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)

Data and Setup

Import numpy and pandas

In [1]:

```
import numpy as np
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 [5]:

```
df = pd.read_csv('/content/911.csv')
```

Check the info() of the df

In [6]:

```
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 [7]:

```
df.head()
```

Out[7]:

	lat	lng	desc	zip	
0	40.297876	-75.581294	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.0	EI PAIN
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.0	EMI
2	40.121182	-75.351975	HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21-St...	19401.0	F OD
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMI
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	D

Basic Questions

What are the top 5 zipcodes for 911 calls?

In [8]:

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

Out[8]:

```
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 [9]:

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

Out[9]:

```
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 [10]:

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

Out[10]:

```
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 [11]:

```
df['Reason'] = df['title'].apply(lambda title
```

In [13]:

```
df['Reason']
```

Out[13]:

```
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 [14]:

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

Out[14]:

```
EMS        48877
Traffic    35695
Fire       14920
Name: Reason, dtype: int64
```

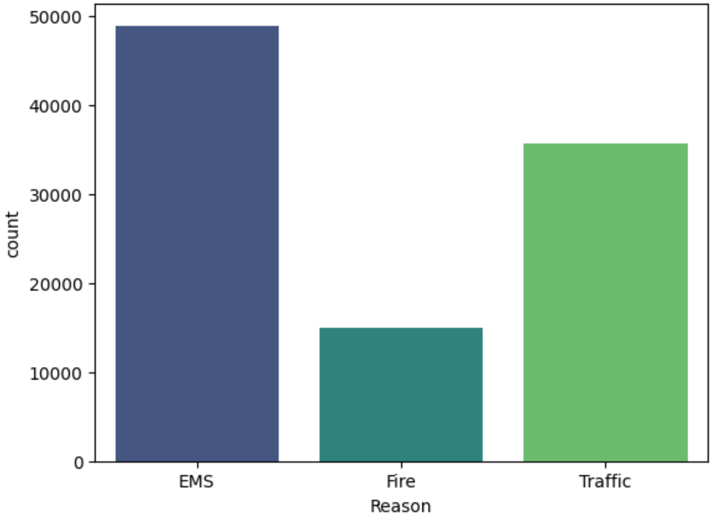
Now use seaborn to create a countplot of 911 calls by Reason.

In [16]:

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

Out[16]:

```
<Axes: 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 [18]:
```
type(df['timeStamp'].iloc[0])
```

Out[18]:
```
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 [19]:
```
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 [20]:
```
time = df['timeStamp'].iloc[0]
time.hour
```

Out[20]:
```
17
```

In [50]:
```
df['Hour'] = df['timeStamp'].apply(lambda time
df['Month'] = df['timeStamp'].apply(lambda tim
df['dayofweek'] = df['timeStamp'].apply(lambda
```

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
```

In [27]:
```
dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri
```

In [28]:
```
df['dayofweek'] = df['dayofweek'].map(dmap)
```

In [51]:
```
df.head()
```

Out[51]:

	lat	lng	desc	zip	
0	40.297876	-75.581294	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.0	EI PAIN
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.0	EMI
2	40.121182	-75.351975	HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21-St...	19401.0	F OD
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMI
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	D

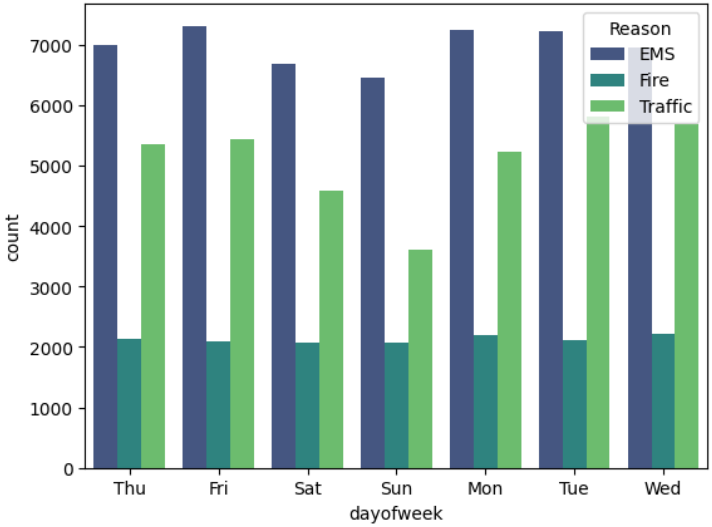
Now use seaborn to create a countplot of the Day of Week column with the hue based off of the Reason column.

