Part 1 Report

Title:
NYPD Crimes Analyze

Authors:
Wenjie Sun (ws854)
Xinyan Yang (xy975)
Yaohan Ke (yk1587)

Google Docs:
https://docs.google.com/a/nyu.edu/document/d/1Ucm_P7rkLDR4e1tl-qNADEEZ95wo1cWLxQVGxNa_RwQ/edit?usp=sharing

Github:
https://github.com/sherylke/ds1004project
All codes and analyses are stored on Github. Please follow the instruction in readme to run the code. Also, github also has marked the respective owner.

Abstract of Part I Report:

Goal of this phase is to investigate if data has any outliers, missing value, or any other values that need to be specially taken care of.

Here is a highlight of some quality issues:

a. Compliant from /to time: the original data set used “24:00:00” for 12 am midnight. However, this is not accepted by Python datetime package. We have summarized number of occurrence in the data quality part in the report. And for the trend analysis purpose, we have fixed the issue by changing it to “23:59:59”.
b. Park name / HA Development: These values are publicly available online. However, after implementing the filter code, we noticed that some “invalid” data was due to a different name convention. For data quality purpose, we have now categorized it is an invalid data. However, we have a code to generate all these potential misclassified invalid name and will be manually reviewed and corrected in the part 2.
c. Offense classification code, offense description: There are a few problems for these 2 values. First of all, some code has multiple descriptions (see examples in the report). Partially was due to misspelling, which will be corrected in the part 2 of this project by implementing nearest clustering. However, some of them are very different. We will continue looking for a second database to see if we can purify this column. Another issue we uncovered is some descriptions are missing.

We will try to fill the description as long as there is one-to-one relationship between code and the description.

d. Internal classification code and description: These two columns are more reliable compared to offense classification code and description. The only data issue is missing data. We are going to investigate if there is publicly available database to help us improve the quality. For data analysis purpose, we will majorly rely on this information for the crime type.

Introduction:

Year over year, total number of crimes in new york city went down. We are motivated to investigate what was the reason contributed to the decrease? Was there a specific crime code that contributed to the decrease or was there a specific area that contributed to the decrease? We have two goals here: 1) Identify the reason that contributed to the decrease, which can help govemment decrease the human and other resources accordingly 2) we want to highlight areas and time that have the highest crime rate, so that government can increase police and resources in these times/areas.

Data Quality and Summary 0-5

April 17, 2017

```
In [1]: import matplotlib.pyplot as plt
        import pandas as pd
        import numpy as np

        %matplotlib inline
```

Authors:
Wenjie Sun (ws854)
Xinyan Yang (xy975)
Yaohan Ke (yk1587)

Google Docs:
https://docs.google.com/a/nyu.edu/document/d/1Ucm_P7rkLDR4e1tl-qNADEEZ95wo1cWLxQVGxNa_RwQ/edit?usp=sharing

Github:
https://github.com/sherylke/ds1004project

0.1 Data Quality

0.1.1 Column 0 - Compliant Number

Code: ***column0_data_quality.py*** This code is used to return the data line by line. The code checks if the compliant number is an integer and if it is a unique value in the whole dataset.

```
In [2]: pd.read_table('column0_data_quality.out', header = -1).head()
```

```
Out[2]:            0        1                          2      3
        0  713796694  INTEGER  A unique compliant number  VALID
        1  905281044  INTEGER  A unique compliant number  VALID
        2  123688898  INTEGER  A unique compliant number  VALID
        3  540746868  INTEGER  A unique compliant number  VALID
        4  450573265  INTEGER  A unique compliant number  VALID
```

Code: ***column0_data_summary.py*** This code is used to return a summary of all data.

```
In [3]: pd.read_table('column0_data_summary.out')
```

```
Out[3]: Empty DataFrame
        Columns: [VALID, 5101231]
        Index: []
```

However, code:**column0_data_quality.py** and **column0_data_summary.py** require a large physical memory. The reason is in the code, I ran count of each value, which then be leftjoined by the value self. I am using this code to make sure each of value is unique (count = 1). However, it seems that these 2 codes don't always work, depending on the hadoop capacity.
Since I have already checked that this column has all unique value (which is qualified for being a primiary key), I uploaded a v2 code that doesn't check uniqueness of each value. Instead, v2 only checks if a value is integer.

```
In [4]: pd.read_table('column0_data_quality_v2.out', header = -1).head()
```

```
Out[4]:            0        1                          2      3
        0  101109527  INTEGER  A unique compliant number  VALID
        1  153401121  INTEGER  A unique compliant number  VALID
        2  569369778  INTEGER  A unique compliant number  VALID
        3  968417082  INTEGER  A unique compliant number  VALID
        4  641637920  INTEGER  A unique compliant number  VALID
```

```
In [5]: pd.read_table('column0_data_summary_v2.out')
```

```
Out[5]: Empty DataFrame
        Columns: [VALID, 5101231]
        Index: []
```

0.1.2 Column 1 - Compliant From Date

Code: ***column1_data_quality.py*** This code is used to return the data line by line. The code checks if the date is legal and if the cmplnt_from date time is smaller than cmplnt_to date time.

```
In [6]: pd.read_table('column1_data_quality.out', header = -1).head()
```

```
Out[6]:             0     1                    2      3
        0  12/31/2015  DATE  Compliant from date  VALID
        1  12/31/2015  DATE  Compliant from date  VALID
        2  12/31/2015  DATE  Compliant from date  VALID
        3  12/31/2015  DATE  Compliant from date  VALID
        4  12/31/2015  DATE  Compliant from date  VALID
```

Code: ***column1_data_summary.py*** This code is used to return a summary of all data. Most data has a valid date, a small percent is missing some date, and 4 invalid date

```
In [7]: pd.read_table('column1_data_summary.out',header=-1)
```

```
Out[7]:          0        1
        0  INVALID        4
        1    VALID  5100572
        2      NaN      655
```

Code: ***column1_invalid_data.py*** This code is used to return all 4 lines that are invalid because to_datetime is earlier than from_datetime.

```
In [8]: pd.read_table('column1_invalid_data.out',header=-1)

Out[8]:                                                    0        1
        0  ['06/22/2014', '23:00:00', '06/22/2014', '00:0...  INVALID
        1  ['11/11/2010', '07:00:00', '11/10/2010', '18:0...  INVALID
        2  ['11/15/2009', '10:00:00', '11/14/2009', '12:0...  INVALID
        3  ['05/19/2006', '16:00:00', '05/14/2006', '14:0...  INVALID
```

0.1.3 Column 2 - Compliant From Time

Code: ***column2_data_quality.py*** This code is used to return the data line by line. The code checks if the time is legal and if the cmplnt_from date time is smaller than cmplnt_to date time.

```
In [9]: pd.read_table('column2_data_quality.out', header = -1).head()

Out[9]:          0     1                    2      3
        0  23:45:00  TIME  Compliant from time  VALID
        1  23:36:00  TIME  Compliant from time  VALID
        2  23:30:00  TIME  Compliant from time  VALID
        3  23:30:00  TIME  Compliant from time  VALID
        4  23:25:00  TIME  Compliant from time  VALID
```

Code: ***column2_data_summary.py*** This code is used to return a summary of all data. There are many invalid data in this column.

```
In [10]: pd.read_table('column2_data_summary.out',header=-1)

Out[10]:          0        1
         0  INVALID      907
         1    VALID  5100276
         2      NaN       48
```

Code: ***column2_invalid_data.py*** Most of the invalid data is due to a time format ("24:00:00") is not accepted in Python. This will be corrected by "23:59:59" in the future data analysis.

```
In [11]: pd.read_table('column2_invalid_data.out',header=-1).head()

Out[11]:                                                    0        1
         0  ['06/22/2014', '23:00:00', '06/22/2014', '00:0...  INVALID
         1  ['11/11/2010', '07:00:00', '11/10/2010', '18:0...  INVALID
         2  ['11/15/2009', '10:00:00', '11/14/2009', '12:0...  INVALID
         3  ['08/31/2009', '24:00:00', '09/01/2009', '08:4...  INVALID
         4                 ['08/31/2009', '24:00:00', '', '']  INVALID
```

0.1.4 Column 3 - Compliant To Date

Code: ***column3_data_quality.py*** This code is used to return the data line by line. The code checks if the date is legal and if the cmplnt_from date time is smaller than cmplnt_to date time.

```
In [12]: pd.read_table('column3_data_quality.out', header = -1).head()
```

```
Out[12]:             0     1                   2      3
         0         NaN  DATE  Compliant to date    NaN
         1         NaN  DATE  Compliant to date    NaN
         2         NaN  DATE  Compliant to date    NaN
         3         NaN  DATE  Compliant to date    NaN
         4  12/31/2015  DATE  Compliant to date  VALID
```

Code: ***column3_data_summary.py*** This code is used to return a summary of all data. Since to_date is not required, there are many NULL data

```
In [13]: pd.read_table('column3_data_summary.out',header=-1)
```

```
Out[13]:          0        1
         0  INVALID        4
         1    VALID  3709749
         2      NaN  1391478
```

Code: ***column3_invalid_data.py*** These 4 invalid data is becasue of from datetime is later than to datetime

```
In [14]: pd.read_table('column3_invalid_data.out',header=-1)
```

```
Out[14]:                                                    0        1
         0  ['06/22/2014', '23:00:00', '06/22/2014', '00:0...  INVALID
         1  ['11/11/2010', '07:00:00', '11/10/2010', '18:0...  INVALID
         2  ['11/15/2009', '10:00:00', '11/14/2009', '12:0...  INVALID
         3  ['05/19/2006', '16:00:00', '05/14/2006', '14:0...  INVALID
```

0.1.5 Column 4 - Compliant To Time

Code: ***column4_data_quality.py*** This code is used to return the data line by line. The code checks if the time is legal and if the cmplnt_from date time is smaller than cmplnt_to date time.

```
In [15]: pd.read_table('column4_data_quality.out', header = -1).head()
```

```
Out[15]:           0     1                   2      3
         0       NaN  TIME  Compliant to time    NaN
         1       NaN  TIME  Compliant to time    NaN
         2       NaN  TIME  Compliant to time    NaN
         3       NaN  TIME  Compliant to time    NaN
         4  23:30:00  TIME  Compliant to time  VALID
```

Code: ***column4_data_summary.py*** This code is used to return a summary of all data. Since to_date is not required, there are many NULL data. Invalid data is still because of "24:00:00".

```
In [16]: pd.read_table('column4_data_summary.out',header=-1)
```

```
Out[16]:          0        1
         0  INVALID     1380
         1    VALID  3712066
         2      NaN  1387785
```

Code: ***column4_invalid_data.py*** These 4 invalid data is becasue of from datetime is later than to datetime

```
In [17]: pd.read_table('column4_invalid_data.out',header=-1).head()
```

```
Out[17]:                                                    0        1
         0  ['06/22/2014', '23:00:00', '06/22/2014', '00:0...  INVALID
         1  ['11/11/2010', '07:00:00', '11/10/2010', '18:0...  INVALID
         2  ['11/15/2009', '10:00:00', '11/14/2009', '12:0...  INVALID
         3  ['08/31/2009', '23:55:00', '08/31/2009', '24:0...  INVALID
         4  ['08/30/2009', '06:30:00', '08/31/2009', '24:0...  INVALID
```

0.1.6 Column 5 - Report Date

Code: ***column5_data_quality.py*** This code is used to return the data line by line. The code checks if the date is legal and also is between 2006 and 2015 (since the report is noted that the time range is based on the report time)

```
In [18]: pd.read_table('column5_data_quality.out', header = -1).head()
```

```
Out[18]:             0     1            2      3
         0  12/31/2015  Date  Report Date  VALID
         1  12/31/2015  Date  Report Date  VALID
         2  12/31/2015  Date  Report Date  VALID
         3  12/31/2015  Date  Report Date  VALID
         4  12/31/2015  Date  Report Date  VALID
```

Code: ***column5_data_summary.py*** This code is used to return a summary of all data. All the data falls in the valid time range.

```
In [19]: pd.read_table('column5_data_summary.out',header=-1)
```

```
Out[19]:        0        1
         0  VALID  5101231
```

Looking at the stacked the line chart, the decreasing trend is not very clear. After plotting the trend borough by borough, it is more clear that broklyn contributes the most to the decrease, followed by manhattan.

