# 01-911 Calls Data Capstone Project

April 27, 2019

## 1 911 Calls Capstone Project

For this capstone project we will be analyzing some 911 call data from Kaggle. The data contains the following fields:

- lat : String variable, Latitude
- lng: String variable, Longitude
- desc: String variable, Description of the Emergency Call
- zip: String variable, Zipcode
- title: String variable, Title
- timeStamp: String variable, YYYY-MM-DD HH:MM:SS
- twp: String variable, Township
- addr: String variable, Address
- e: String variable, Dummy variable (always 1)

Just go along with this notebook and try to complete the instructions or answer the questions in bold using your Python and Data Science skills!

### 1.1 Data and Setup

---

** Import numpy and pandas **
** Import visualization libraries and set %matplotlib inline. **
** Read in the csv file as a dataframe called df **
** Check the info() of the df **

In [132]:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99492 entries, 0 to 99491
Data columns (total 9 columns):
lat          99492 non-null float64
lng          99492 non-null float64
desc         99492 non-null object
zip          86637 non-null float64
title        99492 non-null object
timeStamp    99492 non-null object
```

```
twp          99449 non-null object
addr         98973 non-null object
e            99492 non-null int64
dtypes: float64(3), int64(1), object(5)
memory usage: 6.8+ MB
```

** Check the head of df **

In [155]:

Out[155]:

| | lat | lng | desc | zip | title | timeStamp | twp | addr | e | Reason | Hour | Month | Day of Week |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 40.297876 | -75.581294 | REINDEER CT & DEAD END; NEW HANOVER; Station ... | 19525.0 | EMS: BACK PAINS/INJURY | 2015-12-10 17:40:00 | NEW HANOVER | REINDEER CT & DEAD END | 1 | EMS | 17 | 12 | Thu |
| 1 | 40.258061 | -75.264680 | BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP... | 19446.0 | EMS: DIABETIC EMERGENCY | 2015-12-10 17:40:00 | HATFIELD TOWNSHIP | BRIAR PATH & WHITEMARSH LN | 1 | EMS | 17 | 12 | Thu |
| 2 | 40.121182 | -75.351975 | HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21-St... | 19401.0 | Fire: GAS-ODOR/LEAK | 2015-12-10 17:40:00 | NORRISTOWN | HAWS AVE | 1 | Fire | 17 | 12 | Thu |

## 1.2 Basic Questions

** What are the top 5 zipcodes for 911 calls? **

In [134]:

```
Out[134]: 19401.0    6979
          19464.0    6643
          19403.0    4854
          19446.0    4748
          19406.0    3174
          Name: zip, dtype: int64
```

** What are the top 5 townships (twp) for 911 calls? **

In [135]:

```
Out[135]: LOWER MERION    8443
          ABINGTON        5977
          NORRISTOWN      5890
          UPPER MERION    5227
          CHELTENHAM      4575
          Name: twp, dtype: int64
```

** Take a look at the 'title' column, how many unique title codes are there? **

```
In [136]:
```

```
Out[136]: 110
```

### 1.3 Creating new features

** In the titles column there are "Reasons/Departments" specified before the title code. These are EMS, Fire, and Traffic. Use .apply() with a custom lambda expression to create a new column called "Reason" that contains this string value.**

**For example, if the title column value is EMS: BACK PAINS/INJURY , the Reason column value would be EMS.**

** What is the most common Reason for a 911 call based off of this new column? **

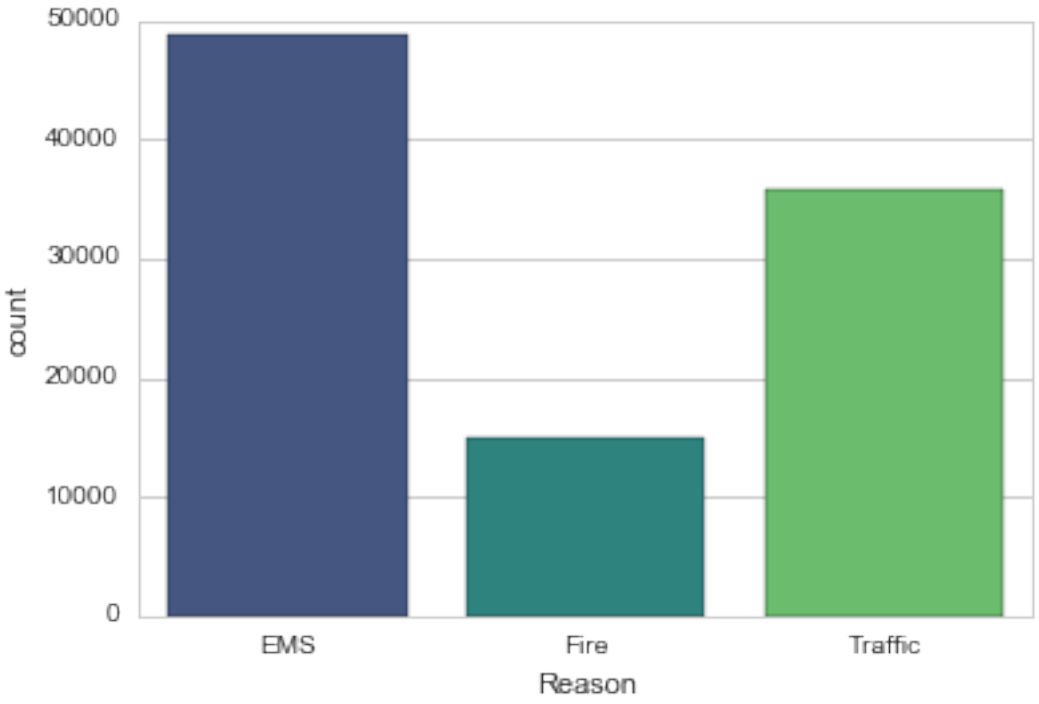
```
In [138]:
```