In [33]:

```
sns.countplot(x = 'dayofweek',data=df,hue = 'F
```

Out[33]:

```
<Axes: xlabel='dayofweek', ylabel='count'>
```



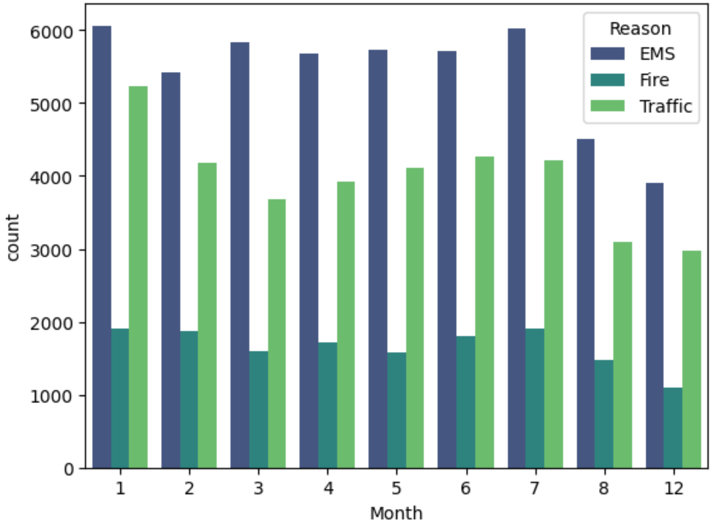
Now do the same for Month:

In [34]:

```
sns.countplot(x = 'Month',data = df,hue = 'Rea
```

Out[34]:

```
<Axes: xlabel='Month', ylabel='count'>
```



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.

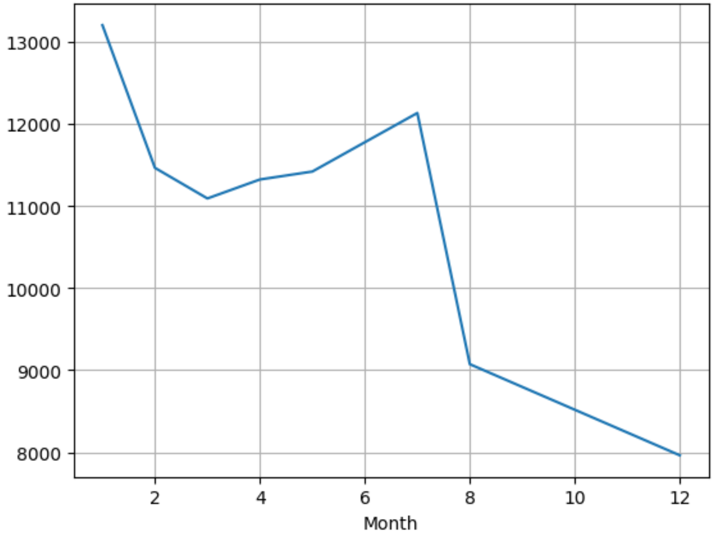
In [38]:
```
bymonth=df.groupby('Month').count()
bymonth.head(5)
```

Out[38]:

	lat	lng	desc	zip	title	timeStamp
Month						
1	13205	13205	13205	11527	13205	13205
2	11467	11467	11467	9930	11467	11467
3	11101	11101	11101	9755	11101	11101
4	11326	11326	11326	9895	11326	11326
5	11423	11423	11423	9946	11423	11423

