911 Calls Data Report

For this project I 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)

Data and Setup

Importing numpy and pandas

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

Importing visualization libraries and set %matplotlib inline

In [3]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline
```

Reading in the csv file as a dataframe called df

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

Checking the info() of the df

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 663522 entries, 0 to 663521
Data columns (total 9 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   lat        663522 non-null  float64
 1   lng        663522 non-null  float64
 2   desc       663522 non-null  object 
 3   zip        583323 non-null  float64
 4   title      663522 non-null  object 
 5   timeStamp  663522 non-null  object 
 6   twp        663229 non-null  object 
 7   addr       663522 non-null  object 
 8   e          663522 non-null  int64  
dtypes: float64(3), int64(1), object(5)
memory usage: 45.6+ MB
```

Checking the head of df

In [6]:
```
df.head(3)
```

Out[6]:

	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 17:10:52	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 17:29:21	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 14:39:21	NORRISTOWN	HAWS AVE	1

Answering Some Basic Questions

What are the top 5 zipcodes for 911 calls?

In [7]:
```
df['zip'].value_counts().head(5)
```

Out[7]:
```
19401.0    45606
19464.0    43910
19403.0    34888
19446.0    32270
19406.0    22464
Name: zip, dtype: int64
```

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

In [8]:
```
df['twp'].value_counts().head(5)
```

Out[8]:
```
LOWER MERION    55490
ABINGTON        39947
NORRISTOWN      37633
UPPER MERION    36010
CHELTENHAM      30574
Name: twp, dtype: int64
```

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

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

Out[9]:
```
148
```

Creating New Features

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

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

The most common Reason for a 911 call based off of this new column.

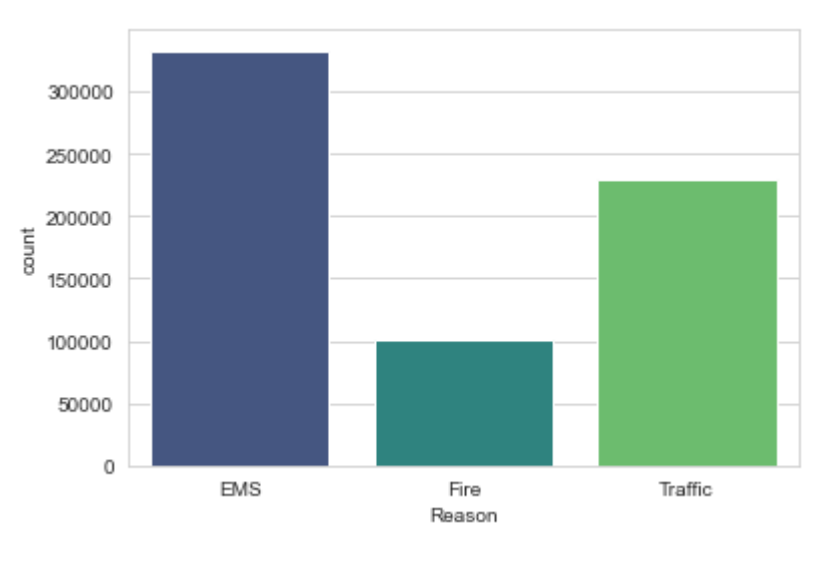
In [11]:
```
df['Reason'].value_counts()
```

Out[11]:
```
EMS        332692
Traffic    230208
Fire       100622
Name: Reason, dtype: int64
```

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

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

Out[12]:
```
<AxesSubplot:xlabel='Reason', ylabel='count'>
```



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

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

Out[13]:
```
str
```

These timestamps are still strings. Using pd.to_datetime to convert the column from strings to DateTime objects.

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

Grabbing specific attributes from a Datetime object by calling them.

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

Using Jupyter's tab method to explore the various attributes we can call. Now that the timestamp column are actually DateTime objects, using .apply() to create 3 new columns called Hour, Month, and Day of Week. I will create these columns based off of the timeStamp column.