```
Out[138]: EMS        48877
          Traffic    35695
          Fire       14920
          Name: Reason, dtype: int64
```

** Now use seaborn to create a countplot of 911 calls by Reason. **

```
In [139]:
```

```
Out[139]: <matplotlib.axes._subplots.AxesSubplot at 0x12d3830b8>
```



** Now let us begin to focus on time information. What is the data type of the objects in the timeStamp column? **

```
In [140]:
```

```
Out[140]: str
```

** You should have seen that these timestamps are still strings. Use pd.to_datetime to convert the column from strings to DateTime objects. **

** You can now grab specific attributes from a Datetime object by calling them. For example:**

```
time = df['timeStamp'].iloc[0]
time.hour
```

**You can use Jupyter's tab method to explore the various attributes you can call. Now that the timestamp column are actually DateTime objects, use .apply() to create 3 new columns called Hour, Month, and Day of Week. You will create these columns based off of the timeStamp column, reference the solutions if you get stuck on this step.**

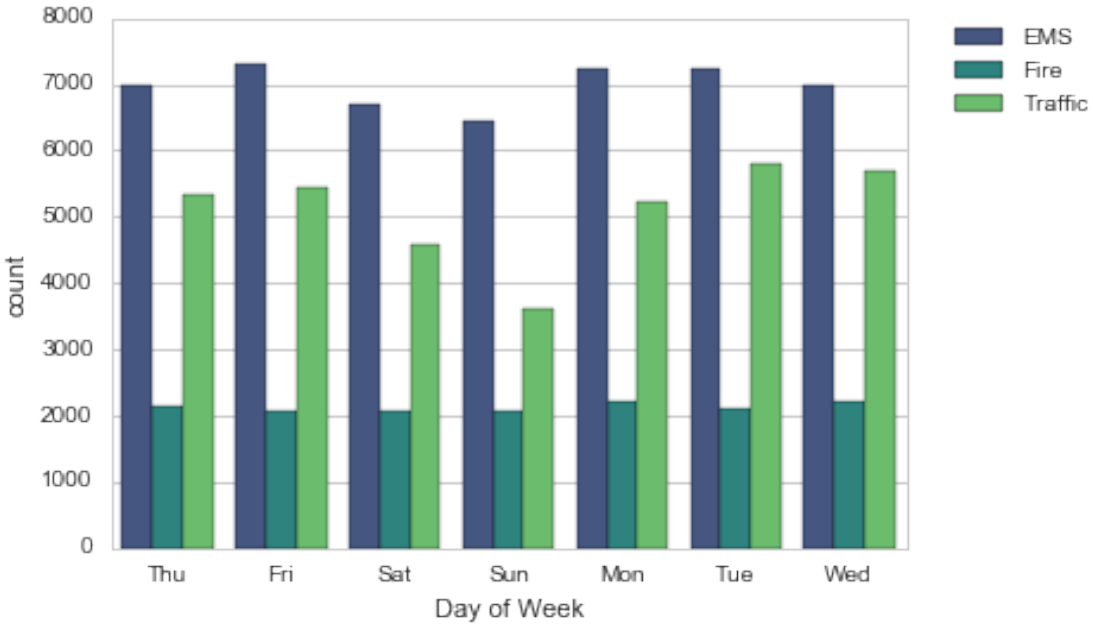
** Notice how the Day of Week is an integer 0-6. Use the .map() with this dictionary to map the actual string names to the day of the week: **

```
dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}
```

** Now use seaborn to create a countplot of the Day of Week column with the hue based off of the Reason column. **

