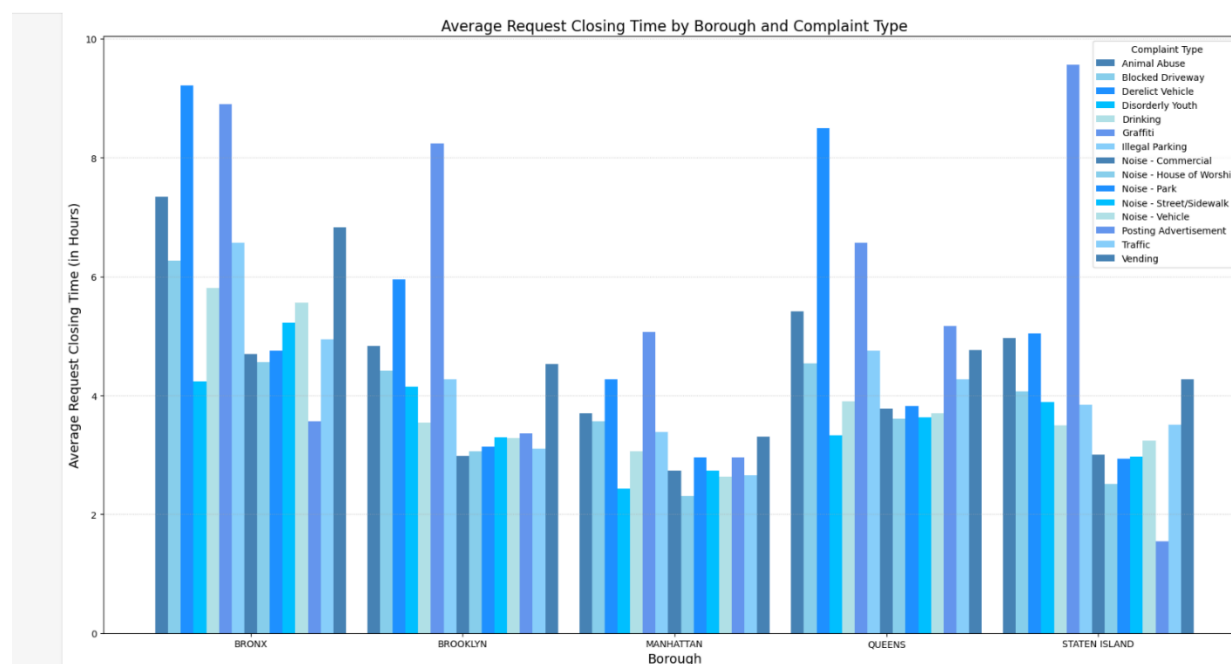


Figure 49: Average response time of each borough grouped by complaint types

From the above stacked bar chart of the average response time of each borough further grouped by complaint types, we can observe the average response time taken for each complaint type in each of the borough.

Below is the processed involved during the comparative visualization process for average response time of complaint types in each location types:

- The average response time was extracted from the dataframe request\_data grouped by location types and complaint type and was stored to avg.
- The avg data was transformed into a pivot table with location types index and complaint types as values to construct a heat table or heat map between the location types and complaint types.
- seaborn library as sns helped in building the heat map, sns.heatmap() created the heatmap between the location types and complaint types for average response time.

Heatmap for average response time of complaint types in various location types

```
[195]: import seaborn as sns

# Grouping and Pivot
avg = request_data.groupby(['Location Type', 'Complaint Type'])['Request_Closing_Time'].mean().reset_index()
pivot = avg.pivot(index='Location Type', columns='Complaint Type', values='Request_Closing_Time').fillna(0)

# Plotting with simpler styling
plt.figure(figsize=(18, 10))
sns.heatmap(
    pivot,
    annot=True, fmt='.2f', cmap='YlGnBu', cbar=True, # Soft blue color palette
    annot_kws={"size": 10}, # Smaller annotation text
    linewidths=0.5, linecolor='gray', # Thin gridlines for a cleaner look
    cbar_kws={"label": "Avg Closing Time (Hours)"}, # Color bar Label
)

# Customize titles and Labels
plt.title('Average Request Closing Time by Complaint Type and Location Type', fontsize=16)
plt.xlabel('Complaint Type', fontsize=14)
plt.ylabel('Location Type', fontsize=14)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Adjust Layout
plt.tight_layout()
plt.show()
```

Figure 50: process involved in visualizing average response time by complaint types and location types

