Emergency - 911 Calls Project

For this project 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)

Evaluation:

During the notebook try to answer the following questions:

- Which features are available in the dataset?
- How many rows and columns does the dataset have?
- Which features are categorical?
- Which features are numerical?
- Which features contain blank, null or empty values?
- What are the data types for various features?
- How many zip codes does the dataset have?
- What are the top 5 zip codes for 911 calls?
- What are the top 5 townships (twp) for 911 calls?
- How many unique title of emergency codes are there?
- What is the most common Reason for a 911 call based off of this new column?

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
```

In [3]:
```
sns.set_style('whitegrid')
```

Read in the csv file as a dataframe called df

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

Check the info() of the df

How many rows and columns does the dataset have?

Which features are available in the dataset?

Which features contain blank, null or empty values?

In [5]:
```
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 [6]:
```
df.head()
```

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
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 16:47:36	NORRISTOWN	AIRY ST & SWEDE ST	1
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 16:56:52	LOWER POTTSGROVE	CHERRYWOOD CT & DEAD END	1

Basic Questions

How many zip codes does the dataset have?

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

Out[7]: 204

What are the top 5 zipcodes for 911 calls?

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

Out[8]:
```
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 [9]:
```
df['twp'].value_counts().head(5)
```

Out[9]:
```
LOWER MERION    55490
ABINGTON        39947
NORRISTOWN      37633
UPPER MERION    36010
CHELTENHAM      30574
Name: twp, dtype: int64
```

How many unique title codes are there in the 'title' column?

In [10]:
```
df['title'].nunique()
```

Out[10]: 148

Creating new features

- In the titles column there can be "Reasons/Departments" specified before the title code: EMS, Fire, and Traffic.
- Use a custom lambda expression to create a new column called "Reason" that contains this string value.
- Eg. the title column value is EMS: BACK PAINS/INJURY , then the Reason column value would be EMS.

In [6]:
```
df['Reason'] = df['title'].apply(lambda title: title.split(':')[0])
df['type'] = df['title'].apply(lambda title: title.split(':')[1]).apply(lambda title:
```

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

In [7]:
```
df['Reason'].value_counts()
```

Out[7]:
```
EMS        48877
Traffic    35695
Fire       14920
Name: Reason, dtype: int64
```

How many unique title of emergency codes are there?

In [8]:
```
df[df['Reason']=='EMS']['title'].apply(lambda title: title.split(':')[1]).nunique()
```

Out[8]: 68

Create the pivot table to count different types and reasons of 911 calls

In [9]:
```
table = pd.pivot_table(df,
                       values='e',
                       index=['Reason'],
                       columns=['type'],
                       aggfunc=np.sum,
              )
table
```

Out[9]:

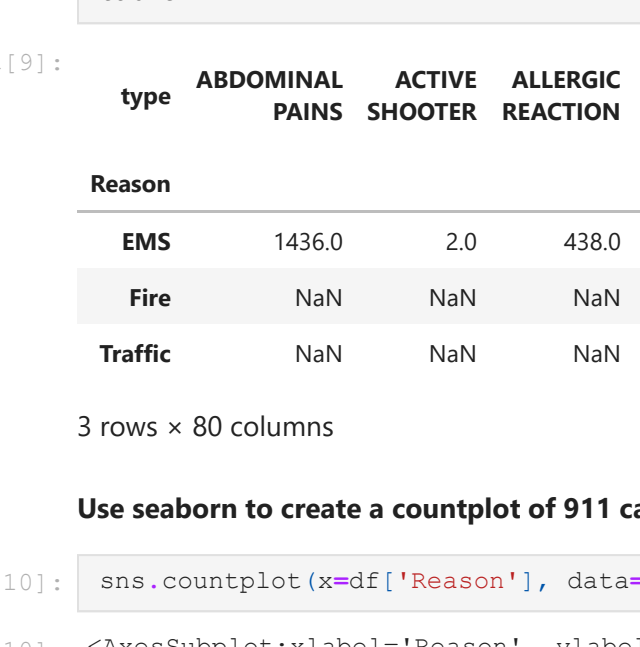
type	ABDOMINAL PAINS	ACTIVE SHOOTER	ALLERGIC REACTION	ALTERED MENTAL STATUS	AMPUTATION	ANIMAL BITE	APPLIANCE FIRE	ASSAULT VICTIM	BACK PAINS
Reason									
EMS	1436.0	2.0	438.0	1386.0	14.0	83.0	11.0	657.0	739.0
Fire	NaN	NaN	NaN	NaN	NaN	NaN	182.0	NaN	NaN
Traffic	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

3 rows × 80 columns

Use seaborn to create a countplot of 911 calls by Reason.

In [10]:
```
sns.countplot(x=df['Reason'], data=df, palette='rainbow')
```

Out[10]: <AxesSubplot:xlabel='Reason', ylabel='count'>



What are the data types for various features?

Which features are numerical?

In [16]:
```
type(df['lat'].iloc[0])
```

Out[16]: numpy.float64

In [17]:
```
type(df['lng'].iloc[0])
```

Out[17]: numpy.float64

In [18]:
```
type(df['zip'].iloc[0])
```

Out[18]: numpy.float64

In [19]:
```
type(df['e'].iloc[0])
```

Out[19]: numpy.int64

Which features are categorical?

In [20]:
```
type(df['desc'].iloc[0])
```

Out[20]: str

In [21]:
```
type(df['title'].iloc[0])
```

Out[21]: str

In [22]:
```
type(df['timeStamp'].iloc[0])
```

Out[22]: str

In [23]:
```
type(df['twp'].iloc[0])
```

Out[23]: str

In [24]:
```
type(df['addr'].iloc[0])
```

Out[24]: str

Convert the column from strings to DateTime objects.

In [11]:
```
df['timeStamp'] = pd.to_datetime(df['timeStamp'])
```

- Now the timestamp column are DateTime objects.
- Create 3 new columns called Hour, Month, and Day of Week based off of the timeStamp column.

In [12]:
```
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)
```

- Define the Day of Week is an integer 0-6.
- Map the actual string names to the day of the week with dictionary

In [13]:
```
dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}
```

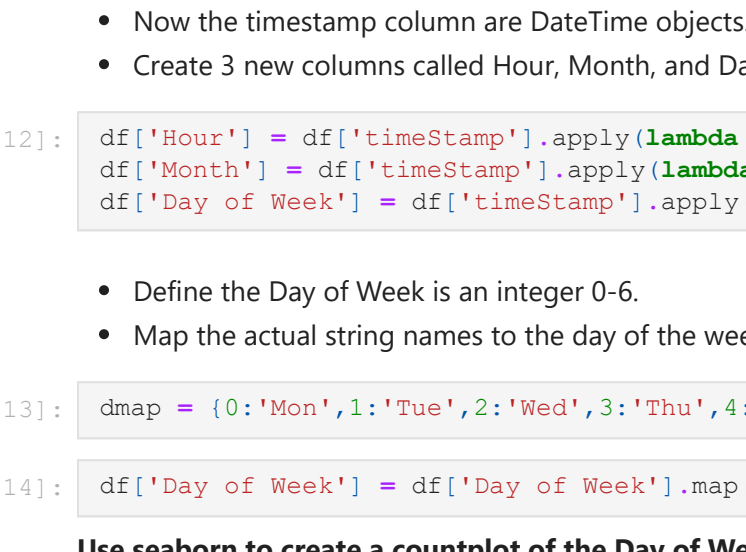
In [14]:
```
df['Day of Week'] = df['Day of Week'].map(dmap)
```

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

In [15]:
```
sns.countplot(x='Day of Week', data=df, hue='Reason', palette='rainbow')

