

Project 1 - Alex and Victor Hugo

September 14, 2017

1 EEC2006 - Data Science

2 Project #1 Finding Patterns in Crime

2.1 Alex Furtunato

2.2 Victor Hugo - 20171003230

3 1. Introduction

This notebook contains the analysis of a specific dataset, understanding each of his columns and answering specific questions made it by the project guide. Besides that, it has also others patterns analysis, as well of theories of the informations founded in the data. To accomplish that it was used python and, more specific, the pandas library.

This notebook is organized as follows. In section 2 we cite the database and analyze each column, telling if they had missing values, how many different values are found and the their range. In section 3 we take a look at the times of the crimes, this is the bigger analysis, once it's divided in days of week, periods of time and months of the year. In section 4 the location of the crimes is studied, but first of all, is discussed how we choose to define the location, and then, it was analyzed the patterns of it. Section 5 is about the types of crimes, basically we analyze the occurrence of violent and nonviolent crimes, according to our classification. Some combined analysis is made in section 6, answering questions about the time and type of crimes, as well of location and type of crimes. At last, in section 7, we pose our own questions and answer a few of them, leaving others to further work.

4 2. Data

In this project we are using a dataset provided by the Montgomery County Government that contains the crimes reported by in 2013, from july to december, and entered into a database. Therefore, each row is a specific report and the columns of the dataset represents the informations about that crime, which in our case, we have 22 different columns.

The code below reads the dataset and stores in a pandas DataFrame. It also prints the names of all the columns and the first five rows.

```
In [1]: import pandas as pd
import numpy as np
import operator
```

```

#Reads the dataset in a .csv file and stores in a pandas DataFrame
crimes = pd.read_csv("MontgomeryCountyCrime2013.csv")

print("List of all the columns of the dataset:")
#Use the columns property to show all columns
print(crimes.columns)

print("\nFirst five rows of the dataset:")
#Use the head() function to return the first 5 rows
crimes.head()

```

List of all the columns of the dataset:

```

Index(['Incident ID', 'CR Number', 'Dispatch Date / Time', 'Class',
      'Class Description', 'Police District Name', 'Block Address', 'City',
      'State', 'Zip Code', 'Agency', 'Place', 'Sector', 'Beat', 'PRA',
      'Start Date / Time', 'End Date / Time', 'Latitude', 'Longitude',
      'Police District Number', 'Location', 'Address Number'],
      dtype='object')

```

First five rows of the dataset:

```

Out[1]:

```

	Incident ID	CR Number	Dispatch Date / Time	Class	\
0	200939101	13047006	10/02/2013 07:52:41 PM	511	
1	200952042	13062965	12/31/2013 09:46:58 PM	1834	
2	200926636	13031483	07/06/2013 09:06:24 AM	1412	
3	200929538	13035288	07/28/2013 09:13:15 PM	2752	
4	200930689	13036876	08/06/2013 05:16:17 PM	2812	

	Class Description	Police District Name	\
0	BURG FORCE-RES/NIGHT	OTHER	
1	CDS-POSS MARIJUANA/HASHISH	GERMANTOWN	
2	VANDALISM-MOTOR VEHICLE	MONTGOMERY VILLAGE	
3	FUGITIVE FROM JUSTICE(OUT OF STATE)	BETHESDA	
4	DRIVING UNDER THE INFLUENCE	BETHESDA	

	Block Address	City	State	Zip Code	...	Sector	\
0	25700 MT RADNOR DR	DAMASCUS	MD	20872.0	...	NaN	
1	GUNNERS BRANCH RD	GERMANTOWN	MD	20874.0	...	M	
2	OLDE TOWNE AVE	GAITHERSBURG	MD	20877.0	...	P	
3	BEACH DR	CHEVY CHASE	MD	20815.0	...	D	
4	BEACH DR	SILVER SPRING	MD	20815.0	...	D	

	Beat	PRA	Start Date / Time	End Date / Time	Latitude	Longitude	\
0	NaN	NaN	10/02/2013 07:52:00 PM		NaN	NaN	NaN
1	5M1	470.0	12/31/2013 09:46:00 PM		NaN	NaN	NaN
2	6P3	431.0	07/06/2013 09:06:00 AM		NaN	NaN	NaN

3	2D1	11.0	07/28/2013	09:13:00 PM	NaN	NaN	NaN
4	2D3	178.0	08/06/2013	05:16:00 PM	NaN	NaN	NaN

	Police District	Number	Location	Address	Number
0		OTHER	NaN		25700.0
1		5D	NaN		NaN
2		6D	NaN		NaN
3		2D	NaN		NaN
4		2D	NaN		NaN

[5 rows x 22 columns]

4.1 2.1. Columns

4.1.1 2.1.1 Incident ID

Is the identification number of the incident. It's a unique integer number, their value vary from 200922213 to 201070057, with no missing value. Therefore, it's the natural key of this database. The code below prove those informations.

```
In [2]: #Stores a list of boolean values, where True means that in that position the 'Incident ID' is null
incident_id_null = crimes["Incident ID"].isnull()
#Stores a list of null 'Incident ID'.
incident_id_null_true = crimes[incident_id_null]["Incident ID"]
#The length of this list is the number of null elements contained in the original column
print("Number of rows which the column Incident ID is blank:", len(incident_id_null_true))

#Use the 'is_unique' propertie to tell if the column has only unique values
print("The Incident ID column has only unique values?", crimes["Incident ID"].is_unique)
#Use the 'dtypes' propertie to tell the type of the data in the column
print("Type of the values in the Incident ID column:", crimes["Incident ID"].dtypes)
#Use the min() function to tell the minimum value of the column
print("Minimum value of Incident ID:", crimes["Incident ID"].min())
#Use the max() function to tell the maximum value of the column
print("Maximum value of Incident ID:", crimes["Incident ID"].max())
```

Number of rows which the column Incident ID is blank: 0

The Incident ID column has only unique values? True

Type of the values in the Incident ID column: int64

Minimum value of Incident ID: 200922213

Maximum value of Incident ID: 201070057

4.1.2 2.1.2. CR Number

Is the police report number. This report is made by the police and matches the incident number with the police investigation. It's also an integer number and we don't have any blank row, e.g., each incident has his police report associated. But we can have repeated values, e.g., one report

can deal with more than one incident. This is actually the case, we have the same police report number for more than one incident, although this percentage is very low.

```
In [3]: cr_number_null_true = crimes[crimes["CR Number"].isnull()]["CR Number"]
        print("Number of rows which the column CR_Number is blank:", len(cr_number_null_true))

        print("The CR Number column has only unique values?", crimes["CR Number"].is_unique)

        #Use the unique() function to return a list of unique values.
        #Therefore, the length of this list is the number of different values in this column.
        #Then, the percentage is just the division of this number by the total length
        print("Percentage of unique values in the CR Number column:",
              len(crimes["CR Number"].unique())/len(crimes["CR Number"]) )

        print("Type of the values in the CR Number column:", crimes["CR Number"].dtypes)
        print("Minimum value of CR Number:", crimes["CR Number"].min())
        print("Maximun value of CR Number:", crimes["CR Number"].max())
```

```
Number of rows which the column CR_Number is blank: 0
The CR Number column has only unique values? False
Percentage of unique values in the CR Number column: 0.9995292909409902
Type of the values in the CR Number column: int64
Minimum value of CR Number: 13030551
Maximun value of CR Number: 201302992
```

4.1.3 2.1.3. Dispatch Date / Time

The actual date and time that a Officer was dispatched to the incident. The date has the following syntax: mm/dd/yyyy hh:mm:ss AM/PM. There is no missing values in this column, e.g., there had a Officer dispatched for each incident and this was also registred. Considering that we are reading a 2013 database, all the dispatches dates occured in this year.

```
In [4]: dispatch_date_null_true = crimes[crimes["Dispatch Date / Time"].isnull()]["Dispatch Date / Time"]
        print("Number of rows which the column 'Dispatch Date / Time' is blank:", len(dispatch_date_null_true))

        print("Minimum value of 'Dispatch Date / Time':", crimes["Dispatch Date / Time"].min())
        print("Maximun value of 'Dispatch Date / Time':", crimes["Dispatch Date / Time"].max())
```

```
Number of rows which the column 'Dispatch Date / Time' is blank: 0
Minimum value of 'Dispatch Date / Time': 07/01/2013 01:03:17 AM
Maximun value of 'Dispatch Date / Time': 12/31/2013 12:58:39 PM
```

4.1.4 2.1.4. Class

Is the identification number (integer) of a type of an incident. The column don't have any blank row. The class of incidents vary from 111 to 3240, and for our data, we have 285 different classes of incident

```
In [5]: class_id_null_true = crimes[crimes["Class"].isnull()]["Class"]
print("Number of rows which the column 'Class' is blank:", len(class_id_null_true))
print("Type of the values in the 'Class' column:", crimes["Class"].dtypes)

class_id_unique = crimes["Class"].unique()
print("Number of different classes of incidents:", len(class_id_unique))

print("Minimum value of 'Class':", crimes["Class"].min())
print("Maximun value of 'Class':", crimes["Class"].max())
```

Number of rows which the column 'Class' is blank: 0
Type of the values in the 'Class' column: int64
Number of different classes of incidents: 285
Minimum value of 'Class': 111
Maximun value of 'Class': 3240

4.1.5 2.1.5. Class Description

Is the description of each Class of incidents. As expected, it has a 1-to-1 relationship with the Class id, e.g., each Class id has one different description. Therefore, this column also don't have any blank row and we have 285 different classes of incident.

```
In [6]: class_description = crimes['Class Description']
class_description_null_true = class_description[class_description.isnull()]
print("Number of rows which the column 'Class Description' is blank:", len(class_description_null_true))

class_description_unique = crimes["Class Description"].unique()
print("Number of different classes of incidents:", len(class_description_unique))
```

Number of rows which the column 'Class Description' is blank: 0
Number of different classes of incidents: 285

4.1.6 2.1.6. Police District Name

Cointains the name of the district associated with the police report, e.g., the district where the crime was reported and, therefore, the district where the crime occur. There is no blank row in this column. This column assumes 8 different values, they are the 6 districts listed in [Montgomery Count Police site](#), plus the 'TAKOMA PARK' and 'OTHER'. Once Takoma Park isn't a district, but only a city, it may be listed separately propably because his own [police department](#) is big enough to be counted, as not every city in US has his own police department.

```
In [7]: district_name = crimes['Police District Name']
district_name_null_true = district_name[district_name.isnull()]
print("Number of rows which the column 'Police District Name' is blank:", len(district_name_null_true))

district_name_unique = district_name.unique()
print("Number of different District Names:", len(district_name_unique))
print("List of District Names: ", district_name_unique)
```

```
Number of rows which the column 'Police District Name' is blank: 0
Number of different District Names: 8
List of District Names:  ['OTHER' 'GERMANTOWN' 'MONTGOMERY VILLAGE' 'BETHESDA' 'ROCKVILLE'
'SILVER SPRING' 'WHEATON' 'TAKOMA PARK']
```

4.1.7 2.1.7. Block Address

Is the address of the incident, containing basically the number, the road and the abbreviation of the road type. It could be considered the area with the smaller granularity with human meaning. Although some rows don't have the number, none of them is blank, e.g., every row has at least the name and the type of the road where the incident occurs. We have 8,143 different Block Addresses in our data.

```
In [8]: block_address = crimes['Block Address']
        block_address_null_true = block_address[block_address.isnull()]
        print("Number of rows which the column 'Block Address' is blank:", len(block_address_null_true))

        block_address_unique = block_address.unique()
        print("Number of different Block Address:", len(block_address_unique))
```

```
Number of rows which the column 'Block Address' is blank: 0
Number of different Block Address: 8143
```

4.1.8 2.1.8. City

Is the city where the incident occurs, it complements the Block Address to make the full address of the incident. Also there is no blank row for this column and we have 32 different cities listed in our crimes data.

```
In [9]: city = crimes['City']
        city_null_true = city[city.isnull()]
        print("Number of rows which the column 'City' is blank:", len(city_null_true))

        print("Number of different cities:", len(city.unique()))
```

```
Number of rows which the column 'City' is blank: 0
Number of different cities: 32
```

4.1.9 2.1.9. State

Is the state where the incident occurs, it complements the Block Address to make the full address of the incident. Also there is no blank row for this column. Although the Montgomery County is one of the counties of the Maryland State, one row of our data corresponds to a crime that occurred in the Florida State, probably because, for some reason, the victim reported the crime in a police department of the Montgomery County.

```
In [10]: state = crimes['State']
state_null_true = state[state.isnull()]
print("Number of rows which the column 'State' is blank:", len(state_null_true))

print("Number of different states:", len(state.unique()))
print("List of the states: ", state.unique())

#The expression "state == 'FL'" returns a list of booleans indicating if each index is
#Therefore, when we pass that list as the index of 'state' itself, is returned the list
#Then, we took the lenght of this list to tell the number of incidents of that state.
print("Number of incidents occured in Florida: ", len(state[state == 'FL']))
```

Number of rows which the column 'State' is blank: 0
Number of different states: 2
List of the states: ['MD' 'FL']
Number of incidents occured in Florida: 1

4.1.10 2.1.10. Zip code

Is the zip code of the address, it also complements the others address informations to make the full address of the incident. The type of this column is 'float64', but it's common sense that the zip code is a integer number. So, for better manipulation of this data, we created a new column, named 'zip' to store the zip code as integer. For the conversion it was necessary to fill the NaN with 0.

