In [1]:

```python
# SECTION A : Structural investigation of the dataset. Here, an exploration of the general make-up of the dataset is done.

# STEP 1. we need to import some useful packages.

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import LabelEncoder
plt.style.use('ggplot') ## this makes plots look more presentable.
```

In [2]:

```python
# STEP 2: Load the dataset.

# Data Source : wwww.data.gov.
# Name : Catalogue.data.gov/dataset/traffic-collision-data-from-2010-to-present.
# Load the dataset.
Traffic_collision_df = pd.read_csv (r"C:\Users\Admin\Downloads\Traffic_Collision_Data_from_2010_to_Present.csv")
```

In [3]:

```python
# STEP 3: Understanding the dataset.

# 3.1. let's see the first 3 rows and all the columns.
Traffic_collision_df.head(3)
```

Out[3]:

	DR Number	Date Reported	Date Occurred	Time Occurred	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Street	L
0	190319651	08/24/2019	08/24/2019	450	3	Southwest	356	997	TRAFFIC COLLISION	3036 3004 3026 3101 4003	22.0	M	H	101.0	STREET	JEFFERSON BL	NORMANDIE AV	(3 11
1	190319680	08/30/2019	08/30/2019	2320	3	Southwest	355	997	TRAFFIC COLLISION	3037 3006 3028 3030 3039 3101	30.0	F	H	101.0	STREET	JEFFERSON BL	W WESTERN	(3 11

	DR Number	Date Reported	Date Occurred	Time Occurred	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Street	L
2	190413769	08/25/2019	08/25/2019	545	4	Hollenbeck	422	997	TRAFFIC COLLISION	4003 3101 3401 3701 3006 3030	NaN	M	X	101.0	STREET	N BROADWAY	W EASTLAKE AV	(3 11

In [4]:

```
# 3.2. let's fetch the last 3 rows and all the columns.
Traffic_collision_df.tail(3)
```

Out[4]:

	DR Number	Date Reported	Date Occurred	Time Occurred	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Street	L
596792	231513297	08/08/2023	08/08/2023	1430	15	N Hollywood	1535	997	TRAFFIC COLLISION	NaN	46.0	M	H	101.0	STREET	FARMDALE AV	OXNARD ST	(1
596793	231610826	08/06/2023	08/06/2023	1845	16	Foothill	1669	997	TRAFFIC COLLISION	3004 3028 4026 3034 3037 3101	50.0	M	H	101.0	STREET	TUJUNGA CANYON BL	LA TUNA CANYON RD	(3 1
596794	232112594	08/07/2023	08/07/2023	1017	21	Topanga	2126	997	TRAFFIC COLLISION	3006 3028 4026 3034 3037 3101	50.0	M	W	101.0	STREET	DEERING AV	SATICOY ST	(1

In [5]:

```
# 3.3. Show the size of the dataset.
Traffic_collision_df.shape
```

Out[5]:

```
(596795, 18)
```

In [6]:

```
# 3.4. In order to know the memory usage, data types and non-null values of the dataset , we will run the info command.
Traffic_collision_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 596795 entries, 0 to 596794
Data columns (total 18 columns):
 #   Column                  Non-Null Count   Dtype  
---  ------                  --------------   -----  
 0   DR Number               596795 non-null  int64  
 1   Date Reported           596795 non-null  object 
 2   Date Occurred           596795 non-null  object 
 3   Time Occurred           596795 non-null  int64  
 4   Area ID                 596795 non-null  int64  
 5   Area Name               596795 non-null  object 
 6   Reporting District      596795 non-null  int64  
 7   Crime Code              596795 non-null  int64  
 8   Crime Code Description  596795 non-null  object 
 9   MO Codes                509637 non-null  object 
 10  Victim Age              511212 non-null  float64
 11  Victim Sex              586844 non-null  object 
 12  Victim Descent          585906 non-null  object 
 13  Premise Code            595836 non-null  float64
 14  Premise Description     595835 non-null  object 
 15  Address                 596795 non-null  object 
 16  Cross Street            568562 non-null  object 
 17  Location                596795 non-null  object 
dtypes: float64(2), int64(5), object(11)
memory usage: 82.0+ MB
```

In [7]:

```
# 3.5. OBSERVATIONS : # 1. Columns 'Date Reported'and'Date Occurred' should be in datetime format.
                       # 2. The values in column 'Time Occurred' need to be converted to AM and PM for better understanding.
                       # 3. The values in column 'Victim age' need to be converted from float to integer.
                       # 4. The values in column 'Premise code' should be converted from float to integer as well.
                       # 5. Column 'Location' should be float data type not object data type.
                       # 6. The series 'Location' can be split into Latitude and longitude Columns.
```

In [8]:

```
# 3.6. Let's take a look at all the columns, to see if there is any spelling mistake.
Traffic_collision_df.columns
```

Out[8]:

```
Index(['DR Number', 'Date Reported', 'Date Occurred', 'Time Occurred',
       'Area ID', 'Area Name', 'Reporting District', 'Crime Code',
       'Crime Code Description', 'MO Codes', 'Victim Age', 'Victim Sex',
       'Victim Descent', 'Premise Code', 'Premise Description', 'Address',
       'Cross Street', 'Location'],
      dtype='object')
```

In [9]:

```
# 3.7. Structure of non-numerical features.

# let’s take a closer look at the non-numerical entries before we start reassigning the data types. Note that column location is v
iewed as object data type as denoted from the original dataset.

# Display non-numerical features

Traffic_collision_df.select_dtypes(exclude="number").head(3)
```

Out[9]:

	Date Reported	Date Occurred	Area Name	Crime Code Description	MO Codes	Victim Sex	Victim Descent	Premise Description	Address	Cross Street	Location
0	08/24/2019	08/24/2019	Southwest	TRAFFIC COLLISION	3036 3004 3026 3101 4003	M	H	STREET	JEFFERSON BL	NORMANDIE AV	(34.0255, -118.3002)
1	08/30/2019	08/30/2019	Southwest	TRAFFIC COLLISION	3037 3006 3028 3030 3039 3101 4003	F	H	STREET	JEFFERSON BL	W WESTERN	(34.0256, -118.3089)
2	08/25/2019	08/25/2019	Hollenbeck	TRAFFIC COLLISION	3101 3401 3701 3006 3030	M	X	STREET	N BROADWAY	W EASTLAKE AV	(34.0738, -118.2078)

In [10]:

```
# 3.8. we can also investigate how many unique values each non-numerical feature has and with which frequency the most prominent v
alue is present.

Traffic_collision_df.describe(exclude="number").head(3)
```

Out[10]:

	Date Reported	Date Occurred	Area Name	Crime Code Description	MO Codes	Victim Sex	Victim Descent	Premise Description	Address	Cross Street	Location
count	596795	596795	596795	596795	509637	586844	585906	595835	596795	568562	596795
unique	5000	5000	21	1	109962	6	20	122	28784	21151	52831
top	04/12/2018	11/17/2017	77th Street	TRAFFIC COLLISION	0605	M	H	STREET	WESTERN AV	VERMONT AV	(0.0, 0.0)

In [11]:

```
# NB: There are some data types that were not properly assigned from the original dataset, that will be taken care of during the d
ata cleaning and preparation stage.
```

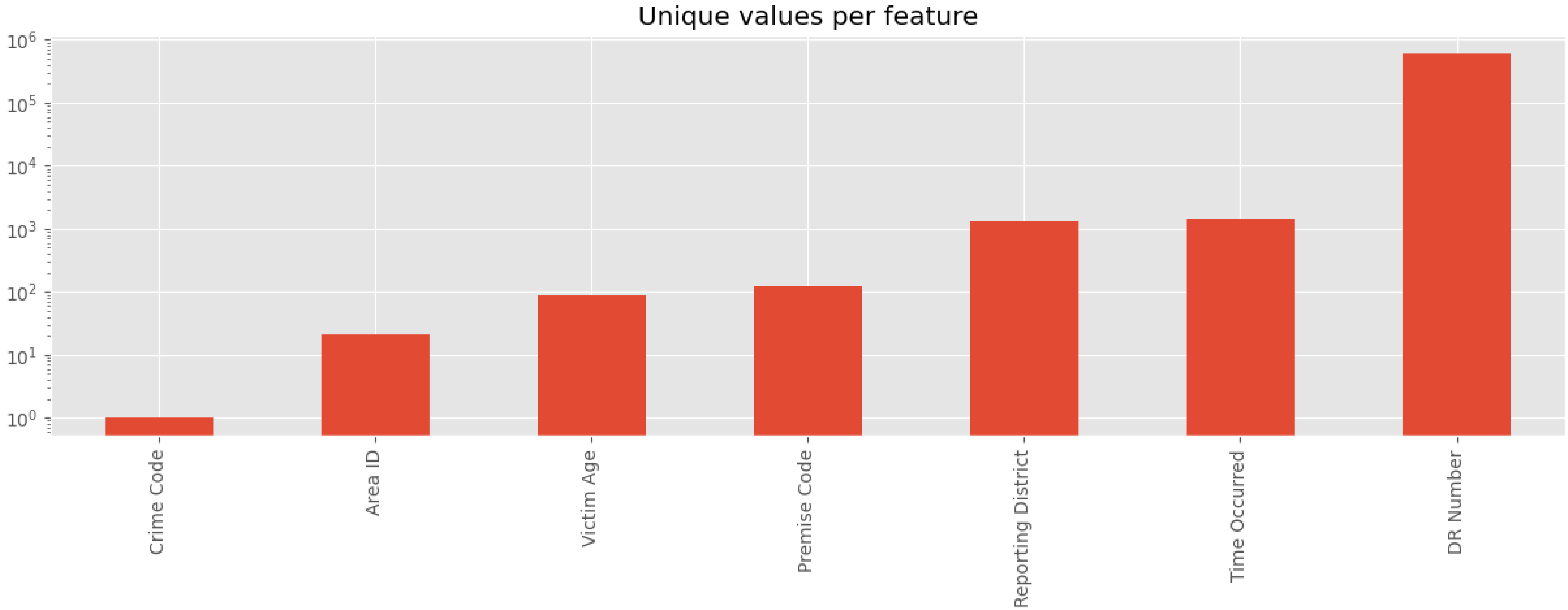
In [12]:

```
# 3.9. Structure of numerical features.

# Next, let’s take a closer look at the numerical features. More precisely, let’s investigate how many unique values each of these
```

```python
features possess.

# For each numerical feature compute number of unique entries
unique_values = Traffic_collision_df.select_dtypes(include="number").nunique().sort_values()

# Plot information with y-axis in log-scale
unique_values.plot.bar(logy=True, figsize=(15, 4), title="Unique values per feature")
```

Out[12]:

```
<Axes: title={'center': 'Unique values per feature'}>
```



In [13]:

```python
# 3.10. Let us have an overview of the statistical summary all the numerical data types in the series before the commencement of d
ata cleaning.
Traffic_collision_df.describe().T
```

Out[13]:

	count	mean	std	min	25%	50%	75%	max
DR Number	596795.0	1.581883e+08	3.462127e+07	100100007.0	130716258.0	160808043.0	182015597.5	239922845.0

	count	mean	std	min	25%	50%	75%	max
Time Occurred	596795.0	1.354249e+03	6.025250e+02	1.0	930.0	1450.0	1820.0	2359.0
Area ID	596795.0	1.107768e+01	5.879268e+00	1.0	6.0	11.0	16.0	21.0
Reporting District	596795.0	1.153704e+03	5.890812e+02	100.0	666.0	1162.0	1653.0	2199.0
Crime Code	596795.0	9.970000e+02	0.000000e+00	997.0	997.0	997.0	997.0	997.0
Victim Age	511212.0	4.130854e+01	1.657334e+01	10.0	28.0	38.0	51.0	99.0
Premise Code	595836.0	1.024388e+02	2.358909e+01	101.0	101.0	101.0	101.0	970.0

In [14]:

```
# 3.11. Conclusion of structure investigation.

# At the end of this first investigation, we have a better understanding of the general structure of our dataset, what kind of dat
a type each feature has, and those to be reassigned to a different data types.
```

In [15]:

```
# SECTION B: Quality Investigation of the dataset.

# In this section, the dataset will be cleaned and prepared for further onward analysis.
```

In [16]:

```
# STEP 4. Data cleaning and preparation.

# From the observation above : # 1. Columns 'Date Reported'and'Date Occurred' should be in datetime format.
                               # 2. The values in column 'Time Occurred' need to be converted to AM and PM for better understandin
g. let's group these times in Period of the day.
                               # 3. The values in column 'Victim age' need to be converted from float to integer.
                               # 4. The values in column 'Premise code' should be converted from float to integer as well.
                               # 5. Column 'Location' should be float data type not object data type.
                               # 6. The series 'Location' can be split into Latitude and longitude Columns.
```

In [17]:

```
# 4.1. Columns 'Date Reported'and 'Date Occurred' should be in datetime format.

Traffic_collision_df['Date Occurred'] = pd.to_datetime(Traffic_collision_df['Date Occurred']).dt.strftime('%d-%m-%Y')

Traffic_collision_df['Date Reported'] = pd.to_datetime(Traffic_collision_df['Date Reported']).dt.strftime('%d-%m-%Y')
```

In [18]:

```
Traffic_collision_df.head(3)
```

Out[18]:

	DR Number	Date Reported	Date Occurred	Time Occurred	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Street	Loca
0	190319651	24-08-2019	24-08-2019	450	3	Southwest	356	997	TRAFFIC COLLISION	3036 3004 3026 3101 4003	22.0	M	H	101.0	STREET	JEFFERSON BL	NORMANDIE AV	(34.0 118.3
1	190319680	30-08-2019	30-08-2019	2320	3	Southwest	355	997	TRAFFIC COLLISION	3037 3006 3028 3030 3039 3101 4003	30.0	F	H	101.0	STREET	JEFFERSON BL	W WESTERN	(34.0 118.3
2	190413769	25-08-2019	25-08-2019	545	4	Hollenbeck	422	997	TRAFFIC COLLISION	3101 3401 3701 3006 3030	NaN	M	X	101.0	STREET	N BROADWAY	W EASTLAKE AV	(34.0 118.2

In [19]:

```
# 4.2. The values found in column 'Time Occurred' need to be converted to AM and PM.
Traffic_collision_df['Time Occurred']= Traffic_collision_df['Time Occurred'].apply(lambda x:f"{x // 100 if x // 100 <= 12 else (x
// 100) - 12 } :{x % 100 :02d} {'AM' if x < 1200 else 'PM'}")
```

In [20]:

```
Traffic_collision_df.head(3)
```

Out[20]:

	DR Number	Date Reported	Date Occurred	Time Occurred	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Street	Loca
0	190319651	24-08-2019	24-08-2019	4 :50 AM	3	Southwest	356	997	TRAFFIC COLLISION	3036 3004 3026 3101 4003	22.0	M	H	101.0	STREET	JEFFERSON BL	NORMANDIE AV	(34.0 118.3
1	190319680	30-08-2019	30-08-2019	11 :20 PM	3	Southwest	355	997	TRAFFIC COLLISION	3037 3006 3028 3030 3039 3101	30.0	F	H	101.0	STREET	JEFFERSON BL	W WESTERN	(34.0 118.3

	DR Number	Date Reported	Date Occurred	Time Occurred	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Street	Loca
2	190413769	25-08-2019	25-08-2019	5 :45 AM	4	Hollenbeck	422	997	TRAFFIC COLLISION	3101 3401 3701 3006 3030	NaN	M	X	101.0	STREET	N BROADWAY	W EASTLAKE AV	(34.0 118.2

In [21]:

```
# 4.3. Let us group the 'Time Occurred' into Morning, Afternoon And Night, and name the new column Period Of The Day.

def convert_to_time_of_day(time_str):
    time_str_lower = time_str.lower()
    if "am" in time_str_lower:
        return "morning"
    elif "pm" in time_str_lower:
        hour = int(time_str_lower.split(":")[0])
        if hour < 6:
            return "afternoon"
        else:
            return "night"
    else:
        return "unknown"

# Apply the conversion function to the 'Time Occurred' column
Traffic_collision_df['Period Of the Day'] = Traffic_collision_df['Time Occurred'].apply(convert_to_time_of_day)
```

In [22]:

```
Traffic_collision_df.head(3)
```

Out[22]:

	DR Number	Date Reported	Date Occurred	Time Occurred	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Street	Loca
0	190319651	24-08-2019	24-08-2019	4 :50 AM	3	Southwest	356	997	TRAFFIC COLLISION	3036 3004 3026 3101 4003	22.0	M	H	101.0	STREET	JEFFERSON BL	NORMANDIE AV	(34.0 118.3
1	190319680	30-08-2019	30-08-2019	11 :20 PM	3	Southwest	355	997	TRAFFIC COLLISION	3037 3006 3028 3030 3039 3101	30.0	F	H	101.0	STREET	JEFFERSON BL	W WESTERN	(34.0 118.3

	DR Number	Date Reported	Date Occurred	Time Occurred	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Street	Loca
										4003								
2	190413769	25-08-2019	25-08-2019	5 :45 AM	4	Hollenbeck	422	997	TRAFFIC COLLISION	3101 3401 3701 3006 3030	NaN	M	X	101.0	STREET	N BROADWAY	W EASTLAKE AV	(34.0 118.2

In [23]:

```python
# 4.4. Reorder the position of the Column 'Period Of The Day'

# Get the name of the column to move
column_to_move = 'Period Of the Day'

# Get the current column order
columns = Traffic_collision_df.columns.tolist()

# Move the column to the desired position
new_position = 4
columns.insert(new_position, columns.pop(columns.index('Period Of the Day')))

# Reorder the DataFrame columns
Traffic_collision_df = Traffic_collision_df[columns]
```

In [24]:

```python
Traffic_collision_df.head(3)
```

Out[24]:

	DR Number	Date Reported	Date Occurred	Time Occurred	Period Of the Day	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Str
0	190319651	24-08-2019	24-08-2019	4 :50 AM	morning	3	Southwest	356	997	TRAFFIC COLLISION	3036 3004 3026 3101 4003	22.0	M	H	101.0	STREET	JEFFERSON BL	NORMAN
1	190319680	30-08-2019	30-08-2019	11 :20 PM	night	3	Southwest	355	997	TRAFFIC COLLISION	3037 3006 3028 3030 3039 3101 4003	30.0	F	H	101.0	STREET	JEFFERSON BL	W WESTE
2	190413769	25-08-	25-08-	5 :45 AM	morning	4	Hollenbeck	422	997	TRAFFIC	3101 3401 3701	NaN	M	X	101.0	STREET	N	EASTLA

	DR Number	Date Reported	Date Occurred	Time Occurred	Period Of the Day	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Sti

In [25]:

```python
# 4.5. The values found in column 'Victim age' need to be converted from float to integer.

# Note that some of the values have NAN, so we have to use the "fillna" command.

Traffic_collision_df['Victim Age']= Traffic_collision_df['Victim Age'].fillna(0).astype(int)
```

In [26]:

```python
Traffic_collision_df.head(3)
```

Out[26]:

	DR Number	Date Reported	Date Occurred	Time Occurred	Period Of the Day	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Sti
0	190319651	24-08-2019	24-08-2019	4 :50 AM	morning	3	Southwest	356	997	TRAFFIC COLLISION	3036 3004 3026 3101 4003	22	M	H	101.0	STREET	JEFFERSON BL	NORMAN
1	190319680	30-08-2019	30-08-2019	11 :20 PM	night	3	Southwest	355	997	TRAFFIC COLLISION	3037 3006 3028 3030 3039 3101 4003	30	F	H	101.0	STREET	JEFFERSON BL	W WESTE
2	190413769	25-08-2019	25-08-2019	5 :45 AM	morning	4	Hollenbeck	422	997	TRAFFIC COLLISION	3101 3401 3701 3006 3030	0	M	X	101.0	STREET	N BROADWAY	EASTLA

In [27]:

```python
# 4.6. The values found in column 'Premise Code' need to be converted from float to integer.

# Note that some of the values have NAN, so we have to use the "fillna" command.

Traffic_collision_df['Premise Code']= Traffic_collision_df['Premise Code'].fillna(0).astype(int)
```

In [28]:

```
Traffic_collision_df.head(3)
```

Out[28]:

	DR Number	Date Reported	Date Occurred	Time Occurred	Period Of the Day	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Str
0	190319651	24-08-2019	24-08-2019	4 :50 AM	morning	3	Southwest	356	997	TRAFFIC COLLISION	3036 3004 3026 3101 4003	22	M	H	101	STREET	JEFFERSON BL	NORMAN
1	190319680	30-08-2019	30-08-2019	11 :20 PM	night	3	Southwest	355	997	TRAFFIC COLLISION	3037 3006 3028 3030 3039 3101 4003	30	F	H	101	STREET	JEFFERSON BL	W WESTE
2	190413769	25-08-2019	25-08-2019	5 :45 AM	morning	4	Hollenbeck	422	997	TRAFFIC COLLISION	3101 3401 3701 3006 3030	0	M	X	101	STREET	N BROADWAY	EASTL

In [29]:

```
# 4.7. Column 'Location' should be float data type not object data type.

# The Column 'Location' can be split into Latitude and longitude Columns for better and easier analysis.

# It is better we split the values found in column 'location' into two different columns, Latitude and Longitude, respectively.

Traffic_collision_df[['Latitude','Longitude']] = Traffic_collision_df ['Location'].str.strip('()').str.split (',',expand = True)
```

In [30]:

```
Traffic_collision_df.head(3)
```

Out[30]:

	DR Number	Date Reported	Date Occurred	Time Occurred	Period Of the Day	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	...	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Street

	DR Number	Date Reported	Date Occurred	Time Occurred	Period Of the Day	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	...	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Street
0	190319651	24-08-2019	24-08-2019	4 :50 AM	morning	3	Southwest	356	997	TRAFFIC COLLISION	...	22	M	H	101	STREET	JEFFERSON BL	NORMANDIE AV
1	190319680	30-08-2019	30-08-2019	11 :20 PM	night	3	Southwest	355	997	TRAFFIC COLLISION	...	30	F	H	101	STREET	JEFFERSON BL	W WESTERN
2	190413769	25-08-2019	25-08-2019	5 :45 AM	morning	4	Hollenbeck	422	997	TRAFFIC COLLISION	...	0	M	X	101	STREET	N BROADWAY	W EASTLAKE AV

3 rows × 21 columns

In [31]:

```
# 4.8. Convert Columns 'Latitude'and'Longitude' from object to float.

Traffic_collision_df [['Latitude','Longitude']] = Traffic_collision_df [['Latitude','Longitude']] .astype(float)
```

In [32]:

```
# 4.9. drop Column 'Location'

Traffic_collision_df.drop(columns= ['Location'] , inplace = True)
```

In [33]:

```
Traffic_collision_df.head(3)
```

Out[33]:

	DR Number	Date Reported	Date Occurred	Time Occurred	Period Of the Day	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross St
0	190319651	24-08-2019	24-08-2019	4 :50 AM	morning	3	Southwest	356	997	TRAFFIC COLLISION	3036 3004 3026 3101 4003	22	M	H	101	STREET	JEFFERSON BL	NORMAN
1	190319680	30-08-2019	30-08-2019	11 :20 PM	night	3	Southwest	355	997	TRAFFIC COLLISION	3037 3006 3028 3030 3039 3101 4003	30	F	H	101	STREET	JEFFERSON BL	W WESTE
		25-08-	25-08-							TRAFFIC	3101 3401						N	

	DR Number	Date Reported	Date Occurred	Time Occurred	Period Of the Day	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross St
2	190413769	25-08-2019	25-08-2019	5 :45 AM	morning	4	Hollenbeck	422	997	TRAFFIC COLLISION	3701 3006 3030	0	M	X	101	STREET	N BROADWAY	EASTLA

In [34]:

```
# STEP 5. Data Preparation.

#  check for missing data.
#  check for duplicates.
#  check for outliers.
#  Numerical features.
#  Non-numerical features.
```

In [35]:

```
# 5.1 check for missing values.

Traffic_collision_df.isna().sum()
```

Out[35]:

```
DR Number                     0
Date Reported                 0
Date Occurred                 0
Time Occurred                 0
Period Of the Day             0
Area ID                       0
Area Name                     0
Reporting District            0
Crime Code                    0
Crime Code Description        0
MO Codes                  87158
Victim Age                    0
Victim Sex                 9951
Victim Descent            10889
Premise Code                  0
Premise Description         960
Address                       0
Cross Street              28233
Latitude                      0
Longitude                     0
dtype: int64
```

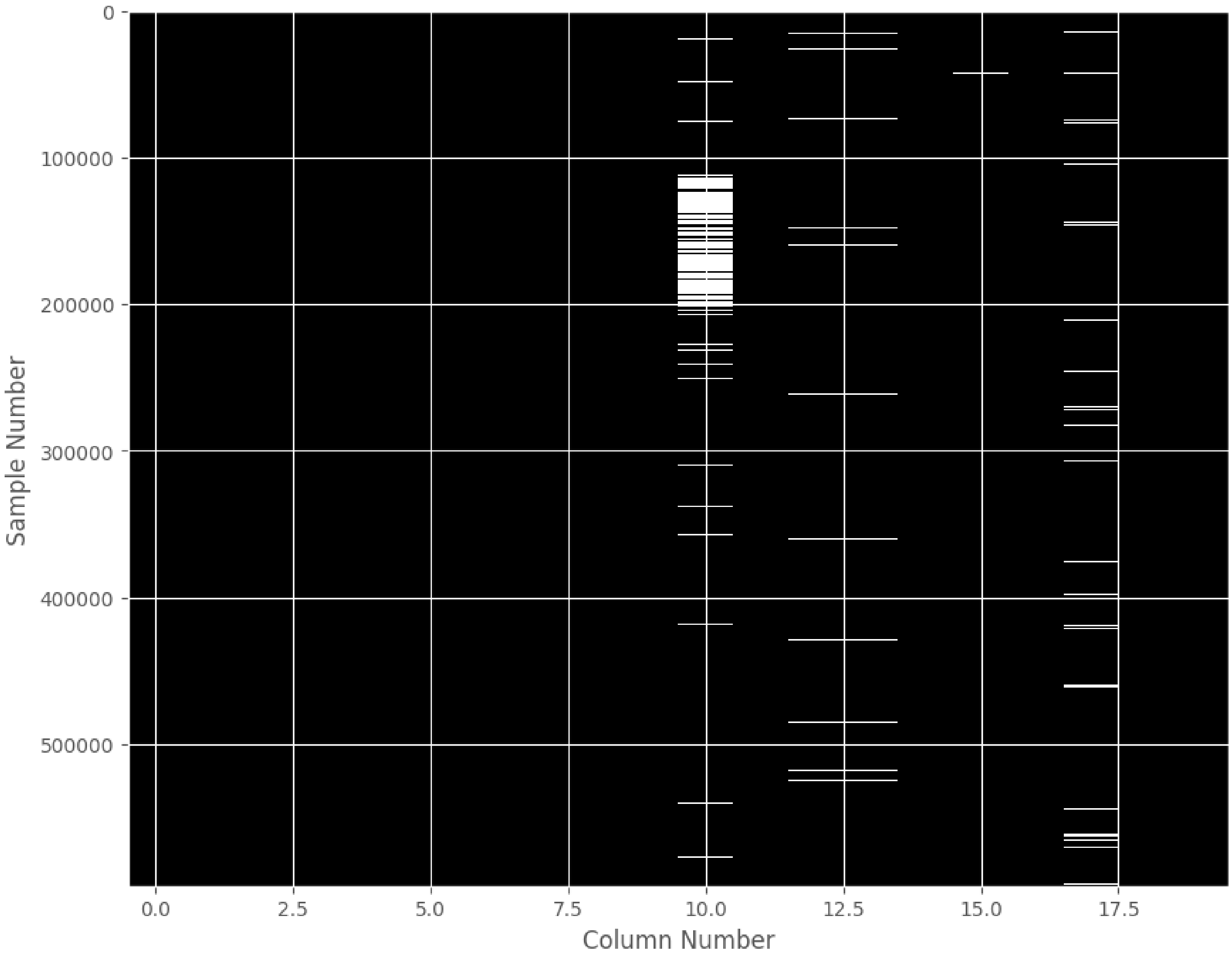
In [36]:

```
# 5.2. check for missing data Per sample.

# To look at number of missing data per sample, we simply visualize the output of Traffic_collision_df_X.isnull().
```

```
plt.figure(figsize=(10, 8))
plt.imshow(Traffic_collision_df.isnull(), aspect="auto", interpolation="nearest", cmap="gray")
plt.xlabel("Column Number")
plt.ylabel("Sample Number")
```

Out[36]:

```
Text(0, 0.5, 'Sample Number')
```

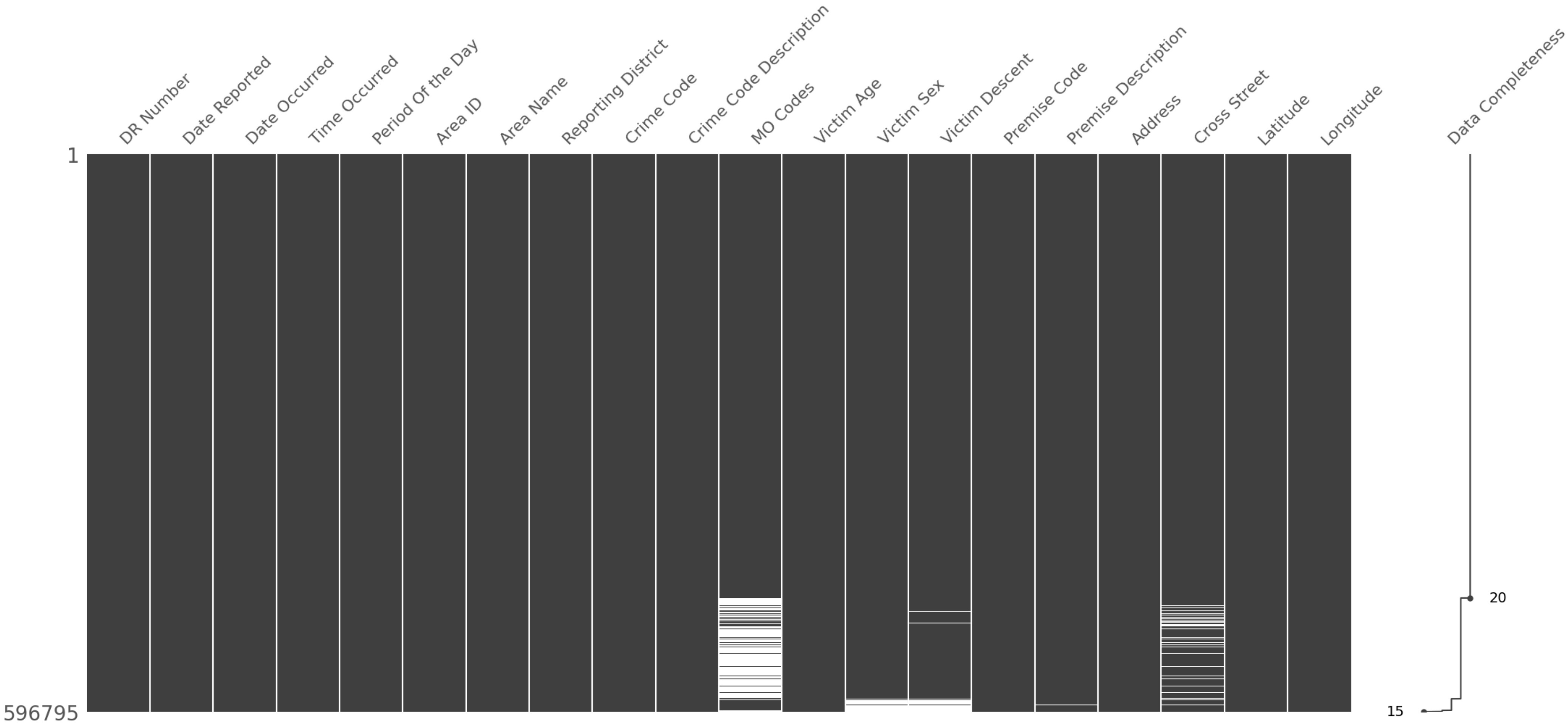


In [37]:

```
# 5.3. For a better visualization, let's use missingno command.

import missingno as msno

msno.matrix(Traffic_collision_df, labels=True, sort="descending")
```

Out[37]:

```
<Axes: >
```



In [38]:

```
# Looking the result above, one can see the the columns with and locations of null values.
```