```
In [168]:
```

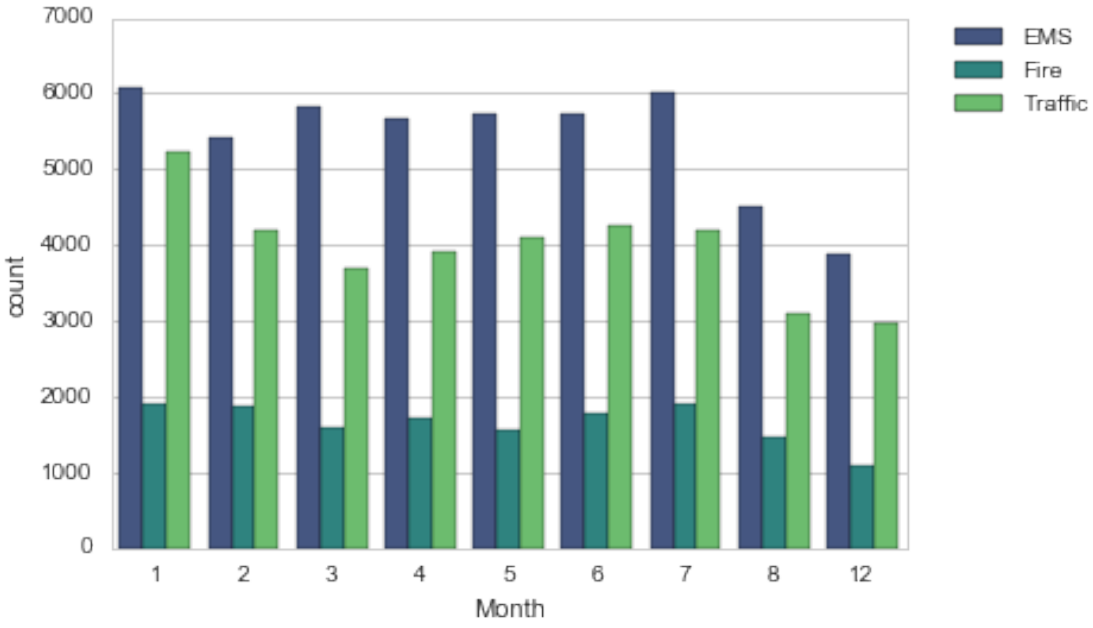
```
Out[168]: <matplotlib.legend.Legend at 0x12f614048>
```



**Now do the same for Month:**

In [3]:

Out[3]: <matplotlib.legend.Legend at 0x10330ada0>




**Did you notice something strange about the Plot?**

---

** You should have noticed it was missing some Months, let's see if we can maybe fill in this information by plotting the information in another way, possibly a simple line plot that fills in the missing months, in order to do this, we'll need to do some work with pandas... **

** Now create a gropuby object called byMonth, where you group the DataFrame by the month column and use the count() method for aggregation. Use the head() method on this returned DataFrame. **

In [169]:

Out[169]:

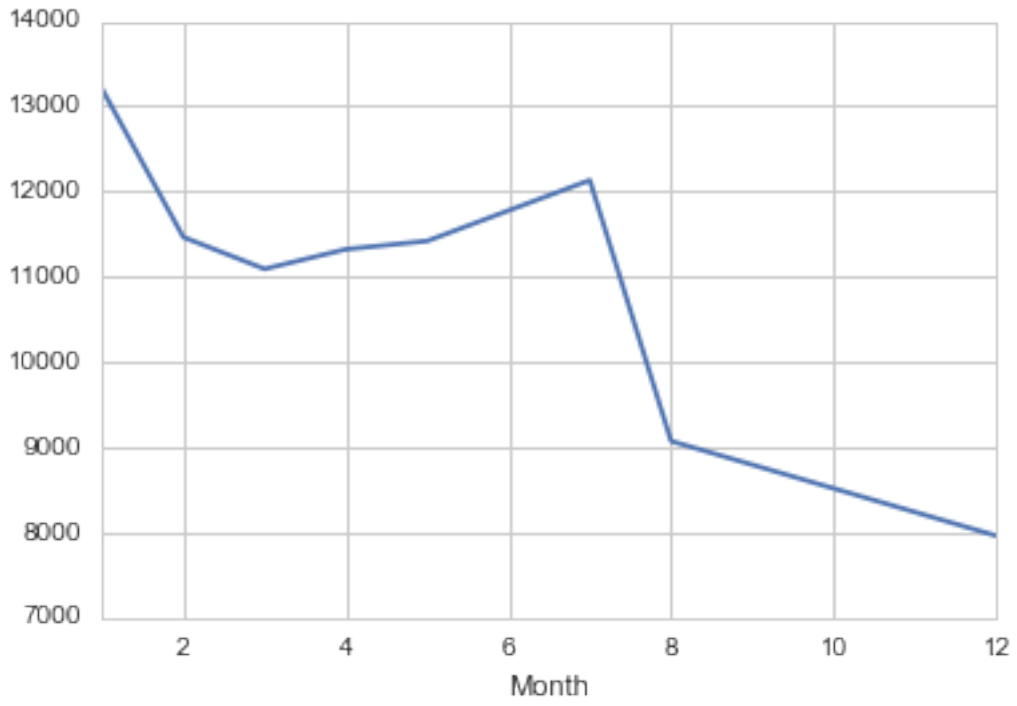
| Month | lat | lng | desc | zip | title | timeStamp | twp | addr | e |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 13205 | 13205 | 13205 | 11527 | 13205 | 13205 | 13203 | 13096 | 13205 |
| 2 | 11467 | 11467 | 11467 | 9930 | 11467 | 11467 | 11465 | 11396 | 11467 |
| 3 | 11101 | 11101 | 11101 | 9755 | 11101 | 11101 | 11092 | 11059 | 11101 |
| 4 | 11326 | 11326 | 11326 | 9895 | 11326 | 11326 | 11323 | 11283 | 11326 |
| 5 | 11423 | 11423 | 11423 | 9946 | 11423 | 11423 | 11420 | 11378 | 11423 |

