02-911 Calls Data Capstone Project - Solutions

November 15, 2025

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 **

[35]:
```
import numpy as np
import pandas as pd
```

** Import visualization libraries and set %matplotlib inline. **

[36]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline
```

** Read in the csv file as a dataframe called df **

[37]:
```
df = pd.read_csv('911.csv')
```

** Check the info() of the df **

[38]:
```
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 **

[39]:
```
df.head(3)
```

[39]:
```
         lat        lng                                               desc  \
0  40.297876 -75.581294  REINDEER CT & DEAD END;  NEW HANOVER; Station …
1  40.258061 -75.264680  BRIAR PATH & WHITEMARSH LN;  HATFIELD TOWNSHIP…
2  40.121182 -75.351975  HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21-St…

       zip                    title            timeStamp                twp  \
0  19525.0   EMS: BACK PAINS/INJURY  2015-12-10 17:40:00        NEW HANOVER
1  19446.0  EMS: DIABETIC EMERGENCY  2015-12-10 17:40:00  HATFIELD TOWNSHIP
2  19401.0      Fire: GAS-ODOR/LEAK  2015-12-10 17:40:00         NORRISTOWN

                         addr  e
0      REINDEER CT & DEAD END  1
1  BRIAR PATH & WHITEMARSH LN  1
2                    HAWS AVE  1
```

1.2 Basic Questions

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

[40]:
```
df['zip'].value_counts().head(5)
```

[40]:
```
zip
19401.0    6979
19464.0    6643
19403.0    4854
19446.0    4748
19406.0    3174
Name: count, dtype: int64
```

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

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

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

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

```
[42]: df['title'].nunique()
```

```
[42]: 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.

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

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

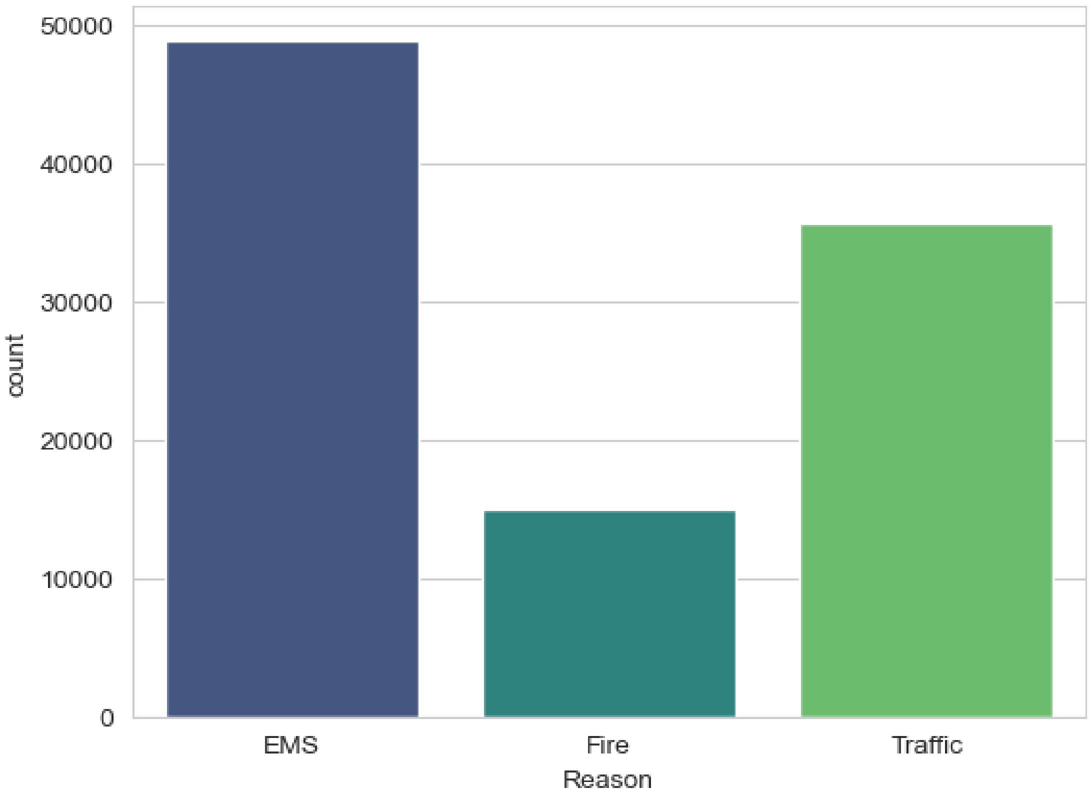
```
[44]: df['Reason'].value_counts()
```

```
[44]: Reason
      EMS        48877
      Traffic    35695
      Fire       14920
      Name: count, dtype: int64
```

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

```
[45]: sns.countplot(x='Reason',data=df,palette='viridis')
      plt.show()
```

```
C:\Users\cakaj\AppData\Local\Temp\ipykernel_93544\260351650.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x='Reason',data=df,palette='viridis')
```



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

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

```
[46]: str
```

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

```
[47]: 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.