# Relocate the legend
plt.legend(bbox_to_anchor=(1.05, 1),
           loc='upper left',
           borderaxespad=0,
           edgecolor='white')
```

Out[15]: <matplotlib.legend.Legend at 0x20caf56fac0>



Now do the same for Month:

In [16]:
```
sns.countplot(x='Month', data=df, hue='Reason', palette='rainbow')

# Relocate the legend
plt.legend(bbox_to_anchor=(1.05, 1),
           loc='upper left',
           borderaxespad=0,
           edgecolor='white')
```

Out[16]: <matplotlib.legend.Legend at 0x20cb1e5c100>

Something strange about the Plot

- There is missing some Months: 9,10, and 11 are not here.
- Try to see if we could fill in this information by plotting the information in another way, possibly a simple line plot that fills in the missing months with pandas.
- Create a groupby object called byMonth to group the DataFrame by the month column.
- Use count method for aggregation.

In [17]:
```
byMonth = df.groupby('Month').count()
byMonth.head()
```

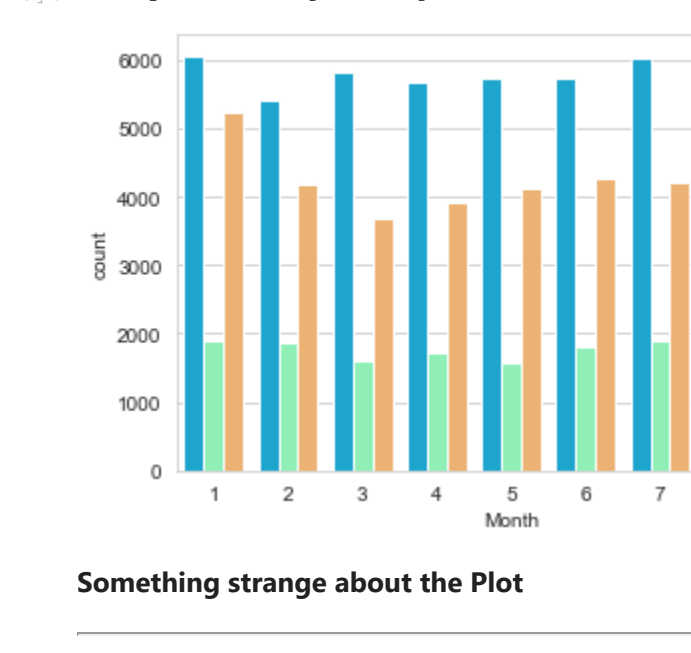
Out[17]:

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

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

In [18]:
```
byMonth['twp'].plot()
```

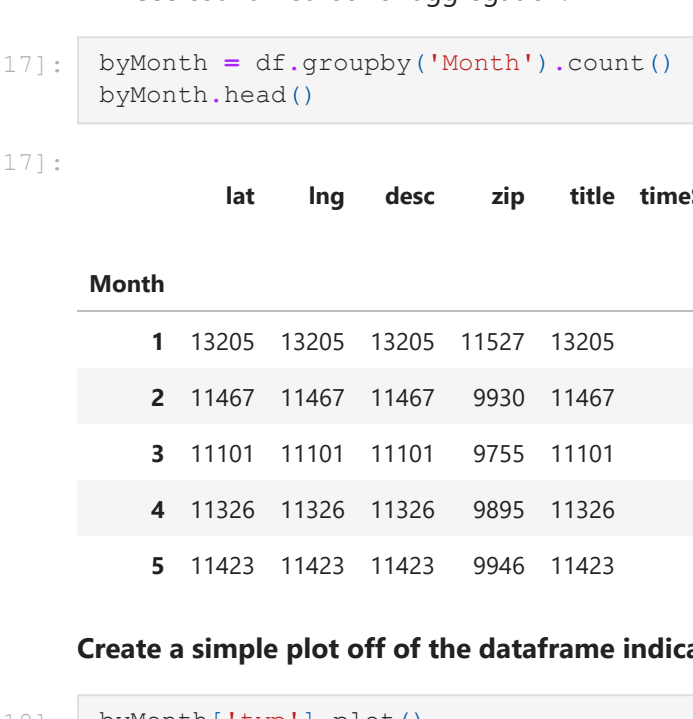
Out[18]: <AxesSubplot:xlabel='Month'>



Use seaborn to create a linear fit on the number of calls per month. May need to reset the index to a column.

In [19]:
```
sns.lmplot(x='Month', y='twp', data=byMonth.reset_index())
```

Out[19]: <seaborn.axisgrid.FacetGrid at 0x20caf7cc070>



Create a new column called 'Date' that contains the date from the timeStamp column.

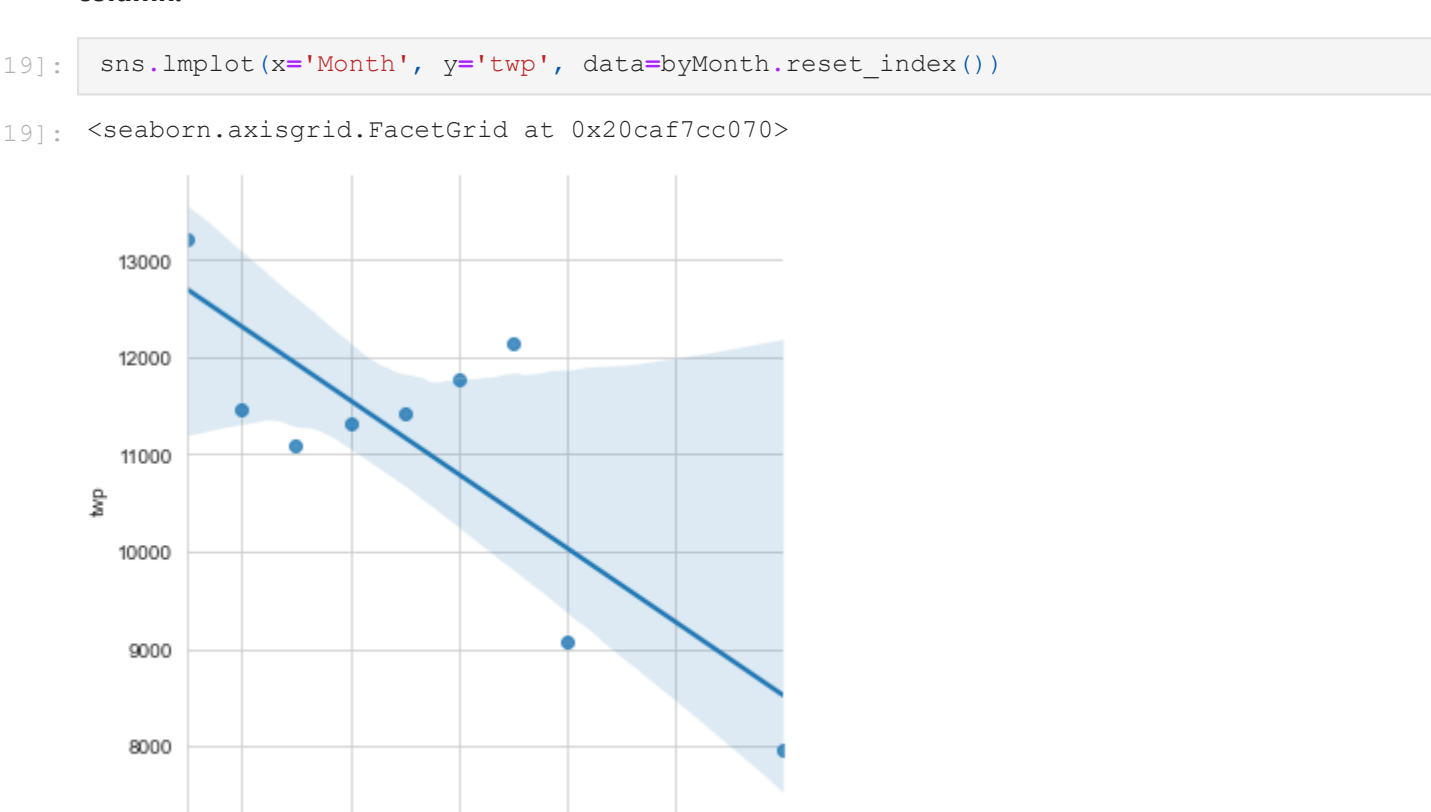