```
       Reason   Hour  Day of Week
Month
1       13205  13205        13205
2       11467  11467        11467
3       11101  11101        11101
4       11326  11326        11326
5       11423  11423        11423
```

** Now create a simple plot off of the dataframe indicating the count of calls per month. **

In [175]:

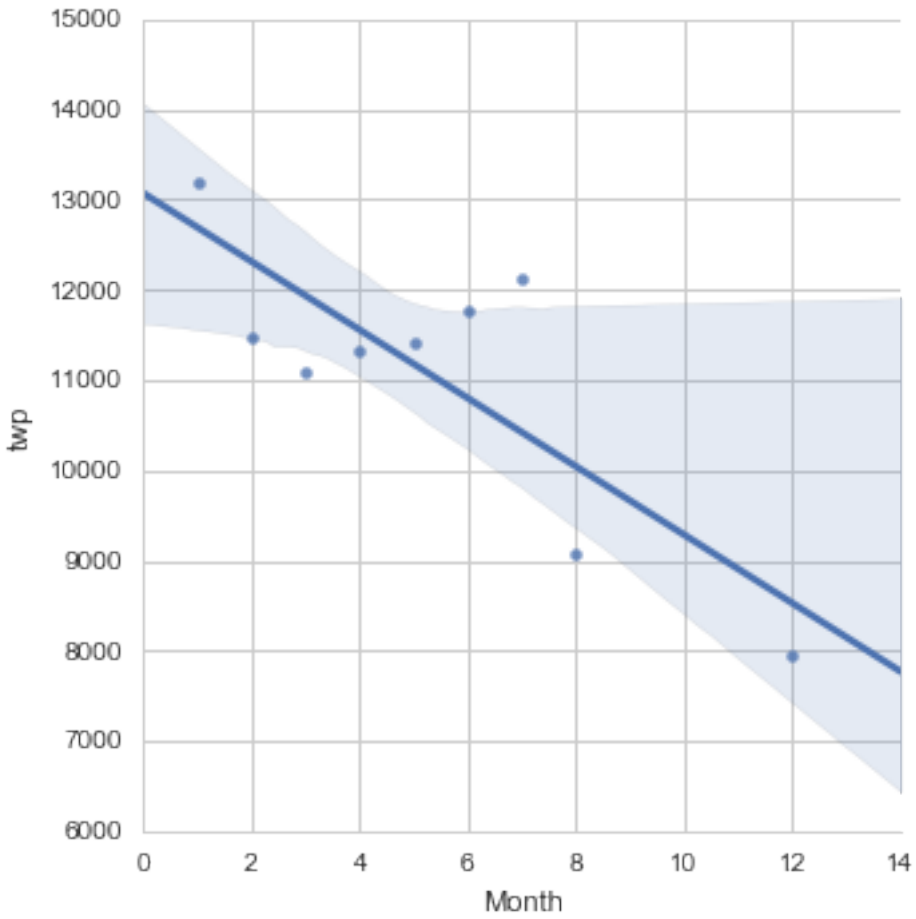
Out[175]: <matplotlib.axes._subplots.AxesSubplot at 0x133a3c080>




** Now see if you can use seaborn's lmplot() to create a linear fit on the number of calls per month. Keep in mind you may need to reset the index to a column. **

In [187]:

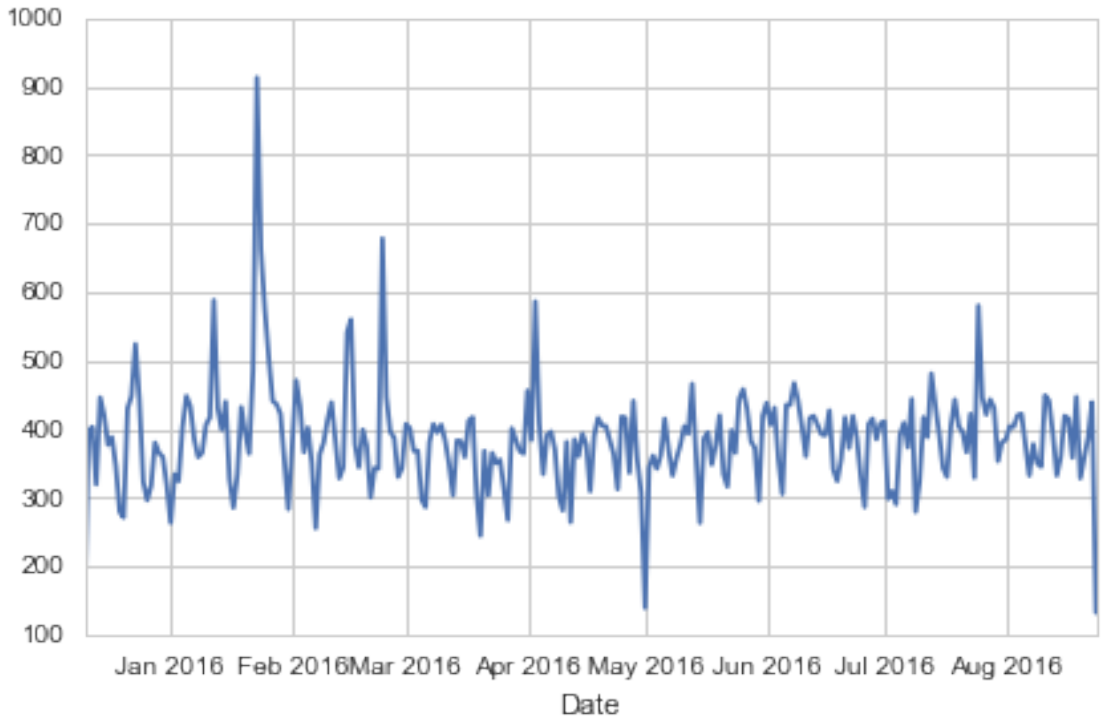
Out[187]: <seaborn.axisgrid.FacetGrid at 0x1342acd30>




**Create a new column called 'Date' that contains the date from the timeStamp column. You'll need to use apply along with the .date() method.**

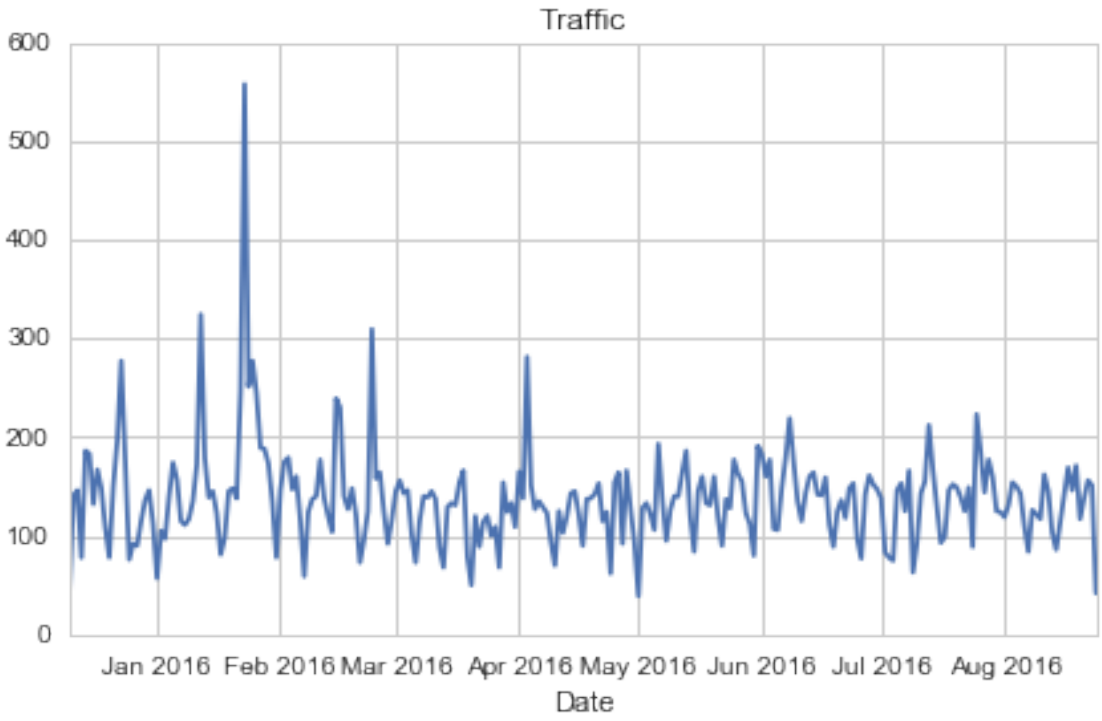