[48]:

```
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)
```

** 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'}

[49]:

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

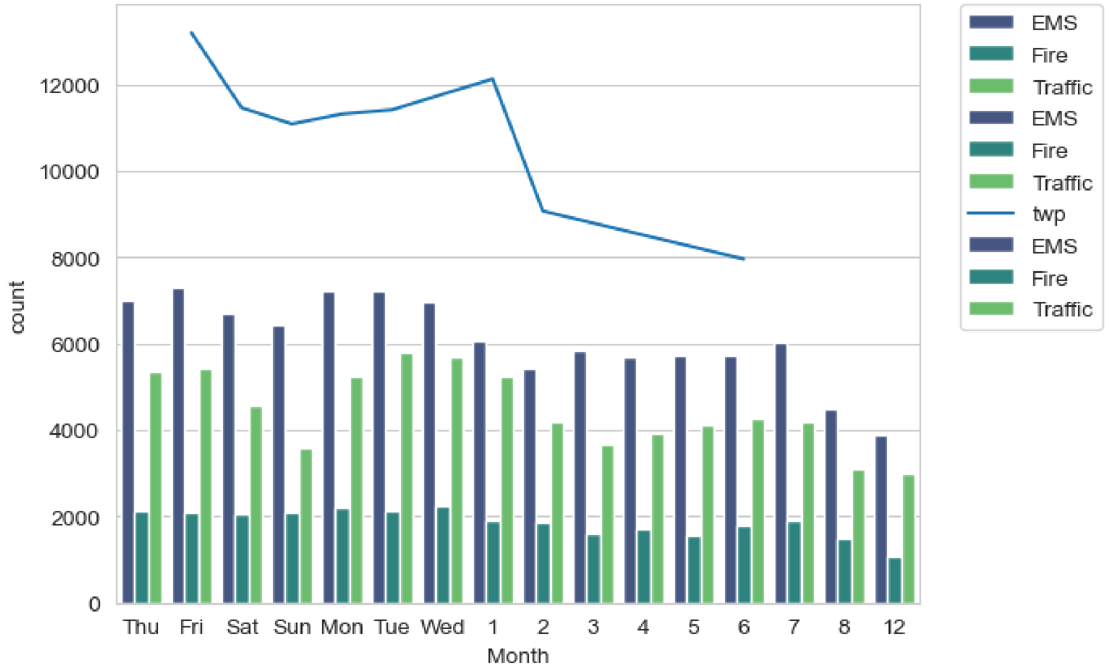
[50]:

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

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

[56]:

```
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.)
plt.show()
```

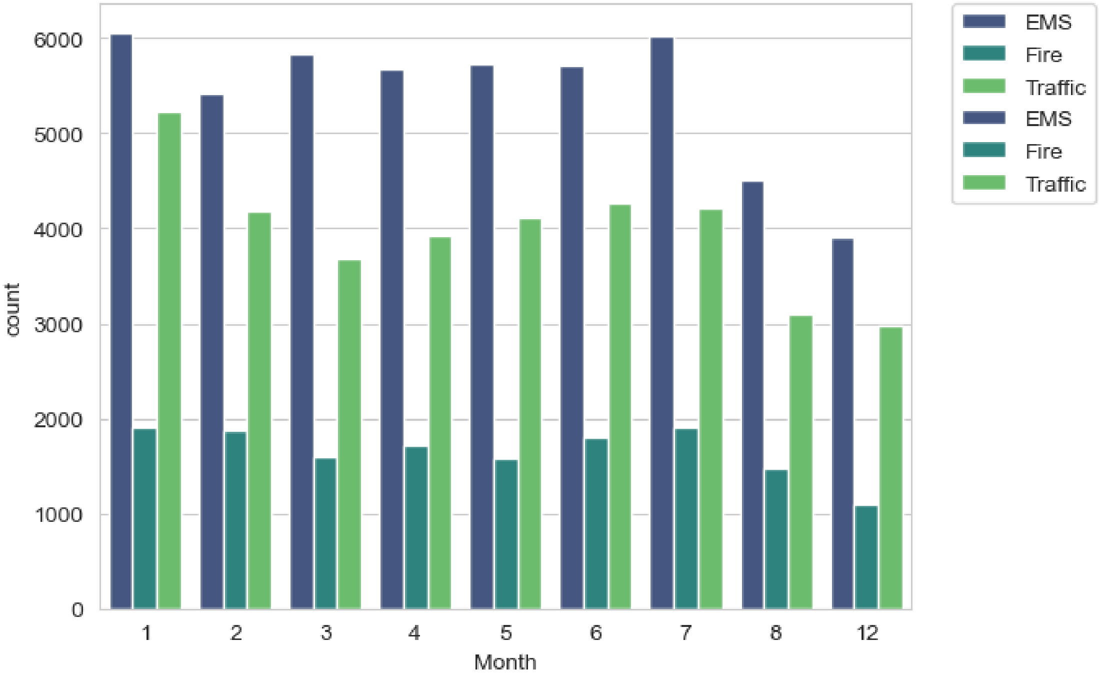


** Now do the same for Month:**

[59]:

```
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.)
plt.show()
```



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

[60]:
```
# It is missing some months! 9,10, and 11 are not there.
```

** 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. **

[61]:
```
byMonth = df.groupby('Month').count()
byMonth.head()
```

[61]:
```
         lat    lng   desc    zip  title  timeStamp    twp   addr      e  \
Month
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
```

