01-911 Calls Data Capstone Project

March 16, 2022

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 **

[2]:
```
import numpy as np
import pandas as pd
```

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

[3]:
```
import matplotlib.pyplot as plt
```

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

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

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

[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 **

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

[6]:
```
         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…
3  40.116153 -75.343513  AIRY ST & SWEDE ST;  NORRISTOWN; Station 308A;…
4  40.251492 -75.603350  CHERRYWOOD CT & DEAD END;  LOWER POTTSGROVE; S…

       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
3  19401.0   EMS: CARDIAC EMERGENCY  2015-12-10 17:40:01         NORRISTOWN
4      NaN           EMS: DIZZINESS  2015-12-10 17:40:01   LOWER POTTSGROVE

                        addr  e
0      REINDEER CT & DEAD END  1
1  BRIAR PATH & WHITEMARSH LN  1
2                    HAWS AVE  1
3          AIRY ST & SWEDE ST  1
4    CHERRYWOOD CT & DEAD END  1
```

1.2 Basic Questions

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

[7]:
```
df.zip.value_counts().nlargest(5)
```

[7]:
```
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? **

[8]:
```
df.twp.value_counts().nlargest(5)
```

[8]:
```
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? **

[9]:
```
df.title.nunique()
```

[9]:
```
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.

[10]:
```
df.title[0].split(':')[0]
```

[10]:
```
'EMS'
```

[11]:
```
df['Reason'] = df.title.apply(lambda x: x.split(':')[0]  )
```

[12]:
```
df.head()
```

[12]:

	lat	lng	desc	zip	title	timeStamp	twp
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
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP…	19446.0	EMS: DIABETIC EMERGENCY	2015-12-10 17:40:00	HATFIELD TOWNSHIP
2	40.121182	-75.351975	HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21-St…				
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;…				
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S…				

```
2  19401.0      Fire: GAS-ODOR/LEAK  2015-12-10 17:40:00         NORRISTOWN
3  19401.0   EMS: CARDIAC EMERGENCY  2015-12-10 17:40:01         NORRISTOWN
4      NaN           EMS: DIZZINESS  2015-12-10 17:40:01   LOWER POTTSGROVE

                         addr  e Reason
0     REINDEER CT & DEAD END  1    EMS
1  BRIAR PATH & WHITEMARSH LN  1    EMS
2                   HAWS AVE  1   Fire
3          AIRY ST & SWEDE ST  1    EMS
4    CHERRYWOOD CT & DEAD END  1    EMS
```

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

[13]:
```
df.Reason.value_counts()
```

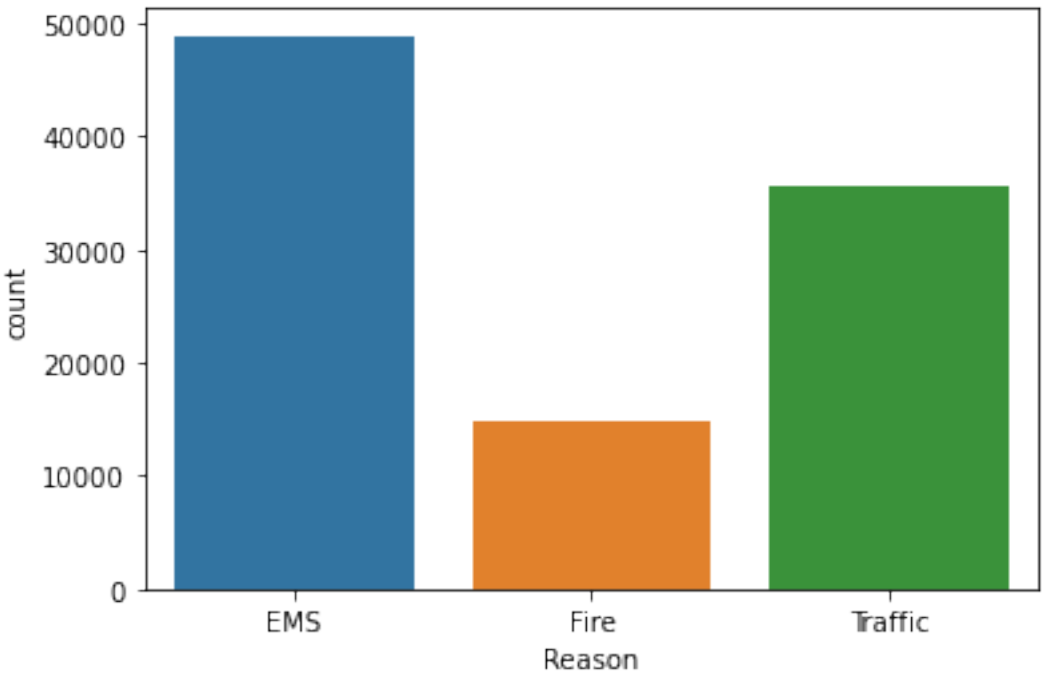
[13]:
```
EMS        48877
Traffic    35695
Fire       14920
Name: Reason, dtype: int64
```

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

[14]:
```
import seaborn as sns
sns.countplot(df.Reason)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36:
FutureWarning: Pass the following variable as a keyword arg: x. From version
0.12, the only valid positional argument will be `data`, and passing other
arguments without an explicit keyword will result in an error or
misinterpretation.
  warnings.warn(
```

[14]:
```
<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? **