** Now groupby this Date column with the count() aggregate and create a plot of counts of 911 calls.**

In [197]:

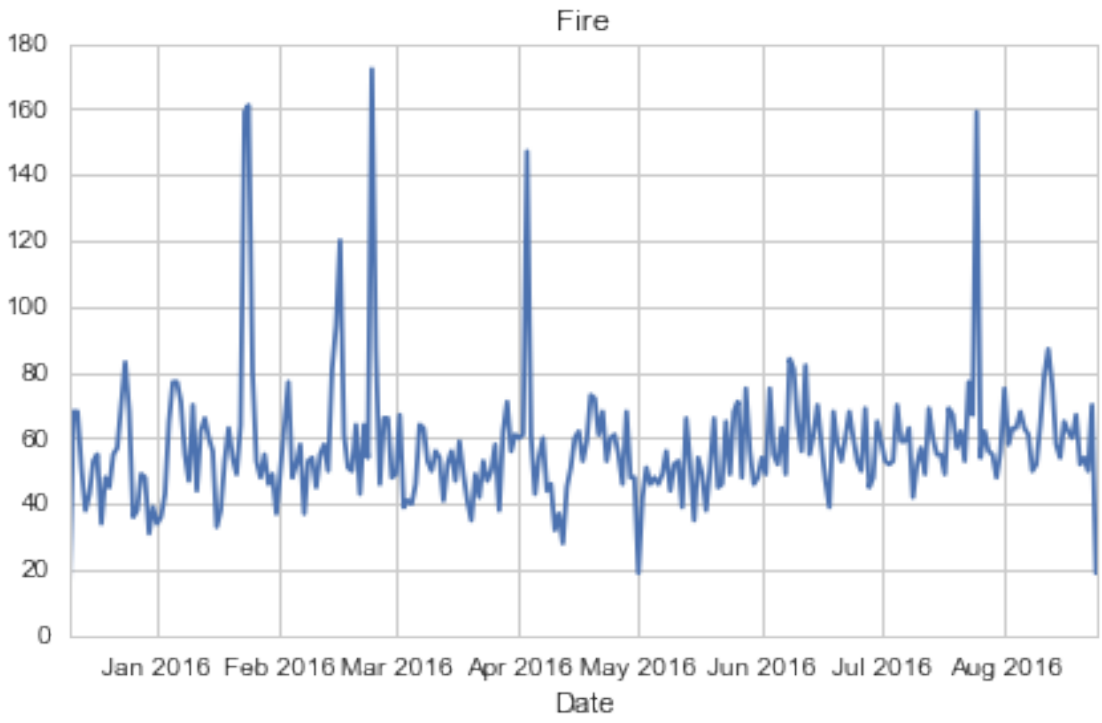



** Now recreate this plot but create 3 separate plots with each plot representing a Reason for the 911 call**

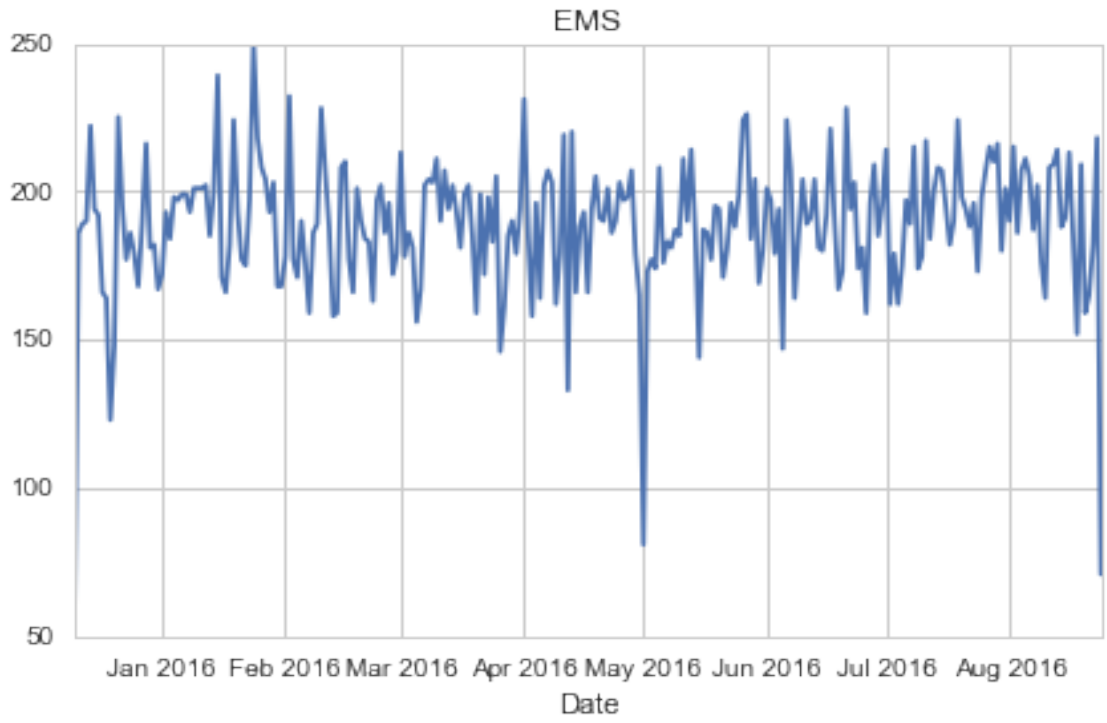
In [199]:



In [201]:


Fire
180
160
140
120
100
80
60
40
20
0
Jan 2016
Feb 2016
Mar 2016
Apr 2016
May 2016
Jun 2016
Jul 2016
Aug 2016
Date

In [202]:




---

** Now let's move on to creating heatmaps with seaborn and our data. We'll first need to restructure the dataframe so that the columns become the Hours and the Index becomes the Day of the Week. There are lots of ways to do this, but I would recommend trying to combine groupby with an unstack method. Reference the solutions if you get stuck on this!**

In [203]:

Out[203]:

| Hour | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 14 | 15 | \ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Day of Week | | | | | | | | | | | ... | | | |