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

The Day of Week is an integer 0-6. Using 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]:
```
dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}
```

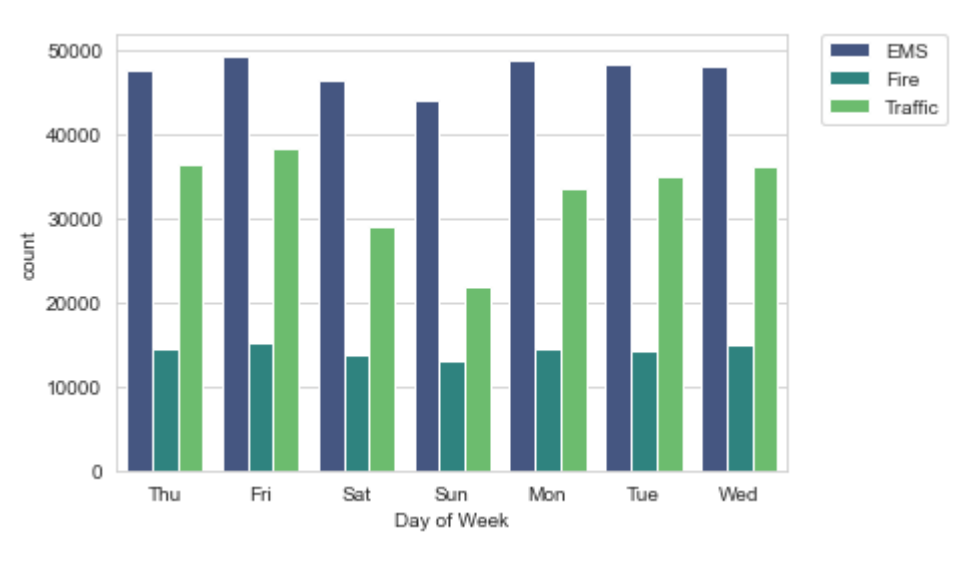
In [17]:
```
df['Day of Week'] = df['Day of Week'].map(dmap)
```

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

In [18]:
```
sns.countplot(x='Day of Week',data=df,hue='Reason',palette='viridis')

# To relocate the legend
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

Out[18]:
```
<matplotlib.legend.Legend at 0x1a44bc86d30>
```

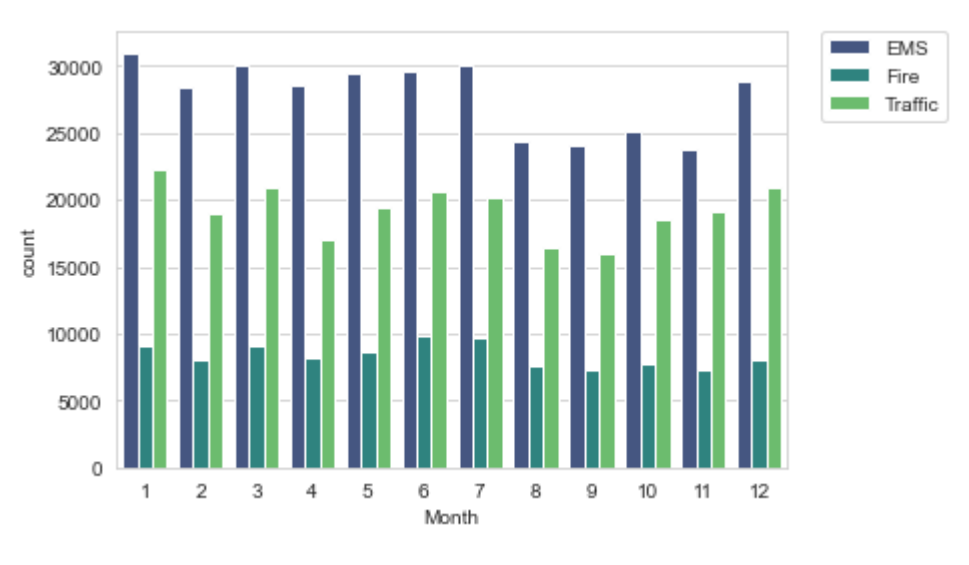


Now doing the same for Month:

In [19]:
```
sns.countplot(x='Month',data=df,hue='Reason',palette='viridis')

