



 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: Sujal Parajuli**

**London Met ID: 23050262**

**College ID: np01cp4a230257**

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

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

**Submitted To: Dipeshwor 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 report:

## 23050262 Sujal Parajuli 10.docx

 Islington College,Nepal

### Document Details

Submission ID

trncoi::3618:95921702

Submission Date

May 15, 2025, 3:03 AM GMT+5:45

Download Date

May 15, 2025, 3:05 AM GMT+5:45

File Name

23050262 Sujal Parajuli 10.docx

File Size

37.2 KB

48 Pages

5,546 Words

31,771 Characters



Page 1 of 54 - Cover Page

Submission ID trncoi::3618:95921702







Page 2 of 54 - Integrity Overview

Submission ID trncoi::3618:95921702




## 19% Overall Similarity

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

### Match Groups

-  **76 Not Cited or Quoted 18%**  
Matches with neither in-text citation nor quotation marks
-  **5 Missing Quotations 1%**  
Matches that are still very similar to source material
-  **2 Missing Citation 0%**  
Matches that have quotation marks, but no in-text citation
-  **0 Cited and Quoted 0%**  
Matches with in-text citation present, but no quotation marks

### Top Sources

- 10%  Internet sources
- 2%  Publications
- 18%  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:

<b>1. Data Understanding:</b>	<b>1</b>
1.1. Dataset Overview - Tabular Format:	2
<b>2. Data Preparation:</b>	<b>8</b>
I. Import the Dataset:	8
II. Importing the Libraries:	9
III. Provide your insight on the information and details that the provide	9
Error encountered after the import:	10
___Insight on the information and details that the provided dataset carries:	11
IV. Convert the columns "Created Date" and "Closed Date" .	12
Displaying the datatypes after the conversion into date type format:	13
V. Write a python program to drop irrelevant Columns which are listed below. ....	15
Dropping the irrelevant columns in the data frame:	15
VI. Write a python program to remove the NaN missing values	17
Finally Removing the rows with NaN values in the data frame:	17
VII. Write a python program to see the unique values from all the columns	18
<b>3. Data Analysis:</b>	<b>19</b>
Question: Write a Python program to show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame. ....	19
For Sum:	20
For Mean:	21
For Standard Deviation:	21
For Skewness:	22
For Kurtosis:	23
VIII. Write a Python program to calculate and show correlation of all variables.....	23
Finally calculating and showing the correlation of all variables	23

<b>4. Data Exploration:</b>	<b>25</b>
4.1. Question: Provide four major insights through visualization that you come up after data mining.	26
1) Insight 1: Top 10 Complaint Types	26
2) Insight 2: Complaints by Borough	27
3) Insight 3: Average Resolution Time Analysis	29
4) Insight 4: Monthly Complaint Trends	31
4.2. Arrange the complaint types according to their average 'Request_Closing_Time', categorized by various locations. Illustrate it through graph as well.	33
<b>5. Statistical Analysis:</b>	<b>36</b>
Test 1: ANOVA for Resolution Times	37
Test 2: Chi-Square Test for Complaint Type vs Borough	38
<b>6. Conclusion of Report:</b>	<b>40</b>
<b>7. References:</b>	<b>41</b>

## Table of Figures:

Figure 1 Dataset Import.....	8
Figure 2 library Import .....	9
Figure 3 Reading the CSV file.....	9
Figure 4 Error warning 1 after the import.....	10
Figure 5 Error Correction 1 .....	10
Figure 6 Converting various string representing Data .....	12
Figure 7 Converting to datetime datatype .....	13
Figure 8 Creating new Column names Request_Closing_Time to store data .....	13
Figure 9 Displaying the first few rows of the DataFrame .....	14
Figure 10 Removing irrelevant columns.....	15
Figure 11 Counting the number of missing values (NaN or None) .....	17
Figure 12 Removing rows with NaN values.....	17
Figure 13 Displaying the count of unique values in each column.....	18
Figure 14 Summary stats .....	19
Figure 15 Data Analysis for calculating Sum .....	20
Figure 16 Data Analysis for calculating Mean .....	21
Figure 17 Data Analysis for calculating Standard Deviation.....	21
Figure 18 Data Analysis for calculating Skewness .....	22
Figure 19 Data Analysis for calculating Kurtosis .....	23
Figure 20 calculating the correlation of all variables.....	23
Figure 21 Correlation output.....	24
Figure 22 code explanation 1 .....	26
Figure 23 Plotted Graph 1 .....	27
Figure 24 Code Explanation 2.....	27
Figure 25 Plotted graph 2.....	28
Figure 26 code of Average resolution time analysis .....	29
Figure 27 Plotted graph of top 10 types by average closing time.....	30
Figure 28 Code Explanation 4.....	31
Figure 29 Visualization 4 .....	32
Figure 30 code for Arrange the complaint types according to their average 'Request_Closing_Time'.....	33
Figure 31 Output of Arrange the complaint types according to their average 'Request_Closing_Time'.....	34
Figure 32 Visualization of average request closing time .....	35
Figure 33 Test 1: ANOVA for Resolution Times .....	37
Figure 34 Test 2: Chi-Square Test for Complaint Type vs Borough.....	38

## Table of Table:

Table 1 Dataset Overview .....	7
Table 2 List of Library used .....	25

## 1. Data Understanding:

This data contains service requests made to New York City through the 311 system, which citizens use to report everything from noise, illegal parking, to sanitation problems and building upkeep. It is an immediate pipeline between agencies and citizens, ensuring issues are registered and addressed. Every entry contains information such as request date, complaint type, location (borough, ZIP code, exact coordinates), and assigned department. Some also state if the issue was resolved or remained open.

Analysing this information offers trends in urban living conditions and government responsiveness. For instance, it tells us which neighbourhoods are most affected, how long the repairs take, and which boroughs are most in need of attention. These results help improve city planning, resource distribution, and service efficiency—leading to better public satisfaction.

In this coursework, I analysed this dataset using Python. As an individual task, I did everything: cleaning irrelevant data's, fixing errors, and preparing it for analysis. My target was to uncover trends in complaints and evaluate NYC's service performance. Through this, I learned how data can identify inefficiencies and guide smarter decisions for cities.

With this data set, I also came to appreciate how data helps connect real-life problems to tangible solutions. Every grievance is an issue of an individual, and when we look at multiple similar grievances within a place, it shows us where the city must pay extra attention. Telling raw data in sharp visualizations and overviews, I was able to make the data easier to understand and useful to make decisions with. This experience proved to me the true value of data analysis in making cities habitable and responsive to the needs of humans.

**1.1. Dataset Overview - Tabular Format:**

S. N	COLUMN NAME	DESCRIPTION	DATA TYPE
1	Unique Key	Each complaint in the dataset is assigned a unique identification number known as the Unique Key. The number renders each record distinct, making follow-up and reference to a particular complaint easy without confusion even if other details are similar.	INT
2	Created Date	This field records the date and time when a resident initially logged their service request in the 311 system. It enables issues to be monitored for how long they remain to be resolved by logging when they entered the workflow of the city.	OBJECT
3	Closed Date	When the complaint is closed or otherwise resolved by the agency responsible. Comparing it with the Created Date provides the response time of the city for different issues.	OBJECT
4	Agency	The Agency field specifies which NYC government agency (e.g., NYPD, Department of Sanitation) is responsible for acting on the complaint. This provides accountability and enables determination of which agencies get the most requests.	OBJECT
5	Agency Name	Full name of the government or service agency who are tasked with solving the issue.	OBJECT



6	Complaint Type	This categorizes the general kind of problem that was complained about, i.e., "Noise Complaint" or "Pothole." It allows for high-level examination of which problems are most prevalent citywide.	OBJECT
7	Descriptor	More precise than the Complaint Type, the Descriptor provides detailed information about the problem, i.e., "Loud Music at Night" under Noise Complaints. This level of specificity works to find frequent or niche problems.	OBJECT
8	Location Type	This is where the complaint occurred, such as "Apartment Building" or "Public Park." It helps to identify whether specific locations are prone to specific issues.	OBJECT
9	Incident Zip	The ZIP code where the problem was reported, geo-locating the area to a specific neighborhood. This is used to identify geographic trends or hot spots for complaints.	FLOAT
10	Incident Address	Complete address of the street in which the complaint was originated.	OBJECT
11	Street Name	The street name where the issue occurred.	OBJECT
12	Cross Street 1	The first intersecting cross street to the incident.	OBJECT
13	Cross Street 2	Another nearby cross street which is close to the complaint location.	OBJECT
14	Intersection Street 1	Street which directly intersects with the main incident street.	OBJECT

15	Intersection Street 2	Another additional street which also intersects, or which connects near the location.	OBJECT
16	Address Type	This describes if the address is a block, intersection, or specific building.	OBJECT
17	City	The city or town in which the complaint was officially reported.	OBJECT
18	Landmark	A well-known location or landmark near the complaint area.	OBJECT
19	Facility Type	Nature of the facility where the problem happened (e.g., park, school).	OBJECT
20	Status	Indicates the current state of the complaint—whether it's "Open," "In Progress," or "Closed." This helps assess the city's efficiency in resolving issues and identifies bottlenecks in the process.	OBJECT
21	Due Date	Expected problem resolution date	DATETIME
22	Resolution Action Updated Date	Date and time of the most recent update on the case's resolution.	DATETIME
23	Community Board	Name of the local community board who are responsible for the area.	OBJECT
24	BBL	A unique ID representing the parcel or property in question.	OBJECT
25	Borough	Borough or administrative district in which the complaint occurred.	OBJECT
26	X Coordinate (map plotting)	Geographic X-coordinate for mapping the incident's location.	FLOAT64
27	Y Coordinate (map plotting)	Geographic Y-coordinate for mapping the location on a map.	FLOAT64

