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!

Data and Setup

Import numpy and pandas

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

Import visualization libraries and set %matplotlib inline.

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

Read in the csv file as a dataframe called df

In [3]:
```
df = pd.read_csv('911.csv')
```

Check the info() of the df

In [4]:
```
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 [5]:
```
df.head()
```

Out[5]:

	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 [9]:
```
# So from the df zip, value_counts will add how many instances
# the zip code appears in descending order so it will sort for you.
#  Now the head(5) component will help you get the top 5

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

Out[9]:
```
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 [12]:
```
#Just repeat the same reasoning here

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

Out[12]:
```
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 [14]:
```
df['title'].nunique()
```

Out[14]:
```
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 [21]:
```
#First let's paint it out,

df['title'].iloc[0]
```

Out[21]:
```
'EMS: BACK PAINS/INJURY'
```

In [18]:
```
#Set df to x and split it by ':'

x = df['title'].iloc[0]

x.split(':')
```

Out[18]:
```
['EMS', ' BACK PAINS/INJURY']
```

In [22]:
```
#Use indexing location

x.split(':')[0]
```

Out[22]:
```
'EMS'
```

In [23]:
```
#Let's create our lambda expression
```

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

In [24]:
```
#We finally have our new column!

df['Reason']
```

Out[24]:
```
0            EMS
1            EMS
2           Fire
3            EMS
4            EMS
          ...
99487    Traffic
99488    Traffic
99489        EMS
99490        EMS
99491    Traffic
Name: Reason, Length: 99492, dtype: object
```

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

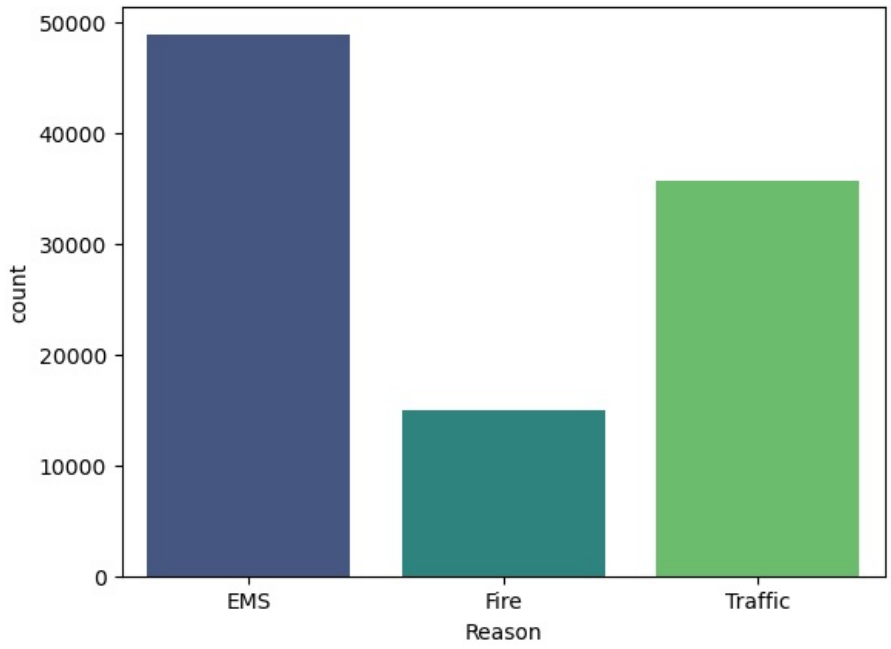
In [26]:
```
df['Reason'].value_counts()
```

Out[26]:
```
EMS        48877
Traffic    35695
Fire       14920
Name: Reason, dtype: int64
```

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

In [30]:
```
sns.countplot(x='Reason',data=df,palette='viridis') #Don't worry about the color it's from solutions
```

Out[30]:
```
<AxesSubplot:xlabel='Reason', ylabel='count'>
```



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

In [31]:
```
#Our info says that timeStamp is an object not a number
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99492 entries, 0 to 99491
Data columns (total 10 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
 9   Reason     99492 non-null  object
dtypes: float64(3), int64(1), object(6)
memory usage: 7.6+ MB
```

In [33]:
```
#Let's check one of it's numbers
```

```
df['timeStamp'].iloc[0] # It is indeed not a number
```

Out[33]:
```
'2015-12-10 17:40:00'
```

In [35]:
```
#So to get the type, you do:

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

Out[35]:
```
str
```

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

In [36]:
```
df['timeStamp'] = pd.to_datetime(df['timeStamp'])
```

In [37]:
```
#Let's recheck our code

type(df['timeStamp'].iloc[0]) #Now it says timestamp
```

Out[37]:
```
pandas._libs.tslibs.timestamps.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 [42]:
```
#With timestamps you can actually call alot of functions
# Like .year, .hour, .month just check shift+tab for more methods
# Here're a few demonstrations

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

In [43]:
```
#It clearly says timestamp so now we can get to work
time
```

Out[43]:
```
Timestamp('2015-12-10 17:40:00')
```

In [ ]:
```
#Example 2

time.year
```

In [39]:
```
#Example 3

time.dayofweek
```

Out[39]:
```
3
```

In [40]:
```
#Now let's set our 3 new columns. Let's start by hour

df['Hour'] = df ['timeStamp'].apply(lambda time: time.hour)
```

In [44]:
```
#Let's finish with Month and Day of the Week

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

In [45]:
```
#All the new columns are set!

df.head()
```

Out[45]:

	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Reason	Hour	Month	Day o 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	
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	

In [ ]:

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 [46]:
```
dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}
```

