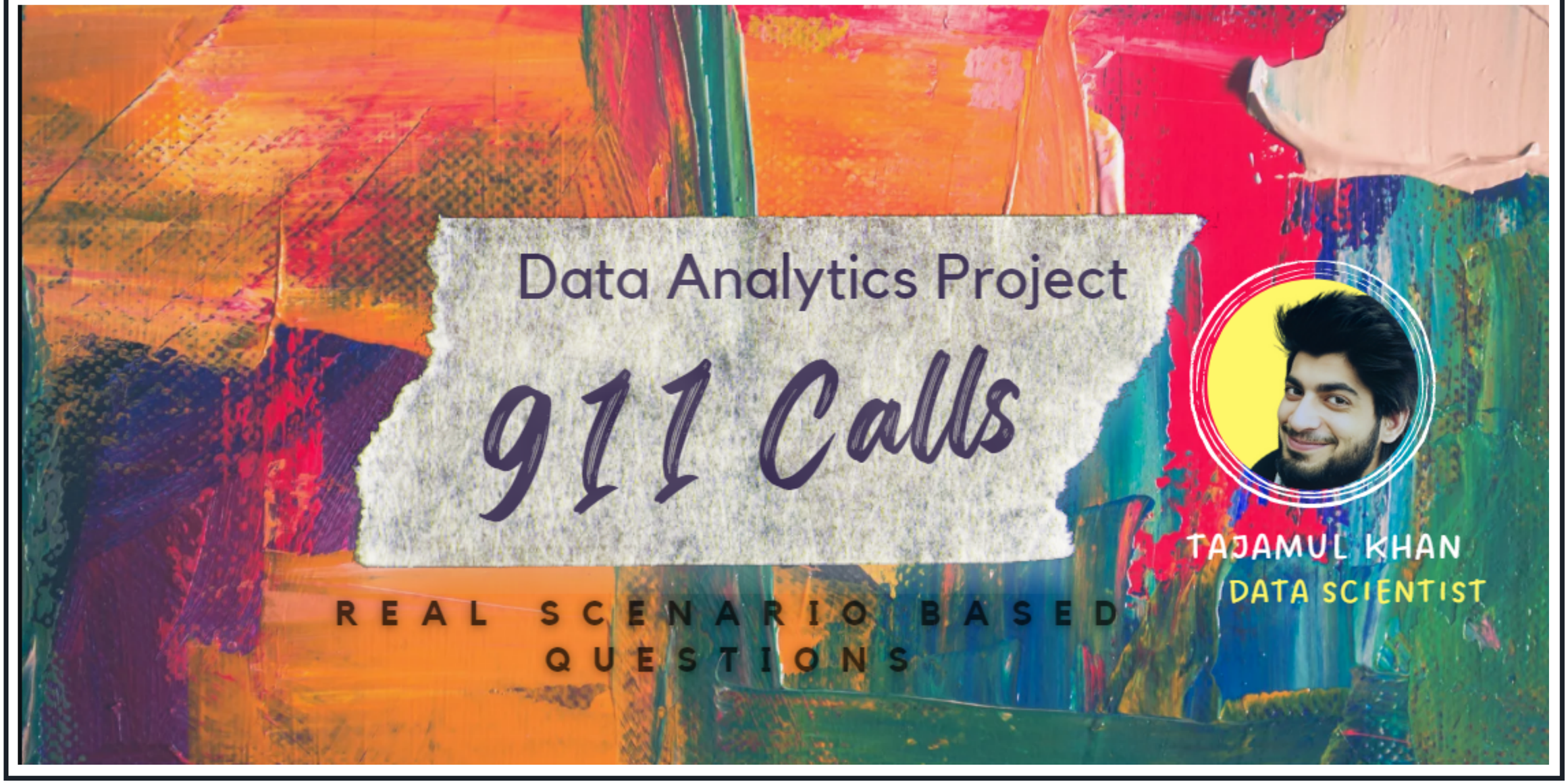
911 Calls Capstone Project



For this capstone project we will be analyzing some 911 call data from [Kaggle (https://www.kaggle.com/mchirico/montcoalert)](https://www.kaggle.com/mchirico/montcoalert). 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!