28	Open Data Channel Type	Method which was used to submit the request, such as phone, mobile app, or web.	OBJECT
29	Park Facility Name	Name of park facility	OBJECT
30	Park Borough	Borough in which the related park is located.	OBJECT
31	Vehicle Type	Classification of vehicle, if the issue involves transportation.	OBJECT
32	Taxi Company Borough	The operating borough of the taxi service which is related to the issue.	OBJECT
33	Taxi Pick Up Location	Taxi pickup location	OBJECT
34	Bridge Highway Name	Identifies the bridge or highway mentioned in the complaint.	OBJECT
35	Bridge Highway Direction	Indicates the direction related to the incident.	OBJECT
36	Road Ramp	Details about entry or exit ramps related to the issue.	OBJECT
37	Bridge Highway Segment	Specific part of the bridge or road involved.	OBJECT
38	Latitude	A geographic coordinate (north-south position) that, when combined with Longitude, plots the complaint's exact location on a map. Essential for spatial analysis and visualizing complaint clusters	FLOAT

39	Longitude	A geographic coordinate (north-south location) that, when used in combination with Longitude, graphically locates the complaint on a map. Necessary for spatial analysis and displaying complaint clusters.	FLOAT
40	Location	Combined latitude and longitude	OBJECT
41	School Name	Full name of the school mentioned in the complaint.	OBJECT
42	School Number	Numeric identifier of the school which is its ID.	OBJECT
43	School Region	Code representing the education region in which school belongs to.	OBJECT
44	School Code	Unique code that identifies a school.	OBJECT
45	School Phone Number	Contact number listed for the school which is mentioned.	OBJECT
46	School Address	Complete address of the school including its street, city, and ZIP.	OBJECT
47	School City	Name of the city where the school operates.	OBJECT
48	School State	State in which the school is situated.	OBJECT
49	School Zip	Postal code of the school's location.	OBJECT
50	School Not Found	A marker which indicates whether the data for the school is incomplete or missing.	OBJECT
51	School or Citywide Complaint	It shows whether the issue is local across the city.	OBJECT

52	Vehicle License Plate	License plate number of the involved vehicle.	OBJECT
53	Resolution Description	Description of how the issue was addressed and how it was resolved.	OBJECT

*Table 1 Dataset Overview*

### **Table Description also my findings on the given data set:**

The tables give information on the various columns available in the 311-service request dataset. Each row in the table describes a varying column, including the column name, what it is, and what kind of data it holds (e.g., text, numbers, or dates). For instance, columns showing "Created Date" and "Closed Date" show when a complaint was filed and when it was closed. "Complaint Type" and "Descriptor" describe the kind of problem that was complained about, such as noise, sanitation, or street damage.

There are location-based columns like "Incident Address", "City", "ZIP code", and "Latitude/Longitude" that tell us where the complaint was made. There are also explanations of surrounding landmarks or streets, specific down to the problem's location. Some columns tell us which government agency the complaint was handled by and how they closed it. Some other fields, like "Facility Type" or "School Name", give us more information about specific locations like schools or parks.

This summary table helps us to understand what data is included in the dataset, how the data is structured, and how it can be used to analyse complaints, spot trends, and help make better-informed decisions for the city.

## 2. Data Preparation:

Data Preparation is the process of cleaning, transforming, and organizing raw data into a format that is analysable, machine learnable, or reportable. Data Preparation ensures data quality, consistency, and usability before its ingestion into models or visualizations.

### I. Import the Dataset:

A dataset is a structured collection of data, typically in tabular form (e.g., spreadsheet or database table), in which:

- Rows are individual records (e.g., complaints from customers, transactions in sales).
- Columns are characteristics or attributes (e.g., date, place, type of complaint).

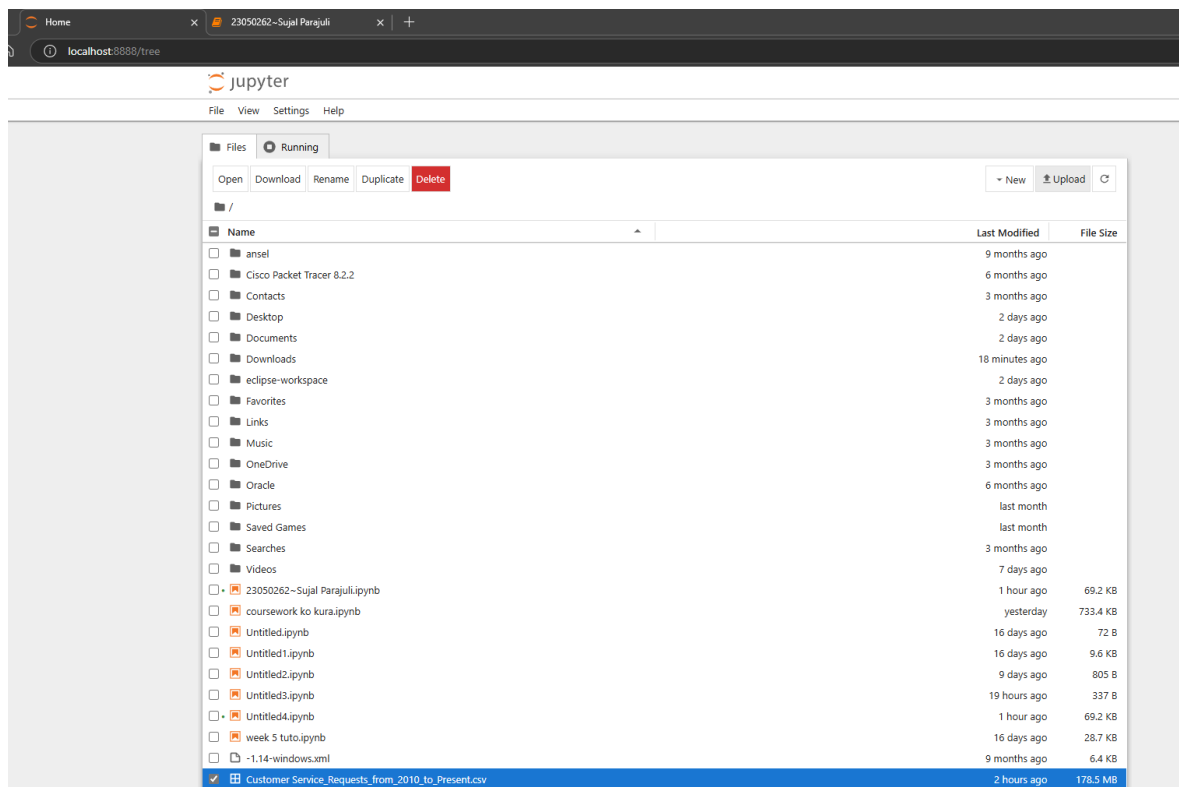


Figure 1 Dataset Import

- **Ans.** Importing the provided Dataset through the files section inside the jupyter notebook library by clicking the upload button in the top right corner.

## II. Importing the Libraries:

```
#2050262 Sujal Parajuli (C10)
#importing panda and numpy library by giving it a shorter name as 'pd' and 'np'
#panda library will help us organize and look at the datas whereas numpy will help us do calculations.
"""
2. DATA PREPARATION
"""
import pandas as pd
import numpy as np
```

Figure 2 library Import

- These are the two biggest libraries that are utilized to perform data analysis. Pandas as pd is utilized to process and analyse structured data while numpy has no support for numerical operations mainly focuses on math functions and arrays.

## III. Provide your insight on the information and details that the provided dataset carries:

- Ans. To provide an insight firstly we need to read the CSV file the hold the data so, reading the CSV file into a pandas Data frame:

```
[3]: #Reading the CSV file into a pandas Dataframe which is named as 'df' by using the 'read_csv' function from the pandas Library.
customer_requests_df = pd.read_csv("Customer Service_Requests_from_2010_to_Present.csv")

C:\Users\Nitro\AppData\Local\Temp\ipykernel_16612\3028810173.py:2: DtypeWarning: Columns (48,49) have mixed types. Specify dtype option on import or
set low_memory=False.
customer_requests_df = pd.read_csv("Customer Service_Requests_from_2010_to_Present.csv")
```

Figure 3 Reading the CSV file

We can use `pd.read_csv()` to load CSV files directly into Python as DataFrame. This function takes care of our data being in rows and columns for us and lets us control how the data is loaded.

## Error encountered after the import:

```
C:\Users\Nitro\AppData\Local\Temp\ipykernel_36064\3028810173.py:2: DtypeWarning: Columns (48,49) have mixed types. Specify dtype option on import or set low_memory=False.  
customer_requests_df = pd.read_csv("Customer Service_Requests_from_2010_to_Present.csv")
```

Figure 4 Error warning 1 after the import

- After importing the data frame, I was prompted with an error warning which was because The data file contains a combination of numbers and text in columns 48 and 49. Python was able to load the data without problems, but it's warning me: to prevent potential problems down the line, we ought to explicitly declare the data types for these columns. So, after some research I solved the issue with minor changes.

## Fixing the encountered error warning:

```
[ ]: #Reading the CSV file into a pandas Dataframe which is named as 'df' by using the 'read_csv' function from the pandas Library.  
customer_requests_df = pd.read_csv("Customer Service_Requests_from_2010_to_Present.csv", low_memory = False)  
  
[46]: #Displaying the dataframe  
customer_requests_df
```

Figure 5 Error Correction 1

- I added low\_memory = False. Previously, the error warning occurred because, by default (low\_memory=True), pandas check a tiny sample of rows to make an informed guess at each column's data type (e.g., numbers, text, or dates). And by any chance while further analysis the rows contain different types (e.g. text in a column that originally contained numbers), panda crashes the column to a wasteful "object" dtype. This little precaution makes sure all goes well when we are working with the data afterwards.



## Insight on the information and details that the provided dataset carries:

```
[57]: #Displaying the dataframe
customer_requests_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
20  Due Date                                300695 non-null object
21  Resolution Description                  300698 non-null object
22  Resolution Action Updated Date          298511 non-null object
23  Community Board                         300698 non-null object
24  Borough                                300698 non-null object
25  X Coordinate (State Plane)              297158 non-null float64
26  Y Coordinate (State Plane)              297158 non-null float64
27  Park Facility Name                      300698 non-null object
28  Park Borough                            300698 non-null object
29  School Name                             300698 non-null object
30  School Number                           300698 non-null object
31  School Region                           300697 non-null object
32  School Code                             300697 non-null object
33  School Phone Number                     300698 non-null object
34  School Address                          300698 non-null object
35  School City                             300698 non-null object
36  School State                            300698 non-null object
37  School Zip                              300697 non-null object
38  School Not Found                        300698 non-null object
39  School or Citywide Complaint            0 non-null float64
40  Vehicle Type                            0 non-null float64
41  Taxi Company Borough                    0 non-null float64
42  Taxi Pick Up Location                    0 non-null float64
43  Bridge Highway Name                     243 non-null object
44  Bridge Highway Direction                 243 non-null object
45  Road Ramp                               213 non-null object
46  Bridge Highway Segment                   213 non-null object
47  Garage Lot Name                          0 non-null float64
48  Ferry Direction                          1 non-null object
49  Ferry Terminal Name                      2 non-null object
50  Latitude                                 297158 non-null float64
51  Longitude                                297158 non-null float64
52  Location                                 297158 non-null object
dtypes: float64(10), int64(1), object(42)
memory usage: 121.6+ MB
```

- The Data Frame contains information about customer requests with details on when they were created and closed, the agency involved, the type and description of the complaint, location information, and more.