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

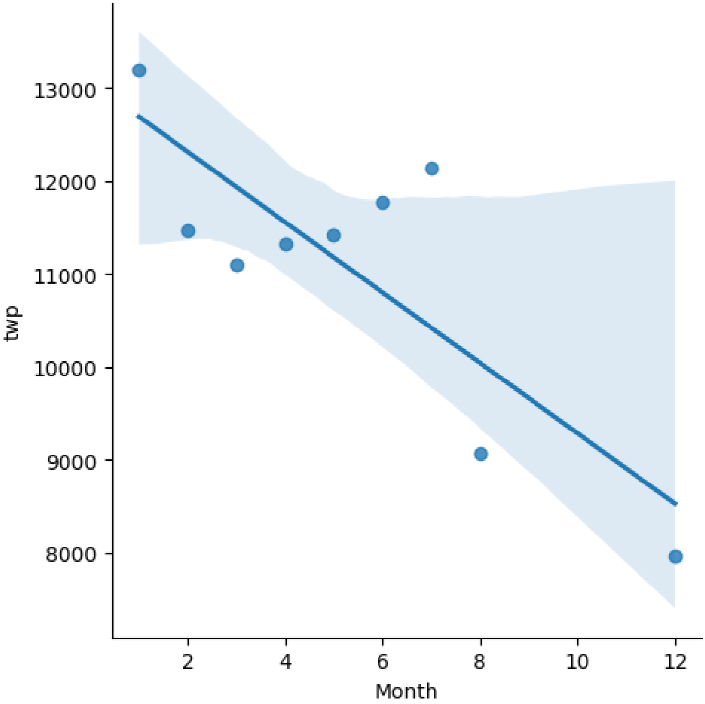
In [40]:
```
bymonth['twp'].plot()
plt.grid()
```



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 [43]:
```
sns.lmplot(x='Month',y='twp',data=bymonth.rese
```

Out[43]:
```
<seaborn.axisgrid.FacetGrid at 0x7dda44874d30
>
```



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 [57]:
```
df['Date'] = df['timeStamp'].apply(lambda time
df['Date']
```

Out[57]:
```
0        2015-12-10
1        2015-12-10
2        2015-12-10
3        2015-12-10
4        2015-12-10
            ...
99487    2016-08-24
99488    2016-08-24
99489    2016-08-24
99490    2016-08-24
99491    2016-08-24
Name: Date, Length: 99492, dtype: object
```

In [58]:

```
df.head(2)
```

Out[58]:

	lat	lng	desc	zip	
0	40.297876	-75.581294	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.0	EM PAINS
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.0	D EME

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

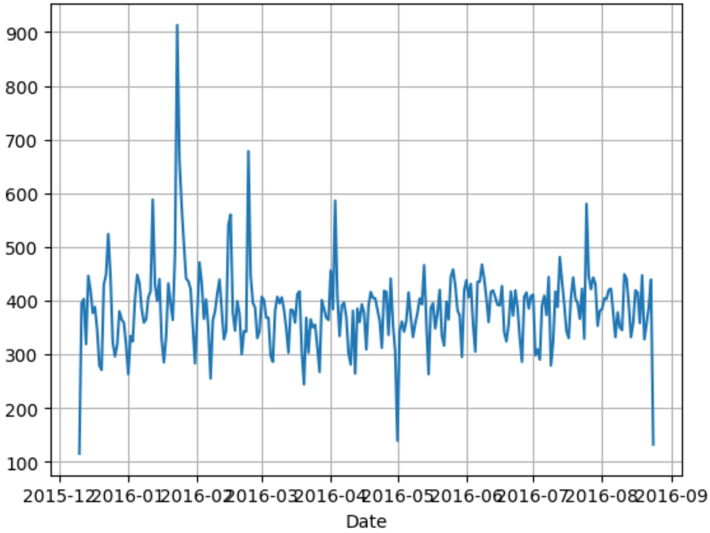
In [63]:

```
bydate = df.groupby(df['Date']).count()
bydate.head()
```

Out[63]:

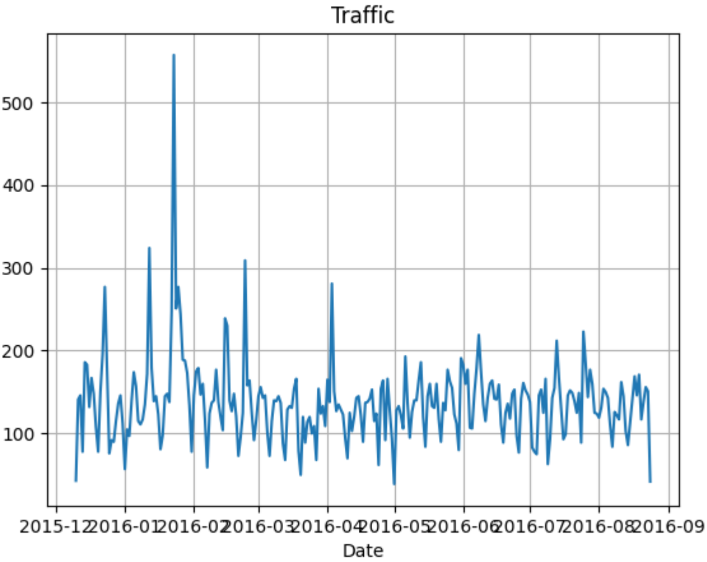
	lat	lng	desc	zip	title	timeStamp	twp	ad
Date								
2015-12-10	115	115	115	100	115	115	115	1
2015-12-11	396	396	396	333	396	396	395	3
2015-12-12	403	403	403	333	403	403	403	4
2015-12-13	319	319	319	280	319	319	319	3
2015-12-14	447	447	447	387	447	447	446	4

In [65]:
```
bydate['twp'].plot()
plt.grid()
```

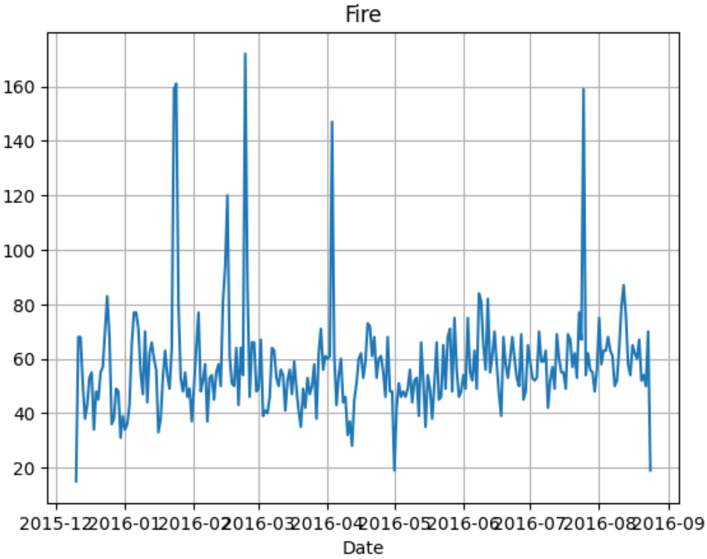


Now recreate this plot but create 3 separate plots with each plot representing a Reason for the 911 call

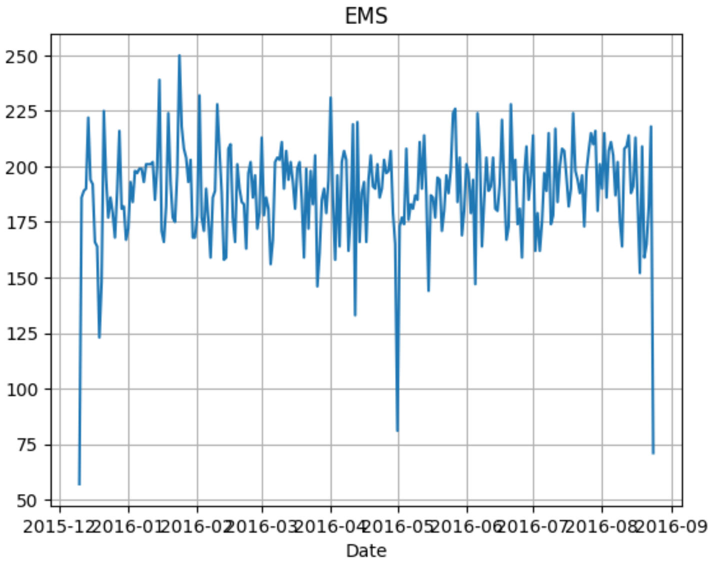
In [69]:
```
df[df['Reason'] == 'Traffic'].groupby('Date').
plt.title('Traffic')

plt.grid()
```



In [73]:
```
df[df['Reason'] == 'Fire'].groupby('Date').cou
plt.title('Fire')
plt.grid()
```



In [75]:
```
df[df['Reason'] == 'EMS'].groupby('Date').cour
plt.title('EMS')
plt.grid()
```



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 [80]:

```
dayHour = df.groupby(by = ['dayofweek','Hour']
dayHour.head()
```

Out[80]:

Hour	0	1	2	3	4	5	6	7
dayofweek								
0	282	221	201	194	204	267	397	653
1	269	240	186	170	209	239	415	655
2	250	216	189	209	156	255	410	701
3	278	202	233	159	182	203	362	570
4	275	235	191	175	201	194	372	598

5 rows × 24 columns

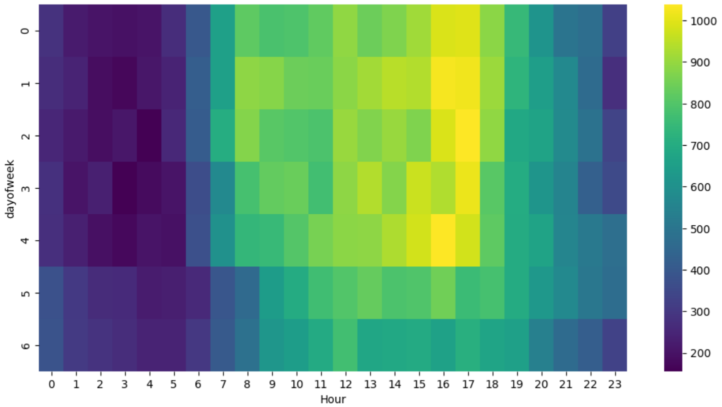
Now create a HeatMap using this new DataFrame.

In [95]:

```
plt.figure(figsize=(12,6))
sns.heatmap(dayHour,cmap='viridis')
```

Out[95]:

```
<Axes: xlabel='Hour', ylabel='dayofweek'>
```



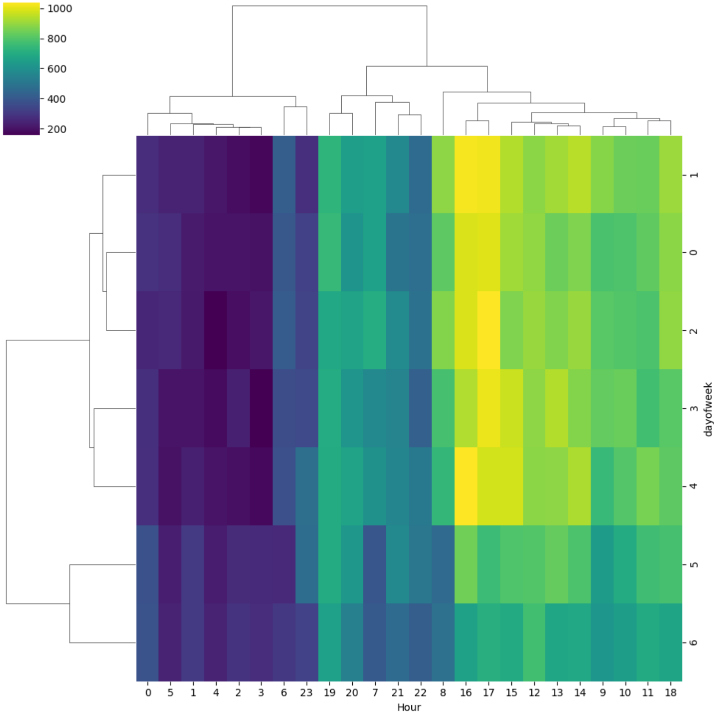
Now create a clustermap using this DataFrame.

In [93]:

```
sns.clustermap(dayHour,cmap='viridis')
```

Out[93]:

```
<seaborn.matrix.ClusterGrid at 0x7dda3a7d4670
>
```



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

In [86]:

```
dayMonth = df.groupby(by = ['dayofweek','Month
dayMonth.head()
```

Out[86]:

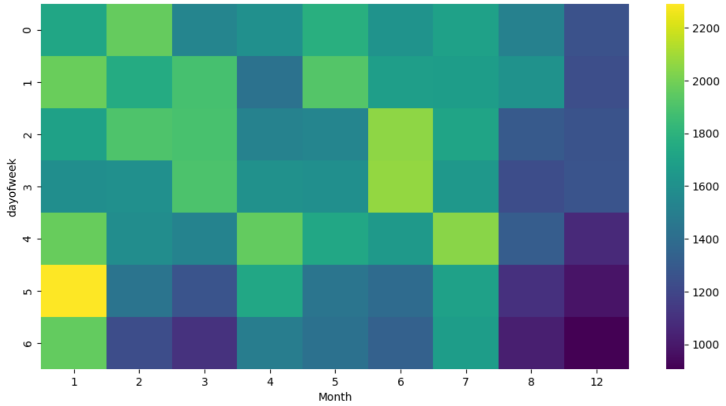
Month	1	2	3	4	5	6	7
dayofweek							
0	1727	1964	1535	1598	1779	1617	1692
1	1973	1753	1884	1430	1918	1676	1670
2	1700	1903	1889	1517	1538	2058	1717
3	1584	1596	1900	1601	1590	2065	1646
4	1970	1581	1525	1958	1730	1649	2045

In [96]:

```
plt.figure(figsize=(12,6))
sns.heatmap(dayMonth,cmap='viridis')
```

Out[96]:

```
<Axes: xlabel='Month', ylabel='dayofweek'>
```

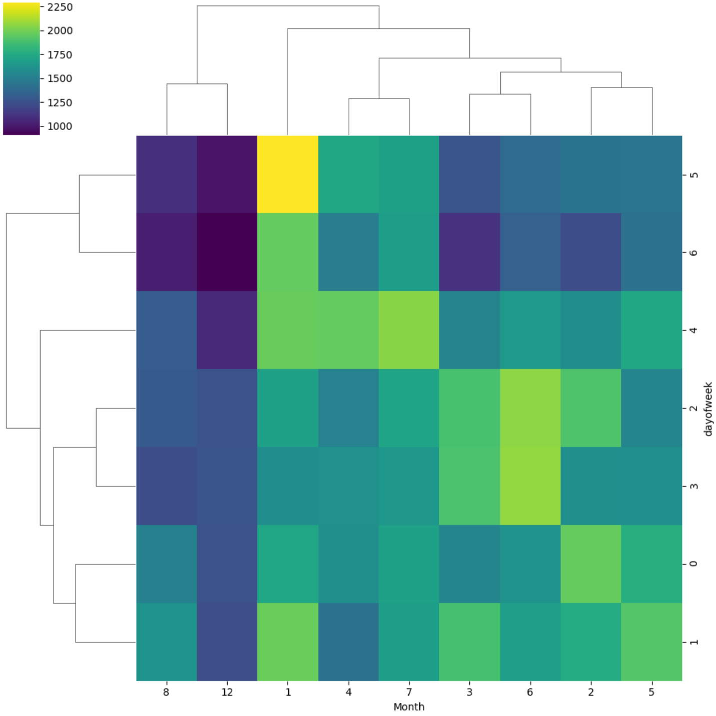


In [97]:

```
sns.clustermap(dayMonth,cmap='viridis')
```

Out[97]:

```
<seaborn.matrix.ClusterGrid at 0x7dda3ad4fa00
>
```


2250
2000
1750
1500
1250
1000
5
6
4
2
3
0
1
dayofweek
8
12
1
4
7
3
6
2
5
Month