Data Quality and Summary 6-18

April 17, 2017

0.0.1 Column 6 - Offense Classification Code

Code: ***column6_data_quality.py*** This code is used to return the data line by line. The code checks if the offense classification codes are three digit integers.

```
In [2]: pd.read_table('column6_data_quality.out', header = -1).head()
```

Out[2]:

	0	1	2	3
0	113	INT	offense classification code	VALID
1	101	INT	offense classification code	VALID
2	117	INT	offense classification code	VALID
3	344	INT	offense classification code	VALID
4	344	INT	offense classification code	VALID

Code: ***column6_data_summary.py*** This code is used to return a summary of all data.

```
In [3]: pd.read_table('column6_data_summary.out', header = -1)
```

Out[3]:

	0	1
0	VALID	5101231

0.0.2 Column 7 - Offense Description

Code: ***column7_data_quality.py*** This code is used to return the data line by line. The code checks if the offense description is a valid text object.

```
In [4]: pd.read_table('column7_data_quality.out', header = -1).head()
```

Out[4]:

	0	1	2	3
0	FORGERY	TEXT	offense description	VALID
1	MURDER & NON-NEGL. MANSLAUGHTER	TEXT	offense description	VALID
2	DANGEROUS DRUGS	TEXT	offense description	VALID
3	ASSAULT 3 & RELATED OFFENSES	TEXT	offense description	VALID
4	ASSAULT 3 & RELATED OFFENSES	TEXT	offense description	VALID

Code: ***column6_data_summary.py*** This code is used to return a summary of all data.

```
In [5]: pd.read_table('column7_data_summary.out', header = -1)
```

Out[5]:

	0	1
0	VALID	5082391
1	NaN	18840

0.0.3 Column 8 - Internal Classification Code

Code: ***column8_data_quality.py*** This code is used to return the data line by line. The code checks if the internal classification code is a three digit integer.

```
In [6]: pd.read_table('column8_data_quality.out', header = -1).head()
```

```
Out[6]:        0    1                              2      3
        0  729.0  INT  internal classification code  VALID
        1    NaN  INT  internal classification code    NaN
        2  503.0  INT  internal classification code  VALID
        3  101.0  INT  internal classification code  VALID
        4  101.0  INT  internal classification code  VALID
```

Code: ***column8_data_summary.py*** This code is used to return a summary of all data.

```
In [7]: pd.read_table('column8_data_summary.out', header = -1)
```

```
Out[7]:        0        1
        0  VALID  5096657
        1    NaN     4574
```

0.0.4 Column 9 - Internal Classification Description

Code: ***column9_data_quality.py*** This code is used to return the data line by line. The code checks if the internal classification description is a valid text object.

```
In [8]: pd.read_table('column9_data_quality.out', header = -1).head()
```

```
Out[8]:                                 0     1                                     2  \
        0  FORGERY,ETC.,UNCLASSIFIED-FELO  TEXT  internal classification description
        1                             NaN  TEXT  internal classification description
        2  CONTROLLED SUBSTANCE,INTENT TO  TEXT  internal classification description
        3                      ASSAULT 3  TEXT  internal classification description
        4                      ASSAULT 3  TEXT  internal classification description

               3
        0  VALID
        1    NaN
        2  VALID
        3  VALID
        4  VALID
```

Code: ***column9_data_summary.py*** This code is used to return a summary of all data.

```
In [9]: pd.read_table('column9_data_summary.out', header = -1)
```

```
Out[9]:        0        1
        0  VALID  5096657
        1    NaN     4574
```

0.0.5 Column 10 - Crime Completeness

Code: ***column10_data_quality.py*** This code is used to return the data line by line. The code checks if the there is only two indicators of 'completed' and 'attempted' representing crime completeness and whether there are mising data in this column.

```
In [10]: pd.read_table('column10_data_quality.out', header = -1).head()
```

```
Out[10]:            0     1                    2      3
         0  COMPLETED  TEXT  crime completeness  VALID
         1  COMPLETED  TEXT  crime completeness  VALID
         2  COMPLETED  TEXT  crime completeness  VALID
         3  COMPLETED  TEXT  crime completeness  VALID
         4  COMPLETED  TEXT  crime completeness  VALID
```

Code: ***column10_data_summary.py*** This code is used to return a summary of all data.

```
In [11]: pd.read_table('column10_data_summary.out', header = -1)
```

```
Out[11]:        0        1
         0  VALID  5101224
         1    NaN        7
```

0.0.6 Column 11 - Offense Level

Code: ***column11_data_quality.py*** This code is used to return the data line by line. The code checks if the level of offense is only in three categories of 'felony', 'misdemeanor' and 'violation'.

```
In [12]: pd.read_table('column11_data_quality.out', header = -1).head()
```

```
Out[12]:              0     1              2      3
         0       FELONY  TEXT  offense level  VALID
         1       FELONY  TEXT  offense level  VALID
         2       FELONY  TEXT  offense level  VALID
         3  MISDEMEANOR  TEXT  offense level  VALID
         4  MISDEMEANOR  TEXT  offense level  VALID
```

Code: ***column11_data_summary.py*** This code is used to return a summary of all data.

```
In [13]: pd.read_table('column11_data_summary.out', header = -1)
```

```
Out[13]:        0        1
         0  VALID  5101231
```

0.0.7 Column 12 - Jurisdiction Discription

Code: ***column12_data_quality.py*** This code is used to return the data line by line. The code checks if the jurisdiction discription is a valid text object.

```
In [14]: pd.read_table('column12_data_quality.out', header = -1).head()
```

```
Out[14]:                    0     1                          2      3
         0  N.Y. POLICE DEPT  TEXT  jurisdiction description  VALID
         1  N.Y. POLICE DEPT  TEXT  jurisdiction description  VALID
         2  N.Y. POLICE DEPT  TEXT  jurisdiction description  VALID
         3  N.Y. POLICE DEPT  TEXT  jurisdiction description  VALID
         4  N.Y. POLICE DEPT  TEXT  jurisdiction description  VALID
```

Code: ***column12_data_summary.py*** This code is used to return a summary of all data.

```
In [15]: pd.read_table('column12_data_summary.out', header = -1)

Out[15]:        0        1
         0  VALID  5101231
```

0.0.8 Column 13 - Borough Name

Code: ***column13_data_quality.py*** This code is used to return the data line by line. The code checks if the borough in which the incident occured is one of the five borough names as 'Bronx', 'Manhattan', 'Brooklyn', 'Queens' and 'Staten island'.

```
In [16]: pd.read_table('column13_data_quality.out', header = -1).head()

Out[16]:            0     1             2      3
         0      BRONX  TEXT  borough name  VALID
         1     QUEENS  TEXT  borough name  VALID
         2  MANHATTAN  TEXT  borough name  VALID
         3     QUEENS  TEXT  borough name  VALID
         4  MANHATTAN  TEXT  borough name  VALID
```

Code: ***column13_data_summary.py*** This code is used to return a summary of all data.

```
In [17]: pd.read_table('column13_data_summary.out', header = -1)

Out[17]:        0        1
         0  VALID  5100768
         1    NaN      463
```

0.0.9 Column 14 - Precinct Code

Code: ***column14_data_quality.py*** This code is used to return the data line by line. The code checks if the precinct code is an integer and is valid compared to the full list of precinct codes of NYPD (http://www.nyc.gov/html/nypd/html/home/precincts.shtml).

```
In [18]: pd.read_table('column14_data_quality.out', header = -1).head()

Out[18]:        0    1              2      3
         0   44.0  INT  precinct code  VALID
         1  103.0  INT  precinct code  VALID
         2   28.0  INT  precinct code  VALID
         3  105.0  INT  precinct code  VALID
         4   13.0  INT  precinct code  VALID
```

Code: ***column14_data_summary.py*** This code is used to return a summary of all data.

```
In [19]: pd.read_table('column14_data_summary.out', header = -1)
```

```
Out[19]:        0        1
         0  VALID  5100841
         1    NaN      390
```

0.0.10 Column 15 - Premises Description

Code: ***column15_data_quality.py*** This code is used to return the data line by line. The code checks if the premises description is a valid text object.

```
In [20]: pd.read_table('column15_data_quality.out', header = -1).head()
```

```
Out[20]:           0     1                    2      3
         0    INSIDE  TEXT  occurrence location  VALID
         1   OUTSIDE  TEXT  occurrence location  VALID
         2       NaN  TEXT  occurrence location    NaN
         3    INSIDE  TEXT  occurrence location  VALID
         4  FRONT OF  TEXT  occurrence location  VALID
```

Code: ***column15_data_summary.py*** This code is used to return a summary of all data.

```
In [21]: pd.read_table('column15_data_summary.out', header = -1)
```

```
Out[21]:        0        1
         0  VALID  3973890
         1    NaN  1127341
```

0.0.11 Column 16 - Occurrence Location Description

Code: ***column16_data_quality.py*** This code is used to return the data line by line. The code checks if the location description of occurrence is a valid text object as one of the location in 'inside', 'outside', 'opposite of', 'front of', 'rear of'.

```
In [22]: pd.read_table('column16_data_quality.out', header = -1).head()
```

```
Out[22]:                 0     1                    2      3
         0   BAR/NIGHT CLUB  TEXT  premises description  VALID
         1              NaN  TEXT  premises description    NaN
         2            OTHER  TEXT  premises description  VALID
         3  RESIDENCE-HOUSE  TEXT  premises description  VALID
         4            OTHER  TEXT  premises description  VALID
```

Code: ***column16_data_summary.py*** This code is used to return a summary of all data.

```
In [23]: pd.read_table('column16_data_summary.out', header = -1)
```

```
Out[23]:        0        1
         0  VALID  5067952
         1    NaN    33279
```

0.0.12 Column 17 - Park Names

Code: ***column17_data_quality.py*** This code is used to return the data line by line. The code checks if the park name is a valid name of an NYC public park, playground or greenspace (https://www.nycgovparks.org/park-features/parks-list?boro=X).

```
In [24]: pd.read_table('column17_data_quality.out', header = -1).head()

//anaconda/lib/python3.5/site-packages/IPython/core/interactiveshell.py:2717: DtypeWarning: Colu
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[24]:

	0	1	2	3
0	NaN	TEXT	park names	NaN
1	NaN	TEXT	park names	NaN
2	NaN	TEXT	park names	NaN
3	NaN	TEXT	park names	NaN
4	NaN	TEXT	park names	NaN

Code: ***column17_data_summary.py*** This code is used to return a summary of all data.

```
In [25]: pd.read_table('column17_data_summary.out', header = -1)
```

Out[25]:

	0	1
0	VALID	7599
1	NaN	5093632

0.0.13 Column 18 - NYCHA Housing Developtment Names

Code: ***column18_data_quality.py*** This code is used to return the data line by line. The code checks if the NYCHA Housing Developtment name is a valid name compared to the full list at http://www1.nyc.gov/site/nycha/about/developments.page.

```
In [26]: pd.read_table('column18_data_quality.out', header = -1).head()
```

Out[26]:

	0	1	2	3
0	NaN	TEXT	housing names	NaN
1	NaN	TEXT	housing names	NaN
2	NaN	TEXT	housing names	NaN
3	NaN	TEXT	housing names	NaN
4	NaN	TEXT	housing names	NaN

Code: ***column18_data_summary.py*** This code is used to return a summary of all data.

```
In [27]: pd.read_table('column18_data_summary.out', header = -1)
```

Out[27]:

	0	1
0	VALID	253205
1	NaN	4848026

0.1 Some Data Quality Issues

0.1.1 Column 17 - Park Names

Run the code ***column17_data_quality_issue.py***.

We found the list of NYC park names from the website (https://www.nycgovparks.org/park-features/parks-list?boro=X) and compared our data in this column to the list (we put it in 'parks_nm.txt'). We treat those names not in this list as 'INVALID' and print them out in the 'column17_data_quality_issue.out' file.