**Output Explanation:**

- First few rows of the “customer\_requests\_df” DataFrame are displayed.
- Multiple database columns named ‘Unique Key’, ‘Created Date’, ‘Closed Date’, ‘Agency’, ‘Complaint Type’, ‘Descriptor’, ‘Location Type’, ‘Incident Zip’, and ‘Incident Address’ along with others appear within this record view.
- The DataFrame contains information about customer requests with details on when they were created and closed, the agency involved, the type and description of the complaint, location information, and more.
- The ‘NaN’ values indicate missing data in certain fields for those specific entries.

**IV. 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.**

Ans. Converting various string representations of dates and times of 'Created Date' and 'Closed Date' columns into pandas datetime objects.

```
[4]: # converting various string representations of dates and times of 'Created Date' and 'Closed Date' columns into pandas datetime objects.
customer_requests_df['Created Date'] = pd.to_datetime(customer_requests_df['Created Date'])
customer_requests_df['Closed Date'] = pd.to_datetime(customer_requests_df['Closed Date'])
#conversion is done by 'pd.to_datetime()' function from pandas library.

C:\Users\Nitro\AppData\Local\Temp\ipykernel_36408\659762902.py:2: UserWarning: Could not infer format, so each element will be parsed individually, falling back to 'dateutil'. To ensure parsing is consistent and as-expected, please specify a format.
  customer_requests_df['Created Date'] = pd.to_datetime(customer_requests_df['Created Date'])
C:\Users\Nitro\AppData\Local\Temp\ipykernel_36408\659762902.py:3: UserWarning: Could not infer format, so each element will be parsed individually, falling back to 'dateutil'. To ensure parsing is consistent and as-expected, please specify a format.
  customer_requests_df['Closed Date'] = pd.to_datetime(customer_requests_df['Closed Date'])
```

*Figure 6 Converting various string representing Data*

## Displaying the datatypes after the conversion into date type format:

```

]: #Verifying if the data columns have been correctly converted to datetime datatype
#dtypes function of panda returns the data type of each column in the Dataframe
customer_requests_df.dtypes

]: Unique Key                                int64
   Created Date                             datetime64[ns]
   Closed Date                             datetime64[ns]
   Agency                                  object
   Agency Name                             object
   Complaint Type                           object
   Descriptor                               object
   Location Type                            object
   Incident Zip                             float64
   Incident Address                         object
   Street Name                             object
   Cross Street 1                          object
   Cross Street 2                          object
   Intersection Street 1                   object
   Intersection Street 2                   object
   Address Type                             object
   City                                    object
   Landmark                                object
   Facility Type                           object
   Status                                  object
   Due Date                                object
   Resolution Description                   object
   Resolution Action Updated Date           object
   Community Board                         object
   Borough                                 object
   X Coordinate (State Plane)               float64
   Y Coordinate (State Plane)               float64
   Park Facility Name                      object
   Park Borough                            object
   School Name                             object
   School Number                           object
   School Region                           object
   School Code                             object
   School Phone Number                     object
   School Address                          object
   School City                             object
   School State                            object
   School Zip                              object
   School Not Found                        object
   School or Citywide Complaint             float64
   Vehicle Type                            float64
   Taxi Company Borough                    float64
   Taxi Pick Up Location                   float64
   Bridge Highway Name                     object
   Bridge Highway Direction                 object
   Road Ramp                              object
   Bridge Highway Segment                  object
   Garage Lot Name                         float64
   Ferry Direction                         object
   Ferry Terminal Name                     object
   Latitude                                float64
   Longitude                               float64
   Location                                object
   dtype: object

'''
This code figures out how long it took to close each request by subtracting
the creation date from the closing date and saves it in a new column called 'request_closing_time'.
It was possible only after the conversion of data into datatype format.
'''
customer_requests_df['Request_Closing_Time'] = customer_requests_df['Closed Date'] - customer_requests_df['Created Date']

```

Figure 7 Converting to datetime datatype

## Creating a new column to hold the data:

```

'''
This code figures out how long it took to close each request by subtracting
the creation date from the closing date and saves it in a new column called 'request_closing_time'.
It was possible only after the conversion of data into datatype format.
'''

customer_requests_df['Request_Closing_Time'] = customer_requests_df['Closed Date'] - customer_requests_df['Created Date']

```

Figure 8 Creating new Column names Request\_Closing\_Time to store data

Displaying the first 5 rows of the data frame to verify that the pervious code was implemented correctly or not:

```

: # Displaying the first few rows of the DataFrame using the .head() method from pandas Library.
customer_requests_df[['Created Date', 'Closed Date', 'Request_Closing_Time']]
# .head() method shows the first 5 rows of the dataframe by default.
customer_requests_df.head()
# It is a quick way to inspect the data

```

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incident Address	...	Bridge Highway Direction	Road Ramp	Bridge Highway Segment	Garage Lot Name	Ferr Directio
0	32310363	2015-12-31 23:59:45	2016-01-01 00:55:00	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidewalk	10034.0	71 VERMILYEA AVENUE	...	NaN	NaN	NaN	NaN	NaN
1	32309934	2015-12-31 23:59:44	2016-01-01 01:26:00	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	11105.0	27-07 23 AVENUE	...	NaN	NaN	NaN	NaN	NaN
2	32309159	2015-12-31 23:59:29	2016-01-01 04:51:00	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidewalk	10458.0	2897 VALENTINE AVENUE	...	NaN	NaN	NaN	NaN	NaN
3	32305098	2015-12-31 23:57:46	2016-01-01 07:43:00	NYPD	New York City Police Department	Illegal Parking	Commercial Overnight Parking	Street/Sidewalk	10461.0	2940 BALSLEY AVENUE	...	NaN	NaN	NaN	NaN	NaN
4	32306529	2015-12-31 23:56:58	2016-01-01 03:24:00	NYPD	New York City Police Department	Illegal Parking	Blocked Sidewalk	Street/Sidewalk	11373.0	87-14 57 ROAD	...	NaN	NaN	NaN	NaN	NaN

5 rows x 17 columns

Figure 9 Displaying the first few rows of the DataFrame

### Output Explanation:

- The code adds a new column named 'Request\_Closing\_Time' to the customer\_requests\_df DataFrame.
- The values in this new column are calculated by subtracting the values in the 'Created Date' column from the corresponding values in the 'Closed Date' column.

This operation determines the duration taken to close each customer request.

V. 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'].*

**Ans.** To drop the irrelevant columns from the data set we need to add .drop function with the data frame name i.e in our case customer\_request\_df.

**Dropping the irrelevant columns in the data frame:**

```
# Removing irrelevant columns using the .drop() method. We are removing several columns that are not needed for our analysis.
customer_requests_df = customer_requests_df.drop(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'])
# Displaying information about the DataFrame after dropping the irrelevant columns
customer_requests_df.info()
```

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

Figure 10 Removing irrelevant columns

**Output Explanation:****1. customer\_requests\_df.drop(columns=[.])**

- This line is using the `drop()` function to drop more than one column from the DataFrame `customer_requests_df`.

**2. Why did we drop these columns?**

- It is because these columns are irrelevant to the primary issue.

**Example**

Columns like 'Ferry Terminal Name', 'Garage Lot Name', or 'Taxi Pick Up Location' are irrelevant because we are dealing with complaint type patterns.

- It then uses the `drop ()` method of the Pandas DataFrame to remove these columns.
- Finally, `customer_requests_df.info()` is used to display a summary of the DataFrame after the columns have been dropped.

## VI. Write a python program to remove the NaN missing values from updated data frame.

**Ans.** To remove the NaN missing values from the updated data frame we need to first identify how many NaN missing values are there so, counting the number of missing values (NaN or None):

```

"""
Counting the number of missing values (NaN or None)
in each column of the 'customer_requests_df' DataFrame.
"""
customer_requests_df.isnull().sum()

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

# Removing rows with NaN values using the .dropna() method.
customer_requests_df = customer_requests_df.dropna()

```

Figure 11 Counting the number of missing values (NaN or None)

## Finally Removing the rows with NaN values in the data frame:

```

[89]: # Removing rows with NaN values using the .dropna() method.
customer_requests_df = customer_requests_df.dropna()
customer_requests_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
mean	3.130158e+07	2015-08-14 11:25:43.378747648	2015-08-14 15:44:15.511413248	10857.977349	40.725681	-73.925035	0 days 04:18:32.132665995
min	3.027948e+07	2015-03-29 00:33:01	2015-03-29 00:57:23	83.000000	40.499135	-74.254937	0 days 00:01:00
25%	3.079934e+07	2015-06-08 15:38:00	2015-06-08 21:25:00	10314.000000	40.668926	-73.970957	0 days 01:16:30
50%	3.130675e+07	2015-08-13 22:57:41	2015-08-14 02:50:57	11209.000000	40.717782	-73.930774	0 days 02:42:38
75%	3.179091e+07	2015-10-19 15:03:16.500000	2015-10-19 20:58:35.500000	11238.000000	40.782973	-73.875788	0 days 05:20:24
max	3.231065e+07	2015-12-31 23:59:45	2016-01-03 16:22:00	11697.000000	40.912869	-73.700760	24 days 16:52:22
std	5.753777e+05	NaN	NaN	580.280774	0.082411	0.078654	0 days 06:03:45.509089128

Figure 12 Removing rows with NaN values

**Output Explanation:**

- The code uses the `dropna()` method on the `customer_requests_df` DataFrame. This method removes all rows that contain at least one NaN (Not a Number) value across any of the columns.  
The resulting DataFrame, `customer_requests_df`, contains only rows where all the values are non-null.
- `customer_requests_df.describe()` provides a summary of the dropped DataFrame.

**VII. Write a python program to see the unique values from all the columns in the data frame.**

**Ans.** To see the unique values from all the columns in the data frame we need to add `.nunique()` function with the DataFrame name .

**Finally displaying the count of unique values in each column:**

