Seattle Terry Stops Prediction Project

By: Shamla Tadese Araya

Moringa School

Aug 2024

INTRODUCTION

In Seattle, Terry stops refer to brief stops and detentions by police officers based on a reasonable suspicion that a person may be involved in criminal activity. The term originates from the U.S. Supreme Court case Terry v. Ohio (1968), which established the legal standard for such stops. In Seattle, these stops are subject to both federal and local regulations, and there has been considerable debate over their impact on communities, particularly concerning concerns about racial profiling and civil liberties. Efforts to refine and improve the practice focus on balancing effective policing with the protection of individual rights.

In this project we aim to achieve the following objectives:

- Determine if there is a racial disparity in the Seattle Terry Stops
- Do the differences in races between the officer and the subject play a role in frisks arrests?
- Determine the most common outcome of the Seattle Terry Stops and what it means
- Develop a model that can accurately predict the likelihood of an arrest occurring during a Terry Stop

This project will be divided into three workbooks, each focusing on a specific aspect of the process. We will start by exploring the data and cleaning, followed by followed by exploratory analysis aiming to addressing the first three objectives which are racial disparity during Terry Stops, role of race in the Terry Stops and the most common outcome of the Terry Stops in Seattle. We will then move on to the third workbook where we will be addressing the fourth objective of developing a comprehensive predictive model that can accurately predict the likelihood of an arrest following a Terry Stop based on various factors.

In this project we will be using the data obtained from City of Seattle on [https://data.seattle.gov/Public-Safety/Terry-Stops (https://data.seattle.gov/Public-Safety/Terry-Stops)](https://data.seattle.gov/Public-Safety/Terry-Stops).

Observing the Data

In [4]:
```
# Importing the relevant libraries for EDA and visualization

import numpy as np
import pandas as pd
from scipy import stats
from datetime import datetime
import warnings
warnings.filterwarnings(action='ignore')
warnings.filterwarnings('ignore')
```

In [5]:
```
# Loading the data
df = pd.read_csv('data/Terry_Stops_20240826.csv')

# Checking the first few rows of the data
df.head()
```

Out[5]:

	Subject Age Group	Subject ID	GO / SC Num	Terry Stop ID	Stop Resolution	Weapon Type	Officer ID	Officer YOB	Officer Gender	C
0	36 - 45	-1	20160000398323	208373	Offense Report	NaN	4852	1953	M	
1	18 - 25	-1	20180000227180	559146	Citation / Infraction	NaN	5472	1964	M	
2	18 - 25	-1	20180000410091	498246	Offense Report	NaN	6081	1962	M	
3	-	-1	20160000001637	146742	Field Contact	NaN	6924	1974	M	
4	46 - 55	-1	20150000006037	104477	Field Contact	NaN	6732	1975	M	

5 rows × 23 columns

In [6]:
```
# Checking the number of rows and columns of the data
df.shape
```

Out[6]: (61009, 23)

The dataset on Terry stops from the City of Seattle's open data portal contains about 61000 entries with 23 columns which typically includes information on interactions between Seattle police officers and individuals during Terry stops. Here's a general description of what this dataset contains:

Stop Date and Time: When the Terry stop occurred, including the specific date and time.

Location: The geographic location where the stop took place, often including neighborhood or precinct information.

Officer Details: Identifiers or information related to the officers who conducted the stop, though specific identifying details might be anonymized.

Demographic Information: Data on the individuals stopped, such as race, gender, and age. This helps in analyzing the demographic breakdown of those stopped.

Reason for Stop: The reason or suspicion that led to the stop, providing context for why the individual was stopped.

Outcome of the Stop: The result of the stop, such as whether a search was conducted, if an arrest was made, or if a citation was issued.

Search Details: Information on whether a search was conducted during the stop, and if so, what was found.

Interaction Type: Information on the nature of the interaction, such as whether it was a stop-and-frisk, a consent stop, or another type of encounter.

Agency and Division: Information about which division or unit within the police department conducted the stop.

The dataset aims to provide transparency and allow for analysis of police practices, helping to ensure accountability and evaluate the impact of Terry stops on different communities.

Column Names and Descriptions

The following descriptions were provided by data.seattle.gov This dataset contains the following data:

Subject Age Group: Subject Age Group (10 year increments) as reported by the officer.

Subject ID: Key, generated daily, identifying unique subjects in the dataset using a character to character match of first name and last name. "Null" values indicate an "anonymous" or "unidentified" subject. Subjects of a Terry Stop are not required to present identification.

GO / SC Num: General Offense or Street Check number, relating the Terry Stop to the parent report. This field may have a one to many relationship in the data.

Terry Stop ID: Key identifying unique Terry Stop reports.

Stop Resolution: Resolution of the stop as reported by the officer.

Weapon Type: Type of weapon, if any, identified during a search or frisk of the subject. Indicates "None" if no weapons was found.

Officer ID: Key identifying unique officers in the dataset.

Officer YOB: Year of birth, as reported by the officer.

Officer Gender: Gender of the officer, as reported by the officer.

Officer Race: Race of the officer, as reported by the officer.

Subject Perceived Race: Perceived race of the subject, as reported by the officer.

Subject Perceived Gender: Perceived gender of the subject, as reported by the officer.

Reported Date: Date the report was filed in the Records Management System (RMS). Not necessarily the date the stop occurred but generally within 1 day.

Reported Time: Time the stop was reported in the Records Management System (RMS). Not the time the stop occurred but generally within 10 hours.

Initial Call Type: Initial classification of the call as assigned by 911.

Final Call Type: Final classification of the call as assigned by the primary officer closing the event.

Call Type: How the call was received by the communication center.

Officer Squad: Functional squad assignment (not budget) of the officer as reported by the Data Analytics Platform (DAP).

Arrest Flag: Indicator of whether a "physical arrest" was made, of the subject, during the Terry Stop. Does not necessarily reflect a report of an arrest in the Records Management System (RMS).

Frisk Flag: Indicator of whether a "frisk" was conducted, by the officer, of the subject, during the Terry Stop.

Precinct: Precinct of the address associated with the underlying Computer Aided Dispatch (CAD) event. Not necessarily where the Terry Stop occurred.

Sector: Sector of the address associated with the underlying Computer Aided Dispatch (CAD) event. Not necessarily where the Terry Stop occurred.

Beat: Beat of the address associated with the underlying Computer Aided Dispatch (CAD) event. Not necessarily where the Terry Stop occurred.