In [20]:
```
df['Date'] = df['timeStamp'].apply(lambda time:time.date())
byDate = df.groupby(['Date']).count()
byDate.head()
```

Out[20]:

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

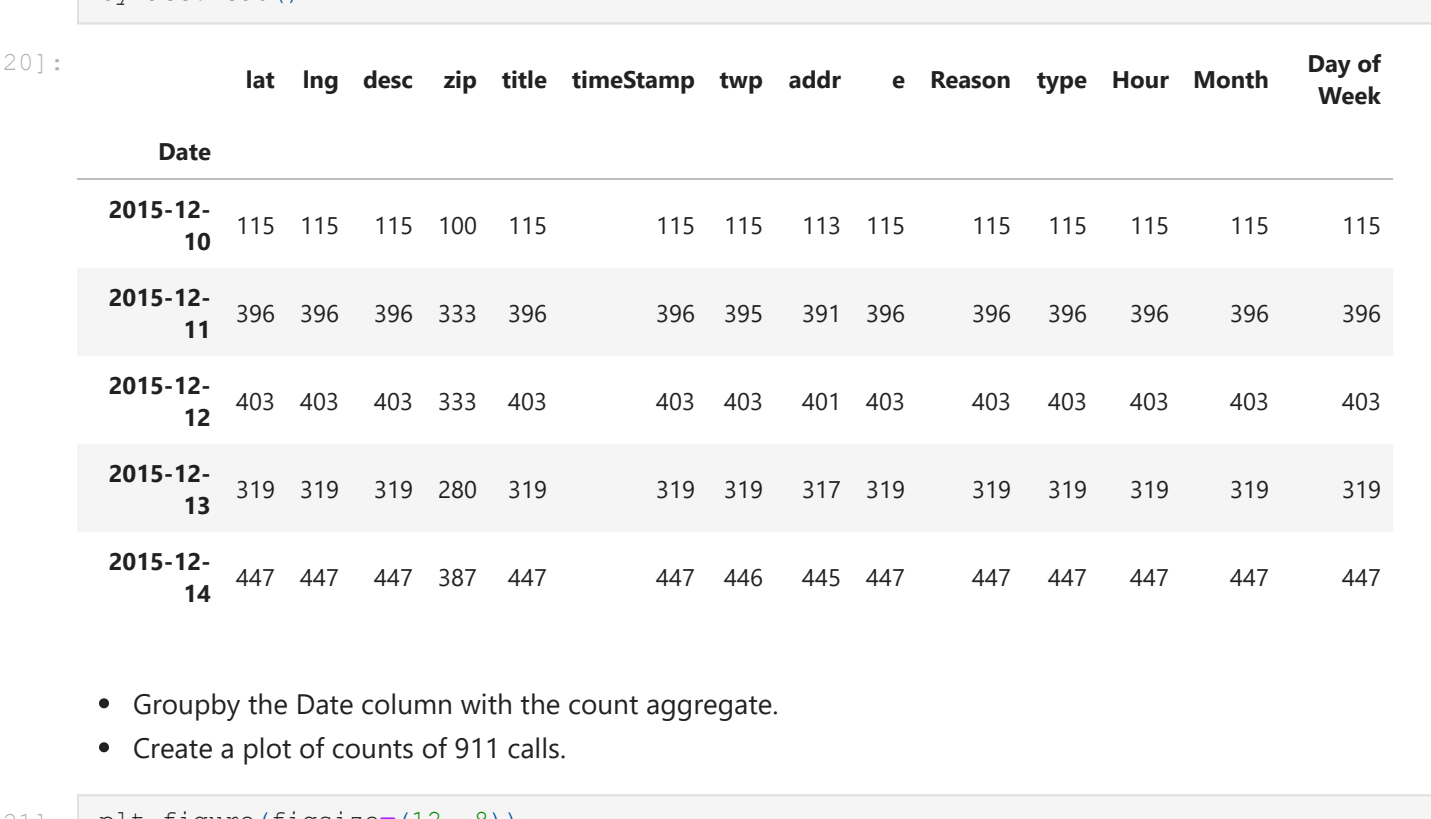
- Groupby the Date column with the count aggregate.
- Create a plot of counts of 911 calls.

In [21]:
```
plt.figure(figsize=(12, 8))
byDate['twp'].plot()
plt.tight_layout()
```

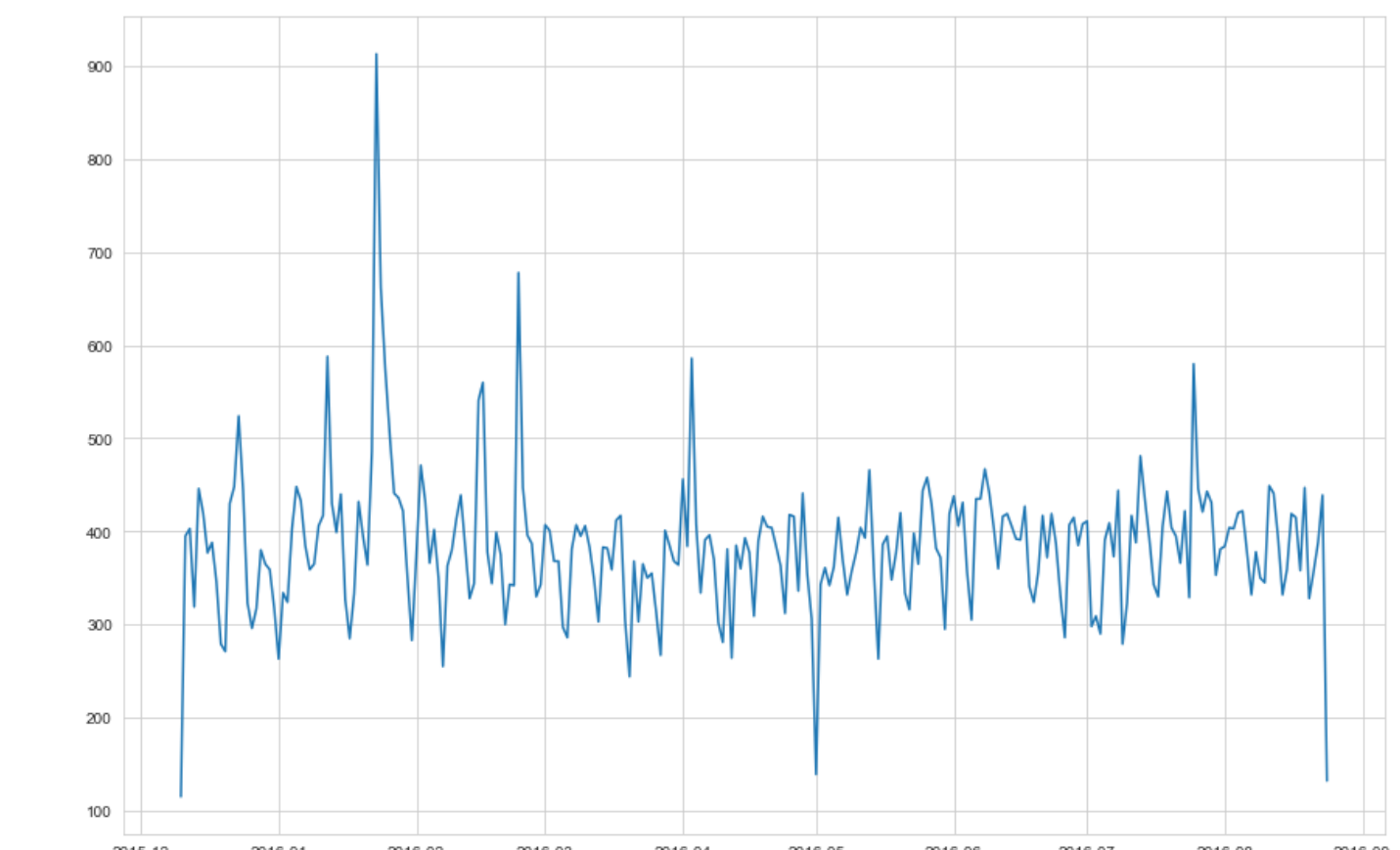


Recreate 3 separate plots with each plot representing a Reason for the 911 call

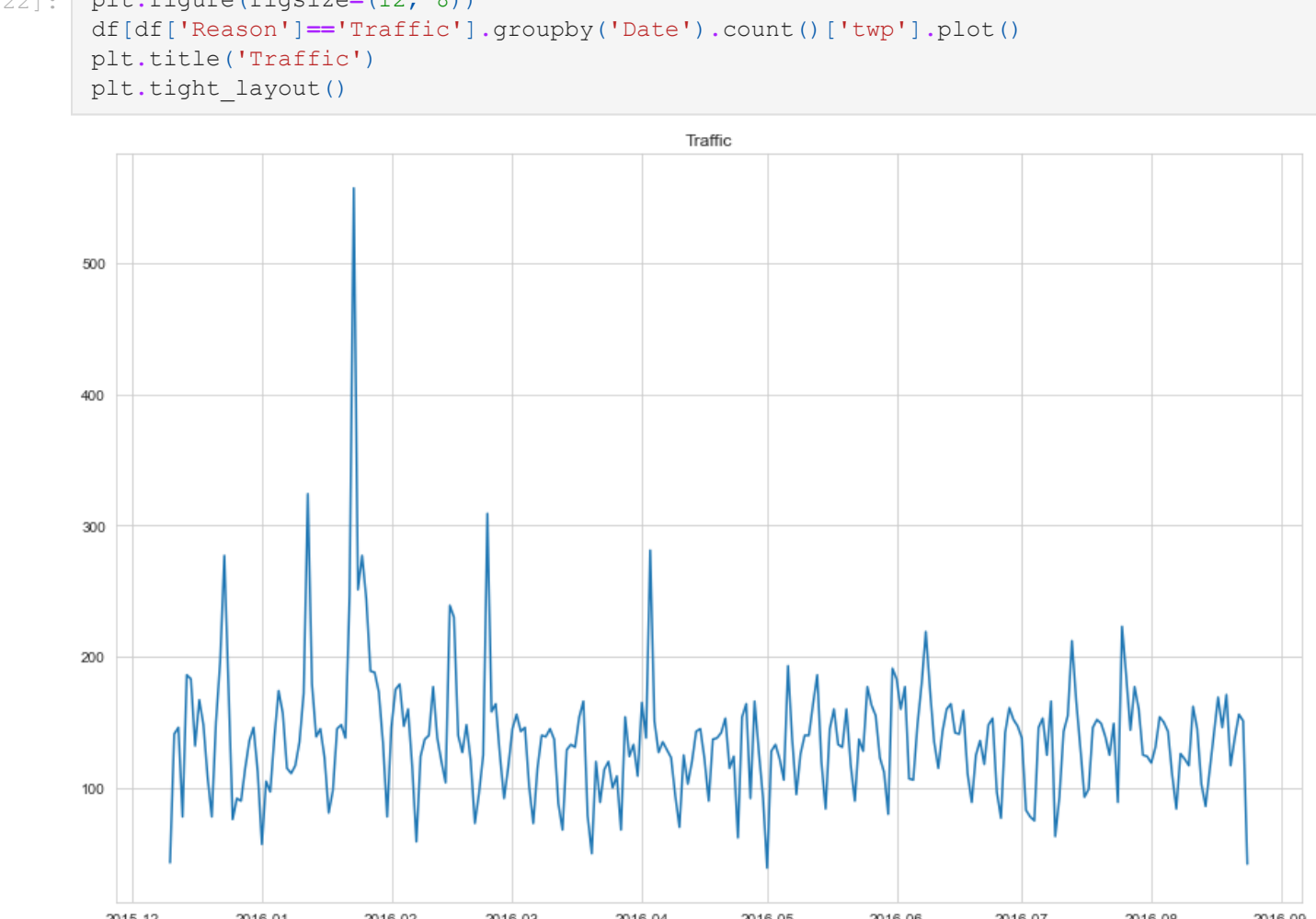
In [22]:
```
plt.figure(figsize=(12, 8))
df[df['Reason']=='Traffic'].groupby('Date').count()['twp'].plot()
plt.title('Traffic')
plt.tight_layout()
```



In [23]:
```
plt.figure(figsize=(12, 8))
df[df['Reason']=='Fire'].groupby('Date').count()['twp'].plot()