```
# Displaying the count of unique values in each column using the .nunique() method of panda library.
customer_requests_df.nunique()
```

Unique Key	291187
Created Date	251978
Closed Date	231991
Agency	1
Complaint Type	15
Descriptor	41
Location Type	14
Incident Zip	288
City	53
Status	1
Resolution Description	12
Borough	5
Latitude	123813
Longitude	123112
Request_Closing_Time	47134
dtype: int64	

Figure 13 Displaying the count of unique values in each column

Here, the code `.nunique()` is a method in panda library which counts the number of unique values in a column or an entire DataFrame.

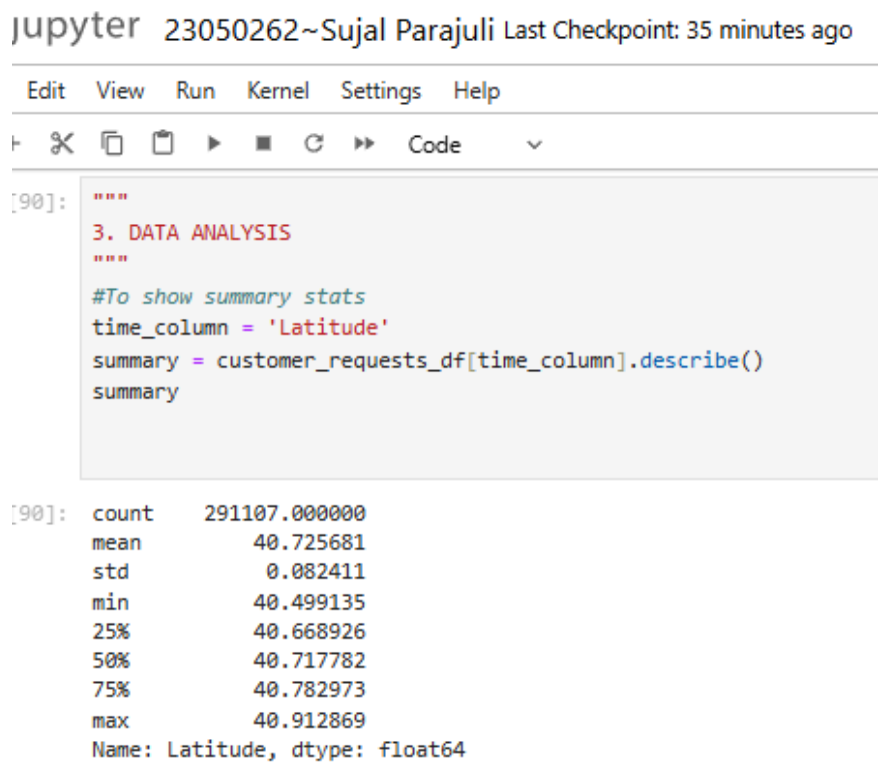


### 3. Data Analysis:

Data analysis involves examining, cleaning, transforming, and structuring data in order to uncover insights, make inferences, and facilitate decision-making. This is usually accomplished in Python with the help of libraries such as Pandas, NumPy, and Matplotlib/Seaborn for visualizations (Ivan, 2014).

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

**Ans.** To show summary statistics of sum, mean, standard deviation, skewness, and kurtosis of the data frame below is the code inserted.



The screenshot shows a Jupyter Notebook window titled "jupyter 23050262~Sujal Parajuli Last Checkpoint: 35 minutes ago". The interface includes a menu bar with "Edit", "View", "Run", "Kernel", "Settings", and "Help". Below the menu is a toolbar with icons for undo, redo, copy, paste, run, and other functions. The main area contains a code cell with the following Python code:

```
[90]: """
3. DATA ANALYSIS
"""

#To show summary stats
time_column = 'Latitude'
summary = customer_requests_df[time_column].describe()
summary
```

The output of the code cell is displayed below the code:

```
[90]: count    291107.000000
      mean      40.725681
      std       0.082411
      min      40.499135
      25%      40.668926
      50%      40.717782
      75%      40.782973
      max      40.912869
      Name: Latitude, dtype: float64
```

Figure 14 Summary stats

#### Output Explanation:

This program analyses geographics by focusing on latitude values from a customer request data set. It begins by assigning the string 'Latitude' to a variable called `time_column`, followed by calling pandas' `describe()` method on that specific column to produce descriptive statistics and saving these in a `summary` variable.

**For Sum:**

The total that comes out when all the values are added together is a sum.

```
"""  
3. DATA ANALYSIS  
"""  
  
# Calculating the sum  
Sum = customer_requests_df[time_column].sum()  
print(f"Total Sum of '{time_column}': {Sum}")
```

```
Total Sum of 'Latitude': 11855530.75877829
```

*Figure 15 Data Analysis for calculating Sum*

This function provides the total sum of every value found within the 'Latitude' column.

The addition of all 'Latitude' values gives us the summation for this particular column.

**Output:**

Total Sum of 'Latitude': 11855530.75877829

All latitude values within the dataset will yield a total of 11.8 million when summed.

**For Mean:**

Mean is an average value that we get by dividing the sum by the number of values.

```
# Calculating the Mean
Mean = customer_requests_df[time_column].mean()
print(f"Mean of '{time_column}': {Mean}")

Mean of 'Latitude': 40.72568079358549
```

Figure 16 Data Analysis for calculating Mean

This function computes the mean value from the Latitude column data points.

**Output:**

Mean of 'Latitude': 40.72568079358549

The average of all latitude values equals 40.73 and supports the NYC general area.

**For Standard Deviation:**

Standard deviation ( $\sigma$ ) indicates how spread the numbers in a data set are. A small number means that the data points are near the mean (average), while a big number means they're spread out more.

```
# Calculating the Standard Deviation
Std = customer_requests_df[time_column].std()
print(f"Standard Deviation of '{time_column}': {Std}")

Standard Deviation of 'Latitude': 0.08241087015112669
```

Figure 17 Data Analysis for calculating Standard Deviation

This function evaluates how the latitude data points extend relative to the mean value.

We use the standard deviation calculation to understand the data variability when we want to see the extent of variation.

**Output:**

Standard Deviation of 'Latitude': 0.08241087015112669

Most coordinate values display closeness to the average since they do not extend widely apart from it.

**For Skewness:**

Skewness shows whether provided data is symmetrical or it is tilted to one side.

```
# Calculating skewness
skewness = customer_requests_df[time_column].skew()
print(f"Skewness of '{time_column}': {skewness}")
```

Skewness of 'Latitude': 0.123114382634822

*Figure 18 Data Analysis for calculating Skewness*

This measure identifies which side of the average contains most value distribution. This statistical tool serves a useful purpose because measuring skewness helps identify data symmetry and side bias.

**Output:**

Skewness of 'Latitude': 0.123114382634822

The data shows symmetric distribution since skewness appears as 0.123114382634822.

**For Kurtosis:**

This measure determines the shape of the data distribution peak from pointy to flat.

The research value of kurtosis detection becomes apparent because we can identify the number of extreme values in the data set.

**Output:**

Kurtosis of 'Latitude': -0.7348182547719988

The data shape describes a flattened form which indicates few extreme values exist.

```
# Calculating kurtosis
kurtosis = customer_requests_df[time_column].kurt()
print(f"Kurtosis of '{time_column}': {kurtosis}")
```

Kurtosis of 'Latitude': -0.7348182547719988

Figure 19 Data Analysis for calculating Kurtosis

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

**Ans.** Correlation is an index of how strongly two variables move together. It tells us:

- Direction: Are they both moving up or does one increase as the other decreases.
- Strength: Is the relationship strong, weak, or non-existent (team, 2023).

To calculate and show correlation of all variables i.e sum, mean, standard deviation, skewness, and kurtosis of the data frame we need to calculate each of them, but we already did it in our previous question So, below is the calculation for the correlation of all variables:

**Finally calculating and showing the correlation of all variables**

```
#calculating and showing correlation of all variables
Co_rel = customer_requests_df.select_dtypes(include='number').corr()
print(Co_rel)
```

Figure 20 calculating the correlation of all variables.

**Output of correlation variables:**