In [47]:
```
#Let's set Day of week!

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

In [48]:
```
#Let's check it

df.head()
```

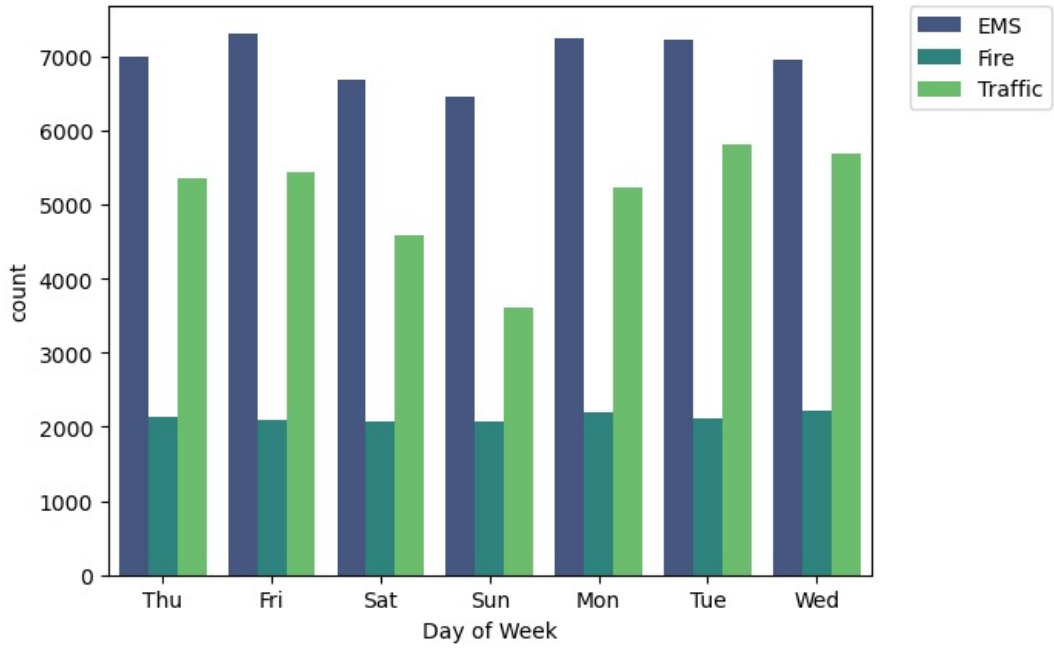
Out[48]:

	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Reason	Hour	Month	Day o 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
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	Thu
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	Thu

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

In [52]:
```
sns.countplot(x='Day of Week',data= df,hue='Reason',palette='viridis')

# The extra code (from before) to relocate the legend box out of the graph is;

plt.legend(bbox_to_anchor=(1.05,1), loc=2 , borderaxespad=0) #Keep this code
```

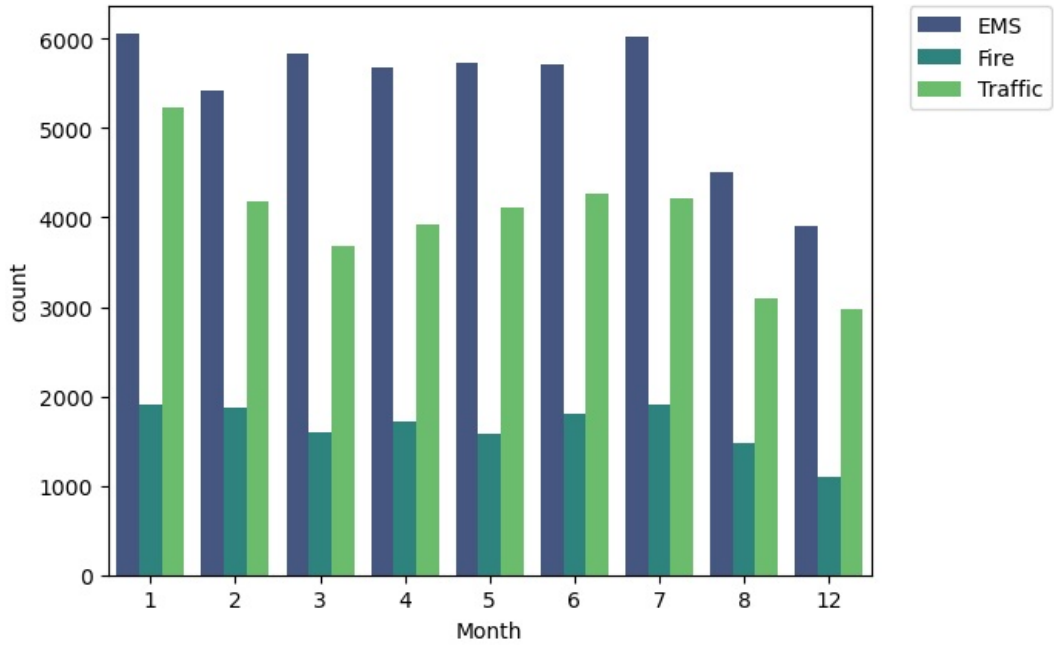
Out[52]:
```
<matplotlib.legend.Legend at 0x27148d8d760>
```



Now do the same for Month:

In [56]:
```
sns.countplot(x='Month', data=df, hue='Reason',palette= 'viridis')