```
In [28]: pd.read_table('column17_data_quality_issue.out', header=-1).head(10)
```

Out[28]:

	0	1
0	RAINEY PARK BRONX	38
1	UNNAMED PARK ON E 164TH STREET	1
2	UNNAMED PARK ON BRUCKNER EXPRESSWAY WEST OF MO...	5
3	EAGLE SLOPE	1
4	UNNAMED PARK ON E 120TH STREET	3
5	UNNAMED PARK ON E 122ND STREET	1
6	UNNAMED PARK ON E 177TH STREET	2
7	CORONA MAC PARK	3
8	WHITE PLAYGROUND MANHATTAN	3
9	CLASSON PLAYGROUND AT CLASSON AVENUE & LAFAYET...	3

Put those 'invalid' names into a txt file:

```
In [29]: with open('column17_data_quality_issue.out','r') as f:
             file = open('column17_data_quality_issue.txt','w')
             for line in f:
                 file.write(line)
             file.close()
```

After comparing the two files 'column17_data_quality_issue.txt' and 'parks_nm.txt' we can find that some park names in column17 which are not in 'parks_nm.txt' are actually not invalid but due to some difference in the naming methods. For example 'White Playground Manhattan' in column17 is actually that of 'White Playground' in 'parks_nm.txt', which are essentially the same park. But here we will treat it as invalid. This is a data quality issue that we should look further into. We will manually correct the name list in Part II.

0.1.2 Column 18 - NYCHA Housing Developtment Names

Run the code ***column18_data_quality_issue.py***.

We found the list of NYCHA housing developtment names from the website (http://www1.nyc.gov/site/nycha/about/developments.page) and compared our data in this column to the list (we put it in 'hadevelopt.txt'). We treat those names not in this list as 'INVALID' and print them out in the 'column18_data_quality_issue.out' file.

```
In [30]: pd.read_table('column18_data_quality_issue.out', header=-1).head(10)
```

```
Out[30]:                                        0     1
         0                  BAYSIDE-OCEAN BAY APTS  1256
         1          UNION AVENUE-EAST 163RD STREET   132
         2                       SARATOGA SQUARE   137
         3                    MILL BROOK EXTENSION   457
         4                          UPACA (SITE 6)   113
         5                   33-35 SARATOGA AVENUE   365
         6              SOUTH BRONX AREA (SITE 402)   281
         7                   BARUCH HOUSES ADDITION   120
         8            EAST 173RD STREET-VYSE AVENUE   459
         9  WASHINGTON HEIGHTS REHAB (GROUPS 1&2)     5
```

Put those 'invalid' names into a txt file:

```
In [31]: with open('column18_data_quality_issue.out','r') as f:
             file = open('column18_data_quality_issue.txt','w')
             for line in f:
                 file.write(line)
             file.close()
```

After comparing the two files 'column18_data_quality_issue.txt' and 'hadevelopt.txt' we can find that some park names in column18 which are not in 'parks_nm.txt' are actually not invalid but due to some difference in the naming methods. For example '33-35 SARATOGA AVENUE' in column18 is actually that of 'Saratoga Village' in 'hadevelopt.txt', which are probably the same place. But here we will treat it as invalid. This is a data quality issue that we should look further into. We will manually correct the name list in Part II.

0.1.3 Column 6-7 - Offense Classification Code & Description

Run the code ***column6_7_quality_issue.py***.

This code intends to check if offense classification code in column6 and offense classification description in column7 correctly one to one correspondent (i.e. one code in column6 corresponds to one description in column7.)

```
In [32]: df = pd.read_table('column6_7_quality_issue.out', header=-1)
         df.head(20)
```

```
Out[32]:       0                                1       2
         0   101  MURDER & NON-NEGL. MANSLAUGHTER    4574
         1   102       HOMICIDE-NEGLIGENT-VEHICLE      93
         2   103   HOMICIDE-NEGLIGENT,UNCLASSIFIE      33
         3   104                             RAPE   13791
         4   105                              NaN       2
         5   105                          ROBBERY  198772
         6   106                              NaN      55
         7   106                  FELONY ASSAULT  184069
         8   107                              NaN       1
         9   107                         BURGLARY  191406
         10  109                              NaN       9
```

```
11  109                     GRAND LARCENY  429196
12  110   GRAND LARCENY OF MOTOR VEHICLE  102061
13  111                              NaN       1
14  111    POSSESSION OF STOLEN PROPERTY    9112
15  112                              NaN       6
16  112                      THEFT-FRAUD   56762
17  113                              NaN       1
18  113                          FORGERY   49303
19  114                            ARSON   13984
```

From 'column6_7_quality_issue.out' we can find out that for some classification codes in column6 there are two or more corresponding descriptions in column7. Print out those indices where there are multiple correspondence between the two columns.

```
In [33]: df = pd.read_table('column6_7_quality_issue.out', header=-1)
         from collections import Counter
         print([k for k,v in Counter(df[0]).items() if v>1])
```

```
[107, 358, 364, 109, 359, 677, 678, 113, 125, 365, 117, 360, 578, 366, 361, 675, 340, 341, 343,
```

Here are some examples of comparison between the two columns.

```
In [34]: df[df[0]==107]
```

```
Out[34]:      0         1       2
         8  107       NaN       1
         9  107  BURGLARY  191406
```

```
In [35]: df[df[0]==364]
```

```
Out[35]:       0                                      1     2
         97   364                                    NaN   402
         98   364  AGRICULTURE & MRKTS LAW-UNCLASSIFIED    83
         99   364        OTHER STATE LAWS (NON PENAL LA  5505
         100  364      OTHER STATE LAWS (NON PENAL LAW)     4
```

```
In [36]: df[df[0]==343]
```

```
Out[36]:      0                                1     2
         61  343                              NaN     2
         62  343  OTHER OFFENSES RELATED TO THEF  9731
         63  343               THEFT OF SERVICES  2778
```

We can find that some of the descriptions in column7 are 'NaN' which could be missing data where there's still codes in column6. Some of the codes in column6 corresponds to more than two descriptions in column7 which could be mistakes in typing or sub-classes etc. This is a quality issue we should look further into. We will probably compare to other data set from the internet in Part II. (We have put the output data in 'column6_7_quality_issue.txt'.)

```
In [37]: with open('column6_7_quality_issue.out','r') as f:
             file = open('column6_7_quality_issue.txt','w')
             for line in f:
                 file.write(line)
             file.close()
```

0.1.4 Column 8-9 - Internal Classification Code & Description

Run the code ***column8_9_quality_issue.py***.

This code intends to check if offense classification code in column8 and offense classification description in column9 correctly one to one correspondent (i.e. one code in column8 corresponds to one description in column9.)

```
In [38]: df = pd.read_table('column8_9_quality_issue.out', header=-1)
         df.head(20)
```

Out[38]:

	0	1	2
0	NaN	NaN	4574
1	101.0	ASSAULT 3	438130
2	104.0	VEHICULAR ASSAULT (INTOX DRIVE	220
3	105.0	STRANGULATION 1ST	11648
4	106.0	ASSAULT 2,1,PEACE OFFICER	18376
5	107.0	END WELFARE VULNERABLE ELDERLY PERSON	51
6	109.0	ASSAULT 2,1,UNCLASSIFIED	154046
7	110.0	MENACING 1ST DEGREE (VICT PEAC	786
8	111.0	MENACING,PEACE OFFICER	863
9	112.0	MENACING 1ST DEGREE (VICT NOT	704
10	113.0	MENACING,UNCLASSIFIED	70185
11	114.0	OBSTR BREATH/CIRCUL	12369
12	115.0	RECKLESS ENDANGERMENT 2	9660
13	117.0	RECKLESS ENDANGERMENT 1	17842
14	119.0	PROMOTING SUICIDE ATTEMPT	3
15	121.0	HOMICIDE,NEGLIGENT,VEHICLE	15
16	122.0	HOMICIDE, NEGLIGENT, VEHICLE,	78
17	125.0	HOMICIDE,NEGLIGENT,UNCLASSIFIE	33
18	143.0	ABORTION 1	4
19	146.0	ABORTION 2, 1, SELF	1

```
In [39]: df = pd.read_table('column8_9_quality_issue.out', header=-1)
         from collections import Counter
         [k for k,v in Counter(df[0]).items() if v>1]
```

```
Out[39]: []
```

From 'column8_9_quality_issue.out' we can find out that these two columns are actually one to one correspondent between each other and there's no multiple correspondence issue that we need to look further into. (We have put the output data in 'column8_9_quality_issue.txt'.)

```
In [40]: with open('column8_9_quality_issue.out','r') as f:
             file = open('column8_9_quality_issue.txt','w')
             for line in f:
                 file.write(line)
             file.close()
```

Data quality and summary 19-23

April 17, 2017

0.0.1 Column 19 - X-coordinate

Code: ***column19_data_quality.py*** This code is used to return the data line by line. The code checks if the X-coordinate is an integer and if it is in New York City. type: X-coordinate for New York State Plane Coordinate System, Long Island Zone, NAD 83, units feet (FIPS 3104).

Code: ***column19_data_summary.py*** This code is used to return a summary of all data. There is no INVALID data in our dataset although some of them are NaN.

```
In [20]: pd.read_table('column19_data_quality.out', header = -1).head()
```

```
Out[20]:            0        1                                                 2  \
         0  1007314.0  INTEGER  X-coordinate for New York State Plane Coordina...
         1  1043991.0  INTEGER  X-coordinate for New York State Plane Coordina...
         2   999463.0  INTEGER  X-coordinate for New York State Plane Coordina...
         3  1060183.0  INTEGER  X-coordinate for New York State Plane Coordina...
         4   987606.0  INTEGER  X-coordinate for New York State Plane Coordina...

                3
         0  VALID
         1  VALID
         2  VALID
         3  VALID
         4  VALID
```

```
In [21]: pd.read_table('column19_data_summary.out', header=-1)
```

```
Out[21]:        0        1
         0  VALID  4913085
         1    NaN   188146
```

0.0.2 Column 20 - Y-coordinate

Code: ***column20_data_quality.py*** This code is used to return the data line by line. The code checks if the Y-coordinate is an integer and if it is in New York City. type: Y-coordinate for New York State Plane Coordinate System, Long Island Zone, NAD 83, units feet (FIPS 3104).

Code:*** column20_data_summary.py *** This code is used to return a summary of all data. There is no INVALID data in our dataset although some of them are NaN.

```
In [15]: pd.read_table('column20_data_quality.out', header = -1).head()
```

```
Out[15]:           0        1                                                 2      3
         0  241257.0  INTEGER  Y-coordinate for New York State Plane Coordina...  VALID
         1  193406.0  INTEGER  Y-coordinate for New York State Plane Coordina...  VALID
         2  231690.0  INTEGER  Y-coordinate for New York State Plane Coordina...  VALID
         3  177862.0  INTEGER  Y-coordinate for New York State Plane Coordina...  VALID
         4  208148.0  INTEGER  Y-coordinate for New York State Plane Coordina...  VALID
```

```
In [7]: pd.read_table('column20_data_summary.out', header = -1)
```

```
Out[7]:        0        1
        0  VALID  4913085
        1    NaN   188146
```

0.0.3 Column 21 - Latitude

Code: ***column21_data_quality.py*** This code is used to return the data line by line. The code checks if the Latitude is an float and if it is valid in New York City. type: Latitude coordinate for Global Coordinate System, WGS 1984, decimal degrees (EPSG 4326)

Code: ***column21_data_summary.py*** This code is used to return a summary of all data. There is no INVALID data in our dataset although some of them are NaN.

```
In [16]: pd.read_table('column21_data_quality.out', header = -1).head()
```

```
Out[16]:           0        1                                       2      3
         0  40.828848  DECIMAL  Latitude for Global Coordinate System  VALID
         1  40.697338  DECIMAL  Latitude for Global Coordinate System  VALID
         2  40.802607  DECIMAL  Latitude for Global Coordinate System  VALID
         3  40.654549  DECIMAL  Latitude for Global Coordinate System  VALID
         4  40.738002  DECIMAL  Latitude for Global Coordinate System  VALID
```

```
In [9]: pd.read_table('column21_data_summary.out', header = -1)
```

```
Out[9]:        0        1
        0  VALID  4913085
        1    NaN   188146
```

0.0.4 Column 22 - Longitud

Code: ***column22_data_quality.py*** This code is used to return the data line by line. The code checks if the Longitude is an float and if it is valid in New York City. type: Longitude coordinate for Global Coordinate System, WGS 1984, decimal degrees (EPSG 4326)

Code: ***column22_data_summary.py*** This code is used to return a summary of all data. There is no INVALID data in our dataset although some of them are NaN.