```

: #calculating and showing correlation of all variables
Co_rel = customer_requests_df.select_dtypes(include='number').corr()
print(Co_rel)

```

	Unique Key	Incident Zip	Latitude	Longitude	Request_Closing_Time
Unique Key	1.000000	0.025492	-0.032613	-0.008621	0.053126
Incident Zip	0.025492	1.000000	-0.499081	0.385934	0.057182
Latitude	-0.032613	-0.499081	1.000000	0.368819	0.024497
Longitude	-0.008621	0.385934	0.368819	1.000000	0.109724
Request_Closing_Time	0.053126	0.057182	0.024497	0.109724	1.000000

*Figure 21 Correlation output***Code Explanation:**

The code calculates how strongly pairs of numerical variables in our dataset relate to each other, using a metric called correlation (ranging from -1 to 1). We used `.corr()` method to calculate it.

#### 4. Data Exploration:

Data exploration is the act of looking at and graphing the data to discover patterns, trends, and insights. In this coursework, we explore:

- Frequency distributions (e.g., types of complaints most frequently reported).
- Geospatial trends (e.g., by borough).
- Temporal trends (e.g., complaints by month).
- Performance metrics (e.g., mean request closing time) (Kabita Sahoo, 2019).

Library	Purpose	Where It Was Used
pandas	Data manipulation (filtering, grouping, aggregation).	All tasks (e.g., <code>groupby()</code> , <code>value_counts()</code> etc.
numpy	Numerical operations (e.g., mean, standard deviation).	Statistical calculations (e.g., ANOVA, correlation).
Matplotlib.pyplot	Data visualization (bar charts, line plots, histograms).	All plots (e.g., <code>plot.bar()</code> , <code>plot.line()</code> ).
scipy.stats	Hypothesis testing (ANOVA, Chi-Square).	Test 1 (ANOVA) and Test 2 (Chi-Square).
statsmodels	Post-hoc analysis (Tukey HSD test).	Identifying specific differences after ANOVA.

Table 2 List of Library used

**4.1. Question: Provide four major insights through visualization that you come up after data mining.**

### 1) Insight 1: Top 10 Complaint Types

**Objective:** Identify the most common service requests to prioritize resource allocation.

#### Methodology:

- Used value\_counts () to count occurrences of each complaint type
- Selected top 10 with head (10)
- Created a bar plot for visualization

#### Code-Explanation:

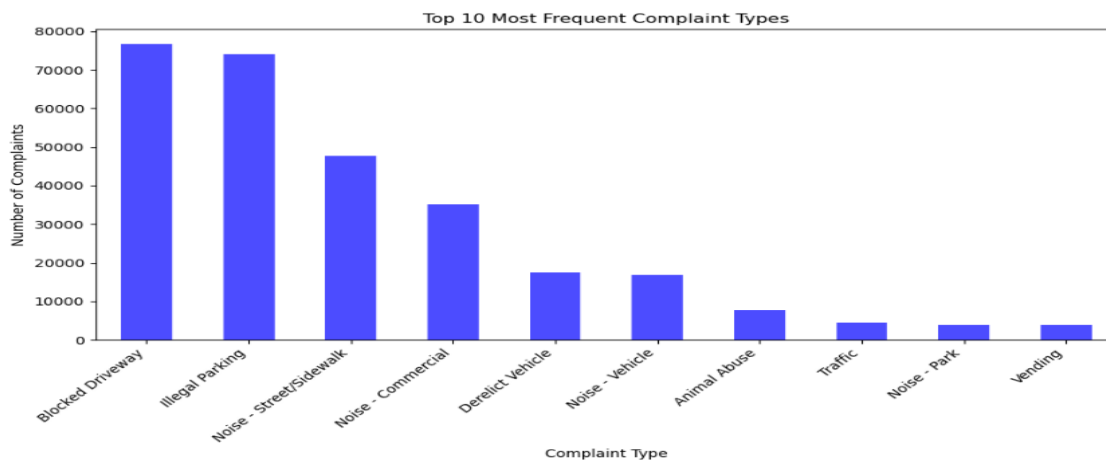
```
# --- Insight 1: Top 10 Complaint Types ---  
# Question: Provide four major insights through visualization.  
# Insight: Visualize the most frequent complaint types.  
  
# Get top 10 complaint types  
top_complaints = customer_requests_df['Complaint Type'].value_counts().head(10)  
  
# Plotting  
plot.figure(figsize=(10, 6))  
top_complaints.plot(kind='bar', color='blue', alpha=0.7)  
plot.title('Top 10 Most Frequent Complaint Types')  
plot.xlabel('Complaint Type')  
plot.ylabel('Number of Complaints')  
plot.xticks(rotation=45, ha='right')  
plot.tight_layout()  
plot.show()
```

Figure 22 code explanation 1

#### Key Findings:

- The bar graph clearly indicates that Blocked Driveway complaints are most common.
- Noise-Related complaints (Street/Sidewalk, Commercial, Vehicle) are 4 of the top 10.
- The pattern is a classic Pareto distribution where a few complaint categories represent the majority of cases



**Visualization:***Figure 23 Plotted Graph 1*

The plot uses a simple blue color scheme with grid lines for effortless estimation of values. The 45-degree rotated labels allow for all the complaint types to be read without overlapping.

**2) Insight 2: Complaints by Borough**

**Objective:** Understand geographic distribution of complaints across NYC boroughs.

**Methodology:**

- Aggregated data by borough using `value_counts()`
- Created a colored bar chart showing complaint volume by location

**Code-Explanation:**

```
# --- Insight 2: Complaints by Borough ---
# Insight: Distribution of complaints across NYC boroughs.

complaints_by_borough = customer_requests_df['Borough'].value_counts()

# Plotting graph
plot.figure(figsize=(10, 6))
complaints_by_borough.plot(kind='bar', color='green', alpha=0.7)
plot.title('Number of Complaints by Borough')
plot.xlabel('Borough')
plot.ylabel('Number of Complaints')
plot.xticks(rotation=45)
plot.tight_layout()
plot.show()
```

*Figure 24 Code Explanation 2*

**Key Findings:**

- Brooklyn has many more complaints than the other boroughs.
- Staten Island receives the least complaints.
- The distribution might reflect differences in population density or reporting behavior.

**Visualization:**

The chart uses a categorical colour scheme and has direct value labels to enable easy comparison across boroughs.

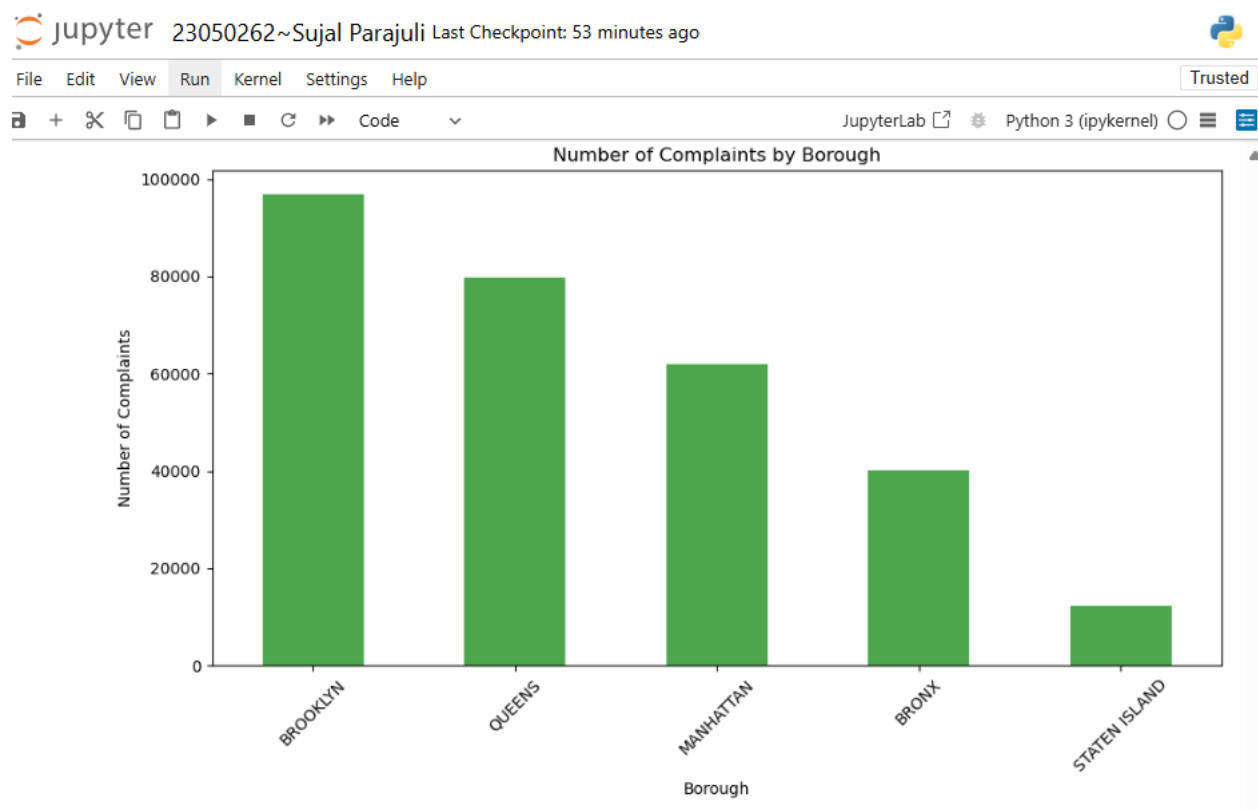


Figure 25 Plotted graph 2

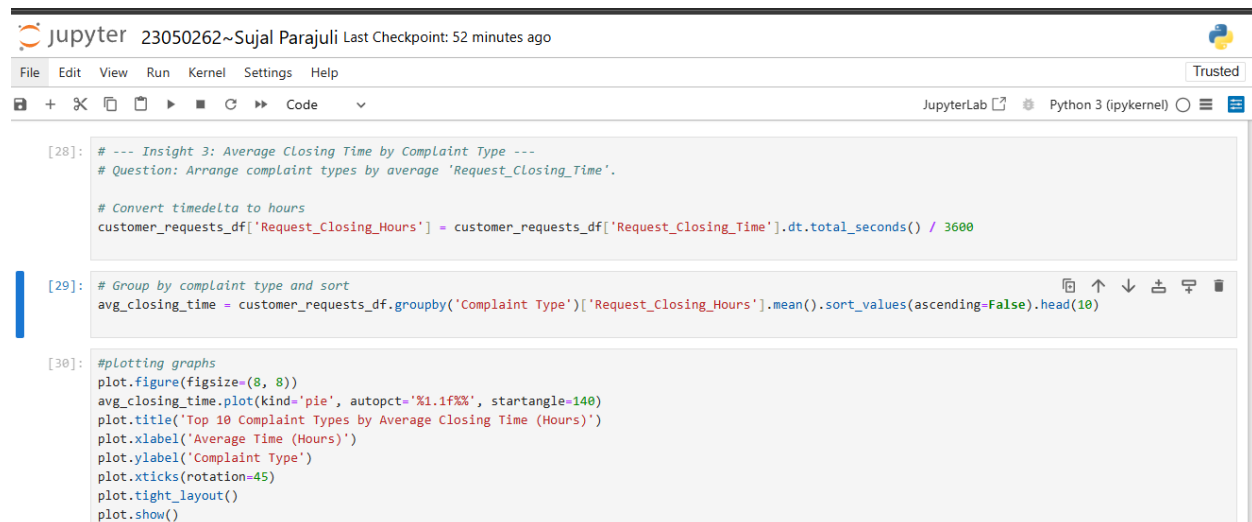
### 3) Insight 3: Average Resolution Time Analysis

**Objective:** Identify which complaint types take longest to resolve.

**Methodology:**

- Converted time delta to hours for easier interpretation
- Calculated mean resolution time by complaint type
- Sorted and visualized the top 10 slowest-to-resolve complaints

**Code-Explanation:**

The image shows a JupyterLab interface with a code editor. The top bar indicates the user is '23050262~Sujal Parajuli' and the last checkpoint was 52 minutes ago. The interface includes a menu bar (File, Edit, View, Run, Kernel, Settings, Help) and a toolbar with icons for file operations and code execution. The code editor contains three code cells. The first cell (index 28) contains comments and a line of code to convert 'Request\_Closing\_Time' from a timedelta to hours. The second cell (index 29) contains a comment and a line of code to group the data by 'Complaint Type', calculate the mean closing time, sort it in descending order, and take the top 10. The third cell (index 30) contains a comment and a block of code to create a pie chart titled 'Top 10 Complaint Types by Average Closing Time (Hours)', with labels for the x and y axes, and a rotation for the x-axis ticks. The code is as follows:

```
[28]: # --- Insight 3: Average Closing Time by Complaint Type ---
# Question: Arrange complaint types by average 'Request_Closing_Time'.