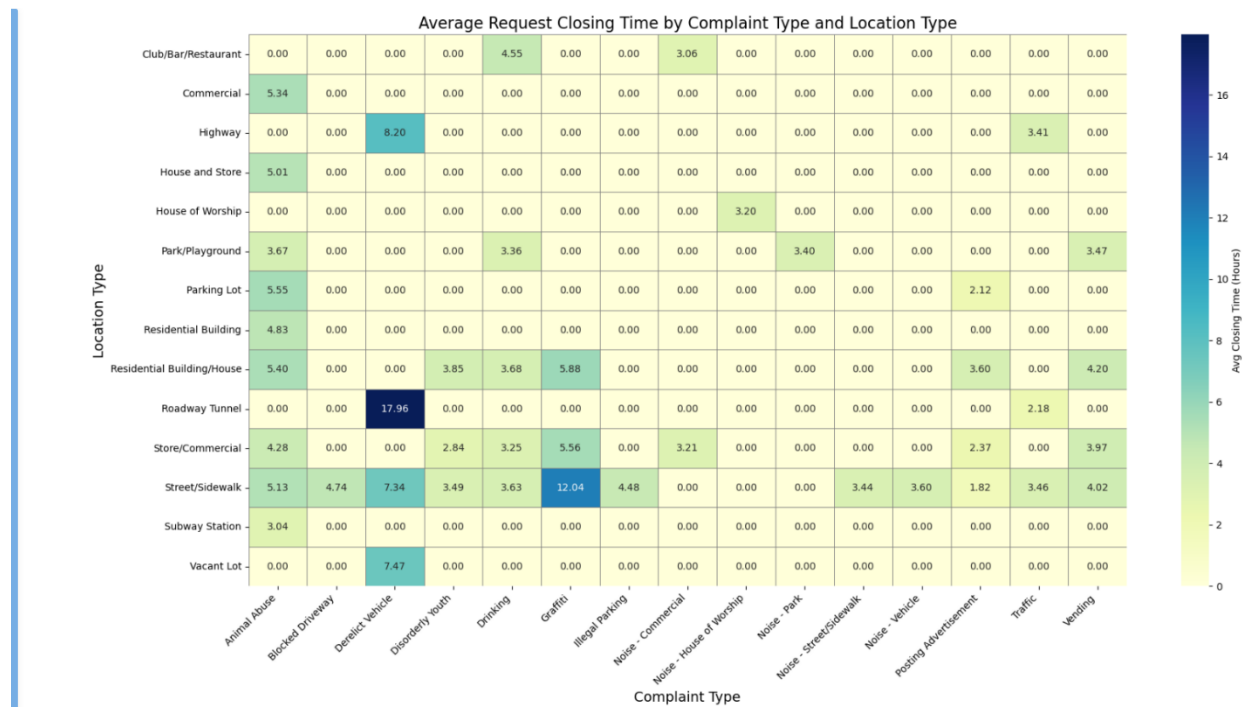


Figure 51: heat map of average response time between complaint types and location types

From the above heat map between location types, complaint types and average response time, we can observe that the complaint types Derelict Vehicle in Roadway Tunnel location had the highest average response time. The zeros in the heat map states that it is unlikely that certain complaint types can be reported though certain location types. For example, Blocked Driveway complaint can never be or is never reported from a Club/Bar/Restaurant. The Posting Advertisement complaint around Streets/Sidewalks had the quickest average response time in comparison to other complaint received in the location or the complaint had least average response time in Street/Sidewalk location type in comparison to other locations.

## STATISTICAL TESTING

### 5.1. Test 1: Similarity of average response time across complaint types

For the test of similarity of average response time across complaint types, a one-way ANOVA (Analysis of Variance) test was performed. It is done in the case when mean across multiple categories are to be tested for similarity or differences (Cuevas, et al., 2004). This test helps determine if the mean closing times vary significantly between the various complaint types. The hypothesis for the Test 1 is as following:

**Null Hypothesis, H0:** Average request closing times are similar across complaint types

**Alternate Hypothesis, H1:** Average request closing times are not similar across complaint types

Below is the process involved during the Statistical testing for test 1: checking the similarity of mean response time across all complaint types:

- Since, one way- ANOVA test is suitable for the scenario the `f_oneway()` function was imported from `scipy.stats` library.
- The average response times of each complain type was extracted by grouping of complaint types in dataframe `request_data` and puts them into a list by the use of `.apply(list)` function. The list was stored to `grouped_data`.
- The `f_oneway(*grouped_data)` called the required test algorithm for one way – ANOVA test and returned `f_stat` and `p_val`.
- If the `p_val` was lesser than 0.05 the null hypothesis would be rejected or else wouldn't be rejected.