# Extra code for the legend that you should always keep on the side, is

plt.legend(bbox_to_anchor=(1.05,1), loc=2 , borderaxespad=0)
```

Out[56]:
```
<matplotlib.legend.Legend at 0x27148eb9dc0>
```



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 groupby 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 [58]:
```
#Groupby month and add each activity up to see the amount of traffic
# or action each month had

byMonth = df.groupby('Month').count()
```

In [59]:
```
byMonth.head()
```

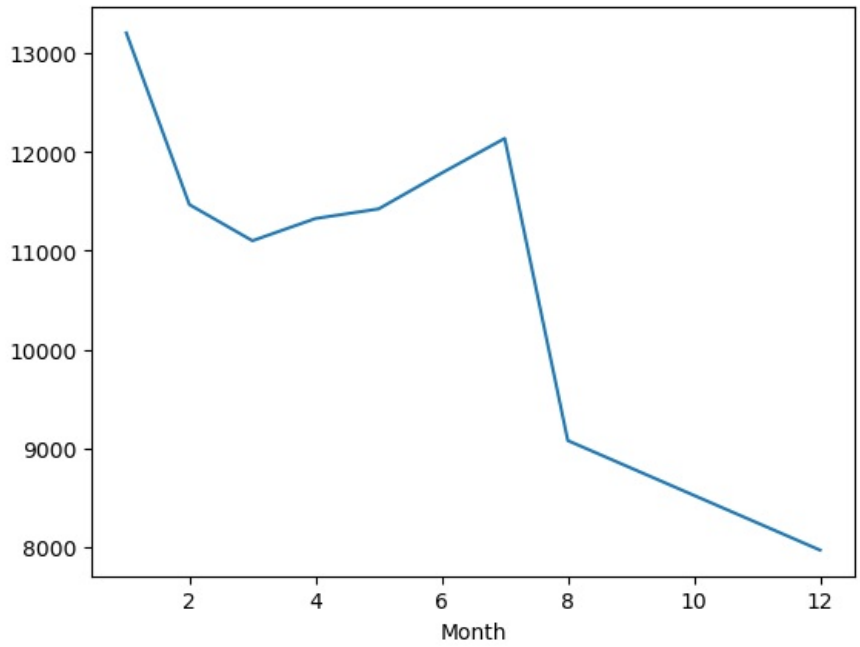
Out[59]:

Month	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Reason	Hour	Day of Week
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 [60]:
```
#In this instance we could've used either lat, long or desc
# This was just made for us to see the amount of call activity in each month
#  This is accurate because each registered 911 call had a lat, long and desc
#   Reason woulda worked too

byMonth['lat'].plot()
```

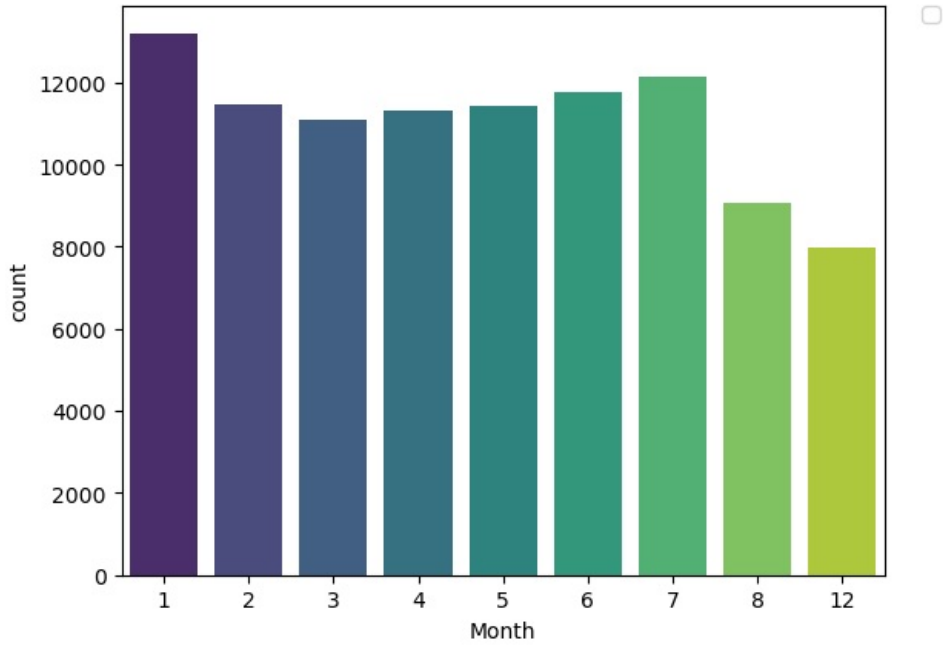
Out[60]:
```
<AxesSubplot:xlabel='Month'>
```



In [61]:
```
# Now let's try to see this again with countplots but no hue

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

# Extra code for the legend

plt.legend(bbox_to_anchor=(1.05,1), loc=2 , borderaxespad=0)

#Still omits the month, we just wanted to show the trend so in that case,
# a line plot is way better since it shows everything
```