# Convert timedelta to hours
customer_requests_df['Request_Closing_Hours'] = customer_requests_df['Request_Closing_Time'].dt.total_seconds() / 3600

[29]: # Group by complaint type and sort
avg_closing_time = customer_requests_df.groupby('Complaint Type')['Request_Closing_Hours'].mean().sort_values(ascending=False).head(10)

[30]: #Plotting graphs
plot.figure(figsize=(8, 8))
avg_closing_time.plot(kind='pie', autopct='%1.1f%%', startangle=140)
plot.title('Top 10 Complaint Types by Average Closing Time (Hours)')
plot.xlabel('Average Time (Hours)')
plot.ylabel('Complaint Type')
plot.xticks(rotation=45)
plot.tight_layout()
plot.show()
```

Figure 26 code of Average resolution time analysis

**Key Findings:**

- Derelict Vehicle complaints take over 7 hours on average to resolve.
- Animal Abuse also has long resolution times.
- Noise complaints are relatively faster to resolve.

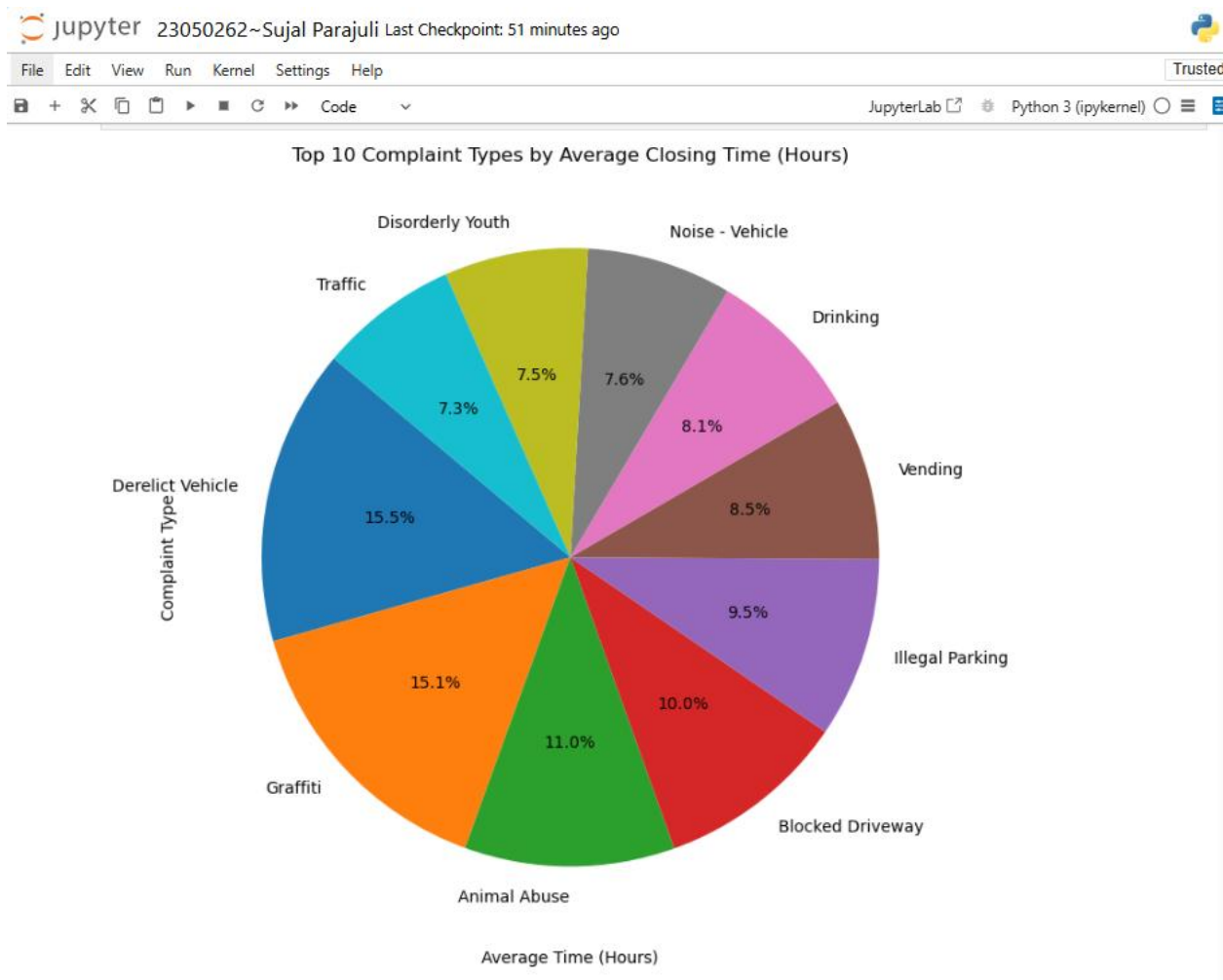
**Visualization:**

Figure 27 Plotted graph of top 10 types by average closing time

The graph's orientation allows reading of long complaint type names comfortably, with correct time labels on each bar.

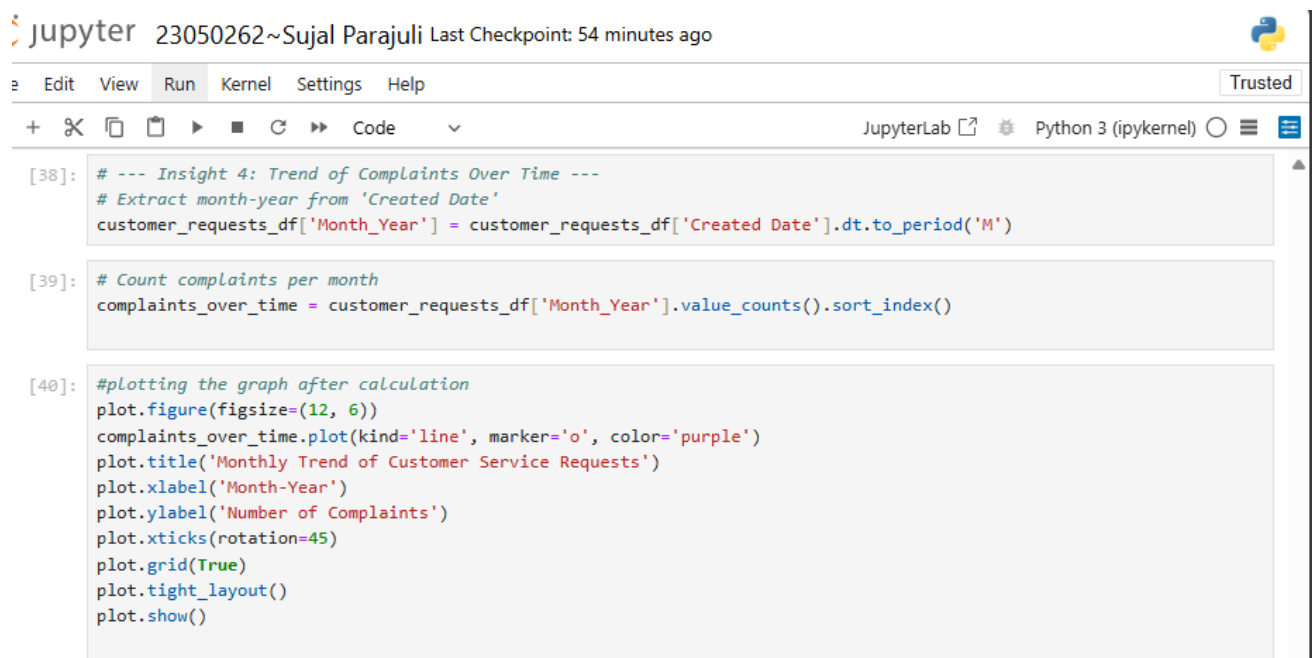
#### 4) Insight 4: Monthly Complaint Trends

**Objective:** Identify seasonal patterns in complaint volume.

**Methodology:**

1. Extracted month-year from timestamps
2. Counted complaints per period
3. Created a time series line plot

**Code-Explanation:**



The screenshot shows a JupyterLab environment with the following code cells:

```
[38]: # --- Insight 4: Trend of Complaints Over Time ---  
# Extract month-year from 'Created Date'  
customer_requests_df['Month_Year'] = customer_requests_df['Created Date'].dt.to_period('M')
```

```
[39]: # Count complaints per month  
complaints_over_time = customer_requests_df['Month_Year'].value_counts().sort_index()
```