In [7]:
```
# Getting a closer look at the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 61009 entries, 0 to 61008
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Subject Age Group         61009 non-null  object
 1   Subject ID                61009 non-null  int64 
 2   GO / SC Num               61009 non-null  int64 
 3   Terry Stop ID             61009 non-null  int64 
 4   Stop Resolution           61009 non-null  object
 5   Weapon Type               28444 non-null  object
 6   Officer ID                61009 non-null  object
 7   Officer YOB               61009 non-null  int64 
 8   Officer Gender            61009 non-null  object
 9   Officer Race              61009 non-null  object
 10  Subject Perceived Race    61009 non-null  object
 11  Subject Perceived Gender  61009 non-null  object
 12  Reported Date             61009 non-null  object
 13  Reported Time             61009 non-null  object
 14  Initial Call Type         61009 non-null  object
 15  Final Call Type           61009 non-null  object
 16  Call Type                 61009 non-null  object
 17  Officer Squad             60448 non-null  object
 18  Arrest Flag               61009 non-null  object
 19  Frisk Flag                61009 non-null  object
 20  Precinct                  61009 non-null  object
 21  Sector                    61009 non-null  object
 22  Beat                      61009 non-null  object
dtypes: int64(4), object(19)
memory usage: 10.7+ MB
```

The data types of the columns are as follows:

Column Classification

Numerical Columns:

Subject ID, GO / SC Num, Terry Stop ID, Officer YOB,

Categorical Columns:

Subject Age Group, Stop Resolution, Weapon Type, Officer ID, Officer Gender, Officer Race, Subject Perceived Race, Subject Perceived Gender, Reported Date, Reported Time, Initial Call Type, Final Call Type, Call Type, Officer Squad, Arrest Flag, Frisk Flag, Precinct, Sector, Beat,

In [8]:
```
# Checking for missing values in the dataset
df.isnull().sum()
```

Out[8]:
```
Subject Age Group               0
Subject ID                      0
GO / SC Num                     0
Terry Stop ID                   0
Stop Resolution                 0
Weapon Type                 32565
Officer ID                      0
Officer YOB                     0
Officer Gender                  0
Officer Race                    0
Subject Perceived Race          0
Subject Perceived Gender        0
Reported Date                   0
Reported Time                   0
Initial Call Type               0
Final Call Type                 0
Call Type                       0
Officer Squad                 561
Arrest Flag                     0
Frisk Flag                      0
Precinct                        0
Sector                          0
Beat                            0
dtype: int64
```

Here we can clearly see that the Weapon Type feature has a lot of values missing and also the Office Squad as well has 561 missing values. Still yet we need to check the dataset in depth to learn if there are any place holders, none or other unnecessary values/Characters.

In [9]:
```
# Creating a function that shows the value counts of each column in th
def col_values(df):
    """
    For use in Preprocessing and cleaning to find placeholder values
    Input: Data frame
    Output: Counts of unique values for each column
    """
    for col in df.columns:
        print(df[col].value_counts())
        print('-----------------------------------------------------------

col_values(df)
```

```
 7738872582        1
 16724979306       1
 19137661313       1
Name: count, Length: 17000, dtype: int64
-------------------------------------------------------
GO / SC Num
20160000378750    16
20150000190790    16
20180000134604    14
20210000267148    14
20230000049052    14
                  ..
20150000006142     1
20180000000272     1
20200000339446     1
20220000283906     1
20220000018102     1
Name: count, Length: 48845, dtype: int64
-------------------------------------------------------
Terry Stop ID
```

In [10]:
```
# For ease of use let us rename the columns
df.columns = ['subject_age_group', 'subject_id', 'go_sc_num', 'terry_s
       'stop_resolution', 'weapon_type', 'officer_id', 'officer_yob',
       'officer_gender', 'officer_race', 'subject_perceived_race',
       'subject_perceived_gender', 'reported_date', 'reported_time',
       'initial_call_type', 'final_call_type', 'call_type', 'officer_s
       'arrest_flag', 'frisk_flag', 'precinct', 'sector', 'beat']

df.columns
```

Out[10]:
```
Index(['subject_age_group', 'subject_id', 'go_sc_num', 'terry_stop_i
d',
       'stop_resolution', 'weapon_type', 'officer_id', 'officer_yo
b',
       'officer_gender', 'officer_race', 'subject_perceived_race',
       'subject_perceived_gender', 'reported_date', 'reported_time',
       'initial_call_type', 'final_call_type', 'call_type', 'officer
_squad',
       'arrest_flag', 'frisk_flag', 'precinct', 'sector', 'beat'],
      dtype='object')
```

In [11]:
```
# Checking the dataframe
df.head()
```

Out[11]:

	subject_age_group	subject_id	go_sc_num	terry_stop_id	stop_resolution	weapon_type
0	36 - 45	-1	20160000398323	208373	Offense Report	NaN
1	18 - 25	-1	20180000227180	559146	Citation / Infraction	NaN
2	18 - 25	-1	20180000410091	498246	Offense Report	NaN
3	-	-1	20160000001637	146742	Field Contact	NaN
4	46 - 55	-1	20150000006037	104477	Field Contact	NaN

5 rows × 23 columns

Data Cleaning

Let us clean the data before we proceed to processing. Let us start by replacing the dashes and place holders with the more workable values first. Then we will go on to the more complex cleaning process.

In [12]:
```
# Replacing the dashes with Unknown
df = df.replace('-', 'Unknown')
df.head()
```

Out[12]:

	subject_age_group	subject_id	go_sc_num	terry_stop_id	stop_resolution	weapon_type
0	36 - 45	-1	20160000398323	208373	Offense Report	NaN
1	18 - 25	-1	20180000227180	559146	Citation / Infraction	NaN
2	18 - 25	-1	20180000410091	498246	Offense Report	NaN
3	Unknown	-1	20160000001637	146742	Field Contact	NaN
4	46 - 55	-1	20150000006037	104477	Field Contact	NaN

5 rows × 23 columns

Officer_gender has 30 'N' values. We cannot be sure if 'N' stands for 'Not Available', 'Not Disclosed', or even 'Non-Gender Binary'. Since it's such a small amount of data, we'll just drop it.

In [13]:
```
# Dropping the entries with 'N' values from the officer_gender column.
df.drop(df[df['officer_gender'] == 'N'].index, inplace=True)
df.officer_gender.value_counts()
```

Out[13]:
```
officer_gender
M    54072
F     6907
Name: count, dtype: int64
```

Officer_squad also has some NAN values. Since this information is less relevant to this particular task, it is better to just drop the column.