```
No artists with labels found to put in legend.  Note that artists whose label start with an underscore are ignored when legend() is called with no argument.
```

Out[61]:
```
<matplotlib.legend.Legend at 0x2714a1f9130>
```



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.

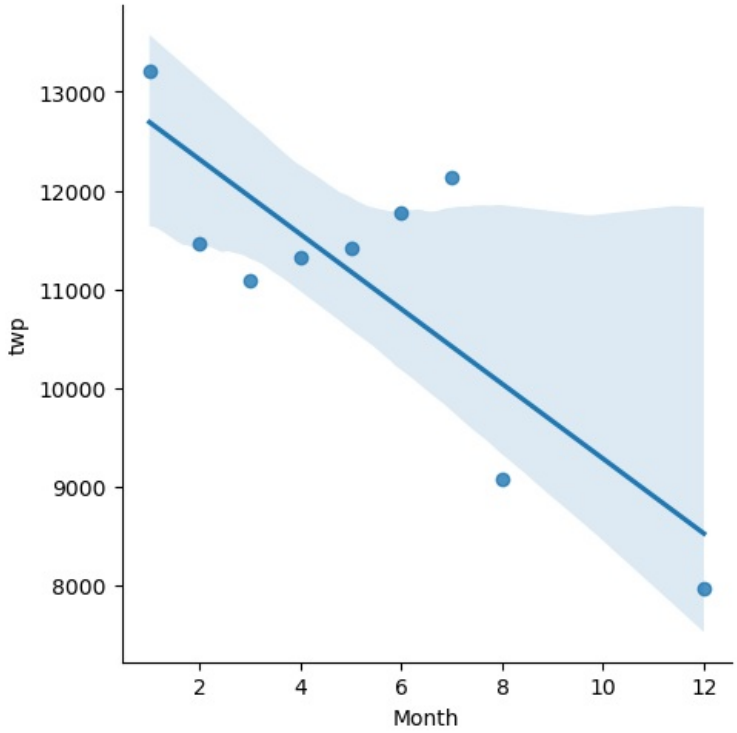
In [62]:
```
byMonth.reset_index()
```

Out[62]:

	Month	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Reason	Hour	Day of Week
0	1	13205	13205	13205	11527	13205	13205	13203	13096	13205	13205	13205	13205
1	2	11467	11467	11467	9930	11467	11467	11465	11396	11467	11467	11467	11467
2	3	11101	11101	11101	9755	11101	11101	11092	11059	11101	11101	11101	11101
3	4	11326	11326	11326	9895	11326	11326	11323	11283	11326	11326	11326	11326
4	5	11423	11423	11423	9946	11423	11423	11420	11378	11423	11423	11423	11423
5	6	11786	11786	11786	10212	11786	11786	11777	11732	11786	11786	11786	11786
6	7	12137	12137	12137	10633	12137	12137	12133	12088	12137	12137	12137	12137
7	8	9078	9078	9078	7832	9078	9078	9073	9025	9078	9078	9078	9078
8	12	7969	7969	7969	6907	7969	7969	7963	7916	7969	7969	7969	7969

In [64]:
```
sns.lmplot(x='Month',y='twp',data=byMonth.reset_index())
```

Out[64]:
```
<seaborn.axisgrid.FacetGrid at 0x271490069a0>
```



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 [65]:
```
t = df['timeStamp'].iloc[0]
```

In [66]:
```
t
```

Out[66]:
```
Timestamp('2015-12-10 17:40:00')
```

In [68]:
```
#Calling .date() will transform it into a datetime

t.date()
```

Out[68]:
```
datetime.date(2015, 12, 10)
```

In [70]:
```
#Now let's say

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

In [71]:
```
df.head()
```

Out[71]:

	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Reason	Hour	Month	Day o 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
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	Thu
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	Thu

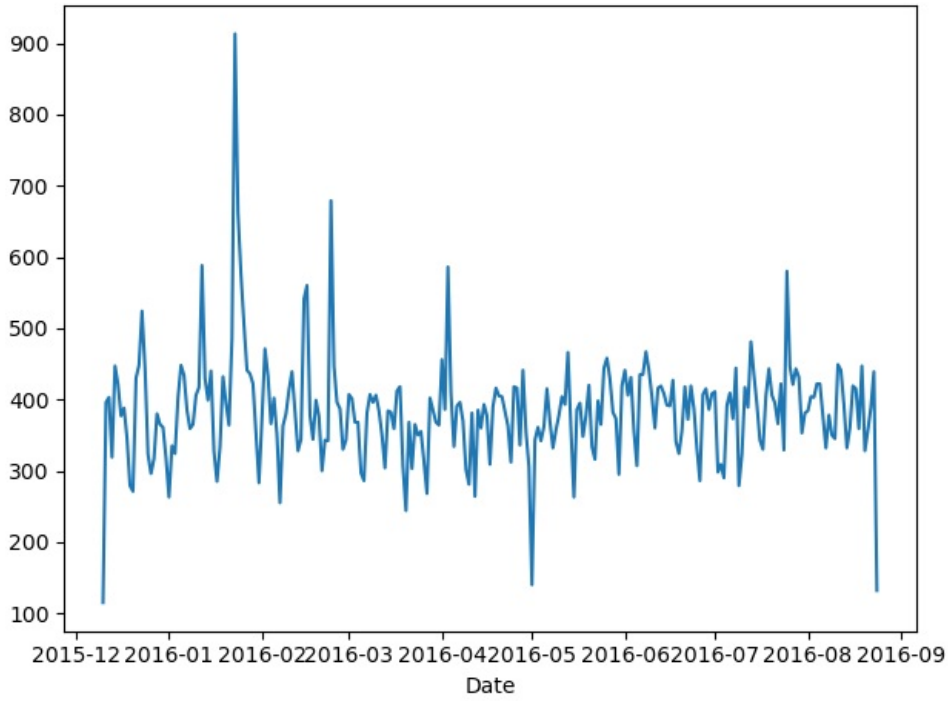
Now groupby this Date column with the count() aggregate and create a plot of counts of 911 calls.

In [72]:
```
df.groupby('Date').count().head() #Choose any from lat,lng,desc etc.
```

Out[72]:

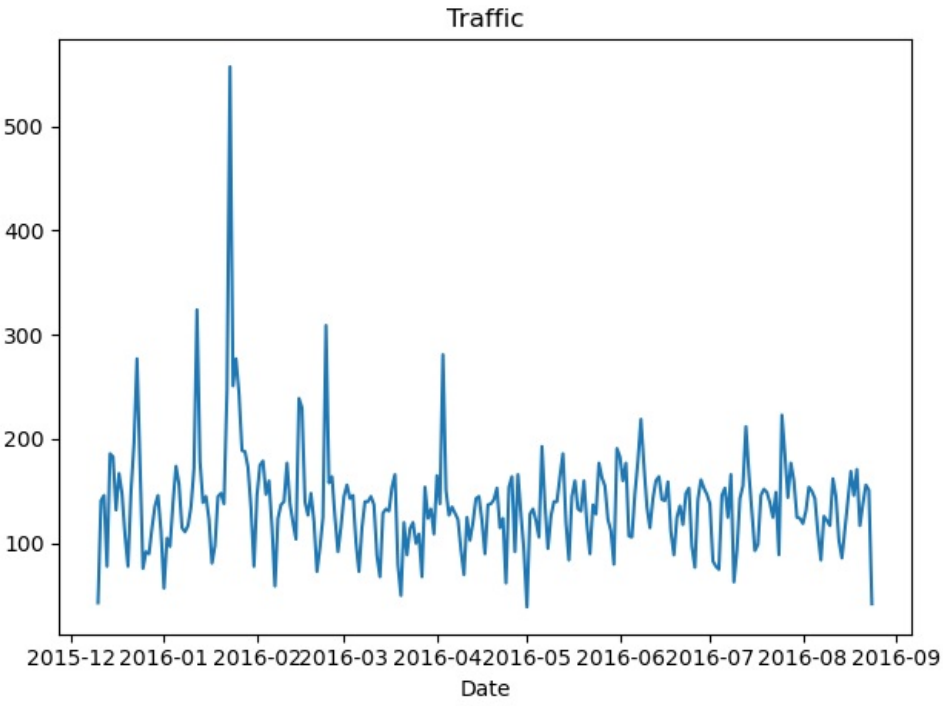
Date	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Reason	Hour	Month	Day of Week
2015-12-10	115	115	115	100	115	115	115	113	115	115	115	115	115
2015-12-11	396	396	396	333	396	396	395	391	396	396	396	396	396
2015-12-12	403	403	403	333	403	403	403	401	403	403	403	403	403
2015-12-13	319	319	319	280	319	319	319	317	319	319	319	319	319
2015-12-14	447	447	447	387	447	447	446	445	447	447	447	447	447

In [75]:
```
df.groupby('Date').count()['lat'].plot()
plt.tight_layout()
```

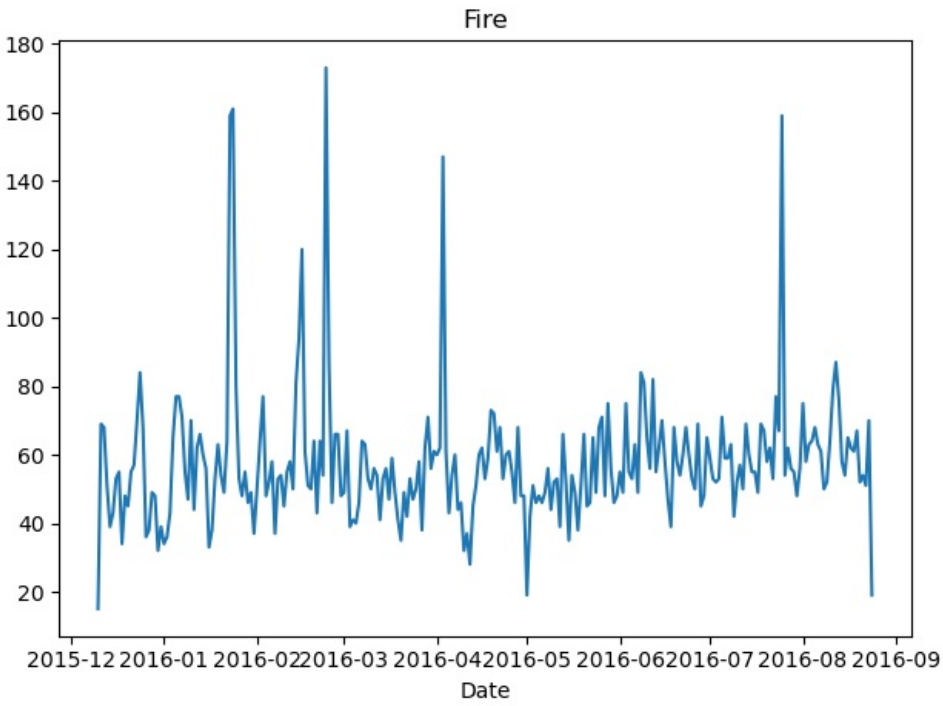


Now recreate this plot but create 3 separate plots with each plot representing a Reason for the 911 call

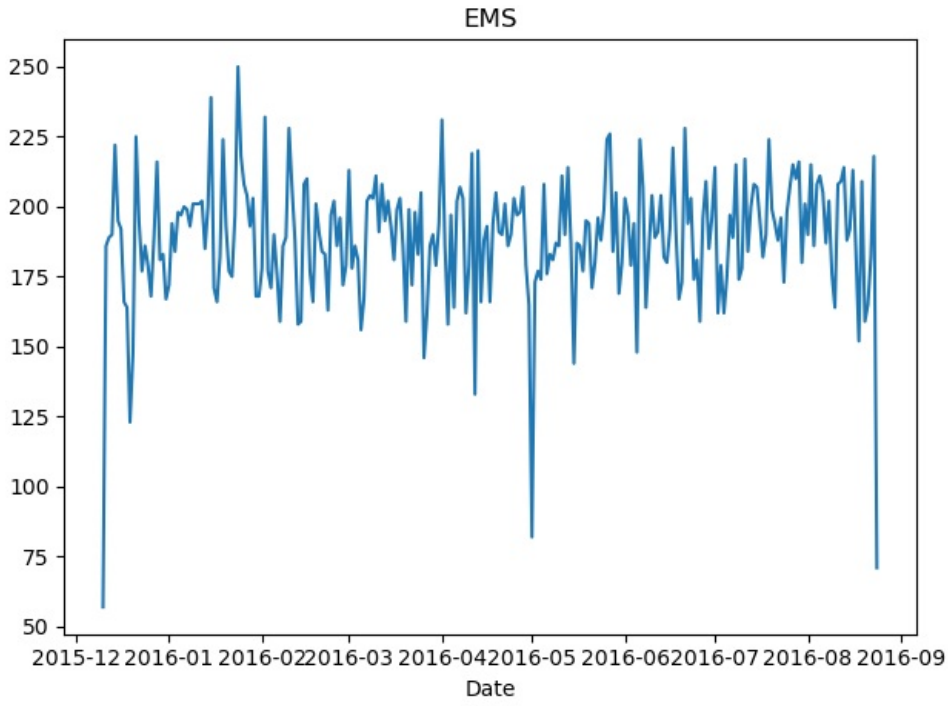
In [77]:
```
df[df['Reason']=='Traffic'].groupby('Date').count()['lat'].plot()
plt.title('Traffic')
plt.tight_layout()
```



In [79]:
```
df[df['Reason']=='Fire'].groupby('Date').count()['lat'].plot()
plt.title('Fire')
plt.tight_layout()
```



In [78]:
```
df[df['Reason']=='EMS'].groupby('Date').count()['lat'].plot()
plt.title('EMS')
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 method. Reference the solutions if you get stuck on this!

In [81]:
```
#Never did this lmao

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

Out[81]:

Hour	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	23
Day of Week																					
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
Tue	269	240	186	170	209	239	415	655	889	880	...	943	938	1026	1019	905	731	647	571	462	274
Wed	250	216	189	209	156	255	410	701	875	808	...	904	867	990	1037	894	686	668	575	490	335

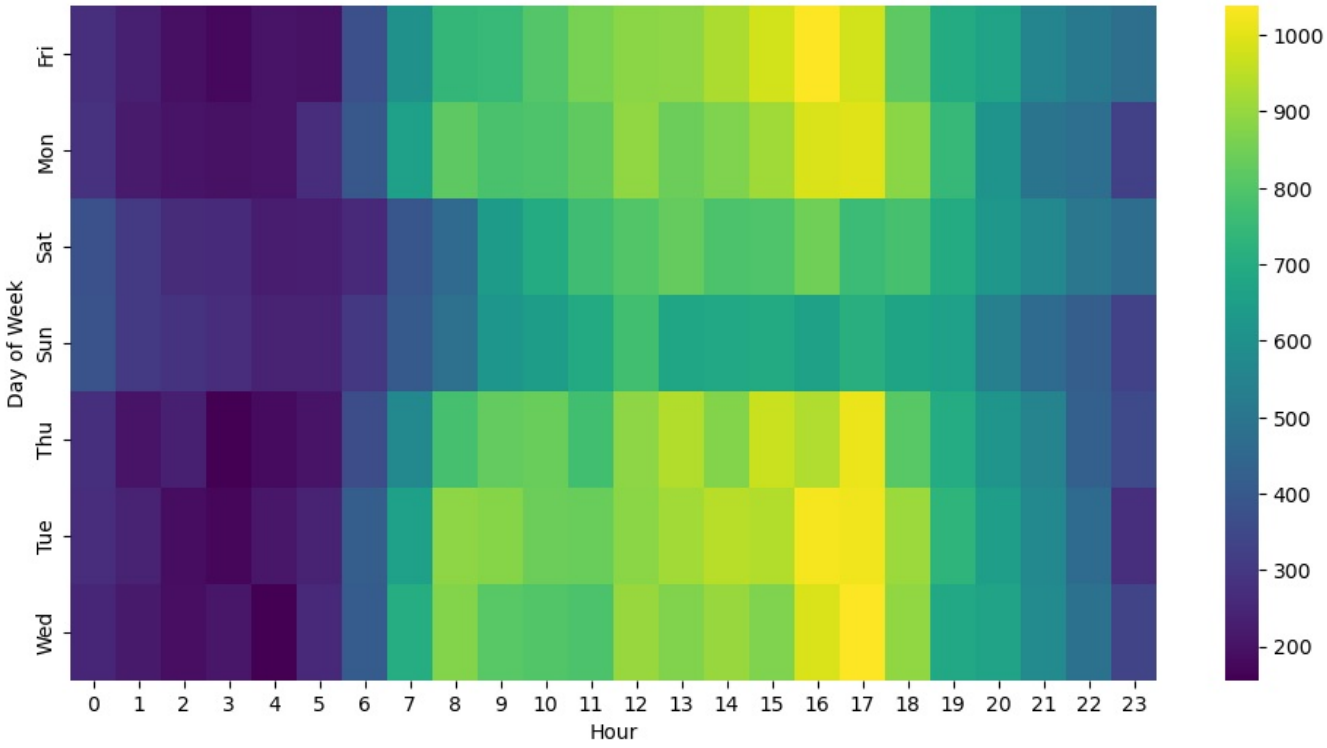
7 rows × 24 columns

In [82]:
```
#Let's name it
dayHour = df.groupby(by=['Day of Week','Hour']).count()['Reason'].unstack()
```

Now create a HeatMap using this new DataFrame.

In [85]:
```
plt.figure(figsize=(12,6))
sns.heatmap(dayHour,cmap='viridis')
```

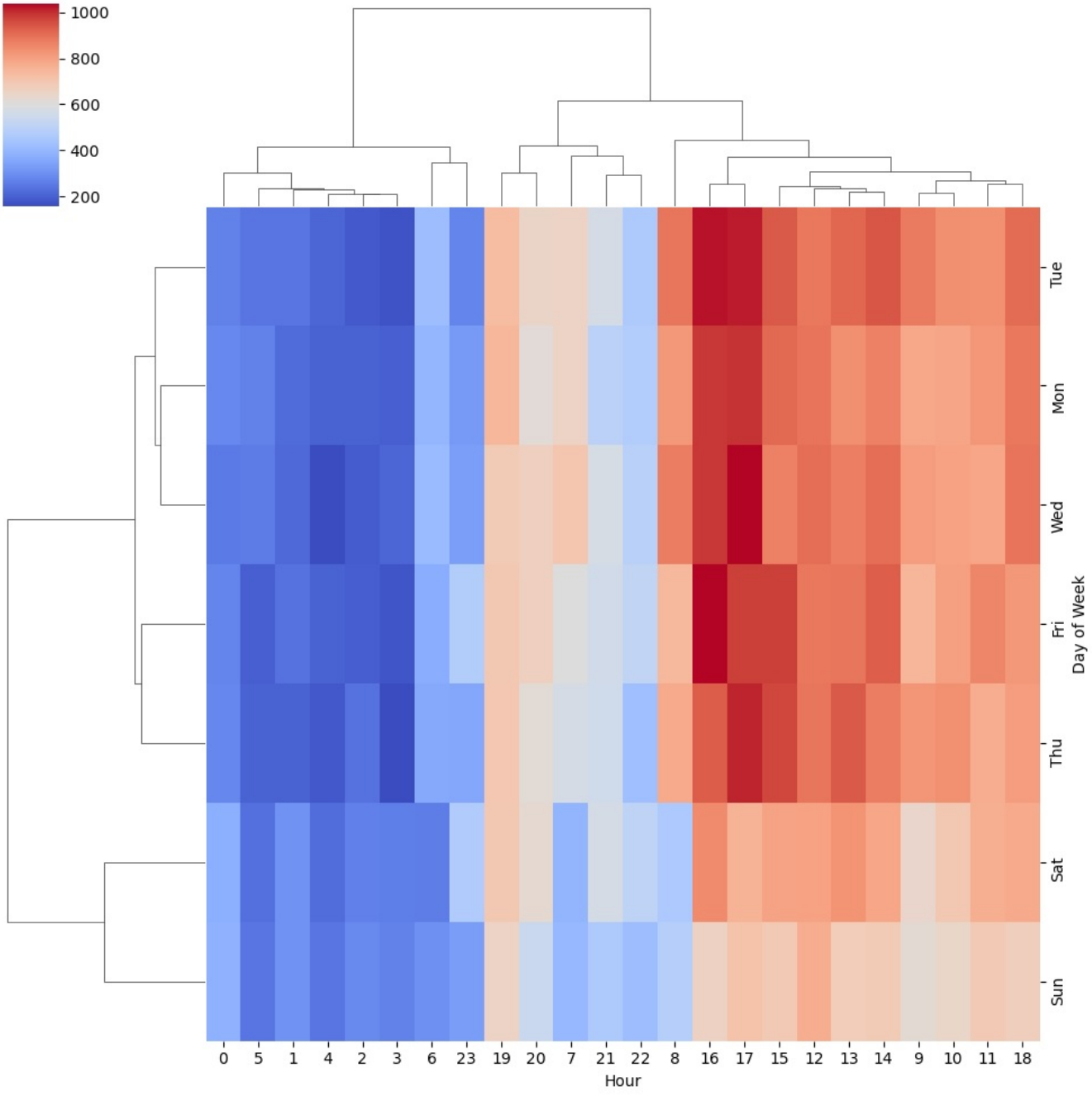
Out[85]:
```
<AxesSubplot:xlabel='Hour', ylabel='Day of Week'>
```



Now create a clustermap using this DataFrame.

In [91]:
```
sns.clustermap(dayHour,cmap='coolwarm')
```

Out[91]:
```
<seaborn.matrix.ClusterGrid at 0x2714cd2deb0>
```



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

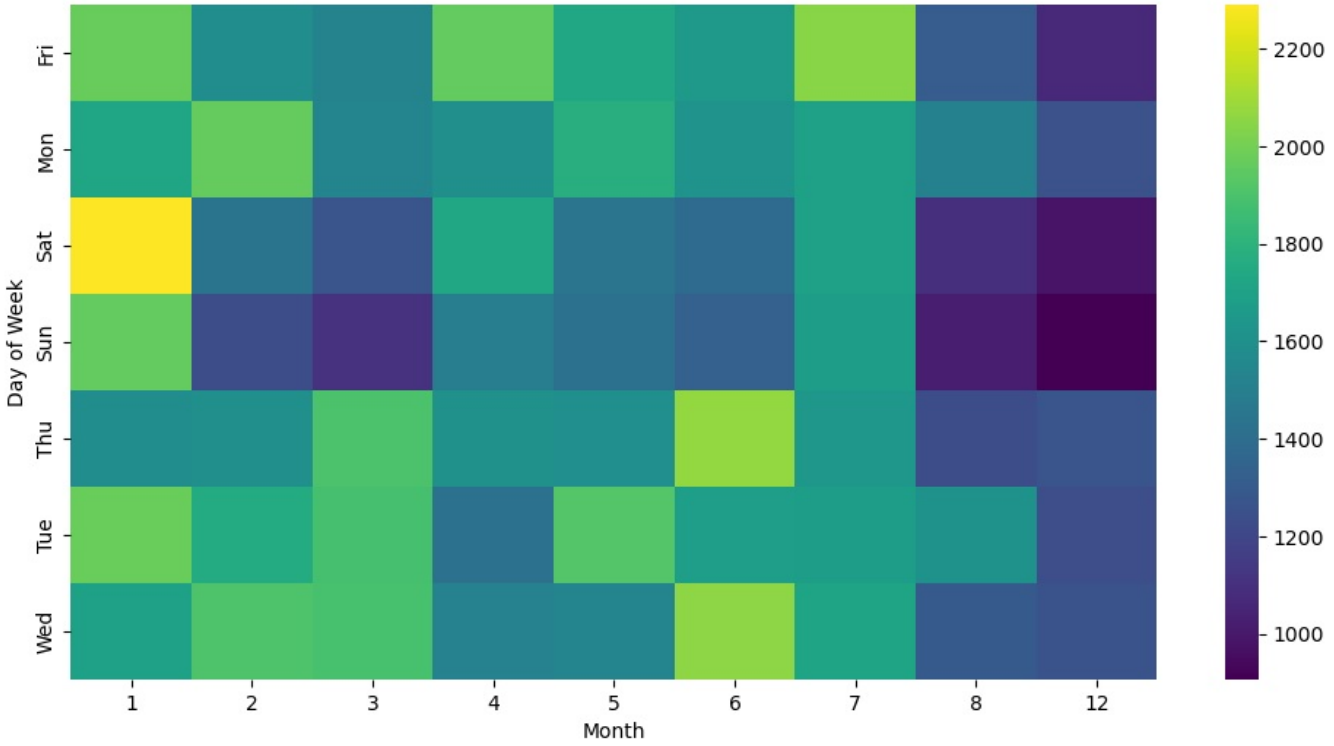
In [88]:
```
dayMonth = df.groupby(by=['Day of Week','Month']).count()['Reason'].unstack()

dayMonth.head()
```

Out[88]:

Day of Week \ Month	1	2	3	4	5	6	7	8	12
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

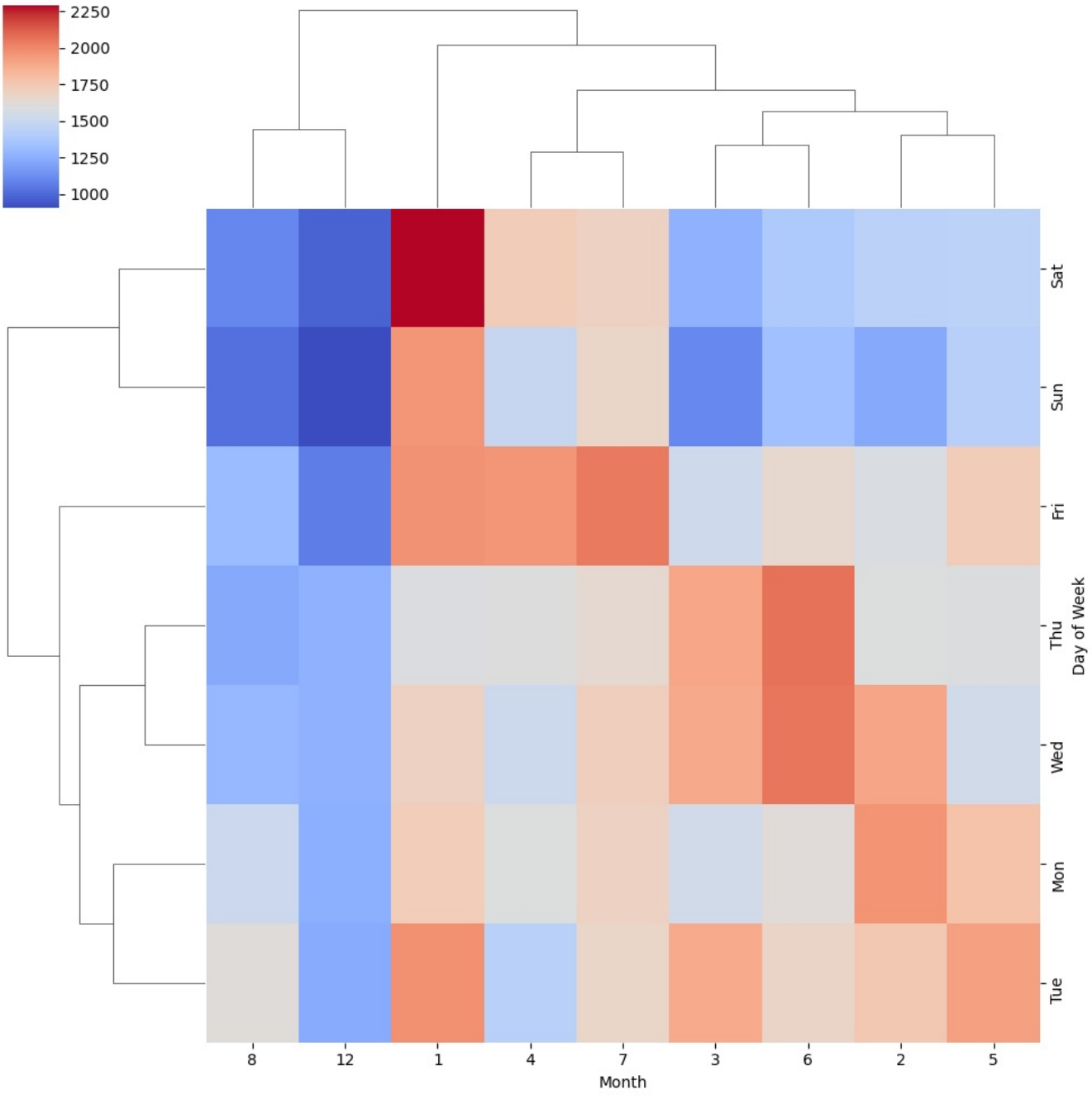
In [89]:
```
plt.figure(figsize=(12,6))
sns.heatmap(dayMonth,cmap='viridis')
```

Out[89]:
```
<AxesSubplot:xlabel='Month', ylabel='Day of Week'>
```



In [92]:
```
sns.clustermap(dayMonth,cmap='coolwarm')
```

Out[92]:
```
<seaborn.matrix.ClusterGrid at 0x2714bc96220>
```



Continue exploring the Data however you see fit!

Great Job!

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js