| Fri | 275 | 235 | 191 | 175 | 201 | 194 | 372 | 598 | 742 | 752 | ... | 932 | 980 | |
| Mon | 282 | 221 | 201 | 194 | 204 | 267 | 397 | 653 | 819 | 786 | ... | 869 | 913 | |
| Sat | 375 | 301 | 263 | 260 | 224 | 231 | 257 | 391 | 459 | 640 | ... | 789 | 796 | |
| Sun | 383 | 306 | 286 | 268 | 242 | 240 | 300 | 402 | 483 | 620 | ... | 684 | 691 | |
| Thu | 278 | 202 | 233 | 159 | 182 | 203 | 362 | 570 | 777 | 828 | ... | 876 | 969 | |

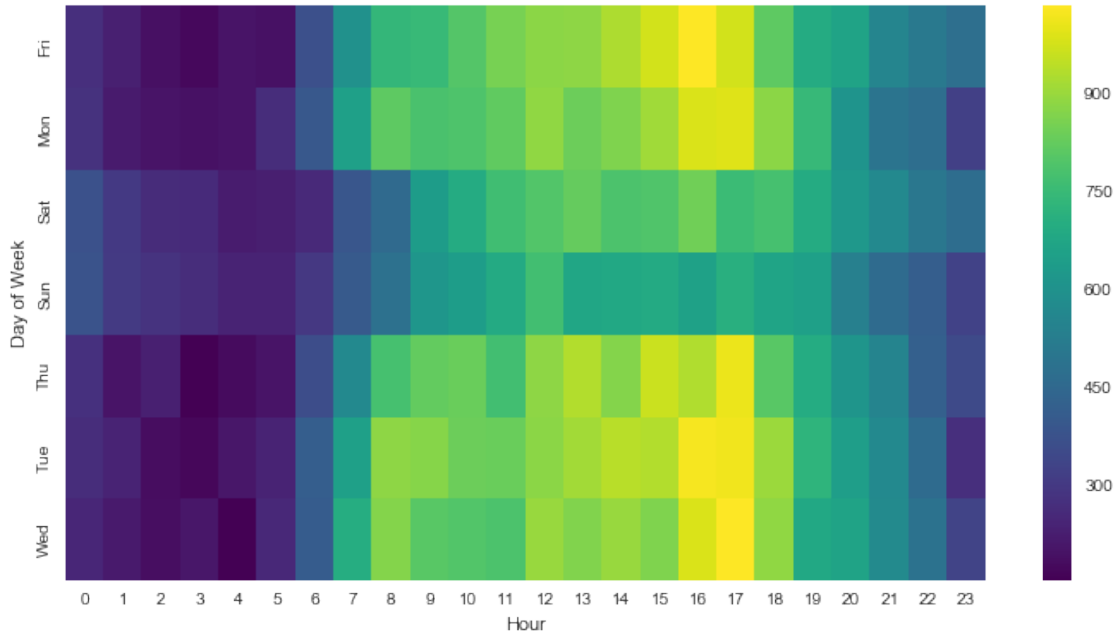
| Hour | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|
| Day of Week | | | | | | | | |
| Fri | 1039 | 980 | 820 | 696 | 667 | 559 | 514 | 474 |
| Mon | 989 | 997 | 885 | 746 | 613 | 497 | 472 | 325 |
| Sat | 848 | 757 | 778 | 696 | 628 | 572 | 506 | 467 |
| Sun | 663 | 714 | 670 | 655 | 537 | 461 | 415 | 330 |
| Thu | 935 | 1013 | 810 | 698 | 617 | 553 | 424 | 354 |