plt.title('Fire')
plt.tight_layout()
```



In [24]:
```
plt.figure(figsize=(12, 8))
df[df['Reason']=='EMS'].groupby('Date').count()['twp'].plot()
plt.title('EMS')
plt.tight_layout()
```



- Create heatmaps with seaborn and data.
- Restructure the dataframe so that the columns become the Hours and the Index becomes the Day of the Week.
- Try to combine groupby with an unstack method.

In [25]:
```
# Solution 1
dayHour = df.groupby(['Day of Week', 'Hour']).count()['twp'].unstack(level=-1)
dayHour.head()
```

Out[25]:

Hour	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22
Day of Week																				
Fri	275	235	191	175	201	194	372	598	742	752	...	931	980	1039	980	819	696	667	558	514
Mon	282	220	201	194	204	267	397	653	819	785	...	869	913	988	997	885	746	612	496	471
Sat	373	301	263	260	224	231	257	391	459	640	...	789	796	848	757	778	696	627	571	[illegible]
Sun	383	304	285	268	242	240	300	401	483	620	...	684	691	663	714	670	654	537	461	415
Thu	278	202	233	159	182	203	362	570	777	828	...	876	969	935	1011	810	698	617	553	424

5 rows × 24 columns

In [26]:
```
#using pivot_table
table2 = pd.pivot_table(df,
                        values='e',
                        index=['Day of Week'],
                        columns=['Hour'],
                        aggfunc=np.sum,
              )
table2
```

Out[26]:

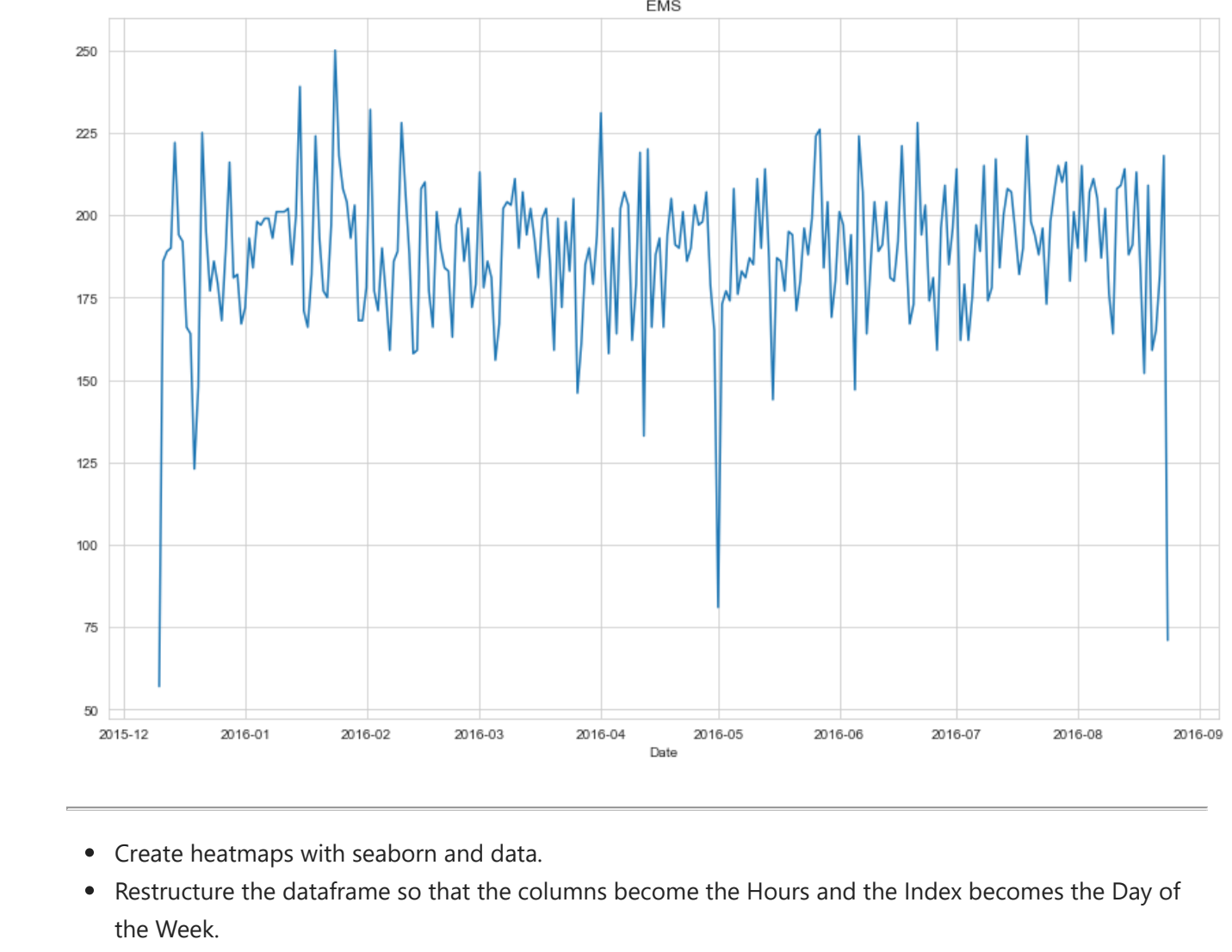
Hour	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22
Day of Week																				
Fri	275	235	191	175	201	194	372	598	742	752	...	932	980	1039	980	820	696	667	559	514
Mon	282	221	201	194	204	267	397	653	819	786	...	869	913	989	997	885	746	613	497	472