This column has a total of 30 rows with missing values, which represents only 0,13% of the total number of rows. Some rows that has this column as a blank value is because the address don't have a obvious zip code, as the row of index 5, which correspond to the Incident ID 200931009 and took place in the middle of the 270 highway, in Gaithersburg city, or the row of index 2300, which correspond to the Incident ID 200928686 and took place in a marathon circuit. But others rows that also have this column as a blank value actually have a obvious zip code, as the row of index 3026, which correspond to the Incident ID 200929516 and took place in 6300 New Hampshire Avenue, Takoma Park city, which the zip code is 20912, also, the row of index 23365 which correspond to the Incident ID 201070054 and took place in 10500 Lester Street, Silver Spring city, which the zip code is 20902. It's not possible to say for sure why this information isn't registred in these rows, but probably is just because by the time of registering those data the zip code information wasn't available by any reason.

As expected, many block addresses has the same zip code and in the data we are analyzing we have 50 different zip codes.

```
In [11]: #The "fillna" function fills all the 'NaN' rows with the value passed as argument.
#In our case, we are filling with 0's. This is necessary to convert from "float" to "int"
#The result of the conversion is stored in a new column, named 'zip'
crimes['zip'] = crimes['Zip Code'].fillna(0).astype(int)
zip_code = crimes['Zip Code']

zip_code_null_true = zip_code[zip_code.isnull()]
print("Type of the 'Zip Code' column:", zip_code.dtype)
```

```

print("Number of rows which the column 'Zip Code' is blank:", len(zip_code_null_true))
print("Percentage of blank zip codes:", (len(zip_code_null_true)/len(zip_code))*100 )
print("Number of different zip codes:", len(zip_code.unique()))

montgomery_county_land_area = 1270
print("Mean area of each zip code (kmš):", montgomery_county_land_area/len(zip_code.uni

```

```

Type of the 'Zip Code' column: float64
Number of rows which the column 'Zip Code' is blank: 30
Percentage of blank zip codes: 0.12837519791176344
Number of different zip codes: 50
Mean area of each zip code (kmš): 25.4

```

4.1.11 2.1.11. Agency

Is the assigned police department of the incident. There is no blank row for this column. There is only 5 different agencies assigned for those incidents and the code below list them. As it can be see, the majority of the incidents are assigned to the Montgomery County Police Department, as would be expected, once the data set is provided by the Montgomery County.

```

In [12]: agency = crimes['Agency']

agency_null_true = agency[agency.isnull()]
print("Number of rows which the column 'Agency' is blank:", len(agency_null_true))
print("List of different agencies:", agency.unique())

# Manually fill a dict with the agencies descriptions
agency_description = {'MCPD': 'Montgomery County Police Department',
                     'GPD': 'Gaithersburg Police Department',
                     'RCPD': 'Rockville City Police Department',
                     'MCFM': 'Montgomery County Fire Marshal',
                     'P': 'P'}

#import the matplotlib to make a pie chart
import matplotlib.pyplot as plt

#List to stores the percentage of each agency
sizes = [(len(agency[ agency == 'MCPD' ])/len(agency))*100,
         (len(agency[ agency == 'GPD' ])/len(agency))*100,
         (len(agency[ agency == 'RCPD' ])/len(agency))*100,
         (len(agency[ agency == 'MCFM' ])/len(agency))*100,
         (len(agency[ agency == 'P' ])/len(agency))*100]

#Plot a pie chart with the percentages as a size of each piece
fig1, ax1 = plt.subplots(figsize=(10,6))
ax1.pie(sizes, autopct='%1.1f%%', pctdistance=1.1,
        shadow=True, startangle=90)

```

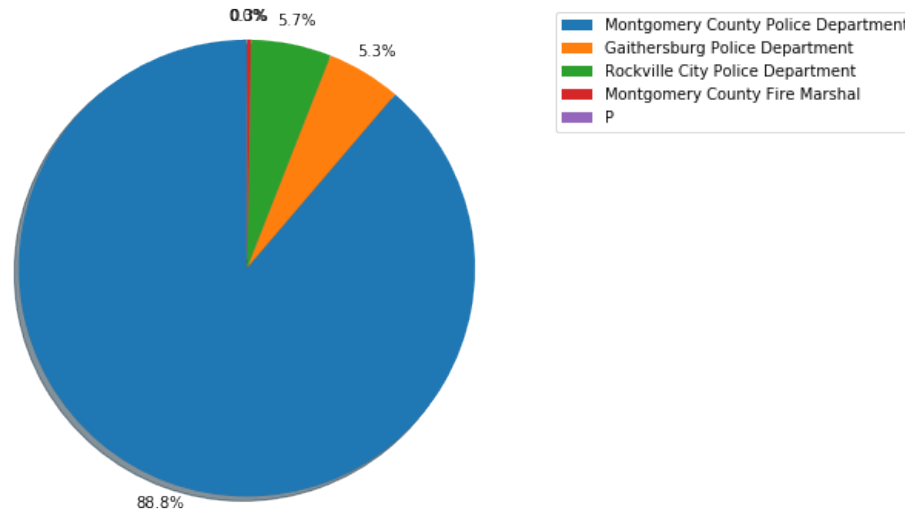


```
ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
```

```
#Plot the legend outside the figure. Labels are made equal to the descriptions of the a
plt.legend(labels=agency_description.values(), bbox_to_anchor=(0.85,1.025))
plt.show()
```

Number of rows which the column 'Agency' is blank: 0

List of different agencies: ['MCPD' 'GPD' 'RCPD' 'MCFM' 'P']



4.1.12 2.1.12. Place

Is a description of the place where the incident occur, examples: 'Street - In vehicle', 'Bar/Night Club', 'Residence - Other' and so on. None of the rows has a blank value on this column. The 'Place' is standardized in 82 different descriptions and the most common is 'Residence - Single Family' with 2729 occurrences.

```
In [13]: place = crimes['Place']
         place_null_true = place[place.isnull()]
         print("Number of rows which the column 'Place' is blank:", len(place_null_true))
         print("Number of different places descriptions:", len(place.unique()))

#The value_counts() function returns a series where the index is the unique values of t
#and the values are the number of occurrence of this value.
#The series returned is sorted, where the first element is the one with most occurrence
print("The place description most common is", place.value_counts().index[0],
      "with", place.value_counts().iloc[0], "occurrences.")

print("The place description least common is", place.value_counts().index[-1],
      "with", place.value_counts().iloc[-1], "occurrences.")
```

Number of rows which the column 'Place' is blank: 0
Number of different places descriptions: 82
The place description most common is Residence - Single Family with 2729 occurrences.
The place description least common is Parking Lot - Park & Ride with 2 occurrences.

4.1.13 2.1.13. Sector

Shows police sector name, which is a subset of the District. The sector are represented by a uppercase letter and in the dataset it varies from 'A' to 'T', but contains only 16 different sectors, as some letters in the range doesn't appear in the column. The Sector column has 46 rows with no data and we couldn't find any pattern for those, but as they represent only 0,19% of all the incidents it shouldn't be representative. Once the Montgomery County has an land area equal to 1.270 kmš, according to [Open Data Network](https://www.opendatanetwork.com/entity/0500000US24031/Montgomery_County_MD/geographic), each sector has a mean area of 79,37 kmš.

```
In [14]: #data manually founded in:
#https://www.opendatanetwork.com/entity/0500000US24031/Montgomery_County_MD/geographic.
montgomery_county_land_area = 1270
sector = crimes['Sector']
sector_null_true = sector[sector.isnull()]
print("Number of rows which the column 'Sector' is blank:", len(sector_null_true))
print("Percentage of blank sectors:", (len(sector_null_true)/len(sector))*100)
print("Number of different sectors:", len(sector.unique()))
print("Mean area of each sector (kmš):", montgomery_county_land_area/len(sector.unique()))
```

Number of rows which the column 'Sector' is blank: 46
Percentage of blank sectors: 0.1968419701313706
Number of different sectors: 16
Mean area of each sector (kmš): 79.375

4.1.14 2.1.13. Beat

Represents a police patrol area and is a subset of Sector. The beat initiate with a number that represent the district which the beat belongs, than it has a uppercase letter that is correspondent to the sector which that beat belongs and finally another number which distinguish the beat itself inside each sector. Normally each sector has 2 to 4 beats. There is 8 rows where the Beat column don't have any data, which is less then the number of blank sectors, therefore, is possible to determine the sector by the beat information. The mean area for each beat is 29,53 kmš.

```
In [15]: beat = crimes['Beat']
beat_null_true = beat[beat.isnull()]
print("Number of rows which the column 'Beat' is blank:", len(beat_null_true))
print("Percentage of blank beats:", (len(beat_null_true)/len(beat))*100)
print("Number of different beats:", len(beat.unique()))
print("Mean number of beats for each sector:", len(beat.unique())/len(sector.unique()))
print("Mean area of each beat (kmš):", montgomery_county_land_area/len(beat.unique()))
```

Number of rows which the column 'Beat' is blank: 8
Percentage of blank beats: 0.03423338610980359
Number of different beats: 43
Mean number of beats for each sector: 2.6875
Mean area of each beat (kmš): 29.53488372093023

4.1.15 2.1.14. PRA

It's a acronym for Police Response Area and it's a subset of a beat. It's an integer number, varying from 0 to 808, although in the dataset is stored as a float (because in pandas, a integer series can't have a 'NaN' value). There is 8 rows where the PRA column don't have any data, which is even less then the number of blank beats, but we couldn't find any pattern that match one PRA number to a specific beat. There is 696 different Police Response Areas in our dataset, which correspond a mean number of 16,18 PRAs for each beat and an mean area of 1,82 kmš for each PRA.

```
In [16]: pra = crimes['PRA']
        pra_null_true = pra[pra.isnull()]
        print("Minimum value of Police Response Area:", pra.min())
        print("Maximun value of Police Response Area:", pra.max())
        print("Number of rows which the column PRA is blank:", len(pra_null_true))
        print("Number of different police response areas:", len(pra.unique()))
        print("Mean number of police response areas for each beat:", len(pra.unique())/len(beat))
        print("Mean area of each police response area (kmš):", montgomery_county_land_area/len(pra.unique()))
```

Minimum value of Police Response Area: 0.0
Maximun value of Police Response Area: 808.0
Number of rows which the column PRA is blank: 6
Number of different police response areas: 696
Mean number of police response areas for each beat: 16.186046511627907
Mean area of each police response area (kmš): 1.8247126436781609

4.1.16 2.1.15. Start Date / Time

Is the date / time of the beggining of the incident. It has the same syntax of 'Dispatch Date / Time'. There is no blank row for this column, as would be expected, once each incident has to be initiated in some date and time. Also, once that a crime can be reported after his occurrence, we can have values for this column from before 2013 and this actually is the case. The minimum value of this column is '01/01/1985 06:00 AM' and there is 55 incidents from before 2013. The code below list the types of crimes that have their start date before 2013.

```
In [17]: start_date = crimes['Start Date / Time']
        start_date_null = start_date[start_date.isnull()]
        print("Number of rows which the column 'Start Date / Time' is blank:", len(start_date_null))
        print("Minimum value of 'Start Date / Time':", start_date.min())
        print("Maximun value of 'Start Date / Time':", start_date.max())

        print("Number of incidents that actually occurred before 2013:",
```

```
len(start_date[start_date < '01/01/2013 00:00:00 AM']) )
print("Types of crimes that has the start date before 2013: ",
      class_description[start_date < '01/01/2013 00:00:00 AM'].unique())
```

Number of rows which the column 'Start Date / Time' is blank: 0

Minimum value of 'Start Date / Time': 01/01/1985 06:00:00 AM

Maximum value of 'Start Date / Time': 12/31/2013 12:58:00 PM

Number of incidents that actually occurred before 2013: 55

Types of crimes that has the start date before 2013: ['LARCENY AUTO PART UNDER \$50' 'FAMILY OFFENSE' 'HARASSMENT/STALKING' 'FORGERY/CNTRFT - IDENTITY THEFT' 'RAPE-FORCE' 'LOST PROPERTY' 'SEX OFFENSE - SEX ASSAULT' 'SEX OFFENSE - ALL OTHER SEX OFFENSE' 'POL INFORMATION' 'VANDALISM-OTHER' 'EMBEZZLEMENT \$300 OR MORE' 'FORGERY/CNTRFT-CRDT CARDS' 'LARCENY FROM BUILDING OVER \$200' 'FORGERY/CNTRFT-CHECKS']

4.1.17 2.1.16. End Date / Time

Is the date / time of the end of the incident. It has the same syntax of 'Dispatch Date / Time'. This column actually have a lot of blank rows, 10.178 in total, which represents 43,55% of the incidents. There could be many reasons for this column is left in blank, and here are some: the incident actually didn't ended yet (a missing person, for example), the end date / time don't make sense for the type of the crime or the end date / time couldn't be estimated.

```
In [18]: end_date = crimes['End Date / Time']
end_date_null = end_date[end_date.isnull()]
print("Number of rows which the column 'End Date / Time' is blank:", len(end_date_null))
print("Percentage of rows with blank 'End Date / Time':", (len(end_date_null)/len(end_date)))
```

Number of rows which the column 'End Date / Time' is blank: 10178

Percentage of rows with blank 'End Date / Time': 43.55342547819761

4.1.18 2.1.17. Latitude

Is the latitude coordinate of the location of the incident. It's displayed in degrees, in a decimal form. It's assumed that positive values correspond to the north.