```
In [17]: pd.read_table('column22_data_quality.out', header = -1).head()
```

```
Out[17]:           0        1                                       2      3
         0 -73.916661  DECIMAL  Longitude for Global Coordinate System  VALID
         1 -73.784557  DECIMAL  Longitude for Global Coordinate System  VALID
         2 -73.945052  DECIMAL  Longitude for Global Coordinate System  VALID
         3 -73.726339  DECIMAL  Longitude for Global Coordinate System  VALID
         4 -73.987891  DECIMAL  Longitude for Global Coordinate System  VALID
```

```
In [11]: pd.read_table('column22_data_summary.out', header = -1)
```

```
Out[11]:        0        1
         0  VALID  4913085
         1    NaN   188146
```

0.0.5 Column 23 - Location

Code: ***column23_data_quality.py*** This code is used to return the data line by line. The code checks if the latitude and longitude is an float and if it is valid in New York City. type: (Latitude, Longitude) coordinate for Global Coordinate System, WGS 1984, decimal degrees (EPSG 4326)

Code: ***column23_data_summary.py*** This code is used to return a summary of all data. There is no INVALID data in our dataset although some of them are NaN.

```
In [18]: pd.read_table('column23_data_quality.out', header = -1).head()
```

```
Out[18]:                                0        1  \
         0  (40.828848333, -73.916661142)  DECIMAL
         1  (40.697338138, -73.784556739)  DECIMAL
         2  (40.802606608, -73.945051911)  DECIMAL
         3  (40.654549444, -73.726338791)  DECIMAL
         4     (40.7380024, -73.98789129)  DECIMAL

                                            2      3
         0  Location for Global Coordinate System  VALID
         1  Location for Global Coordinate System  VALID
         2  Location for Global Coordinate System  VALID
         3  Location for Global Coordinate System  VALID
         4  Location for Global Coordinate System  VALID
```

```
In [13]: pd.read_table('column23_data_summary.out', header = -1)
```

```
Out[13]:        0        1
         0  VALID  4913085
         1    NaN   188146
```

Data Trend 0-5

April 17, 2017

0.1 Some Trends

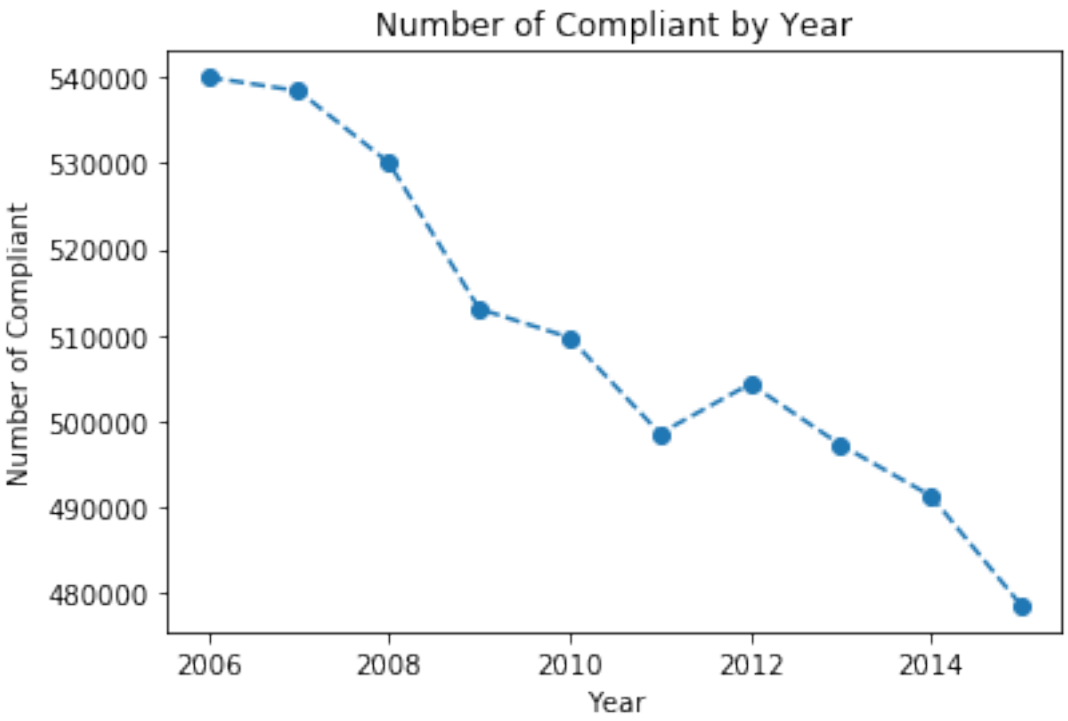
0.1.1 1. Number of Crimes by Year

Code: ***col5_report_year.py*** This code returns number of crimes by year.

```
In [1]: import matplotlib.pyplot as plt
        import pandas as pd
        import numpy as np

        %matplotlib inline
```

```
In [2]: df = pd.read_table('col5_report_year.out',header=-1)
        df = df.sort([0], ascending=True)
        plt.plot(list(df[0]),list(df[1]),linestyle='--', marker='o')
        plt.title("Number of Compliant by Year")
        plt.xlabel("Year")
        plt.ylabel("Number of Compliant")
        plt.show()
```

```
/Users/sunevan/anaconda/lib/python3.6/site-packages/ipykernel/__main__.py:2: FutureWarning: sort
  from ipykernel import kernelapp as app
```



Trend: Overall, the crime decreased year over year. Our goal is to identify when, where, and why the crime decreased.

0.1.2 2. Number of crimes by year and by month

Code: ***col5_report_year_month.py*** This code returns number of crimes by each year and each month.

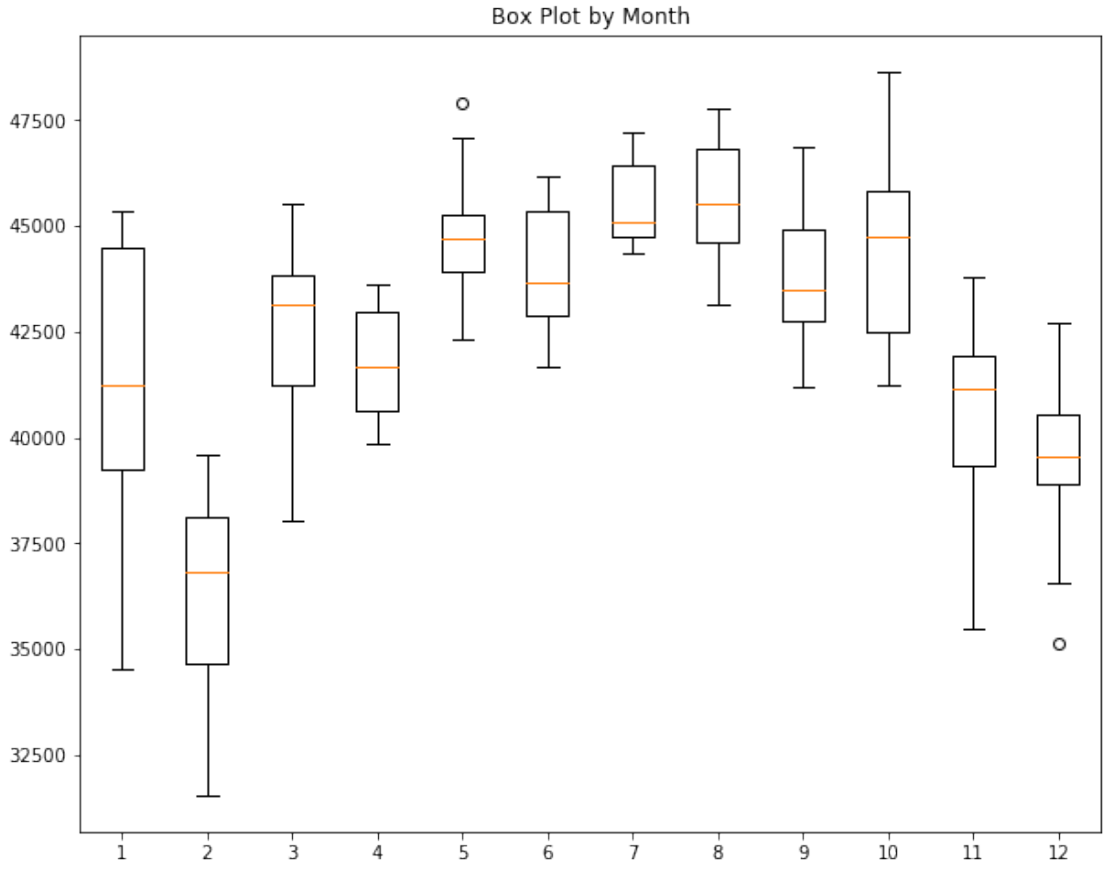
```
In [3]: fig = plt.figure(figsize=(10, 8))
        df = pd.read_table('col5_report_year_month.out',header=-1)
        df = df.sort([0,1], ascending=True)
        for i in df[0].unique():
            plt.plot(list(df[df[0]==i][1]),list(df[df[0]==i][2]),linestyle='--', marker='o',labe
        plt.xticks(list(range(1,13)))
        plt.legend(loc="upper left")
        plt.title("Number of compliants by year and by month")
        plt.show()
```

```
/Users/sunevan/anaconda/lib/python3.6/site-packages/ipykernel/__main__.py:3: FutureWarning: sort
  app.launch_new_instance()
```

It seems that Jan and Feb in 2015 has some abnormal number of crimes compared to the other month.

I also plot a boxplot and clearly Feburay has the lowest number of crimes on average (partialy is becasue of a shorter month). It seems there is an outliar in May and December.

```
In [4]: fig = plt.figure(figsize=(10, 8))
        month = list()
        for i in (list(range(1,13))):
            month.append(df[df[1]==i][2])

        plt.boxplot(month)
        plt.title("Box Plot by Month")
        plt.show()
```

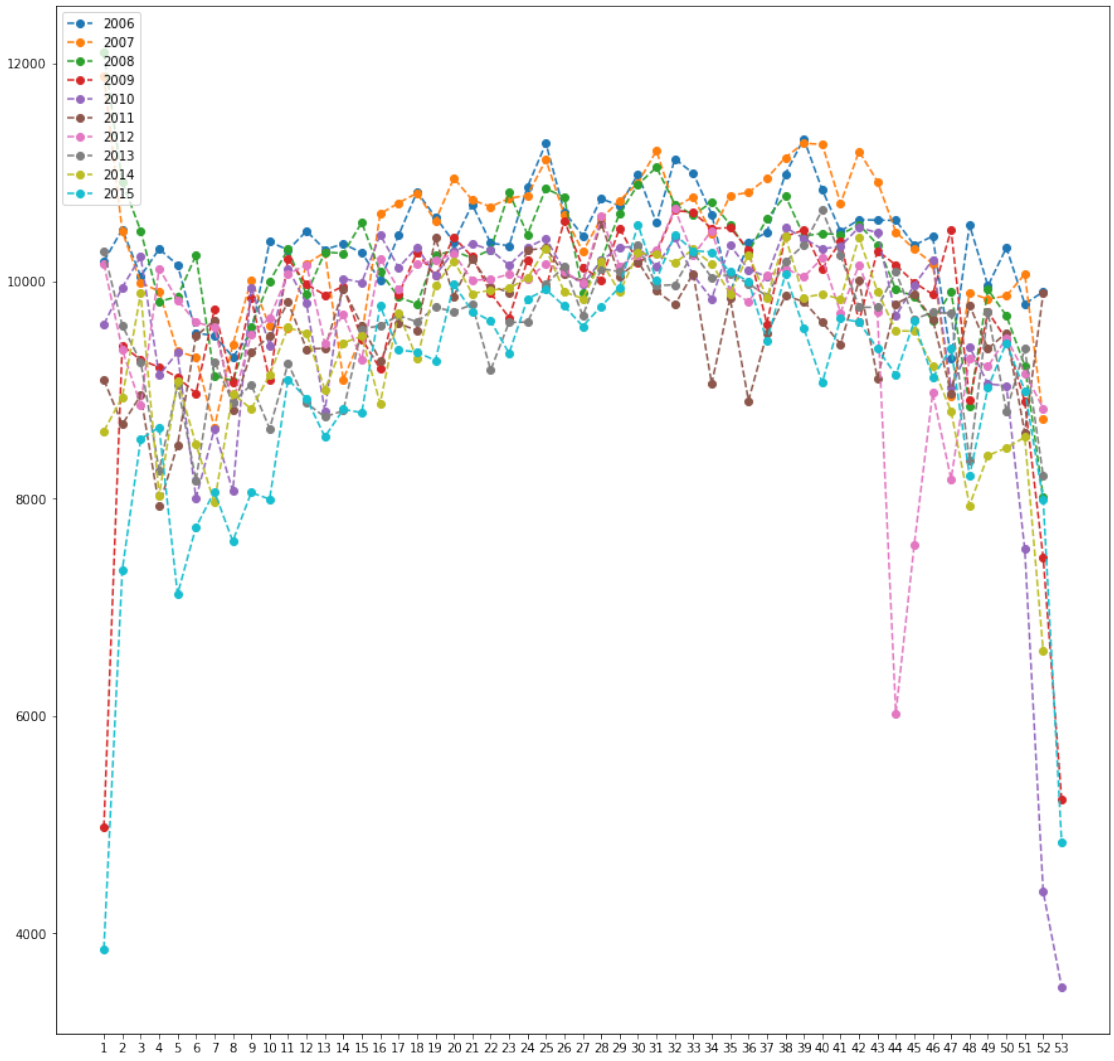


0.1.3 3. Number of crimes by year and by week

Code: ***col5_report_year_week.py*** This code provides number of crimes by year and by week.

```
In [5]: fig = plt.figure(figsize=(15, 15))
        df = pd.read_table('col5_report_year_week.out',header=-1)
        df = df.sort([0,1], ascending=True)
        for i in df[0].unique():
            plt.plot(list(df[df[0]==i][1]),list(df[df[0]==i][2]),linestyle='--', marker='o',labe
        plt.legend(loc="upper left")
        plt.xticks(list(df[1].unique()))
        plt.show()
```

```
/Users/sunevan/anaconda/lib/python3.6/site-packages/ipykernel/__main__.py:3: FutureWarning: sort
  app.launch_new_instance()
```



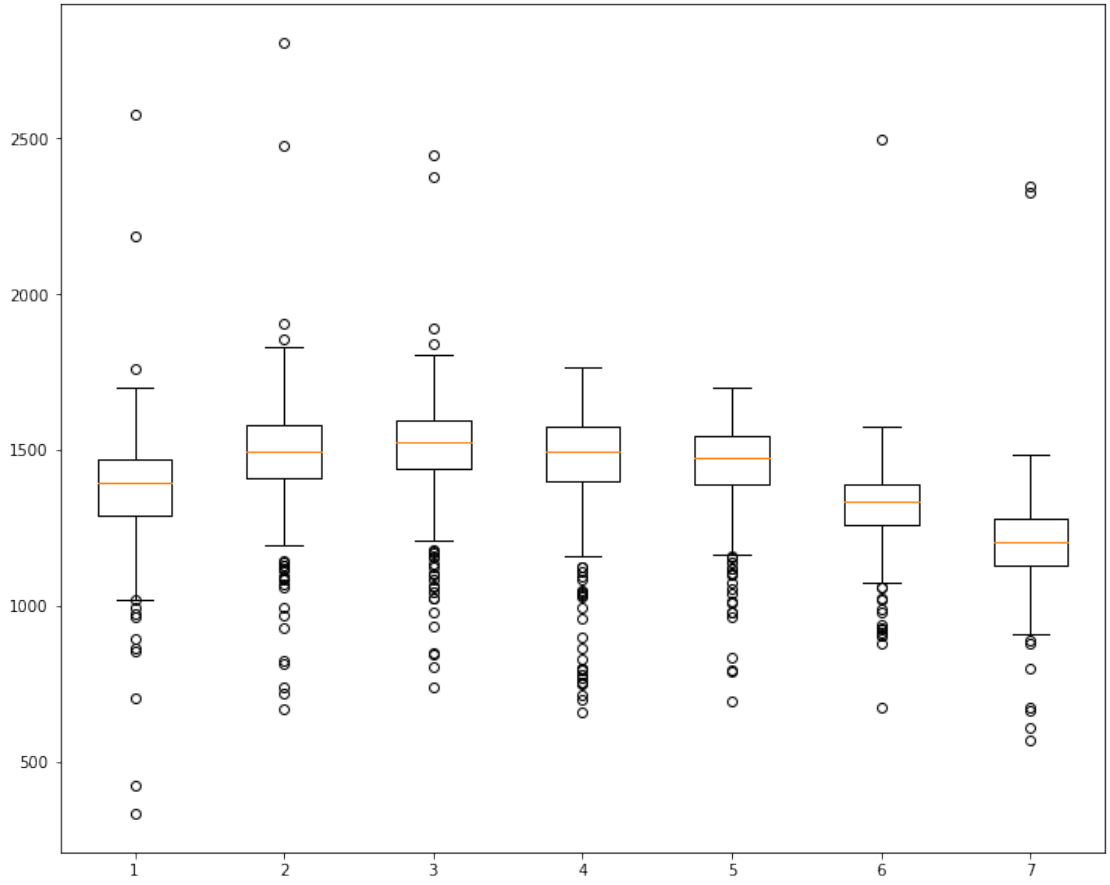
After plotting by week, it is clear that the week 44 in 2012 has an abnormal trend. It is probably because of hurricane sandy happened during that period.

0.1.4 4. Number of crimes by day of week

Code: ***col5_report_year_week_day.py*** This code provides number of crimes by year and weeknum in each day of the week.

```
In [6]: fig = plt.figure(figsize=(12, 10))
        df = pd.read_table('col5_report_year_week_day.out',header=-1)
        df = df.sort([0,1], ascending=True)
        dow = list()
        for i in (list(range(1,8))):
            dow.append(df[df[2]==i][3])
        plt.boxplot(dow)
        plt.show()
```

```
/Users/sunevan/anaconda/lib/python3.6/site-packages/ipykernel/__main__.py:3: FutureWarning: sort
  app.launch_new_instance()
```



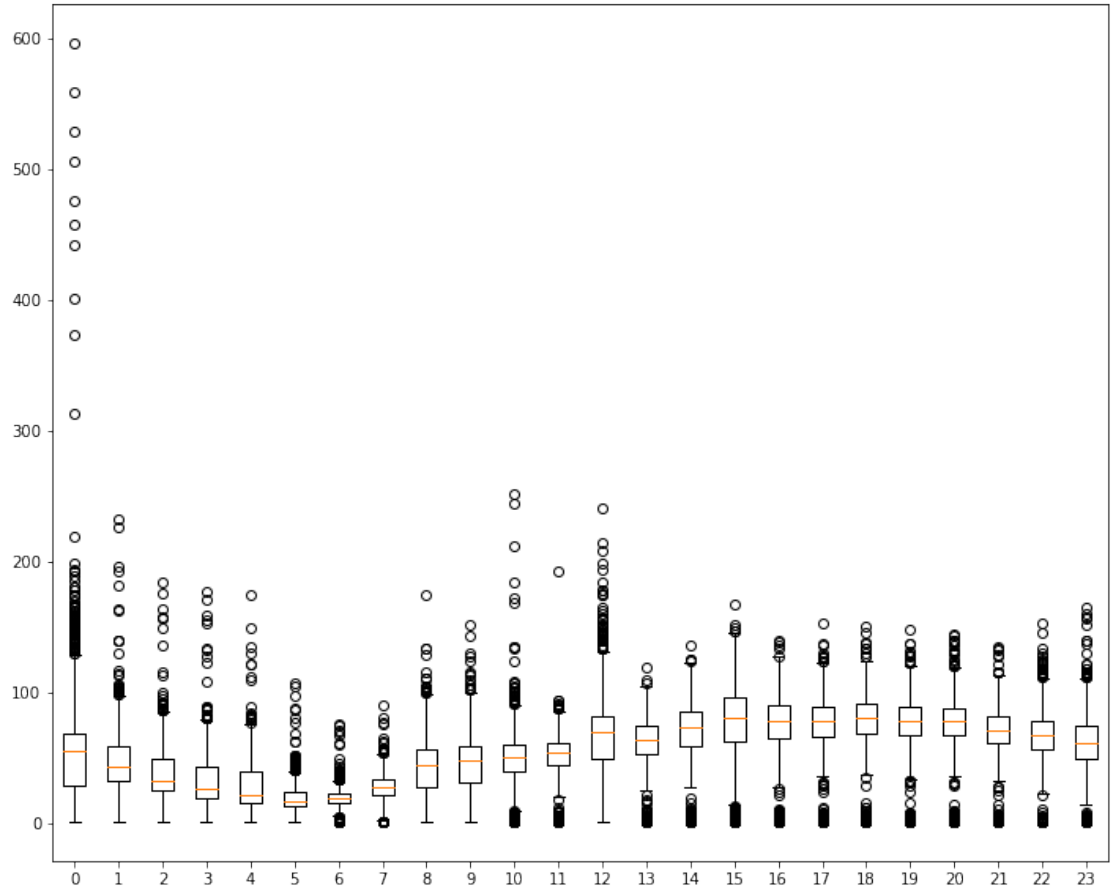
By looking at the boxplot, Tue-Fri is the peak and weekend has less compliants. Also, weekdays have more outliars.

0.1.5 5. Number of compliants by hour

Code: ***col1_2_hour.py***. This code is to check number of crimes by hour.

```
In [7]: fig = plt.figure(figsize=(12, 10))
        df = pd.read_table('col1_2_hour.out',header=-1)
        df = df.sort([0,1,2,3], ascending=True)
        dow = list()
        for i in (list(range(0,24))):
            dow.append(df[df[3]==i][4])
        plt.boxplot(dow)
        plt.xticks(list(range(1,25)),list(range(0,24)))
        plt.show()
```

```
/Users/sunevan/anaconda/lib/python3.6/site-packages/ipykernel/__main__.py:3: FutureWarning: sort
  app.launch_new_instance()
```



Number of crimes is gradually increasing after 7 am and then start decreasing after midnight. However, there are some very high abnormal outliars happened between 12 - 1 am.

```
In [8]: dfzero = df[df[3]==0]
        dfoutliar = dfzero[dfzero[4]>300]
        dfoutliar
```

Out[8]:

	0	1	2	3	4
17430	2006	52	7	0	506
17454	2007	1	1	0	596
26214	2008	1	2	0	529
34926	2009	1	4	0	373
52374	2010	53	5	0	475
61134	2011	52	6	0	558
69894	2012	52	7	0	442
69942	2013	1	2	0	401

```
78702  2014   1  3  0  458
87390  2015   1  4  0  313
```

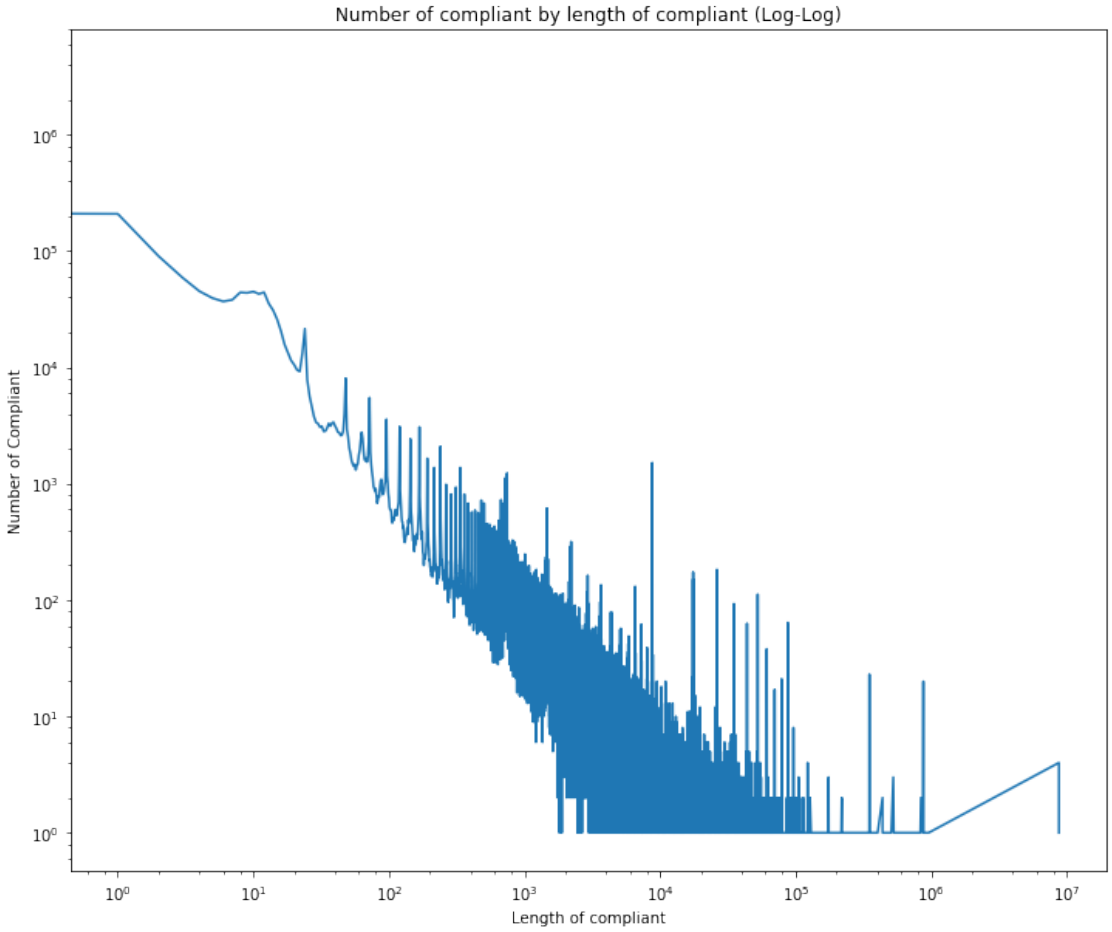
The outliars are usually between 12 and 1 am on the new year. It makes sense since the NYE celebrations.

0.1.6 6. Number of crimes by length of compliant

Code: ***length_of_compliant.py***. The code provides number of crimes by the length of complaint. When the crime has no to_date and to_time, the length will be counted as **0**.

```
In [9]: fig = plt.figure(figsize=(12, 10))
        df = pd.read_table('col1_2_3_4_length_of_compliant.out',header=-1)
        df = df.sort([0], ascending=True)

        plt.plot(list(df[0]),list(df[1]),linestyle='-')
        plt.title("Number of compliant by length of compliant (Log-Log)")
        plt.xlabel("Length of compliant")
        plt.ylabel("Number of Compliant")
        plt.xscale('log')
        plt.yscale('log')
        plt.show()
```

```
/Users/sunevan/anaconda/lib/python3.6/site-packages/ipykernel/__main__.py:3: FutureWarning: sort
  app.launch_new_instance()
```



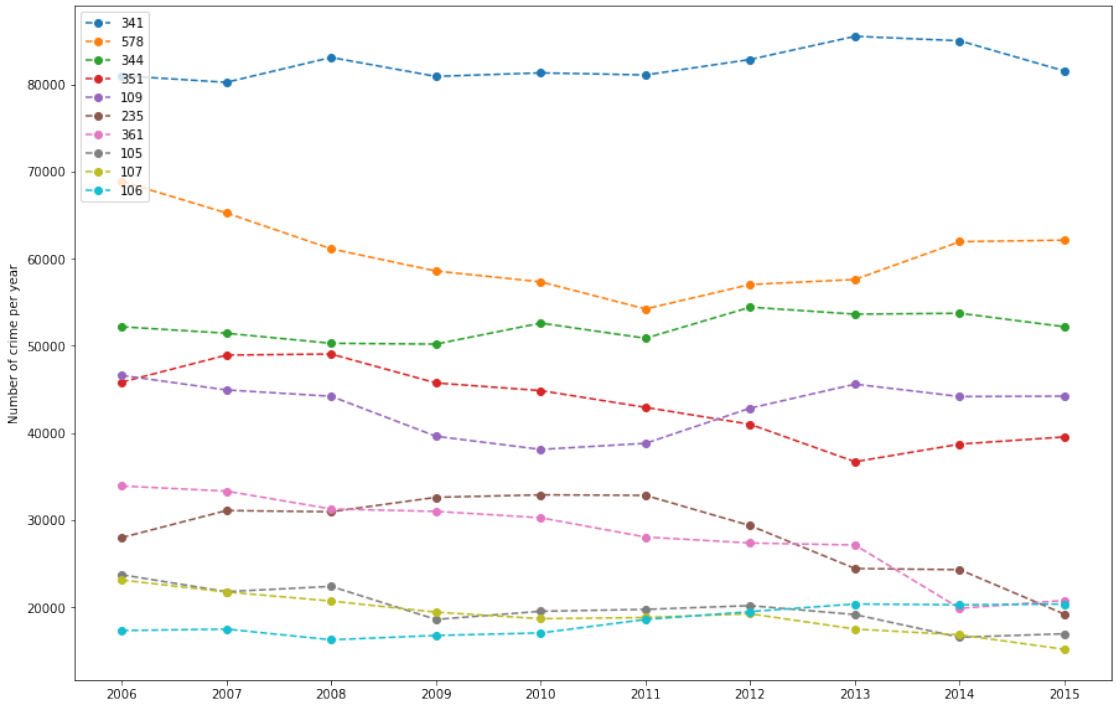
Overall,number of crimes decays over the length of the compliant after transforming to log-log.

0.1.7 7. Number of compliants by year and by crime type

Code: ***col5_6_report_kycd.py***. The code provides number of crimes by year and by KY_CD, which stands for the crime type.

```
In [10]: fig = plt.figure(figsize=(15, 10))
         df = pd.read_table('col5_6_report_year_kycd.out',header=-1)
         df.columns = ["year","kycd","count"]
         #find the top 10 KY_CD
         top_kycd = df.groupby(["kycd"], as_index=False).sum().sort(["count"], ascending=False)[
         df = df.sort(["year"],ascending=True)
         for i in list(top_kycd):
             plt.plot(list(df[df["kycd"]==i]["year"]),list(df[df["kycd"]==i]["count"]),linestyle
         plt.legend(loc="upper left")
         plt.xticks(list(df["year"].unique()))
         plt.ylabel("Number of crime per year")
         plt.show()
```

```
/Users/sunevan/anaconda/lib/python3.6/site-packages/ipykernel/__main__.py:5: FutureWarning: sort
/Users/sunevan/anaconda/lib/python3.6/site-packages/ipykernel/__main__.py:6: FutureWarning: sort
```



From the list, the crime 341 is the most popluar and did not decrease over the year. It seems that crime 235 and 361 may contribute the year over year decrease as the trend continued going down after 2012.

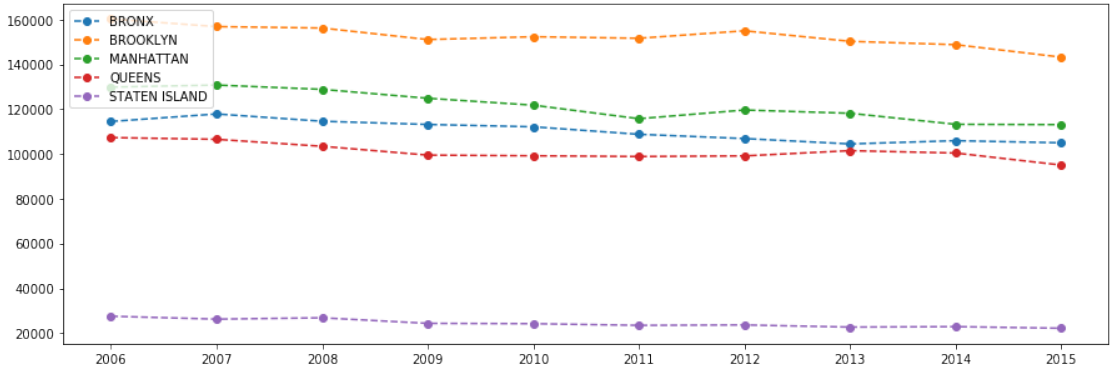
0.1.8 8. Number of compliants by year and by borough

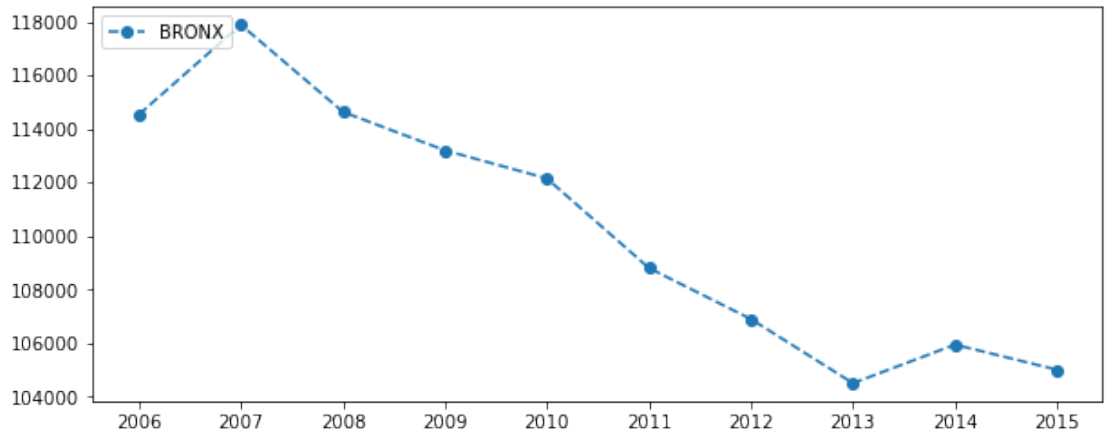
Code: ***col5_13_report_year_borough.py***. The code provides number of crime by number of compliants, year, and borough.

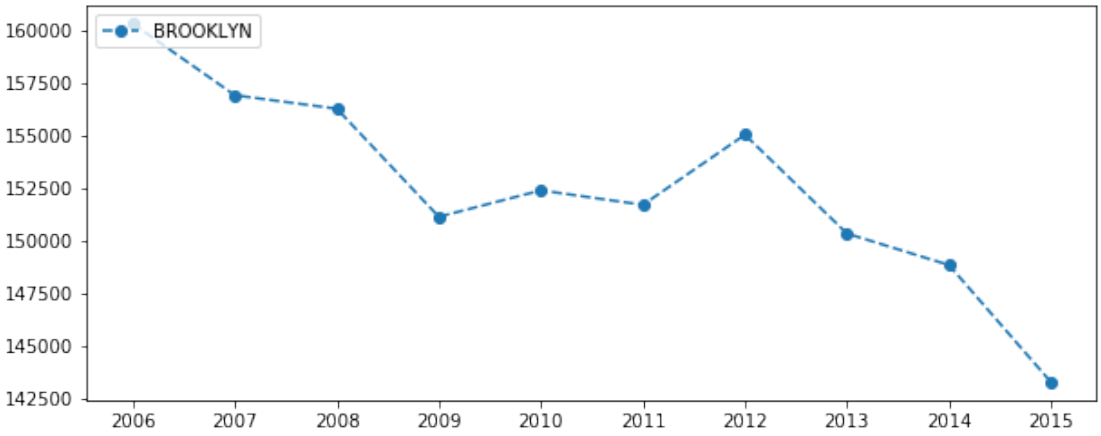
```
In [11]: fig = plt.figure(figsize=(15,5))
         df = pd.read_table('col5_13_report_year_borough.out',header=-1)
         df = df.sort([0,1], ascending=True)
         for i in df[1].unique():
             plt.plot(list(df[df[1]==i][0]),list(df[df[1]==i][2]),linestyle='--', marker='o',lab
         plt.legend(loc="upper left")
         plt.xticks(list(df[0].unique()))
         plt.show()



         for i in df[1].unique():
             fig = plt.figure(figsize=(10,4))
```

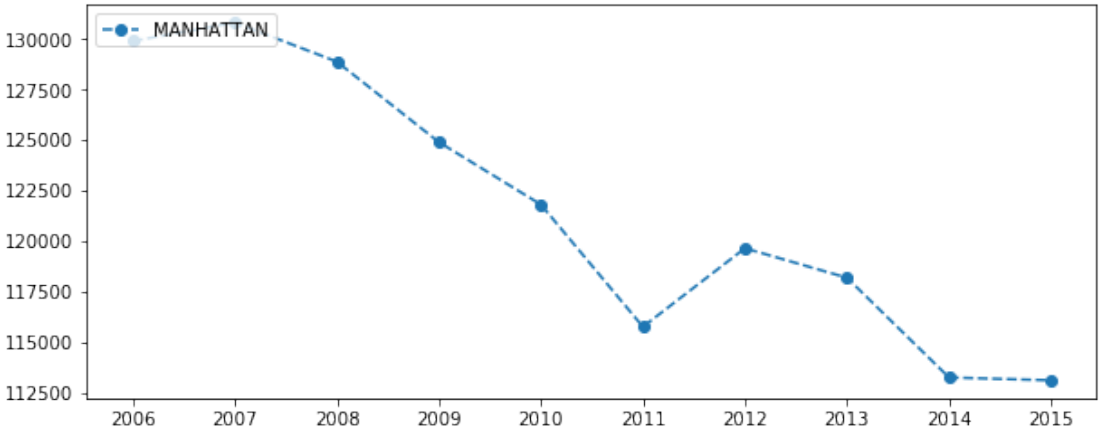
```
        plt.plot(list(df[df[1]==i][0]),list(df[df[1]==i][2]),linestyle='--', marker='o',lab
        plt.legend(loc="upper left")
        plt.xticks(list(df[0].unique()))
        plt.show()
```

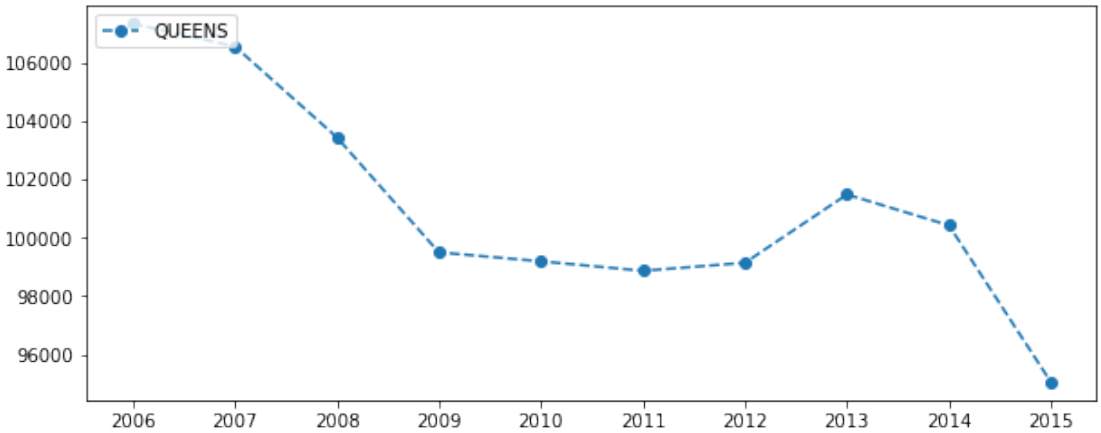
```
/Users/sunevan/anaconda/lib/python3.6/site-packages/ipykernel/__main__.py:3: FutureWarning: sort
  app.launch_new_instance()
```

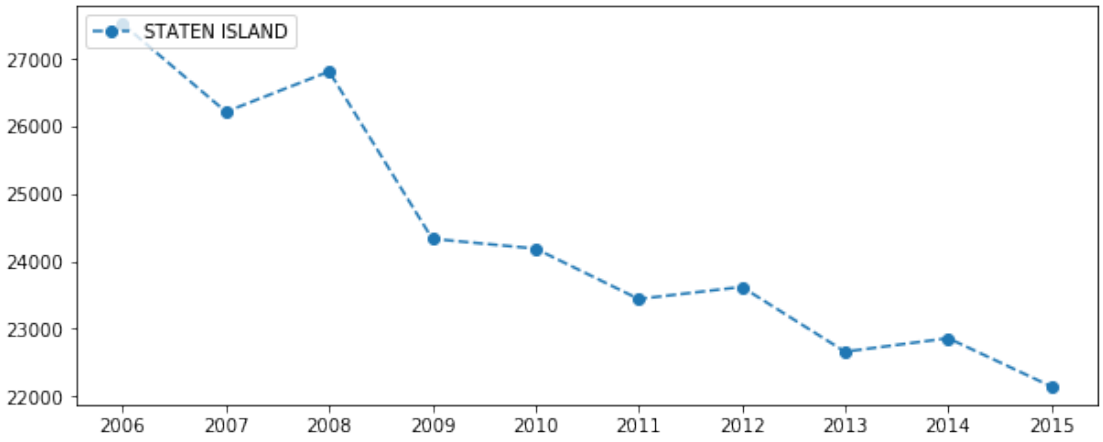





BROOKLYN
160000
157500
155000
152500
150000
147500
145000
142500
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015


MANHATTAN
130000
127500
125000
122500
120000
117500
115000
112500
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015


QUEENS
106000
104000
102000
100000
98000
96000
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015



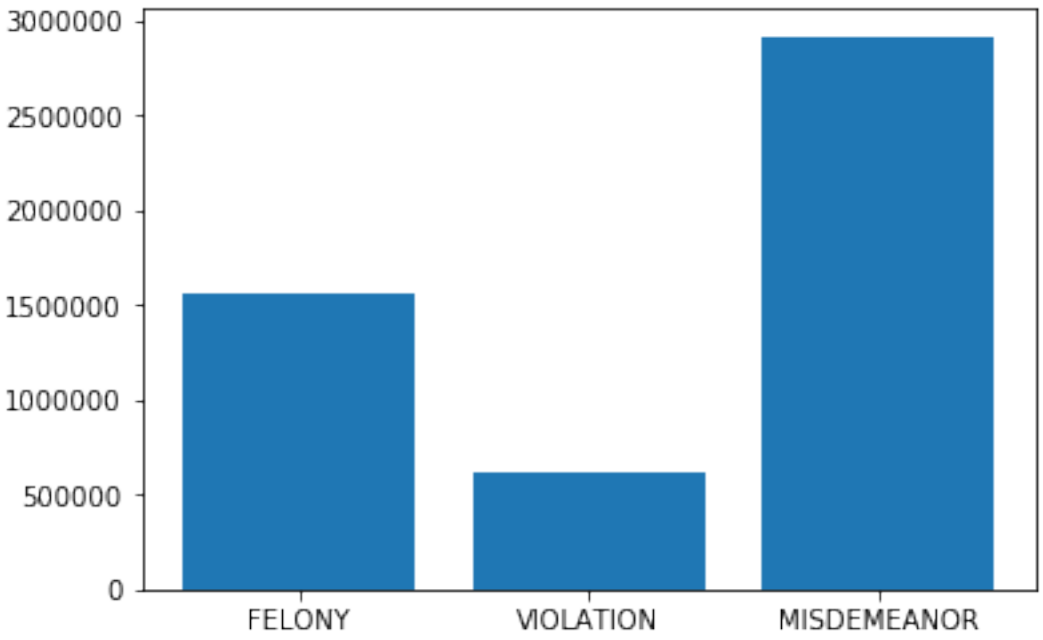
Looking at the stacked the line chart, the decreasing trend is not very clear. After plotting the trend borough by borough, it is more clear that broklyn contributes the most to the decrease, followed by manhattan.

Data Trend 6-18

April 17, 2017

0.0.1 Column 11 - Offense Level

```
In [2]: data = pd.read_table('column11_data_quality.out', header = -1)

In [3]: stats = {}
        for level in ['FELONY','MISDEMEANOR','VIOLATION']:
            stats[level] = len(data[data[0]==level])

In [4]: plt.bar(range(len(stats)),stats.values(),align='center')
        plt.xticks(range(len(stats)), stats.keys())

        plt.show()
```

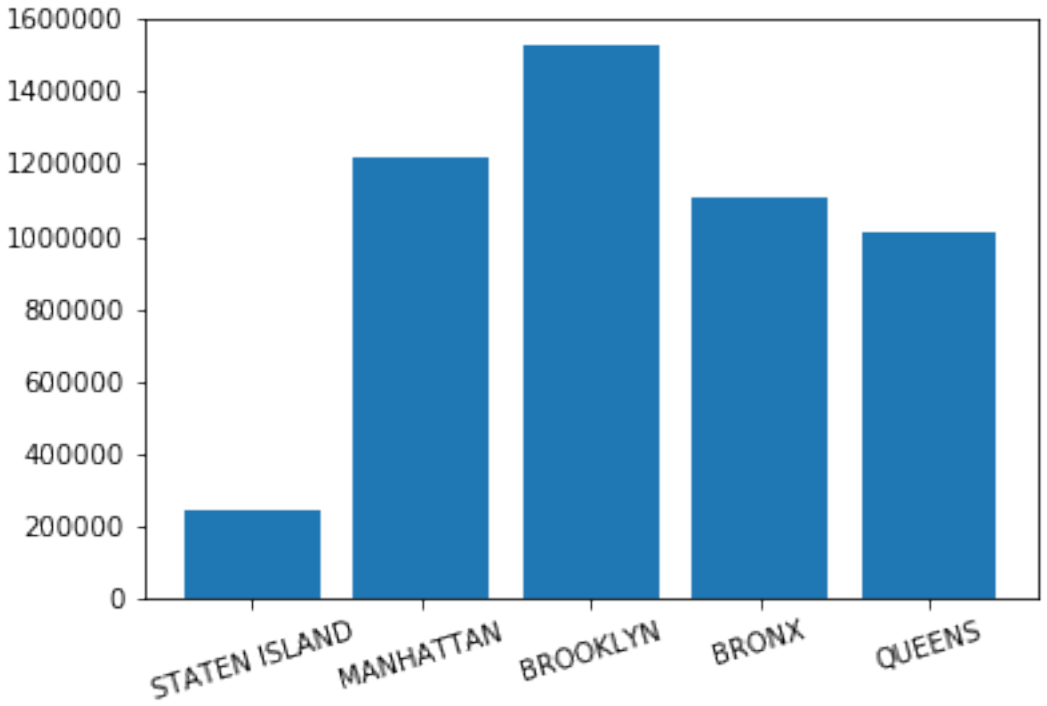


0.0.2 Column 13 - Borough Names

```
In [5]: data = pd.read_table('column13_data_quality.out', header = -1)
```

```
In [6]: stats = {}
        for boro in ['BRONX','BROOKLYN','MANHATTAN','QUEENS','STATEN ISLAND']:
            stats[boro] = len(data[data[0]==boro])
```

```
In [7]: plt.bar(range(len(stats)),stats.values(),align='center')
        plt.xticks(range(len(stats)), stats.keys(), rotation=17)

        plt.show()
```

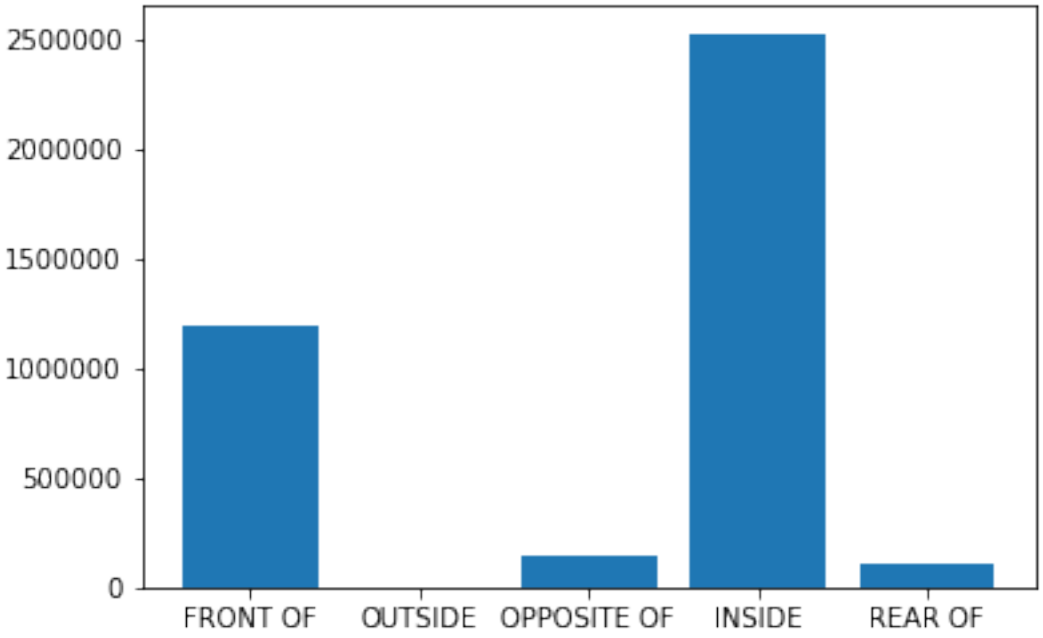


0.0.3 Column 15 - Occurrence Location Description

```
In [8]: data = pd.read_table('column15_data_quality.out', header = -1)
```

```
In [9]: stats = {}
        for loc in ['INSIDE', 'OUTSIDE', 'OPPOSITE OF', 'FRONT OF', 'REAR OF']:
            stats[loc] = len(data[data[0]==loc])
```

```
In [10]: plt.bar(range(len(stats)),stats.values(),align='center')
         plt.xticks(range(len(stats)), stats.keys())

         plt.show()
```


2500000
2000000
1500000
1000000
500000
0
FRONT OF
OUTSIDE
OPPOSITE OF
INSIDE
REAR OF

Heatmap of Crimes

April 17, 2017

0.1 Heatmap of Crimes

Now, we can plot a heatmap of crimes through history. From the heatmap we can find out whether there is any crime in an invalid location, for example, on the river.

0.1.1 Map of NYC

In [2]:
```
# You should have installed mpl_toolkits.basemap first. If not, try conda install basema
# or use anaconda prompt 'conda install -c conda-forge basemap-data-hires=1.0.8.dev0'

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.basemap import Basemap
```

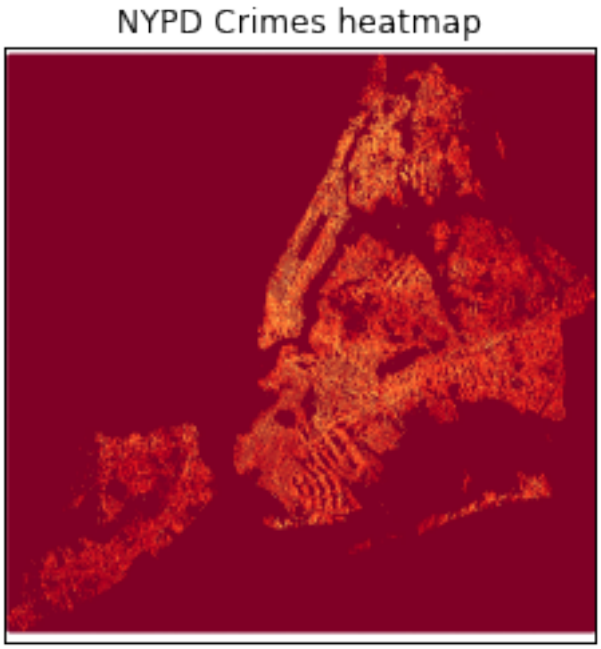
In [3]:
```
df = pd.read_csv("/Users/xinyan/Downloads/NYPD_Complaint_Data_Historic.csv")
```

```
/Users/xinyan/.local/lib/python3.5/site-packages/IPython/core/interactiveshell.py:2717: DtypeWar
  interactivity=interactivity, compiler=compiler, result=result)
```

In [5]:
```
"""
Heatmap of geolocated collisions in New York City area of a selected year.
This map only plot data with location information, i.e. latitude and longitude.
Data without location will be ignored.

Baesd on tweets heatmap by Kelsey Jordahl, Enthought in Scipy 2013 geospatial tutorial.
See more in github page: https://github.com/kjordahl/SciPy2013.

"""
def heatmap(df):
    """
    The function turns input dataframe into heatmap. Input should contain feature
    'LATITUDE' and 'LONGITUDE', otherwise it will not be plotted.
    """
    west, south, east, north = -74.26, 40.49, -73.70, 40.92 # NYC
    a = np.array(df['Latitude'].dropna())
```

```
    b = np.array(df['Longitude'].dropna())
    N = len(a)

    fig = plt.figure()
    m = Basemap(projection='merc', llcrnrlat=south, urcrnrlat=north,
                llcrnrlon=west, urcrnrlon=east, lat_ts=south, resolution='i')
    x, y = m(b, a)
    m.hexbin(x, y, gridsize=650, bins='log', cmap=cm.YlOrRd_r)
    plt.title("NYPD Crimes heatmap")
    plt.show()
```

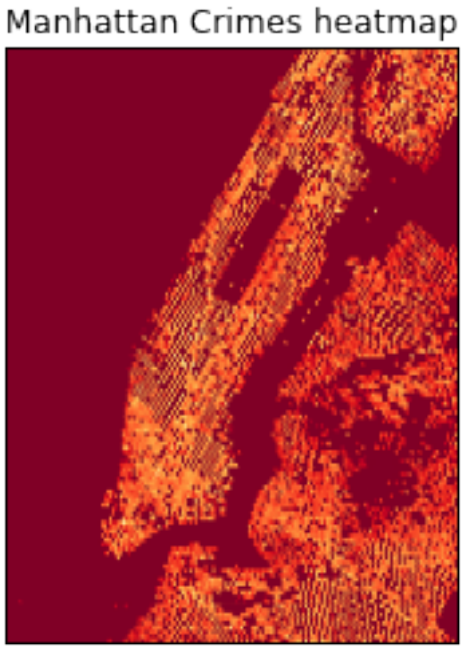
```
In [10]: heatmap(df)
```



0.1.2 Map of 5 boroughs

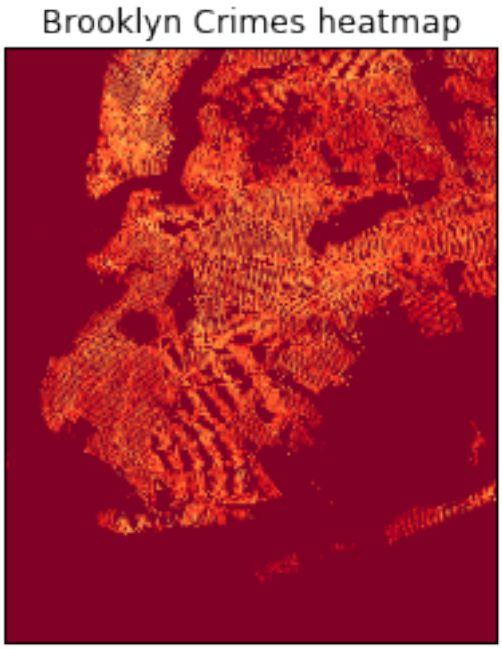
```
In [11]: def heatmap_mht(df):
             """
             The function turns input dataframe into heatmap. Input should contain feature
             'LATITUDE' and 'LONGITUDE', otherwise it will not be plotted.
             """
             west, south, east, north = -74.05, 40.68, -73.90, 40.83 # Manhattan, gridsize=650
             a = np.array(df['Latitude'].dropna())
             b = np.array(df['Longitude'].dropna())
             N = len(a)

             fig = plt.figure()
```

```
    m = Basemap(projection='merc', llcrnrlat=south, urcrnrlat=north,
                llcrnrlon=west, urcrnrlon=east, lat_ts=south, resolution='i')
    x, y = m(b, a)
    m.hexbin(x, y, gridsize=650, bins='log', cmap=cm.YlOrRd_r)
    plt.title("Manhattan Crimes heatmap")
    plt.show()

heatmap_mht(df)
```

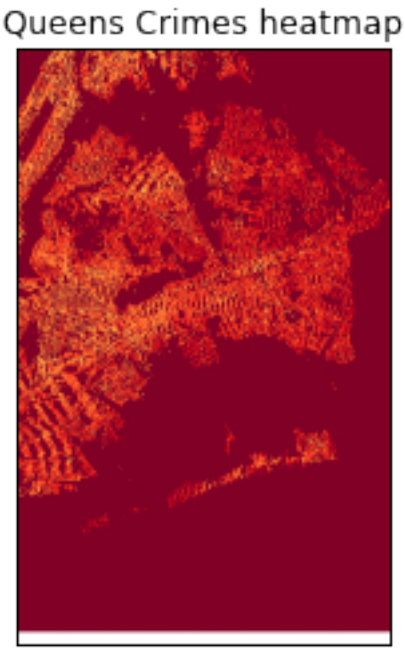



```
In [12]: def heatmap_bk(df):
             """
             The function turns input dataframe into heatmap. Input should contain feature
             'LATITUDE' and 'LONGITUDE', otherwise it will not be plotted.
             """

             a = np.array(df['Latitude'].dropna())
             b = np.array(df['Longitude'].dropna())
             N = len(a)
             west, south, east, north = -74.06, 40.53, -73.81, 40.76 # BK
             fig = plt.figure()
             m = Basemap(projection='merc', llcrnrlat=south, urcrnrlat=north,
                         llcrnrlon=west, urcrnrlon=east, lat_ts=south, resolution='i')
             x, y = m(b, a)
             m.hexbin(x, y, gridsize=650, bins='log', cmap=cm.YlOrRd_r)
             plt.title("Brooklyn Crimes heatmap")
             plt.show()
```

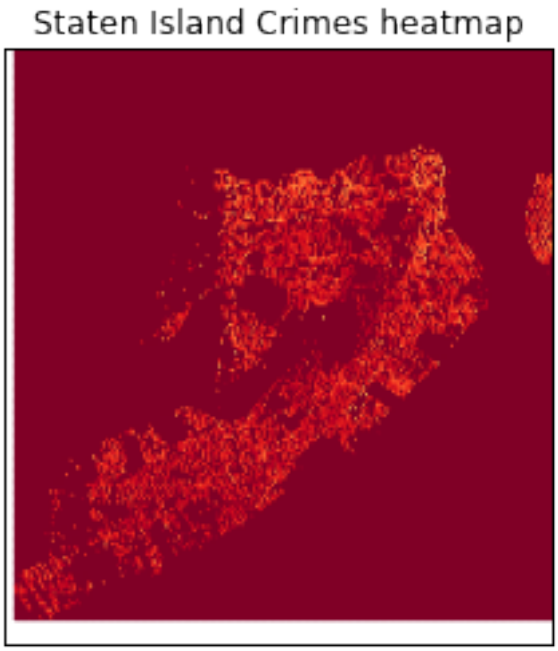
```
heatmap_bk(df)
```



In [13]:
```
def heatmap_q(df):
    """
    The function turns input dataframe into heatmap. Input should contain feature
    'LATITUDE' and 'LONGITUDE', otherwise it will not be plotted.
    """
    west, south, east, north = -73.98, 40.49, -73.70, 40.83 # QS
    a = np.array(df['Latitude'].dropna())
    b = np.array(df['Longitude'].dropna())
    N = len(a)

    fig = plt.figure()
    m = Basemap(projection='merc', llcrnrlat=south, urcrnrlat=north,
                llcrnrlon=west, urcrnrlon=east, lat_ts=south, resolution='i')
    x, y = m(b, a)
    m.hexbin(x, y, gridsize=650, bins='log', cmap=cm.YlOrRd_r)
    plt.title("Queens Crimes heatmap")
    plt.show()

heatmap_q(df)
```




```
In [14]: def heatmap_si(df):
             """
             The function turns input dataframe into heatmap. Input should contain feature
             'LATITUDE' and 'LONGITUDE', otherwise it will not be plotted.
             """
             west, south, east, north = -74.26, 40.49, -74.03, 40.68 # SI
             a = np.array(df['Latitude'].dropna())
             b = np.array(df['Longitude'].dropna())
             N = len(a)

             fig = plt.figure()
             m = Basemap(projection='merc', llcrnrlat=south, urcrnrlat=north,
                         llcrnrlon=west, urcrnrlon=east, lat_ts=south, resolution='i')
             x, y = m(b, a)
             m.hexbin(x, y, gridsize=650, bins='log', cmap=cm.YlOrRd_r)
             plt.title("Staten Island Crimes heatmap")
             plt.show()

         heatmap_si(df)
```




```
In [5]: def heatmap_br(df):
            """
            The function turns input dataframe into heatmap. Input should contain feature
            'LATITUDE' and 'LONGITUDE', otherwise it will not be plotted.
            """
            west, south, east, north = -74.05, 40.56, -73.83, 40.74 # SI
            a = np.array(df['Latitude'].dropna())
            b = np.array(df['Longitude'].dropna())
            N = len(a)

            fig = plt.figure()
            m = Basemap(projection='merc', llcrnrlat=south, urcrnrlat=north,
                        llcrnrlon=west, urcrnrlon=east, lat_ts=south, resolution='i')
            x, y = m(b, a)
            m.hexbin(x, y, gridsize=650, bins='log', cmap=cm.YlOrRd_r)
            plt.title("Bronx Crimes heatmap")
            plt.show()

        heatmap_br(df)
```

```
/Users/xinyan/anaconda/lib/python3.5/site-packages/mpl_toolkits/basemap/__init__.py:3459: Matplo
  b = ax.ishold()
/Users/xinyan/anaconda/lib/python3.5/site-packages/mpl_toolkits/basemap/__init__.py:3472: Matplo
    See the API Changes document (http://matplotlib.org/api/api_changes.html)
    for more details.
```

```
ax.hold(b)
```