```
[5 rows x 24 columns]
```

** Now create a HeatMap using this new DataFrame. **

In [204]:

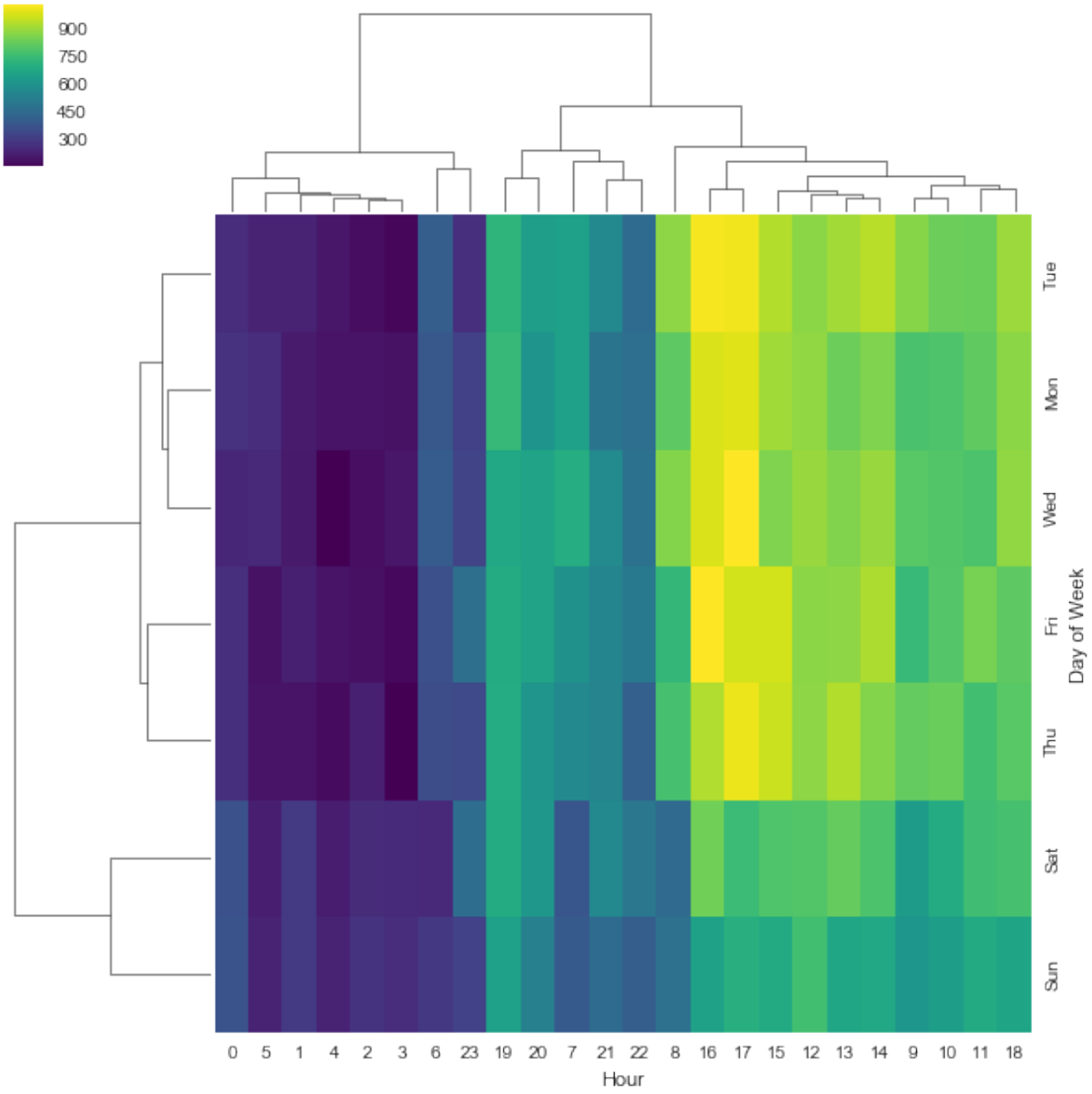
Out[204]: <matplotlib.axes._subplots.AxesSubplot at 0x1253fa198>




** Now create a clustermap using this DataFrame. **

In [205]:

Out[205]: <seaborn.matrix.ClusterGrid at 0x1304fb668>




** Now repeat these same plots and operations, for a DataFrame that shows the Month as the column. **

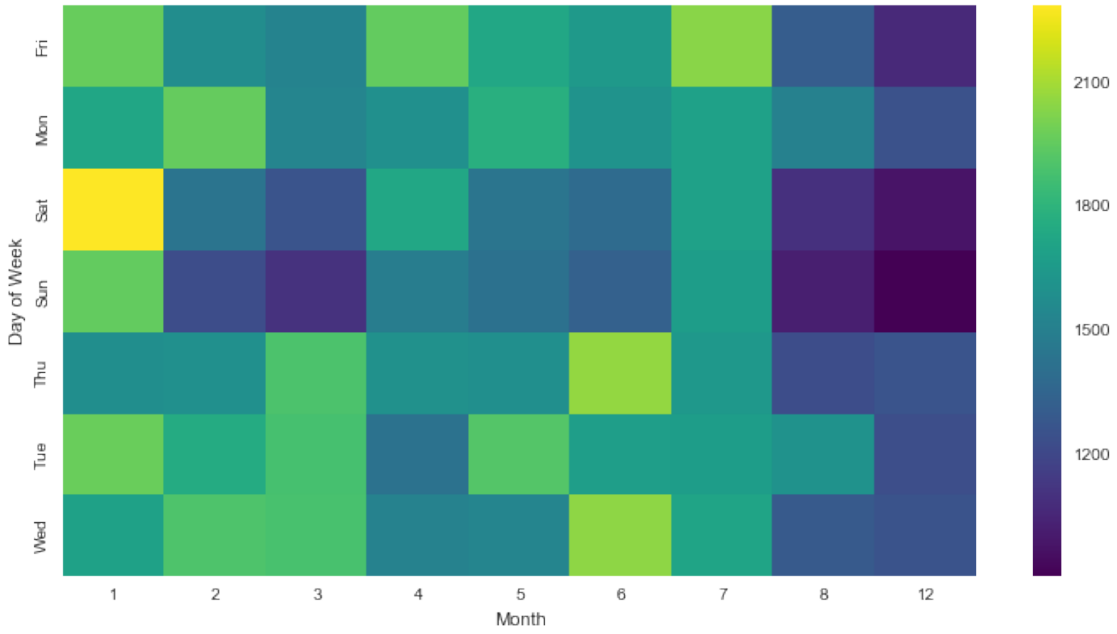
In [207]:

Out[207]:

| Month | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 12 |
|---|---|---|---|---|---|---|---|---|---|
| Day of Week | | | | | | | | | |