There is 161 blank rows in this column, which represents, approximately, 0,7% of all incidents. Combining with the longitude, forms the most precise information of the location of the incident, once that tell the exact coordinates of the incident.

We couldn't find a pattern of the missing values, once that all 6 districts has at least one row with no latitude information and 46 different class descriptions has at least one row with the Latitude column without data.

Also, it's interesting to think about how those coordinates are estimated. We guess that it's estimated based in the block address, when this one has the address number, once in all the rows that have a latitude value it also has a address number. But we should be in mind that a police report is usually filled based in the victims description or even by some witness and it's common not to know the precise location of the incident or even be mistaken by the stress of the event. Therefore, this coordinates should be taken with caution, considering the possibility of a estimation error.

```

In [19]: latitude = crimes['Latitude']
latitude_null = latitude[latitude.isnull()]
print("Number of rows which the column 'Latitude' is blank:", len(latitude_null))
print("Percentage of rows with blank 'Latitude':", (len(latitude_null)/len(latitude))*100)

print("List of districts which has blank 'Latitude':", district_name[latitude.isnull()])
print("Number of class descriptions which has blank 'Latitude':", len(class_descriptions[latitude.isnull()]))

#The "isnull()" function returns a list of booleans, where True means that in that index there is a null element.
#Therefore, his negation (~) indicates the index which don't have a null element.
#So, crimes[~latitude.isnull()] returns all rows that don't have the latitude column as null.
#Then, in these rows, we call the 'isnull()' function to the 'Address Number' column
#And finally, the function 'any()' returns True if there is any True element in a list
print("There is a row with 'Latitude' value and no 'Address Number'?",
      crimes[~latitude.isnull()]['Address Number'].isnull().any() )

```

Number of rows which the column 'Latitude' is blank: 161

Percentage of rows with blank 'Latitude': 0.6889468954597971

List of districts which has blank 'Latitude': ['OTHER' 'GERMANTOWN' 'MONTGOMERY VILLAGE' 'BETHESDA' 'SILVER SPRING' 'WHEATON']

Number of class descriptions which has blank 'Latitude': 46

There is a row with 'Latitude' value and no 'Address Number'? False

4.1.19 2.1.18. Longitude

Is the longitude coordinate of the location of the incident. It's displayed in degrees, in a decimal form. It's assumed that negative values correspond to the east. It has the same properties of the Latitude column, proved by the code below.

```

In [20]: longitude = crimes['Longitude']
longitude_null = longitude[longitude.isnull()]

print("Percentage of rows with blank 'Longitude':", (len(longitude_null)/len(longitude))*100)

print("List of districts which has blank 'Longitude':", district_name[longitude.isnull()])
print("Number of class descriptions which has blank 'Longitude':", len(class_descriptions[longitude.isnull()]))

print("There is a row with 'Longitude' value and no 'Address Number'?",
      crimes[~longitude.isnull()]['Address Number'].isnull().any() )

```

Percentage of rows with blank 'Longitude': 0.6889468954597971

List of districts which has blank 'Longitude': ['OTHER' 'GERMANTOWN' 'MONTGOMERY VILLAGE' 'BETHESDA' 'SILVER SPRING' 'WHEATON']

Number of class descriptions which has blank 'Longitude': 46

There is a row with 'Longitude' value and no 'Address Number'? False

4.1.20 2.1.19. Police District Number

It's the number of the police district. The 6 district as listed in the [Montgomery Count Police site](#) has his own number, and it's shown as 1D, 2D and so on. The Takoma Park district has the 'TPPD' value in this column and the district classified as 'OTHER' in the Police District Name column has also the value 'OTHER' in the Police District Number column. As it would be expected, this column has the same properties as the Police District Name column.

```
In [21]: district_number = crimes['Police District Number']
        district_number_null_true = district_number[district_number.isnull()]
        print("Number of rows which the column 'Police District Number' is blank:", len(district_number_null_true))

        district_number_unique = district_number.unique()
        print("Number of different District Numbers:", len(district_number_unique))
        print("List of District Numbers: ", district_number_unique)
```

```
Number of rows which the column 'Police District Number' is blank: 0
Number of different District Numbers: 8
List of District Numbers:  ['OTHER' '5D' '6D' '2D' '1D' '3D' '4D' 'TPPD']
```

4.1.21 2.1.20. Location

It's the combination of the Latitude and Longitude columns, displayed in the format: "(latitude, longitude)". And has the same number of blank cells.

```
In [22]: location = crimes['Location']
        location_null = location[ location.isnull() ]
        print("Number of rows which the column 'Location' is blank:", len(location_null))
```

```
Number of rows which the column 'Location' is blank: 161
```

4.1.22 2.1.21. Address Number

It's the number of the address, the same one informed in the first information of the block address. It has 132 rows with no information and it's stored as float64. The result of the code below show those informations.

```
In [23]: address_number = crimes['Address Number']
        address_number_null = address_number[ address_number.isnull() ]
        print("Type of the 'Address Number' column:", address_number.dtype)
        print("Number of rows which the column 'Address Number' is blank:", len(address_number_null))

        #Defines a function to return the number of the block address
        #The number of the block address is always the first word in the 'Block Address' column
        #therefore, the function returns the conversion to float of the first word
        #Nonetheless, there are some 'Block Address' that don't have the number and instead, al
        #name in the first word. So, if the conversion to float isn't possible, return a 'NaN'
```

```
def block_number(row):
    try:
        return float(row.split(' ')[0])
    except ValueError:
        return float('NaN')

#We want to check if the values in the "Address Number" column is equal to the number of
#column. The problem with that isn't possible to compare 'NaN' values. So, we have to:
#(a) fill the 'NaN' with some value (eg: 0) and then to the comparison, or
#(b) take the list of the nonnull values
#we choose the second approach

#First of all, the apply method, passing the 'block_number' function returns the number
#So, the "~(block_address.apply(block_number).isnull())" return a list of booleans where
#index don't have a 'NaN' value. We take this list and pass it as a index of the
#"block_address.apply(block_number)" series. Then, we compared to the list of non-null
#'Address Number' column. But this comparison returns a list of booleans and we use the
#to check if all elements of the list is True.
print('The first information of the Block Address column is the same value in the Address Number?')
print((block_address.apply(block_number)[ ~(block_address.apply(block_number).isnull())
address_number[~address_number.isnull()]).all() )
```

Type of the 'Address Number' column: float64

Number of rows which the column 'Address Number' is blank: 132

The first information of the Block Address column is the same value in the Address Number? True

5 3. Analyzing the times of crimes

According to [dataMontgomery site](#) the Dispatch Date / Time column inform "The actual date and time a Officer was dispatched". Therefore, is a little bit different of the date / time of the occurrence of the crime. Because of this, we will make two different analysis: (1) Considering the the date / time of the dispatch of a Officer, which will be assumed to be the time of the police report (but it should be in mind the errors of this assumption, as a crime can be reported in a date/time and a Officer dispatched later on) and (2) Considering the date / time of the start of the crime, which will be assumed to be the time of the occurrence of that crime.

5.1 3.1 Analyzing the times of the crimes reports

5.1.1 3.1.1 What day of the week are the most crimes reported on? (ie Monday, Tuesday, etc)

As it can be seen in the code below, the day of the week were the most crimes are reported is Tuesday, follow closed by the rest of the weekdays. The weekends has considerably fewer reports.

This can be mainly explained by two reasons: - A lot of crimes don't have the urgency to be reported on, so the victimns choose a more convenient day to report the crime. So, it will be expected that the first days of the week has more reports, because of the accumulated crimes of

the weekend (obvious, the ones that doesn't have the urgency to be reported). - Even if the victim reported the crime on the weekend it's possible that the Officer is only dispatched if the crimes has the urgency of it, otherwise, the Officer can be dispatched only in the weekdays. If this is the case, it also be expected that the firts days of the weekend has more occurrences, once it's accumulate the occurrences of the weekend. This also can explain why Tuesday has more number than Monday, because this last one is probably the day off of the Officers that worked on the weekend.

```
In [24]: dispatch_datetime = pd.to_datetime(crimes['Dispatch Date / Time'])
        print("List of the days of the week with most crimes reports:")
        print(dispatch_datetime.dt.weekday_name.value_counts())
```

List of the days of the week with most crimes reports:

Tuesday	3836
Monday	3734
Wednesday	3611
Friday	3594
Thursday	3404
Saturday	2807
Sunday	2383

Name: Dispatch Date / Time, dtype: int64

5.1.2 3.1.2 During what time of day are the most crimes reported?

To answer this question we need first to define what is 'time'. We considered that for a better analysis, it should be considered the parts of the day. Therefore, we categorized as follows: - **Morning:** from 5:01 am to 12:00 am; - **Afternoon:** from 12:01 pm to 5:00 pm; - **Evening:** from 5:01 pm to 9:00 pm; - **Night:** from 9:01 pm to 5:00 am.

For doing this categorization, we build a function to analyze the time of each row of the dispatch_datetime (which was already in the 'datetime' format) and return the period of the day. For this we use the apply() method of the DataFrame.

With the categorization in a series, we use the value_counts() method to summarize the quantities of crimes per periodo of the day. As we can see, the period with most crimes reports is the Morning, followed by Night. The period with least crimes reports is Evening.

First of all we should be in mind that these two periods (Morning and Night) are the ones with most hours, both of them has 7 hours in total. If we took a mean value per hour for each period, the Afternoon is the one with most (relatively) crimes and Night are the least one. For complement this visualization, it's also important to see how many crimes is reported per hour, and therefore, we also show this list.