Data and Setup

Import numpy and pandas

In [2]:
```
import numpy as np
import pandas as pd
```

Import visualization libraries and set %matplotlib inline.

In [3]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import plotly
import cufflinks as cf
import warnings
warnings.filterwarnings('ignore')
```

Read in the csv file as a dataframe called df

In [4]:
```
df = pd.read_csv('911.csv')
```

Check the info() of the df

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99492 entries, 0 to 99491
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   lat        99492 non-null  float64
 1   lng        99492 non-null  float64
 2   desc       99492 non-null  object 
 3   zip        86637 non-null  float64
 4   title      99492 non-null  object 
 5   timeStamp  99492 non-null  object 
 6   twp        99449 non-null  object 
 7   addr       98973 non-null  object 
 8   e          99492 non-null  int64  
dtypes: float64(3), int64(1), object(5)
memory usage: 6.8+ MB
```

Check the head of df

In [6]:
```
df.head()
```

	lat	lng	desc	zip	title	timeStamp	twp	addr	e
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
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
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
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 17:40:01	NORRISTOWN	AIRY ST & SWEDE ST	1
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 17:40:01	LOWER POTTSGROVE	CHERRYWOOD CT & DEAD END	1

Basic Questions

What are the top 5 zipcodes for 911 calls?

In [7]:
```
df.zip.value_counts().head()
```

```
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 [8]:
```
df['twp'].value_counts().head()
```

```
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 [9]:
```
df.title.nunique()
```

```
110
```

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.

In [10]:
```
df['Reason']= df['title'].apply(lambda x: x.split(':')[0])
```

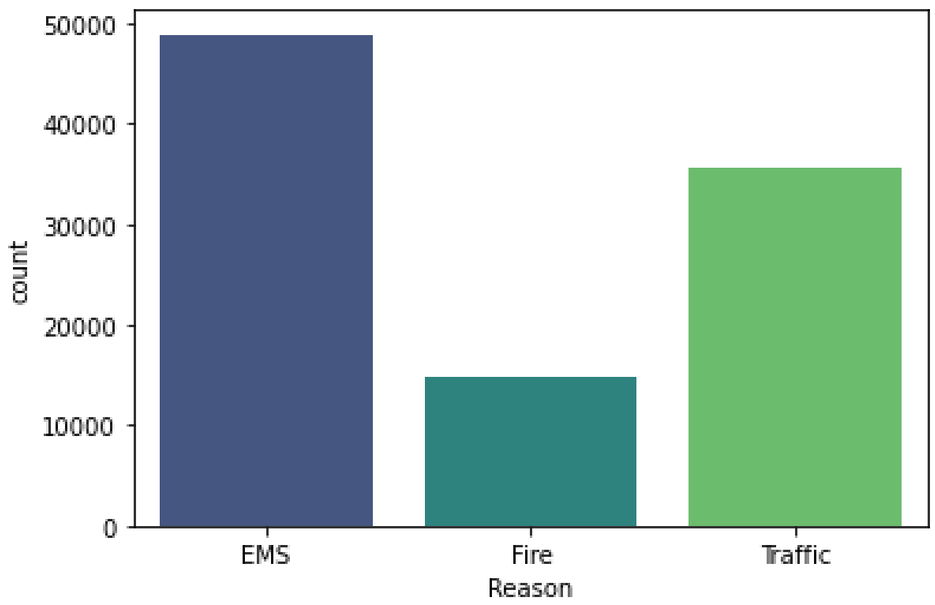
What is the most common Reason for a 911 call based off of this new column?

In [11]:
```
df['Reason'].value_counts().head()[:1]
```

```
EMS    48877
Name: Reason, dtype: int64
```

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

In [12]:
```
sns.countplot(df.Reason, palette='viridis');
```



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

In [13]:
```
type(df.timeStamp[0])
```

```
str
```

You should have seen that these timestamps are still strings. Use pd.to_datetime (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.to_datetime.html) to convert the column from strings to DateTime objects.

In [14]:
```
df.timeStamp = pd.to_datetime(df.timeStamp)
```

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.