# To relocate the legend
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
```

Out[19]:
```
<matplotlib.legend.Legend at 0x1a44b5d32b0>
```



Did you notice something strange about the Plot?

In [20]:
```
# It is missing some months! 9,10, and 11 are not there.
```

We notice 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 creating a gropuby object called byMonth, where I group the DataFrame by the month column and use the count() method for aggregation. Using the head() method on this returned DataFrame.

In [21]:
```
byMonth = df.groupby('Month').count()
byMonth.head()
```

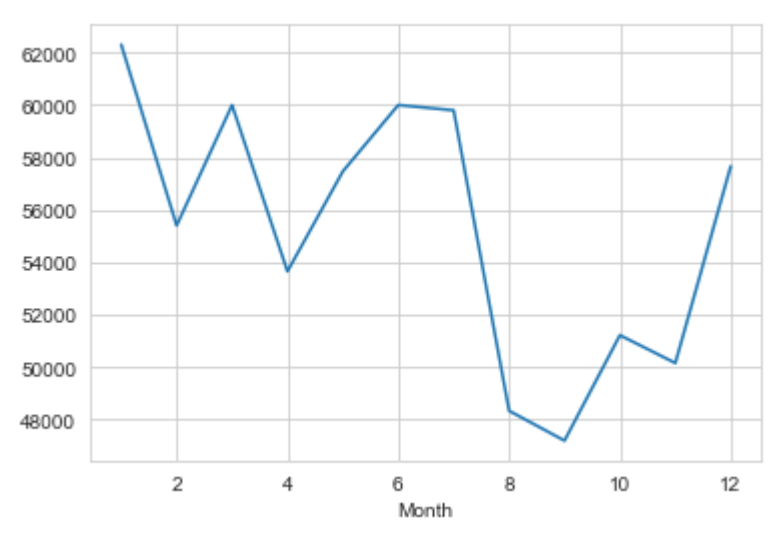
Out[21]:

Month	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Reason	Hour	Day of Week