In [39]:

```
Traffic_collision_df.shape
```

Out[39]:

```
(596795, 20)
```

In [40]:

```
# 5.4. Let's find the percentage of  some null data.

# 1: Percentage of missing data in Column'MO Codes'.

len(Traffic_collision_df)
Count_MO_Codes = Traffic_collision_df['MO Codes'].count()
Null_Count_MO_Codes= Traffic_collision_df['MO Codes'].isnull().sum()
percentage_of_null_values_of_MO_Codes  = Null_Count_MO_Codes/len(Traffic_collision_df) *100
print(percentage_of_null_values_of_MO_Codes , '  is the percentage_of_null_values_of_MO_Codes ' )
```

```
14.604344875543529   is the percentage_of_null_values_of_MO_Codes
```

In [41]:

```
# 2: Percentage of missing data in Column'Victim Sex'.

len(Traffic_collision_df)
Count_Victim_Sex = Traffic_collision_df['Victim Sex'].count()
Null_Count_Victim_Sex= Traffic_collision_df['Victim Sex'].isnull().sum()
percentage_of_null_values_of_Victim_Sex  = Null_Count_Victim_Sex/len(Traffic_collision_df) *100
print(percentage_of_null_values_of_Victim_Sex , '  is the percentage_of_null_values_of_Victim_Sex ' )
```

```
1.6674067309545153   is the percentage_of_null_values_of_Victim_Sex
```

In [42]:

```
# 3: Percentage of missing data in Column 'Victim Descent'.

len(Traffic_collision_df)
Count_Victim_descent = Traffic_collision_df['Victim Descent'].count()
Null_Count_Victim_Descent= Traffic_collision_df['Victim Descent'].isnull().sum()
percentage_of_null_values_of_Victim_Descent  = Null_Count_Victim_Descent/len(Traffic_collision_df) *100
print(percentage_of_null_values_of_Victim_Descent , '  is the percentage_of_null_values_of_Victim_Descent ' )
```

```
1.8245796295210246   is the percentage_of_null_values_of_Victim_Descent
```

In [43]:

```
# 4: Percentage of missing data in Column 'Premise Description'.

len(Traffic_collision_df)
Count_Premise_Description = Traffic_collision_df['Premise Description'].count()
```

```
Null_Count_Premise_Description= Traffic_collision_df['Premise Description'].isnull().sum()
percentage_of_null_values_of_Premise_Description  = Null_Count_Premise_Description/len(Traffic_collision_df) *100
print(percentage_of_null_values_of_Premise_Description , '  is the percentage_of_null_values_of_Premise_Description ' )
```

```
0.16085925652862373   is the percentage_of_null_values_of_Premise_Description
```

In [44]:

```
# 5: Percentage of missing data in Column 'Cross Street'.

len(Traffic_collision_df)
Count_Cross_Street = Traffic_collision_df['Cross Street'].count()
Null_Count_Cross_Street= Traffic_collision_df['Cross Street'].isnull().sum()
percentage_of_null_values_of_Cross_Street = Null_Count_Cross_Street/len(Traffic_collision_df) *100
print(percentage_of_null_values_of_Cross_Street , '  is the percentage_of_null_values_of_Cross_Street ' )
```

```
4.730770197471493   is the percentage_of_null_values_of_Cross_Street
```

In [45]:

```
# 5.5. let’s go ahead and drop rows that have missing values. The threshold 15% - 20% is inspired by the information from the ‘Dat
a Completeness’ column on the extreme right of msno.matrix.

Traffic_collision_df.dropna(inplace=True)
Traffic_collision_df.shape
```

Out[45]:

```
(475108, 20)
```

In [46]:

```
# 5.6. check for duplicates.

Traffic_collision_df.duplicated().sum()
```

Out[46]:

```
0
```

In [47]:

```
# NB: There are no duplicates.
```

In [48]:

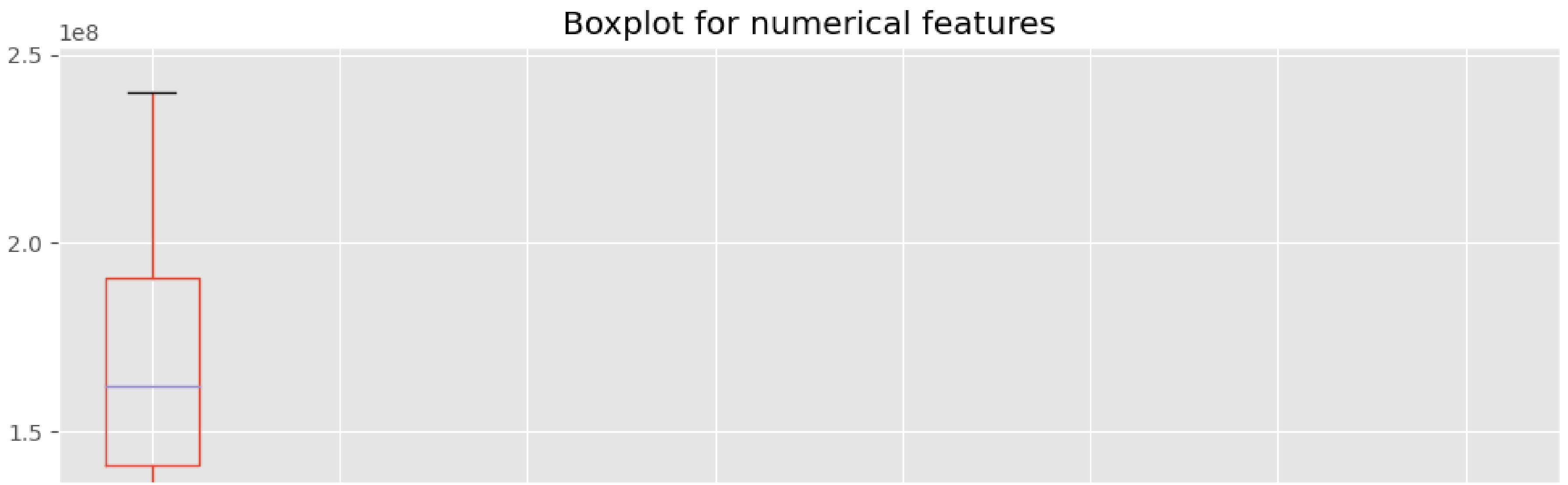
```
# 5.7. check for unique values.

Traffic_collision_df.nunique()
```

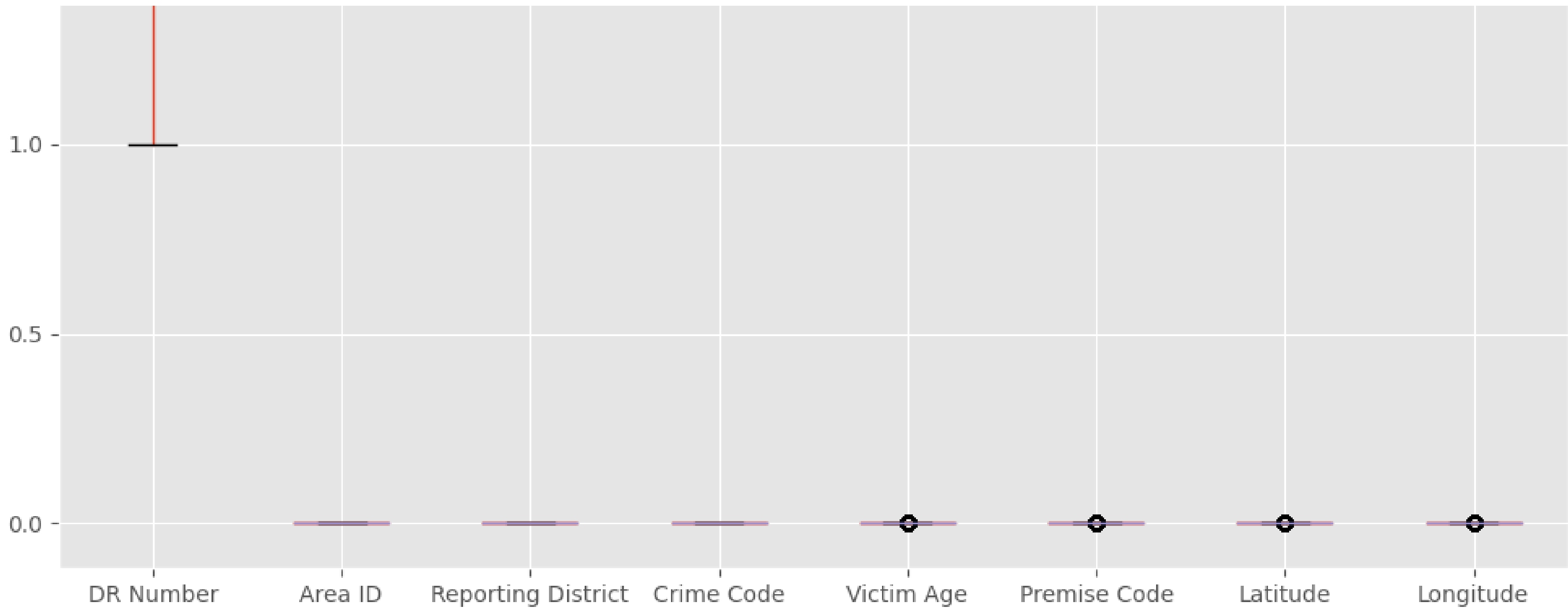
Out[48]:

```
DR Number                  475108
Date Reported                5000
Date Occurred                5000
Time Occurred                1439
Period Of the Day               3
Area ID                        21
Area Name                      21
Reporting District           1326
Crime Code                      1
Crime Code Description          1
MO Codes                   103822
Victim Age                     91
Victim Sex                      6
Victim Descent                 20
Premise Code                   93
Premise Description            93
Address                     13246
Cross Street                19301
Latitude                     4878
Longitude                    4707
dtype: int64
```

In [49]:

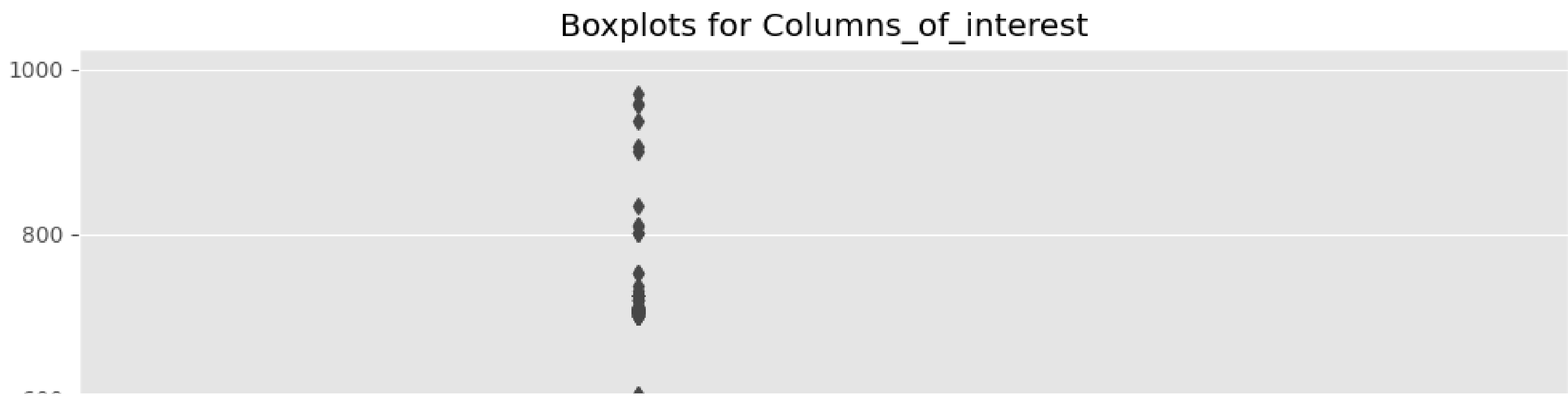
```
# 5.8. check for outliers.

# To do this, we need to draw a box plot for the numerical features in the entire dataset.

Traffic_collision_df.boxplot(figsize=(12,8))
plt.title('Boxplot for numerical features')
plt.show()
```

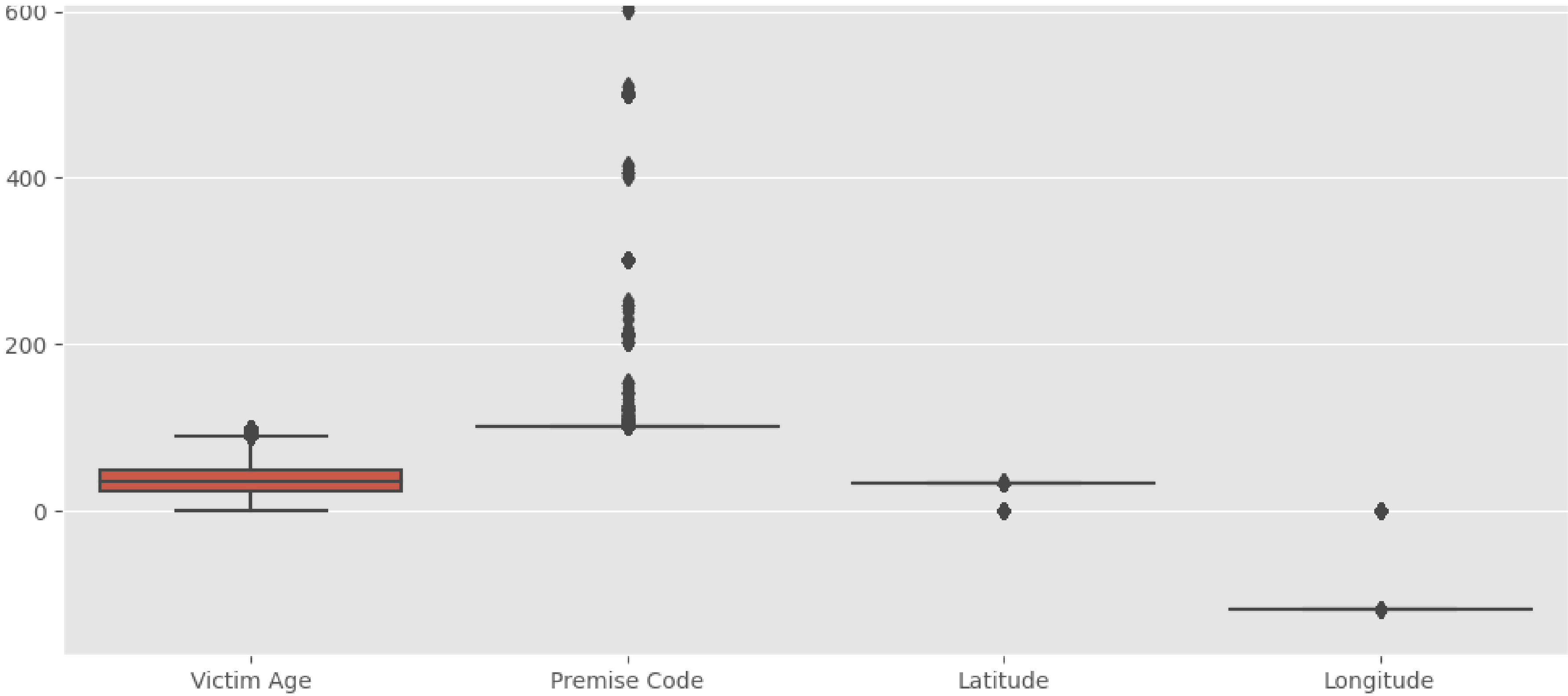




In [50]:

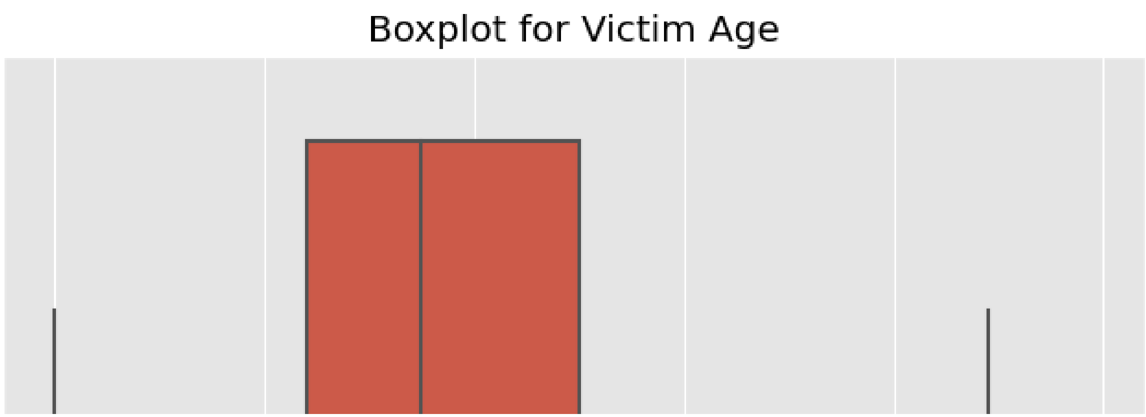
```
# 5.9. let's dig deeper by selecting some columns that we are interested in.

columns_of_interest = [ 'Victim Age' ,'Premise Code', 'Latitude','Longitude']

# Draw boxplots for the specified columns using seaborn
plt.figure(figsize=(12, 8))
sns.boxplot(data=Traffic_collision_df[columns_of_interest])
plt.title('Boxplots for Columns_of_interest')
plt.show()
```