It's possible to see that in the period from 7:00 am to 18:00 pm the number of reports is almost equivalent, with the hours in the early morning (7-9 am) having a litle more reports followed by all the hours classified as Afternoon. The late night period (0:00 am to 6:00 am) have clearly the least number of reports. The reason for this could be similar to the one gave in the 3.1.1, which it means that only the urgent crimes have Officer dispatched in the late night and for this reason, the early morning hours have more Officers dispatched because they are attending the reports done in the late night. Thereafter, the late morning don't have much Officers dispatched, probably because in the morning is the actual period with least crimes. Then, in the Afternoon the crimes began to occur, which leads to more Officers dispatched.


```

In [25]: #Function that returns the period of the day, based on the following:
#Morning: from 5:01 am to 12:00 am;
MORNING = 12
#Afternoon: from 12:01 pm to 5:00 pm;
AFTERNOON = 17
#Evening: from 5:01 pm to 9:00 pm;
EVENING = 21
#Night: from 9:01 pm to 5:00 am.
NIGHT = 5

def period(row):
    parts = ['Morning', 'Afternoon', 'Evening', 'Night']

    #hour receives the hour of the row plus one if minute or second is > 0
    hour = row.hour + (1 if (row.minute > 0 or row.second > 0) else 0)

    #return the string of the list based on the hour
    if hour > NIGHT and hour <= MORNING:
        return parts[0]
    elif hour > MORNING and hour <= AFTERNOON:
        return parts[1]
    elif hour > AFTERNOON and hour <= EVENING:
        return parts[2]
    else:
        return parts[3]

#apply the period function to the dispatch_datetime for calculate the periods of the day
#then, use the value_counts() function to return the list of periods with more rows
print("List of the periods of a days with most crimes reports:")
print(dispatch_datetime.apply(period).value_counts())

#stores the mean crimes per hour of each period
periods_normalized = pd.DataFrame( data={'Dispatch Date':
    { 'Morning':
        dispatch_datetime.apply(period).value_counts()['Morning'] / (MO
        'Afternoon': dispatch_datetime.apply(period).value_counts()['Aft
        'Evening': dispatch_datetime.apply(period).value_counts()['Eveni
        'Night': dispatch_datetime.apply(period).value_counts()['Night']

print("\nList of the periods of a day with most crimes reports, considering the mean va
print(periods_normalized.sort_values(by='Dispatch Date',ascending=False)['Dispatch Date

import matplotlib.pyplot as plt

#Plot a histogram with the hours of the day as the x axis and the number of crimes in t
fig = plt.figure(figsize=(12,6))
ax = fig.add_subplot(1,1,1)

```

```

ax.hist(dispatch_datetime.dt.hour, range=(0, 24), bins=24, align='left', edgecolor='gray')
ax.set_xticks(range(24))
ax.set_xlabel('Hour of the day')
ax.set_ylabel('Number of crimes')
ax.set_title('Number of crimes for each hour of the day')
plt.show()

```

List of the periods of a days with most crimes reports:

```

Morning      7193
Night        6120
Afternoon    5727
Evening      4329
Name: Dispatch Date / Time, dtype: int64

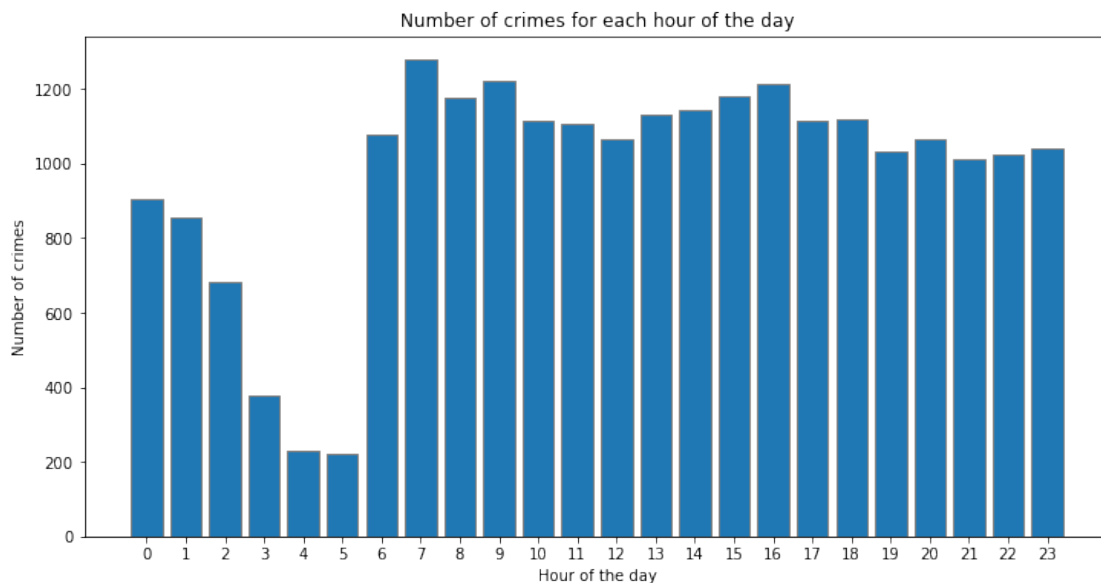
```

List of the periods of a day with most crimes reports, considering the mean value per hour:

```

Afternoon    1145.400000
Evening      1082.250000
Morning      1027.571429
Night        765.000000
Name: Dispatch Date, dtype: float64

```



5.1.3 3.1.3 During what month are the most crimes reported?

To better visualize the results, we create a function to return the name of the month based on his number, once that this is the return of datetime properties. Then, we list the months with most crimes reports.

As it can be seen, there is no information from january to june, which we can conclude that in this period the reports weren't been registered. Therefore, the registration of the reports began in

July. We can note that the months from september to december has a equivalent number of crimes reports (varying from 3904 to 4075) and only july has significant less crimes reports, and we think that this can be explain by the fact that was the begging of the registration and as such, it's normal to have problems and a late begging, especially when it aggregate various players (in this case, the various districts).

```
In [26]: #Function that returns the name of the month
def month_name(row):
    months = ['January', 'February', 'March', 'April', 'May', 'June',
              'July', 'August', 'September', 'October', 'November', 'December']

    return months[row.month - 1]

print('List of months with most crimes reports:')
print(dispatch_datetime.apply(month_name).value_counts())
```

```
List of months with most crimes reports:
October      4075
August       4002
November     3941
September    3927
December     3904
July         3520
Name: Dispatch Date / Time, dtype: int64
```

5.2 3.2 Analyzing the times of the occurance of crimes

As stated before, in this section we will consider the 'Start Date / Time' column as the actual time of the occurance of the crime.

5.2.1 3.2.1 What day of the week are the most crimes committed on? (ie Monday, Tuesday, etc)

In this case we can see that Friday is the day with most crimes committed and sunday the day with least crimes. This is much more as expected, once it can be said that those are the day with most and least people in the streets which is direct proportional to the amount of crimes.

Taken a closer look, we can see that Friday distant a little bit of the others days as the one with most crimes and then Tuesday to Saturday has basically the same amount of crimes. Then we have Monday and Sunday with less crimes with a close amount.

```
In [27]: crime_datetime = pd.to_datetime(crimes['Start Date / Time'])
print("List of the days of the week with most crimes committed:")
print(crime_datetime.dt.weekday_name.value_counts())
```

```
List of the days of the week with most crimes committed:
Friday      3719
Tuesday     3463
Wednesday   3430
```

```

Thursday    3349
Saturday    3343
Monday      3157
Sunday      2908
Name: Start Date / Time, dtype: int64

```

5.2.2 3.2.2 During what time of day are the most crimes committed?

In this case the analysis of the actual time of the crime occurrence is very different then the dispatch time as this one is much more as expected. The period with most crimes is the Night, by far, and the period with less crimes is the Morning. Considering a mean per hour, the period with most (relative) crimes is the Evening and the period with less crimes is the Morning, by far.

Analyzing the list of the crimes per hour clarify that the late night period (from 02:00 am to 05:00 am) push down the mean value of the Night period as the period of early night (from 09:00 pm to 01:00 am) concentrate the most crimes. It also confirms the fact that the hours of the evening has a occurrence of much crimes.

```

In [28]: print("List of the periods of a days with most crimes committed:")
         print(crime_datetime.apply(period).value_counts())

#stores the mean crimes per hour of each period
periods_normalized = pd.DataFrame( data={'Start Date':
    { 'Morning': crime_datetime.apply(period).value_counts()['Morning']
    'Afternoon': crime_datetime.apply(period).value_counts()['Afternoon']
    'Evening': crime_datetime.apply(period).value_counts()['Evening']
    'Night': crime_datetime.apply(period).value_counts()['Night'] }

print("\nList of the periods of a day with most crimes committed, considering the mean
print(periods_normalized.sort_values(by='Start Date',ascending=False)['Start Date'])

import matplotlib.pyplot as plt

#Plot a histogram with the hours of the day as the x axis and the number of crimes in t
fig = plt.figure(figsize=(12,6))
ax = fig.add_subplot(1,1,1)
ax.hist(crime_datetime.dt.hour, range=(0, 24), bins=24, align='left', edgecolor='gray',
ax.set_xticks(range(24))
ax.set_xlabel('Hour of the day')
ax.set_ylabel('Number of crimes')
ax.set_title('Number of crimes for each hour of the day')
plt.show()

```

List of the periods of a days with most crimes committed:

```

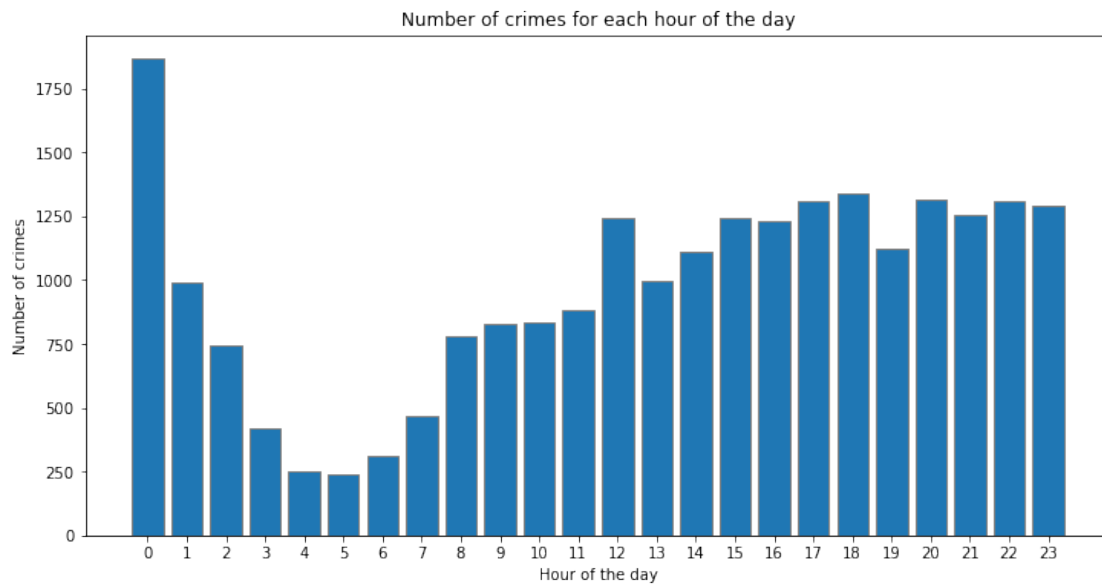
Night        7841
Afternoon    5620
Evening      5097

```

```
Morning      4811
Name: Start Date / Time, dtype: int64
```

List of the periods of a day with most crimes committed, considering the mean value per hour:

```
Evening      1274.250000
Afternoon    1124.000000
Night        980.125000
Morning      687.285714
Name: Start Date, dtype: float64
```



5.2.3 3.2.3 During what month are the most crimes committed?

This analysis is the most similar once it takes a large period of time, therefore, the difference between the occurrence of the crime and the dispatch of a Officer is almost irrelevant. But there is still some nuances that should be observed. It appears all the months of the year, meaning that in the period from July to December of 2013 was reported crimes committed in all the months of the year, which is kindly expected, once we have a universe of 23369 crimes and a lot of reports (55) from years before 2013.

The months from August to November have a similar quantity of crimes (3846 to 3993), which can be said as the regular amount of crimes per month. July has a less incidence of crimes probably because was the first month of the registration and therefore it's normal to have some missing information. Finally, December also has a less number of crimes probably because some of the crimes occurred in this month, especially the ones between christmas and new years eve, will be reported in January of 2014.

```
In [29]: print('List of months with most crimes committed:')
         print(crime_datetime.apply(month_name).value_counts())
```

List of months with most crimes committed:

October	3993
August	3971
September	3883
November	3846
December	3560
July	3450
June	307
May	108
January	95
April	66
March	56
February	34

Name: Start Date / Time, dtype: int64

5.3 3.3 Other analysis

The dataset contains another column that deal with the time of the crime, is the 'End Date / Time'. According to [dataMontgomery site](#) their description is "Occurred to date/time". Also, from our analysis in the section 2.1 we know that this column has a lot of missing values.

We couldn't find a specific pattern for the missing values and it's not clear if this column contain the date / time of the conclusion (or to solve) of the crime or it's the actual date / time that the crime ended. Also, it's not clear if the row is updated as this information is known or not. Therefore, it's hard to make any conclusions of this column, especially considering the large amount of missing values.

We think that a column that informs the conclusion of the investigation would be helpful in the analysis of the crimes, as would be possible to know if the crimes occurrences have been effectively dealt with.

6 4. Analyzing locations of crimes

The dataset that we are analyzing has many (12, to be precise) columns about the location of the crime. So, in order to define which column will be used to analyze the location and show some information about it, we need to obey some criteria. Below we list the used ones:

- **Comprehensibility:** the location needs to be a information that your public of interest can understand and use in his normal day, without having to research about it.
- **Public of interest:** is important to know which type of persons you want to reach. If this project is designed to reach police officers, we can define the location as the PRA, nevertheless, if is designed to reach a 'regular' citizen, they wouldn't have an idea of what PRA, so, it's better to define the location with some regular address information.
- **Granularity:** defines the amount of area we want to analyze. A large area wouldn't have much effect, because inside of that area we probably will have very distinct patterns of the crimes, for example, if a say that the Montgomery County has much crimes, compared to Los Angeles County, it wouldn't be that helpful, once the Silver Spring city has a total different

crime pattern then Laurel city. In contrary, a area too small wouldn't be that representative, once that is hard to locate (i can't say when i am in the location defined by the coordenates 35.234, -75.478) and restrict the quantity of people that can benefit from that information, once very few people use one street, for example.