## ▼ Statistical Testing

One way - ANOVA test to check whether the average response time across complaint types is similar or not

Importing f\_oneway from scipy.stats

```
[85]: from scipy.stats import f_oneway
```

Null Hypothesis, H0: Average request closing times, i.e. response times, are similar across the complaint types

```
[87]: grouped_data = request_data.groupby('Complaint Type')['Request_Closing_Time'].apply(list)

f_stat, p_val = f_oneway(*grouped_data)

print(f"F-statistic: {f_stat}")
print(f"P-value: {p_val}")
if p_val < 0.05:
    print("Reject H0: At least one of the complaint types has a different average request closing time.")
else:
    print("Failed to reject H0: Average request closing times are similar across complaint types.")
```

F-statistic: 578.9120337398356

P-value: 0.0

Reject H0: At least one of the complaint types has a different average request closing time.

```
[ ]:
```

Figure 52: Statistical test 1, one way - ANOVA test, for similar mean response time across complaint types

From the one way – ANOVA test conducted for the similarity of average response time across the complaint types, we get the f\_statistic value and the p\_value.

### f\_statistic value

The f\_statistic value signifies the ratio of the variation of means of the different groups to the variation of individual values within each group.

i.e., Mathematically,

$$f\_stat = \frac{\text{variance between different complaint types}}{\text{variance within a single complaint type}}$$

The f\_statistic value informs if the groups/columns are vastly different from each other or similar to each other. The bigger the f\_statistic value, the group means are different in comparison to other group means.

In the Statistical Test 1 done using the one way – ANOVA test, the f\_statistic value is calculated to be 578.9120337398356 which is a very high f\_statistic value which suggests that the average response times across the complaint types are significantly different from each other.

### p-value



The p-value defines the chances of difference or variation being random. It determines whether the observed differences are statistically significant or not. The p-value tells us the chance of getting an  $f\_statistic$  value as high as the above calculated value if average response time across the complaint types were the same.

Mathematically,

$$p - value = P(F \geq F_{calculated})$$

Where,  $P()$  is probability value observed from the  $f\_statistic$  value's probability distribution table,  $F$  is the variable for checking the  $F_{calculated}$  in the  $F$  distribution table.

In the Statistical Test 1 done using one way – ANOVA test, the p-value is calculated to be extremely small, i.e. close to 0, which rejects the stated null hypothesis  $H_0$ . This states that at least one of the complaint types has a different average response time compared to the average response time of the other complaint types.

## 5.2. Test 2: Whether the complaint types and location types are related

For the test of relation or association between complaint types and location types, a Chi-square test of independence was performed. This test is suitable when the statistically significant relationship between 2 categorical variables is to be determined (West & Kempthorne, 1972). In this case, the objective of the test was to assess whether the type of complaint is related with the type of location it was reported from. The hypothesis for the Test 2 is as following:

**Null Hypothesis,  $H_0$ :** Complaint type and Location type are not related.

**Alternate Hypothesis,  $H_1$ :** Complaint type and Location are related.

Below is the process involved during the statistical testing for Test 2: checking the relation between complaint types and location types:

- Since, chi-squared test is suitable for the scenario the `chi2_contingency()` function was imported from `scipy.stats` library.
- A contingency table was created using `pd.crosstab()` function to count the frequency of each combination between complaint types and location types which

was stored to contingency\_table. The contingency table shows how often each complain type occurs at each location type (Simpson, 1951).

- The chi2\_contingency() function was applied to the contingency table as chi2\_contingency(contingency\_table) which returned chi\_sqaure statistic, p-value, degrees of freedom and expected frequency table, which was stored in 4 variables chi2, p, dof and expected.
- If the p\_val was lesser than 0.05 the null hypothesis would be rejected or else wouldn't be rejected.