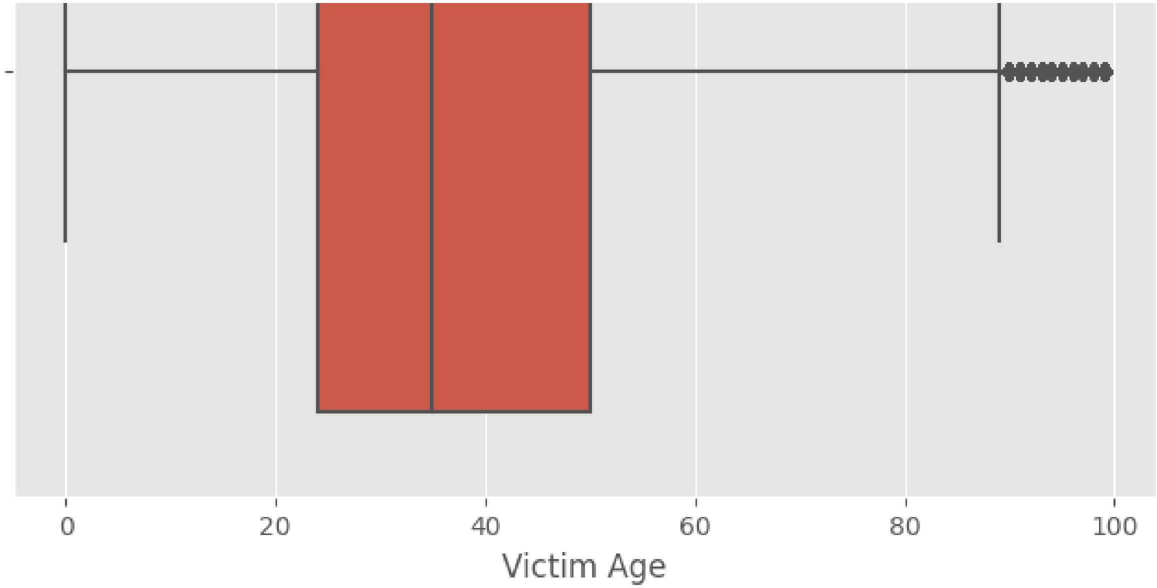




In [51]:

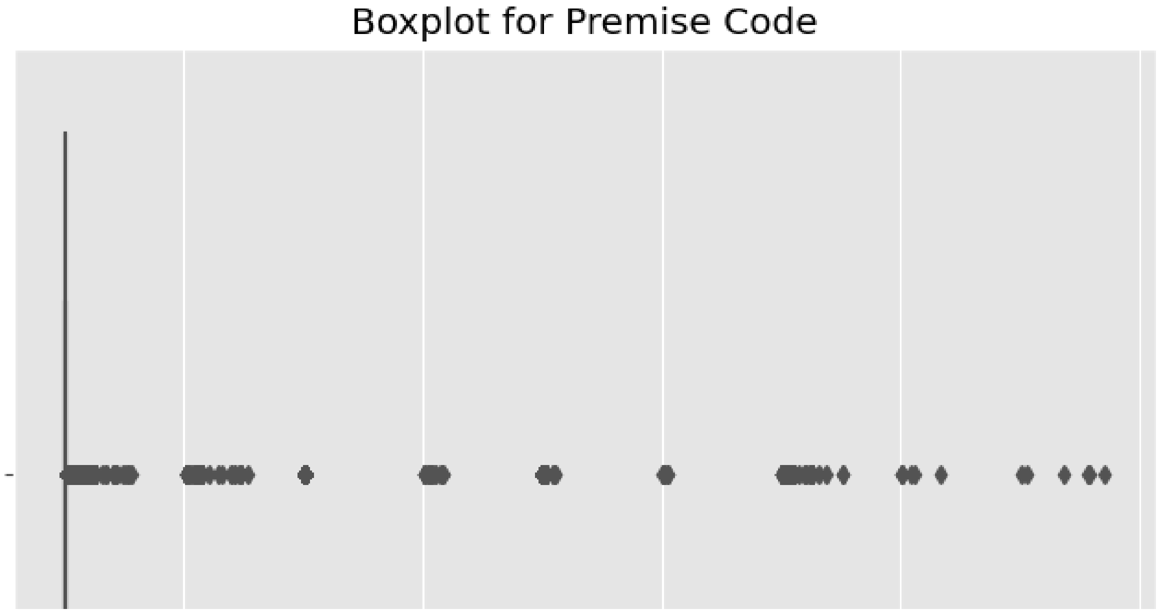
```
# 5.10. Draw boxplot for Victim Age.

Victim_Age = Traffic_collision_df['Victim Age']
plt.figure(figsize=(8, 6))
sns.boxplot(x=Traffic_collision_df['Victim Age'])
plt.title('Boxplot for Victim Age')
plt.show()
```

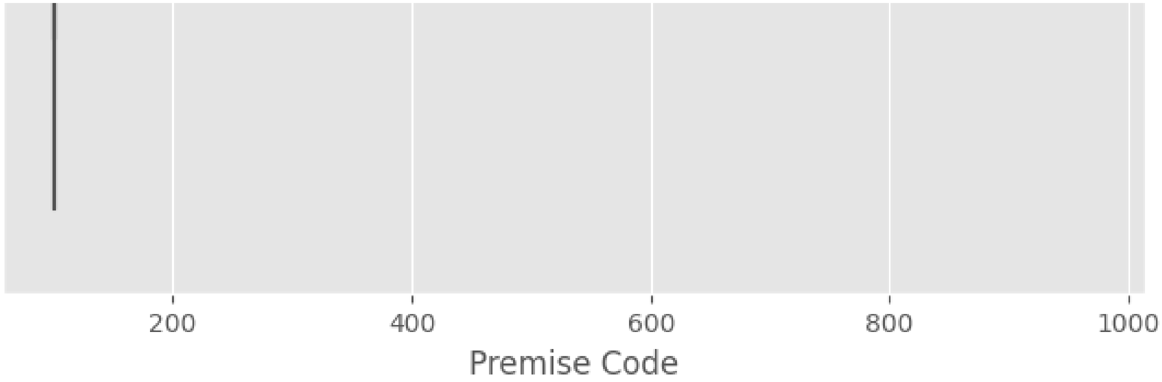




In [52]:

```
# 5.11.  Draw a boxplot for  Premise Code.

Premise_Code = Traffic_collision_df['Premise Code']
plt.figure(figsize=(8, 6))
sns.boxplot(x=Traffic_collision_df['Premise Code'])
plt.title('Boxplot for Premise Code')
plt.show()
```





In [53]:

```
# 5.12.  NB: From the boxplot we notice that there are some outliers.

# let's get the values of those outliers.

# Specify the column for which you want to identify outliers

Victim_Age = 'Victim Age'

# Calculate the Z-scores for each value in the column
z_scores = stats.zscore(Traffic_collision_df[Victim_Age])

# Set a threshold for Z-scores (e.g., 3 standard deviations)
threshold = 3

# Identify the indices of outliers based on the threshold
outlier_indices = (z_scores.abs() > threshold)

# Get the values of outliers
outlier_values = Traffic_collision_df[Victim_Age][outlier_indices]


# Count occurrences of each unique outlier value
outlier_counts = outlier_values.value_counts()

# Display unique outlier values and their counts
print("Unique Outlier Values and Counts:")
print(outlier_counts)
```

```
Unique Outlier Values and Counts:
Victim Age
99    6682
Name: count, dtype: int64
```

In [54]:

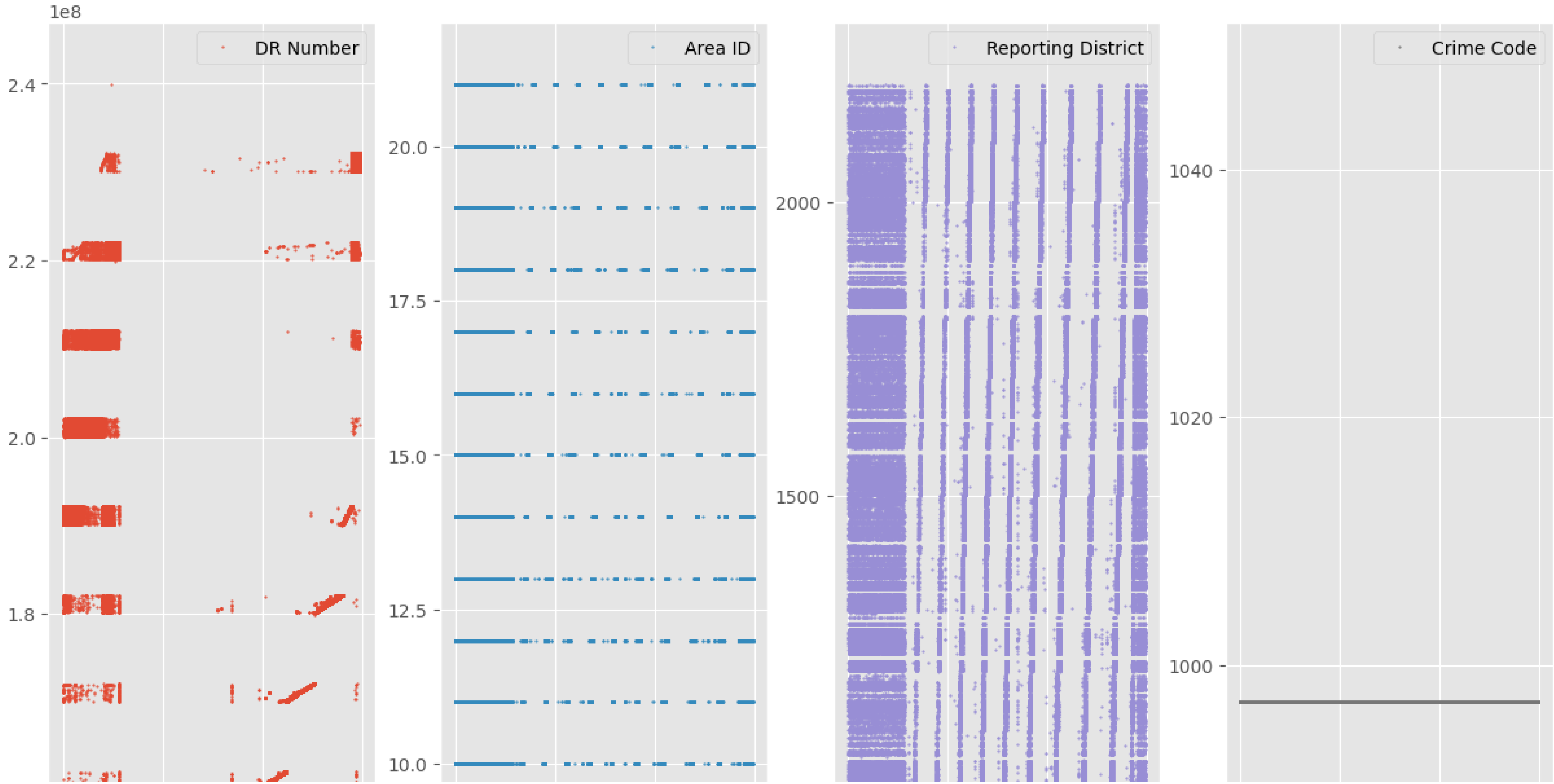
```
# From the dataset, one will notice that victims that were 99 years old had a very high occurrence rate 6,682 to be precise , this looks unusual and is an outlier.
```

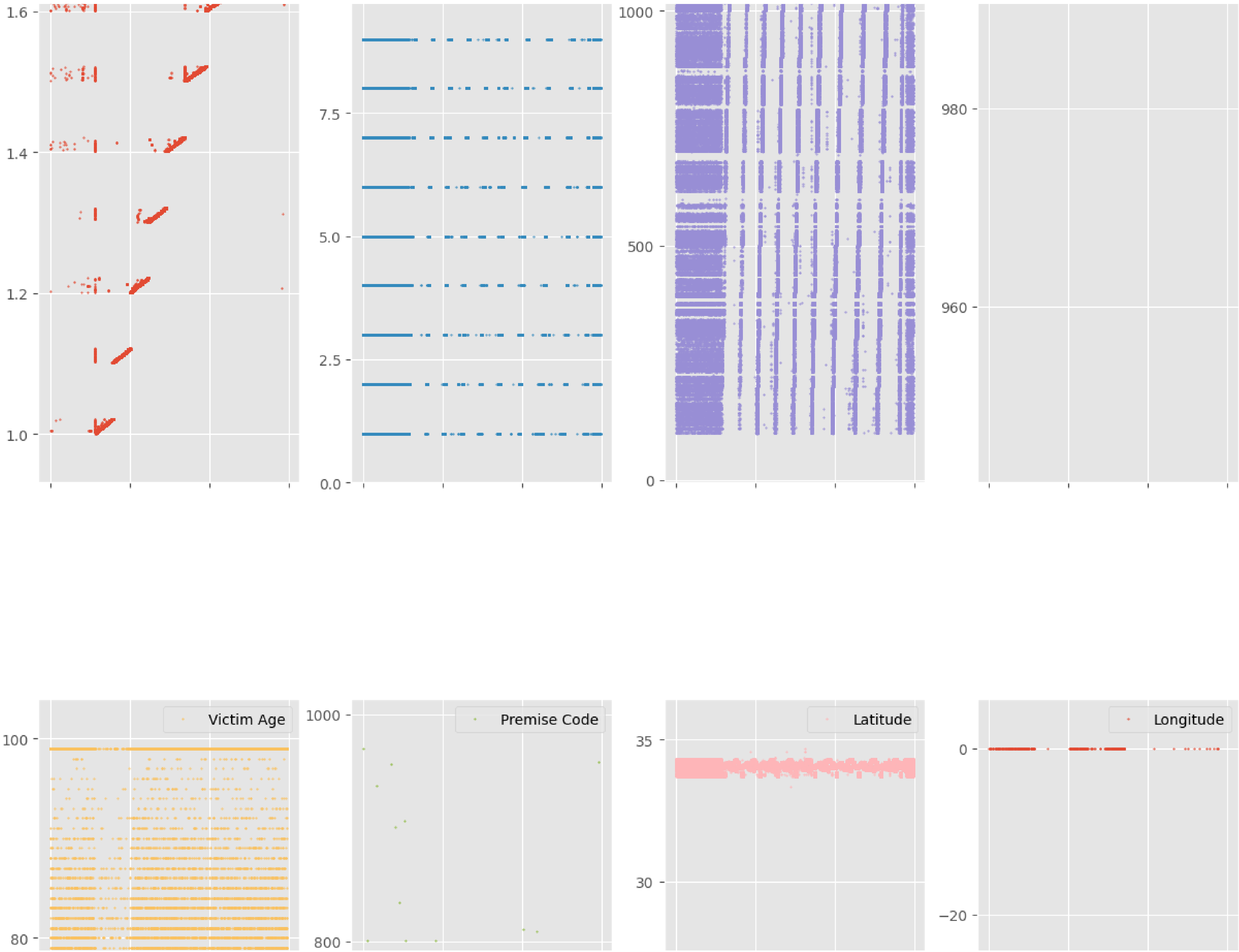
In [55]:

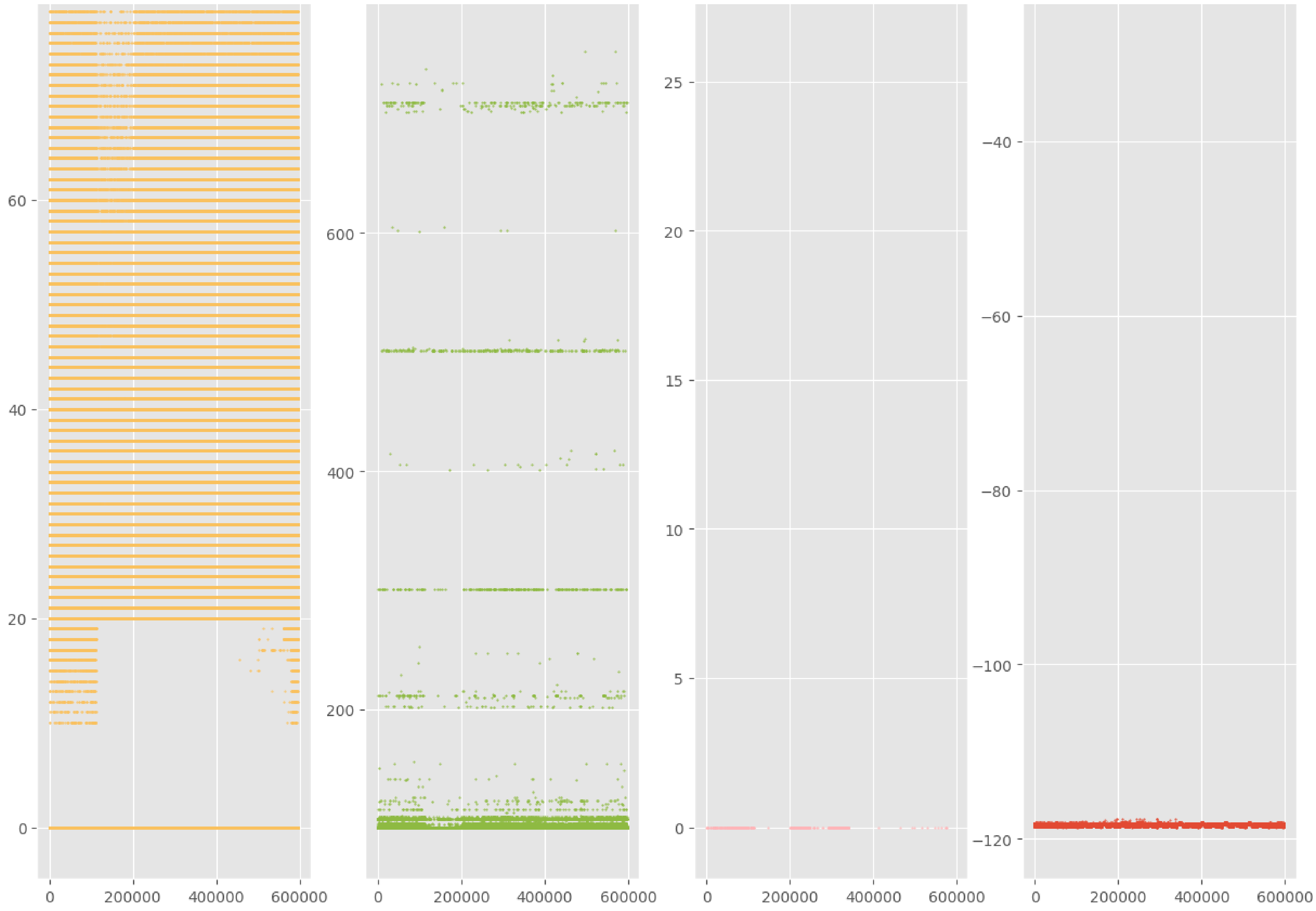
```
# 5.13.  Numerical features.

# There is need to plot a holistic view of the dataset for the numerical features.

Traffic_collision_df.plot(lw=0, marker=".", subplots=True, layout=(-1, 4),
          figsize=(15, 30), markersize=1)
```

Out[55]:

```
array([[<Axes: >, <Axes: >, <Axes: >, <Axes: >],
       [<Axes: >, <Axes: >, <Axes: >, <Axes: >]], dtype=object)
```




1.6
1.4
1.2
1.0
7.5
5.0
2.5
0.0
1000
500
0
980
960
Victim Age
100
80
Premise Code
1000
800
Latitude
35
30
Longitude
0
−20



In [56]:

```
# 5.14. Non-numerical features.
```

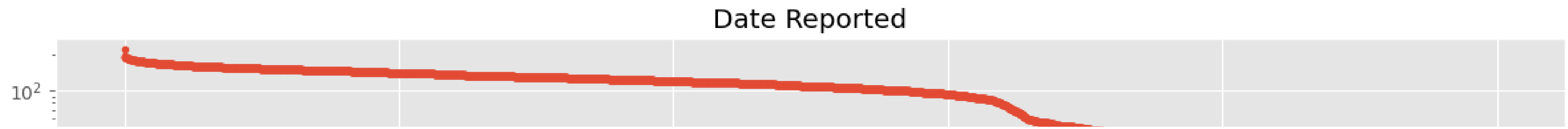
```
# Given that at this point, we only want to investigate the general quality of the dataset. So what we can do is take a general look at how many unique values each of these non-numerical features contain, and how often their most frequent category is represented.

# Extract descriptive properties of non-numerical features.

Traffic_collision_df.describe(exclude=["number", "datetime",])
```

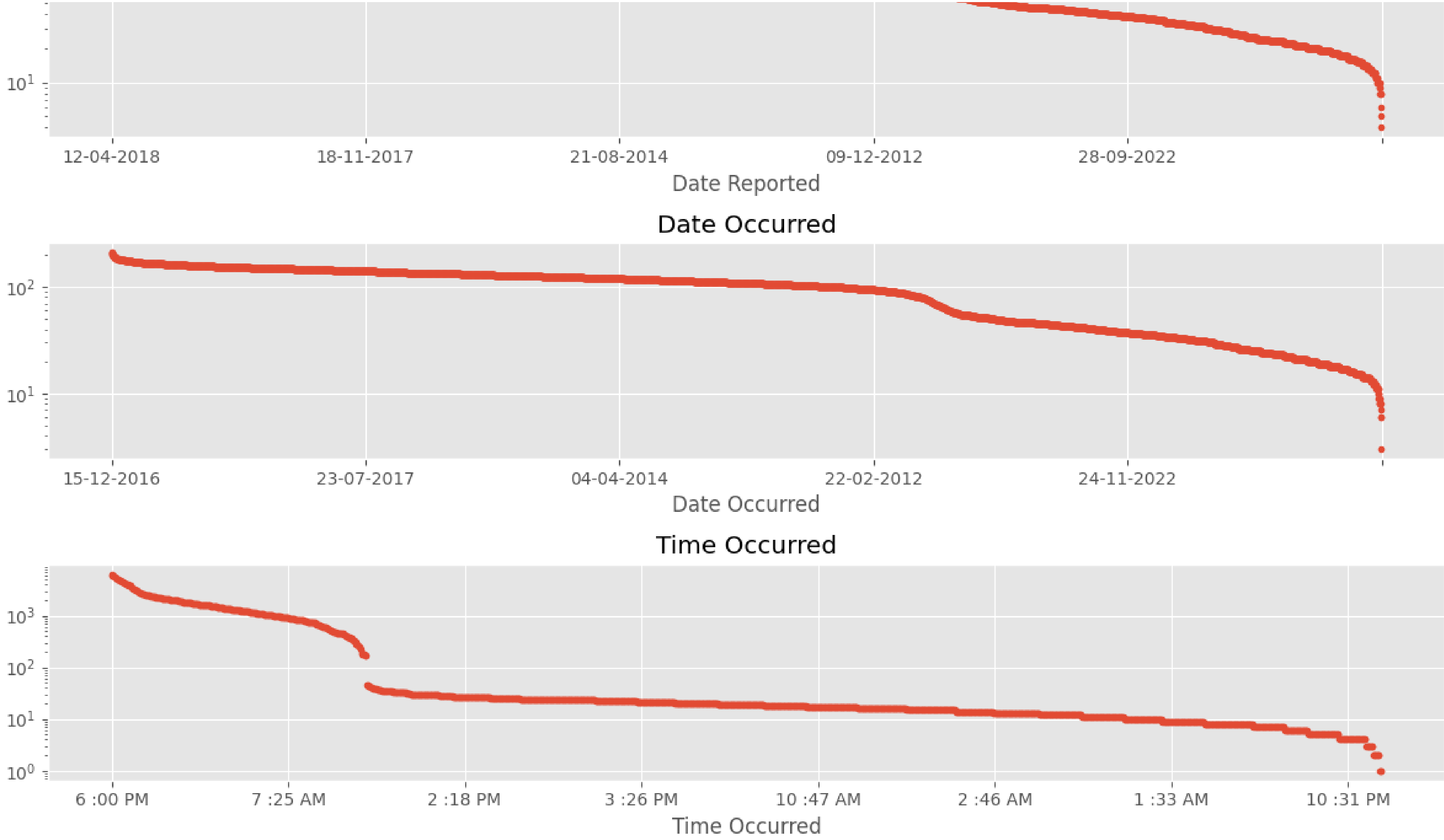
Out[56]:

	Date Reported	Date Occurred	Time Occurred	Period Of the Day	Area Name	Crime Code Description	MO Codes	Victim Sex	Victim Descent	Premise Description	Address	Cross Street
count	475108	475108	475108	475108	475108	475108	475108	475108	475108	475108	475108	475108
unique	5000	5000	1439	3	21	1	103822	6	20	93	13246	19301
top	12-04-2018	15-12-2016	6 :00 PM	morning	77th Street	TRAFFIC COLLISION	0605	M	H	STREET	WESTERN AV	VERMONT AV
freq	217	211	6440	168209	33063	475108	12912	284098	184438	454022	6817	3711

In [57]:

```
# 5.15. we could loop through all non-numerical features and plot for each of them the number of occurrences per unique value, to get more details from the them.


# Create figure object with 3 subplots of the first 3 features.
fig, axes = plt.subplots(ncols=1, nrows=3, figsize=(12, 8))

# Identify non-numerical features
df_non_numerical = Traffic_collision_df.select_dtypes(exclude=["number", "datetime"])

# Loop through features and put each subplot on a matplotlib axis object
for col, ax in zip(df_non_numerical.columns, axes.ravel()):

    # Selects one single feature and counts number of occurrences per unique value
    df_non_numerical[col].value_counts().plot(

        # Plots this information in a figure with log-scaled y-axis
        logy=True, title=col, lw=0, marker=".", ax=ax)

plt.tight_layout()
```





In [58]:

```
# 5.16. Conclusion of quality investigation.

# Now, we have a better understanding of the general quality of our dataset. We looked at missing values, duplicates, outliers, nu
merical and non-numerical and unique values .
```

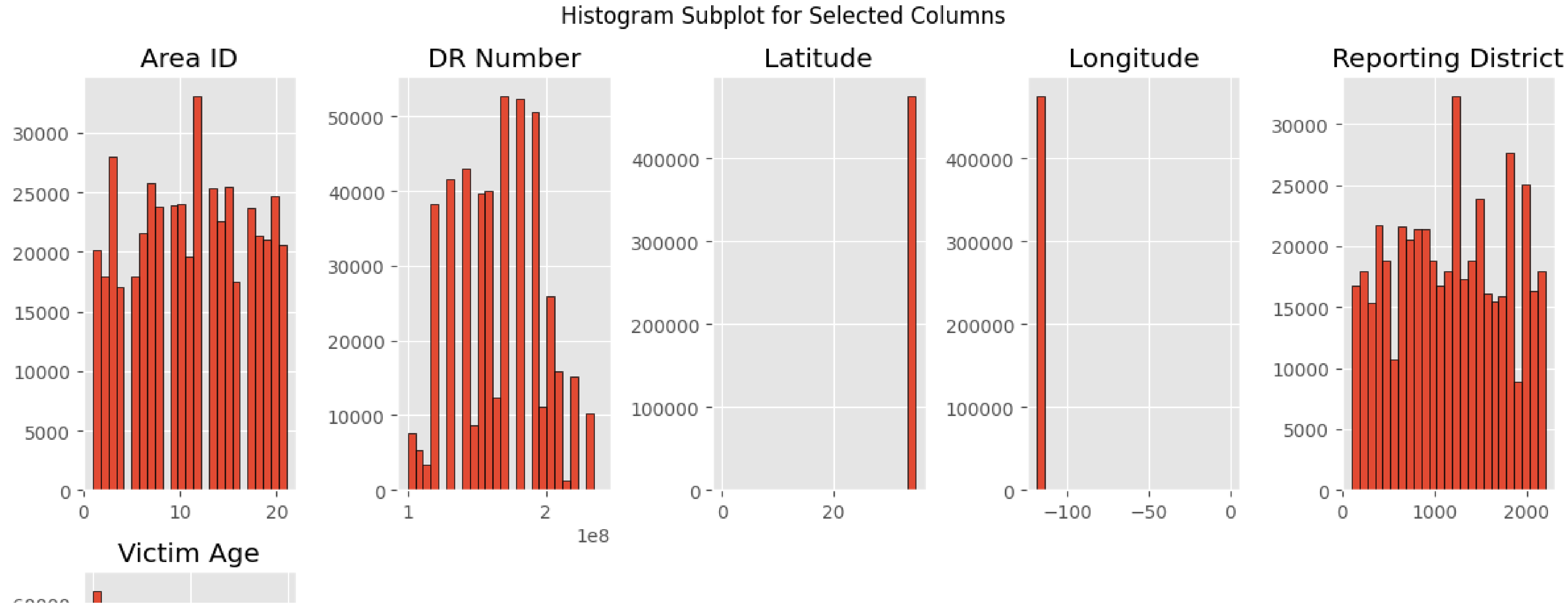
In [59]:

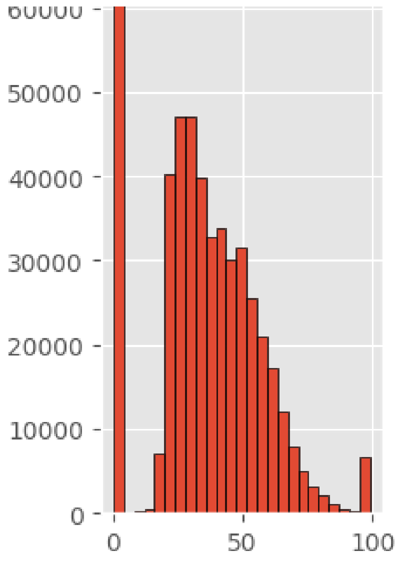
```
# Section C.
```

```
# 6. Content Investigation.(univariate, bivariate and correlation).

# In this section, we would try to answer some question and visualize our answers.

# Let’s go a step further and take a look at the actual content of the dataset, this wll be done feature by feature.
```

In [60]:

```
# 6.1. Feature Distribution.(univariate analysis)

# Plots the histogram for each numerical feature in a separate subplot.

# Specify the columns to exclude from the histogram subplot
exclude_columns = ['Time Occurred', 'Crime Code' ,'Premise Code']

# Select columns for the histogram subplot (excluding the specified columns)
columns_to_plot =Traffic_collision_df.columns.difference(exclude_columns)

# Plot histogram subplot for selected columns
Traffic_collision_df[columns_to_plot].hist(bins=25, figsize=(12 , 8), layout=(-1, 5), edgecolor="black")
plt.suptitle('Histogram Subplot for Selected Columns')
plt.tight_layout()
```





In [61]:

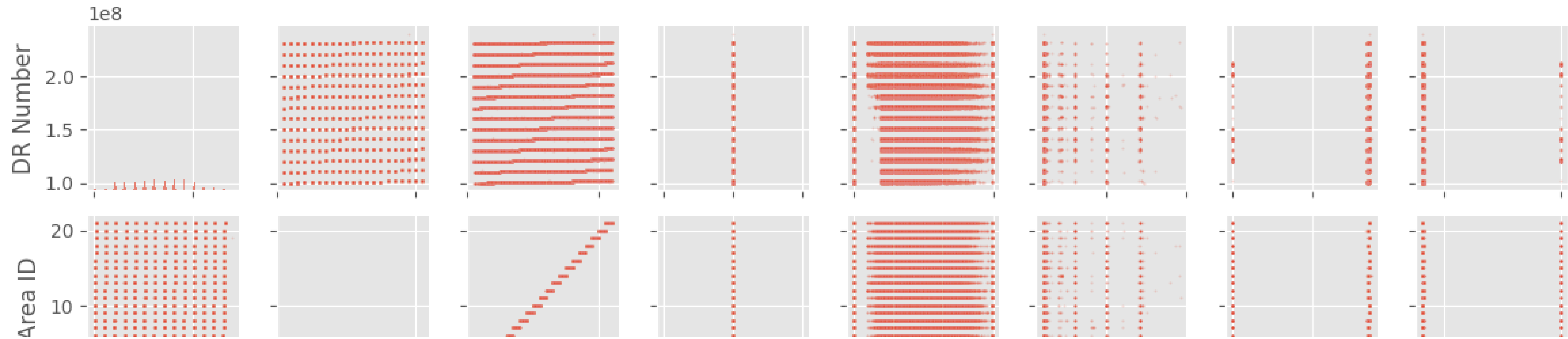
```
# There are a lot of very interesting things visible in these plots. We will use these distribution to answer some questions.
```