Sat	375	301	263	260	224	231	257	391	459	640	...	789	796	848	757	778	696	628	572	506
Sun	383	306	286	268	242	240	300	402	483	620	...	684	691	663	714	670	655	537	461	415
Thu	278	202	233	159	182	203	362	570	777	828	...	876	969	935	1013	810	698	617	553	424
Tue	269	240	186	170	209	239	415	655	889	880	...	943	938	1026	1019	905	731	647	571	462
Wed	250	216	189	209	156	255	410	701	875	808	...	904	867	990	1037	894	686	668	575	490

7 rows × 24 columns

Create a HeatMap with new DataFrame.

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

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



Create a clustermap whit new DataFrame.

In [29]:
```
sns.clustermap(dayHour, cmap='coolwarm')
```

Out[29]: <seaborn.matrix.ClusterGrid at 0x20cb003cd00>

Repeat these same plots and operations for a DataFrame that shows the Month as the column.

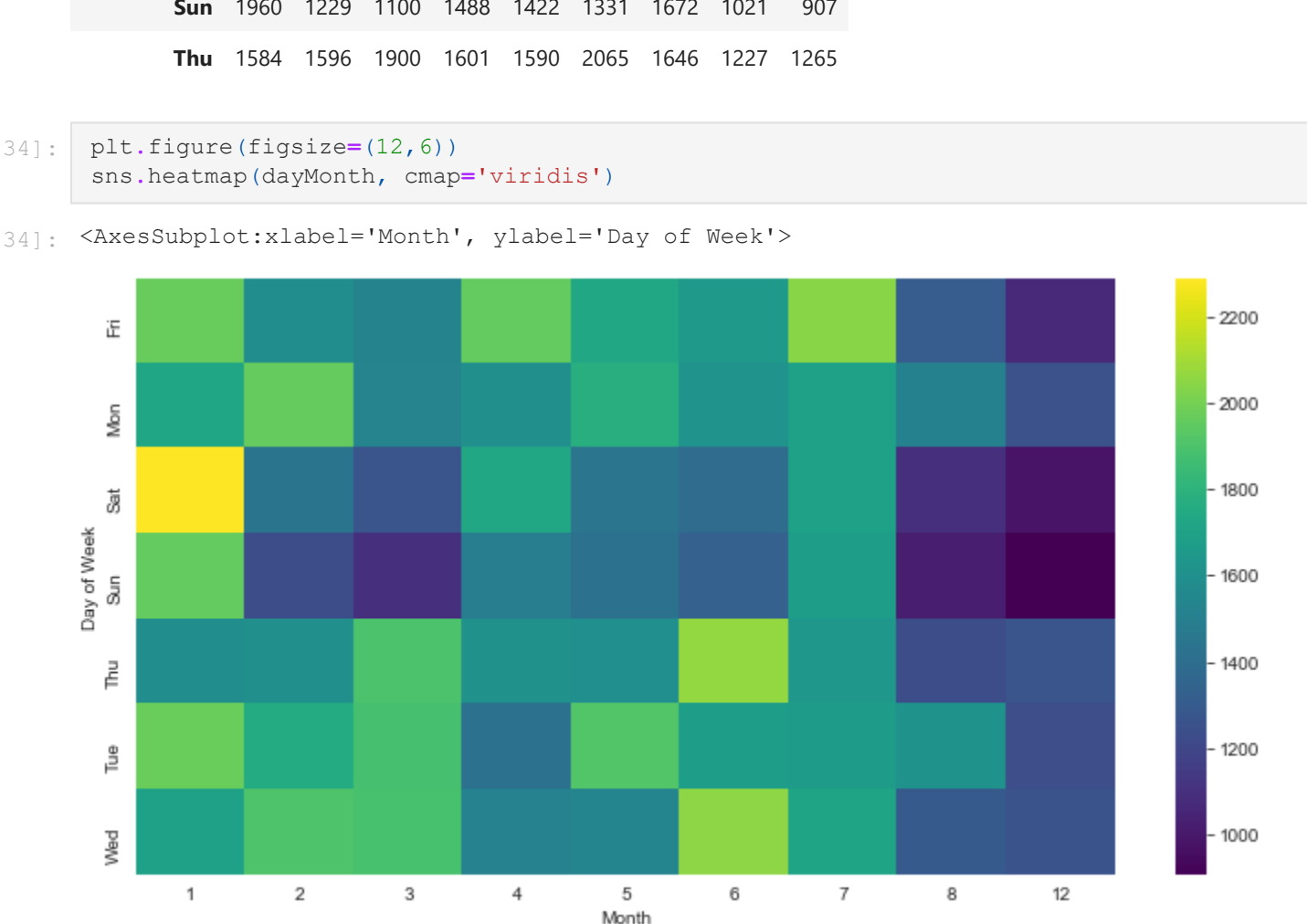
In [31]:
```
dayMonth = df.groupby(['Day of Week', 'Month']).count()['twp'].unstack(level=-1)
dayMonth.head()
```

Out[31]:

Month	1	2	3	4	5	6	7	8	12
Day of Week									
Fri	1970	1581	1523	1958	1730	1649	2045	1310	1064
Mon	1727	1964	1533	1597	1779	1617	1692	1509	1256
Sat	2290	1440	1264	1732	1444	1388	1695	1099	978
Sun	1960	1229	1100	1488	1422	1331	1672	1021	907
Thu	1584	1596	1900	1601	1590	2065	1646	1227	1265

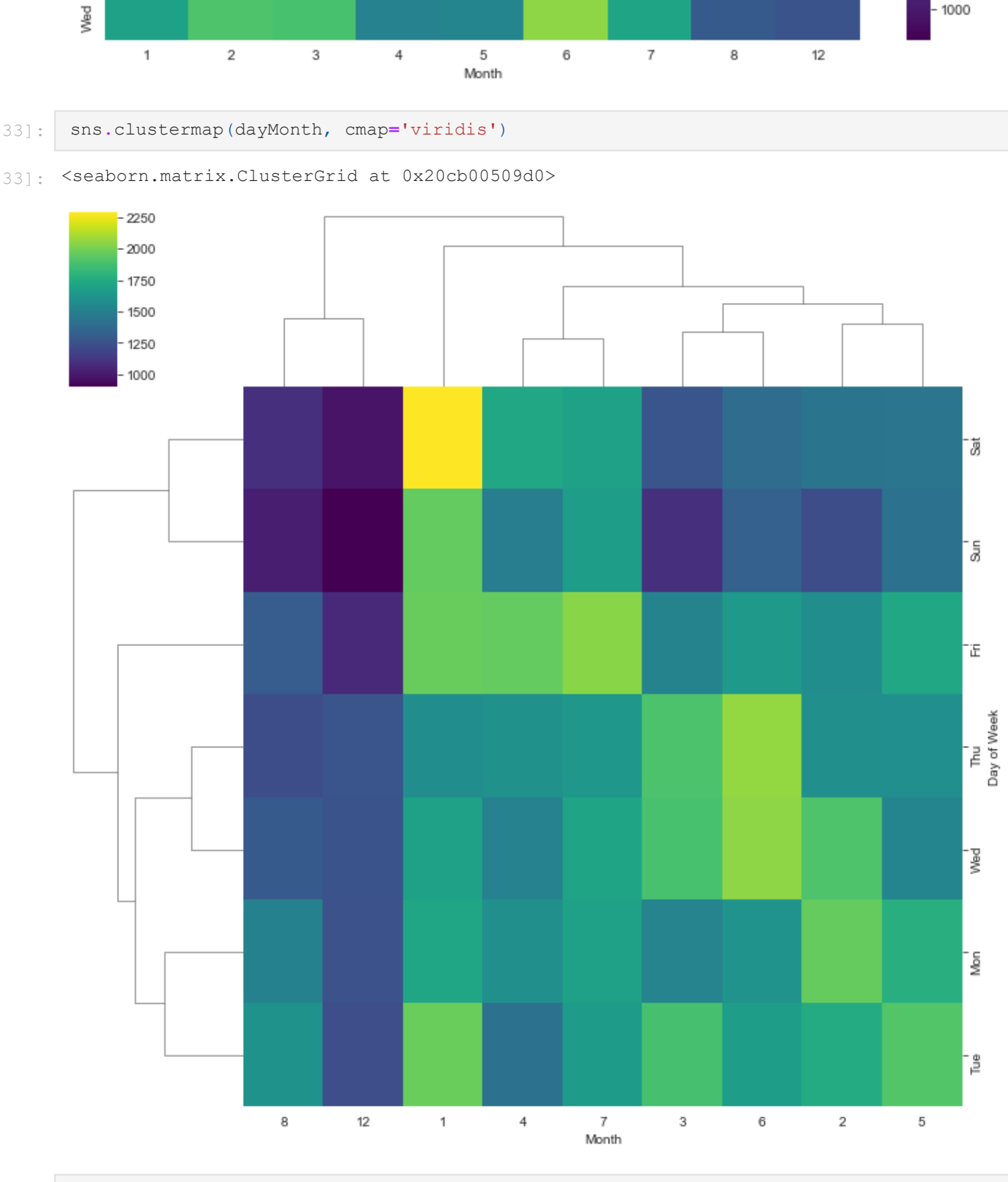
In [34]:
```
plt.figure(figsize=(12,6))
sns.heatmap(dayMonth, cmap='viridis')
```

Out[34]: <AxesSubplot:xlabel='Month', ylabel='Day of Week'>



In [33]:
```
sns.clustermap(dayMonth, cmap='viridis')
```

Out[33]: <seaborn.matrix.ClusterGrid at 0x20cb00509d0>



In [ ]: