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]:
```python
import numpy as np
import pandas as pd
```

Import visualization libraries and set %matplotlib inline.

In [2]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
%matplotlib inline
```

Read in the csv file as a dataframe called df

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

Check the info() of the df

In [4]:
```python
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]:
```python
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

In [ ]:

Basic Questions

What are the top 5 zipcodes for 911 calls?

In [6]:
```python
df['zip'].value_counts().head(5)
```

Out[6]:
```
19401.0    6979
19464.0    6643
19403.0    4854
19446.0    4748
19406.0    3174
Name: zip, dtype: int64
```

In [ ]:

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

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

Out[7]:
```
LOWER MERION    8443
ABINGTON        5977
NORRISTOWN      5890
UPPER MERION    5227
CHELTENHAM      4575
Name: twp, dtype: int64
```

In [ ]:

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

In [8]:
```python
df['title'].nunique()
```

Out[8]:
```
110
```

In [ ]:

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 [9]:
```python
def sepreason(s):
    temp = s.split(':')
    reason = ['EMS', 'Fire', 'Traffic']
    if temp[0] in reason:
        return temp[0]
    else:
        return None
```

In [10]:
```python
df['Reason'] = df['title'].apply(sepreason)
```

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

Out[11]:
```
99492
```

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

In [12]:
```python
df['Reason'].value_counts().head()
```

Out[12]:
```
EMS        48877
Traffic    35695
Fire       14920
Name: Reason, dtype: int64
```

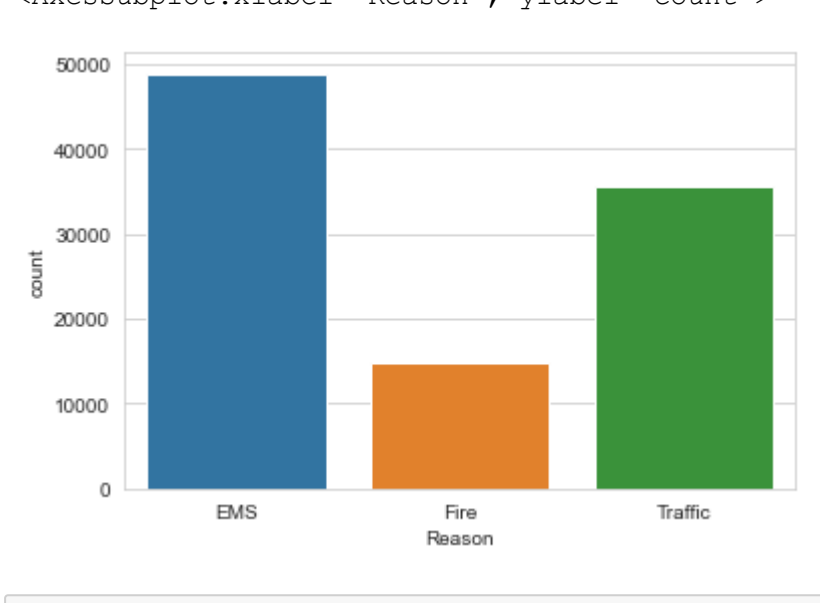
In [ ]:

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

In [13]:
```python
sns.set_style('whitegrid')
```

In [14]:
```python
sns.countplot(x='Reason', data=df)
```

Out[14]:
```
<AxesSubplot:xlabel='Reason', ylabel='count'>
```



In [ ]:

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

In [15]:
```python
type(df['timeStamp'].iloc[0])
```

Out[15]:
```
str
```

In [ ]:

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

In [16]:
```python
df['timeStamp'] = pd.to_datetime(df['timeStamp'])
```

In [17]:
```python
time = df['timeStamp'].iloc[0]
time.hour
```

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

In [ ]:

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 [18]:
```python
df['Hour'] = df['timeStamp'].apply(lambda time: time.hour)
df['Hour'].head()
```

Out[18]:
```
0    17
1    17
2    17
3    17
4    17
Name: Hour, dtype: int64
```

In [19]:
```python
df['Month'] = df['timeStamp'].apply(lambda time: time.month)
df['Month']
```

Out[19]:
```
0        12
1        12
2        12
3        12
4        12
         ..
99487     8
99488     8
99489     8
99490     8
99491     8
Name: Month, Length: 99492, dtype: int64
```

In [20]:
```python
dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}

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

Out[20]:
```
0        3
1        3
2        3
3        3
4        3
        ..
99487    2
99488    2
99489    2
99490    2
99491    2
Name: Day of Week, Length: 99492, dtype: int64
```

In [21]:
```python
df['Day of Week'] = df['Day of Week'].map(dmap)
df['Day of Week'].head()
```

Out[21]:
```
0    Thu
1    Thu
2    Thu
3    Thu
4    Thu
Name: Day of Week, dtype: object
```

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'}
```

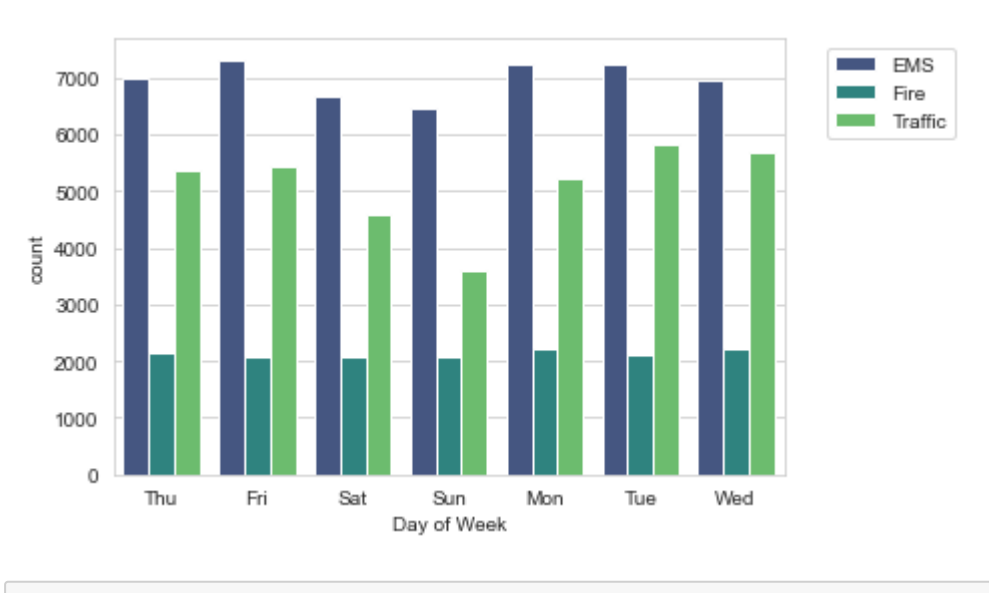
In [ ]:

In [ ]:

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

In [22]:
```python
sns.countplot(x='Day of Week',data=df, hue = df['Reason'], palette='viridis')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
```

Out[22]:
```
<matplotlib.legend.Legend at 0x1b12fdf9dc0>
```

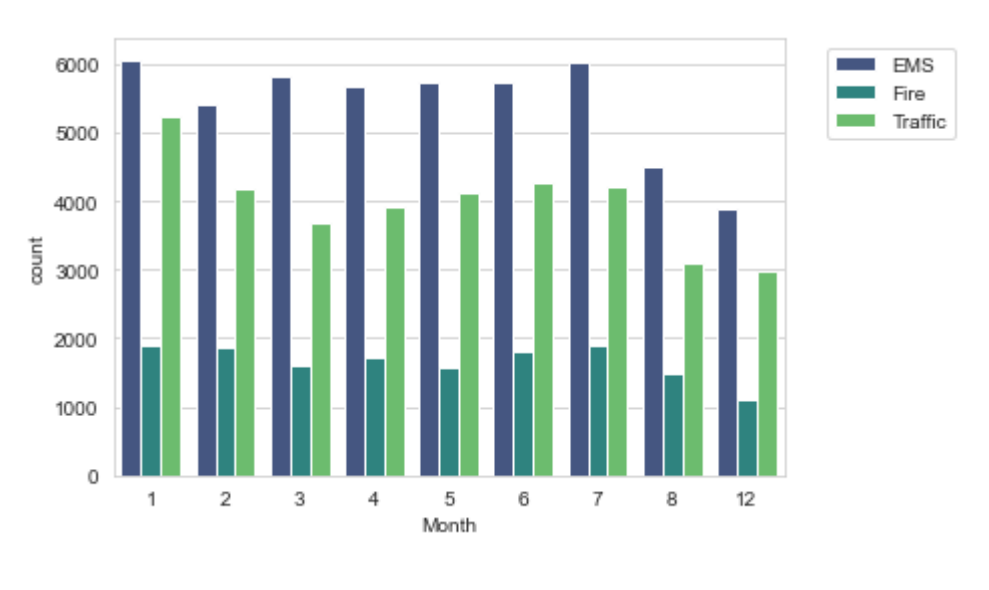


In [ ]:

Now do the same for Month:

In [23]:
```python
sns.countplot(x='Month',data=df, hue = df['Reason'], palette='viridis')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
```

Out[23]:
```
<matplotlib.legend.Legend at 0x1b12fdf9ca0>
```



This Plot is missing the value of a few months. (Sep Oct Nov)

In [ ]:

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 [32]:
```python
byMonth = df.groupby('Month').count()
byMonth.head()
```

Out[32]:

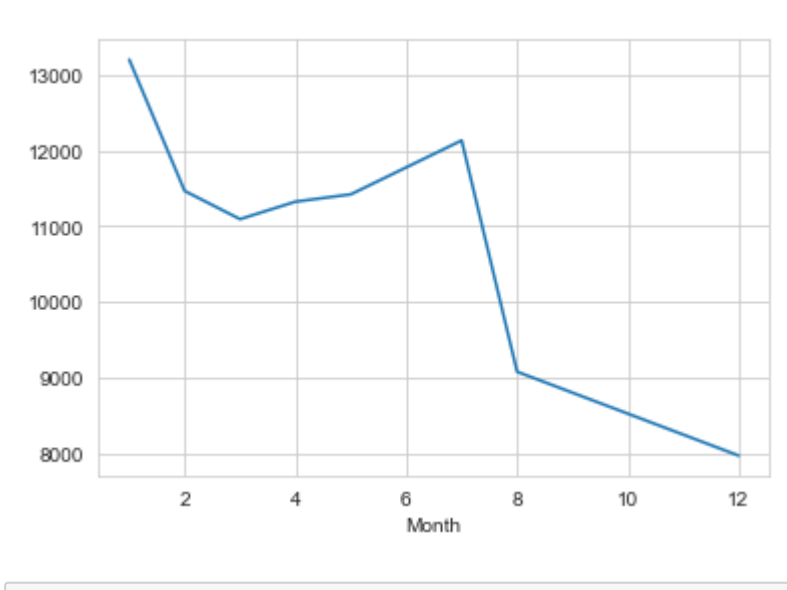
Month	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Reason	Hour	Day of Week
1	13205	13205	13205	11527	13205	13205	13203	13096	13205	13205	13205	13205
2	11467	11467	11467	9930	11467	11467	11465	11396	11467	11467	11467	11467
3	11101	11101	11101	9755	11101	11101	11092	11059	11101	11101	11101	11101
4	11326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

In [ ]:

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

In [33]:
```python
byMonth['twp'].plot()
```

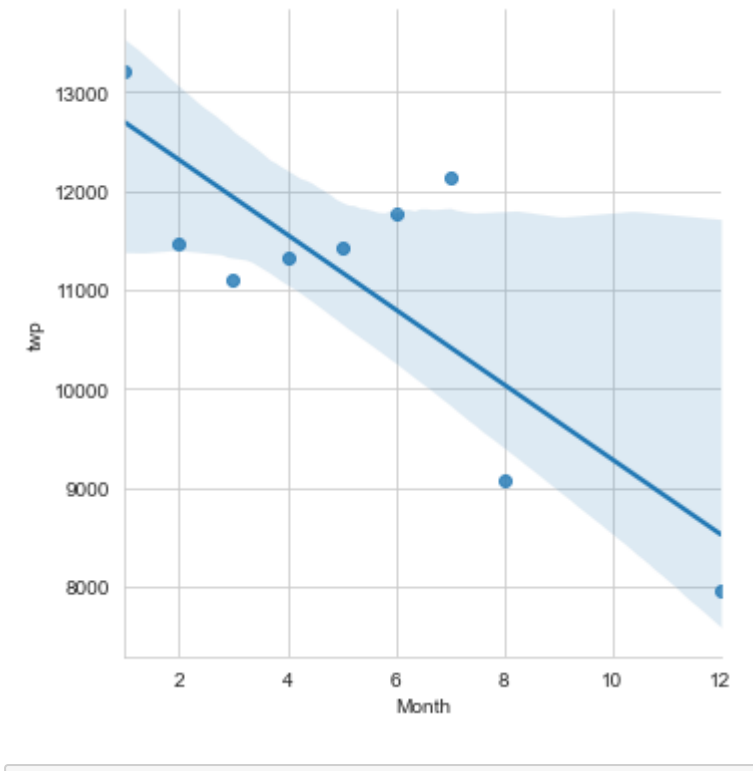
Out[33]:
```
<AxesSubplot:xlabel='Month'>
```



In [34]:
```python
byMonth.reset_index(inplace = True)
```

In [35]:
```python
sns.lmplot(x = 'Month', y = 'twp', data = byMonth)
```

Out[35]:
```
<seaborn.axisgrid.FacetGrid at 0x1b12fe09190>
```



In [ ]:

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 [ ]:

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 [71]:
```python
df['Date'] = df['timeStamp'].apply(lambda time: time.date())
df.head()
```

Out[71]:

	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

In [57]:
```python
df3 = df.groupby('Date').count()
df3.reset_index(inplace = True)
```

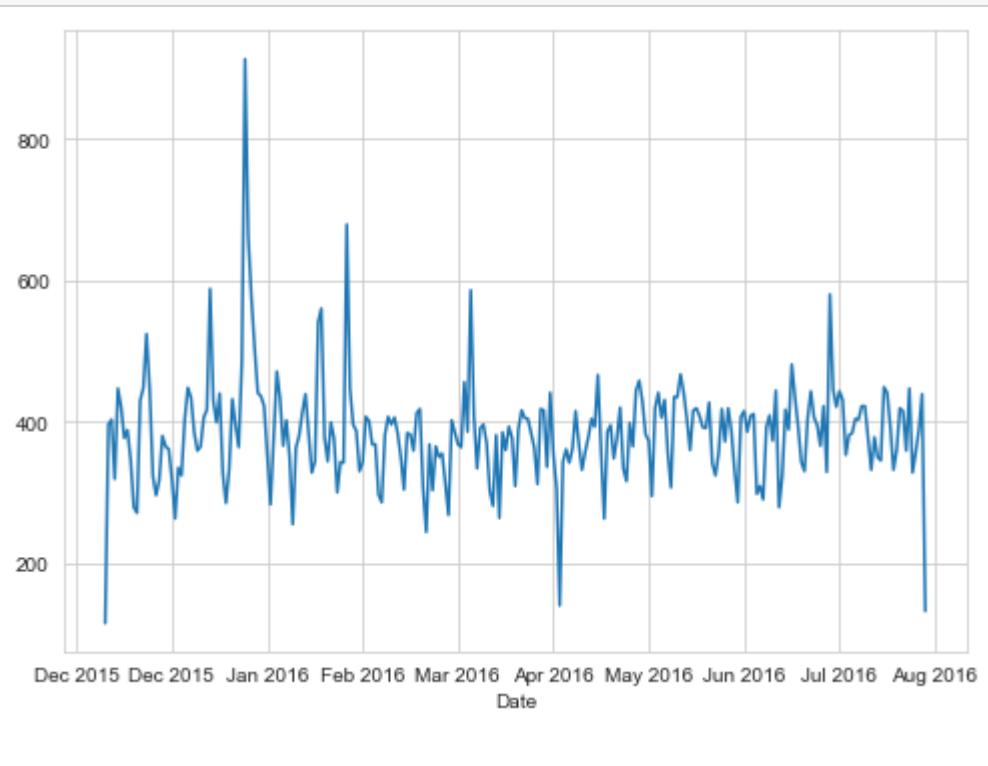
In [58]:
```python
df3.head()
```

Out[58]:

	Date	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Reason	Hour	Month	Day of Week
0	2015-12-10	115	115	115	100	115	115	115	113	115	115	115	115	115
1	2015-12-11	396	396	396	333	396	396	395	391	396	396	396	396	396
2	2015-12-12	403	403	403	333	403	403	403	401	403	403	403	403	403
3	2015-12-13	319	319	319	280	319	319	319	317	319	319	319	319	319
4	2015-12-14	447	447	447	387	447	447	446	445	447	447	447	447	447

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

In [69]:
```python
# Set the figure size and axis
fig, ax = plt.subplots(figsize = (7, 5))

# Plot the data
plt.plot(df3['Date'], df3['timeStamp'])
ax.set_xlabel('Date')

# Set number of major ticks
ax.xaxis.set_major_locator(plt.MaxNLocator(10))
ax.yaxis.set_major_locator(plt.MaxNLocator(5))

#set major ticks format
ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))

plt.tight_layout()
```



This plot above 👆 is the total counts of 911 calls in this year.

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

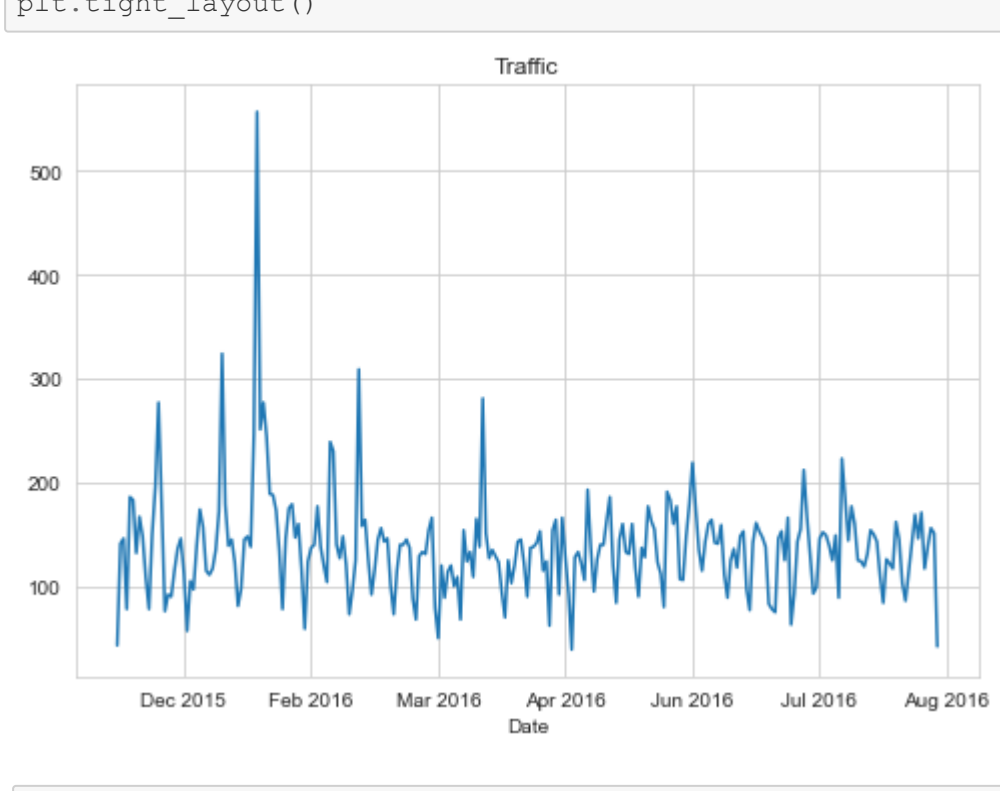
In [80]:
```python
traffic = df[df['Reason'] == 'Traffic'].groupby('Date').count().reset_index()
```

In [82]:
```python
traffic.head()
```

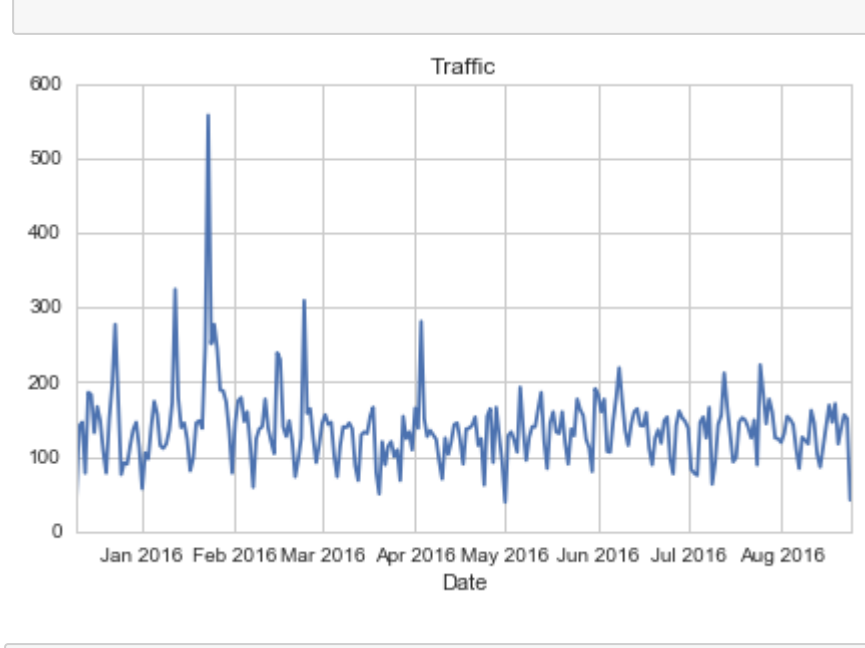
Out[82]:

	Date	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Reason	Hour	Month	Day of Week
0	2015-12-10	43	43	43	35	43	43	43	41	43	43	43	43	43
1	2015-12-11	141	141	141	108	141	141	141	137	141	141	141	141	141
2	2015-12-12	146	146	146	109	146	146	146	144	146	146	146	146	146
3	2015-12-13	78	78	78	54	78	78	78	76	78	78	78	78	78
4	2015-12-14	186	186	186	150	186	186	186	184	186	186	186	186	186

In [87]:
```python
# Set the figure size and axis
fig, ax = plt.subplots(figsize = (7, 5))

# Plot the data
plt.plot(traffic['Date'], traffic['timeStamp'])
ax.set_xlabel('Date')

# Set number of major ticks
ax.xaxis.set_major_locator(plt.MaxNLocator(9))
ax.yaxis.set_major_locator(plt.MaxNLocator(6))

#set major ticks format
ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))

plt.title('Traffic')
plt.tight_layout()
```



In [199]:



In [88]:
```python
fire = df[df['Reason'] == 'Fire'].groupby('Date').count().reset_index()
```

In [99]:
```python
fire.info()
```

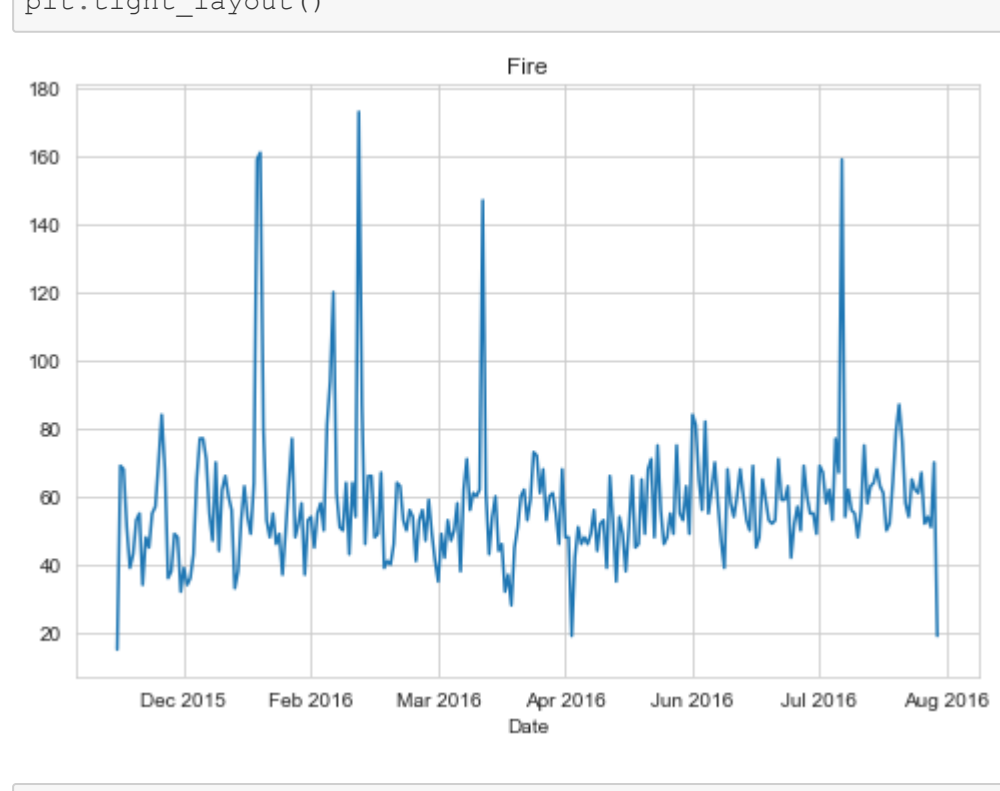
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 259 entries, 0 to 258
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Date         259 non-null    object
 1   lat          259 non-null    int64
 2   lng          259 non-null    int64
 3   desc         259 non-null    int64
 4   zip          259 non-null    int64
 5   title        259 non-null    int64
 6   timeStamp    259 non-null    int64
 7   twp          259 non-null    int64
 8   addr         259 non-null    int64
 9   e            259 non-null    int64
 10  Reason       259 non-null    int64
 11  Hour         259 non-null    int64
 12  Month        259 non-null    int64
 13  Day of Week  259 non-null    int64
dtypes: int64(13), object(1)
memory usage: 28.5+ KB
```

In [103]:
```python
fire[fire['Date'].map(lambda x: x.month) == 1].head()
# Check the Jan data exist.
# It's ambiguous in the graph shown below
# The x-axis didn't display Jan 2016.
```

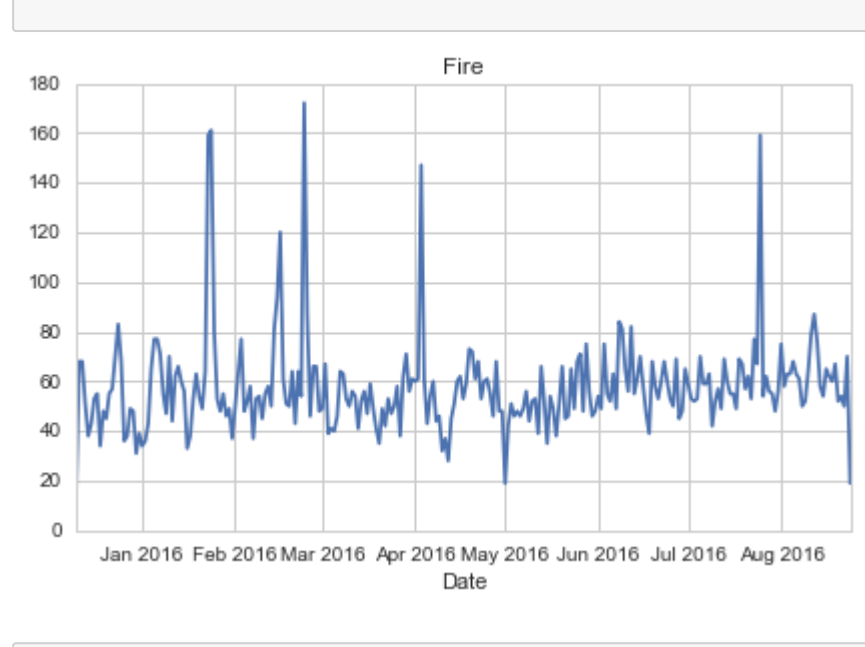
Out[103]:

	Date	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Reason	Hour	Month	Day of Week
22	2016-01-01	34	34	34	31	34	34	34	34	34	34	34	34	34
23	2016-01-02	36	36	36	34	36	36	36	36	36	36	36	36	36
24	2016-01-03	43	43	43	36	43	43	43	43	43	43	43	43	43
25	2016-01-04	65	65	65	56	65	65	65	65	65	65	65	65	65
26	2016-01-05	77	77	77	62	77	77	77	77	77	77	77	77	77

In [104]:
```python
# Set the figure size and axis
fig, ax = plt.subplots(figsize = (7, 5))

# Plot the data
plt.plot(fire['Date'], fire['timeStamp'])
ax.set_xlabel('Date')

# Set number of major ticks
ax.xaxis.set_major_locator(plt.MaxNLocator(8))
ax.yaxis.set_major_locator(plt.MaxNLocator(9))

#set major ticks format
ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))

plt.title('Fire')
plt.tight_layout()
```



In [201]:



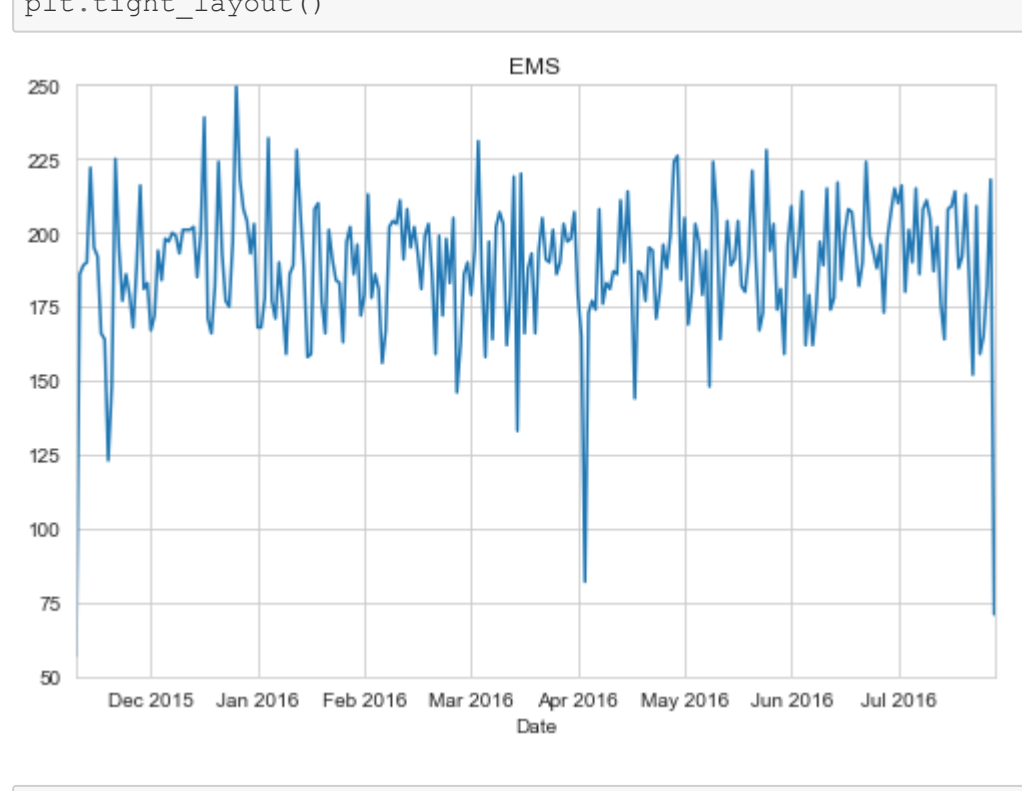
In [105]:
```python
ems = df[df['Reason'] == 'EMS'].groupby('Date').count().reset_index()
```

In [111]:
```python
ems.head()
```

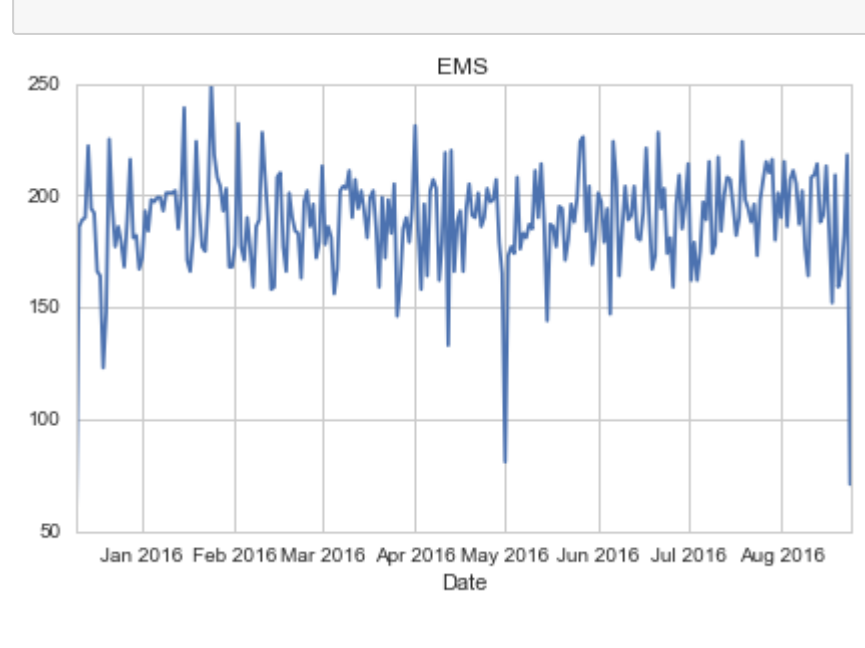
Out[111]:

	Date	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Reason	Hour	Month	Day of Week
0	2015-12-10	57	57	57	52	57	57	57	57	57	57	57	57	57
1	2015-12-11	186	186	186	166	186	186	186	186	186	186	186	186	186
2	2015-12-12	189	189	189	171	189	189	189	189	189	189	189	189	189
3	2015-12-13	190	190	190	178	190	190	190	190	190	190	190	190	190
4	2015-12-14	222	222	222	201	222	222	222	222	222	222	222	222	222

In [113]:
```python
# Set the figure size and axis
fig, ax = plt.subplots(figsize = (7, 5))

# Plot the data
plt.plot(ems['Date'], ems['timeStamp'])
ax.set_xlabel('Date')

# Set number of major ticks
ax.xaxis.set_major_locator(plt.MaxNLocator(10))
ax.yaxis.set_major_locator(plt.MaxNLocator(9))

#set major ticks format
ax.xaxis.set_major_formatter(mdates.DateFormatter('%b %Y'))


plt.title('EMS')


plt.xlim(ems['Date'].iloc[0], ems['Date'].iloc[-1])
plt.ylim(50,250)
plt.tight_layout()
```



In [202]:



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 [114]:
```python
df.head()
```

Out[114]:

	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

In [118]:
```python
wk_hr_smy = df.groupby(['Hour', 'Day of Week']).count()['lat']
```

In [119]:
```
wk_hr_smy.head()
```

Out[119]:
```
Hour  Day of Week
0     Fri            275
      Mon            282
      Sat            375
      Sun            383
      Thu            278
Name: lat, dtype: int64
```

In [125]:
```
wk_hr_smy_pivot = wk_hr_smy.unstack(level = 0)
```

In [126]:
```
wk_hr_smy_pivot.head()
```

Out[126]:

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

5 rows × 24 columns

In [203]:

Out[203]:

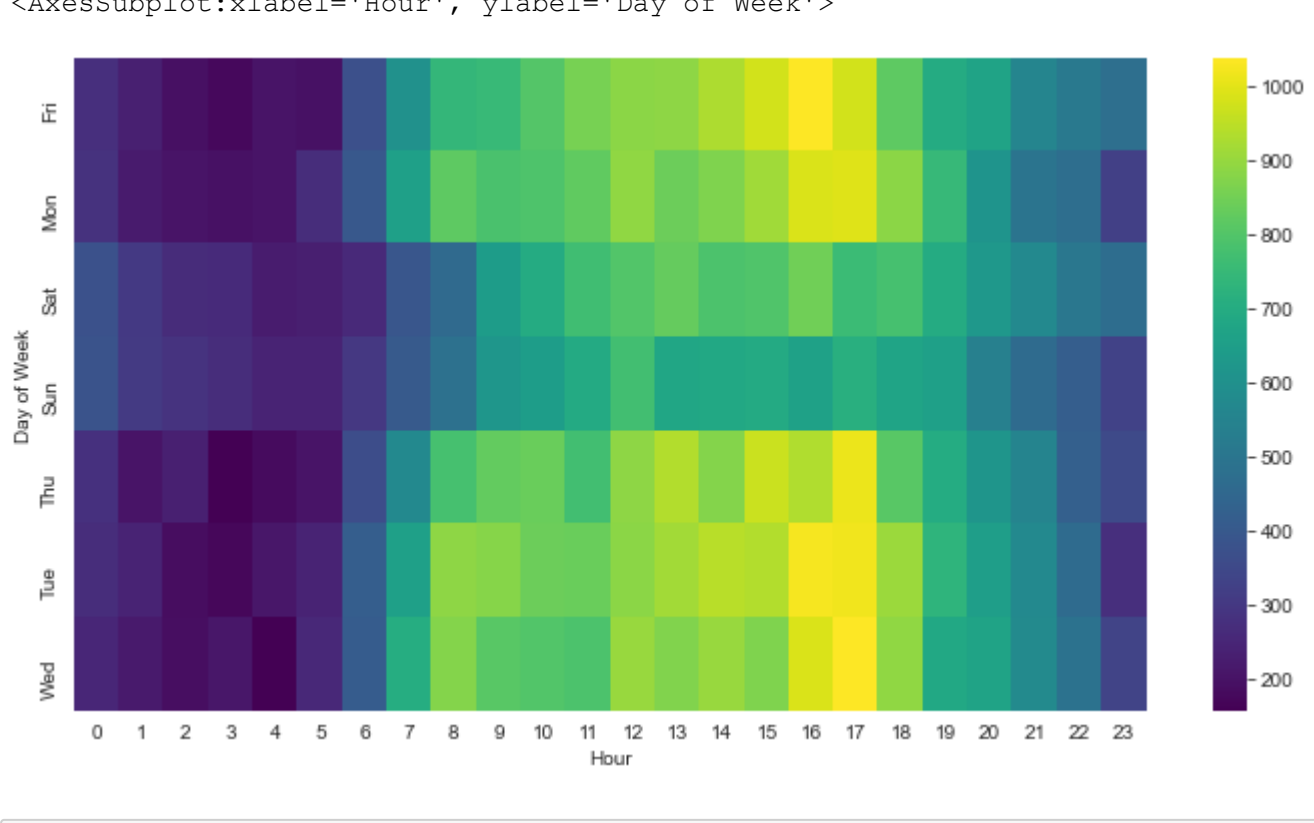
Hour	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	23
Day of Week																					
Fri	275	235	191	175	201	194	372	598	742	752	...	932	980	1039	980	820	696	667	559	514	474
Mon	282	221	201	194	204	267	397	653	819	786	...	869	913	989	997	885	746	613	497	472	325
Sat	375	301	263	260	224	231	257	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

5 rows × 24 columns

Now create a HeatMap using this new DataFrame.

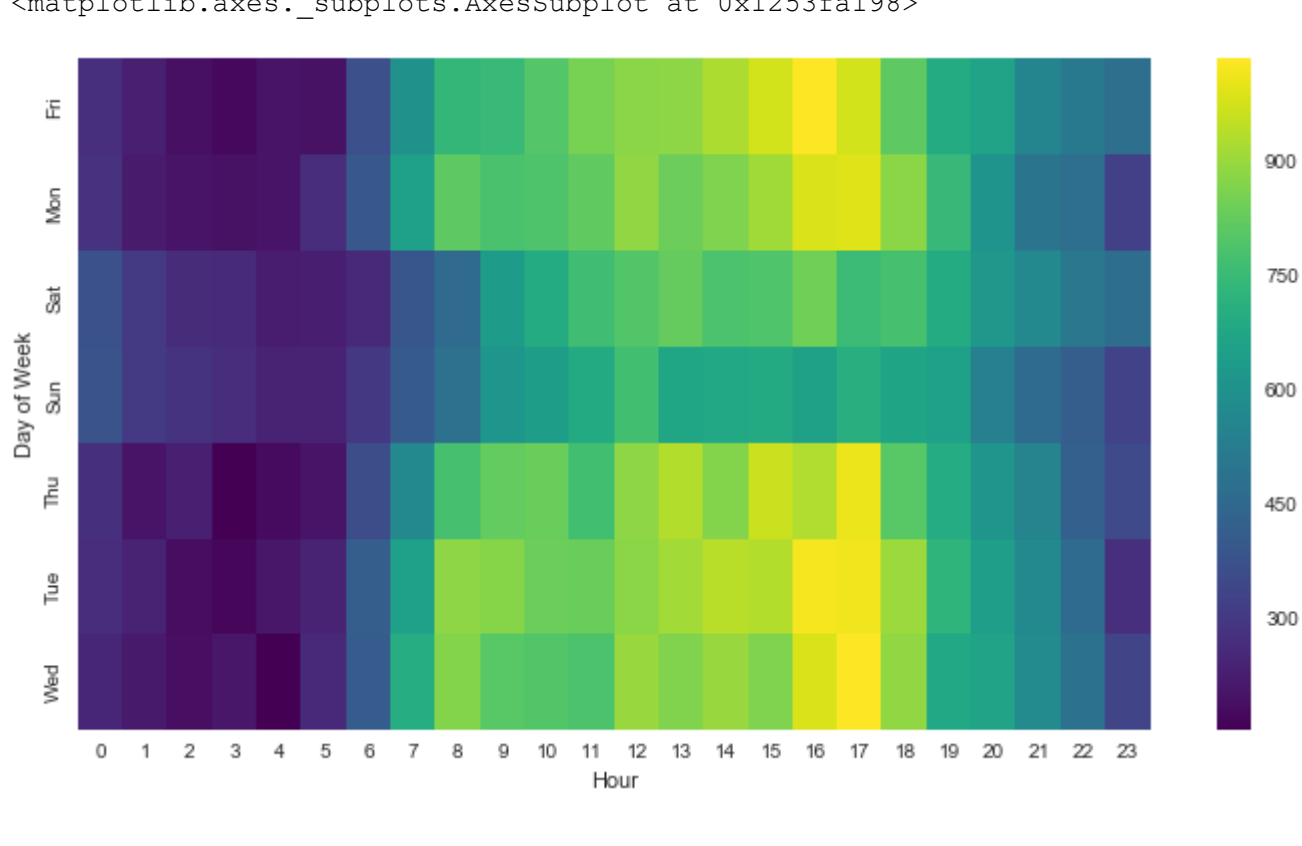
In [128]:
```
plt.figure(figsize=(12,6))
sns.heatmap(wk_hr_smy_pivot, cmap = 'viridis')
```

Out[128]:
```
<AxesSubplot:xlabel='Hour', ylabel='Day of Week'>
```



In [204]:

Out[204]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1253fa198>
```

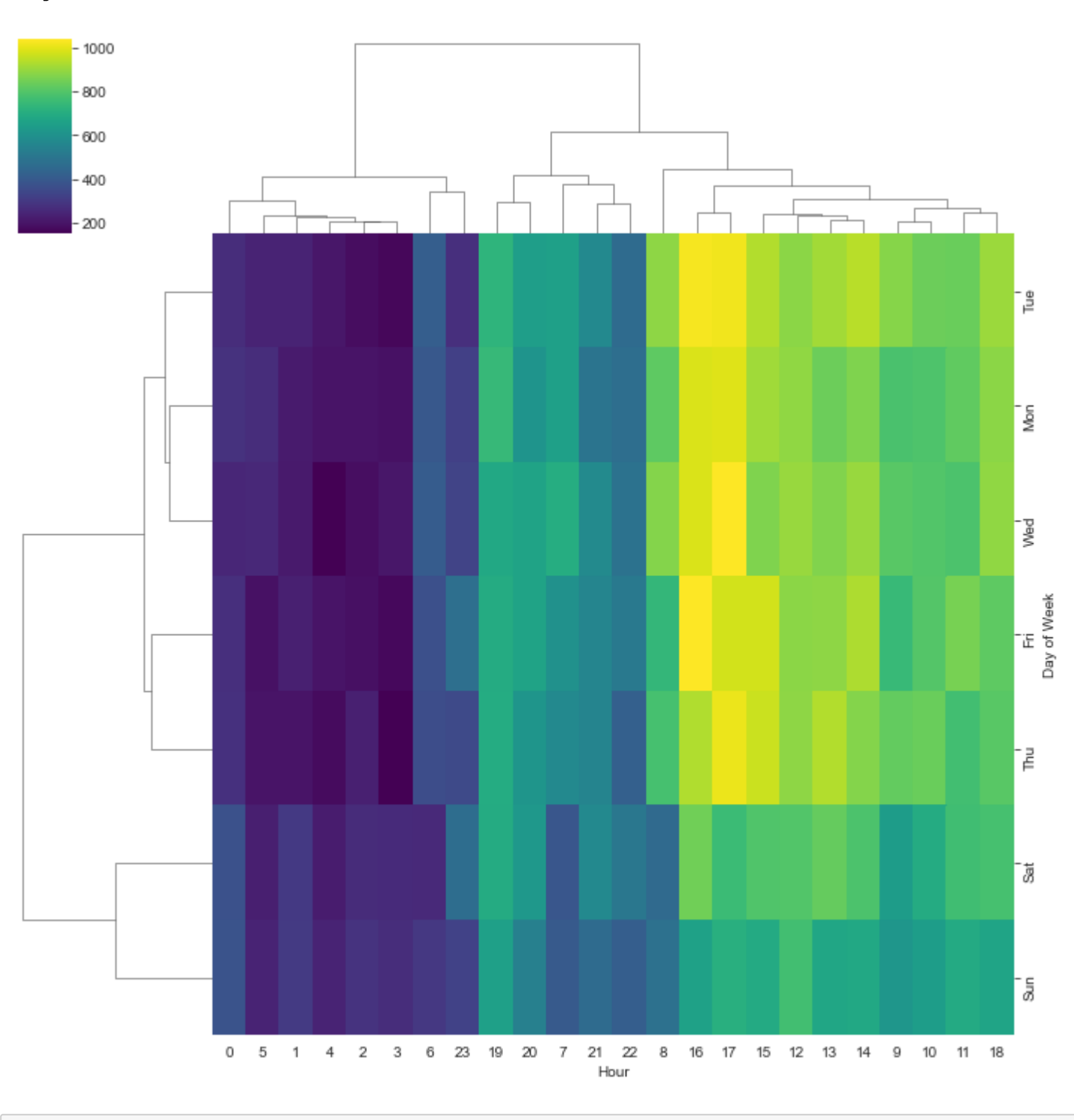


Now create a clustermap using this DataFrame.

In [133]:
```
plt.figure(figsize=(3,2))
sns.clustermap(wk_hr_smy_pivot, cmap = 'viridis')
```

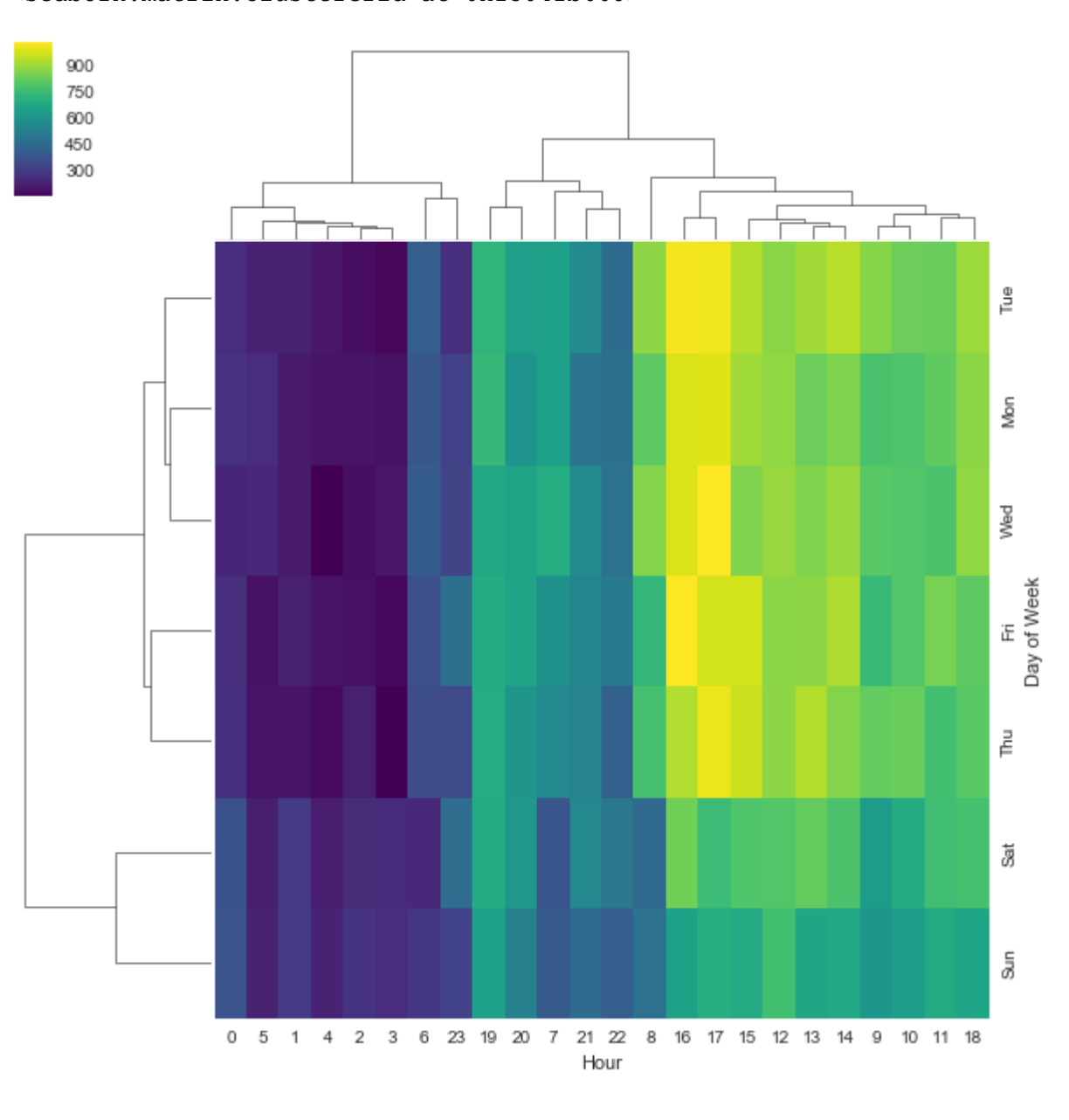
Out[133]:
```
<seaborn.matrix.ClusterGrid at 0x1b138513c40>
```

```
<Figure size 216x144 with 0 Axes>
```



In [205]:

Out[205]:
```
<seaborn.matrix.ClusterGrid at 0x1304fb668>
```



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

In [136]:
```
wk_mon_smy = df.groupby(['Day of Week', 'Month']).count()['lat']
```

In [139]:
```
wk_mon_smy_pivot = wk_mon_smy.unstack(level = -1)
```

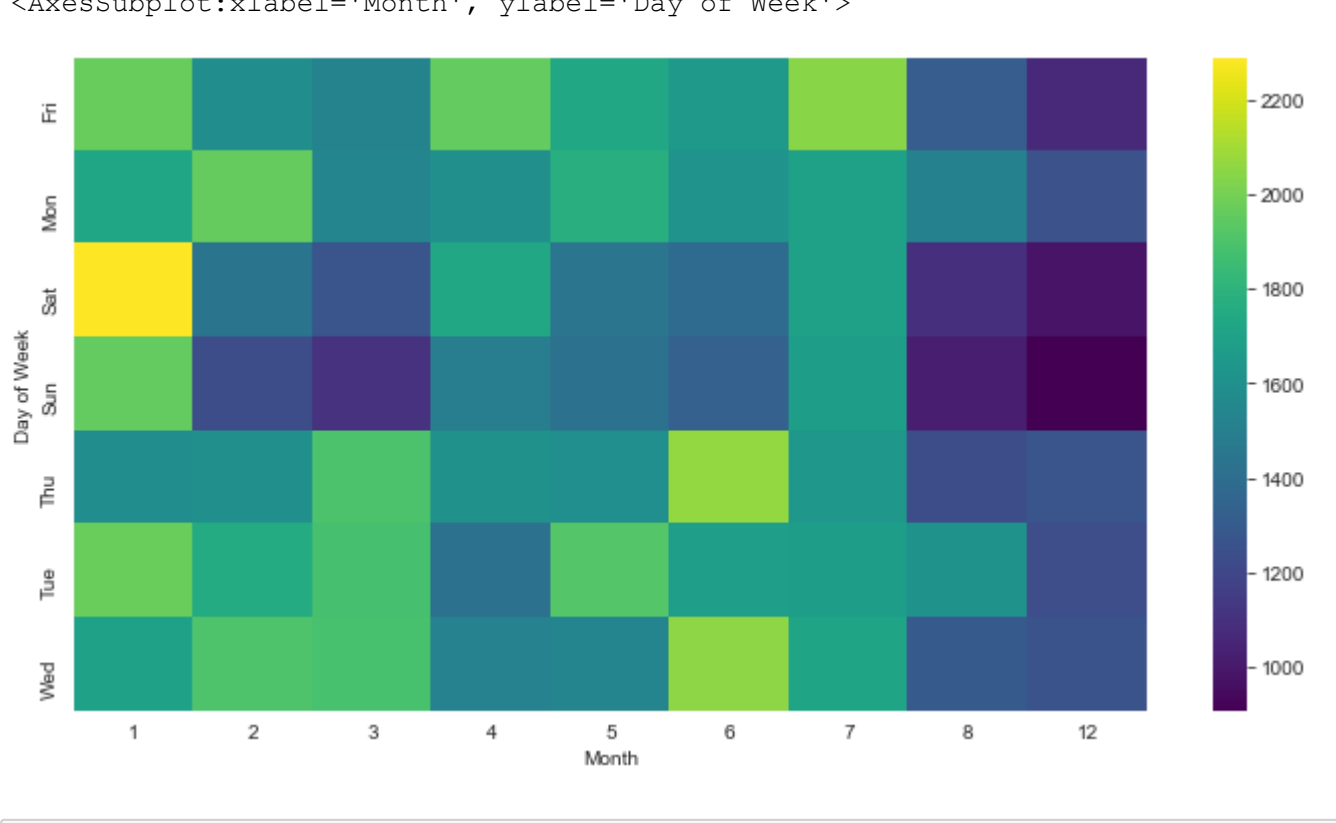
In [140]:
```
wk_mon_smy_pivot.head()
```

Out[140]:

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

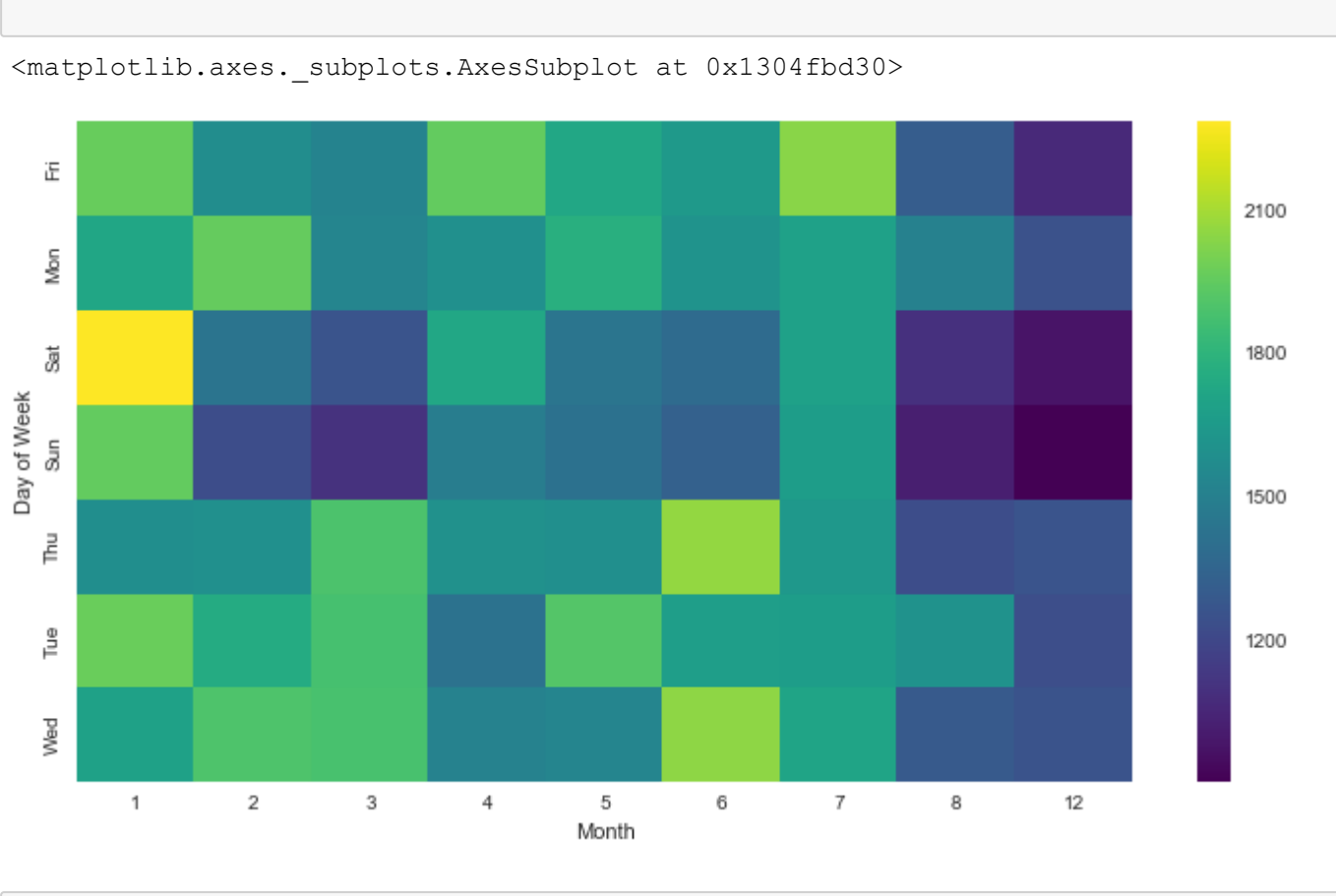
In [144]:
```
plt.figure(figsize = (12, 6))
sns.heatmap(wk_mon_smy_pivot, cmap = 'viridis')
```

Out[144]:
```
<AxesSubplot:xlabel='Month', ylabel='Day of Week'>
```



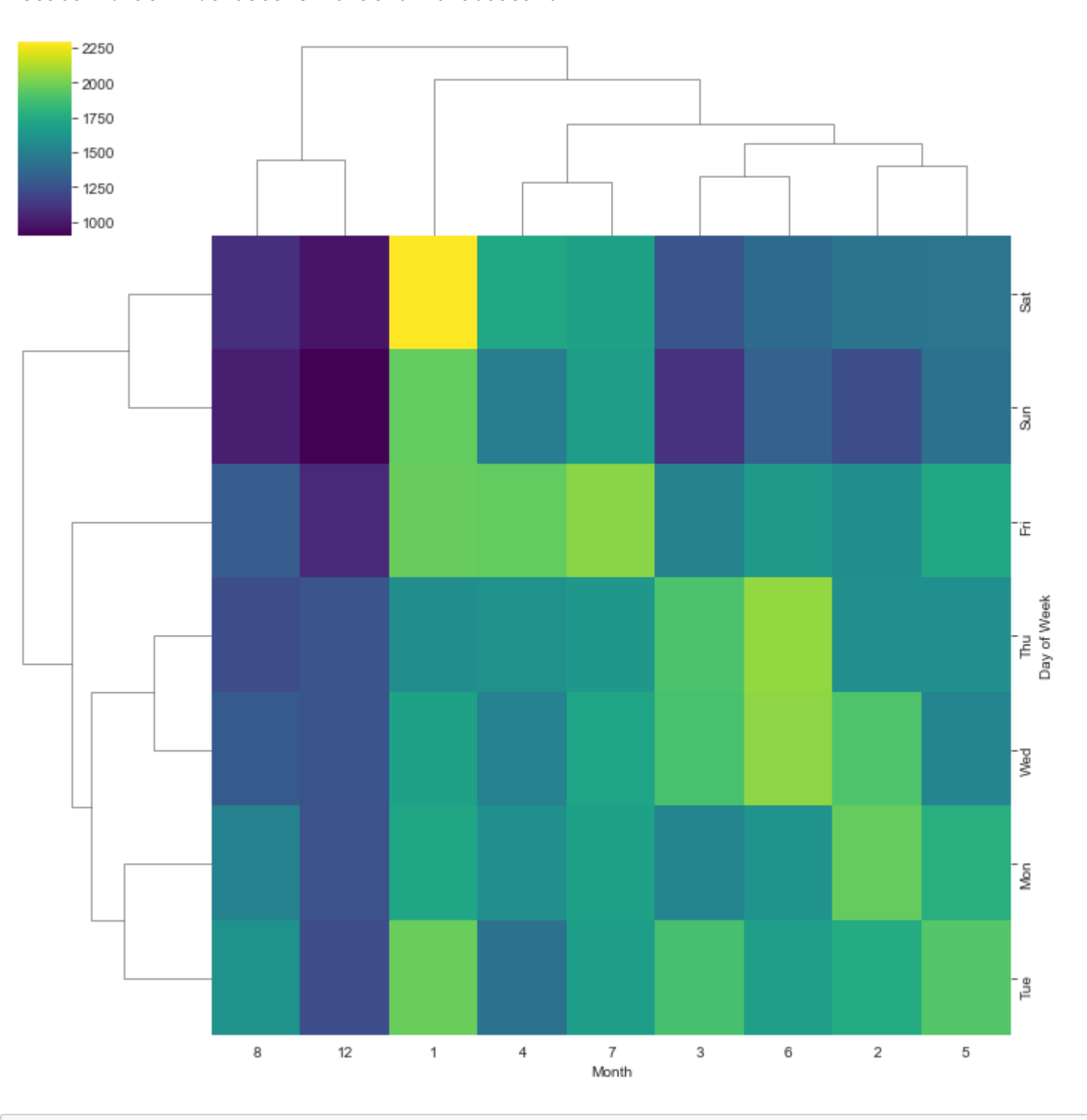
In [208]:

Out[208]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1304fbd30>
```



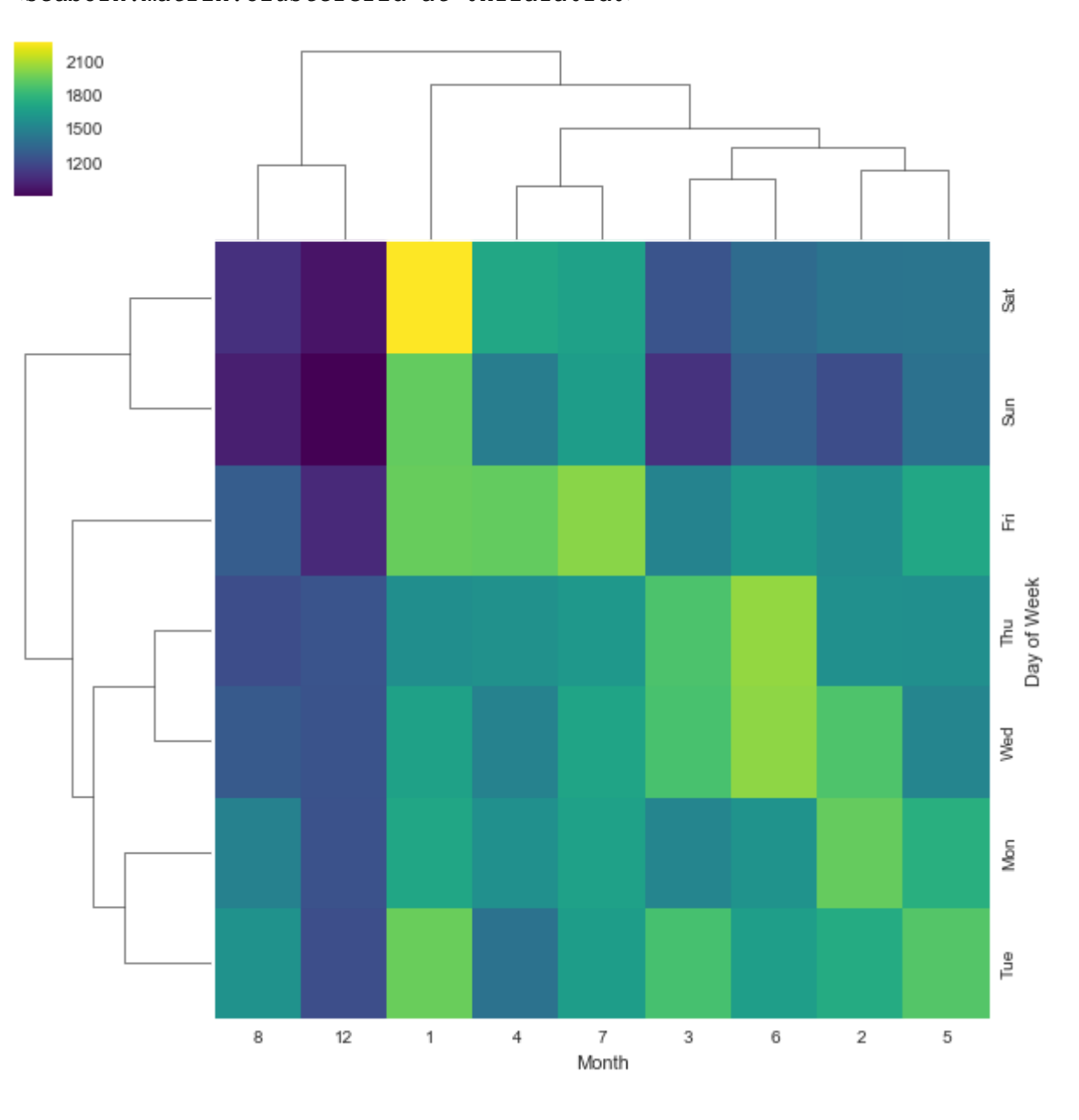
In [146]:
```
sns.clustermap(wk_mon_smy_pivot, cmap = 'viridis')
```

Out[146]:
```
<seaborn.matrix.ClusterGrid at 0x1b136b3caf0>
```



In [209]:

Out[209]:
```
<seaborn.matrix.ClusterGrid at 0x12a1a61d0>
```



Continue exploring the Data however you see fit!

Great Job!