1	62336	62336	62336	55294	62336	62336	62312	62336	62336	62336	62336	62336
2	55427	55427	55427	48922	55427	55427	55405	55427	55427	55427	55427	55427
3	60027	60027	60027	53252	60027	60027	60001	60027	60027	60027	60027	60027
4	53671	53671	53671	47349	53671	53671	53655	53671	53671	53671	53671	53671
5	57509	57509	57509	50354	57509	57509	57474	57509	57509	57509	57509	57509

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

In [22]:
```
# Could be any column
byMonth['twp'].plot()
```

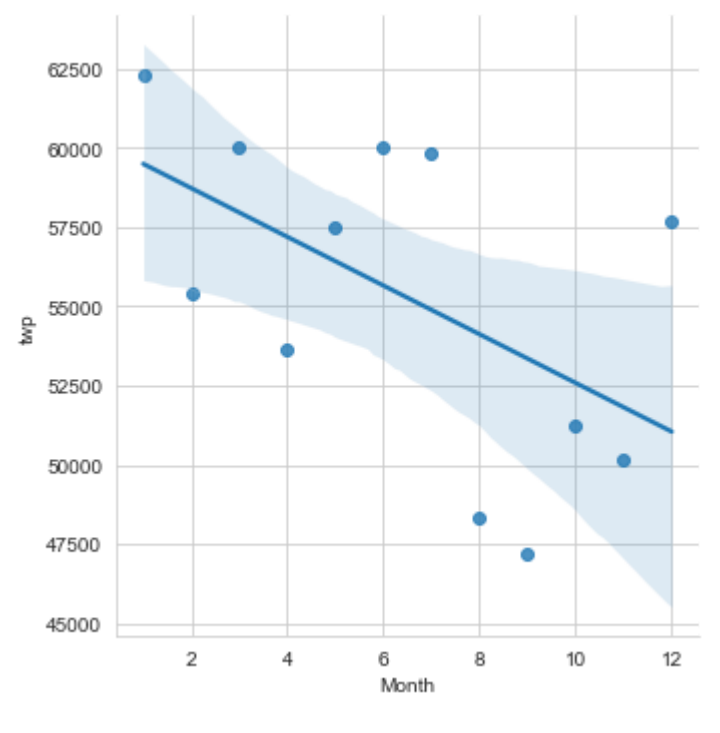
Out[22]:
```
<AxesSubplot:xlabel='Month'>
```



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

In [23]:
```
sns.lmplot(x='Month',y='twp',data=byMonth.reset_index())
```

Out[23]:
```
<seaborn.axisgrid.FacetGrid at 0x1a44e30c6d0>
```

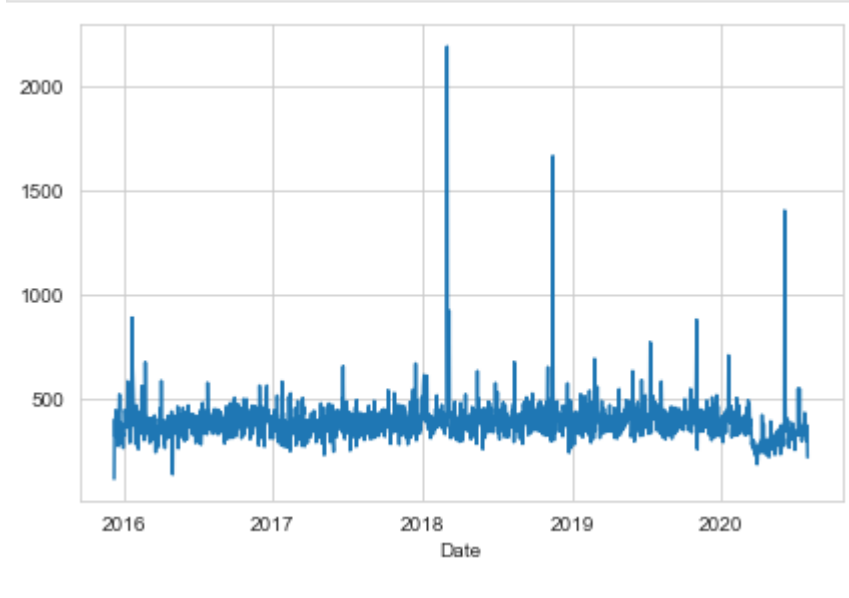