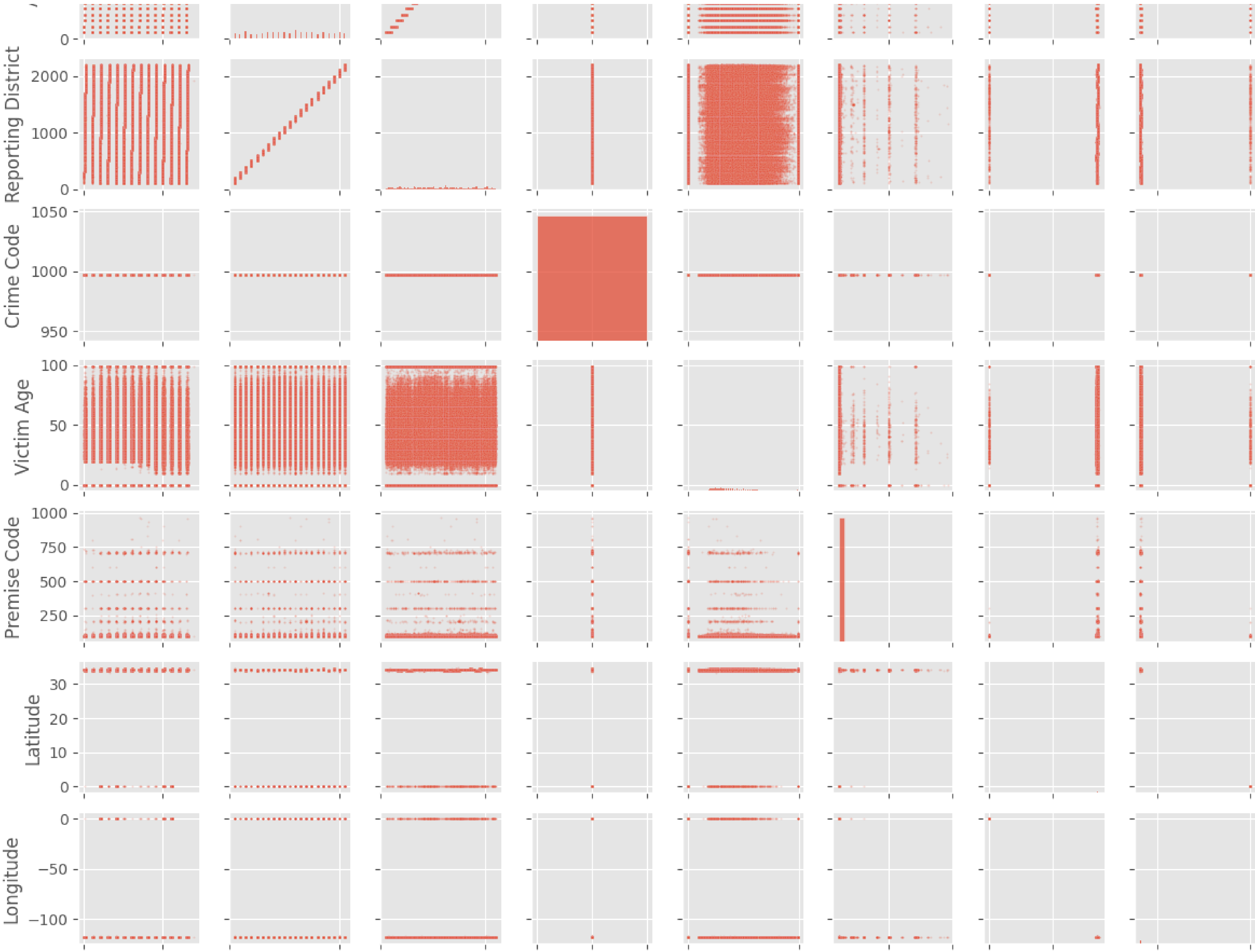
In [62]:

```
# 6.2. Feature Distribution.(Bivariate Analysis).

# Let's check the relationship between the continuous features.
# To do this ,we have to plot a pairplot to see if there is any relationship between any two features.

sns.pairplot(Traffic_collision_df, height=1.5, plot_kws={"s": 2, "alpha": 0.2})
```

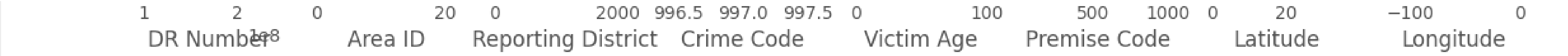
Out[62]:

```
<seaborn.axisgrid.PairGrid at 0x1eb402a5e70>
```





Reporting District
Crime Code
Victim Age
Premise Code
Latitude
Longitude



In [63]:

```
# 6.3. There seem to be a correlation between Area ID and Reporting District.
```

In [64]:

```
# 6.4. Let us answer some questions.

#  Dates that recorded the highest occurrence of collision
#  Dates that recorded the lowest occurrence of collision
#  Time which had the highest amount of collision
#  Time which had the highest amount of collision
#  The period of the day with the highest , and the lowest amount of collision.
#  Victim Age the highest and least amount of collision.
#  Victim Sex the highest and least amount of collision.
#  The top 10 locations with the highest and least amount of collision.
```

In [65]:

```
# 6.5. Date that recorded the highest occurrence of collision


# Use value_counts to get the count of each unique value in the specified column
highest_value_counts = Traffic_collision_df['Date Occurred'].value_counts().head()

# Find the value with the highest occurrence
max_occurrence_value = highest_value_counts.idxmax()

# Get the count of the highest occurrence value
max_occurrence_count = highest_value_counts.max()

print(f"The value with the highest occurrence in {'Date Occurred'} is: {max_occurrence_value}")
print(f"The count of the highest occurrence value is: {max_occurrence_count}")
```

```
The value with the highest occurrence in Date Occurred is: 15-12-2016
The count of the highest occurrence value is: 211
```

In [66]:

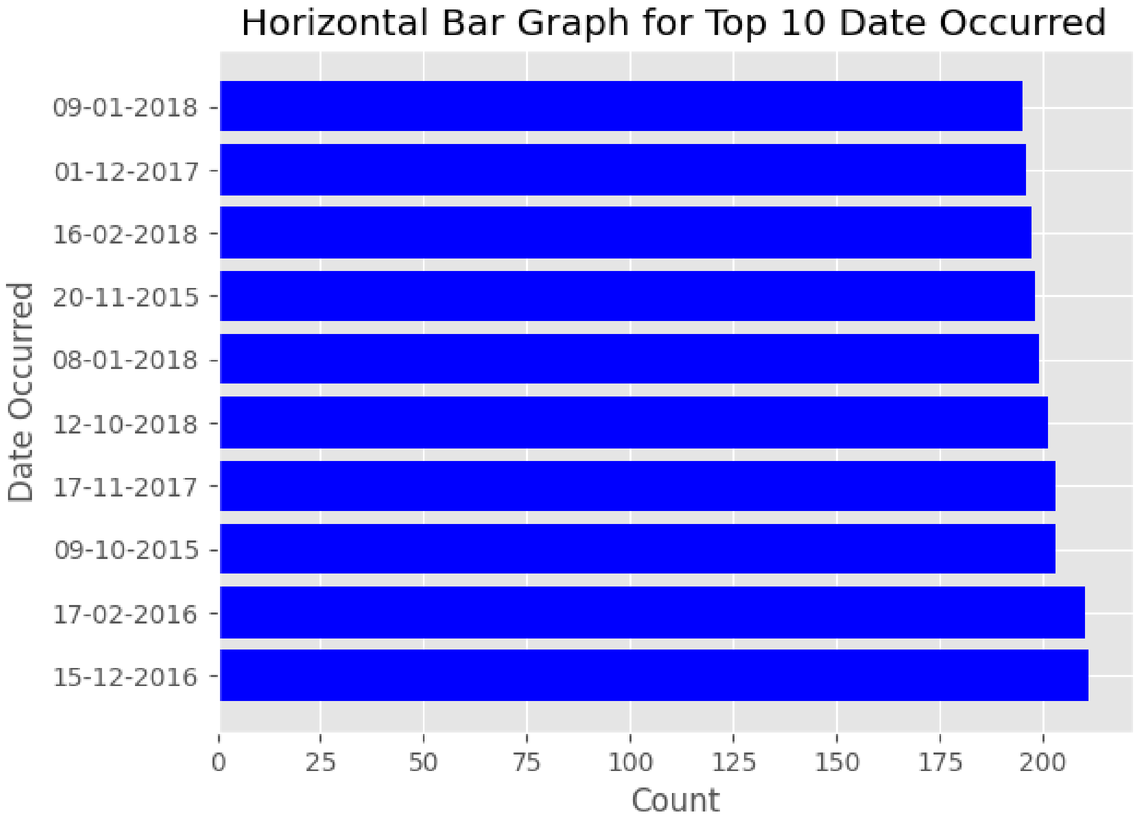
```
# 6.6. check for the top 10 dates with the highest collsion occurrence.
Top_10_Date_occured = Traffic_collision_df['Date Occurred'].value_counts().head(10)
print (Top_10_Date_occured)
```

```
Date Occurred
15-12-2016    211
17-02-2016    210
```

```
09-10-2015    203
17-11-2017    203
12-10-2018    201
08-01-2018    199
20-11-2015    198
16-02-2018    197
01-12-2017    196
09-01-2018    195
Name: count, dtype: int64
```

In [67]:

```
# 6.7.let's visualize the top_10_Date_occured.
# Count the occurrences of each unique value in the Date Occurred column
Top_10_Date_occured = Traffic_collision_df['Date Occurred'].value_counts().head(10)

# Create a horizontal bar graph
plt.barh(Top_10_Date_occured.index, Top_10_Date_occured.values, color='blue')
plt.xlabel('Count')
plt.ylabel('Date Occurred')
plt.title('Horizontal Bar Graph for Top 10 Date Occurred')
plt.show()
```



In [68]:

```
# 6.8. Dates that recorded the lowest occurrence of collision.


# Use value_counts to get the count of each unique value in the specified column
lowest_value_counts = Traffic_collision_df['Date Occurred'].value_counts()

# Find the value with the lowest occurrence
min_occurrence_value = lowest_value_counts.idxmin()

# Get the count of the lowest occurrence value
min_occurrence_count = lowest_value_counts.min()

print(f"The value with the lowest occurrence in {'Date Occurred'} is: {min_occurrence_value}")
print(f"The count of the lowest occurrence value is: {min_occurrence_count}")
```

```
The value with the lowest occurrence in Date Occurred is: 09-09-2023
The count of the lowest occurrence value is: 3
```

In [69]:

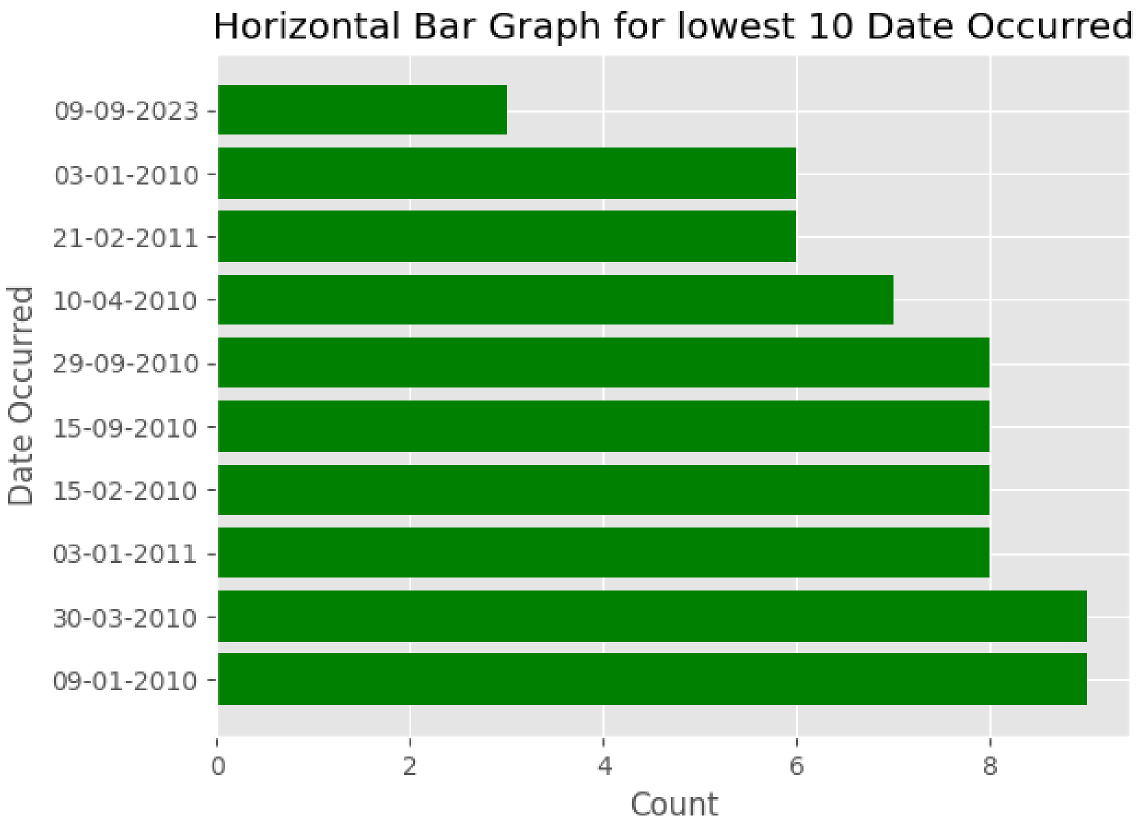
```
# 6.9. check for the top 10 dates with the highest collsion occurrence.
lowest_10_value_counts = Traffic_collision_df['Date Occurred'].value_counts().tail(10)
print (lowest_10_value_counts)
```

```
Date Occurred
09-01-2010    9
30-03-2010    9
03-01-2011    8
15-02-2010    8
15-09-2010    8
29-09-2010    8
10-04-2010    7
21-02-2011    6
03-01-2010    6
09-09-2023    3
Name: count, dtype: int64
```

In [70]:

```
# 6.10 let's visualize the lowest_10_value_counts.
# Count the occurrences of each unique value in the Date Occurred column
lowest_10_value_counts = Traffic_collision_df['Date Occurred'].value_counts().tail(10)

# Create a bar graph
plt.barh(lowest_10_value_counts.index, lowest_10_value_counts.values, color='green')
plt.xlabel('Count')
plt.ylabel('Date Occurred')
```

```
plt.title('Horizontal Bar Graph for lowest 10 Date Occurred')
plt.show()
```



In [71]:

```
# 6.11. Time which had the highest amount of collision.
        #Time which had the lowest amount of collision.

# Use value_counts to get the count of each unique time value
time_value_counts = Traffic_collision_df['Time Occurred'].value_counts()

# Find the time with the highest occurrence
max_occurrence_time = time_value_counts.idxmax()

# Get the count of the highest occurrence time
max_occurrence_count = time_value_counts.max()

# Find the time with the lowest occurrence
min_occurrence_time = time_value_counts.idxmin()

# Get the count of the lowest occurrence time
min_occurrence_count = time_value_counts.min()
```

```
print(f"The time with the highest occurrence in  is: {max_occurrence_time}")
print(f"The count of the highest occurrence time is: {max_occurrence_count}")
print(f"The time with the lowest occurrence in is: {min_occurrence_time}")
print(f"The count of the lowest occurrence time is: {min_occurrence_count}")
```

```
The time with the highest occurrence in  is: 6 :00 PM
The count of the highest occurrence time is: 6440
The time with the lowest occurrence in is: 3 :44 AM
The count of the lowest occurrence time is: 1
```