- **Missing values:** the data we will use to define the location should be representative, we can't use a column which has a very large number of missing values as we don't know if those data exists what pattern will lead the dataset to.

We considered that the public of readers of this project is the 'regular' citizen. Based on those criterias, we analyzed the columns to choose one to use for the study of the location.

6.1 4.1. Columns

6.1.1 4.1.1. Sector, beat and PRA

Those columns don't have a meaning for the 'regular' citizen and as such, it wouldn't be helpful to analyze the location based on those.

6.1.2 4.1.2. Police district name and number

Probably an 'regular' citizen would know the location of the Police District and their numbers, although it wouldn't be that universal as a regular address information, but they represent a large area, with many cities inside of them. There are only 6 districts for the whole Montgomery County, which will lead to an average of more than 5 cities per district.

6.1.3 4.1.3. Block address

Although is meaningful to the citizen, it's a very small area, a point actually, once it has the number of the address. Therefore, we think it wouldn't be that helpful.

6.1.4 4.1.4. City

It's meaningful to every citizen and don't cover an area that large or small and don't have any missing values. For that reason, it's a good way to analyze the location of the crimes.

6.1.5 4.1.5. State

It's a very large area, larger then the County itself. So, it's not useful for the location analysis.

6.1.6 4.1.6. Zip code

It should be meaningful to almost every citizen, at least the zip codes that he lives and works in. Don't cover an area too large or too small, once that, in average, a zip code has 25,4 kmš. Has only 0,13% of missing values. We can see that this column should be a good one to analyze the location of crimes.

6.1.7 4.1.7. Latitude, Longitude and Location

They actually carry the same information, that is the 'exact' location of the crime in a coordinate of latitude and longitude, in degrees. As we already said before, besides, that information represents a very small area, actually a unique point and it can be hard to know where this coordinate correspond to. Even through, with the help of a map, we can visualize the crimes, as a crime map, and visually can infer what regions have more crimes. Therefore, it can be a good way to analyze the location of the crimes.

6.2 4.2. Selected column

Therefore, based on the analysis of the subsection 4.1 we think that the best column, that has good properties for the four major criterias, is the Zip Code information. And we will use it to respond the proposed questions. The City also could be use it, but we think that the Zip Code can represent better some areas, because they don't have the political boundaries of the geographic, eg, a zip code can contain more then one city and one city can contain more then one zip code. The zip code also represent, in average, a smaller area, once that we have 50 zip codes in the dataset, against 32 different cities.

Besides that, we think that a analysis of the location (latitude and longitude coordinates), visualizing in a map is also a very good way to look for the crime map, considering that the location column don't have that much of missing values.

6.3 4.3. In what area did the most crimes occur?

Once the column for characterize the area is determined, the answer of this question is very straightforward and the area with most crimes is the one with the zip code equal to 20910.

```
In [30]: zip_code = crimes['zip']
         print("The list of the 3 Zip Codes with most crimes:")
         print(zip_code.value_counts().iloc[:3])

         print("\nThe list of the 3 Zip Codes with less crimes:")
         print(zip_code.value_counts().iloc[-3:])
```

The list of the 3 Zip Codes with most crimes:

20910	2017
-------	------

20904	1779
-------	------

20902	1647
-------	------

Name: zip, dtype: int64

The list of the 3 Zip Codes with less crimes:

20782	1
-------	---

20707	1
-------	---

20723	1
-------	---

Name: zip, dtype: int64

6.4 4.4. What physical locations (like cities) does this area correspond to?

Even the Zip Code isn't very comprehensible, but at least is officially part of an address and can easily be associated with a more comprehensible information with a help of a map. In here, we will use a dataset provided by [United States Zip Codes site](#) which list the primary cities and the 'secondary cities' (actually the column is named as 'acceptable cities') for each zip code.

As we can see in the results of the code below, the zip code with more crimes correpond only to the city of Silver Spring.

```
In [31]: #stores the external dataset that contains the informations about the zip codes
zip_df = pd.read_excel('zip_code_database.xls')
#print the 5 first rows
zip_df.head()
```

```
Out[31]:
```

	zip	type	decommissioned	primary_city	acceptable_cities	\
0	501	UNIQUE	0	Holtsville	NaN	
1	544	UNIQUE	0	Holtsville	NaN	
2	601	STANDARD	0	Adjuntas	NaN	
3	602	STANDARD	0	Aguada	NaN	
4	603	STANDARD	0	Aguadilla	Ramey	

	unacceptable_cities	state	\
0	I R S Service Center	NY	
1	Irs Service Center	NY	
2	Colinas Del Gigante, Jard De Adjuntas, Urb San...	PR	
3	Alts De Aguada, Bo Guaniquilla, Comunidad Las ...	PR	
4	Bda Caban, Bda Esteves, Bo Borinquen, Bo Ceiba...	PR	

	county	timezone	area_codes	world_region	country	\
0	Suffolk County	America/New_York	631	NaN	US	
1	Suffolk County	America/New_York	631	NaN	US	
2	Adjuntas Municipio	America/Puerto_Rico	787939	NaN	US	
3	Aguada Municipio	NaN	787939	NaN	US	
4	Aguadilla Municipio	America/Puerto_Rico	787	NaN	US	

	latitude	longitute	irs_estimated_population_2014
0	40.81	-73.04	562
1	40.81	-73.04	0
2	18.16	-66.72	0
3	18.38	-67.18	0
4	18.43	-67.15	0

```
In [32]: #function that return a list of strings, where each string is one city inside the zip_c
#basically, the function joins the data on the primary_city column and the acceptable_c
def cities_name(row):
    cities = []
    cities.append(row['primary_city'])
    for c in str(row['acceptable_cities']).split(','):
        if c != 'nan':
```

```

        cities.append(c)
    return cities

#stores the cities names in a new column on the DataFrame 'zip_df'
zip_df['Cities_names'] = zip_df.apply(cities_name, axis=1)

print("List of the cities that correspond to the zip code with most crimes:")
print(zip_df[ zip_df['zip'] == zip_code.value_counts().index[0] ]['Cities_names'])

print("\nList (unsorted) of the cities that correspond to the 3 zip codes with most crimes:")
print(zip_df[ zip_df['zip'].isin(zip_code.value_counts().index[:3]) ]['Cities_names'])

print("\nList of the cities that correspond to the zip code with less crimes:")
print(zip_df[ zip_df['zip'] == zip_code.value_counts().index[-1] ]['Cities_names'])

print("\nList (unsorted) of the cities that correspond to the 3 zip codes with less crimes:")
print(zip_df[ zip_df['zip'].isin(zip_code.value_counts().index[-3:]) ]['Cities_names'])

```

List of the cities that correspond to the zip code with most crimes:

8785 [Silver Spring]

Name: Cities_names, dtype: object

List (unsorted) of the cities that correspond to the 3 zip codes with most crimes:

8778 [Silver Spring, Wheaton]

8780 [Silver Spring, Colesville]

8785 [Silver Spring]

Name: Cities_names, dtype: object

List of the cities that correspond to the zip code with less crimes:

8659 [Laurel, Scaggsville]

Name: Cities_names, dtype: object

List (unsorted) of the cities that correspond to the 3 zip codes with less crimes:

8644 [Laurel]

8659 [Laurel, Scaggsville]

8706 [Hyattsville, Chillum, University Pa, Univer...]

Name: Cities_names, dtype: object

6.5 4.5. Which area has the highest number of crimes per capita?

The dataset found for the zip codes also has the information about the estimated population of that zip code in 2014, which is very close to the date of our crime dataset and therefore it will be used for this analysis. Unfortunately, this dataset has some missing values for the population information. More specifically, it has 3 zip codes inside the Montgomery County with no population information.

As we can see in the result of the code below, the area with most crimes per capita is still Silver Spring. We also listed two others areas with most crimes per capita.

```

In [33]: print("Number of zip codes with no information of the population:")
print(len(zip_df[zip_df['irs_estimated_population_2014'] == 0]))
print("\nList of zip codes inside the Montgomery County with no information of the population:")
print( zip_df[ zip_df['zip'].isin(zip_code.unique()) ]
        [zip_df[ zip_df['zip'].isin(zip_code.unique()) ]['irs_estimated_population_2014']
        [['zip', 'irs_estimated_population_2014']])

crimes_per_capita = pd.DataFrame(data=zip_code.value_counts())
crimes_per_capita.reset_index(inplace=True)
crimes_per_capita.columns = ['zip', 'Number_of_Crimes']
crimes_per_capita = crimes_per_capita[ crimes_per_capita['zip'] != 0 ]
crimes_per_capita = crimes_per_capita.sort_values(by='zip')

zip_montgomery = zip_df[ zip_df['zip'].isin(zip_code.unique()) ].sort_values(by='zip')

crimes_per_capita = pd.merge(crimes_per_capita, zip_montgomery, on=['zip'])
crimes_per_capita['crimes_per_capita'] = (crimes_per_capita['Number_of_Crimes'] /
                                         crimes_per_capita['irs_estimated_population_2014'])

crimes_per_capita = crimes_per_capita[ crimes_per_capita['irs_estimated_population_2014'] != 0 ]

print("\nList of the 3 zip codes with most crimes per capita:")
print(crimes_per_capita.sort_values(by='crimes_per_capita', ascending=False)[['zip', 'crimes_per_capita']])

print("\nCities names of those areas:")
print(crimes_per_capita.sort_values(by='crimes_per_capita', ascending=False)[['Cities_names']])

```

Number of zip codes with no information of the population:
4011

List of zip codes inside the Montgomery County with no information of the population:

	zip	irs_estimated_population_2014
8770	20892	0
8771	20894	0
8776	20899	0

List of the 3 zip codes with most crimes per capita:

	zip	crimes_per_capita
46	20910	0.053572
31	20877	0.038256
4	20814	0.036027

Cities names of those areas:

	Cities_names
46	[Silver Spring]

```
31 [Gaithersburg, Montgomery Village, Montgomery ...  
4 [Bethesda]
```

6.6 4.6. Map of the crimes

Although the use of the location as coordinates isn't that useful to specify the place with most crimes, it's still useful and very intuitive to see the crimes occurrences in a map. In that way, we can see regions with most incidence of crimes and make our conclusions. With that in mind, we plot a map of the Montgomery County, where each crime is represented by a red point.

Analyzing the map we can confirm that the Silver Spring region has in fact a lot of crimes incident. But we also note that others regions with a lot of crimes, as Bethesda and Gaithersburg.

```
In [34]: from bokeh.io import output_notebook, show  
        from bokeh.models import (  
            GMapPlot, GMapOptions, ColumnDataSource, Circle, DataRange1d, PanTool, WheelZoomTool,  
        )  
  
        crimes_lat = crimes[~pd.isnull(crimes['Latitude'])]['Latitude']  
        crimes_lon = crimes[~pd.isnull(crimes['Longitude'])]['Longitude']  
  
        output_notebook()  
  
        map_style = '''  
[  
    {  
        "elementType": "geometry",  
        "stylers": [  
            {  
                "color": "#f5f5f5"  
            }  
        ]  
    },  
    {  
        "elementType": "labels.icon",  
        "stylers": [  
            {  
                "visibility": "off"  
            }  
        ]  
    },  
    {  
        "elementType": "labels.text.fill",  
        "stylers": [  
            {  
                "color": "#000000"  
            }  
        ]  
    }  
]
```

```

},
{
  "elementType": "labels.text.stroke",
  "stylers": [
    {
      "color": "#f5f5f5"
    }
  ]
},
{
  "featureType": "administrative.land_parcel",
  "elementType": "labels.text.fill",
  "stylers": [
    {
      "color": "#bdbdbd"
    }
  ]
},
{
  "featureType": "poi",
  "elementType": "geometry",
  "stylers": [
    {
      "color": "#eeeeee"
    }
  ]
},
{
  "featureType": "poi",
  "elementType": "labels.text.fill",
  "stylers": [
    {
      "color": "#757575"
    }
  ]
},
{
  "featureType": "poi.park",
  "elementType": "geometry",
  "stylers": [
    {
      "color": "#e5e5e5"
    }
  ]
},
{
  "featureType": "poi.park",
  "elementType": "labels.text.fill",

```

```

    "stylers": [
      {
        "color": "#9e9e9e"
      }
    ]
  },
  {
    "featureType": "road",
    "elementType": "geometry",
    "stylers": [
      {
        "color": "#ffffff"
      }
    ]
  },
  {
    "featureType": "road.arterial",
    "elementType": "labels.text.fill",
    "stylers": [
      {
        "color": "#757575"
      }
    ]
  },
  {
    "featureType": "road.highway",
    "elementType": "geometry",
    "stylers": [
      {
        "color": "#dadada"
      }
    ]
  },
  {
    "featureType": "road.highway",
    "elementType": "labels.text.fill",
    "stylers": [
      {
        "color": "#616161"
      }
    ]
  },
  {
    "featureType": "road.local",
    "elementType": "labels.text.fill",
    "stylers": [
      {
        "color": "#9e9e9e"