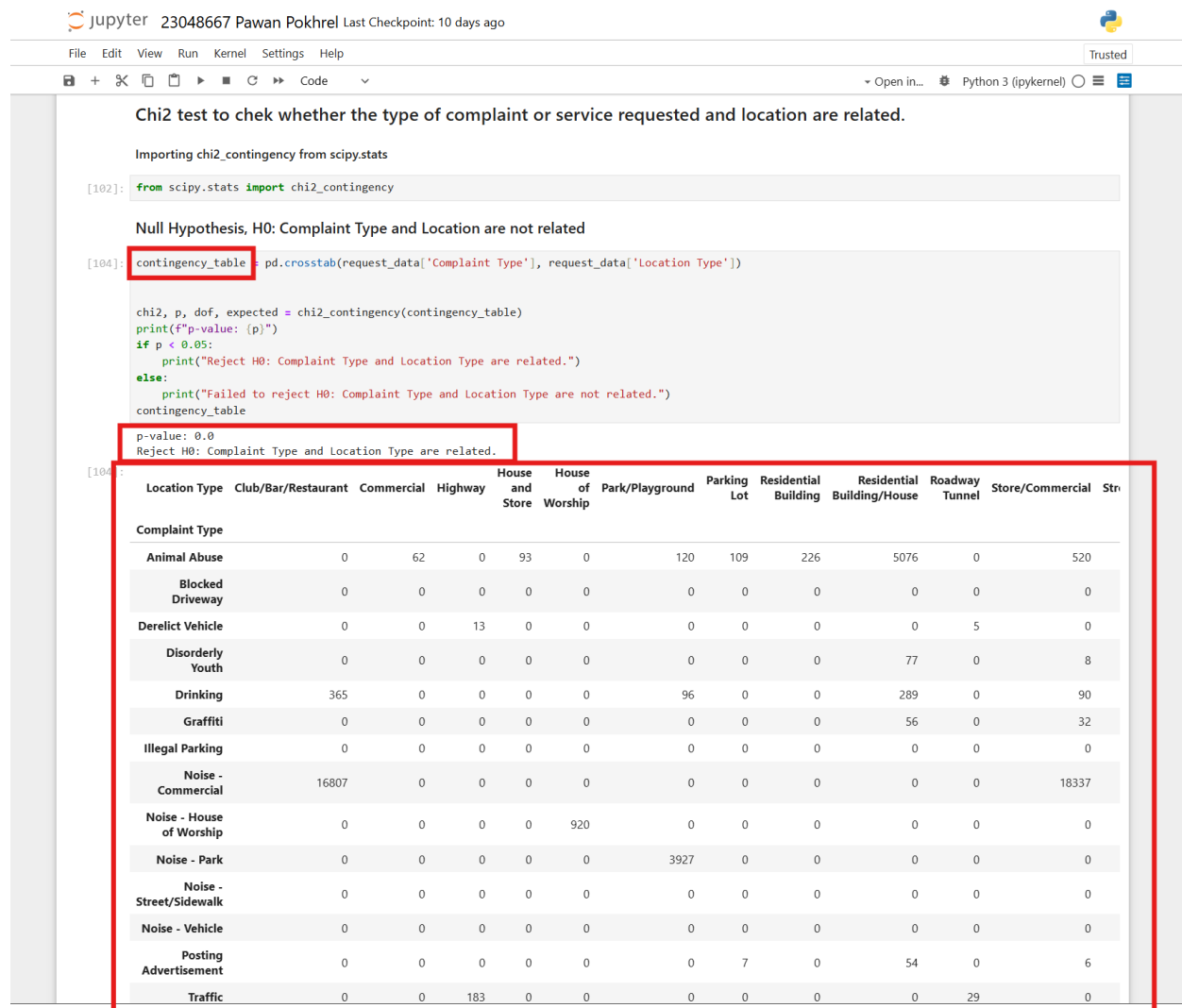


Figure 53: Statistical Test 2, chi-square contingency test, relation between complaint types and location types

In Statistical Test 2 done using chi-square contingency test, the p-value is calculated to be extremely small, i.e. close to 0, which rejects the null hypothesis  $H_0$ . This states that the complaint types and the location typed are related and a significant change in either of the value can change the other value accordingly.

## REFERENCES

Cuevas, A., Febrero, M. & Fraiman, R., 2004. An anova test for functional data. *Computational Statistics & Data Analysis*, Volume 47, pp. 111-122.

Simpson, E., 1951. The Interpretation of Interaction in Contingency Tables. *Journal of the Royal Statistical Society: Series B (Methodological)*, 13(2), pp. 238-241.

West, E. N. & Kempthorne, O., 1972. A comparison of the chi<sup>2</sup> and likelihood ratio tests for composite alternatives. *Journal of Statistical Computation and Simulation*, Volume 1, pp. 1-33.