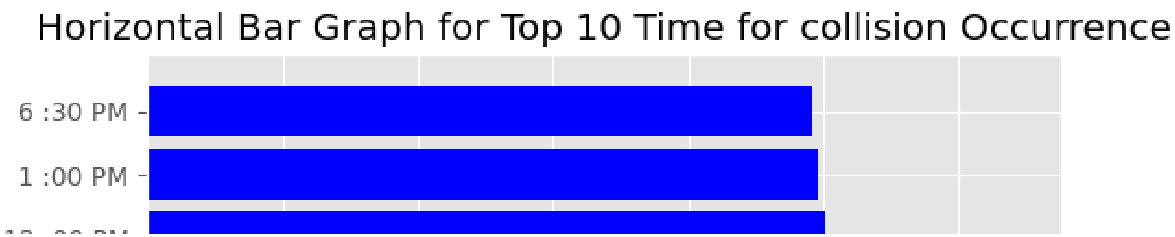
In [72]:

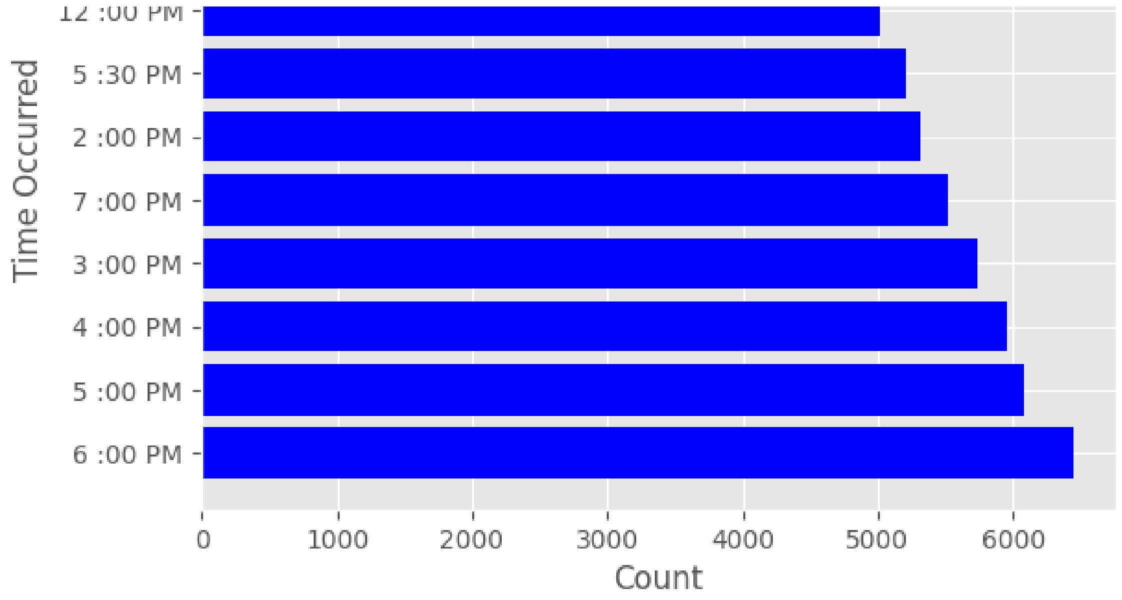
```
# 6.12. check for the top 10 time with the highest collsion occurrence.
Top_10_Time_occured = Traffic_collision_df['Time Occurred'].value_counts().head(10)
print (Top_10_Time_occured)
```

```
Time Occurred
6 :00 PM     6440
5 :00 PM     6075
4 :00 PM     5950
3 :00 PM     5731
7 :00 PM     5519
2 :00 PM     5320
5 :30 PM     5199
12 :00 PM    5016
1 :00 PM     4954
6 :30 PM     4925
Name: count, dtype: int64
```

In [73]:

```
# 6.13. let's visualize the top_10_Time_occured.
# Count the occurrences of each unique value in the Time Occurred column
Top_10_Time_occured = Traffic_collision_df['Time Occurred'].value_counts().head(10)

# Create a horizontal bar graph
plt.barh(Top_10_Time_occured.index, Top_10_Time_occured.values, color='blue')
plt.xlabel('Count')
plt.ylabel('Time Occurred')
plt.title('Horizontal Bar Graph for Top 10 Time for collision Occurrence')
plt.show()
```





In [74]:

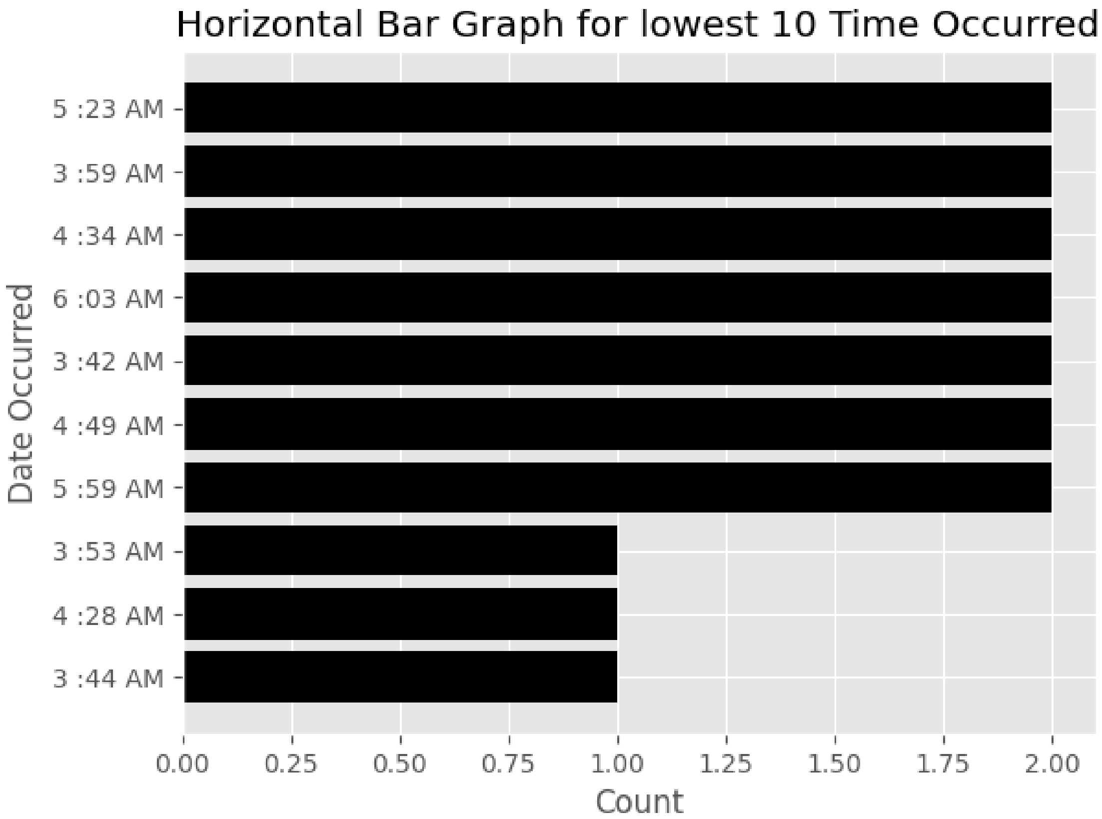
```
# 6.14.  get the lowest 10 times of collision occurrence.
#Count the occurrences of each unique value in the Time Occurred column.

lowest_10_Time_occured = Traffic_collision_df['Time Occurred'].value_counts().tail(10)
print(lowest_10_Time_occured)
```

```
Time Occurred
5 :59 AM    2
4 :49 AM    2
3 :42 AM    2
6 :03 AM    2
4 :34 AM    2
3 :59 AM    2
5 :23 AM    2
3 :44 AM    1
4 :28 AM    1
3 :53 AM    1
Name: count, dtype: int64
```

In [75]:

```
# 6.15. let's visualize the lowest_10_value_counts.
# Count the occurrences of each unique value in the Date Occurred column
lowest_10_value_counts = Traffic_collision_df['Time Occurred'].value_counts().tail(10).sort_values(ascending=True)

# Create a bar graph
plt.barh(lowest_10_value_counts.index, lowest_10_value_counts.values, color='black')
```

```
plt.xlabel('Count')
plt.ylabel('Date Occurred')
plt.title('Horizontal Bar Graph for lowest 10 Time Occurred')
plt.show()
```



In [86]:

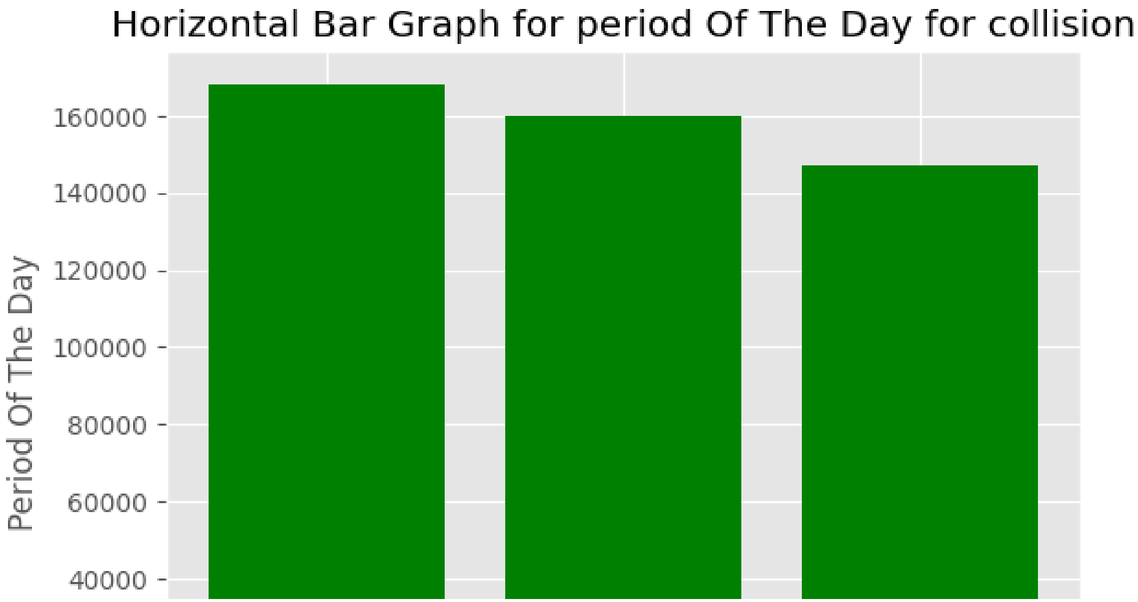
```
#6.16. The period of the day with the highest , and the lowest amount of collision.

# Use value_counts to get the count of each unique time value
period_value_counts = Traffic_collision_df['Period Of the Day'].value_counts()

# Find the period of the day with the highest occurrence
max_occurrence_period = time_value_counts.idxmax()

# Get the count of the highest occurrence time
max_occurrence_period_count = period_value_counts.max()

# Find the period with the lowest occurrence
min_occurrence_period = period_value_counts.idxmin()

# Get the count of the lowest occurrence time
```

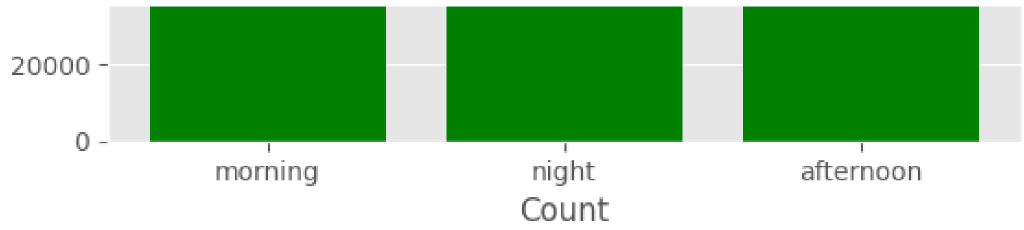
```
min_occurrence_period_count = period_value_counts.min()
print(f"The total of the different periods are:{period_value_counts}")

print(f"The period with the highest occurrence in  is: {max_occurrence_period}")
print(f"The count of the highest occurrence period is: {max_occurrence_period_count}")
print(f"The period with the lowest occurrence in is: {min_occurrence_period}")
print(f"The count of the lowest occurrence period is: {min_occurrence_period_count}")
```

```
The total of the different periods are:Period Of the Day
morning      168209
night        159838
afternoon    147061
Name: count, dtype: int64
The period with the highest occurrence in  is: 6 :00 PM
The count of the highest occurrence period is: 168209
The period with the lowest occurrence in is: afternoon
The count of the lowest occurrence period is: 147061
```

In [85]:

```
# 6.17. let's visualize the total distribution of the column Period of the day
# Count the occurrences of each unique value in the Time Occurred column
period_value_counts = Traffic_collision_df['Period Of the Day'].value_counts()

# Create a horizontal bar graph
plt.bar(period_value_counts.index, period_value_counts.values, color='green')
plt.xlabel('Count')
plt.ylabel('Period Of The Day')
plt.title('Horizontal Bar Graph for period Of The Day for collision')
plt.show()
```





In [87]:

```
# 6.18. Victim's Age with the highest and least amount of collision.

# Use value_counts to get the count of each unique time value
Victim_Age_value_counts = Traffic_collision_df['Victim Age'].value_counts()

# Find the time with the highest occurrence
max_occurrence_time = Victim_Age_value_counts.idxmax()

# Get the count of the highest occurrence time
max_occurrence_count = Victim_Age_value_counts.max()

# Find the time with the lowest occurrence
min_occurrence_time = Victim_Age_value_counts.idxmin()

# Get the count of the lowest occurrence time
min_occurrence_count = Victim_Age_value_counts.min()

print(f"The Victim Age with the highest occurrence is: {max_occurrence_time}")
print(f"The count of the highest occurring Victim Age is: {max_occurrence_count}")
print(f"The Victim Age with the lowest occurrence is: {min_occurrence_time}")
print(f"The count of the lowest occurrence Victim Age is: {min_occurrence_count}")
```

```
The Victim Age with the highest occurrence is: 0
The count of the highest occurring Victim Age is: 61923
The Victim Age with the lowest occurrence is: 97
The count of the lowest occurrence Victim Age is: 27
```

In [88]:

```
# 6.19. check for the top 10 Victim Age with the highest collsion occurrence.
Top_10_Victim_Age = Traffic_collision_df['Victim Age'].value_counts().head(10)
print (Top_10_Victim_Age)
```

```
Victim Age
0     61923
30    14379
25    12871
27    11709
40    11564
28    11461
```

```
24    11295
35    11251
26    11245
29    10923
Name: count, dtype: int64
```

In [89]:

```python
# 6.20. check for the 10 Victim Age with the lowest collsion occurrence.
lowest_10_Victim_Age = Traffic_collision_df['Victim Age'].value_counts().tail(10)
print (lowest_10_Victim_Age)
```

```
Victim Age
12    94
11    87
91    81
92    80
93    46
94    46
96    40
95    38
98    30
97    27
Name: count, dtype: int64
```

In [90]:

```python
# 6.21. Victim Sex with the highest and least amount of collision.

 #Use value_counts to get the count of each unique Victim sex value
Victim_Sex_value_counts = Traffic_collision_df['Victim Sex'].value_counts()

# Find the time with the highest occurrence
max_occurrence_time = Victim_Sex_value_counts.idxmax()

# Get the count of the highest occurrence time
max_occurrence_count = Victim_Sex_value_counts.max()

# Find the time with the lowest occurrence
min_occurrence_time = Victim_Sex_value_counts.idxmin()

# Get the count of the lowest occurrence time
min_occurrence_count = Victim_Sex_value_counts.min()

print(f"The Victim Sex with the highest occurrence is: {max_occurrence_time}")
print(f"The count of the highest occurring Victim Sex is: {max_occurrence_count}")
print(f"The Victim Sex with the lowest occurrence is: {min_occurrence_time}")
print(f"The count of the lowest occurrence Victim Sex is: {min_occurrence_count}")
```

```
The Victim Sex with the highest occurrence is: M
```

```
The count of the highest occurring Victim Sex is: 284098
The Victim Sex with the lowest occurrence is: -
The count of the lowest occurrence Victim Sex is: 2
```

In [91]:

```
# 6.22. check for the total number of females involved in traffic collsion .
Top_5_Victim_Sex = Traffic_collision_df['Victim Sex'].value_counts().head(10)
print (Top_5_Victim_Sex)
```

```
Victim Sex
M    284098
F    176758
X     14111
H       128
N        11
-         2
Name: count, dtype: int64
```