```

```

        }
    ]
},
{
    "featureType": "transit.line",
    "elementType": "geometry",
    "stylers": [
        {
            "color": "#e5e5e5"
        }
    ]
},
{
    "featureType": "transit.station",
    "elementType": "geometry",
    "stylers": [
        {
            "color": "#eeeeee"
        }
    ]
},
{
    "featureType": "water",
    "elementType": "geometry",
    "stylers": [
        {
            "color": "#c9c9c9"
        }
    ]
},
{
    "featureType": "water",
    "elementType": "labels.text.fill",
    "stylers": [
        {
            "color": "#9e9e9e"
        }
    ]
}
]'''

```

```

map_options = GMapOptions(lat=39.15, lng=-77.193058, map_type="roadmap", zoom=10, style=style)

plot = GMapPlot(
    x_range=DataRange1d(), y_range=DataRange1d(), map_options=map_options
)
plot.title.text = "Austin"

```

```

# For GMaps to function, Google requires you obtain and enable an API key:
#
#     https://developers.google.com/maps/documentation/javascript/get-api-key
#
# Replace the value below with your personal API key:
plot.api_key = "AIzaSyC_MGrhFdUkuKWwyfX9AG583Epb4uVkk4c"

source = ColumnDataSource(
    data=dict(
        lat=crimes_lat,
        lon=crimes_lon,
    )
)

circle = Circle(x="lon", y="lat", size=2, fill_color="red", fill_alpha=0.2, line_color=)
plot.add_glyph(source, circle)

plot.add_tools(PanTool(), WheelZoomTool(), BoxSelectTool())
show(plot)

```

7 5. Analyzing types of crime

In the crime dataset used for this analysis we find two columns about the type of the crime: 'Class' and 'Class Description'. The last one contains a brief description of the type of the crime and the first is his identification number. We will use both of them to analyze the types of crimes.

7.1 5.1. Which crimes are the most common? Least common?

As we can see in the results of the code below, the 3 most common crimes are: (1) Driving under the influence (alcohol and drugs), (2) CDS - Possession of marijuana/hashish and (3) Police information.

As for the least common crimes we have many of them with only 1 occurrence, so, we listed the 3 least ones, but we also list all the crimes with only 1 occurrence. And it's nice to see that some extremely violent crimes are in this list, as kidnapping, some types of homicides, some types of suicides and a lot of types of assault.

```

In [35]: print("List of the 3 most common crimes:")
         print(class_description.value_counts().iloc[:3])

         print("\nList of the 3 least common crimes:")
         print(class_description.value_counts().iloc[-3:])

         print('\nList of all types of crimes with only 1 occurrence:')
         print(class_description.unique()[ class_description.value_counts() == 1 ])

```

```

List of the 3 most common crimes:
DRIVING UNDER THE INFLUENCE      1710
CDS-POSS MARIJUANA/HASHISH       1334

```



```
POL INFORMATION                                1191
Name: Class Description, dtype: int64
```

List of the 3 least common crimes:

```
HOMICIDE-OTHER                                1
ANIMAL OFFENSE - HOT CAR                      1
PARKING OFFENSES                             1
```

```
Name: Class Description, dtype: int64
```

List of all types of crimes with only 1 occurrence:

```
['KIDNAPPING' 'CDS-MANU-HALLUC/LSD/PCP/ETC' 'TRAFFIC HAZARD' 'LOITERING'
 'HOME IMPROVEMENT VIOLATION' 'BAD CHECKS-MERCHANDISE/UNDER $300'
 'SUICIDE - CUT/STAB' 'CDS IMPLMENT-HALLUC/LSD/PCP/ET' 'ABANDONED AUTO'
 'VANDALISM GRAFFITI CHURCH/TEMP' 'CDS-USE HALLUC/LSD/PCP/ETC.'
 'VANDALISM POSSESSION GRAFFITI MATERIAL' 'WEAPON POSSESSION OTHER FIREARM'
 'SIMPLE ASSAULT - ELDERLY' 'EMBEZZLE/THEFT-UNDER $300 OTHER'
 'HOMICIDE-FIREARM' 'HOMICIDE-OTHER' 'SUICIDE-ATTEMPT-FIREARM'
 'DISCONDUCT-DISORDERLY HOUSE(NOT SEX)' 'BURG NO FORCE - SCH/TIME UNK'
 'ROB KNIFE/CUT - CONV. STORE' 'CDS RX-FORGERY SYNTH DEMEROL/METHA'
 'ROB STN ARM - GAS/SVC STA' 'AGG ASSLT CUT/STAB P.O.'
 'BURG FORCE-SCH/DAY' 'ESCAPEE' 'AGG ASSLT FIREARM P.O.'
 'AGG ASSLT BEAT/INJ P.O.' 'SANE COLLECTION NON-STRANGER'
 'AGG ASSLT OTHER WPN ON ELDERLY' 'CDS RX FORGERY INHALANT/GLUE/AEROS']
```

7.2 5.2. What's the most common violent crime?

To answer those question, first of all, we need to classify the crimes in violent and nonviolent as this information aren't provided by our dataset. This classification is very subjective, but we use as the main guide the power of the crime to cause harm to others or even himself, physically or psychologically. The classification was made by the matchup of some words that characterize the main type of the crime, eg, if the class description contains 'homicide' it's violent. The default case is nonviolent, eg, if the class description don't fall in any of the violent conditions it's nonviolent.

Bellow we list the words that make a crime to be classified as violent, with a brief explanation:

- **Homicide and suicide:** The first one is for obvious reasons, as this one can be classified as the most violent crime. Although suicide can be classified as nonviolent for some people we should be in mind that the result of both these crimes are the loss of human lives and the society should work to avoid them. So, the classification of suicide as a violent crime has this signification;
- **Rape and sex:** Although the first word is common sense that it should be classified as violent, the 'sex' term can be controversial, as we have some crimes containing the word 'sex' that can't cause that much of a physically harm, as a indecent phone call, but we think that this one can cause a lot of psychologically harm, and therefore we choose to maintain as a violent crime;
- **Assault, asslt and rob:** Those are the most common category of violent crimes. The [dictionary.com](#) defines assault as a "sudden, violent attack". And rob is characterize by use of force or threat of violence;

- **CDS:** We decided to classify every crime related to Controlled Dangerous Substance (CDS) to violent, understanding that they have a significant impact in the crimes as a role, even if one specific crime seems harmless, as the case of possession of marijuana/hashish;
- **Weapon:** The crimes that involved a weapon can be classify as violent, considering the potential of the harm that can be caused;
- **Kidnapping:** It's common sense that this crime should be considered as a violent one;
- **Family offense:** We noticed the presence of this type of crime and it's most associated with child or elder harm and we thought of them as harmful.
- **Driving under the influence:** This crime can also be controversial, but we considered the potential of this crime to cause harm to the car occupants and the others, which can be very significant.

Therefore, the crimes as 'Larceny', 'Embezzle', 'Forgery' and so on are classified as non-violent.

Because of our classification of violent crimes, the 2 most common ones is also the 2 most common crimes in general, namely (a) Driving under the influence of alcohol and (b) Possession of marijuana/hashish. The next most common violent crime is 'assault & battery - citizen'. We also list all the violent crimes with only 1 occurrence.

```
In [36]: violent_words = ['homicide', 'suicide', 'rape', 'sex', 'assault', 'asslt',
                          'rob', 'cds', 'weapon', 'kidnapping', 'family offense',
                          'driving under the influence']

crimes['Violent'] = crimes['Class Description'].str.contains(violent_words[0], case=False)
for w in violent_words:
    crimes['Violent'] = crimes['Violent'] | crimes['Class Description'].str.contains(w, case=False)

print("\nNumber of class descriptions classified as violent: ",
      crimes[~crimes['Class Description'].duplicated()]['Violent'].tolist().count(True))

print("\nList of the 3 most common violent crimes:")
print( crimes[ crimes['Violent'] == True ]['Class Description'].value_counts()[:3] )

print('\nList of all violent crimes with only 1 occurrence:')
print(crimes[ crimes['Violent'] == True ]['Class Description'].unique()
      [ crimes[ crimes['Violent'] == True ]['Class Description'].value_counts() == 1 ])
```

Number of class descriptions classified as violent: 128

List of the 3 most common violent crimes:

DRIVING UNDER THE INFLUENCE 1710

CDS-POSS MARIJUANA/HASHISH 1334

ASSAULT & BATTERY - CITIZEN 382

Name: Class Description, dtype: int64

List of all violent crimes with only 1 occurrence:

['CDS-USE HALLUC/LSO/PCP/ETC.' 'WEAPON POSSESSION OTHER FIREARM'

'SIMPLE ASSAULT - ELDERLY' 'HOMICIDE-FIREARM' 'HOMICIDE-OTHER'

```
'SUICIDE-ATTEMPT-FIREARM' 'DISCONDUCT-DISORDERLY HOUSE(NOT SEX)'
'ROB KNIFE/CUT - CONV. STORE' 'CDS RX-FORGERY SYNTH DEMEROL/METHA'
'ROB STN ARM - GAS/SVC STA' 'AGG ASSLT CUT/STAB P.O.'
'AGG ASSLT FIREARM P.O.' 'AGG ASSLT BEAT/INJ P.O.'
'AGG ASSLT OTHER WPN ON ELDERLY' 'CDS RX FORGERY INHALANT/GLUE/AEROS']
```

7.3 5.3. The most common nonviolent crime?

The most common nonviolent crime is 'Pol information', followed closed by 'Larceny from auto over \$200' and 'Larceny from building over \$200'. We also list all the nonviolent crimes with only 1 occurrence.

```
In [37]: print("List of the 3 most common nonviolent crimes:")
         print( crimes[ crimes['Violent'] == False ]['Class Description'].value_counts()[:3] )

         print('\nList of all nonviolent crimes with only 1 occurrence:')
         print(crimes[ crimes['Violent'] == False ]['Class Description'].unique()
               [ crimes[ crimes['Violent'] == False ]['Class Description'].value_counts() == 1 ] )
```

List of the 3 most common nonviolent crimes:

```
POL INFORMATION          1191
LARCENY FROM AUTO OVER $200    914
LARCENY FROM BUILDING OVER $200  895
Name: Class Description, dtype: int64
```

List of all nonviolent crimes with only 1 occurrence:

```
['ARSON ATTEMPT UNOCCUPIED STRUCTURE/OTH PROP'
'BAD CHECKS-CASH/OTHER UNDER $300' 'ANIMAL OFFENSE - AGG ANIMAL/BITE'
'BAD CHECKS-LABOR/SERVICE UNDER $300' 'TRAFFIC HAZARD' 'LOITERING'
'HOME IMPROVEMENT VIOLATION' 'BAD CHECKS-MERCHANDISE/UNDER $300'
'ABANDONED AUTO' 'VANDALISM GRAFFITI CHURCH/TEMP'
'VANDALISM POSSESSION GRAFFITI MATERIAL' 'EMBEZZLE/THEFT-UNDER $300 OTHER'
'BURG NO FORCE - SCH/TIME UNK' 'BURG FORCE-SCH/DAY' 'ESCAPEE'
'SANE COLLECTION NON-STRANGER']
```

8 6. Combine Analysis

In this section we will analyze the crimes with some combination of criterias.

8.1 6.1. Where are the most violent crimes committed?

As we can see in the results of the code below, the most violent crimes are committed in the zip code 20904, followed by 20874 and 20910. We also name the cities that compose those zip codes.

```
In [38]: print('List of the 3 zip codes with most violent crimes committed:')
         print(crimes[ crimes['Violent'] == True ]['zip'].value_counts().iloc[:3])
```

```

print('\nList of the cities names of those zip codes:')
for i in range(3):
    print(crimes_per_capita[ crimes_per_capita['zip'] ==
                           crimes[ crimes['Violent'] == True ]['zip'].value_counts().index[

```

List of the 3 zip codes with most violent crimes committed:

20904 497

20874 490

20910 479

Name: zip, dtype: int64

List of the cities names of those zip codes:

['Silver Spring', 'Colesville']

['Germantown', 'Darnestown']

['Silver Spring']

8.2 6.2. Where are the most nonviolent crimes committed?

As we can see in the results of the code below, the most nonviolent crimes are committed in the zip code 20910, followed by 20904 and 20902. We also name the cities that compose those zip codes.