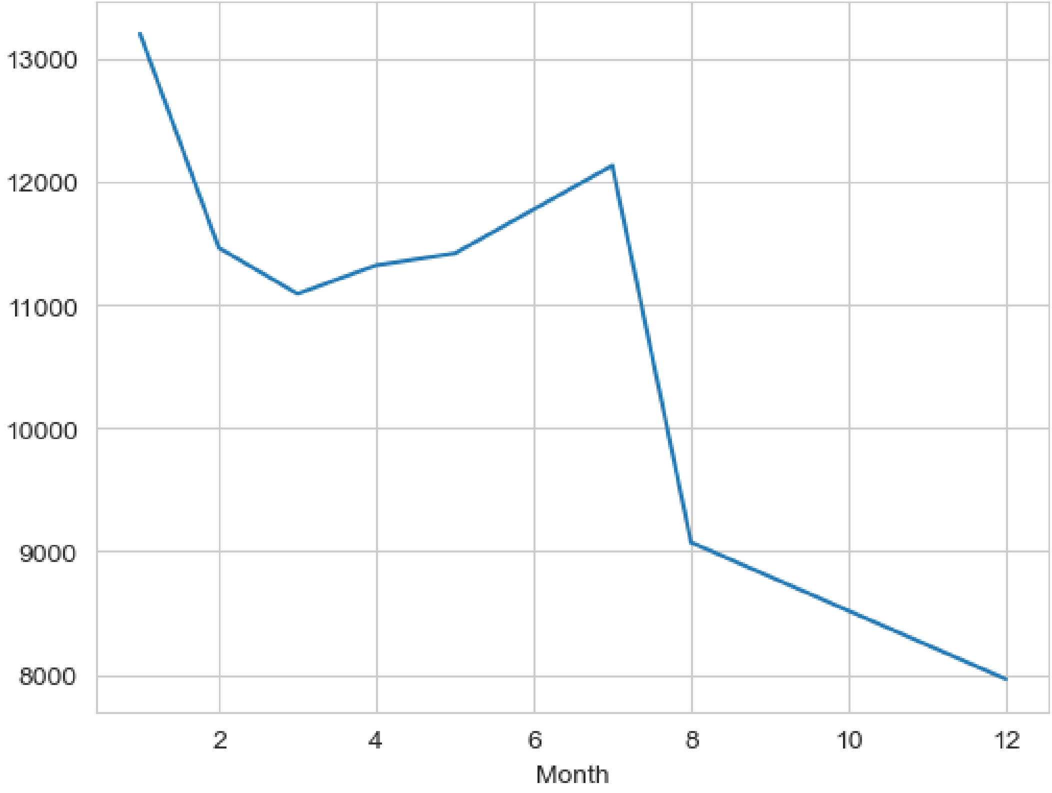
** Now create a simple plot off of the dataframe indicating the count of calls per month. **

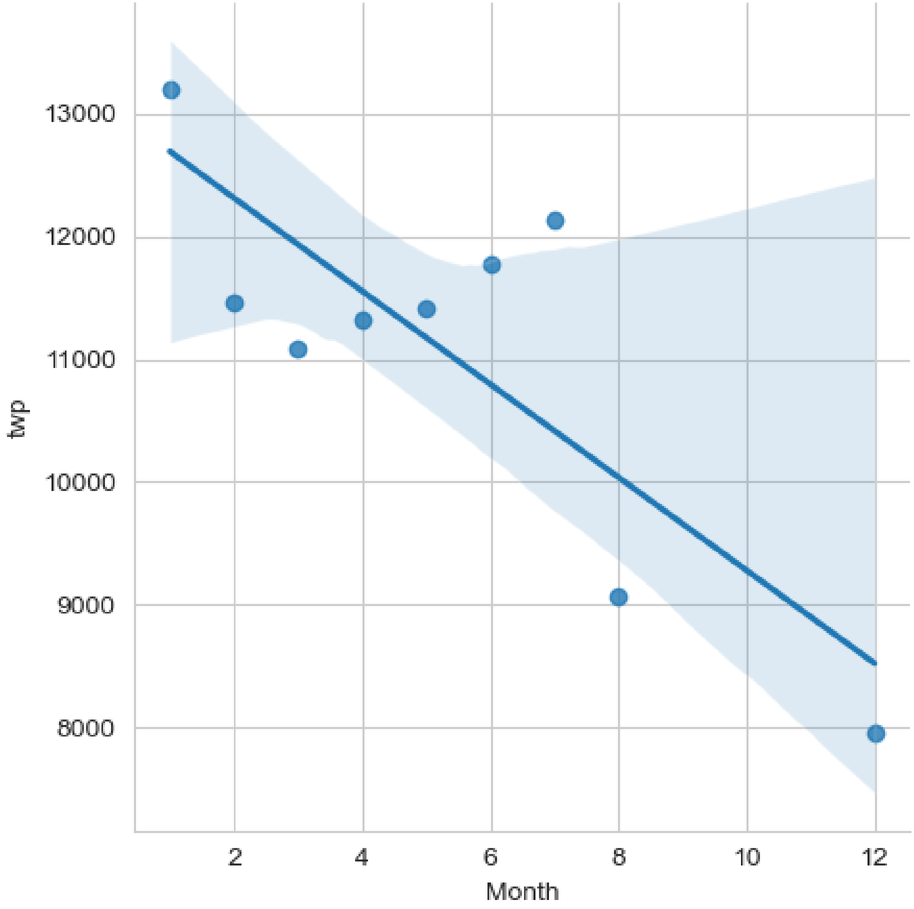
```
[62]: # Could be any column
      byMonth['twp'].plot()
```

```
[62]: <Axes: xlabel='Month'>
```