In [15]:
```
df['Hour'] = df.timeStamp.apply(lambda x : x.hour)
df['Month'] = df.timeStamp.apply(lambda x : x.month)
df['dayofweek'] = df.timeStamp.apply(lambda x : x.dayofweek)
```

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'}
```

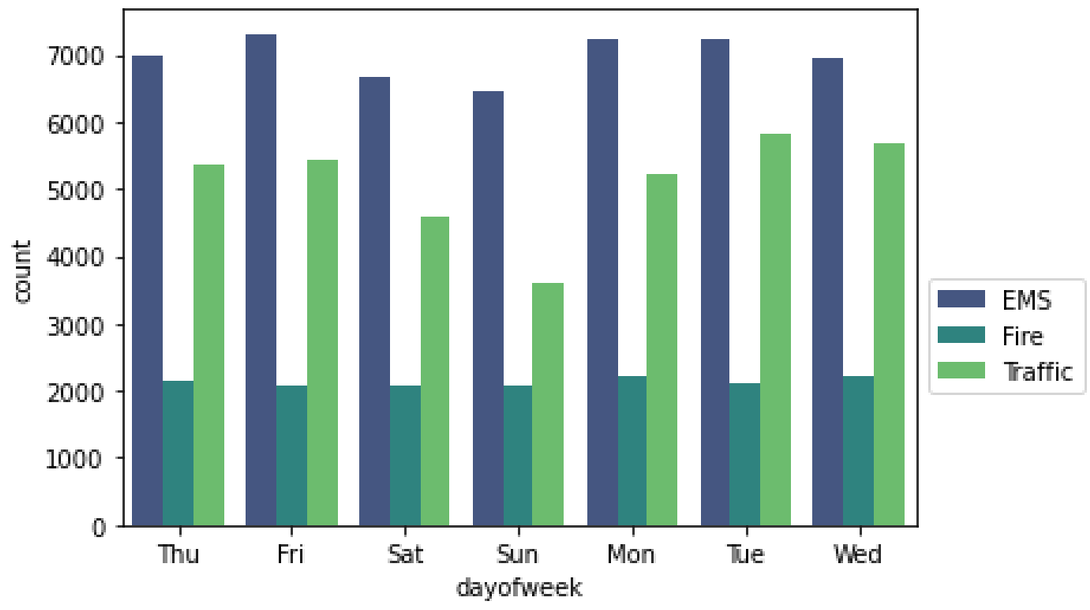
In [16]:
```
df['dayofweek'].head()
```