Creating a new column called 'Date' that contains the date from the timeStamp column. We'll need to use apply along with the .date() method.

In [24]:
```
df['Date']=df['timeStamp'].apply(lambda t: t.date())
```

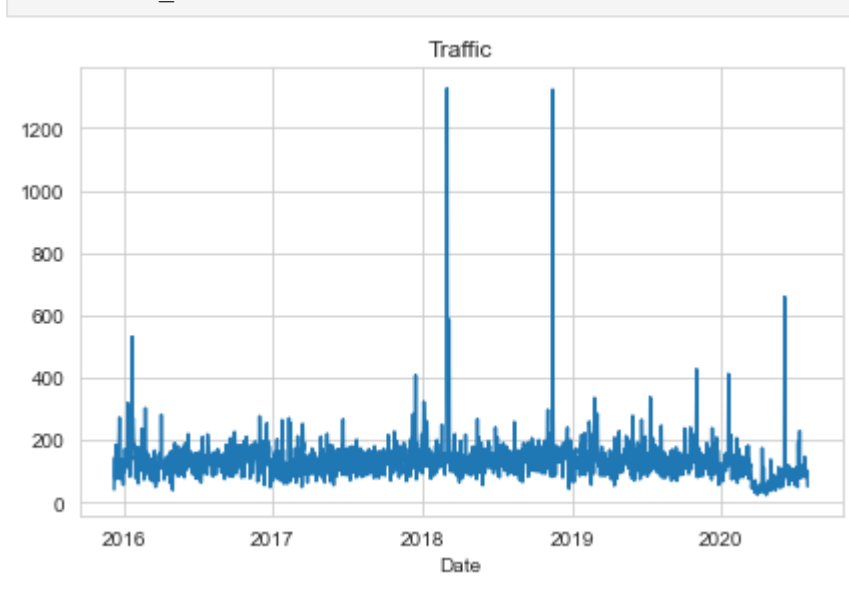
Now groupby this Date column with the count() aggregate and creating a plot of counts of 911 calls.

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

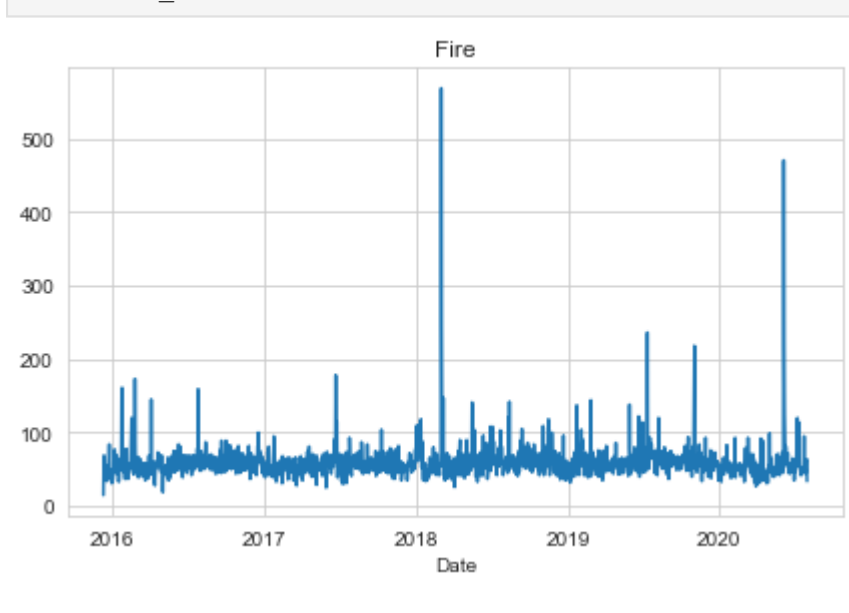


Now recreating this plot but creating 3 separate plots with each plot representing a Reason for the 911 call.

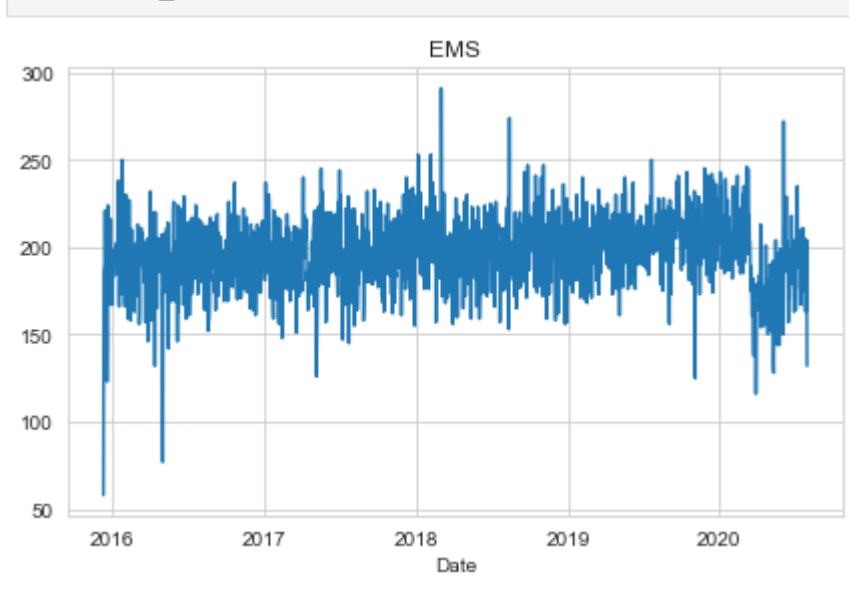
In [26]:
```
df[df['Reason']=='Traffic'].groupby('Date').count()['twp'].plot()
plt.title('Traffic')
plt.tight_layout()
```