** 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. **

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



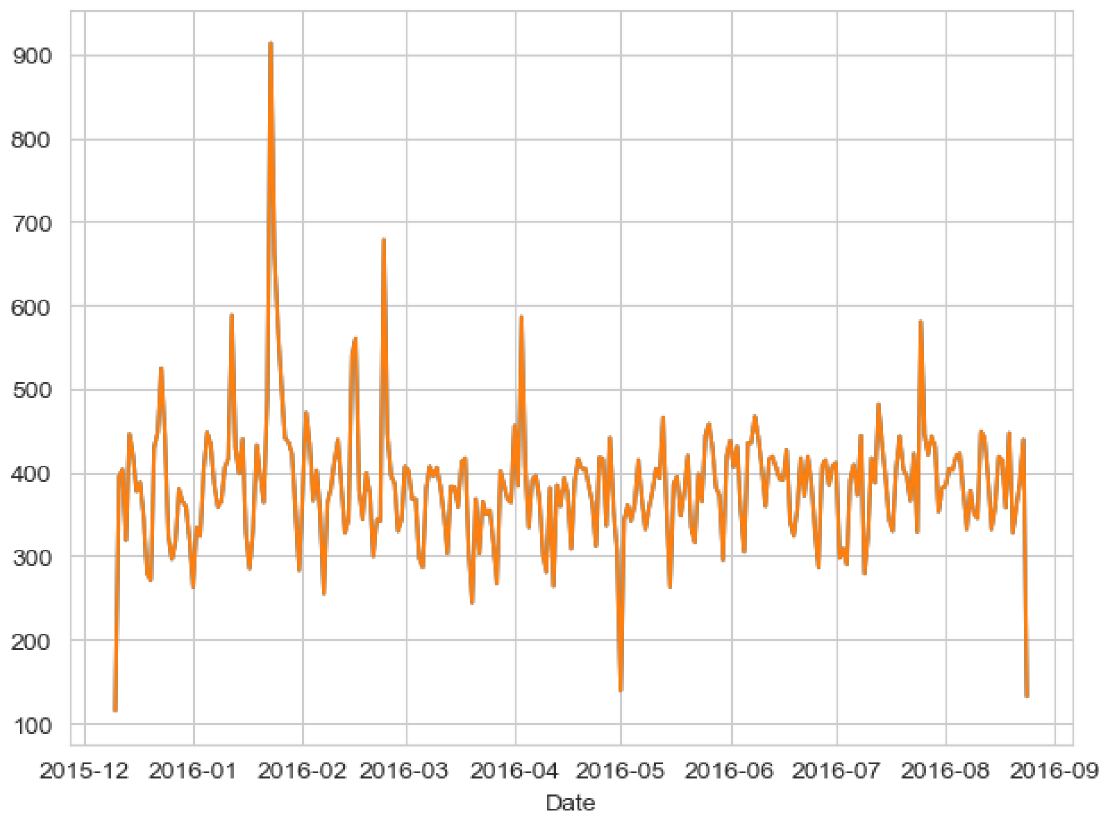


Create a new column called 'Date' that contains the date from the timeStamp column. You'll need to use apply along with the .date() method.

[64]:
```
df['Date']=df['timeStamp'].apply(lambda t: t.date())
```

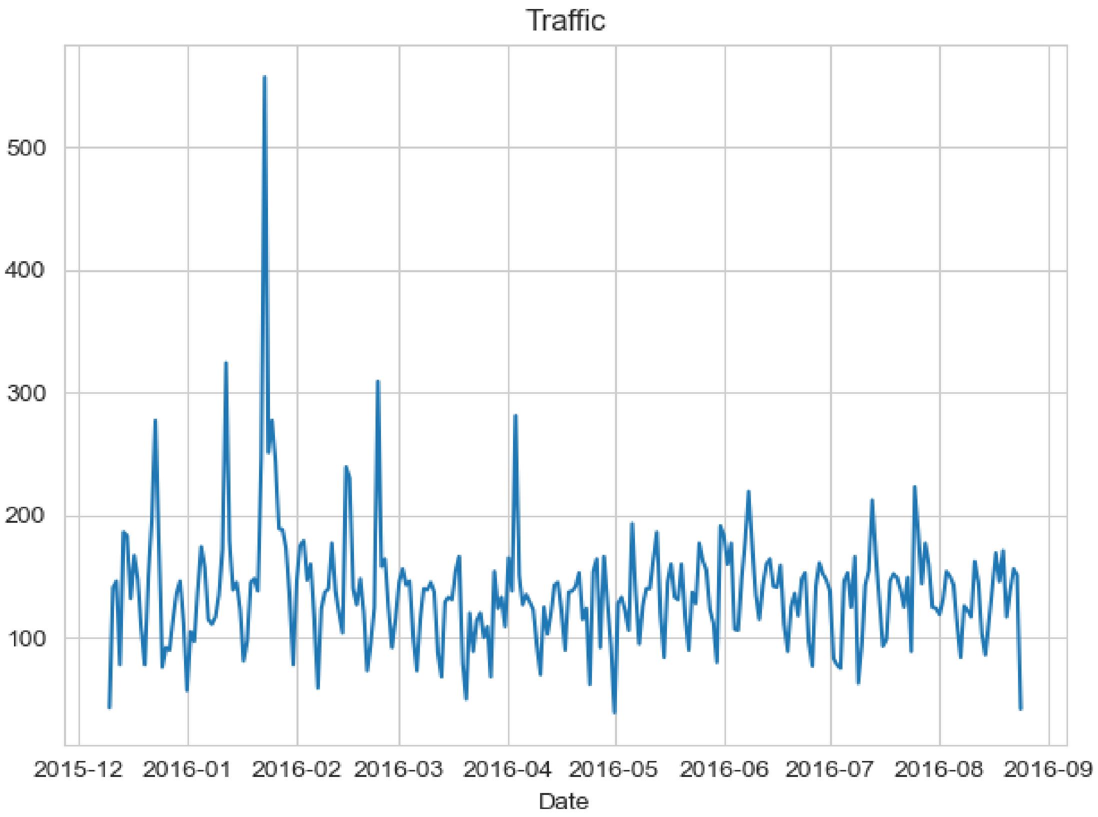
** Now groupby this Date column with the count() aggregate and create a plot of counts of 911 calls.**