It's possible to observe that from the 6 zip codes listed (3 with most violent crimes and 3 with most nonviolent crimes), 5 of them is in the top 3 areas with most crimes.

```

In [39]: print('List of the 3 zip codes with most nonviolent crimes committed:')
         print(crimes[ crimes['Violent'] == False ]['zip'].value_counts()[:3])

         print('\nList of the cities names of those zip codes:')
         for i in range(3):
             print(crimes_per_capita[ crimes_per_capita['zip'] ==
                                     crimes[ crimes['Violent'] == False ]['zip'].value_counts().index[

```

List of the 3 zip codes with most nonviolent crimes committed:

20910 1538

20904 1282

20902 1192

Name: zip, dtype: int64

List of the cities names of those zip codes:

['Silver Spring']

['Silver Spring', 'Colesville']

['Silver Spring', 'Wheaton']

8.3 6.3. When are the most violent crimes committed?

As was analyzed in the section 3, we think it's important to tell at least the day of the week which has more violent crimes and also the period of the day that they are likely to occur.

8.3.1 6.3.1. What day of the week has more violent crimes committed?

The results of the code below show that saturday and friday are the days of week with most violent crimes committed, as they ave basically the same amount of violent crimes. The others days also have similar quantity of violent crimes (893 to 930), with the exception of monday, that has only 696 violent crimes. As posted in previous analysis, this can be due to the number of citizens in the streets, especially at night.

```
In [40]: crimes['Start_d_time'] = crime_datetime
         print("List of the days of the week with most violent crimes committed:")
         print(crimes[ crimes['Violent'] == True ]['Start_d_time'].dt.weekday_name.value_counts())
```

List of the days of the week with most violent crimes committed:

Saturday	1048
Friday	1044
Wednesday	938
Tuesday	933
Sunday	918
Thursday	902
Monday	700

Name: Start_d_time, dtype: int64

8.3.2 6.3.2. What period of the day has more violent crimes committed?

When we filter to only violent crimes, the period of the night highlights from the others periods. It have more then three times the number of violent crimes then the next period, which is the evening, and has almost four times the number of the violent crimes of the afternoon. Even if we took the mean number per hour, the night period has much more violent crimes then the others periods.

If we go back to the section 3.2.2, we will see that when we analyze the all types of crimes (violent and nonviolent), the numbers are much closer, with the afternoon having more crimes when considering the mean number per hour. This means that most of the crimes there are committed in the night are violent and in the other way, most of the crimes that are committed in the morning and afternoon are nonviolent. We can say that the evening period has both types of crimes, as it is in the trasition of the afternoon to the night.

Looking for the number of violent crimes per hour we can observe that the top5 is from 10:00 pm to 02:59 am, and we can continue in this analysis observing that the top7 is from 8:00 pm to 02:59 am, and so on.

```
In [41]: print("List of the periods of day with most violent crimes committed:")
         print(crimes[ crimes['Violent'] == True ]['Start_d_time'].apply(period).value_counts())

periods_normalized = pd.DataFrame( data={'Start Date':
                                         { 'Morning':
                                           crimes[ crimes['Violent'] == True ]['Start_d_time'].
                                           apply(period).value_counts()['Morning'] / (MORNING-NIGHT),
                                           'Afternoon':
                                           crimes[ crimes['Violent'] == True ]['Start_d_time'].
```

```

        apply(period).value_counts()['Afternoon'] / (AFTERNOON-MORNING),
        'Evening':
        crimes[ crimes['Violent'] == True ]['Start_d_time'].
        apply(period).value_counts()['Evening'] / (EVENING - AFTERNOON),
        'Night':
        crimes[ crimes['Violent'] == True ]['Start_d_time'].
        apply(period).value_counts()['Night'] / (NIGHT - (EVENING-24)),

print("\nList of the periods of day with most violent crimes committed, considering the
print(periods_normalized.sort_values(by='Start Date',ascending=False)['Start Date'])

import matplotlib.pyplot as plt

#Plot a histogram with the hours of the day as the x axis and the number of crimes in t
fig = plt.figure(figsize=(12,6))
ax = fig.add_subplot(1,1,1)
ax.hist(crimes[ crimes['Violent'] == True ]['Start_d_time'].dt.hour,
        range=(0, 24), bins=24, align='left', edgecolor='gray', rwidth=0.8)
ax.set_xticks(range(24))
ax.set_xlabel('Hour of the day')
ax.set_ylabel('Number of violent crimes')
ax.set_title('Number of violent crimes for each hour of the day')
plt.show()

```

List of the periods of day with most violent crimes committed:

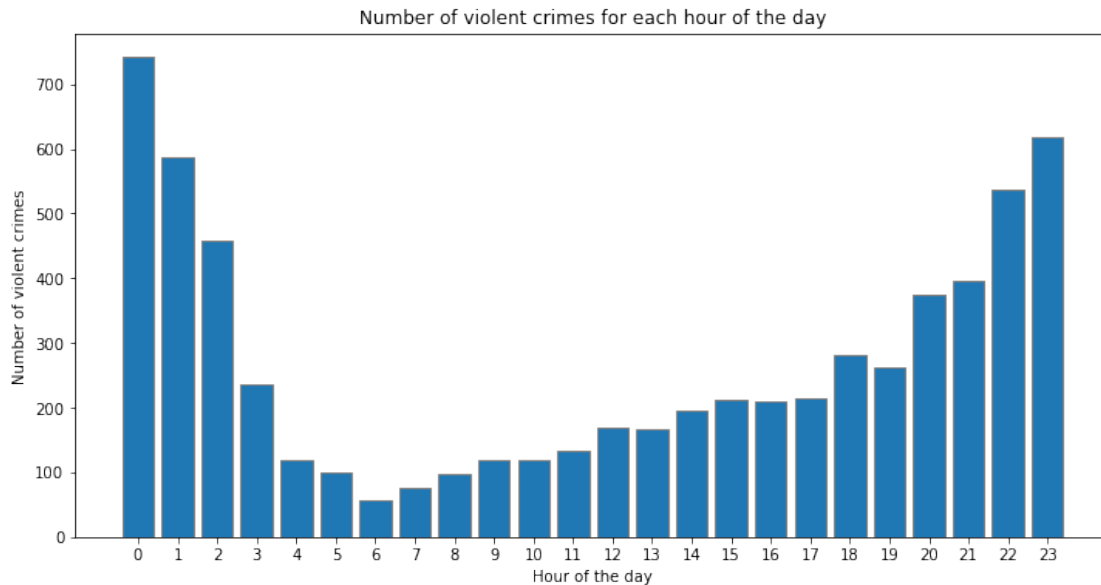
Night	3683
Evening	1148
Afternoon	934
Morning	718

Name: Start_d_time, dtype: int64

List of the periods of day with most violent crimes committed, considering the mean value per ho

Night	460.375000
Evening	287.000000
Afternoon	186.800000
Morning	102.571429

Name: Start Date, dtype: float64



8.4 6.4. When are the most nonviolent crimes committed?

8.4.1 6.4.1. What day of the week has more nonviolent crimes committed?

We can note that the pattern of the numbers of nonviolent and violent crimes isn't that different when analyzing the days of the week. It's possible to state that the difference from the day with most crimes to the day with least crimes is bigger in the violent crimes, and also one or other day has their 'position' changed in the list, but both have Saturday/Friday as the day with more crimes and Sunday as the day with less crimes.

```
In [42]: print("List of the days of the week with most nonviolent crimes committed:")
         print(crimes[ crimes['Violent'] == False ]['Start_d_time'].dt.weekday_name.value_counts)
```

List of the days of the week with most nonviolent crimes committed:

```
Friday      2675
Tuesday     2530
Wednesday   2492
Monday      2457
Thursday    2447
Saturday    2295
Sunday      1990
```

Name: Start_d_time, dtype: int64

8.4.2 6.4.2. What period of the day has more nonviolent crimes committed?

As for the period of the day, we can see two very different behaviours of violent and nonviolent crimes. If for the violent crimes the night period has a lot more crimes, for the nonviolent the

number is very equilibrated, especially when looking the absolute value. We can observe that the afternoon period has the most nonviolent crimes, with 4678, and the evening period has the least nonviolent crimes, with 3945, therefore, we can say that the difference between the two edges is quite little.

If we consider the mean number of crimes per hour, we have the evening and afternoon periods very close to each other (986 and 935, respectively) and then the night and morning periods also very close to each other (592 and 584, respectively) but quite distant from the prior ones.

```
In [43]: print("List of the periods of day with most nonviolent crimes committed:")
         print(crimes[ crimes['Violent'] == False ]['Start_d_time'].apply(period).value_counts())

periods_normalized = pd.DataFrame( data={'Start Date':
    { 'Morning':
      crimes[ crimes['Violent'] == False ]['Start_d_time'].
      apply(period).value_counts()['Morning'] / (MORNING-NIGHT),
      'Afternoon':
      crimes[ crimes['Violent'] == False ]['Start_d_time'].
      apply(period).value_counts()['Afternoon'] / (AFTERNOON-MORNING),
      'Evening':
      crimes[ crimes['Violent'] == False ]['Start_d_time'].
      apply(period).value_counts()['Evening'] / (EVENING - AFTERNOON),
      'Night':
      crimes[ crimes['Violent'] == False ]['Start_d_time'].
      apply(period).value_counts()['Night'] / (NIGHT - (EVENING-24)),

    print("\nList of the periods of day with most nonviolent crimes committed, considering
    print(periods_normalized.sort_values(by='Start Date',ascending=False)['Start Date'])

import matplotlib.pyplot as plt

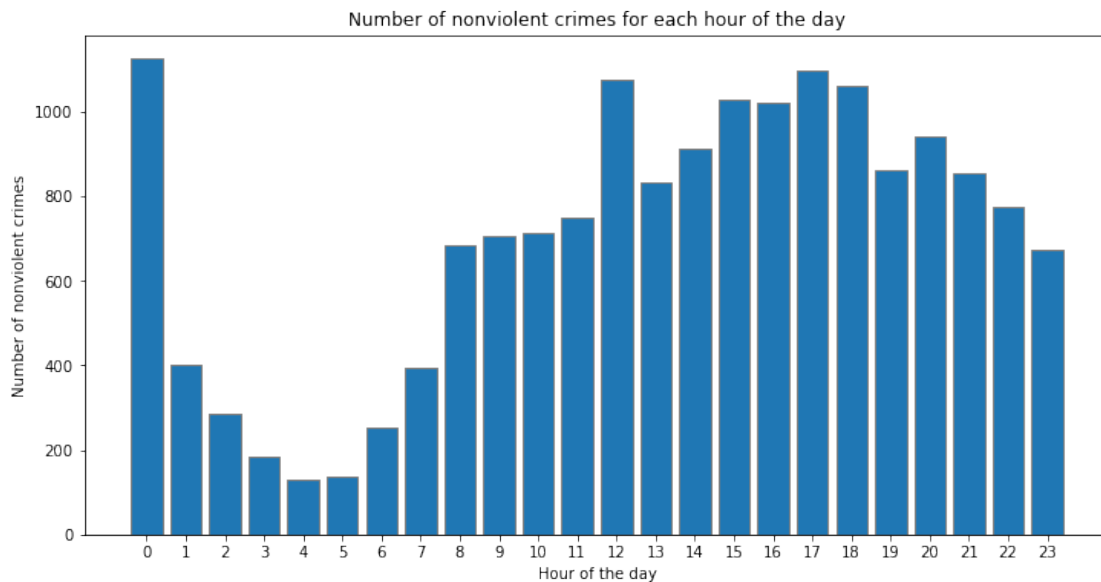
#Plot a histogram with the hours of the day as the x axis and the number of crimes in t
fig = plt.figure(figsize=(12,6))
ax = fig.add_subplot(1,1,1)
ax.hist(crimes[ crimes['Violent'] == False ]['Start_d_time'].dt.hour,
        range=(0, 24), bins=24, align='left', edgecolor='gray', rwidth=0.8)
ax.set_xticks(range(24))
ax.set_xlabel('Hour of the day')
ax.set_ylabel('Number of nonviolent crimes')
ax.set_title('Number of nonviolent crimes for each hour of the day')
plt.show()
```

```
List of the periods of day with most nonviolent crimes committed:
Afternoon    4686
Night        4158
Morning      4093
Evening      3949
Name: Start_d_time, dtype: int64
```


List of the periods of day with most nonviolent crimes committed, considering the mean value per

Evening	987.250000
Afternoon	937.200000
Morning	584.714286
Night	519.750000

Name: Start Date, dtype: float64



9 7. Posing and answering your own questions

This section was dedicated to explore some questions that was not covered by the previous sections, in a not guided way. This means that we are free to set our own questions and analyze the dataset to seek the answer for those. With that in mind, we made some analysis, described in the subsections below.

9.1 7.1. A good 'mini-area' inside a bad area

If a person is forced to pass in an area that we know that happens a lot of crimes (lives or works in one), she probably want to know which 'mini-areas' (like streets) is the safer ones (has less crimes) in order to try to walk by those ones. With that in mind, we took the worst area (based on the selected criteria) and find the streets with less number of crimes and compared with the mean number of crimes per street of that area, just for reference. We made this analyze considering all crimes (violent and nonviolent) and only the violent crimes.