```
0    3
1    3
2    3
3    3
4    3
Name: dayofweek, dtype: int64
```

In [17]:
```
df['dayofweek'] = df['dayofweek'].map({0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'})
```

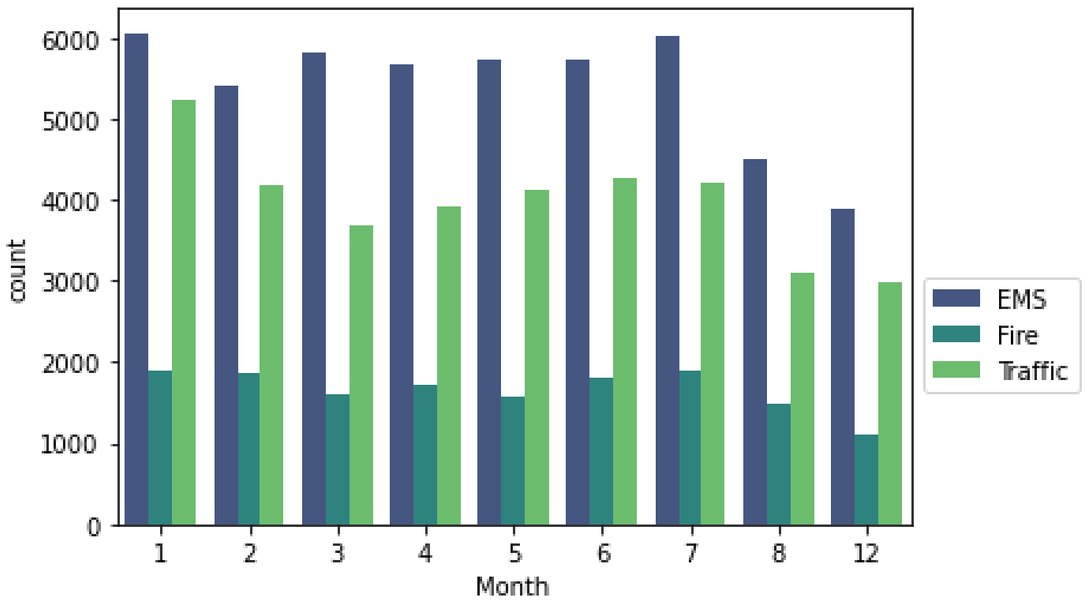
Now use seaborn to create a countplot of the Day of Week column with the hue based off of the Reason column.

In [18]:
```
sns.countplot(df['dayofweek'], hue = df.Reason, palette = 'viridis')
plt.legend(bbox_to_anchor=(1,0.5));
```



Now do the same for Month:

In [19]:
```
sns.countplot(df['Month'], hue = df.Reason, palette = 'viridis')
plt.legend(bbox_to_anchor=(1,0.5));
```



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...

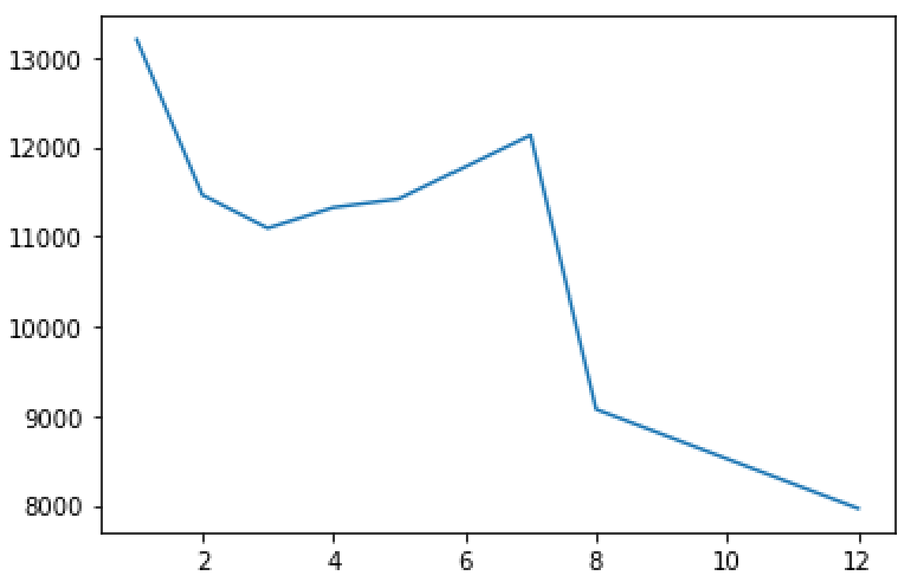
Now create a gropuby object called byMonth, where you group the DataFrame by the month column and use the count() method for aggregation. Use the head() method on this returned DataFrame.

In [20]:
```
byMonth = df.groupby('Month').count()
byMonth.head()
```

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

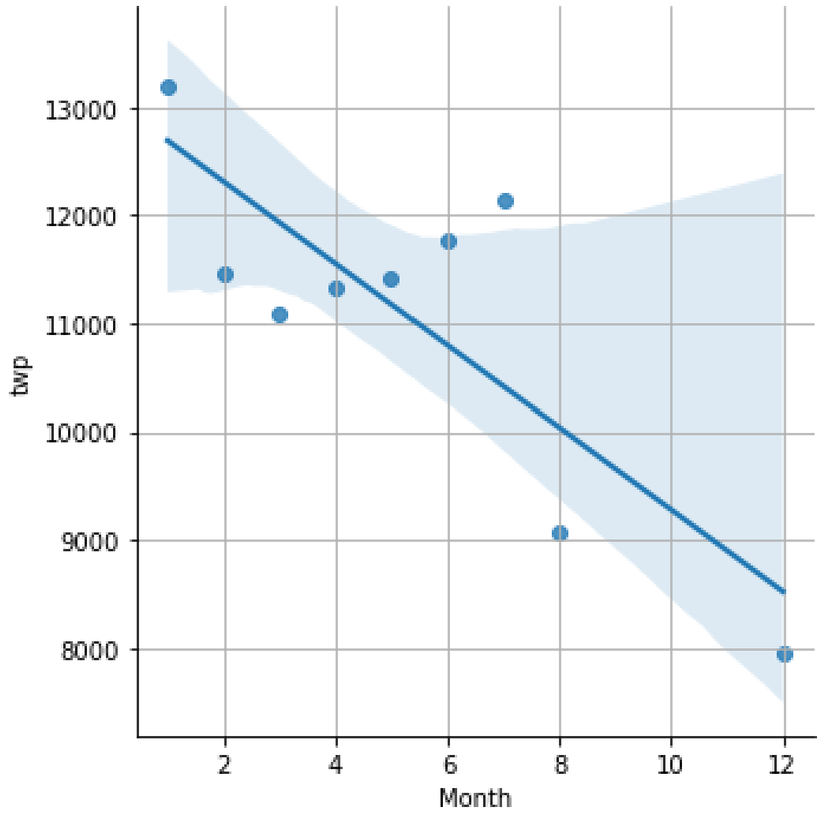
Now create a simple plot off of the dataframe indicating the count of calls per month.

In [21]:
```
plt.plot(byMonth['twp']);
```



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 [22]:
```
sns.lmplot(y='twp', x = 'Month', data = byMonth.reset_index())
plt.grid();
```



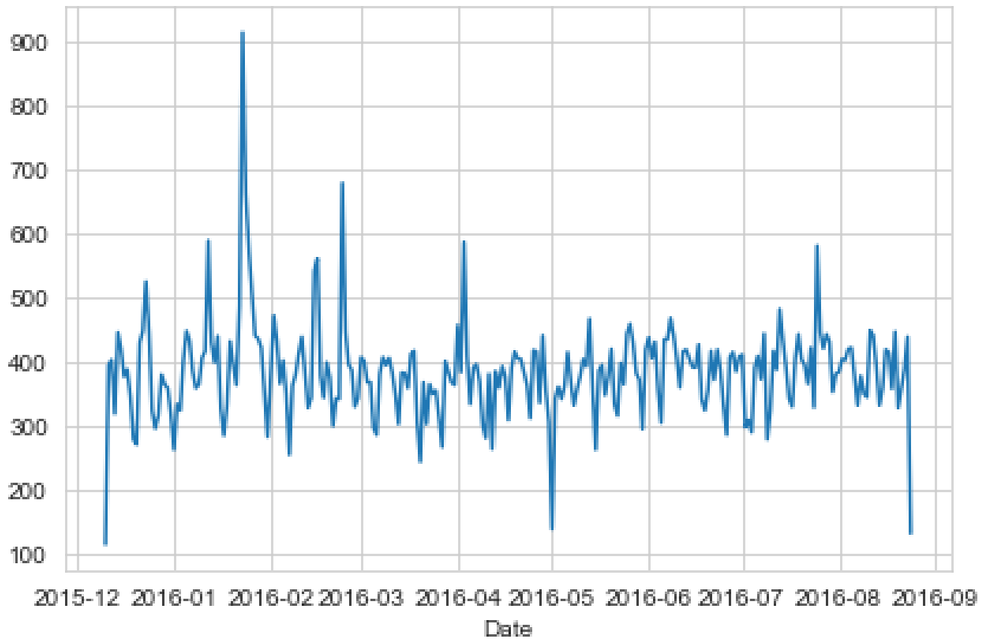
Create a new column called 'Date' that contains the date from the timeStamp column. You'll need to use apply along with the .date() method.

In [23]:
```
df['Date']= df['timeStamp'].apply(lambda x: x.date())
df.head()
```

	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Reason	Hour	Month
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