[66]:
```
df.groupby('Date').count()['twp'].plot()
plt.tight_layout()
plt.show()
```

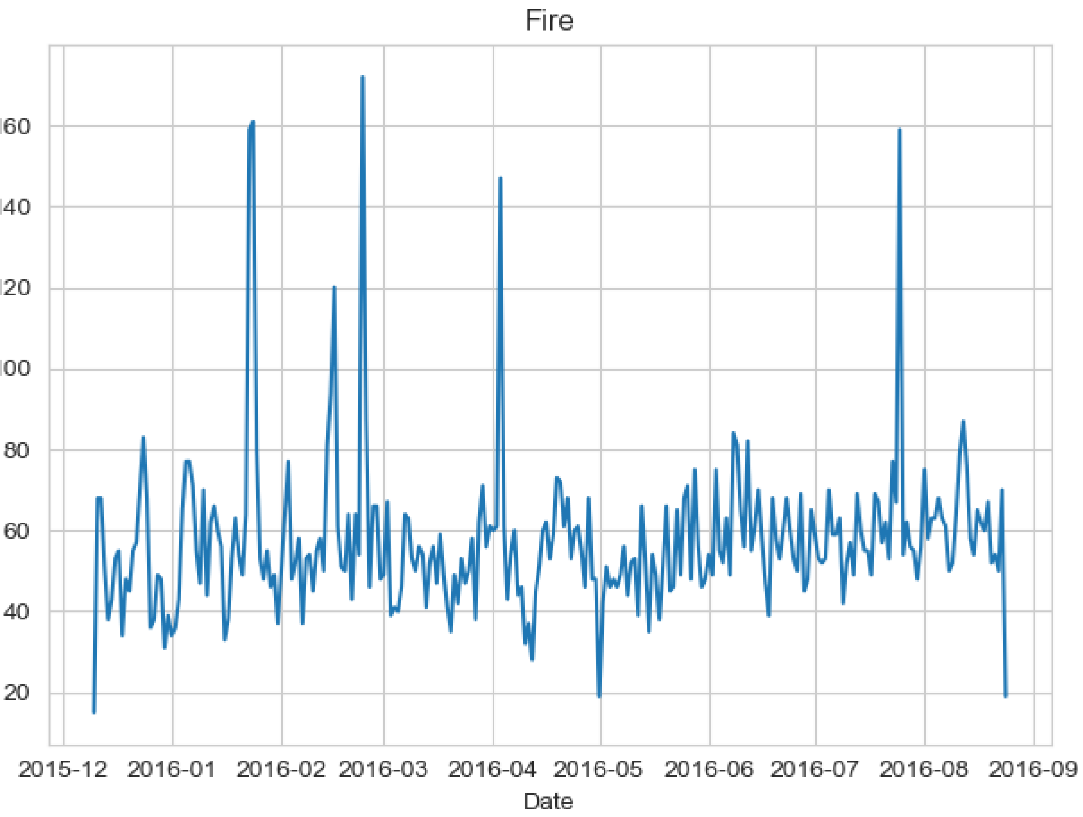


** Now recreate this plot but create 3 separate plots with each plot representing a Reason for the 911 call**

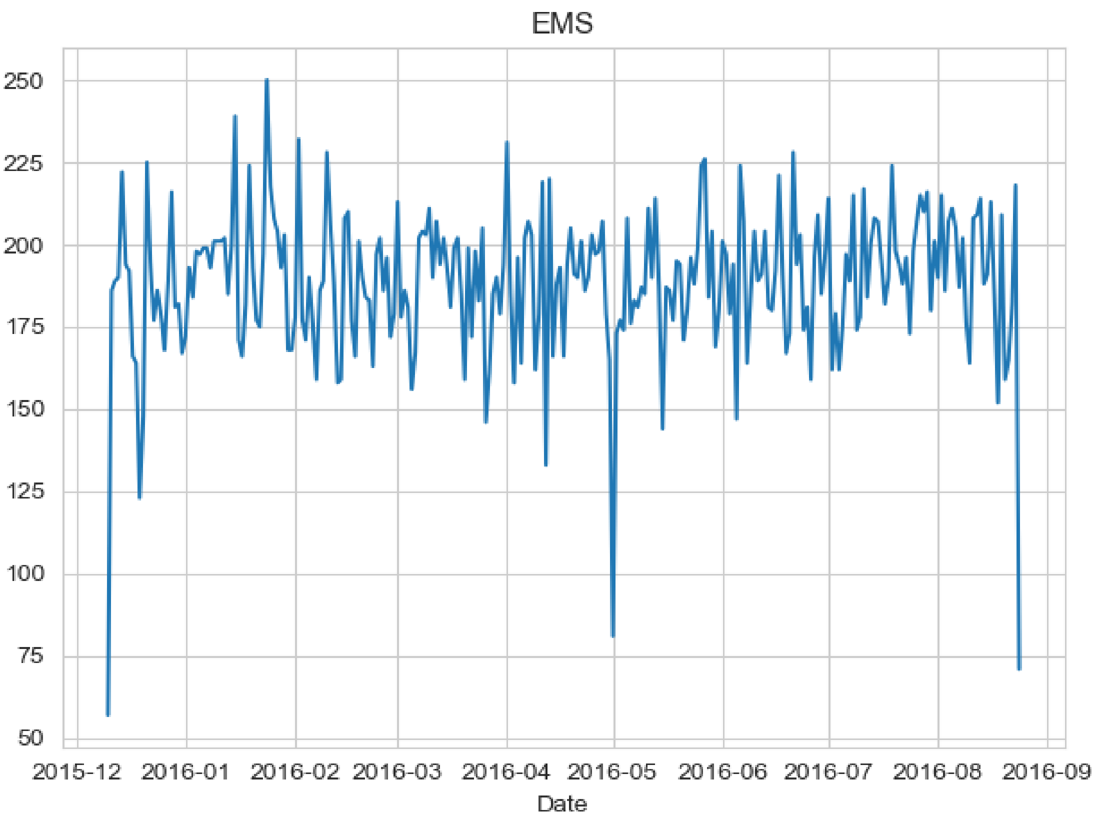
```
[67]: df[df['Reason']=='Traffic'].groupby('Date').count()['twp'].plot()
plt.title('Traffic')