In [14]:
```
# Dropping the officer_squad column and assigning the data to a copy d
df.drop('officer_squad', axis=1, inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 60979 entries, 0 to 61008
Data columns (total 22 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   subject_age_group         60979 non-null  object
 1   subject_id                60979 non-null  int64 
 2   go_sc_num                 60979 non-null  int64 
 3   terry_stop_id             60979 non-null  int64 
 4   stop_resolution           60979 non-null  object
 5   weapon_type               28419 non-null  object
 6   officer_id                60979 non-null  object
 7   officer_yob               60979 non-null  int64 
 8   officer_gender            60979 non-null  object
 9   officer_race              60979 non-null  object
 10  subject_perceived_race    60979 non-null  object
 11  subject_perceived_gender  60979 non-null  object
 12  reported_date             60979 non-null  object
 13  reported_time             60979 non-null  object
 14  initial_call_type         60979 non-null  object
 15  final_call_type           60979 non-null  object
 16  call_type                 60979 non-null  object
 17  arrest_flag               60979 non-null  object
 18  frisk_flag                60979 non-null  object
 19  precinct                  60979 non-null  object
 20  sector                    60979 non-null  object
 21  beat                      60979 non-null  object
dtypes: int64(4), object(18)
memory usage: 10.7+ MB
```

As we saw it above, there are some subject IDs that are repeated multiple times. This could be either duplicates or repeat offenders. So it is crucial to investigate that feature.

In [15]:
```
# Checking the subject_id column
df['subject_id'].value_counts()
```

Out[15]:
```
subject_id
-1                35095
 7753260438          28
 7774286580          22
 7726918259          21
 7731717691          20
                  ...  
 15606702593          1
 7735943699           1
 7738872582           1
 16724979306          1
 19137661313          1
Name: count, Length: 16988, dtype: int64
```

In [16]:
```
# Let us replace those -1 values in 'Subject_ID' with 'unassigned'
df['subject_id'] = df['subject_id'].replace(-1, 'unassigned')
df.subject_id.value_counts()
```

Out[16]:
```
subject_id
unassigned     35095
7753260438        28
7774286580        22
7726918259        21
7731717691        20
               ...
15606702593        1
7735943699         1
7738872582         1
16724979306        1
19137661313        1
Name: count, Length: 16988, dtype: int64
```

Here it looks like we have multiple duplicates in 'Subject_IDs'. If that is the case this could make our dataset biased so we need to check closely to decide whether we have duplicates or not. We can do this by checking a number of columns namely 'subject_id', 'terry_stop_id' and 'officer_id'.

In [17]:
```
# Group by 'subject_id', 'terry_stop_id', and 'officer_id' and count o
df['count'] = df.groupby(['subject_id', 'terry_stop_id', 'officer_id']

# Create 'repeat_offenders' column based on count
df['repeat_offenders'] = df['count'].apply(lambda x: 'Yes' if x > 1 el

# Drop the 'count' column as it is no longer needed
df.drop(columns=['count'], axis=1, inplace=True)

df['repeat_offenders'].value_counts()
```

Out[17]:
```
repeat_offenders
No     60784
Yes      195
Name: count, dtype: int64
```

This tells us we have 195 duplicated in our dataset but still we need to dig deeper before we conclusively decide.

Terry Stop ID also has some duplicate values worth checking.

In [18]:
```
# Checking terry stop id value counts
df['terry_stop_id'].value_counts()
```

Out[18]:
```
terry_stop_id
19324329995    3
19268585233    3
27511831225    3
36014210659    3
32633045284    3
              ..
87443          1
108886         1
274766         1
12093615563    1
31342435997    1
Name: count, Length: 60877, dtype: int64
```

In [19]:
```
# Listing the duplicates
dup_ids = df[df['terry_stop_id'].duplicated(keep=False)].sort_values(t
# dup_ids = dup_ids[['subject_age_group', 'subject_id', 'go_sc_num',
#                      'terry_stop_id', 'stop_resolution', 'weapon_type'
#                      'officer_id', 'reported_date', 'reported_time',
#                      'initial_call_type', 'final_call_type', 'arrest_i
#                      'frisk_flag', ]]
dup_ids
```

Out[19]:

	subject_age_group	subject_id	go_sc_num	terry_stop_id	stop_resolution	
52	26 - 35	7810387129	20190000254490	8611673538	Field Contact	Knife/(
43012	26 - 35	7810387129	20190000254490	8611673538	Field Contact	Blun
14583	26 - 35	7730805128	20190000268604	8677596250	Offense Report	
8805	26 - 35	7730805128	20190000268604	8677596250	Offense Report	Knife/(
19773	18 - 25	9458419522	20190000285750	9585545373	Field Contact	
...	...	...	...	...	...	
20234	26 - 35	53848066671	20240000133855	56110860878	Arrest	Knife/(
46922	36 - 45	57754429915	20240000198136	57754515446	Arrest	Knife/(
42736	36 - 45	57754429915	20240000198136	57754515446	Arrest	Blun
9130	18 - 25	7741755512	20240000210107	57961741719	Arrest	
21376	18 - 25	7741755512	20240000210107	57961741719	Arrest	Knife/(

195 rows × 23 columns

This may look like a confirmation of duplicate entries at first glance. However if we look carefully we can see that all these incidents have different weapon types even though they have identical subject, stop and officer ids. From this we can understand that these incidents are entries done by the same officer at the same time with the same subject who happen to be with multiple weapon types. After all it is a normal procedure for officers to place multiple entries of the same subject based on each weapon type found with.

In [20]:
```
# Group by 'subject_id', 'terry_stop_id', and 'officer_id' and count c
df['count'] = df.groupby(['subject_id', 'terry_stop_id', 'officer_id',

# Create 'repeat_offenders' column based on count
df['repeat_offenders'] = df['count'].apply(lambda x: 'Yes' if x > 1 el

# Drop the 'count' column as it is no longer needed
df.drop(columns=['count'], axis=1, inplace=True)

df['repeat_offenders'].value_counts()
```

Out[20]:
```
repeat_offenders
No    60979
Name: count, dtype: int64
```

In [21]:
```
df.head()
```

Out[21]:

	subject_age_group	subject_id	go_sc_num	terry_stop_id	stop_resolution	weapon_type
0	36 - 45	unassigned	20160000398323	208373	Offense Report	NaN
1	18 - 25	unassigned	20180000227180	559146	Citation / Infraction	NaN
2	18 - 25	unassigned	20180000410091	498246	Offense Report	NaN
3	Unknown	unassigned	20160000001637	146742	Field Contact	NaN
4	46 - 55	unassigned	20150000006037	104477	Field Contact	NaN

5 rows × 23 columns

This confirms our observation. Therefore since having this multiple entries of the same subjects can bloat our dataset and since the incidents are only 195, it is better to drop the duplicates and keep only the first entries.