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



In [28]:
```
df[df['Reason']=='EMS'].groupby('Date').count()['twp'].plot()
plt.title('EMS')
plt.tight_layout()
```



Now moving 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.

In [29]:
```
dayHour = df.groupby(by=['Day of Week','Hour']).count()['Reason'].unstack()
dayHour.head()
```

Out[29]:

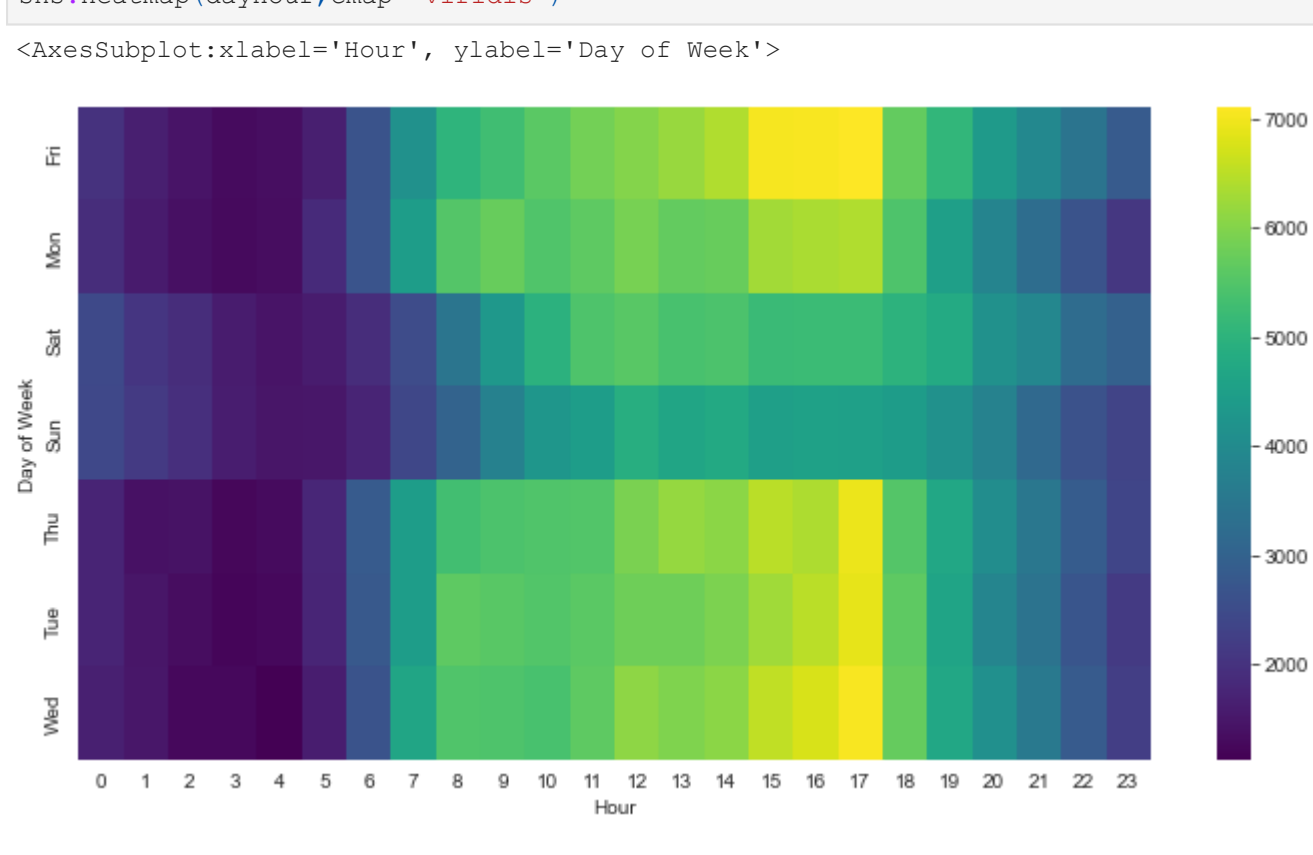
Day of Week \ Hour	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	23
Fri	1983	1635	1449	1296	1339	1639	2670	4143	5018	5288	...	6394	7040	7065	7113	5668	5056	4375	3913	3422	2834
Mon	1894	1571	1368	1272	1336	1844	2675	4430	5504	5724	...	5713	6289	6346	6408	5441	4488	3823	3254	2658	2072
Sat	2447	2059	1883	1592	1451	1580	1880	2489	3457	4315	...	5421	5181	5211	5213	4980	4753	4127	3895	3226	2965