plt.tight_layout()
plt.show()
```



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



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



** 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!**

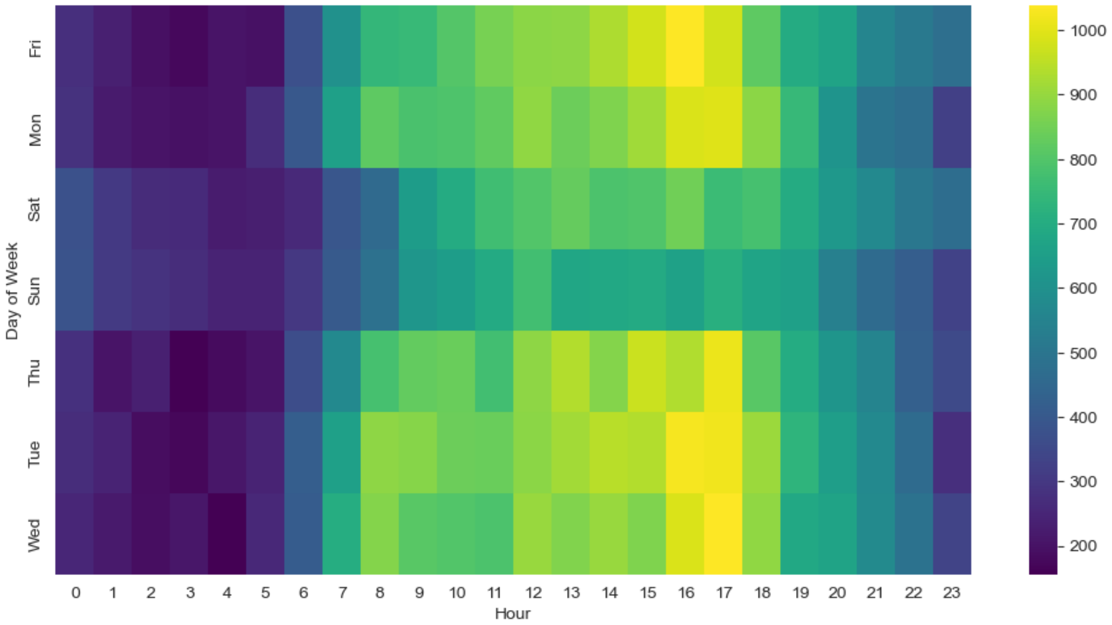
[70]:
```
dayHour = df.groupby(by=['Day of Week','Hour']).count()['Reason'].unstack()
dayHour.head()
```