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
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
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 17:40:01	NORRISTOWN	AIRY ST & SWEDE ST	1	EMS	17	12
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 17:40:01	LOWER POTTSGROVE	CHERRYWOOD CT & DEAD END	1	EMS	17	12

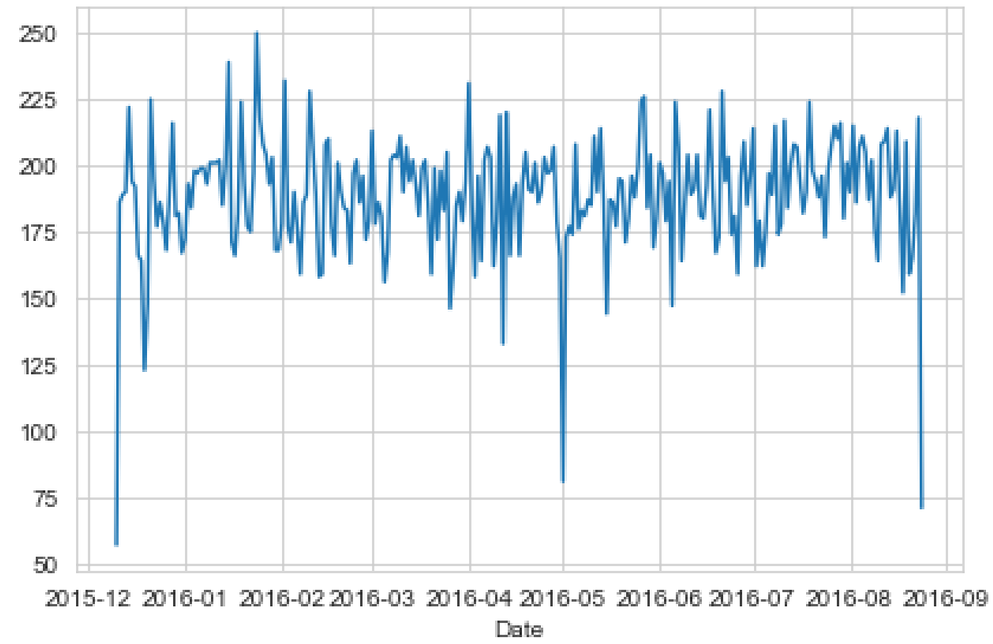
Now groupby this Date column with the count() aggregate and create a plot of counts of 911 calls.

In [24]:
```
sns.set_style('whitegrid')
df.groupby('Date').count()['twp'].plot()
plt.tight_layout()
```

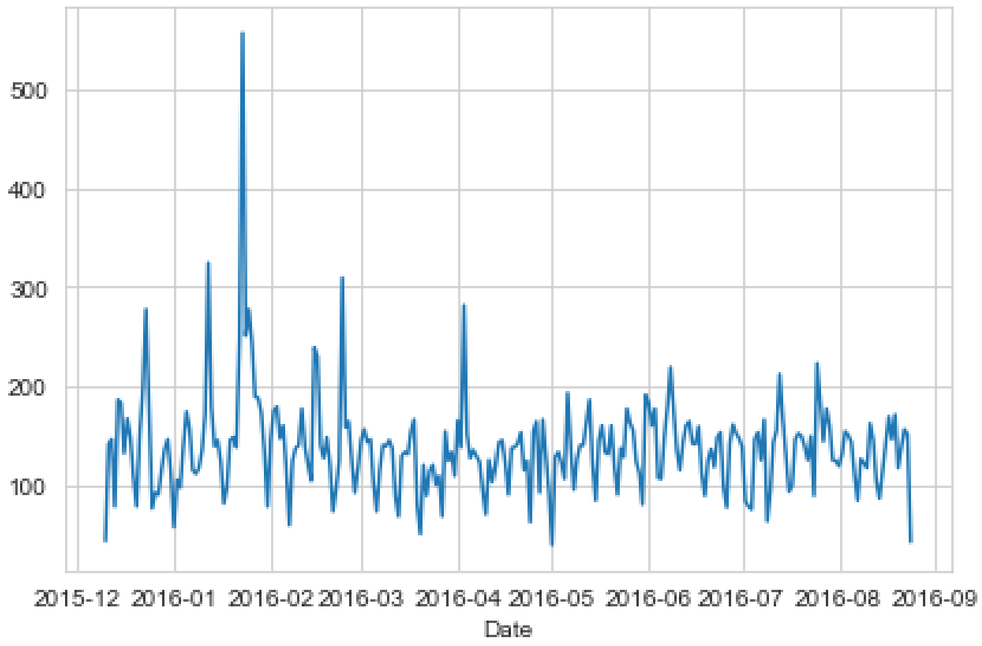


Now recreate this plot but create 3 separate plots with each plot representing a Reason for the 911 call

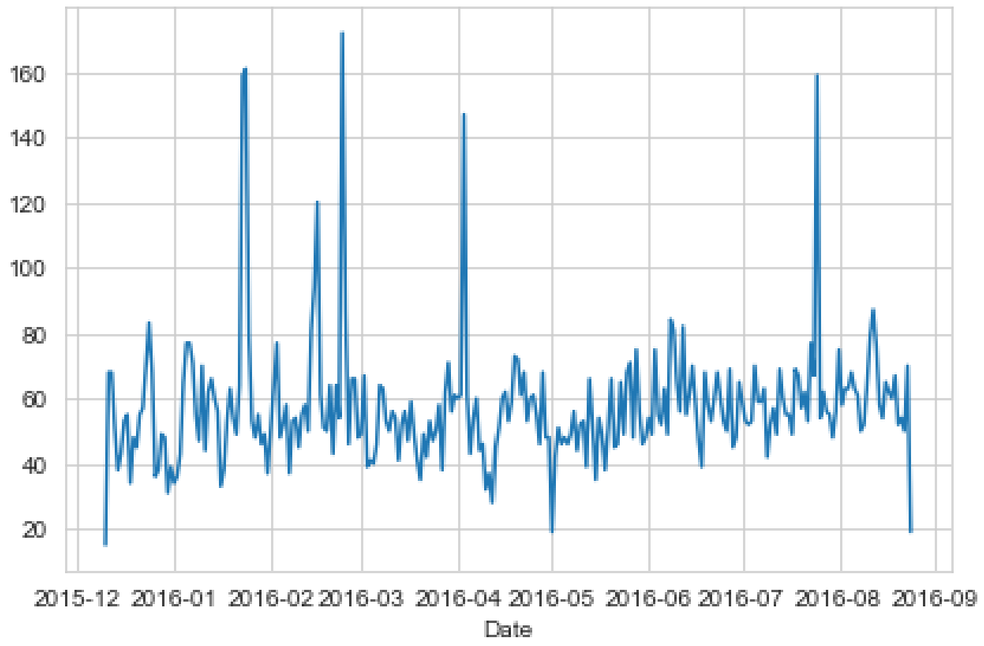
In [25]:
```
df[df['Reason']=='EMS'].groupby('Date').count()['twp'].plot()
plt.tight_layout()
```



In [26]:
```