Sun	2424	2135	1946	1614	1471	1488	1726	2408	3001	3728	...	4744	4475	4560	4505	4402	4135	3748	3161	2629	2323
Thu	1731	1408	1426	1236	1293	1775	2816	4432	5297	5412	...	6079	6493	6375	6935	5512	4703	4045	3490	2844	2354

5 rows × 24 columns

Now creating a HeatMap using this new DataFrame.

In [30]:
```
plt.figure(figsize=(12,6))
sns.heatmap(dayHour,cmap='viridis')
```

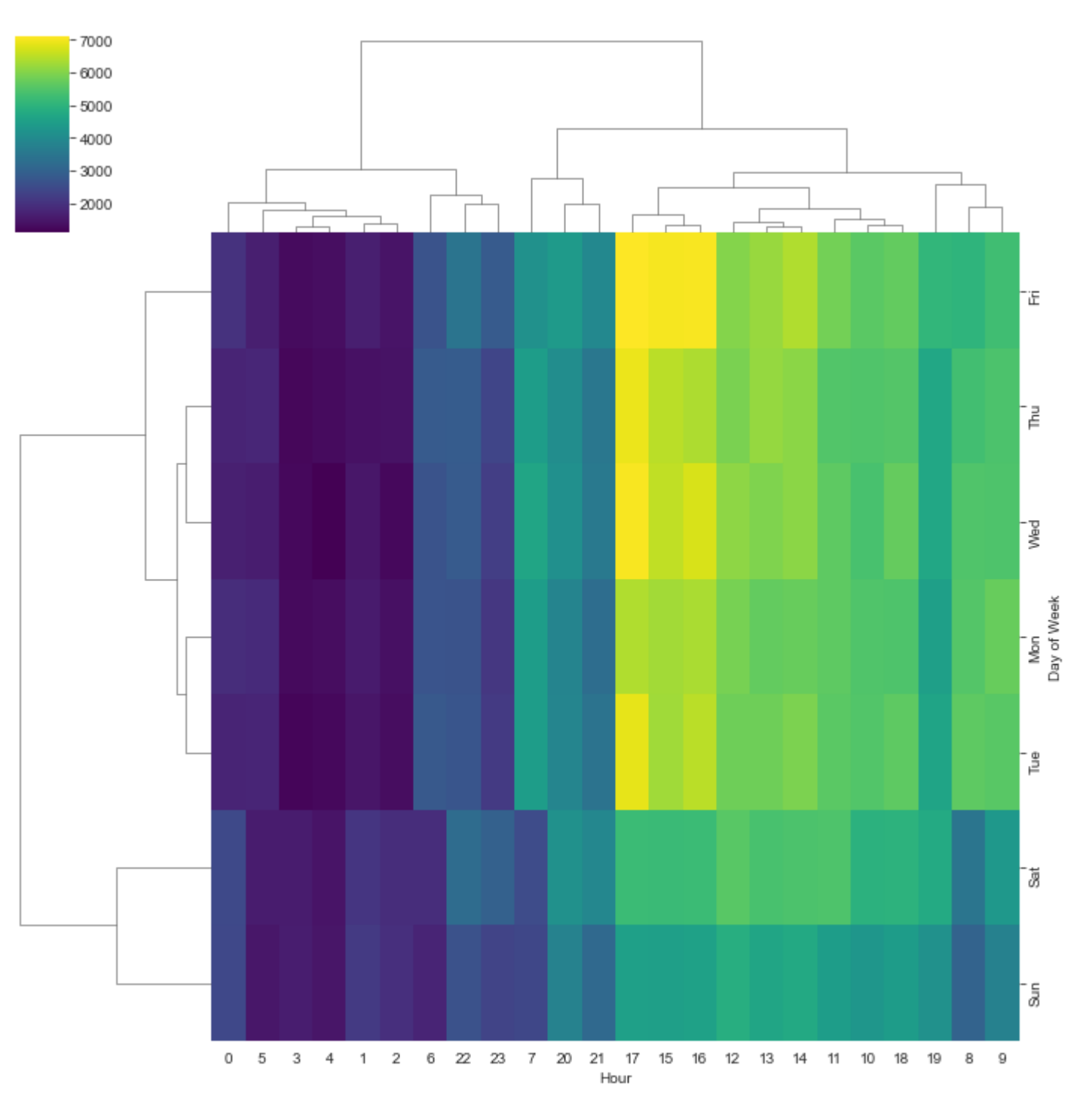
Out[30]:
```
<AxesSubplot:xlabel='Hour', ylabel='Day of Week'>
```



Now creating a clustermap using this DataFrame.

In [31]:
```
sns.clustermap(dayHour,cmap='viridis')
```

Out[31]:
```
<seaborn.matrix.ClusterGrid at 0x1a43c551610>
```



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

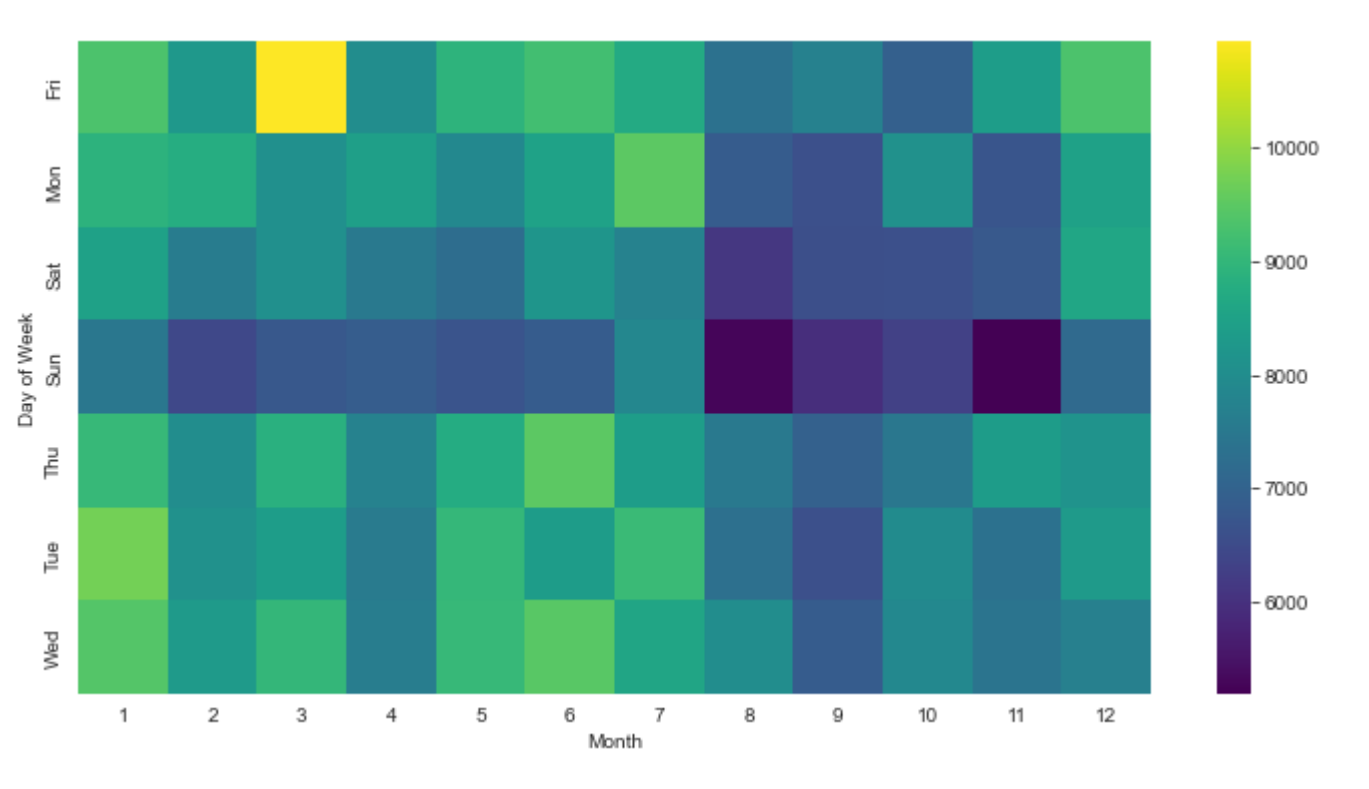
In [32]:
```
dayMonth = df.groupby(by=['Day of Week','Month']).count()['Reason'].unstack()
dayMonth.head()
```

Out[32]:

Day of Week \ Month	1	2	3	4	5	6	7	8	9	10	11	12
Fri	9309	8255	10941	7997	8904	9207	8681	7336	7694	6934	8379	9305
Mon	8896	8747	8060	8410	7881	8511	9499	6854	6598	8075	6722	8492
Sat	8475	7593	8050	7514	7223	8198	7748	6111	6566	6609	6773	8592
Sun	7478	6452	6766	6865	6694	6837	7859	5275	5956	6316	5196	7165
Thu	9055	7997	8849	7722	8740	9489	8378	7508	6954	7482	8358	8151

In [33]:
```
plt.figure(figsize=(12,6))
sns.heatmap(dayMonth,cmap='viridis')
```

Out[33]:
```
<AxesSubplot:xlabel='Month', ylabel='Day of Week'>
```



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

Out[34]:
```
<seaborn.matrix.ClusterGrid at 0x1a43c6c3a30>
```