[70]:
```
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]
```

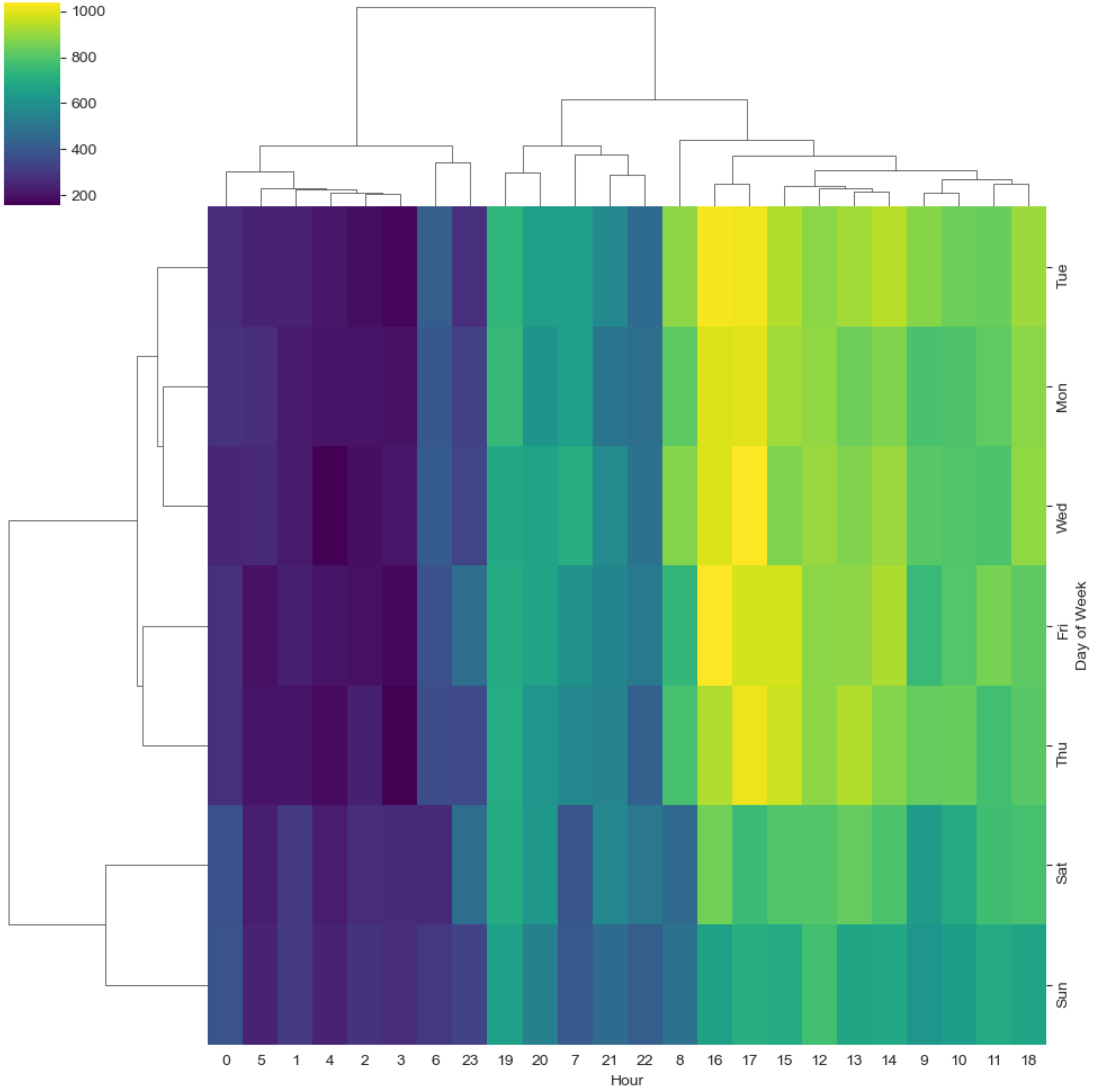
** Now create a HeatMap using this new DataFrame. **

[71]:
```
plt.figure(figsize=(12,6))
sns.heatmap(dayHour,cmap='viridis')
plt.show()
```