df[df['Reason']=='Traffic'].groupby('Date').count()['twp'].plot()
plt.tight_layout()
```



In [27]:
```
df[df['Reason']=='Fire'].groupby('Date').count()['twp'].plot()
plt.tight_layout()
```



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 (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.unstack.html) method. Reference the solutions if you get stuck on this!

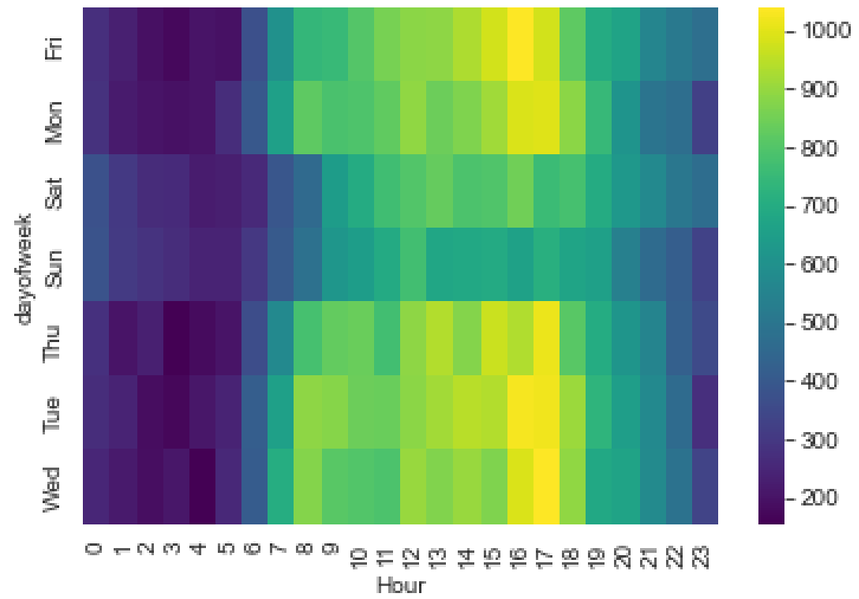
In [28]:
```
dayHour = df.groupby(by=['dayofweek','Hour']).count()['Reason'].unstack()
dayHour.head()
```

Hour	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	23
dayofweek																					
Fri	275	235	191	175	201	194	372	598	742	752	...	932	980	1039	980	820	696	667	559	514	474
Mon	282	221	201	194	204	267	397	653	819	786	...	869	913	989	997	885	746	613	497	472	325
Sat	375	301	263	260	224	231	257	391	459	640	...	789	796	848	757	778	696	628	572	506	467
