311_Service_Requests_from_2010_to_Present

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

In [2]:
```
df_311 = pd.read_csv('311_Service_Requests_from_2010_to_Present.csv')
```

```
C:\Users\SujitSonar\Anaconda3\lib\site-packages\IPython\core\interact
iveshell.py:3049: DtypeWarning: Columns (48,49) have mixed types. Spe
cify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [3]:
```
df_311.head(2)
```

Out[3]:

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location T
0	32310363	12/31/2015 11:59:45 PM	01-01-16 0:55	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidev
1	32309934	12/31/2015 11:59:44 PM	01-01-16 1:26	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sidev

2 rows × 53 columns

basic data check

In [4]:
```
df_311.shape
```

Out[4]: (300698, 53)

In [5]:
```
df_311.columns
```

Out[5]:
```
Index(['Unique Key', 'Created Date', 'Closed Date', 'Agency', 'Agency
Name',
       'Complaint Type', 'Descriptor', 'Location Type', 'Incident Zip
',
       'Incident Address', 'Street Name', 'Cross Street 1', 'Cross St
reet 2',
       'Intersection Street 1', 'Intersection Street 2', 'Address Typ
e',
       'City', 'Landmark', 'Facility Type', 'Status', 'Due Date',
       'Resolution Description', 'Resolution Action Updated Date',
       'Community Board', 'Borough', 'X Coordinate (State Plane)',
       'Y Coordinate (State Plane)', 'Park Facility Name', 'Park Boro
ugh',
       'School Name', 'School Number', 'School Region', 'School Code
',
       'School Phone Number', 'School Address', 'School City', 'Schoo
l State',
       'School Zip', 'School Not Found', 'School or Citywide Complain
t',
       'Vehicle Type', 'Taxi Company Borough', 'Taxi Pick Up Location
',
       'Bridge Highway Name', 'Bridge Highway Direction', 'Road Ramp
',
       'Bridge Highway Segment', 'Garage Lot Name', 'Ferry Direction
',
       'Ferry Terminal Name', 'Latitude', 'Longitude', 'Location'],
      dtype='object')
```

In [6]:
```
#checking Missing values
record_count=df_311.isnull().sum().sort_values(ascending=False)
record_count=pd.DataFrame(record_count)
record_count.rename(columns={0:'record count'}, inplace=True)
record_count['%']=(record_count/len(df_311))
record_count.style.format({'%': "{:.2%}"})
```

Out[6]:

	record count	%
Taxi Company Borough	300698	100.00%
Taxi Pick Up Location	300698	100.00%
School or Citywide Complaint	300698	100.00%
Garage Lot Name	300698	100.00%
Vehicle Type	300698	100.00%
Ferry Direction	300697	100.00%
Ferry Terminal Name	300696	100.00%
Bridge Highway Segment	300485	99.93%
Road Ramp	300485	99.93%
Bridge Highway Name	300455	99.92%
Bridge Highway Direction	300455	99.92%
Landmark	300349	99.88%
Intersection Street 2	257336	85.58%
Intersection Street 1	256840	85.41%
Cross Street 2	49779	16.55%
Cross Street 1	49279	16.39%
Incident Address	44410	14.77%
Street Name	44410	14.77%
Descriptor	5914	1.97%
Location	3540	1.18%
Longitude	3540	1.18%
X Coordinate (State Plane)	3540	1.18%
Y Coordinate (State Plane)	3540	1.18%
Latitude	3540	1.18%
Address Type	2815	0.94%
Incident Zip	2615	0.87%
City	2614	0.87%
Resolution Action Updated Date	2187	0.73%
Facility Type	2171	0.72%
Closed Date	2164	0.72%

	record count	%
Location Type	131	0.04%
Due Date	3	0.00%
School Code	1	0.00%
School Region	1	0.00%
School Zip	1	0.00%
School Name	0	0.00%
Complaint Type	0	0.00%
Agency Name	0	0.00%
Agency	0	0.00%
Created Date	0	0.00%
School Not Found	0	0.00%
School Phone Number	0	0.00%
School State	0	0.00%
School City	0	0.00%
School Address	0	0.00%
Park Borough	0	0.00%
Status	0	0.00%

verifying columns and checking missing value columns to decide whether to keep or drop the columns

In [7]:
```
 # based on the above missing data, we can drop the below columns
# Taxi Company Borough               300698
# Taxi Pick Up Location              300698
# School or Citywide Complaint       300698
# Garage Lot Name                    300698
# Vehicle Type                       300698
# Ferry Direction                    300697
# Ferry Terminal Name                300696
# Bridge Highway Segment             300485
# Road Ramp                          300485
# Bridge Highway Name                300455
# Bridge Highway Direction           300455
# Landmark                           300349
# Intersection Street 2              257336
# Intersection Street 1              256840
```

In [8]:
```
# checking values under each of these columns to check if we can drop t
hese columns
#df_311['Address Type'].unique()   #drop
#df_311['Facility Type'].unique() #drop
#df_311['City'].unique() keep
#df_311['Location'] keep
#df_311['Descriptor'].unique() keep
#check=df_311[['Complaint Type','Descriptor']]
#check.drop_duplicates().sort_values(by='Complaint Type')

#df_311['Incident Address']
#df_311['Cross Street 2'].unique() #drop
#df_311['Cross Street 1'].unique() #drop
#df_311['Street Name']
```

In [9]:
```
df_311.describe(include='object').transpose()
```

Out[9]:

	count	unique	top	freq
Created Date	300698	259493	07-11-15 23:04	9
Closed Date	298534	237165	11-08-15 7:34	24
Agency	300698	1	NYPD	300698
Agency Name	300698	3	New York City Police Department	300690
Complaint Type	300698	24	Blocked Driveway	77044
Descriptor	294784	45	Loud Music/Party	61430
Location Type	300567	18	Street/Sidewalk	249299
Incident Address	256288	107652	1207 BEACH AVENUE	904
Street Name	256288	7320	BROADWAY	3237
Cross Street 1	251419	5982	BROADWAY	4338
Cross Street 2	250919	5823	BEND	4391
Intersection Street 1	43858	4413	BROADWAY	672
Intersection Street 2	43362	4172	BROADWAY	1358
Address Type	297883	5	ADDRESS	238644
City	298084	53	BROOKLYN	98307
Landmark	349	116	CENTRAL PARK	67
Facility Type	298527	1	Precinct	298527
Status	300698	4	Closed	298471
Due Date	300695	259851	07-12-15 7:04	9
Resolution Description	300698	18	The Police Department responded to the complai...	90490
Resolution Action Updated Date	298511	237895	11-08-15 7:34	24
Community Board	300698	75	12 MANHATTAN	12390
Borough	300698	6	BROOKLYN	98307
Park Facility Name	300698	2	Unspecified	300697
Park Borough	300698	6	BROOKLYN	98307
School Name	300698	2	Unspecified	300697
School Number	300698	2	Unspecified	300697
School Region	300697	1	Unspecified	300697
School Code	300697	1	Unspecified	300697
School Phone Number	300698	2	Unspecified	300697
School Address	300698	2	Unspecified	300697
School City	300698	2	Unspecified	300697

	count	unique	top	freq
School State	300698	2	Unspecified	300697
School Zip	300697	1	Unspecified	300697
School Not Found	300698	1	N	300698
Bridge Highway Name	243	29	FDR Dr	33
Bridge Highway Direction	243	34	East/Queens Bound	21
Road Ramp	213	2	Roadway	162
Bridge Highway Segment	213	160	East 96th St (Exit 14) - Triborough Br (Exit 17)	6

checking date fields in the data set

In [10]:
```
df_311[['Due Date','Created Date','Closed Date','Resolution Action Updated Date']].head(2)
```

Out[10]:

	Due Date	Created Date	Closed Date	Resolution Action Updated Date
0	01-01-16 7:59	12/31/2015 11:59:45 PM	01-01-16 0:55	01-01-16 0:55
1	01-01-16 7:59	12/31/2015 11:59:44 PM	01-01-16 1:26	01-01-16 1:26

In [11]:
```
df_311[['Due Date','Created Date','Closed Date','Resolution Action Updated Date']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300698 entries, 0 to 300697
Data columns (total 4 columns):
Due Date                          300695 non-null object
Created Date                      300698 non-null object
Closed Date                       298534 non-null object
Resolution Action Updated Date    298511 non-null object
dtypes: object(4)
memory usage: 9.2+ MB
```

Observations 1:

- date fields are in Object/string format. we need to apply datetime methods to convert it to correct date time format
- The data is only for the year 2015
- there are 14 columsn with almost >80% no data, we can drop those columns

Columns	Number of NAN
Taxi Company Borough	300698
Taxi Pick Up Location	300698
School or Citywide Complaint	300698
Garage Lot Name	300698
Vehicle Type	300698
Ferry Direction	300697
Ferry Terminal Name	300696
Bridge Highway Segment	300485
Road Ramp	300485
Bridge Highway Name	300455
Bridge Highway Direction	300455
Landmark	300349
Intersection Street 2	257336
Intersection Street 1	256840

- also, after looking at the data we can also drop these columns as, we do not have much information to use for the analysis

Cross Street 2	49779
Cross Street 1	49279
Facility Type	2171
Address Type	2815

- checking the content of the fields, we also see that the below fields in the dataset has no meaningful information and can be dropped

fields	count	unique	top	freq
Park Facility Name	300698	2	Unspecified	300697
School Name	300698	2	Unspecified	300697
School Number	300698	2	Unspecified	300697
School Region	300697	1	Unspecified	300697
School Code	300697	1	Unspecified	300697
School Phone Number	300698	2	Unspecified	300697
School Address	300698	2	Unspecified	300697
School City	300698	2	Unspecified	300697
School State	300698	2	Unspecified	300697
School Zip	300697	1	Unspecified	300697
School Not Found	300698	1	N	300698

to Analyze:

- Import a 311 NYC service request.
- Read or convert the columns ‘Created Date’ and Closed Date’ to datetime datatype and create a new column ‘Request_Closing_Time’ as the time elapsed between request creation and request closing. (Hint: Explore the package/module datetime)
- Provide major insights/patterns that you can offer in a visual format (graphs or tables); at least 4 major conclusions that you can come up with after generic data mining.
- Order the complaint types based on the average ‘Request_Closing_Time’, grouping them for different locations.
- Perform a statistical test for the following:

Please note: For the below statements you need to state the Null and Alternate and then provide a statistical test to accept or reject the Null Hypothesis along with the corresponding ‘p-value’.

```
Whether the average response time across complaint types is similar or not
(overall)
Are the type of complaint or service requested and location related?
```

Data Preparation & Approach

Looking at the above set of details to analyse, we need to first prep the given data

- The date fields provided in the data set are in object/string format,which needs to be formatted into correct datetime format to be able to aplply datetime functionsand compute the time elapsed between request creation and request closing
- We can use groupby methods and other aggretaion functions to summarise the data and create charts to analyse
  - Avg number of complaint by types, locations,
  - Top 5 complaint types
  - Avg time elaspsed in closing the coplaints
  - trends of complaints by Month
  - complaint composition by Types and Status
  - Summary table of complaint Types by location, showing avg closting time and avg number of complaints by type
- statistical test - Creating Null and alternate hypothese
- corelation between complaint or service requested and location

Reading the data and parsing the date to correct format

In [12]:
```
date_fields = ['Created Date', 'Closed Date', 'Due Date']

csr_df = pd.read_csv('311_Service_Requests_from_2010_to_Present.csv',pa
rse_dates = date_fields,
                     infer_datetime_format = True)
```

```
C:\Users\SujitSonar\Anaconda3\lib\site-packages\IPython\core\interact
iveshell.py:3049: DtypeWarning: Columns (48,49) have mixed types. Spe
cify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [13]:
```
csr_df[['Due Date','Created Date','Closed Date']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300698 entries, 0 to 300697
Data columns (total 3 columns):
Due Date        300695 non-null datetime64[ns]
Created Date    300698 non-null datetime64[ns]
Closed Date     298534 non-null datetime64[ns]
dtypes: datetime64[ns](3)
memory usage: 6.9 MB
```

In [14]:
```
# we see the the three date columns are now converetd to correct date t
ime format
```

In [15]:
```
csr_df.shape
```

Out[15]:
```
(300698, 53)
```

In [16]:
```
#Droping unwanted columns for analysis
csr_df = csr_df.drop(['Taxi Company Borough','Taxi Pick Up Location','S
chool or Citywide Complaint',
                      'Garage Lot Name','Vehicle Type','Ferry Direct
ion','Bridge Highway Segment',
                      'Road Ramp','Bridge Highway Name','Bridge High
way Direction','Landmark',
                       'Intersection Street 2','Intersection Street
1','Cross Street 2',
                      'Cross Street 1','Facility Type','Address Type
','Ferry Terminal Name',
                      'Park Facility Name','School Name','School Num
ber','School Region',
                      'School Code','School Phone Number','School Ad
dress','School City',
                      'School State','School Zip','School Not Found
'],axis=1)
```

In [17]:
```
csr_df.shape
```

Out[17]:
```
(300698, 24)
```

In [18]:
```
# Total number of columns is now 24
```

In [19]:
```
#Checking if any duplicate rows based on all columns
print(csr_df['Unique Key'].duplicated().unique())
print(len(csr_df))
dup_df = csr_df[csr_df.duplicated()]
dup_df
```

```
[False]
300698
```

Out[19]:

Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Type	Incident Zip	Incide Addre

0 rows × 24 columns

Observations 2:

- the date date fields are correctly formatted to datetime format
- unwanted columns are dropped and we see 3000698 records and 24 columns
- no duplicate entry based on all columns
- There are total of 300698 unique complaints in the dataset

Data Preparation

In [20]:
```
# adding helper column for total complaint counts
csr_df['request_count'] = 1
csr_df.head(2)
```

Out[20]:

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Locatior
0	32310363	2015-12-31 23:59:45	2016-01-01 00:55:00	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sic
1	32309934	2015-12-31 23:59:44	2016-01-01 01:26:00	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sic

2 rows × 25 columns

In [21]:
```
csr_df['request_count'].sum()
```

Out[21]: 300698

In [22]:
```
# Further Droping unwanted columns which will not be used for analysis
csr_df = csr_df.drop(['Incident Address','Street Name','Y Coordinate (State Plane)',
                      'X Coordinate (State Plane)','Incident Zip','Resolution Action Updated Date'],axis=1)
```

In [23]:
```
csr_df.shape
```

Out[23]:
```
(300698, 19)
```

In [24]:
```
#handling Missing values
csr_df.isnull().sum().sort_values(ascending=False)
```

Out[24]:
```
Descriptor                5914
Longitude                 3540
Latitude                  3540
Location                  3540
City                      2614
Closed Date               2164
Location Type              131
Due Date                     3
request_count                0
Complaint Type               0
Created Date                 0
Agency                       0
Agency Name                  0
Status                       0
Resolution Description       0
Community Board              0
Borough                      0
Park Borough                 0
Unique Key                   0
dtype: int64
```

In [25]:
```
csr_df.describe(include='object')
```

Out[25]:

	Agency	Agency Name	Complaint Type	Descriptor	Location Type	City	Status	Resolution Desc
count	300698	300698	300698	294784	300567	298084	300698	
unique	1	3	24	45	18	53	4	
top	NYPD	New York City Police Department	Blocked Driveway	Loud Music/Party	Street/Sidewalk	BROOKLYN	Closed	The Department responds co
freq	300698	300690	77044	61430	249299	98307	298471	

Observations 3:

- There are some nul values in closed date and those is becasue the status for thoe complaints/request is still either Open, Assigned or draft, so we will keep these closed date entries as NAN
- other NANs we will handle as we do groupby to summarise and either replace it with freq or mean ccordingly

In [ ]:

Q1.Read or convert the columns 'Created Date' and Closed Date' to datetime datatype and create a new column 'Request_Closing_Time' as the time elapsed between request creation and request closing. (Hint: Explore the package/module datetime)

In [26]:
```
import datetime
```

In [27]:
```
# timedelat column for request create ddate and Closed date
csr_df['Request_Closing_Time'] = csr_df['Closed Date']-csr_df['Created Date']
```

In [28]:
```
csr_df.head(2)
```

Out[28]:

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Locatior
0	32310363	2015-12-31 23:59:45	2016-01-01 00:55:00	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sic
1	32309934	2015-12-31 23:59:44	2016-01-01 01:26:00	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sic

In [29]:
```
#mean of time delta
csr_df['Request_Closing_Time'].mean()
```

Out[29]:
```
Timedelta('0 days 04:18:51.832782')
```

In [30]:
```
# Creating two additional columns 'Request_Closing_in_days' and 'Reques
t_Closing_in_hrs' to maniplulate the data
# and apply aggregate functions while doing further analysis

csr_df['Request_Closing_in_days'] = (csr_df['Closed Date']-csr_df['Crea
ted Date']).dt.days
csr_df['Request_Closing_in_hrs'] = (csr_df['Closed Date']-csr_df['Creat
ed Date'])/pd.Timedelta('1 hour')
csr_df.head(2)
```

Out[30]:

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Locatior
0	32310363	2015-12-31 23:59:45	2016-01-01 00:55:00	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sic
1	32309934	2015-12-31 23:59:44	2016-01-01 01:26:00	NYPD	New York City Police Department	Blocked Driveway	No Access	Street/Sic

2 rows × 22 columns

In [31]:
```
csr_df.describe()
```

Out[31]:

	Unique Key	Latitude	Longitude	request_count	Request_Closing_Time	Re
count	3.006980e+05	297158.000000	297158.000000	300698.0	298534	
mean	3.130054e+07	40.725885	-73.925630	1.0	0 days 04:18:51.832782	
std	5.738547e+05	0.082012	0.078454	0.0	0 days 06:05:22.141833	
min	3.027948e+07	40.499135	-74.254937	1.0	0 days 00:01:00	
25%	3.080118e+07	40.669796	-73.972142	1.0	0 days 01:16:33	
50%	3.130436e+07	40.718661	-73.931781	1.0	0 days 02:42:55.500000	
75%	3.178446e+07	40.781840	-73.876805	1.0	0 days 05:21:00	
max	3.231065e+07	40.912869	-73.700760	1.0	24 days 16:52:22	

In [32]:
```
print(f"mean of Requests_Closing_in_day is: {csr_df.Request_Closing_in_
days.mean()}")
print(f"mean of Requests_Closing_in_hrs is: {csr_df.Request_Closing_in_
hrs.mean()}")
```

```
mean of Requests_Closing_in_day is: 0.016041723890746113
mean of Requests_Closing_in_hrs is: 4.314397995240427
```

In [33]:
```
csr_df['Request_Closing_Time'].value_counts(dropna=False).sort_values(ascending=False).head()
```

Out[33]:
```
NaT         2164
00:36:00     492
00:44:00     473
00:51:00     472
00:53:00     468
Name: Request_Closing_Time, dtype: int64
```

In [34]:
```
csr_df['Request_Closing_Time'].max()
```

Out[34]:
```
Timedelta('24 days 16:52:22')
```

In [35]:
```
csr_df['Request_Closing_Time_Bin'] = pd.cut(csr_df['Request_Closing_in_
hrs'], 100)
csr_df['Request_Closing_Time_Bin'].value_counts(dropna = False)
```

Out[35]:

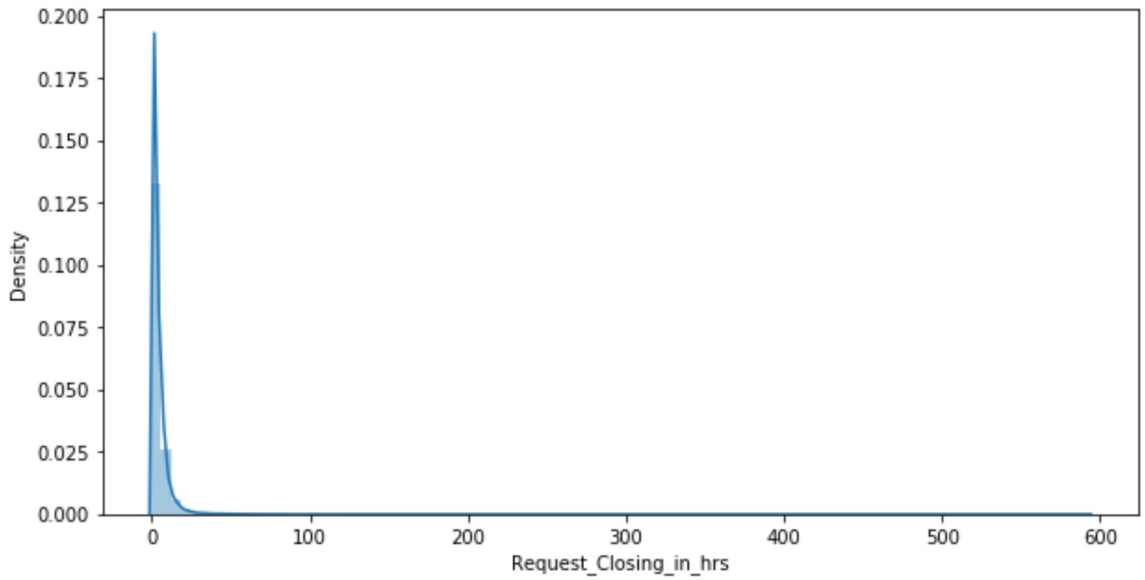
(-0.576, 5.945]	234336
(5.945, 11.874]	46387
(11.874, 17.802]	10535
(17.802, 23.731]	3502
NaN	2164
(23.731, 29.659]	1494
(29.659, 35.588]	832
(35.588, 41.517]	451
(41.517, 47.445]	319
(47.445, 53.374]	221
(53.374, 59.302]	110
(59.302, 65.231]	72
(65.231, 71.159]	57
(71.159, 77.088]	53
(77.088, 83.017]	38
(124.516, 130.445]	21
(83.017, 88.945]	18
(88.945, 94.874]	16
(94.874, 100.802]	16
(118.588, 124.516]	11
(142.302, 148.231]	8
(106.731, 112.659]	7
(100.802, 106.731]	6
(160.088, 166.016]	4
(112.659, 118.588]	4
(136.374, 142.302]	3
(195.659, 201.588]	2
(219.373, 225.302]	2
(148.231, 154.159]	2
(166.016, 171.945]	1
...	
(521.73, 527.659]	0
(527.659, 533.587]	0
(533.587, 539.516]	0
(539.516, 545.444]	0
(545.444, 551.373]	0
(551.373, 557.301]	0
(426.873, 432.802]	0
(420.945, 426.873]	0
(415.016, 420.945]	0
(409.087, 415.016]	0
(284.588, 290.516]	0
(290.516, 296.445]	0
(302.373, 308.302]	0
(308.302, 314.23]	0
(314.23, 320.159]	0
(320.159, 326.088]	0
(326.088, 332.016]	0
(337.945, 343.873]	0
(343.873, 349.802]	0
(349.802, 355.73]	0
(355.73, 361.659]	0
(361.659, 367.587]	0
(367.587, 373.516]	0
(373.516, 379.445]	0
(379.445, 385.373]	0

```
(385.373, 391.302]          0
(391.302, 397.23]           0
(397.23, 403.159]           0
(403.159, 409.087]          0
(296.445, 302.373]          0
Name: Request_Closing_Time_Bin, Length: 101, dtype: int64
```

In [36]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [37]:
```
f, ax = plt.subplots(figsize=(10,5))
sns.distplot(csr_df['Request_Closing_in_hrs'], bins=100)
plt.show()
```

```
C:\Users\SujitSonar\Anaconda3\lib\site-packages\seaborn\distribution
s.py:2557: FutureWarning: `distplot` is a deprecated function and wil
l be removed in a future version. Please adapt your code to use eithe
r `displot` (a figure-level function with similar flexibility) or `hi
stplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [38]:
```
234336/len(csr_df)
```

Out[38]:
```
0.779306812815516
```

In [39]:
```
csr_df.describe()
```

Out[39]:

	Unique Key	Latitude	Longitude	request_count	Request_Closing_Time	Re
count	3.006980e+05	297158.000000	297158.000000	300698.0	298534	
mean	3.130054e+07	40.725885	-73.925630	1.0	0 days 04:18:51.832782	
std	5.738547e+05	0.082012	0.078454	0.0	0 days 06:05:22.141833	
min	3.027948e+07	40.499135	-74.254937	1.0	0 days 00:01:00	
25%	3.080118e+07	40.669796	-73.972142	1.0	0 days 01:16:33	
50%	3.130436e+07	40.718661	-73.931781	1.0	0 days 02:42:55.500000	
75%	3.178446e+07	40.781840	-73.876805	1.0	0 days 05:21:00	
max	3.231065e+07	40.912869	-73.700760	1.0	24 days 16:52:22	

In [ ]:
```
```

In [ ]:
```
```

Observations 4:

- we see that majority of the average request/complaints closing time is within 4 hours of it is being created
- from the binning table, we see that around 78% of the requests/complaints are closed within 0 to 5.9 hrs, which is close to the average closing time
- 0 days 04:18:51.832782 is the average request closing time

Q2 Provide major insights/patterns that you can offer in a visual format (graphs or tables); at least 4 major conclusions that you can come up with after generic data mining

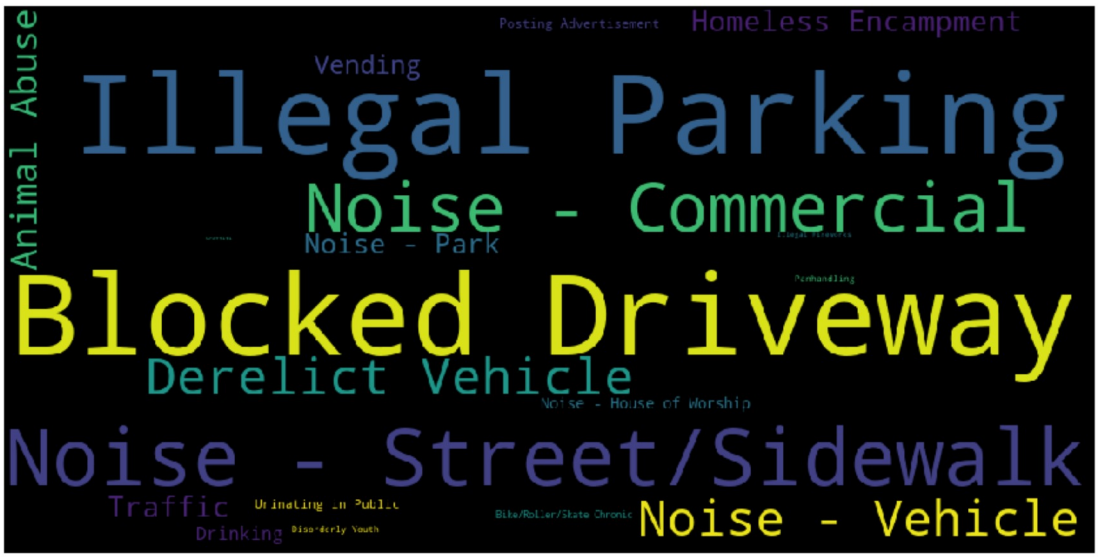
In [40]:
```
csr_df['Complaint Type'].isnull().sum()
```

Out[40]: 0

In [41]:
```
# finding the frequency of the words appear in the Customer Complaints Column and visualing using wordcloud
from wordcloud import WordCloud, STOPWORDS
```

In [42]:
```
complaints_list= csr_df['Complaint Type']
```

In [43]:
```
from collections import Counter
word_could_dict=Counter(complaints_list)
wordcloud = WordCloud(width = 1000, height = 500).generate_from_frequen
cies(word_could_dict)

plt.figure(figsize=(15,8))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```



In [44]:
```
csr_df.columns
```

Out[44]:
```
Index(['Unique Key', 'Created Date', 'Closed Date', 'Agency', 'Agency
Name',
       'Complaint Type', 'Descriptor', 'Location Type', 'City', 'Stat
us',
       'Due Date', 'Resolution Description', 'Community Board', 'Boro
ugh',
       'Park Borough', 'Latitude', 'Longitude', 'Location', 'request_
count',
       'Request_Closing_Time', 'Request_Closing_in_days',
       'Request_Closing_in_hrs', 'Request_Closing_Time_Bin'],
      dtype='object')
```

In [ ]:

In [45]:
```
complaints_df=csr_df.groupby(['Complaint Type']).agg({'request_count':'
sum','Request_Closing_in_days':'mean',
                                                      'Request_Closing_in_hrs':'mean
'}).reset_index()

complaints_df.rename(columns={'Complaint Type':'Complaint_Type',
                              'Request_Closing_in_days':'avg_Request_Cl
osing_in_days',
                              'Request_Closing_in_hrs':'avg_Request_Clo
sing_in_hrs'}, inplace = True)



complaints_df.sort_values('request_count', ascending=False).style.hide_
index()
```

Out[45]:

Complaint_Type	request_count	avg_Request_Closing_in_days	avg_Request_Closing_in_hrs
Blocked Driveway	77044	0.0148418	4.74091
Illegal Parking	75361	0.0149198	4.50115
Noise - Street/Sidewalk	48612	0.00931858	3.44522
Noise - Commercial	35577	0.00649701	3.14715
Derelict Vehicle	17718	0.0739709	7.36414
Noise - Vehicle	17083	0.00886514	3.58899
Animal Abuse	7778	0.0244593	5.21324
Traffic	4498	0.0104537	3.44868
Homeless Encampment	4416	0.011096	4.36557
Noise - Park	4042	0.00646445	3.41073
Vending	3802	0.00869565	4.01392
Drinking	1280	0.0101961	3.86183
Noise - House of Worship	931	0.00968784	3.1933
Posting Advertisement	650	0.00154321	1.9758
Urinating in Public	592	0.00844595	3.62666
Bike/Roller/Skate Chronic	427	0.00943396	3.76646
Panhandling	307	0.0262295	4.37277
Disorderly Youth	286	0.0034965	3.55858
Illegal Fireworks	168	0.00595238	2.76114
Graffiti	113	0.0619469	7.15125
Agency Issues	6	0	5.26032
Squeegee	4	0	4.04563
Ferry Complaint	2	nan	nan
Animal in a Park	1	14	336.835

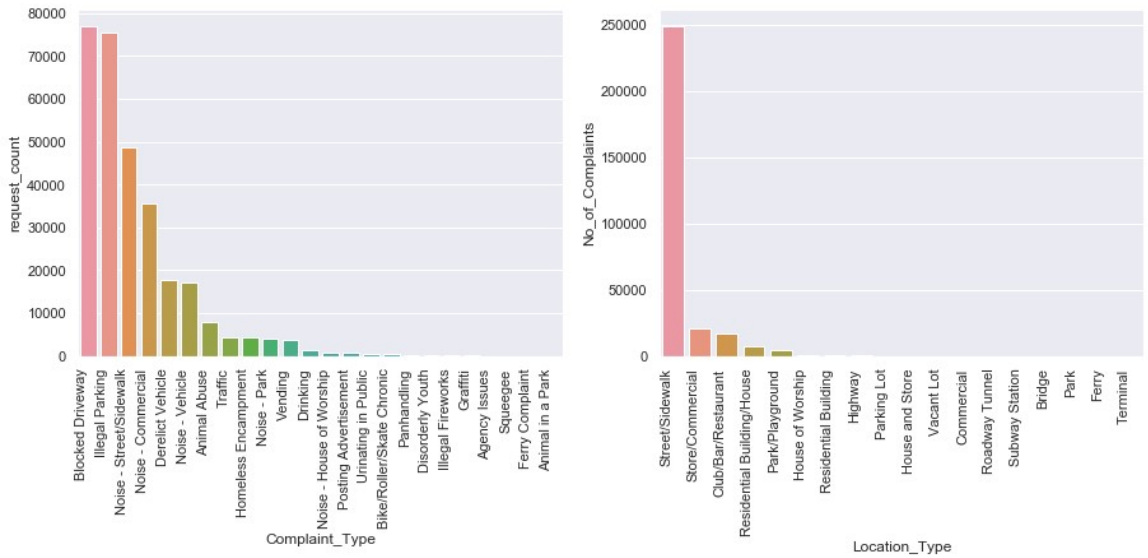
Observations:

- Looking at the Wordcloud and the above table, Blocked Driveway and Illegal Praking are the two top most type of complaints received by NYC311

In [46]:
```
# No of complaints by Location Type
df_location_Type=csr_df.groupby(['Location Type'])[['Unique Key']].coun
t().reset_index()
df_location_Type.rename(columns={'Location Type':'Location_Type','Uniqu
e Key':'No_of_Complaints'},inplace=True)
```

In [47]:
```
sns.set() # Setting seaborn as default style even if use only matplotli
b
```

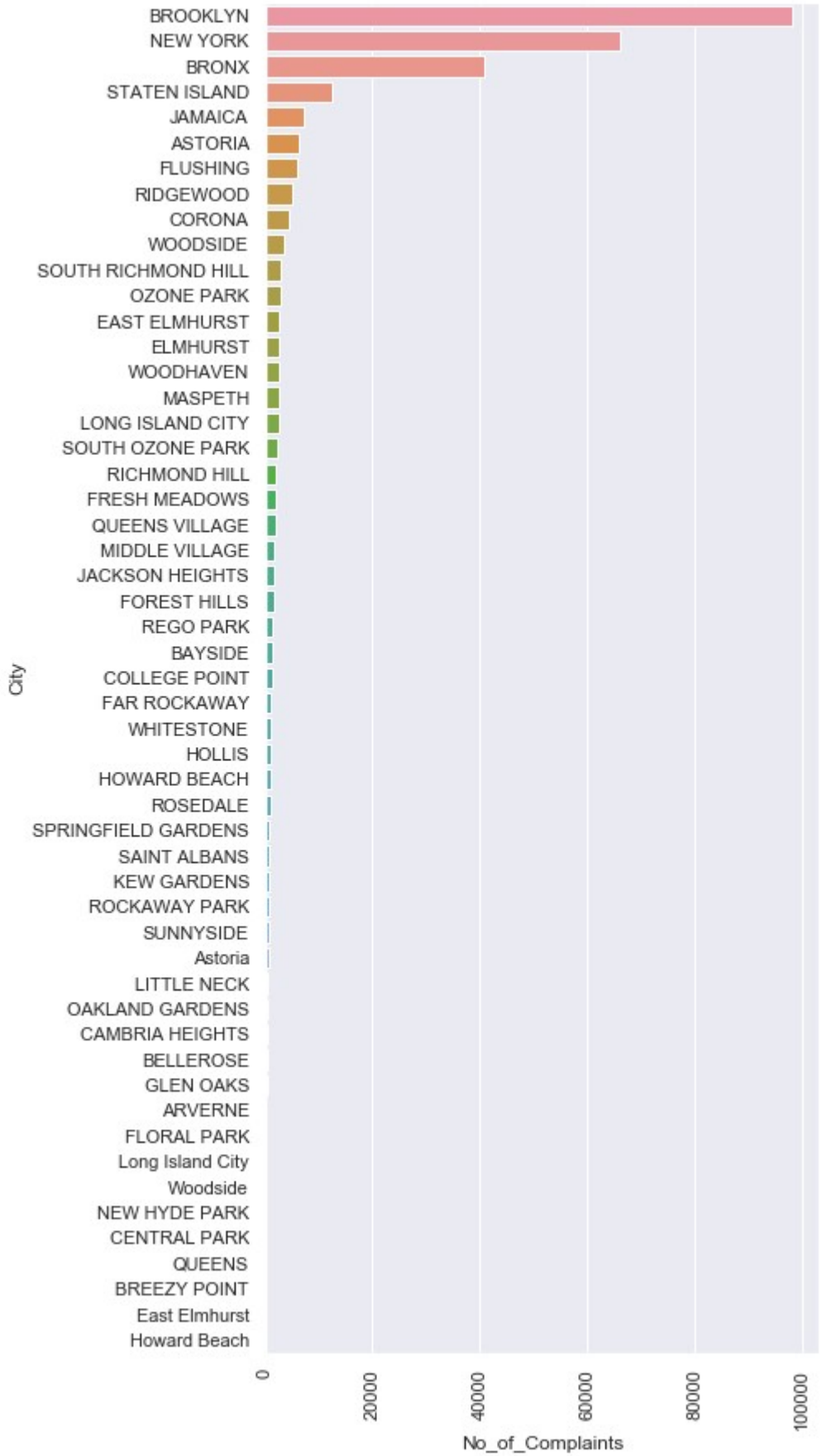
In [48]:
```
fig, axs = plt.subplots(ncols=2, figsize=(15,5))

ax0=sns.barplot(x='Complaint_Type',y='request_count',data = complaints_
df,
            order=complaints_df.sort_values('request_count',ascending =
False).Complaint_Type,ax=axs[0])
ax0.set_xticklabels(ax0.get_xticklabels(), rotation=90, ha="right")


ax1=sns.barplot(x='Location_Type',y='No_of_Complaints',data = df_locati
on_Type,
            order=df_location_Type.sort_values('No_of_Complaints',ascend
ing = False).Location_Type,ax=axs[1])
ax1.set_xticklabels(ax1.get_xticklabels(), rotation=90, ha="right")

plt.show()
```



Observations:

- At a very high level without any statistical proof, we can see from the above two charts that the type of complaints like the top three complaint types, 'bloked driveway', 'illegal parking', 'noise, stret/sidewalk', do occur mostly in 'street/sidewalk' location type. We will try to check this in our statistical testing.

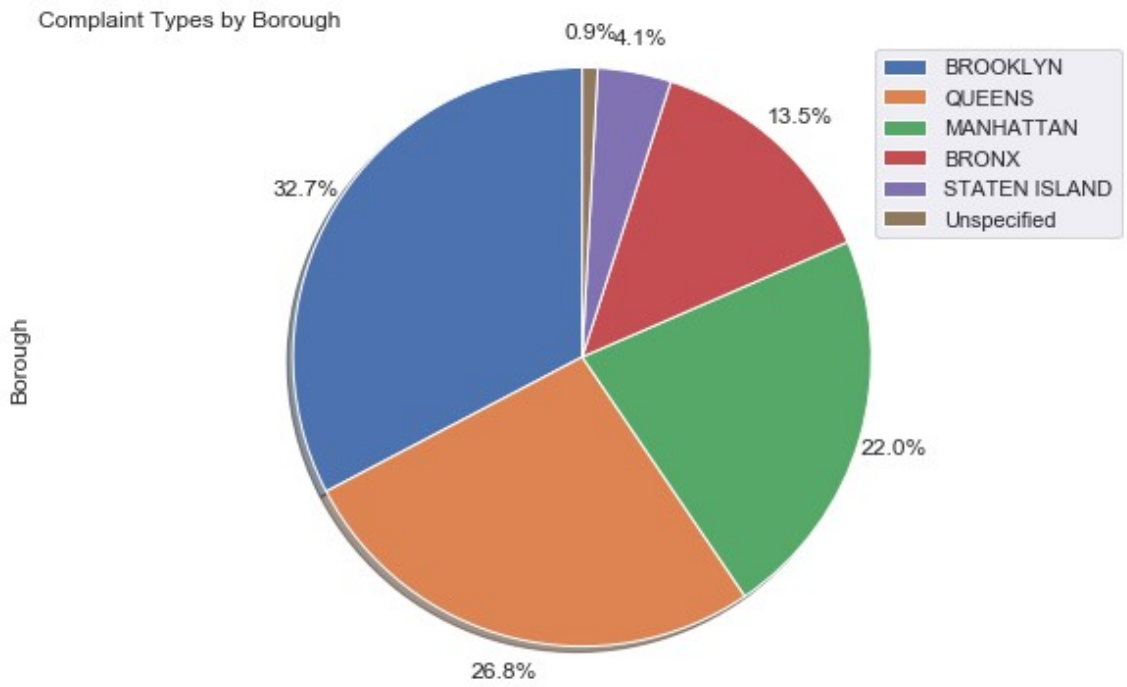
In [49]:
```
df_City=csr_df.groupby(['City'])[['Unique Key']].count().reset_index()
df_City.rename(columns={'Unique Key':'No_of_Complaints'},inplace=True)
df_City
#df_3= df_3.head(10)

f, ax = plt.subplots(figsize=(6, 15))
#CT = list(df_1['Complaint Type'])
sns.barplot(y='City',x='No_of_Complaints',data = df_City,
            order=df_City.sort_values('No_of_Complaints',ascending = Fal
se).City)
plt.xticks(rotation=90)
plt.show()
```


BROOKLYN
NEW YORK
BRONX
STATEN ISLAND
JAMAICA
ASTORIA
FLUSHING
RIDGEWOOD
CORONA
WOODSIDE
SOUTH RICHMOND HILL
OZONE PARK
EAST ELMHURST
ELMHURST
WOODHAVEN
MASPETH
LONG ISLAND CITY
SOUTH OZONE PARK
RICHMOND HILL
FRESH MEADOWS
QUEENS VILLAGE
MIDDLE VILLAGE
JACKSON HEIGHTS
FOREST HILLS
REGO PARK
BAYSIDE
COLLEGE POINT
FAR ROCKAWAY
WHITESTONE
HOLLIS
HOWARD BEACH
ROSEDALE
SPRINGFIELD GARDENS
SAINT ALBANS
KEW GARDENS
ROCKAWAY PARK
SUNNYSIDE
Astoria
LITTLE NECK
OAKLAND GARDENS
CAMBRIA HEIGHTS
BELLEROSE
GLEN OAKS
ARVERNE
FLORAL PARK
Long Island City
Woodside
NEW HYDE PARK
CENTRAL PARK
QUEENS
BREEZY POINT
East Elmhurst
Howard Beach
City
0
20000
40000
60000
80000
100000
No_of_Complaints

In [52]:
```
df_complaints_by_Month =csr_df[['Unique Key','Created Date','Closed Dat
e','Status','Complaint Type','Location Type','City','Borough']]
df_complaints_by_Month['Request_Created_Month']=df_complaints_by_Month
['Created Date'].dt.strftime('%b')
#df_complaints_by_Month['Request_Closed_Month']=df_complaints_by_Month
['Closed Date'].dt.strftime('%b')
#df_complaints_by_Month
```

```
C:\Users\SujitSonar\Anaconda3\lib\site-packages\ipykernel_launcher.p
y:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas
-docs/stable/indexing.html#indexing-view-versus-copy
```

In [53]:
```
#(df_complaints_by_Month['Status'].value_counts()).plot(kind='pie')

(df_complaints_by_Month['Borough'].value_counts()).plot(kind='pie',
                         figsize =(10,6),
                         autopct='%1.1f%%',
                         startangle=90,
                         shadow=True,
                        labels=None,
                        pctdistance=1.12)

plt.title('Complaint Types by Borough',loc='left')

plt.axis('equal')
plt.legend(labels=(df_complaints_by_Month['Borough'].value_counts()).in
dex, loc='upper right')
plt.show()
```

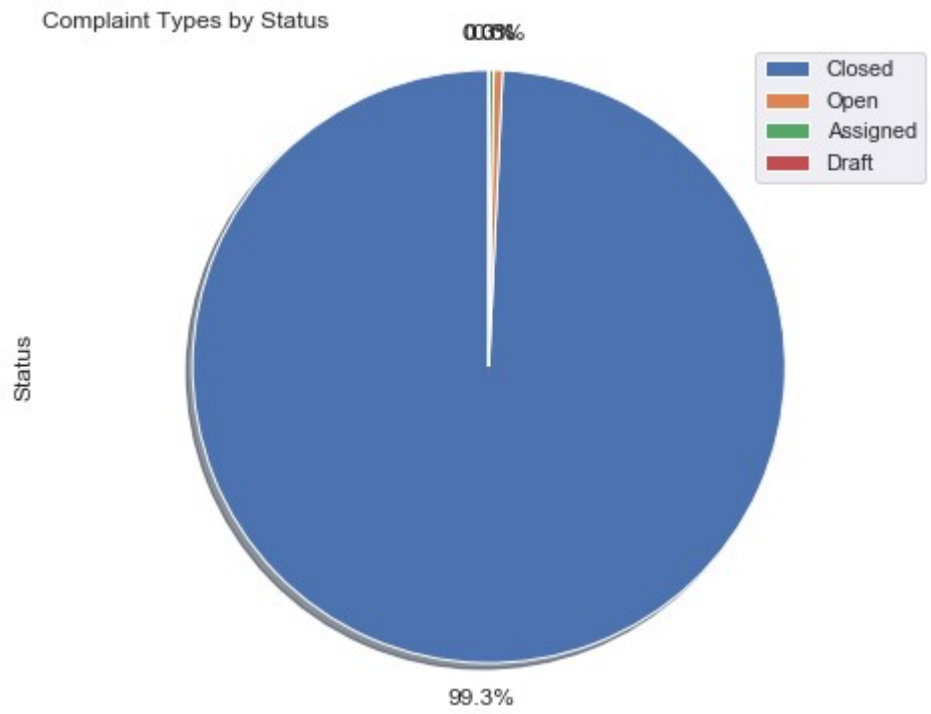


Observations:

- Brooklyn, New york and Bronx are listed as top three cities where complaints/Requests being recorded
- Brooklyn, Queens and Manhatta are the Top three Borough where complaints/Requests being recorded

In [ ]:

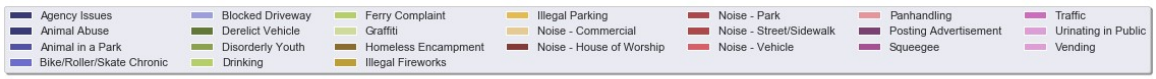
In [54]:
```
#(df_complaints_by_Month['Status'].value_counts()).plot(kind='pie')

(df_complaints_by_Month['Status'].value_counts()).plot(kind='pie',
                          figsize =(8,6),
                          autopct='%1.1f%%',
                          startangle=90,
                          shadow=True,
                         labels=None,
                         pctdistance=1.12)

plt.title('Complaint Types by Status',loc='left')

plt.axis('equal')
plt.legend(labels=(df_complaints_by_Month['Status'].value_counts()).ind
ex, loc='upper right')
plt.show()
```



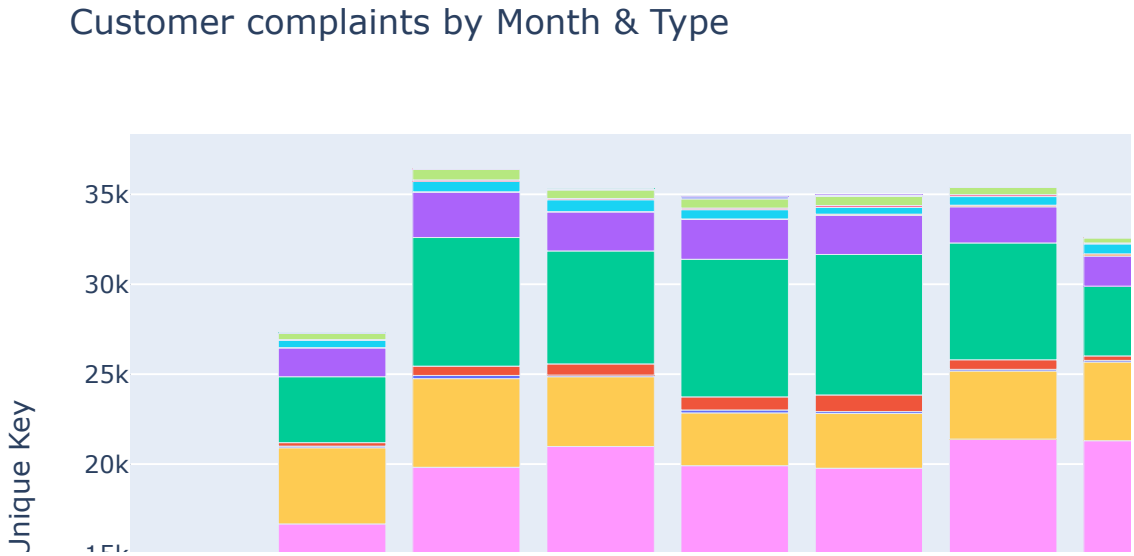
Observations:

Majority of the Complaint/Requests are already closed

In [55]:
```
df_month_1= df_complaints_by_Month.groupby(['Request_Created_Month','Complaint Type'])[['Unique Key']].count().unstack().fillna(0)
df_month_1.head()

df_month_1.columns = df_month_1.columns.droplevel()

df_month_1.reset_index(inplace=True)

df_month_1

#Complaint Types by month

from sorted_months_weekdays import *
from sort_dataframeby_monthorweek import *

df_month_1=Sort_Dataframeby_Month(df=df_month_1,monthcolumnname='Request_Created_Month')
df_month_1.head()

df_month_1.plot(kind='bar', x='Request_Created_Month', stacked=True, figsize=(16,5),colormap='tab20b')
plt.legend(loc='upper center', bbox_to_anchor=(0.5, 1.45),
           fancybox=True, shadow=True, ncol=7)
plt.title('Customer complaints by Month & Type')
plt.ylabel('Requests/Complaints Count')
plt.show()
```



In [56]:
```
df_month= df_complaints_by_Month.groupby(['Request_Created_Month','Complaint Type'])[['Unique Key']].count().reset_index()
df_month.head()

# sorting the monthly_tickets_df in calendar months order
from sorted_months_weekdays import *
from sort_dataframeby_monthorweek import *

df_month=Sort_Dataframeby_Month(df=df_month,monthcolumnname='Request_Created_Month')
df_month.head()

import plotly.express as px

fig = px.bar(df_month, x="Request_Created_Month", y="Unique Key",color='Complaint Type',
             title='Customer complaints by Month & Type')
fig.update_xaxes(tickangle = 0)
fig.show()
```



Observations:

Throughout the year, we see that Blocked Driveway, Illegal parking, Noise-Street/Sidewalk has been reported as the highest complaint types

Plotting the Complaint types in a map

In [57]:
```
df_loc=csr_df[['Unique Key','Complaint Type','City','Location','Longitude','Latitude']]
df_loc.head()
df_loc.dropna(inplace=True) # dropping all nan values
df_loc.isnull().sum()
```

```
C:\Users\SujitSonar\Anaconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
```

Out[57]:
```
Unique Key        0
Complaint Type    0
City              0
Location          0
Longitude         0
Latitude          0
dtype: int64
```

In [58]:
```
df_loc.shape
```

Out[58]:
```
(297117, 6)
```

In [60]:
```
#selecting a 50% sample of the data
df_loc_sample=df_loc.sample(frac=0.5, replace=False, random_state=1)
```

In [61]:
```
df_loc_sample.shape
```

Out[61]:
```
(148558, 6)
```

In [62]:
```
#limit = 100
#df_incidents = df_loc.iloc[0:limit, :]
df_incidents = df_loc_sample
```

In [63]:
```
df_incidents.shape
```

Out[63]:
```
(148558, 6)
```

In [ ]:
```
#df_incidents.dropna(inplace=True)
```

In [64]:
```
df_incidents.isnull().sum()
```

Out[64]:
```
Unique Key         0
Complaint Type     0
City               0
Location           0
Longitude          0
Latitude           0
dtype: int64
```

In [65]:
```
import folium
from folium import plugins

# let's start again with a clean copy of the map of San Francisco
nyc_map=folium.Map(location=[40.730610,-73.935242],zoom_start=7)

# instantiate a mark cluster object for the incidents in the dataframe
incidents = plugins.MarkerCluster().add_to(nyc_map)

# loop through the dataframe and add each data point to the mark cluste
r
for lat, lng, label, in zip(df_incidents.Latitude, df_incidents.Longitu
de, df_incidents['Complaint Type']):
    folium.Marker(
        location=[lat, lng],
        icon=None,
        popup=label,
    ).add_to(incidents)

# display map
nyc_map
```

Out[65]:
Make this Notebook Trusted to load map: File -> Trust Notebook

In [ ]:

Q3 Order the complaint types based on the average 'Request_Closing_Time', grouping them for different locations.

In [68]:
```
df1=csr_df.groupby(['City','Location Type','Complaint Type'])[['Request
_Closing_in_hrs']].mean().reset_index()
df1.fillna(0, inplace=True)
df1['Key']=df1['City']+df1['Location Type']+df1['Complaint Type']

df1.head()

df2=csr_df.groupby(['City','Location Type','Complaint Type'])[['request
_count']].sum().reset_index()
df2.fillna(0, inplace=True)
df2['Key']=df2['City']+df2['Location Type']+df2['Complaint Type']
df2.head()

df3=pd.merge(left=df1,
         right= df2,left_on='Key',right_on='Key',how='left')


df3.drop(['City_y','Location Type_y','Complaint Type_y','Key'],axis=1,
inplace=True)
df3.head()
```

Out[68]:

	City_x	Location Type_x	Complaint Type_x	Request_Closing_in_hrs	request_count
0	ARVERNE	Club/Bar/Restaurant	Drinking	0.238611	1
1	ARVERNE	House of Worship	Noise - House of Worship	1.562197	11
2	ARVERNE	Park/Playground	Noise - Park	1.283333	2
3	ARVERNE	Residential Building/House	Animal Abuse	2.081484	35
4	ARVERNE	Residential Building/House	Homeless Encampment	2.548333	1

Q5. Perform a statistical test for the following:

Please note: For the below statements you need to state the Null and Alternate and then provide a statistical test to accept or reject the Null Hypothesis along with the corresponding 'p-value'.

- Whether the average response time across complaint types is similar or not (overall)
- Are the type of complaint or service requested and location related?

In [69]:
```
df_statistics= csr_df[['Unique Key','Complaint Type','City','Location Type','Request_Closing_Time',
                       'Request_Closing_in_days','Request_Closing_in_hrs','request_count']]

df_statistics.isnull().sum()
```

Out[69]:
```
Unique Key                   0
Complaint Type               0
City                      2614
Location Type              131
Request_Closing_Time      2164
Request_Closing_in_days   2164
Request_Closing_in_hrs    2164
request_count                0
dtype: int64
```

In [70]:
```
print(df_statistics['Request_Closing_Time'].mean())
print(df_statistics['Request_Closing_in_days'].mean())
print(df_statistics['Request_Closing_in_hrs'].mean())
```

```
0 days 04:18:51.832782
0.016041723890746113
4.314397995240427
```

In [71]:
```
df_statistics['Request_Closing_Time'].fillna(df_statistics['Request_Closing_Time'].mean(),inplace=True)
df_statistics['Request_Closing_in_days'].fillna(df_statistics['Request_Closing_in_days'].mean(),inplace=True)
df_statistics['Request_Closing_in_hrs'].fillna(df_statistics['Request_Closing_in_hrs'].mean(),inplace=True)
df_statistics['City'].fillna('Unknown',inplace=True)
df_statistics['Location Type'].fillna('Unknown',inplace=True)
```

```
C:\Users\SujitSonar\Anaconda3\lib\site-packages\pandas\core\generic.py:6130: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
```

In [72]:
```
df_statistics.isnull().sum()
```

Out[72]:
```
Unique Key                  0
Complaint Type              0
City                        0
Location Type               0
Request_Closing_Time        0
Request_Closing_in_days     0
Request_Closing_in_hrs      0
request_count               0
dtype: int64
```

In [73]:
```
df_statistics.shape
```

Out[73]:
```
(300698, 8)
```

In [ ]:
```
################################################
```

In [74]:
```
df_hypo_test = df_statistics[['Complaint Type','Request_Closing_in_hrs
']]
df_hypo_test.rename(columns={'Complaint Type':'Complaint_Type'},inplace
=True)
df_hypo_test.shape
```

```
C:\Users\SujitSonar\Anaconda3\lib\site-packages\pandas\core\frame.py:
4025: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas
-docs/stable/indexing.html#indexing-view-versus-copy
```

Out[74]:
```
(300698, 2)
```

In [75]:
```
df_hypo_test.describe()
```

Out[75]:

	Request_Closing_in_hrs
count	300698.000000
mean	4.314398
std	6.067532
min	0.016667
25%	1.283333
50%	2.739722
75%	5.319167
max	592.872778

Null and Alternate hypothese

- null hypo (H0) = average response time across complaint types is similar
- alternate hypo (H1) = average response time across complaint types is not similar
- we will be doing two sided hypothesis testing as we stating the null hypothesis to be average response time across complaint types is similar (equal)

since significance level (Cofidence level) or alpha is not given, so we will assume Cofidenece level of 95%, thereofore alpha = 0.05

In [76]:
```
df_hypo_test_grp = df_hypo_test.groupby(['Complaint_Type'])[['Request_Closing_in_hrs']].mean().reset_index()
df_hypo_test_grp.rename(columns={'Request_Closing_in_hrs':'avg_resolution_time_in_hrs'}, inplace=True)
df_hypo_test_grp.sort_values('avg_resolution_time_in_hrs',ascending=False)
```

Out[76]:

	Complaint_Type	avg_resolution_time_in_hrs
2	Animal in a Park	336.834722
5	Derelict Vehicle	7.341763
9	Graffiti	7.151251
0	Agency Issues	5.260324
1	Animal Abuse	5.212088
4	Blocked Driveway	4.739610
12	Illegal Parking	4.499097
18	Panhandling	4.372388
10	Homeless Encampment	4.365570
8	Ferry Complaint	4.314398
20	Squeegee	4.045625
23	Vending	4.014472
7	Drinking	3.863601
3	Bike/Roller/Skate Chronic	3.770312
22	Urinating in Public	3.626664
17	Noise - Vehicle	3.591117
6	Disorderly Youth	3.558584
16	Noise - Street/Sidewalk	3.454808
21	Traffic	3.449066
15	Noise - Park	3.415204
14	Noise - House of Worship	3.195707
13	Noise - Commercial	3.157981
11	Illegal Fireworks	2.761139
19	Posting Advertisement	1.982999

In [ ]:
```
# just by looking at the above table, average resolution time (in hrs) across complaint type is not similar.
# There are mean is mostly between  3 to 4 hrs
```

In [ ]:
```

```

In [77]:
```
# Anova test to test the hypothesis

import statsmodels.api as sm
from statsmodels.formula.api import ols

mod = ols('Request_Closing_in_hrs ~ Complaint_Type',data=df_hypo_test).
fit()

aov_table=sm.stats.anova_lm(mod,type=2)
print(aov_table)
```

```
                        df         sum_sq       mean_sq           F  PR(>
F)
Complaint_Type        23.0  4.019952e+05  17478.050228  492.60606
0.0
Residual          300674.0  1.066815e+07     35.480786         NaN     N
aN
```

p value (0.0) is less than alpha (0.05), hence we reject the Null Hypothesis and accept the alretnate hypothesis

alternate hypo (H1) = average response time across complaint types is not similar

In [ ]:
```

```

Are the type of complaint or service requested and location related?

- H0: There is no relationship between complaint/service requested and location related
- H1: There is a relationship between complaint/service requested and location related

since significance level (Cofidence level) or alpha is not given, so we will assume Cofidenece level of 95%, thereofore alpha = 0.05

In [78]:
```
#selecting a 50% sample of the data
df_statistics_sample=df_statistics.sample(frac=0.5, replace=False, rand
om_state=1)
```

In [79]:
```
df_statistics_sample.shape
```

Out[79]: (150349, 8)

In [80]:
```
from scipy.stats import chi2
from scipy.stats import chi2_contingency
```

In [81]:
```
# Checking by Location Type

df_location_type = pd.crosstab(df_statistics_sample['Location Type'], d
f_statistics_sample['Complaint Type'])
```

In [82]:
```
stat, p, dof, expected = chi2_contingency(df_location_type)
CI = 0.95
alpha = 1-CI
critical = chi2.ppf(CI, dof)

print('dof=%d' % dof)
print(f'pvalue is: {p}')
print(f'alpha is: {alpha}')
```

```
dof=391
pvalue is: 0.0
alpha is: 0.050000000000000044
```

In [83]:
```
p<alpha
```

Out[83]:
```
True
```

In [84]:
```
# Checking by City

df_city_type = pd.crosstab(df_statistics_sample['City'], df_statistics_
sample['Complaint Type'])
stat, p, dof, expected = chi2_contingency(df_city_type)
CI = 0.95
alpha = 1-CI
critical = chi2.ppf(CI, dof)

print('dof=%d' % dof)
print(f'pvalue is: {p}')
print(f'alpha is: {alpha}')
```

```
dof=1196
pvalue is: 0.0
alpha is: 0.050000000000000044
```

In [85]:
```
p<alpha
```

Out[85]:
```
True
```

- p values is less than alpha, hence reject null hypothesei, H0 and accept altentae hypothesis H1
- Hence there is a relationship between complaint/service requested and location related

In [ ]:
```
##########################################################################
################
```

Conclusions

- we see that majority of the average request/complaints closing time is within 4 hours of it is being created
- from the binning table, we see that around 78% of the requests/complaints are closed within 0 to 5.9 hrs, which is close to the average closing time
- 0 days 04:18:51.832782 is the average request closing time
- Looking at the Wordcloud , Blocked Driveway and Illegal Praking are the two top most type of complaints received by NYC311
- At a very high level without any statistical proof, looking at the two column charts we see that the type of complaints like the top three complaint types, 'bloked driveway', 'illegal parking', 'noise, stret/sidewalk', do occur mostly in 'street/sidewalk' location type.
- Brooklyn, New york and Bronx are listed as top three cities where complaints/Requests being recorded as shown in the bar chart
- Brooklyn, Queens and Manhatta are the Top three Borough where complaints/Requests being recorded as shown in the pie chart
- Majority of the Complaint/Requests are already closed as shown in the pie chart
- Throughout the year, we see that Blocked Driveway, Illegal parking, Noise-Street/Sidewalk has been reported as the highest complaint types as shon in the olumn cahrt by month

Statistical testing

- p value (0.0) is less than alpha (0.05), hence we reject the Null Hypothesis and accept the alretnate hypothesis alternate hypo (H1) = average response time across complaint types is not similar
- p value (0.0) is less than alpha (0.05), hence reject null hypothesei, H0 and accept altentae hypothesis H1 Hence there is a relationship between complaint/service requested and location related

In [ ]:
```
##################################################################
################
```