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

[72]:
```
sns.clustermap(dayHour,cmap='viridis')
plt.show()
```

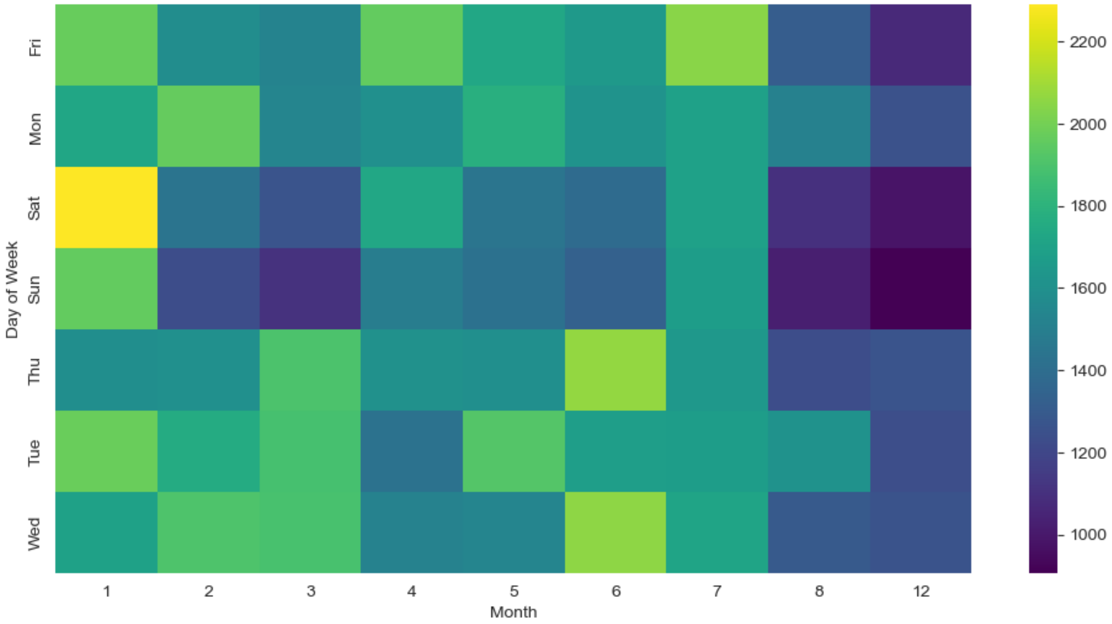


** Now repeat these same plots and operations, for a DataFrame that shows the Month as the column. **

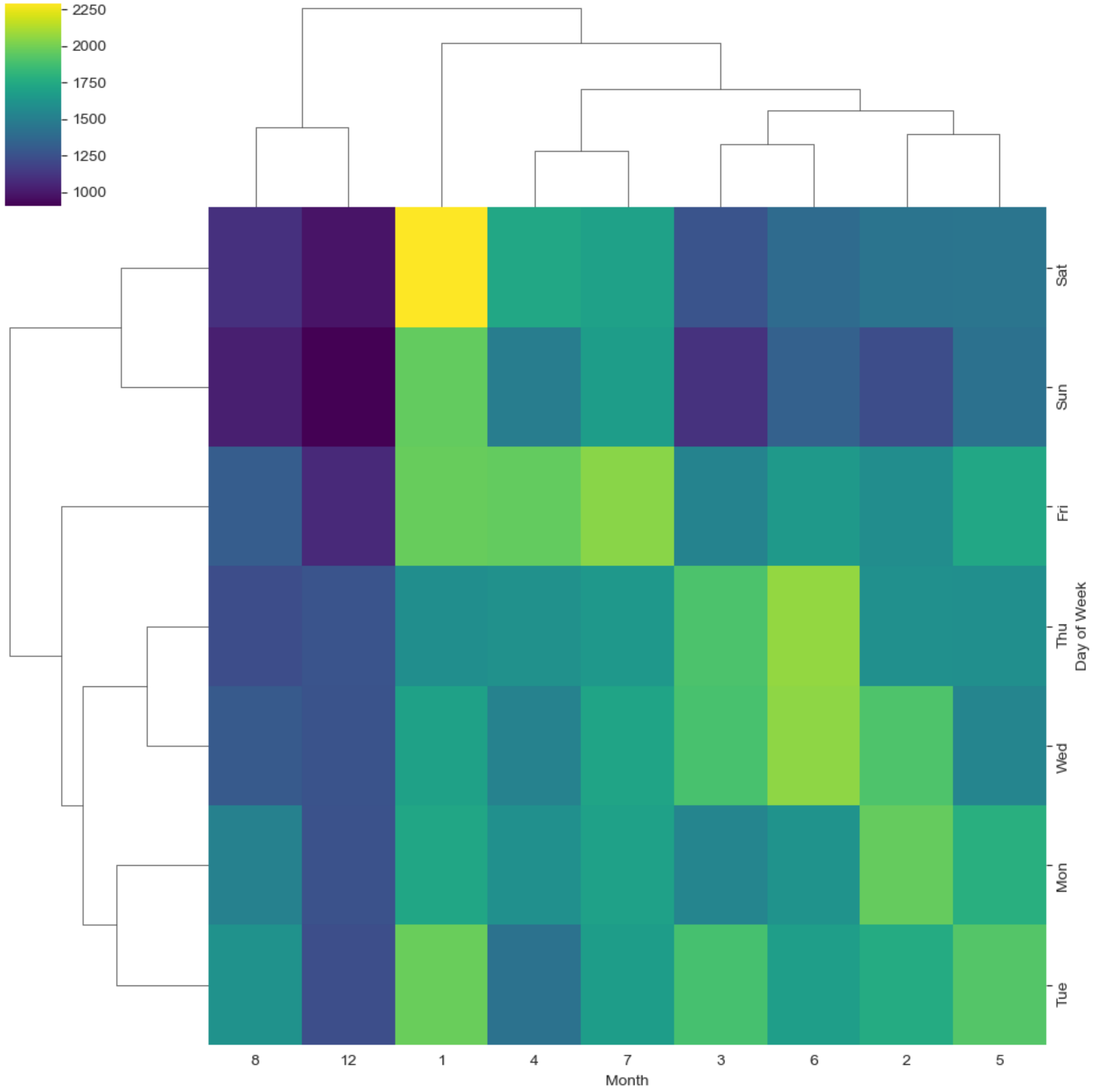
[73]:
```
dayMonth = df.groupby(by=['Day of Week','Month']).count()['Reason'].unstack()
dayMonth.head()
```

[73]:
```
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
```

[74]:
```
plt.figure(figsize=(12,6))
sns.heatmap(dayMonth,cmap='viridis')
plt.show()
```



[75]:
```
sns.clustermap(dayMonth,cmap='viridis')
plt.show()
```



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