In order to answer those questions, we need first to separate only the street name from the 'Block Address' column, by removing the number of the first position, if has one.

9.1.1 7.1.1. A 'mini-area' with less crimes inside an area with most crimes

In this case, we are considering all of the crimes. The code below show the list of the 10 streets with lesser crimes. We can see that all of them had only 1 crime. Actually, we showed that 63 streets has only 1 crime. To analyze how this is the area with most number of crimes, we also listed the 3 streets with most crimes, which gives us some streets with 321 crimes (Georgia Avenue) to 111 crimes (Ellsworth DR). Bases on that, we conclude that most of the crimes are concentrated in a few streets, therefore, is easier for the citizens to try to avoid them.

```
In [44]: def street_name(row):
        try:
            float(row.split()[0])
            return str.upper(' '.join( i for i in row.split()[1:] ))
        except ValueError:
            return str.upper(row)

crimes['Street Address'] = crimes['Block Address'].apply(street_name)

most_crime_zip = zip_code.value_counts().index[0]

streets_most_crime_zip = crimes[ crimes['zip'] == most_crime_zip ]['Street Address']
mean_crimes_street = len(streets_most_crime_zip) / len(streets_most_crime_zip.unique())

print("\nThe mean number of crimes per street in this area are:", mean_crimes_street)

print("\nThe 10 streets with lesser number of crimes are:")
print(streets_most_crime_zip.value_counts()[-10:])

print('\nNumber of streets with only 1 crime:')
print(len(streets_most_crime_zip.value_counts()[streets_most_crime_zip.value_counts().v

print("\nThe 5 streets with most number of crimes are:")
print(streets_most_crime_zip.value_counts()[:5])
```

The mean number of crimes per street in this area are: 12.450617283950617

The 10 streets with lesser number of crimes are:

MILFORD AVE	1
OL495 X31 TO X33 HWY	1
PIN OAK DR	1
ELLIS ST	1
CHICAGO AVE	1
PORTER RD	1
MC MILLAN AVE	1
KERMIT RD	1
W FALKLAND LN	1
AMENT ST	1

Name: Street Address, dtype: int64

Number of streets with only 1 crime:
63

The 5 streets with most number of crimes are:

GEORGIA AVE	321
COLESVILLE RD	239
FENTON ST	152
EAST WEST HWY	116
ELLSWORTH DR	111

Name: Street Address, dtype: int64

9.1.2 7.1.2. A safe 'mini-area' inside a dangerous area

In this case, we are considering only the violent crimes.

```
In [45]: most_violent_crime_zip = crimes[ crimes['Violent'] == True ]['zip'].value_counts().index
        streets_most_crime_zip = crimes[ crimes['zip'] == most_violent_crime_zip ]['Street Address']
        mean_crimes_street = len(streets_most_crime_zip) / len(streets_most_crime_zip.unique())

        print("\nThe mean number of violent crimes per street in this area are:", mean_crimes_street)

        print("\nThe 10 streets with lesser number of violent crimes are:")
        print(streets_most_crime_zip.value_counts()[-10:])

        print('\nNumber of streets with only 1 violent crime:')
        print(len(streets_most_crime_zip.value_counts()[streets_most_crime_zip.value_counts().value_counts() == 1]))

        print("\nThe 5 streets with most number of violent crimes are:")
        print(streets_most_crime_zip.value_counts()[:5])
```

The mean number of violent crimes per street in this area are: 6.895348837209302

The 10 streets with lesser number of violent crimes are:

HERSHFIELD CT	1
PAULA LYNN DR	1
PEBBLE BEACH DR	1
SPRING CLUB PL	1
MONTCLAIR DR	1
WENDOVER RD	1
BRUTON PARISH WAY	1
CALVERTON GALWAY LP ENT	1
BRANDON GREEN DR	1
ROYAL FOREST LN	1

Name: Street Address, dtype: int64

Number of streets with only 1 violent crime:
113

The 5 streets with most number of violent crimes are:

NEW HAMPSHIRE AVE	214
CHERRY HILL RD	187
CASTLE BLVD	159
LOCKWOOD DR	102
OLD COLUMBIA PIKE	78

Name: Street Address, dtype: int64

9.2 7.2. Types of places with more crimes

The population probably want to know what kind of place they are more susceptible to be victim of a crime. So, with that information, people can try to avoid been a victim.

To fullfil this need, we list the types of places (the description of it) with more crimes in general and also classified as violent or nonviolent.

```
In [46]: place = crimes['Place']
        place_violent = crimes[ crimes['Violent'] == True ]['Place']
        place_nonviolent = crimes[ crimes['Violent'] == False ]['Place']

        print("\nTypes os places with most crimes, in general:")
        print(place.value_counts().iloc[:5])

        print("\nTypes os places with most violent crimes:")
        print(place_violent.value_counts().iloc[:5])

        print("\nTypes os places with most noviolent crimes:")
        print(place_nonviolent.value_counts().iloc[:5])
```

Types os places with most crimes, in general:

Residence - Single Family	2729
Street - In vehicle	2581
Residence - Apartment/Condo	1847
Street - Residential	1681
Other/Unknown	1404

Name: Place, dtype: int64

Types os places with most violent crimes:

Street - In vehicle	2197
Street - Residential	579
Residence - Apartment/Condo	529
Residence - Single Family	513
Street - Commercial	414

Name: Place, dtype: int64

Types of places with most nonviolent crimes:

Residence - Single Family	2216
Other/Unknown	1361
Residence - Apartment/Condo	1318
Street - Residential	1102
Parking Lot - Residential	893

Name: Place, dtype: int64

9.3 7.3. Relationship between marijuana and other crimes

Is common to hear that the marijuana is the first step in the crime world or, at least, for others drugs. To instigate that discussion, we analyze if the cities with more crimes involving marijuana also has more crimes that do not involves marijuana.

The code below shows that all of the 6 cities with most crimes involving marijuana is the same 6 cities with most crimes that don't involves this drug, although they are not in the same order. We also show the number of crimes for each of this cities in both conditions. Therefore, we can say that is a correlation between them, but is hard to tell which one comes first, eg, if the cities has more crimes with marijuana because they already have more crimes or if it has more crimes because of the marijuana.

```
In [47]: crimes_marijuana = crimes[crimes['Class Description'].str.contains('MARIJUANA')]
        crimes_not_marijuana = crimes[~crimes['Class Description'].str.contains('MARIJUANA')]

#store the 6 cities with most marijuana crimes and sort this list
cities_more_crimes_marijuana = crimes_marijuana['City'].value_counts().index[:6].sort_v

#store the 6 cities with most non-marijuana crimes and sort this list
cities_more_crimes_not_marijuana = crimes_not_marijuana['City'].value_counts().index[:6].sort_v

#once both lists is sorted, we can compare if these list are equal to each other
#the '==' comparison will return a list of booleans, thus, we use the function all()
#that only return True if all elements are True, eg, all elements are equal
print('\nThe 6 cities with the most marijuana crimes is also the 6 cities with the most non-marijuana crimes')
print( (cities_more_crimes_marijuana == cities_more_crimes_not_marijuana).all() )

print('\nCrimes involving Marijuana: ')
print(crimes_marijuana.groupby('City').size().sort_values()[-6:])
print('\nCrimes not involving Marijuana: ')
print(crimes_not_marijuana.groupby('City').size().sort_values()[-6:])
```

The 6 cities with the most marijuana crimes is also the 6 cities with the most non-marijuana crimes. True

Crimes involving Marijuana:

City	
BETHESDA	48
MONTGOMERY VILLAGE	83
ROCKVILLE	188
GERMANTOWN	260
GAITHERSBURG	320
SILVER SPRING	699

dtype: int64

Crimes not involving Marijuana:

City	
MONTGOMERY VILLAGE	604
BETHESDA	1688
GERMANTOWN	1910
GAITHERSBURG	3083
ROCKVILLE	3265
SILVER SPRING	7927

dtype: int64

9.4 7.4. Comparing the number of homicides between Montgomery County and Rio Grande do Norte

The citizens of Rio Grande do Norte, Brazil, are witnessing the complete chaos in the public security since march of 2015, where prisoners took the control of one of the biggest prisons in the state, located in Alcaçuz. From them, the crimes rates are just increasing as the days pass by. In that context, we thought of making a comparison in the number of homicides between the Rio Grande do Norte state and the Montgomery County, which is the crime that best characterize the violence of a place.

The first difficulty of this comparison is to find the number of homicides in Brazil. The [Atlas da Violência 2017](#), published by the IPEA, has the information of the number of homicides per 100.000 of population for each year of the period between 2005 to 2015, per state. For Rio Grande do Norte, in 2013, the rate of homicides per 100.000 of population was 42,9.

It's important to note that the dataset of the Montgomery County only has the crimes of 6 months of the year of 2013 (july to december). Therefore, for better comparison, we divided the number given in the 2013 Violence Map by 2, once that those information is for the whole year.

We also wanted to analyze the current year. So, in order to make this comparison, we took the crimes data for the 2017 year for the Montgomery County. For the Rio Grande do Norte, we consider the information stated by [OBVIO](#) which says that in 2017 occurred 1.647 homicides in the RN state. We also used the [IBGE](#) information about the population of the RN state, that was estimated in 3.474.998 for 2016.

With those data we did the same comparison, and for our surprise, the relative numbers for 2013 and 2017 was equivalent, but in both years, the RN state has 54 times more homicides per 100k of population then the Montgomery County.

```
In [48]: crimes2016_2017 = pd.read_csv('MontgomeryCountyCrime20162017.csv')
        crimes2016_2017['Start_d_time'] = pd.to_datetime(crimes2016_2017['Start Date/Time'])
```

```
/home/pattousai/anaconda3/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2717: Dty
interactivity=interactivity, compiler=compiler, result=result)
```

```
In [49]: #Read the dataset that contains the estimation of population of all couties of the Mary
#and stores the population of 2013 of Montgomery County
population_maryland = pd.read_csv('Population_Estimates_2000-2016.csv')
population_montgomery_2013 = population_maryland[ population_maryland['Year'] == 2013 ]

#stores all crimes that cointains the 'homicide' word in his description
homicide_montgomery = crimes[crimes['Class Description'].str.contains('HOMICIDE')]
#normalize the population by 100.000, to find the number of homicides per 100k
population_montgomery_norm = population_montgomery_2013/100000
homicide_montgomery_100k = len(homicide_montgomery)/population_montgomery_norm
print('Number of homicides per 100K in Montgomery County in 2013 (july to december):',
      float(homicide_montgomery_100k))

#value informed in the Atlas da Violencia - IPEA
homicide_RN_100k_2013 = 42.9
print('Homicide number per 100K in RN (2013 - Atlas da Violencia - IPEA):', homicide_RN_100k_2013)
print('The Rio Grande do Norte state had', (homicide_RN_100k_2013/2)/float(homicide_mon
'times more homicides than the Montgomery County in 2013.')
```

```
crimes2017 = crimes2016_2017[ crimes2016_2017['Start_d_time'].dt.year == 2017 ]
homicides_2017 = crimes2017['CRIMENAME3'].str.contains('HOMICIDE').tolist().count(True)
population_montgomery_2016 = population_maryland[ population_maryland['Year'] == 2016 ]
population_montgomery_2016_norm = population_montgomery_2016/100000
homicide_montgomery_100k_2017 = homicides_2017 / population_montgomery_2016_norm
print('\nNumber of homicides per 100K in Montgomery County in 2017 (until september 11)
      float(homicide_montgomery_100k_2017))

#number informed by OBVIO
homicide_RN_2017 = 1647
#by IBGE
population_RN_2016 = 3474998
population_RN_2016_norm = population_RN_2016/100000
homicide_RN_100k_2017 = homicide_RN_2017 / population_RN_2016_norm
print('Homicide number per 100K in RN in 2017, according to OBVIO:', homicide_RN_100k_2017)
print('The Rio Grande do Norte state had', homicide_RN_100k_2017/float(homicide_montgom
'times more homicides than the Montgomery County in 2017.')
```

Number of homicides per 100K in Montgomery County in 2013 (july to december): 0.3930203515763555
Homicide number per 100K in RN (2013 - Atlas da Violencia - IPEA): 42.9
The Rio Grande do Norte state had 54.577326375 times more homicides than the Montgomery County in 2013.

Number of homicides per 100K in Montgomery County in 2017 (until september 11): 0.86218210627256
Homicide number per 100K in RN in 2017, according to OBVIO: 47.3957107313443
The Rio Grande do Norte state had 54.971809767948066 times more homicides than the Montgomery County in 2017.