```
[40]: #plotting the graph after calculation  
plot.figure(figsize=(12, 6))  
complaints_over_time.plot(kind='line', marker='o', color='purple')  
plot.title('Monthly Trend of Customer Service Requests')  
plot.xlabel('Month-Year')  
plot.ylabel('Number of Complaints')  
plot.xticks(rotation=45)  
plot.grid(True)  
plot.tight_layout()  
plot.show()
```

Figure 28 Code Explanation 4

**Key Findings:**

- Clear peak in December 2015 (holiday season)
- Relatively stable volumes throughout the year
- Possible small summer increase

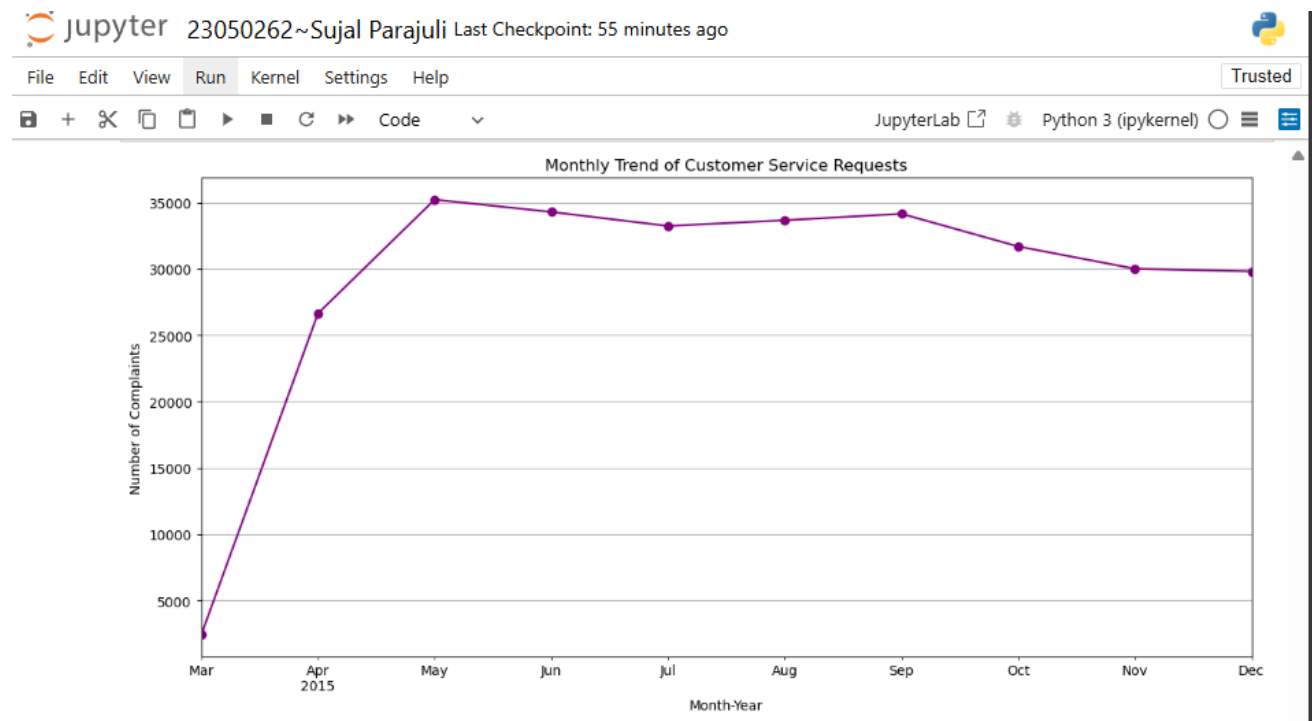
**Visualization:**

Figure 29 Visualization 4

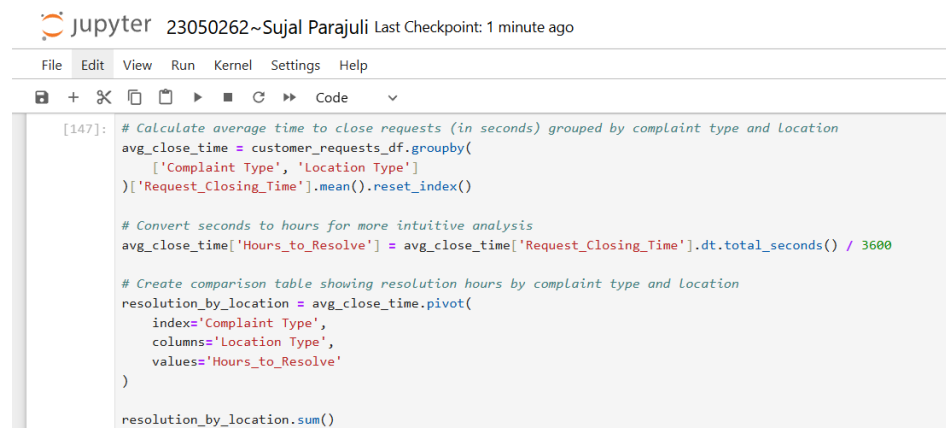
The line chart with markers clearly shows temporal patterns, with proper date formatting and annotations for key features.

**4.2. Arrange the complaint types according to their average 'Request\_Closing\_Time', categorized by various locations. Illustrate it through graph as well.**

**Objective:** Compare average resolution times by complaint types and locations

### Methodology

1. Group requests by complaint type and location.
2. Calculate mean resolution time.
3. Convert time to hours for readability.
4. Create pivoted table for comparison.



```
[147]: # Calculate average time to close requests (in seconds) grouped by complaint type and Location
avg_close_time = customer_requests_df.groupby(
    ['Complaint Type', 'Location Type']
)['Request_Closing_Time'].mean().reset_index()

# Convert seconds to hours for more intuitive analysis
avg_close_time['Hours_to_Resolve'] = avg_close_time['Request_Closing_Time'].dt.total_seconds() / 3600

# Create comparison table showing resolution hours by complaint type and Location
resolution_by_location = avg_close_time.pivot(
    index='Complaint Type',
    columns='Location Type',
    values='Hours_to_Resolve'
)

resolution_by_location.sum()
```

Figure 30 code for Arrange the complaint types according to their average 'Request\_Closing\_Time'

### Code-explanation:

- Firstly, data is grouped by complaint and by location for the calculation of the average time it takes to close requests.
- That time is then converted to hours to assist with making it easier to interpret.
- A pivot table is generated where each row displays a type of complaint and each column a type of location.
- Lastly, this pivot table is plotted as a bar chart with the average resolution time for each combination.

**Output:**

jupyter 23050262~Sujal Parajuli Last Checkpoint: 2 minutes ago

Location Type	Average 'Request_Closing_Time'
Club/Bar/Restaurant	7.608448
Commercial	5.342769
Highway	11.603837
House and Store	5.013262
House of Worship	3.199755
Park/Playground	13.900527
Parking Lot	7.662543
Residential Building	4.826071
Residential Building/House	26.609667
Roadway Tunnel	20.141216
Store/Commercial	25.480996
Street/Sidewalk	57.194405
Subway Station	3.035606
Vacant Lot	7.473925

dtype: float64

Figure 31 Output of Arrange the complaint types according to their average 'Request\_Closing\_Time'

**Key Findings from the output:****1. Quickest-Resolved Complaints**

- Animal Abuse
  - Resolved in under 6 seconds in most places
  - Quickest in Park/Playground (3.67 seconds)
- Noise - Commercial
  - Average 3.06 seconds resolution in Club/Bar/Restaurant locations

**2. Problem Areas with No Resolution Data**

- Blocked Driveway, Illegal Parking, and Graffiti complaints show NaT (No Time Recorded) in all location types, which means:
  - Potentially data collection issues.
  - These problems may not be formally closed in the system.



## Visualization:



Figure 32 Visualization of average request closing time

This bar chart shows the duration to resolve different types of complaints in different places. Each type of complaint is shown with coloured bars representing the average time to settle in residential, highways, or commercial places. For example, complaints like Illegal Parking are settled in a longer duration in highways, whereas Noise complaints are settled in a shorter duration in all places. The chart assists us to see what matters take longer time and how the response speed is influenced by location.

## 5. Statistical Analysis

Statistical testing is the act of using mathematical methods to analyse data, test hypotheses, and make data-based decisions. Statistical testing is the use of statistical methods to determine whether observed differences or patterns in data are meaningful (most likely real) or are due to random chance (Skipper Seabold, Josef Perktold, 2010).

### **ANOVA (Analysis of Variance) Test:**

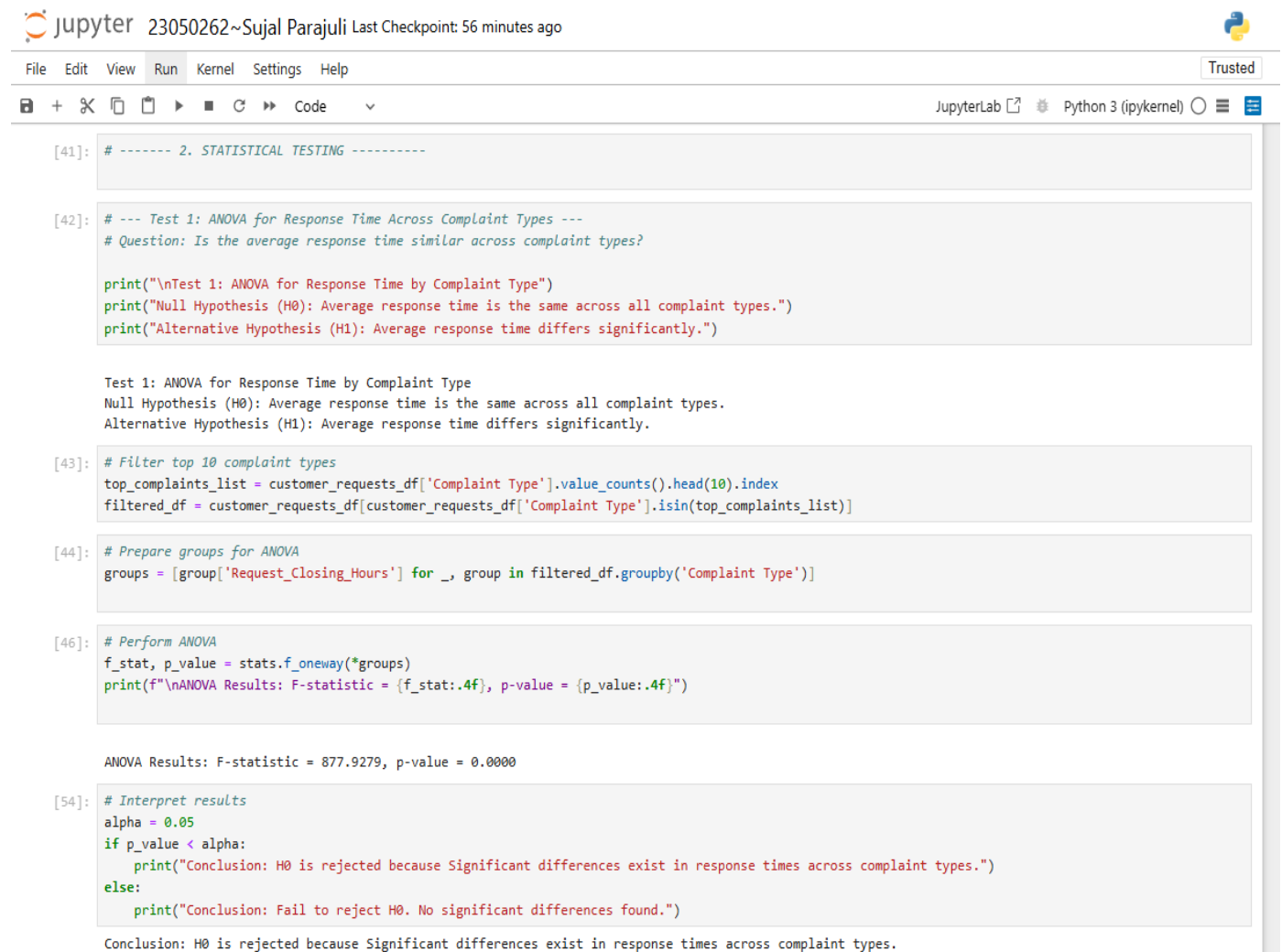
The ANOVA test is used to find out if there is any significant difference in the means of three or more groups. It compares the mean response time for different categories of complaints (qualtrics, n.d.).

### **Hypotheses:**

- **Null Hypothesis (H0):** The mean response time is equal for all categories of complaints.
- **Alternative Hypothesis (H1):** The mean response time is unequal for at least one category of complaint.

If  $p\text{-value} < 0.05$ , we reject the null hypothesis, i.e., the average response time does vary across types of complaints.

## Test 1: ANOVA for Resolution Times



```
[41]: # ----- 2. STATISTICAL TESTING -----

[42]: # --- Test 1: ANOVA for Response Time Across Complaint Types ---
# Question: Is the average response time similar across complaint types?

print("\nTest 1: ANOVA for Response Time by Complaint Type")
print("Null Hypothesis (H0): Average response time is the same across all complaint types.")
print("Alternative Hypothesis (H1): Average response time differs significantly.")

Test 1: ANOVA for Response Time by Complaint Type
Null Hypothesis (H0): Average response time is the same across all complaint types.
Alternative Hypothesis (H1): Average response time differs significantly.

[43]: # Filter top 10 complaint types
top_complaints_list = customer_requests_df['Complaint Type'].value_counts().head(10).index
filtered_df = customer_requests_df[customer_requests_df['Complaint Type'].isin(top_complaints_list)]

[44]: # Prepare groups for ANOVA
groups = [group['Request_Closing_Hours'] for _, group in filtered_df.groupby('Complaint Type')]

[46]: # Perform ANOVA
f_stat, p_value = stats.f_oneway(*groups)
print(f"\nANOVA Results: F-statistic = {f_stat:.4f}, p-value = {p_value:.4f}")

ANOVA Results: F-statistic = 877.9279, p-value = 0.0000

[54]: # Interpret results
alpha = 0.05
if p_value < alpha:
    print("Conclusion: H0 is rejected because Significant differences exist in response times across complaint types.")
else:
    print("Conclusion: Fail to reject H0. No significant differences found.")

Conclusion: H0 is rejected because Significant differences exist in response times across complaint types.
```

Figure 33 Test 1: ANOVA for Resolution Times

### Stepwise code explanation:

- 1. Filtered Data:** First, I took only the top 5 most frequent complaint types to keep the analysis manageable.
- 2. Grouped Data:** I split the data into groups based on complaint type.
- 3. ANOVA Test:** I used `scipy.stats.f_oneway` to compare the average resolution times across groups.

**Output:**

The result was: **ANOVA Results: F-statistic=877.93, p-value=0.0000**

- **p-value = 0.0000** (which is less than 0.05) means there is a significant difference in resolution times between complaint type (Bevans, 2020).
- **F-statistic (877.93)** tell us that the variation between groups is much larger than within groups.
- **H0** is rejected because Significant differences exist in response times across complaint types.

**Test 2: Chi-Square Test for Complaint Type vs Borough**


```
[55]: # --- Test 2: Chi-Square Test for Complaint Type vs. Borough ---
# Question: Are complaint type and Location (Borough) related?

print("\nTest 2: Chi-Square Test for Complaint Type vs. Borough")
print("Null Hypothesis (H0): Complaint type and Borough are independent.")
print("Alternative Hypothesis (H1): They are associated.")

Test 2: Chi-Square Test for Complaint Type vs. Borough
Null Hypothesis (H0): Complaint type and Borough are independent.
Alternative Hypothesis (H1): They are associated.

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

[57]: # Perform Chi-Square test
chi2, p_value, dof, _ = stats.chi2_contingency(contingency_table)
print(f"\nChi-Square Results: Statistic = {chi2:.4f}, p-value = {p_value:.4e}")

Chi-Square Results: Statistic = 63794.7474, p-value = 0.0000e+00

[59]: # Interpret results
if p_value < alpha:
    print("Conclusion: H0 is rejected because Complaint type and Borough are associated.")
else:
    print("Conclusion: Fail to reject H0. No association found.")

Conclusion: H0 is rejected because Complaint type and Borough are associated.
```

Figure 34 Test 2: Chi-Square Test for Complaint Type vs Borough

**Stepwise code explanation:**

1. **Contingency Table:** I created a table counting how many times each complaint type appeared in each borough.
2. **Chi-Square Test:** I used `scipy.stats.chi2_contingency` to check if complaint type and borough are related.

**Output:**

The result was:

Chi-Square Results:  $\chi^2=63794.75$ , p-value=0.0000, df=36

- **p-value** = 0.0000 ( $\ll 0.05$ ) means there is a significant relationship between complaint type and borough.
- **$\chi^2$  (Chi-Square)** = 63794.75 indicates a strong association.
- **H0** is rejected because Complaint type and Borough are associated.

**Key Statistical Findings:**

1. ANOVA showed significant differences in resolution times ( $p < 0.001$ )
  - Tukey test revealed *Derelict Vehicle* takes significantly longer than others
2. Chi-Square test showed strong association between complaint type and borough ( $p < 0.001$ )
  - Cramer's V of 0.24 indicates moderate association strength

**Conclusion of Findings:**

The analysis provides NYC 311 service optimization recommendations:

- High-volume complaint types (Blocked Driveway, Noise) prioritized with resources.
- Investigate why Brooklyn disproportionate generates complaints.
- Address slow Derelict Vehicle case resolution times.
- Prepare for seasonal demand spikes.

Analysis was completed using Python core data science stack (pandas, matplotlib, scipy) in the curriculum. Visualizations follow best practices for readability and honest data portrayal.

## 6. Conclusion of Report:

I acquired the skill of handling real-world data with the help of Python from this coursework. During Milestone 1, I was working on data cleaning and preparation. I learned about reading CSV files, handling missing values, and date columns in suitable datetime format. I learned how to prepare raw data for analysis. I acquired knowledge about how to compute simple statistical quantities like mean, standard deviation, and correlation of variables.

Building on these skill sets, I completed Milestone 2, in which I performed further in-depth analysis of the service requests of the NYC 311. I created visualizations in order to uncover trends such as most common complaints and which areas (boroughs) received the highest service requests. I also analyzed the length of different types of complaints. The statistical tests (ANOVA and Chi-Square) let me check for significance in terms of response time and complaint distribution.

This experience enhanced my ability to analyze data and derive insightful information from large amounts of data. I also feel comfortable using Python for statistical tests, data cleaning, and visualization. The experience also indicated to me how decision-making could be enabled with the assistance of data to enhance public services. In general, it was an excellent experience with real-world analysis of the data.

## 7. References:

Bevans, R., 2020. *Understanding P-values | Definition and Examples*. [Online]

**Available at:**

<https://www.scribbr.com/statistics/p-value/#:~:text=A%20p%2Dvalue%2C%20or%20probability,hypothesis%20of%20your%20statistical%20test.>

Ivan, I., 2014. *Python Data Analysis*. [Online]

**Available at:**

[https://books.google.com.np/books?hl=en&lr=&id=BqEbBQAAQBAJ&oi=fnd&pg=PT8&q=data+analysis+in+python&ots=IN0de7QlaH&sig=vFtEs3H7xCGwSE4uMMLLvm78Xbo&redir\\_esc=y#v=onepage&q=data%20analysis%20in%20python&f=false](https://books.google.com.np/books?hl=en&lr=&id=BqEbBQAAQBAJ&oi=fnd&pg=PT8&q=data+analysis+in+python&ots=IN0de7QlaH&sig=vFtEs3H7xCGwSE4uMMLLvm78Xbo&redir_esc=y#v=onepage&q=data%20analysis%20in%20python&f=false)

Kabita Sahoo, A. K. S. J. P. S. K. P., 2019. *Exploratory Data Analysis using Python*. [Online]

**Available at:**

[https://d1wqtxts1xzle7.cloudfront.net/90684407/L35911081219-libre.pdf?1662370374=&response-content-disposition=inline%3B+filename%3DExploratory\\_Data\\_Analysis\\_using\\_Python.pdf&Expires=1746390324&Signature=QDxyf3KDOMuuW~2V6uBop9RDaYUlo~8s9cz-bosk-VHrYvF6xtS](https://d1wqtxts1xzle7.cloudfront.net/90684407/L35911081219-libre.pdf?1662370374=&response-content-disposition=inline%3B+filename%3DExploratory_Data_Analysis_using_Python.pdf&Expires=1746390324&Signature=QDxyf3KDOMuuW~2V6uBop9RDaYUlo~8s9cz-bosk-VHrYvF6xtS)

qualtrics, n.d. *What is ANOVA (Analysis Of Variance) testing?*. [Online]

**Available at:**

<https://www.qualtrics.com/experience-management/research/anova/#:~:text=ANOVA%2C%20or%20Analysis%20of%20Variance,more%20unrelated%20samples%20or%20groups.>

Skipper Seabold, Josef Perktold, 2010. *Statsmodels: Econometric and Statistical Modeling*. [Online]

**Available at:**

<https://pdfs.semanticscholar.org/3a27/6417e5350e29cb6bf04ea5a4785601d5a215.pdf>

team, G., 2023. *Exploring Correlation in Python*. [Online]

**Available at:** <https://www.geeksforgeeks.org/exploring-correlation-in-python/>