Sun	383	306	286	268	242	240	300	402	483	620	...	684	691	663	714	670	655	537	461	415	330
Thu	278	202	233	159	182	203	362	570	777	828	...	876	969	935	1013	810	698	617	553	424	354

5 rows × 24 columns

Now create a HeatMap using this new DataFrame.

In [35]:
```
sns.heatmap(dayHour, cmap = 'viridis');
```



Now create a clustermap using this DataFrame.

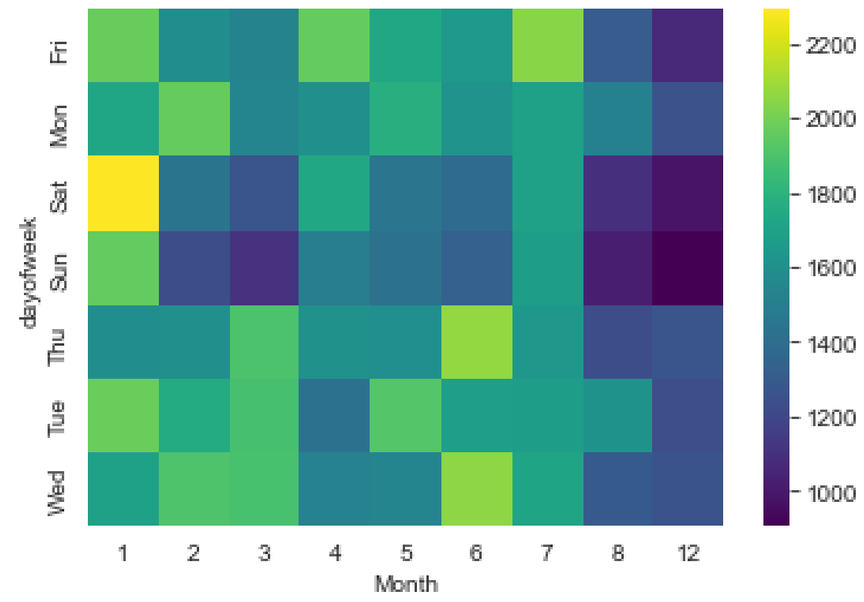
In [37]:
```
sns.clustermap(dayHour, cmap = 'viridis');
```

1000
800
600
400
200
Tue
Mon
Wed
Fri
dayofweek
Thu
Sat
Sun

Now repea **nn.**

In [31]:
```
daymonth = df.groupby(by= ['dayofweek','Month']).count()['Reason'].unstack()
```

In [36]:
```
sns.heatmap(daymonth, cmap ='viridis');
```



In [34]:
```
sns.clustermap(daymonth, cmap = 'viridis');
```

2250
2000
1750
1500
1250
1000
Sat
Sun
Fri
Thu
Wed
Mon
Tue
dayofweek

Continue exploring the Data however you see fit!

Great Job!