In [92]:

```
# 6.23. the top 10 locations with the highest amount of collision.

# Find the highest occurrences of traffic collisions for each latitude and longitude pair

top_10_locations = Traffic_collision_df.groupby(['Latitude', 'Longitude']).size().reset_index(name='Collision_Count') \
                              .sort_values(by='Collision_Count', ascending=False).head(10)
print(top_10_locations)
```

```
       Latitude  Longitude  Collision_Count
0        0.0000     0.0000              637
7656    33.9892  -118.3089              507
4982    33.9601  -118.2827              461
40217   34.2355  -118.5536              435
36431   34.2012  -118.4662              423
38862   34.2216  -118.4488              407
32988   34.1721  -118.4662              372
36454   34.2012  -118.4313              364
34546   34.1867  -118.4662              358
4957    33.9600  -118.3090              329
```

In [93]:

```
# Find the lowest occurrences of traffic collisions for each latitude and longitude pair
lowest_10_locations = Traffic_collision_df.groupby(['Latitude', 'Longitude']).size().reset_index(name='Collision_Count') \
                              .sort_values(by='Collision_Count', ascending=True).head(10)
print(lowest_10_locations)
```

```
       Latitude  Longitude  Collision_Count
45361   34.6903  -118.3053                1
10657   34.0145  -118.4235                1
```

```
10658   34.0145  -118.4197                 1
10659   34.0145  -118.3408                 1
29683   34.1340  -118.1899                 1
29681   34.1340  -118.3441                 1
10664   34.0145  -118.2565                 1
10666   34.0146  -118.4278                 1
29674   34.1339  -118.3454                 1
10668   34.0146  -118.3361                 1
```

In [94]:

```
# 6.24. Compute feature correlation

# Identify non-numeric columns
non_numeric_columns = Traffic_collision_df.select_dtypes(exclude=['number']).columns

# Apply label encoding to non-numeric columns
label_encoder = LabelEncoder()
Traffic_collision_df[non_numeric_columns] = Traffic_collision_df[non_numeric_columns].apply(lambda col: label_encoder.fit_transfor
m(col.astype(str)))

# Calculate the correlation matrix
correlation_matrix = Traffic_collision_df.corr()

# Display the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
                        DR Number  Date Reported  Date Occurred  \
DR Number                1.000000      -0.006794      -0.007230
Date Reported           -0.006794       1.000000       0.895227
Date Occurred           -0.007230       0.895227       1.000000
Time Occurred           -0.003161       0.001654       0.002520
Period Of the Day        0.018706       0.000592       0.001117
Area ID                  0.020804      -0.001653      -0.000401
Area Name               -0.044457       0.000786       0.001428
Reporting District       0.020338      -0.001548      -0.000288
Crime Code                    NaN            NaN            NaN
Crime Code Description        NaN            NaN            NaN
MO Codes                 0.119236      -0.000213      -0.000916
Victim Age               0.059560      -0.001186      -0.000281
Victim Sex               0.060080       0.001779       0.002657
Victim Descent          -0.037839       0.000765       0.000852
Premise Code            -0.001112      -0.000296      -0.000938
Premise Description     -0.008612       0.001120       0.001036
Address                 -0.013754       0.000353       0.000720
Cross Street            -0.018539       0.001485       0.001719
Latitude                -0.004088      -0.010543      -0.010471
Longitude                0.004034       0.010637       0.010649
```

	Time Occurred	Period Of the Day	Area ID	Area Name
DR Number	-0.003161	0.018706	0.020804	-0.044457
Date Reported	0.001654	0.000592	-0.001653	0.000786
Date Occurred	0.002520	0.001117	-0.000401	0.001428
Time Occurred	1.000000	0.245942	0.003558	0.004785
Period Of the Day	0.245942	1.000000	0.009267	-0.010135
Area ID	0.003558	0.009267	1.000000	-0.053113
Area Name	0.004785	-0.010135	-0.053113	1.000000
Reporting District	0.003577	0.009141	0.998931	-0.056982
Crime Code	NaN	NaN	NaN	NaN
Crime Code Description	NaN	NaN	NaN	NaN
MO Codes	-0.004309	0.008720	-0.154683	-0.122470
Victim Age	0.010787	-0.046444	0.002687	0.022375
Victim Sex	-0.031444	0.035028	-0.020866	-0.024424
Victim Descent	-0.025243	-0.000217	0.016072	0.109158
Premise Code	-0.004817	-0.000821	0.003927	0.000114
Premise Description	0.017417	0.003747	0.002061	0.006511
Address	0.000738	-0.006907	0.100008	0.110642
Cross Street	-0.002600	-0.003238	0.067426	0.081997
Latitude	0.000518	0.006058	0.023825	0.005175
Longitude	-0.000622	-0.006335	-0.001554	-0.006300

	Reporting District	Crime Code
DR Number	0.020338	NaN
Date Reported	-0.001548	NaN
Date Occurred	-0.000288	NaN
Time Occurred	0.003577	NaN
Period Of the Day	0.009141	NaN
Area ID	0.998931	NaN
Area Name	-0.056982	NaN
Reporting District	1.000000	NaN
Crime Code	NaN	NaN
Crime Code Description	NaN	NaN
MO Codes	-0.155037	NaN
Victim Age	0.003021	NaN
Victim Sex	-0.021225	NaN
Victim Descent	0.015269	NaN
Premise Code	0.003989	NaN
Premise Description	0.001660	NaN
Address	0.101208	NaN
Cross Street	0.067924	NaN
Latitude	0.023664	NaN
Longitude	-0.001627	NaN

	Crime Code Description	MO Codes	Victim Age
DR Number	NaN	0.119236	0.059560
Date Reported	NaN	-0.000213	-0.001186
Date Occurred	NaN	-0.000916	-0.000281
Time Occurred	NaN	-0.004309	0.010787
Period Of the Day	NaN	0.008720	-0.046444

```
Area ID                               NaN -0.154683    0.002687
Area Name                             NaN -0.122470    0.022375
Reporting District                    NaN -0.155037    0.003021
Crime Code                            NaN       NaN         NaN
Crime Code Description                NaN       NaN         NaN
MO Codes                              NaN  1.000000    0.001107
Victim Age                            NaN  0.001107    1.000000
Victim Sex                            NaN  0.026564   -0.115716
Victim Descent                        NaN -0.085585   -0.070710
Premise Code                          NaN  0.002974   -0.006697
Premise Description                   NaN -0.040141   -0.007492
Address                               NaN -0.100418    0.026051
Cross Street                          NaN -0.089542    0.011633
Latitude                              NaN -0.010551   -0.000978
Longitude                             NaN  0.002185    0.001645

                        Victim Sex  Victim Descent  Premise Code  \
DR Number                 0.060080       -0.037839     -0.001112
Date Reported             0.001779        0.000765     -0.000296
Date Occurred             0.002657        0.000852     -0.000938
Time Occurred            -0.031444       -0.025243     -0.004817
Period Of the Day         0.035028       -0.000217     -0.000821
Area ID                  -0.020866        0.016072      0.003927
Area Name                -0.024424        0.109158      0.000114
Reporting District       -0.021225        0.015269      0.003989
Crime Code                     NaN             NaN           NaN
Crime Code Description         NaN             NaN           NaN
MO Codes                  0.026564       -0.085585      0.002974
Victim Age               -0.115716       -0.070710     -0.006697
Victim Sex                1.000000        0.160068      0.005146
Victim Descent            0.160068        1.000000      0.012182
Premise Code              0.005146        0.012182      1.000000
Premise Description      -0.001952       -0.010745     -0.243713
Address                  -0.011842        0.068590      0.006074
Cross Street             -0.010167        0.083862      0.005836
Latitude                 -0.002521        0.009743      0.001067
Longitude                 0.001234       -0.000681     -0.000480

                        Premise Description   Address  Cross Street  Latitude  \
DR Number                         -0.008612 -0.013754     -0.018539 -0.004088
Date Reported                      0.001120  0.000353      0.001485 -0.010543
Date Occurred                      0.001036  0.000720      0.001719 -0.010471
Time Occurred                      0.017417  0.000738     -0.002600  0.000518
Period Of the Day                  0.003747 -0.006907     -0.003238  0.006058
Area ID                            0.002061  0.100008      0.067426  0.023825
Area Name                          0.006511  0.110642      0.081997  0.005175
Reporting District                 0.001660  0.101208      0.067924  0.023664
Crime Code                              NaN       NaN           NaN       NaN
Crime Code Description                  NaN       NaN           NaN       NaN
MO Codes                          -0.040141 -0.100418     -0.089542 -0.010551
Victim Age                        -0.007492  0.026051      0.011633 -0.000978
```

```
Victim Age                    -0.007492  0.026051      0.011633 -0.000978
Victim Sex                    -0.001952 -0.011842     -0.010167 -0.002521
Victim Descent                -0.010745  0.068590      0.083862  0.009743
Premise Code                  -0.243713  0.006074      0.005836  0.001067
Premise Description            1.000000 -0.006409     -0.007307  0.001260
Address                       -0.006409  1.000000      0.049346  0.018029
Cross Street                  -0.007307  0.049346      1.000000  0.018070
Latitude                       0.001260  0.018029      0.018070  1.000000
Longitude                     -0.000777 -0.003856     -0.004420 -0.997014

                        Longitude
DR Number                0.004034
Date Reported            0.010637
Date Occurred            0.010649
Time Occurred           -0.000622
Period Of the Day       -0.006335
Area ID                 -0.001554
Area Name               -0.006300
Reporting District      -0.001627
Crime Code                    NaN
Crime Code Description        NaN
MO Codes                 0.002185
Victim Age               0.001645
Victim Sex               0.001234
Victim Descent          -0.000681
Premise Code            -0.000480
Premise Description     -0.000777
Address                 -0.003856
Cross Street            -0.004420
Latitude                -0.997014
Longitude                1.000000
```

In [95]:

```
# There seem to be a relationship between Area ID and Reporting District.
```

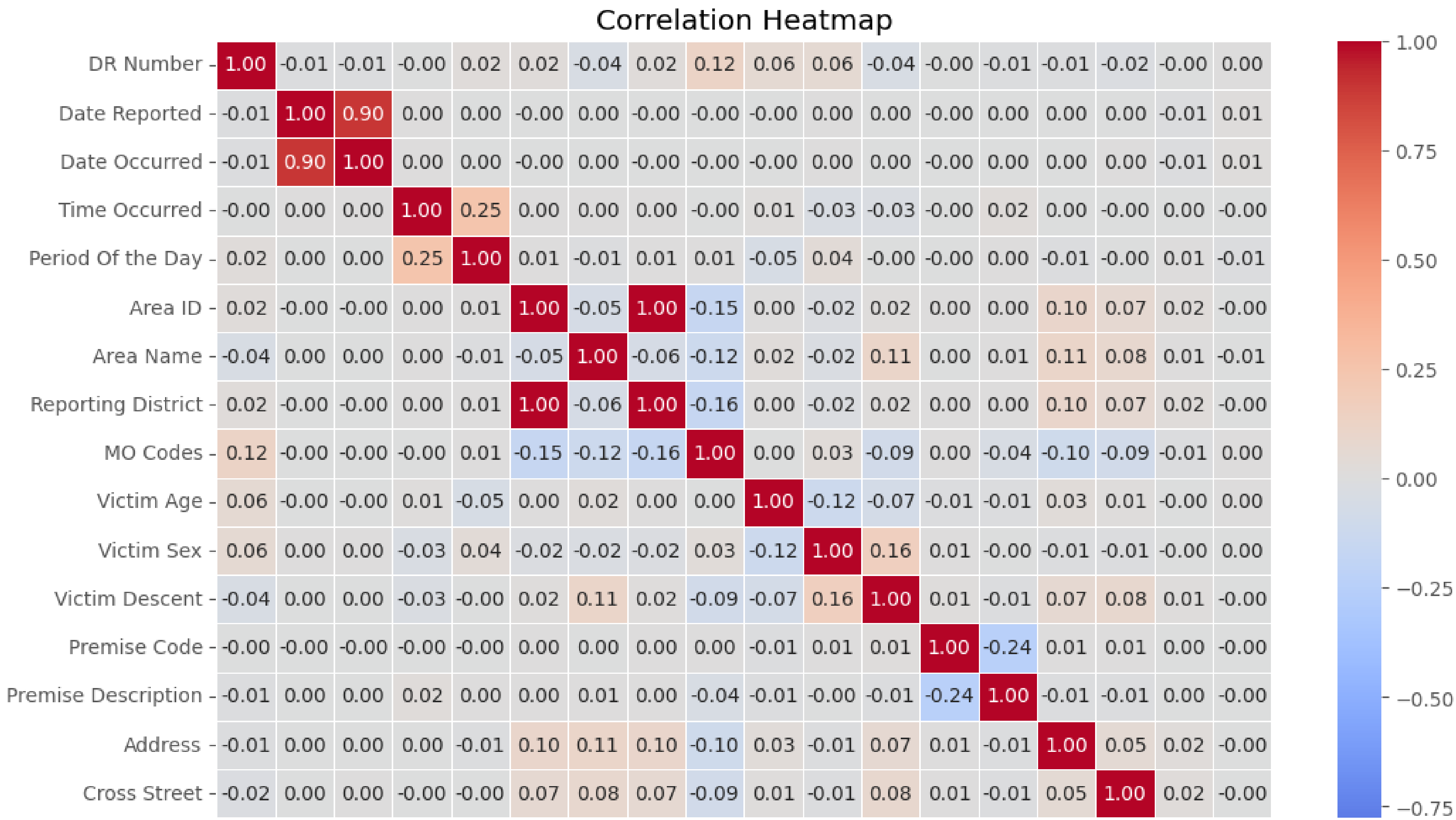
In [96]:

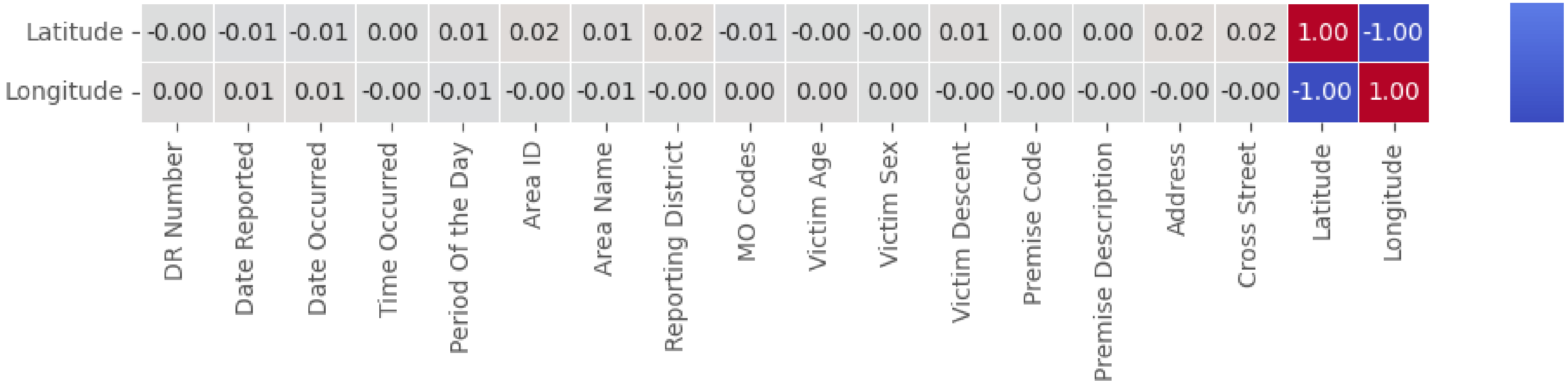
```
# For a better visualization, let's drop colums Crime Code and Crime Code Description since we may not need them for now.

Traffic_collision_df.drop(columns= ['Crime Code','Crime Code Description'] , inplace = True)
```

In [97]:

```
# 6.6. 2 let's create a Heatmap for of the remaining features. let's numerize the entire date.

# Identify non-numeric and datetime columns


non_numeric_columns = Traffic_collision_df.select_dtypes(exclude=['number', 'datetime']).columns
```

```python
datetime_columns = Traffic_collision_df.select_dtypes(include='datetime').columns

# Apply label encoding to non-numeric columns
label_encoder = LabelEncoder()
Traffic_collision_df[non_numeric_columns] = Traffic_collision_df[non_numeric_columns].apply(lambda col: label_encoder.fit_transfor
m(col.astype(str)))

# Calculate the correlation matrix
correlation_matrix = Traffic_collision_df.corr()

# Create a heatmap using seaborn
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```





In [98]:

```
# There is a strong relationship between Date Reported and Date Occurred.
```

In [ ]:

```

```