| Fri | 1970 | 1581 | 1525 | 1958 | 1730 | 1649 | 2045 | 1310 | 1065 |
| Mon | 1727 | 1964 | 1535 | 1598 | 1779 | 1617 | 1692 | 1511 | 1257 |
| Sat | 2291 | 1441 | 1266 | 1734 | 1444 | 1388 | 1695 | 1099 | 978 |
| Sun | 1960 | 1229 | 1102 | 1488 | 1424 | 1333 | 1672 | 1021 | 907 |
| Thu | 1584 | 1596 | 1900 | 1601 | 1590 | 2065 | 1646 | 1230 | 1266 |

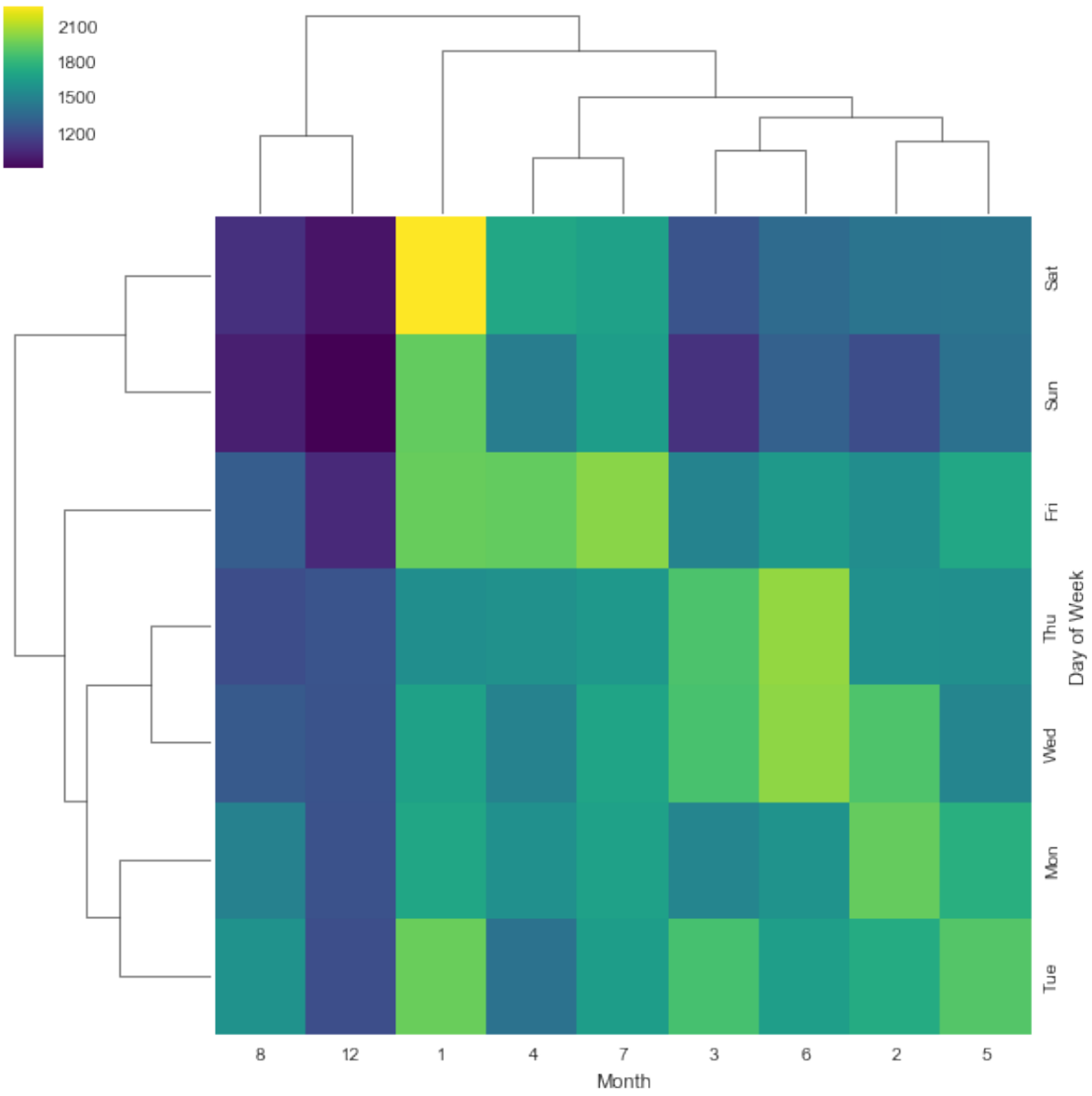
In [208]:

Out[208]: <matplotlib.axes._subplots.AxesSubplot at 0x1304fbd30>




In [209]:

Out[209]: <seaborn.matrix.ClusterGrid at 0x12a1a61d0>




**Continue exploring the Data however you see fit!** # Great Job!