In [22]:
```
# Dropping the duplicates and keeping the first instance
df.drop_duplicates('terry_stop_id', keep='first', inplace=True)
df.sort_values(by='terry_stop_id')
```

Out[22]:

	subject_age_group	subject_id	go_sc_num	terry_stop_id	stop_resolution	weapc
5536	1 - 17	unassigned	20150000084533	28020	Referred for Prosecution	Lethal Ins
34249	36 - 45	unassigned	20150000001428	28092	Field Contact	
22176	18 - 25	unassigned	20150000001428	28093	Field Contact	
46800	26 - 35	unassigned	20150000001437	28381	Field Contact	
35165	36 - 45	unassigned	20150000087329	28462	Offense Report	
...	...	...	...	...	...	
8952	18 - 25	33970989734	20240000173704	58505996345	Field Contact	H
43019	1 - 17	7734651568	20240000173704	58506837543	Field Contact	Ui
31174	1 - 17	53252554865	20240000173704	58506902793	Field Contact	Ui
32618	26 - 35	7729016487	20240000238040	58508727659	Field Contact	Ui
12853	18 - 25	58509790601	20240000238282	58509776131	Field Contact	Ui

60877 rows × 23 columns

Next let us check some repetitions in the general offense street check column.

In [23]:
```
# Investigating the repeated values in the go_sc_num column
stops = df[df['go_sc_num'] > 1]
stops['go_sc_num'].value_counts()
```

Out[23]:
```
go_sc_num
20160000378750    16
20150000190790    16
20230000049052    14
20180000134604    14
20210000267148    14
                  ..
20170000437667     1
20210000238907     1
20220000218677     1
20220000320502     1
20220000018102     1
Name: count, Length: 48826, dtype: int64
```

In [24]:
```
# Looking closely
stops = stops[stops['go_sc_num'] == 20160000378750]
stops
```

Out[24]:

	subject_age_group	subject_id	go_sc_num	terry_stop_id	stop_resolution	weapon_
5940	46 - 55	unassigned	20160000378750	208306	Offense Report	
11366	36 - 45	unassigned	20160000378750	208312	Offense Report	
13031	26 - 35	unassigned	20160000378750	208300	Offense Report	
33240	36 - 45	unassigned	20160000378750	208314	Arrest	
33810	46 - 55	unassigned	20160000378750	208309	Arrest	
36586	46 - 55	unassigned	20160000378750	208304	Offense Report	
36631	36 - 45	unassigned	20160000378750	208299	Offense Report	
39754	26 - 35	unassigned	20160000378750	208307	Offense Report	
41484	26 - 35	unassigned	20160000378750	208301	Offense Report	
43337	18 - 25	unassigned	20160000378750	208311	Arrest	
48555	26 - 35	unassigned	20160000378750	208303	Offense Report	
49658	18 - 25	unassigned	20160000378750	208302	Offense Report	
55394	36 - 45	unassigned	20160000378750	208308	Offense Report	
57795	46 - 55	unassigned	20160000378750	208313	Arrest	
58626	36 - 45	unassigned	20160000378750	208305	Offense Report	
59513	36 - 45	unassigned	20160000378750	208310	Offense Report	

16 rows × 23 columns

Taking into account the dates, the separate Terry Stop ID's, the different Stop Resolutions and it all roughly happening within the same hour, it appears that this was a **dispute** of some sort in which an officer **collected Offense Reports from 12 people** and issued out **tickets 4 people** (because there was **no physical arrest** denoted by the column 'arrest_flag', these were **non-custodial** arrests/citations).

Looking back at the Column Description document, the GO/SC Number is considered the **"parent report"** that contain **associated Terry Stops**. This confirms our observations.

Report Date

Ok so now lets remove the timestamp from date, create a new columns "incident year" and "incident month" with the year and month of the incidents and drop the reported date.

In [25]:
```
# Checking the column reported date
df.reported_date.dtype, df.reported_date.head()
```

Out[25]:
```
(dtype('O'),
 0    2016-11-03T00:00:00Z
 1    2018-06-22T00:00:00Z
 2    2018-11-02T00:00:00Z
 3    2016-04-17T00:00:00Z
 4    2015-11-29T00:00:00Z
 Name: reported_date, dtype: object)
```

In [26]:
```
# Converting to date time format
df['reported_date'] = pd.to_datetime(df['reported_date'])

# Creating a new column with the year of the incident
df['incident_year'] = df['reported_date'].dt.year

# Creating a new column with the month of the incident
df['incident_month'] = df['reported_date'].dt.month

# Dropping the reported date column
df.drop('reported_date', axis=1, inplace=True)
df.head()
```

Out[26]:

	subject_age_group	subject_id	go_sc_num	terry_stop_id	stop_resolution	weapon_type
0	36 - 45	unassigned	20160000398323	208373	Offense Report	NaN
1	18 - 25	unassigned	20180000227180	559146	Citation / Infraction	NaN
2	18 - 25	unassigned	20180000410091	498246	Offense Report	NaN
3	Unknown	unassigned	20160000001637	146742	Field Contact	NaN
4	46 - 55	unassigned	20150000006037	104477	Field Contact	NaN

5 rows × 24 columns

Officer Age

Let us create a new column 'officer_age' that holds the age value of the officer at the time of the incident. We can do this by subtracting the officer year of birth from the incident year.

In [27]:
```
# Creating a column that holds the officer's year.
df['officer_age'] = df['incident_year'] - df['officer_yob']
df.officer_age.unique()
```

Out[27]:
```
array([ 63,  54,  56,  42,  40,  48,  32,  39,  31,  38,  24,  28,
37,
        34,  27,  23,  43,  33,  26,  35,  30, 121,  55,  25,  29,
52,
        51,  57,  22,  49,  47,  36,  44,  45,  46,  50,  58,  41,
60,
        61,  53,  62,  64,  65,  59,  69,  71, 120,  21,  67, 118,
70,
        66,  68, 119])
```

Wow we have some entries for officer age that are unrealistic. Let us fix that.

In [28]:
```
# Dropping unrealistic ages from the officers age column
# df[df['officer_age'] <= 100]
df.drop(df[df['officer_age'] >= 100].index, inplace=True)

# Confirming our change
df['officer_age'].describe()
```

Out[28]:
```
count    60807.000000
mean        34.488102
std          8.267055
min         21.000000
25%         28.000000
50%         33.000000
75%         39.000000
max         71.000000
Name: officer_age, dtype: float64
```

So now our officer age looks more realistic ranging from 21 years to 71 years old. Let us now drop the officer year of birth column from the dataframe.

In [29]:
```
# Dropping the officer_yob column
df.drop('officer_yob', axis=1, inplace=True)
df.columns
```

Out[29]:
```
Index(['subject_age_group', 'subject_id', 'go_sc_num', 'terry_stop_i
d',
       'stop_resolution', 'weapon_type', 'officer_id', 'officer_gend
er',
       'officer_race', 'subject_perceived_race', 'subject_perceived_
gender',
       'reported_time', 'initial_call_type', 'final_call_type', 'cal
l_type',
       'arrest_flag', 'frisk_flag', 'precinct', 'sector', 'beat',
       'repeat_offenders', 'incident_year', 'incident_month', 'offic
er_age'],
      dtype='object')
```

Let us now proceed to the stop resolution. From common knowledge, we know that any arrest which is not flagged as one in the appropriate column is considered a "non-custodial arrest" or an instance where a citation was issued.

In [30]:
```
# Checking the stop resolution
df['stop_resolution'].value_counts()
```

Out[30]:
```
stop_resolution
Field Contact               29439
Offense Report              15701
Arrest                      14722
Referred for Prosecution      728
Citation / Infraction         217
Name: count, dtype: int64
```

Even though this column tells us what happened after the incident, the `Field Contact` and `Offense Report` values do give us insight as to why an officer may have initiated a stop. So let us create columns for these values and drop the stop resolution column.

In [31]:
```
# Creating field_contact column that contains 'y' and 'n' values
df['field_contact'] = df['stop_resolution'].str.contains('Field Contac
df['field_contact'] = df['field_contact'].map({True: 'Y', False: 'N'})

# Creating offense_report column that contains 'y' and 'n' values
df['offense_report'] = df['stop_resolution'].str.contains('Offense Rep
df['offense_report'] = df['offense_report'].map({True: 'Y', False: 'N

df.head()
```

Out[31]:

	subject_age_group	subject_id	go_sc_num	terry_stop_id	stop_resolution	weapon_type
0	36 - 45	unassigned	20160000398323	208373	Offense Report	NaN
1	18 - 25	unassigned	20180000227180	559146	Citation / Infraction	NaN
2	18 - 25	unassigned	20180000410091	498246	Offense Report	NaN
3	Unknown	unassigned	20160000001637	146742	Field Contact	NaN
4	46 - 55	unassigned	20150000006037	104477	Field Contact	NaN

5 rows × 26 columns

In [32]:
```
# Checking our new columns
df.offense_report.value_counts(), df.field_contact.value_counts()
```

Out[32]:
```
(offense_report
 N    45106
 Y    15701
 Name: count, dtype: int64,
 field_contact
 N    31368
 Y    29439
 Name: count, dtype: int64)
```

The weapon type column contains a lot of redundant values. Let us clean it up and organize it well.

In [33]:
```
# Checking the weapon type column
df['weapon_type'].value_counts()
```

Out[33]:
```
weapon_type
Unknown                                24493
Lethal Cutting Instrument               1482
Knife/Cutting/Stabbing Instrument       1289
Handgun                                  384
Blunt Object/Striking Implement          150
Firearm                                  102
Firearm Other                            100
Other Firearm                             73
Club, Blackjack, Brass Knuckles           49
Mace/Pepper Spray                         48
None/Not Applicable                       18
Firearm (unk type)                        15
Taser/Stun Gun                            14
Fire/Incendiary Device                    12
Rifle                                     10
Club                                       9
Shotgun                                    5
Automatic Handgun                          2
Personal Weapons (hands, feet, etc.)       2
Poison                                     1
Blackjack                                  1
Brass Knuckles                             1
Name: count, dtype: int64
```

In [34]:
```
# Weapon type categories
none = ['None/Not Applicable']

knife = ['Lethal Cutting Instrument', 'Knife/Cutting/Stabbing Instrume

blunt_obj = ['Club, Blackjack, Brass Knuckles', 'Club', 'Blackjack',
firearm = ['Firearm Other', 'Firearm (unk type)', 'Other Firearm', 'R
           'Shotgun', 'Automatic Handgun', 'Handgun']
other = ['Taser/Stun Gun', 'Mace/Pepper Spray', 'Fire/Incendiary Devi

# Creating a function called replace_val that takes the source data, 
def replace_val(df, col, old_val, new_val):
    for i in range(len(df[col])):
        for j in range(len(old_val)):
            if df[col].iloc[i] == old_val[j]:
                df[col].iloc[i] = df[col].iloc[i].replace(old_val[j],


# Applying the function to replace weapon type values
# replacing none
replace_val(df, 'weapon_type', none, 'None')

# replacing knife
replace_val(df, 'weapon_type', knife, 'Knife/Stabbing Instrument')

# replacing blunt object
replace_val(df, 'weapon_type', blunt_obj, 'Blunt Object/Striking Imple

# replacing firearm
replace_val(df, 'weapon_type', firearm, 'Firearm')

# other
replace_val(df, 'weapon_type', other, 'Other')

df['weapon_type'].value_counts()
```

Out[34]:
```
weapon_type
Unknown                            24493
Knife/Stabbing Instrument           2771
Firearm                              691
Blunt Object/Striking Implement      210
Other                                 77
None                                  18
Name: count, dtype: int64
```

Let us tidy up the reported time column as well

In [35]:
```
# Converting the time column to datetime format and keeping only the h
df['reported_time'] = pd.to_datetime(df['reported_time'])
df['reported_hour'] = df['reported_time'].dt.hour
df.drop('reported_time', axis=1, inplace=True)
df.reported_hour.head()
```

Out[35]:
```
0    15
1     0
2     2
3     1
4     2
Name: reported_hour, dtype: int32
```

Great the reported time now has been arranged by hour in 24 hour format.

In [36]:
```
# CHecking the copy dataframe so far
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 60807 entries, 0 to 61007
Data columns (total 26 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   subject_age_group         60807 non-null  object
 1   subject_id                60807 non-null  object
 2   go_sc_num                 60807 non-null  int64 
 3   terry_stop_id             60807 non-null  int64 
 4   stop_resolution           60807 non-null  object
 5   weapon_type               28260 non-null  object
 6   officer_id                60807 non-null  object
 7   officer_gender            60807 non-null  object
 8   officer_race              60807 non-null  object
 9   subject_perceived_race    60807 non-null  object
 10  subject_perceived_gender  60807 non-null  object
 11  initial_call_type         60807 non-null  object
 12  final_call_type           60807 non-null  object
 13  call_type                 60807 non-null  object
 14  arrest_flag               60807 non-null  object
 15  frisk_flag                60807 non-null  object
 16  precinct                  60807 non-null  object
 17  sector                    60807 non-null  object
 18  beat                      60807 non-null  object
 19  repeat_offenders          60807 non-null  object
 20  incident_year             60807 non-null  int32 
 21  incident_month            60807 non-null  int32 
 22  officer_age               60807 non-null  int64 
 23  field_contact             60807 non-null  object
 24  offense_report            60807 non-null  object
 25  reported_hour             60807 non-null  int32 
dtypes: int32(3), int64(3), object(20)
memory usage: 11.8+ MB
```

Moving on let us work with the call types column. There are 13000+ entries in this column with an 'unknown' values which indicates that these instances were not put in the CAD system. This could be unanimous calls for privacy reasons so we will keep them. However we will drop all the other entries of this column that have very little values.

In [37]:
```
# Checking the call types column
df['initial_call_type'].value_counts()
```

Out[37]:
```
initial_call_type
Unknown                                         13406
SUSPICIOUS STOP - OFFICER INITIATED ONVIEW       4769
SUSPICIOUS PERSON, VEHICLE, OR INCIDENT          4200
DISTURBANCE                                      3060
ASLT - CRITICAL (NO SHOOTINGS)                   2702
                                                ...
ESCAPE - PRISONER                                   1
PHONE - OBSCENE OR NUISANCE PHONE CALLS             1
EXPLOSION                                           1
ORDER - ASSIST DV VIC W/SRVC OF COURT ORDER         1
-ASSIGNED DUTY - STAKEOUT                           1
Name: count, Length: 181, dtype: int64
```

ESCAPE - PRISONER, PHONE - OBSCENE OR NUISANCE PHONE CALLS, EXPLOSION, ORDER - ASSIST DV VIC W/SRVC OF COURT ORDER, ASSIGNED DUTY - STAKEOUT, TEXT MESSAGE and SCHEDULED EVENT (RECURRING) all of these have very small values, it is better to drop them as well. We are keeping the ones with the unknown values as they are so many and we assume that these instances could be meant for privacy reasons.

In [38]:
```
# Dropping the call types with small values
df = df[(df['call_type'] != 'ESCAPE - PRISONER')&
                  (df['call_type'] != 'PHONE - OBSCENE OR NUISANCE PH
                  (df['call_type'] != 'EXPLOSION')&
                  (df['call_type'] != 'ORDER - ASSIST DV VIC W/SRVC OF
                  (df['call_type'] != 'ASSIGNED DUTY - STAKEOUT')&
                  (df['call_type'] != 'SCHEDULED EVENT (RECURRING)')&
                  (df['call_type'] != 'TEXT MESSAGE')]


df['call_type'].value_counts()
```

Out[38]:
```
call_type
911                            28734
ONVIEW                         14013
Unknown                        13406
TELEPHONE OTHER, NOT 911        4099
ALARM CALL (NOT POLICE ALARM)    525
Name: count, dtype: int64
```

Alright now that the initial call type is cleared let us proceed frisk flag.

In [39]:
```
# Checking our dataframe.
df.shape
```

Out[39]:
```
(60777, 26)
```

Let us check and clean the frisk flag instances.

In [40]:
```
# Checking the frisk flag column
df['frisk_flag'].value_counts()

#Dropping the frisk flag instances with unknown values only 478 instar
df.drop(df[df['frisk_flag'] == 'Unknown'].index, inplace=True)

df['frisk_flag'].value_counts()
```

Out[40]:
```
frisk_flag
N    45834
Y    14465
Name: count, dtype: int64
```

In [41]:
```
# Checking the arrest flag column for any missing values
df['arrest_flag'].value_counts()
```

Out[41]:
```
arrest_flag
N    53789
Y     6510
Name: count, dtype: int64
```

The arrest flag column seems clean with no missing values. Out of the total instances we have only 6510 arrests which make up to 10%.

Officer Race has some instances with unknown or not specified values. It is better to combine them together and tidying up the column.

In [42]:
```
# Checking the officer race column
df['officer_race'].value_counts()
```

Out[42]:
```
officer_race
White                              43220
Two or More Races                   4201
Hispanic or Latino                  3986
Asian                               2876
Not Specified                       2828
Black or African American           2403
Nat Hawaiian/Oth Pac Islander        545
American Indian/Alaska Native        240
Name: count, dtype: int64
```

In [43]:
```
# Combining the Unknown values into the not specified values of the o
replace_val(df, 'officer_race', ['Unknown/Unspecified', 'Not Stated'],

df.officer_race.value_counts()
```

Out[43]:
```
officer_race
White                              43220
Two or More Races                   4201
Hispanic or Latino                  3986
Asian                               2876
Not Specified                       2828
Black or African American           2403
Nat Hawaiian/Oth Pac Islander        545
American Indian/Alaska Native        240
Name: count, dtype: int64
```

Subject Gender same as officer race has unknown and unable to determine values that need to be combined together.

In [44]:
```
# Combining the Unknown values into the unable to determine values of
Unknown = ['Unknown']
replace_val(df,'subject_perceived_gender', Unknown, 'Unable to Determi

df.subject_perceived_gender.value_counts()
```

Out[44]:
```
subject_perceived_gender
Male                                                       47640
Female                                                     12004
Unable to Determine                                          608
Gender Diverse (gender non-conforming and/or transgender)     45
MULTIPLE SUBJECTS                                              2
Name: count, dtype: int64
```

In [45]:
```
# Dropping the gender diverse and multiple subjects values as they are
df.drop(df[df['subject_perceived_gender'].isin([
    'Gender Diverse (gender non-conforming and/or transgender)',
    'MULTIPLE SUBJECTS'
])].index, inplace=True)

df.subject_perceived_gender.value_counts()
```

Out[45]:
```
subject_perceived_gender
Male                   47640
Female                 12004
Unable to Determine      608
Name: count, dtype: int64
```

Precinct, Sector and Beat

These are location data which can be very important in this process as they determine the probability of one getting stopped. However they have some place holder values which need to be cleaned up.

In [46]:
```
# Checking the precinct data
df['precinct'].value_counts()
```

Out[46]:
```
precinct
West         16657
North        12693
Unknown      10661
East          8159
South         7290
Southwest     4673
OOJ             97
FK ERROR        22
Name: count, dtype: int64
```

In [47]:
```
# Let us check what the FK ERROR is
df[df['precinct'] == 'FK ERROR']
```

Out[47]:

	subject_age_group	subject_id	go_sc_num	terry_stop_id	stop_resolution	weap
1996	56 and Above	7760748894	20190000369575	10569761986	Field Contact	U
2697	1 - 17	34202427492	20220000012513	31222898051	Arrest	
6011	36 - 45	21896920197	20210000064178	21897246304	Arrest	U
18028	18 - 25	10392618417	20190000350728	10392612439	Field Contact	U
22615	36 - 45	7727091519	20190000325595	10042391872	Field Contact	Knife/S Ins
22642	26 - 35	12172435351	20200000021260	12172421137	Field Contact	U
24264	26 - 35	7732925068	20200000028389	12221869321	Arrest	U
32431	46 - 55	7729016287	20210000021036	20132790775	Field Contact	U
35677	26 - 35	7726837499	20190000468247	12108530793	Offense Report	U
36841	18 - 25	58002696823	20200000187751	13477897443	Field Contact	U
38123	26 - 35	8333698983	20190000222535	8333750884	Field Contact	U
38209	26 - 35	9804531492	20210000005854	19280652489	Offense Report	U
40088	46 - 55	7736599528	20190000196167	8258190629	Field Contact	U
41551	26 - 35	7726362993	20190000283674	9258189887	Field Contact	U
49974	1 - 17	8194784044	20190000224323	8335625151	Field Contact	U
50407	56 and Above	7728365691	20190000240737	8544232751	Offense Report	U
51247	26 - 35	7728529496	20190000222535	8333754250	Field Contact	U
51597	36 - 45	7732113716	20200000202980	13811156623	Field Contact	U
52274	26 - 35	7726955629	20210000021036	20132828518	Field Contact	U
53464	26 - 35	7749300947	20190000215001	8317517528	Field Contact	U
54055	46 - 55	7744306937	20210000101823	23375775291	Field Contact	Knife/S Ins
57220	46 - 55	19401241744	20210000007326	19403608968	Arrest	Knife/S Ins

22 rows × 26 columns

It looks very interesting the data we have here is between 2019 and 2022 half of them being in 2019. This suggests that the error is most likely due to system failure. Since this incidents really occurred we cannot ignore them and must clean them.

In [48]:
```
prec = ['FK ERROR', 'OOJ'] # OOJ stands for Obstruction of Justice
sect = ['99']
beats = ['99', '99', 'OOJ']

# precinct
replace_val(df, col='precinct', old_val=prec, new_val='Unknown')
# sector
replace_val(df, col='sector', old_val=sect, new_val='Unknown')
# beat
replace_val(df, col='beat', old_val=beats, new_val='Unknown')

df['precinct'].value_counts(), df['sector'].value_counts(), df['beat']
```

Out[48]:
```
(precinct
 West          16657
 North         12693
 Unknown       10780
 East           8159
 South          7290
 Southwest      4673
 Name: count, dtype: int64,
 sector
 Unknown    10739
 K           5597
 M           5091
 E           4284
 N           3554
 D           3427
 F           2815
 R           2763
 B           2733
 Q           2538
 L           2466
 O           2326
 S           2200
 U           2199
 G           2084
 W           1856
 C           1791
 J           1739
 OOJ           50
 Name: count, dtype: int64,
 beat
 Unknown    10783
 K3          3209
 M3          2472
 E2          1789
 N3          1761
 E1          1439
 M2          1349
 D1          1346
 N2          1340
 K2          1303
 R2          1286
 D2          1278
 M1          1273
 Q3          1213
 F2          1159
 K1          1085
 E3          1054
 B2          1042
 B1          1018
 U2          1015
 O1           945
 S2           857
 L2           852
 F3           836
 F1           820
 L1           816
 D3           803
 R1           801
 L3           798
 W2           791
 G2           780
```

```
U1           751
O3           747
Q2           746
S3           733
C1           726
G3           709
B3           677
R3           676
J3           657
J1           639
O2           634
C3           623
S1           610
W1           608
G1           594
Q1           579
W3           458
N1           451
C2           443
J2           443
U3           433
S              2
Name: count, dtype: int64)
```

Final Check

Ok so let us now check what we have done with our dataset before proceeding further.

In [49]:
```
# Checking the copy dataframe we have cleaned up
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 60252 entries, 0 to 61007
Data columns (total 26 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   subject_age_group          60252 non-null  object
 1   subject_id                 60252 non-null  object
 2   go_sc_num                  60252 non-null  int64 
 3   terry_stop_id              60252 non-null  int64 
 4   stop_resolution            60252 non-null  object
 5   weapon_type                28175 non-null  object
 6   officer_id                 60252 non-null  object
 7   officer_gender             60252 non-null  object
 8   officer_race               60252 non-null  object
 9   subject_perceived_race     60252 non-null  object
 10  subject_perceived_gender   60252 non-null  object
 11  initial_call_type          60252 non-null  object
 12  final_call_type            60252 non-null  object
 13  call_type                  60252 non-null  object
 14  arrest_flag                60252 non-null  object
 15  frisk_flag                 60252 non-null  object
 16  precinct                   60252 non-null  object
 17  sector                     60252 non-null  object
 18  beat                       60252 non-null  object
 19  repeat_offenders           60252 non-null  object
 20  incident_year              60252 non-null  int32 
 21  incident_month             60252 non-null  int32 
 22  officer_age                60252 non-null  int64 
 23  field_contact              60252 non-null  object
 24  offense_report             60252 non-null  object
 25  reported_hour              60252 non-null  int32 
dtypes: int32(3), int64(3), object(20)
memory usage: 11.7+ MB
```

In [50]:
```
# Fixing the format of subject age group values
replace_val(df, 'subject_age_group', ['26 - 35'], '26_35')
replace_val(df, 'subject_age_group', ['18 - 25'], '18_25')
replace_val(df, 'subject_age_group', ['36 - 45'], '36_45')
replace_val(df, 'subject_age_group', ['46 - 55'], '46_55')
replace_val(df, 'subject_age_group', ['56 and Above'], '56_up')
replace_val(df, 'subject_age_group', ['1 - 17'], '1_17')


df['subject_age_group'].value_counts()
```

Out[50]:
```
subject_age_group
26_35      20163
36_45      13474
18_25      11414
46_55       7651
56_up       3189
1_17        2263
Unknown     2098
Name: count, dtype: int64
```

With that we say we have cleaned up the dataset for EDA and modeling.

Feature Engineering

Same Races

Here we'll make a binary column 'same_race' that displays as 1 if the officer and the subject were of the same race and 0 if they are of different.

To accomplish this, we need to make sure that the categories in 'Officer_Race' and 'Subject_Perceived_Race' have the same values and make any necessary changes.

In [51]:
```
# Checking the values of both columns
races = df[['officer_race', 'subject_perceived_race']]
col_values(races)
```

```
officer_race
White                              43191
Two or More Races                   4195
Hispanic or Latino                  3982
Asian                               2872
Not Specified                       2827
Black or African American           2401
Nat Hawaiian/Oth Pac Islander        545
American Indian/Alaska Native        239
Name: count, dtype: int64
-------------------------------------------------------
subject_perceived_race
White                                        29548
Black or African American                    18119
Unknown                                       6108
Asian                                         2070
Hispanic                                      1666
American Indian or Alaska Native              1638
Multi-Racial                                   796
Native Hawaiian or Other Pacific Islander      158
Other                                          149
Name: count, dtype: int64
-------------------------------------------------------
```

Ok so we can see that we don't have the same values for both columns. The differences are (Hispanic or Latino, Hispanic), (American Indian/Alaska Native, American Indian or Alaska Native), (Two or More Races, Multi-Racial), (Nat Hawaiian/Oth Pac Islander, Other), and (Not Specified, Unknown). Lets sort this out.

In [52]:
```
# Aligning the column values
native = ['American Indian/Alaska Native', 'American Indian or Alaska
multi = ['Two or More Races']
other = ['Nat Hawaiian/Oth Pac Islander', 'Native Hawaiian or Other Pa
unknown = ['Unknown']
hispanic = ['Hispanic or Latino']

# native
replace_val(df, 'officer_race', native, 'Native American')
replace_val(df, 'subject_perceived_race', native, 'Native American')
# multi
replace_val(df, 'officer_race', multi, 'Multi-Racial')
# other
replace_val(df, 'officer_race', other, 'Other')
replace_val(df, 'subject_perceived_race', other, 'Other')
# unknown
replace_val(df, 'subject_perceived_race', unknown, 'Not Specified')
# hispanic
replace_val(df, 'officer_race', hispanic, 'Hispanic')

df.officer_race.unique()
```

Out[52]:
```
array(['Asian', 'White', 'Multi-Racial', 'Hispanic', 'Not Specifie
d',
       'Black or African American', 'Other', 'Native American'],
      dtype=object)
```

In [53]:
```
df.subject_perceived_race.unique()
```

Out[53]:
```
array(['White', 'Hispanic', 'Not Specified', 'Asian',
       'Black or African American', 'Native American', 'Multi-Racia
l',
       'Other'], dtype=object)
```

In [54]:
```
# Now that the values of the two fields are identical, let us create a
df['same_race'] = np.nan
for i in range(len(df['officer_race'])):
    if df['officer_race'].iloc[i] == df['subject_perceived_race'].iloc
        df['same_race'].iloc[i] = 'Y'
    else:
        df['same_race'].iloc[i] = 'N'

df['same_race'].value_counts()
```

Out[54]:
```
same_race
N    37263
Y    22989
Name: count, dtype: int64
```

Gender Race

Let us do the same thing with the officer gender and the subject gender. First let us make sure that the genders in both columns match.

In [55]:
```
# Matching both genders
male = ['Male']
female = ['Female']

replace_val(df, 'subject_perceived_gender', male, 'M')
replace_val(df, 'subject_perceived_gender', female, 'F')

# Now that the values of the two fields are identical, let us create a
df['same_gender'] = np.nan
for g in range(len(df['officer_gender'])):
    if df['officer_gender'].iloc[g] == df['subject_perceived_gender'].
        df['same_gender'].iloc[g] = 'Y'
    else:
        df['same_gender'].iloc[g] = 'N'

# df_copy['subject_perceived_gender']!= df_copy['officer_gender']

df['same_gender'].value_counts()
```

Out[55]:
```
same_gender
Y    43800
N    16452
Name: count, dtype: int64
```

In [56]:
```
# Creating a new dataframe for EDA
df_clean = df
df_clean.head()
```

Out[56]:

	subject_age_group	subject_id	go_sc_num	terry_stop_id	stop_resolution	weapon_type
0	36_45	unassigned	20160000398323	208373	Offense Report	NaN
1	18_25	unassigned	20180000227180	559146	Citation / Infraction	NaN
2	18_25	unassigned	20180000410091	498246	Offense Report	NaN
3	Unknown	unassigned	20160000001637	146742	Field Contact	NaN
4	46_55	unassigned	20150000006037	104477	Field Contact	NaN

5 rows × 28 columns

In [57]:
```
df_clean.columns
```

Out[57]:
```
Index(['subject_age_group', 'subject_id', 'go_sc_num', 'terry_stop_i
d',
       'stop_resolution', 'weapon_type', 'officer_id', 'officer_gend
er',
       'officer_race', 'subject_perceived_race', 'subject_perceived_
gender',
       'initial_call_type', 'final_call_type', 'call_type', 'arrest_
flag',
       'frisk_flag', 'precinct', 'sector', 'beat', 'repeat_offender
s',
       'incident_year', 'incident_month', 'officer_age', 'field_cont
act',
       'offense_report', 'reported_hour', 'same_race', 'same_gende
r'],
      dtype='object')
```

Exporting to CSV

We are done with cleaning, feature engineering and preprocessing the dataset. Let us export it to a new CSV file that we will use for EDA.

In [58]:
```
# #  Exporting to csv file
# df_clean.to_csv('data/clean_Terry_stops_data.csv', index=False)

# print('Data exported to clean_data.csv successfully.')
```