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

[15]:
```
str
```

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

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

[17]:
```
df.timeStamp.dt.dayofweek
```

[17]:
```
0        3
1        3
2        3
3        3
4        3
        ..
99487    2
99488    2
99489    2
99490    2
```

```
99491    2
Name: timeStamp, Length: 99492, dtype: int64
```

[18]:
```
df['Year'] = df.timeStamp.apply(lambda x: x.year)
```

[19]:
```
df['Hour'] = df.timeStamp.apply(lambda x: x.hour)
```

[20]:
```
df['Month'] = df.timeStamp.apply(lambda x: x.month)
df['Day'] = df.timeStamp.apply(lambda x: x.dayofweek)
df.head()
```

[20]:

	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…	
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;…	
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S…	

	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	
3	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 17:40:01	NORRISTOWN	
4	NaN	EMS: DIZZINESS	2015-12-10 17:40:01	LOWER POTTSGROVE	

	addr	e	Reason	Year	Hour	Month	Day
0	REINDEER CT & DEAD END	1	EMS	2015	17	12	3
1	BRIAR PATH & WHITEMARSH LN	1	EMS	2015	17	12	3
2	HAWS AVE	1	Fire	2015	17	12	3
3	AIRY ST & SWEDE ST	1	EMS	2015	17	12	3
4	CHERRYWOOD CT & DEAD END	1	EMS	2015	17	12	3

[21]:
```
dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}
```

[22]:
```
df['Day_of_Week']= df['Day'].map(dmap)
```

[23]:
```
df.head()
```

[23]:

	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…	
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;…	
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S…	

	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
3  19401.0   EMS: CARDIAC EMERGENCY 2015-12-10 17:40:01         NORRISTOWN
4      NaN           EMS: DIZZINESS 2015-12-10 17:40:01   LOWER POTTSGROVE

                        addr  e Reason  Year  Hour  Month  Day Day_of_Week
0      REINDEER CT & DEAD END  1    EMS  2015    17     12    3         Thu
1  BRIAR PATH & WHITEMARSH LN  1    EMS  2015    17     12    3         Thu
2                   HAWS AVE  1   Fire  2015    17     12    3         Thu
3          AIRY ST & SWEDE ST  1    EMS  2015    17     12    3         Thu
4    CHERRYWOOD CT & DEAD END  1    EMS  2015    17     12    3         Thu
```

** You can now grab specific attributes from a Datetime object by calling them. For example:**

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

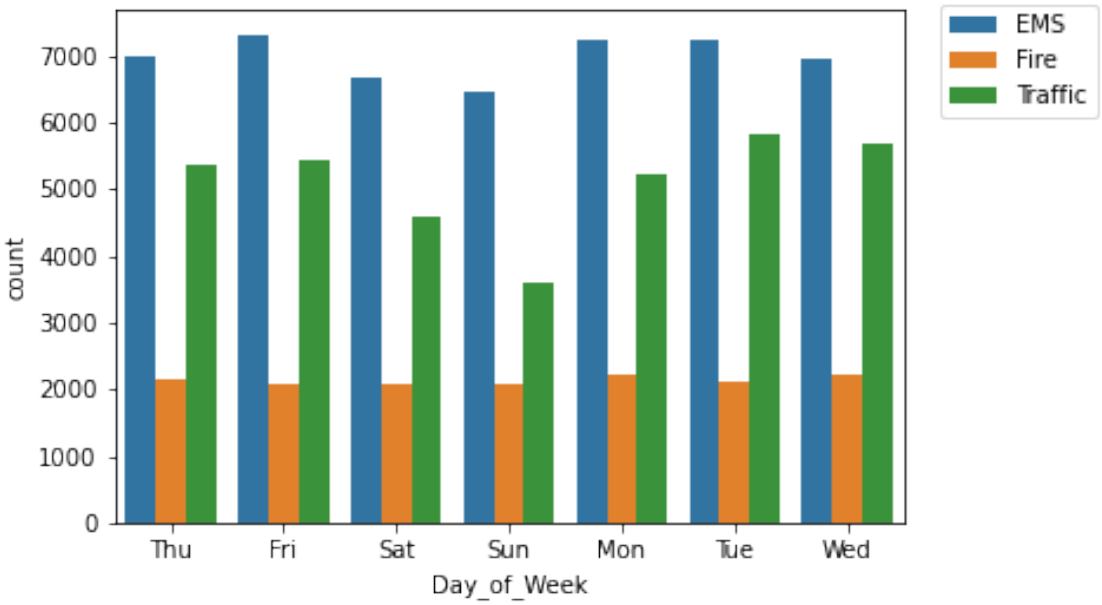
You can use Jupyter's tab method to explore the various attributes you can call. Now that the timestamp column are actually DateTime objects, use .apply() to create 3 new columns called Hour, Month, and Day of Week. You will create these columns based off of the timeStamp column, reference the solutions if you get stuck on this step.

** 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'}
```

[24]:
```
sns.countplot(x = "Day_of_Week", data = df, hue = "Reason")
plt.legend(loc='center right',bbox_to_anchor=(1.26, .9))
```

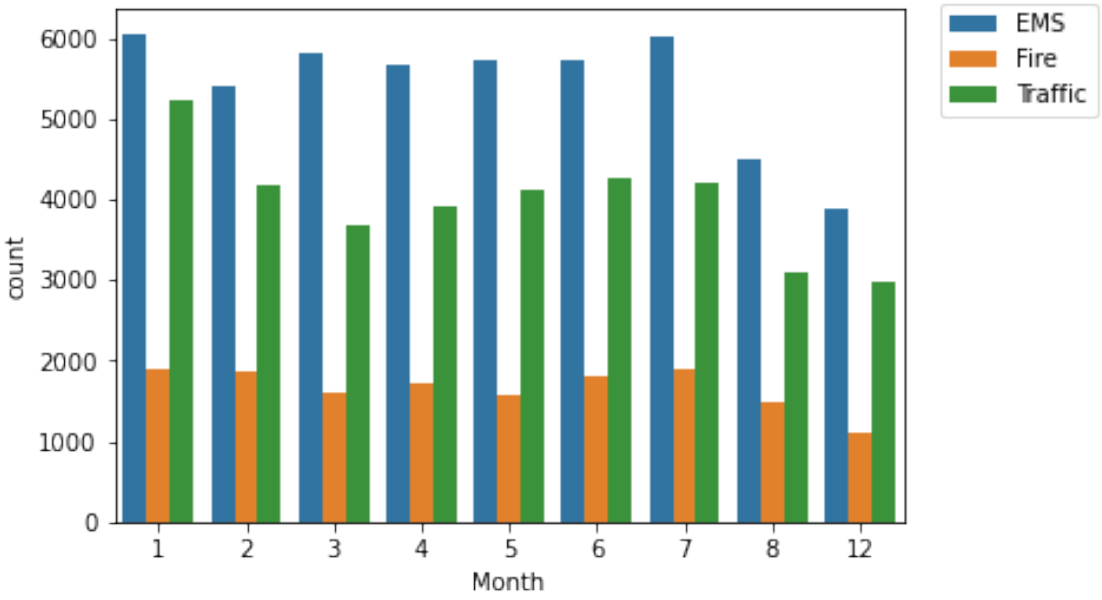
[24]:
```
<matplotlib.legend.Legend at 0x1d11ec39070>
```



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

[25]:
```
sns.countplot(x = "Month", data = df, hue = "Reason")
plt.legend(loc='center right',bbox_to_anchor=(1.26, .9))
```

[25]: <matplotlib.legend.Legend at 0x1d125820550>



Now do the same for Month:

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. **

[26]:
```
byMonth = df.groupby('Month').count()
```

[27]:
```
byMonth
```

[27]:

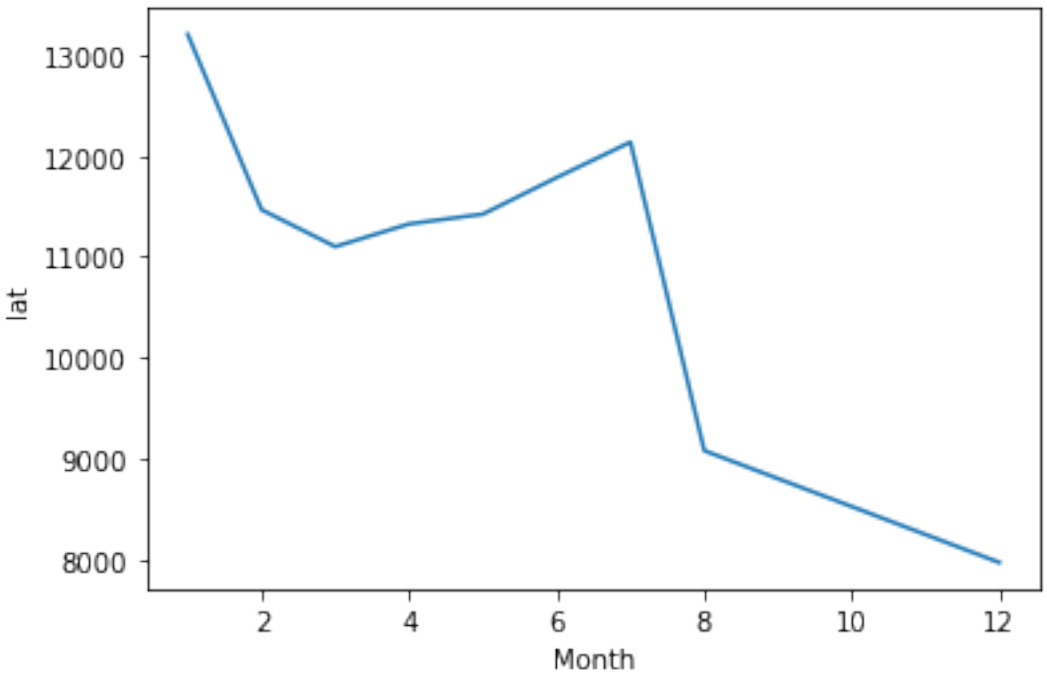
Month	lat	lng	desc	zip	title	timeStamp	twp	addr	e
1	13205	13205	13205	11527	13205	13205	13203	13096	13205
2	11467	11467	11467	9930	11467	11467	11465	11396	11467
3	11101	11101	11101	9755	11101	11101	11092	11059	11101
4	11326	11326	11326	9895	11326	11326	11323	11283	11326
5	11423	11423	11423	9946	11423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

Month	Reason	Year	Hour	Day	Day_of_Week
1	13205	13205	13205	13205	13205
2	11467	11467	11467	11467	11467
3	11101	11101	11101	11101	11101
4	11326	11326	11326	11326	11326
5	11423	11423	11423	11423	11423
6	11786	11786	11786	11786	11786
7	12137	12137	12137	12137	12137
8	9078	9078	9078	9078	9078
12	7969	7969	7969	7969	7969

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

[28]:
```
sns.lineplot(y = 'lat', x = 'Month',data = byMonth)
```

[28]:
```
<AxesSubplot:xlabel='Month', ylabel='lat'>
```

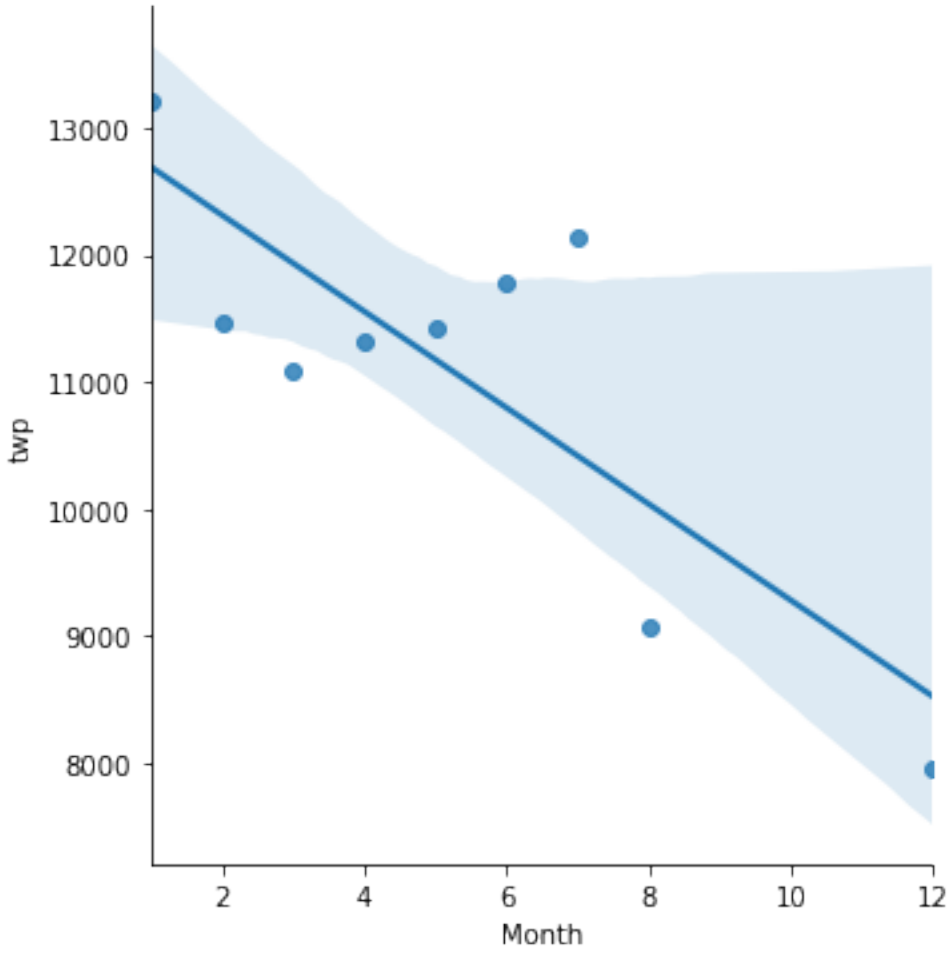


** Now see if you can use seaborn's lmplot() to create a linear fit on the number of calls per month. Keep in mind you may need to reset the index to a column. **

[29]:
```
byMonth.reset_index(inplace = True)
```

[30]:
```
sns.lmplot(x = 'Month', y = 'twp', data = byMonth)
```

[30]:
```
<seaborn.axisgrid.FacetGrid at 0x1d124b66a30>
```



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.

[31]:
```
df.timeStamp.iloc[0].date()
```

[31]:
```
datetime.date(2015, 12, 10)
```

[32]:
```
df["Date"] = df.timeStamp.apply(lambda x: x.date())
```

[33]:
```
df.head()
```

[33]:

	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…
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;…
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S…

```
       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
3  19401.0   EMS: CARDIAC EMERGENCY 2015-12-10 17:40:01         NORRISTOWN
4      NaN           EMS: DIZZINESS 2015-12-10 17:40:01   LOWER POTTSGROVE

                        addr  e Reason  Year  Hour  Month  Day Day_of_Week  \
0      REINDEER CT & DEAD END  1    EMS  2015    17     12    3         Thu
1  BRIAR PATH & WHITEMARSH LN  1    EMS  2015    17     12    3         Thu
2                    HAWS AVE  1   Fire  2015    17     12    3         Thu
3          AIRY ST & SWEDE ST  1    EMS  2015    17     12    3         Thu
4    CHERRYWOOD CT & DEAD END  1    EMS  2015    17     12    3         Thu

         Date
0  2015-12-10
1  2015-12-10
2  2015-12-10
3  2015-12-10
4  2015-12-10
```

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

[34]:
```
byDate = df.groupby('Date').count()
byDate.reset_index(inplace = True)
```

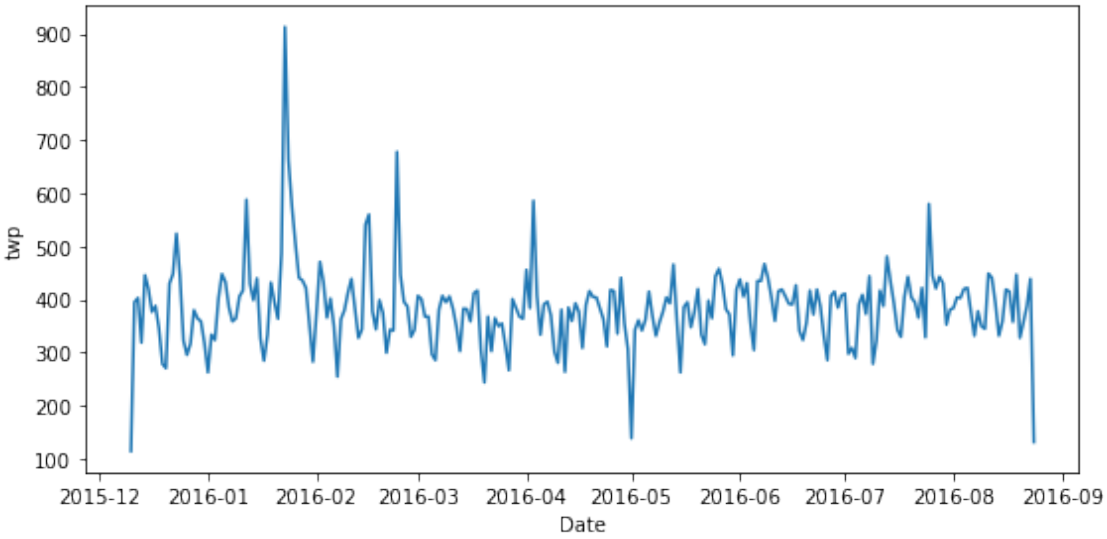
[35]:
```
byDate.head()
```

[35]:
```
         Date  lat  lng  desc  zip  title  timeStamp  twp  addr    e  Reason  \
0  2015-12-10  115  115   115  100    115        115  115   113  115     115
1  2015-12-11  396  396   396  333    396        396  395   391  396     396
2  2015-12-12  403  403   403  333    403        403  403   401  403     403
3  2015-12-13  319  319   319  280    319        319  319   317  319     319
4  2015-12-14  447  447   447  387    447        447  446   445  447     447

   Year  Hour  Month  Day  Day_of_Week
0   115   115    115  115          115
1   396   396    396  396          396
2   403   403    403  403          403
3   319   319    319  319          319
4   447   447    447  447          447
```

[37]:
```
fig = plt.figure(figsize = (8,4))
x1 = byDate.loc[ : ,['Date', 'twp']]
sns.lineplot(x = 'Date', y = 'twp', data = x1)
```

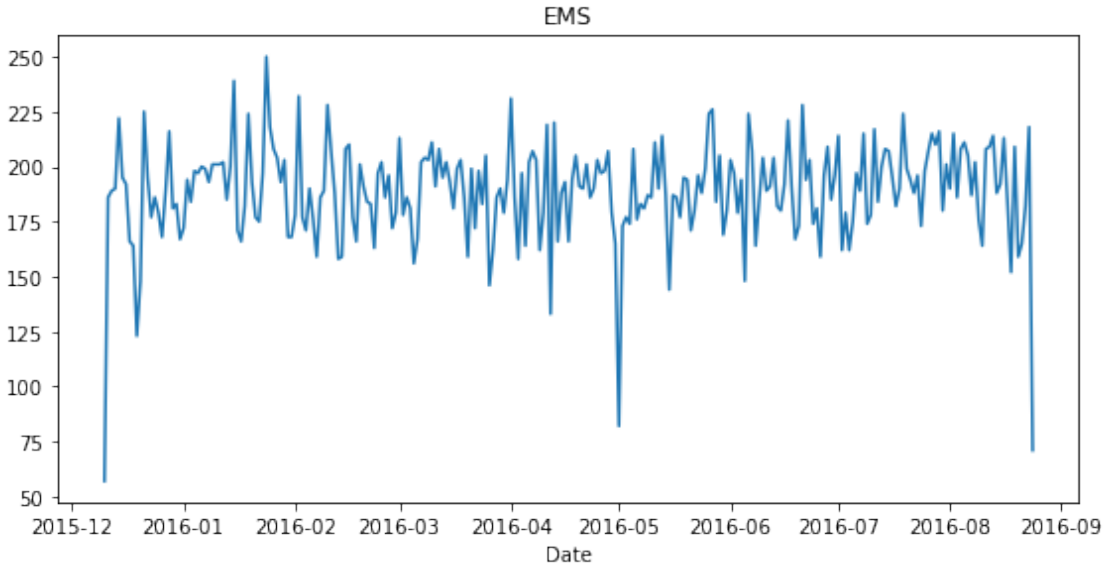
```
plt.tight_layout()
```



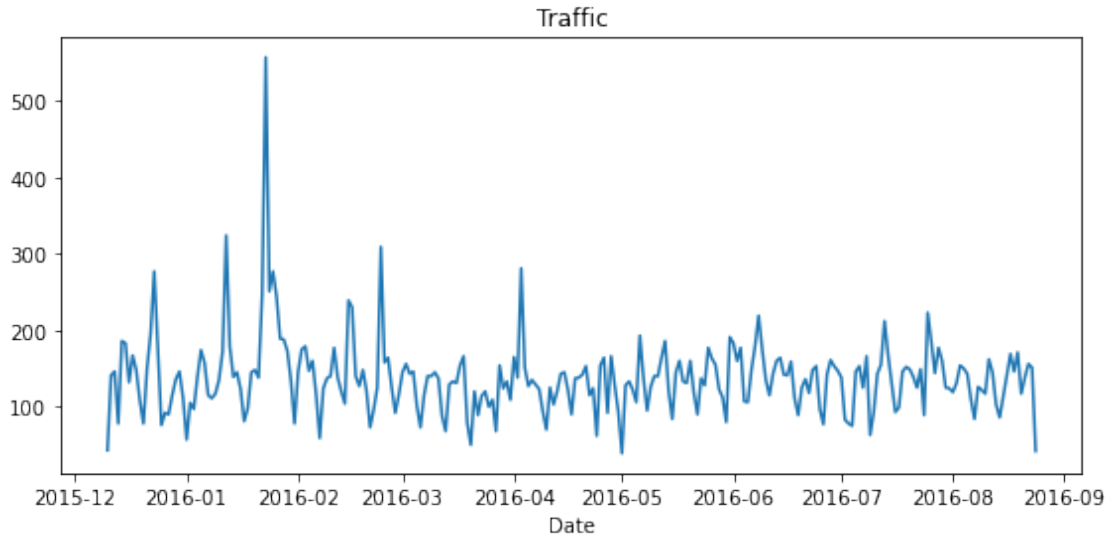
** Now recreate this plot but create 3 separate plots with each plot representing a Reason for the 911 call**

[38]:
```
fig = plt.figure(figsize = (8,4))
df.loc[df['Reason'] == 'EMS'].groupby('Date').count()['lat'].plot()
plt.tight_layout()
plt.title('EMS')
```

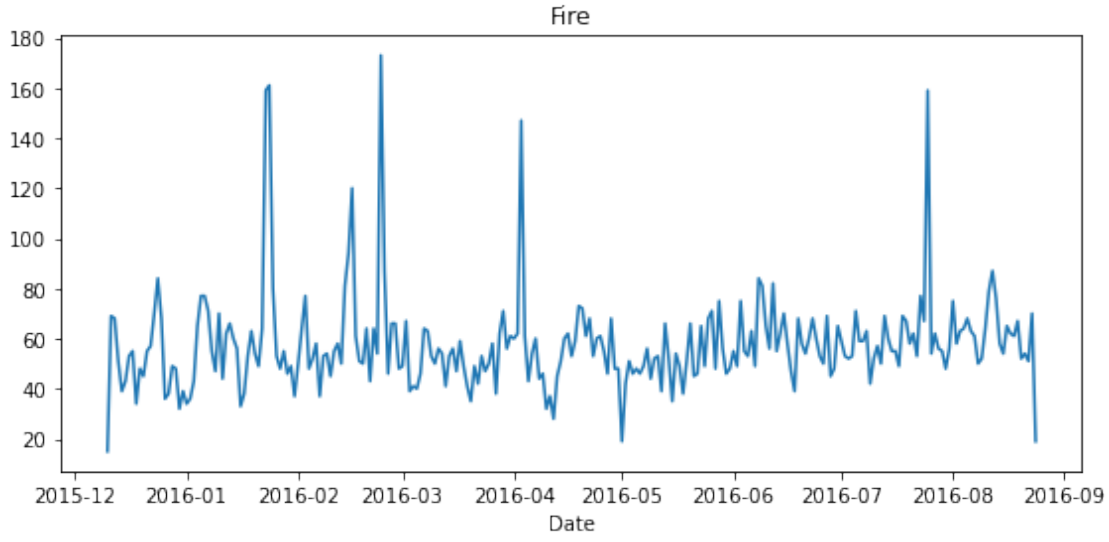
[38]:
```
Text(0.5, 1.0, 'EMS')
```



[39]:
```
fig = plt.figure(figsize = (8,4))
plt.title('Traffic')
df.loc[df['Reason'] == 'Traffic'].groupby('Date').count()['lat'].plot()
plt.tight_layout()
```



[40]:
```
fig = plt.figure(figsize = (8,4))
plt.title('Fire')
df.loc[df['Reason'] == 'Fire'].groupby('Date').count()['lat'].plot()
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!**

[41]:
```
dayhour = df.groupby(['Day_of_Week', 'Hour']).count()['Reason'].unstack()
```

[42]:
```
dayhour
```

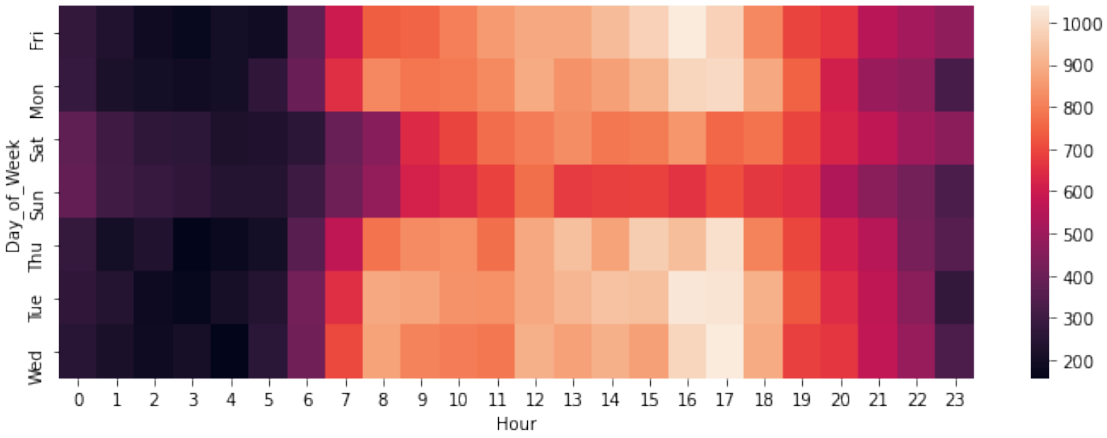
[42]:
```
Hour           0    1    2    3    4    5    6    7    8    9   …   14   15  \
Day_of_Week                                                      …
Fri          275  235  191  175  201  194  372  598  742  752   …  932  980
Mon          282  221  201  194  204  267  397  653  819  786   …  869  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
Tue          269  240  186  170  209  239  415  655  889  880   …  943  938
Wed          250  216  189  209  156  255  410  701  875  808   …  904  867

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
Tue          1026  1019  905  731  647  571  462  274
Wed           990  1037  894  686  668  575  490  335

[7 rows x 24 columns]
```

[43]:
```
fig = plt.figure(figsize = (12,4))
sns.heatmap(dayhour)
```

[43]:
```
<AxesSubplot:xlabel='Hour', ylabel='Day_of_Week'>
```



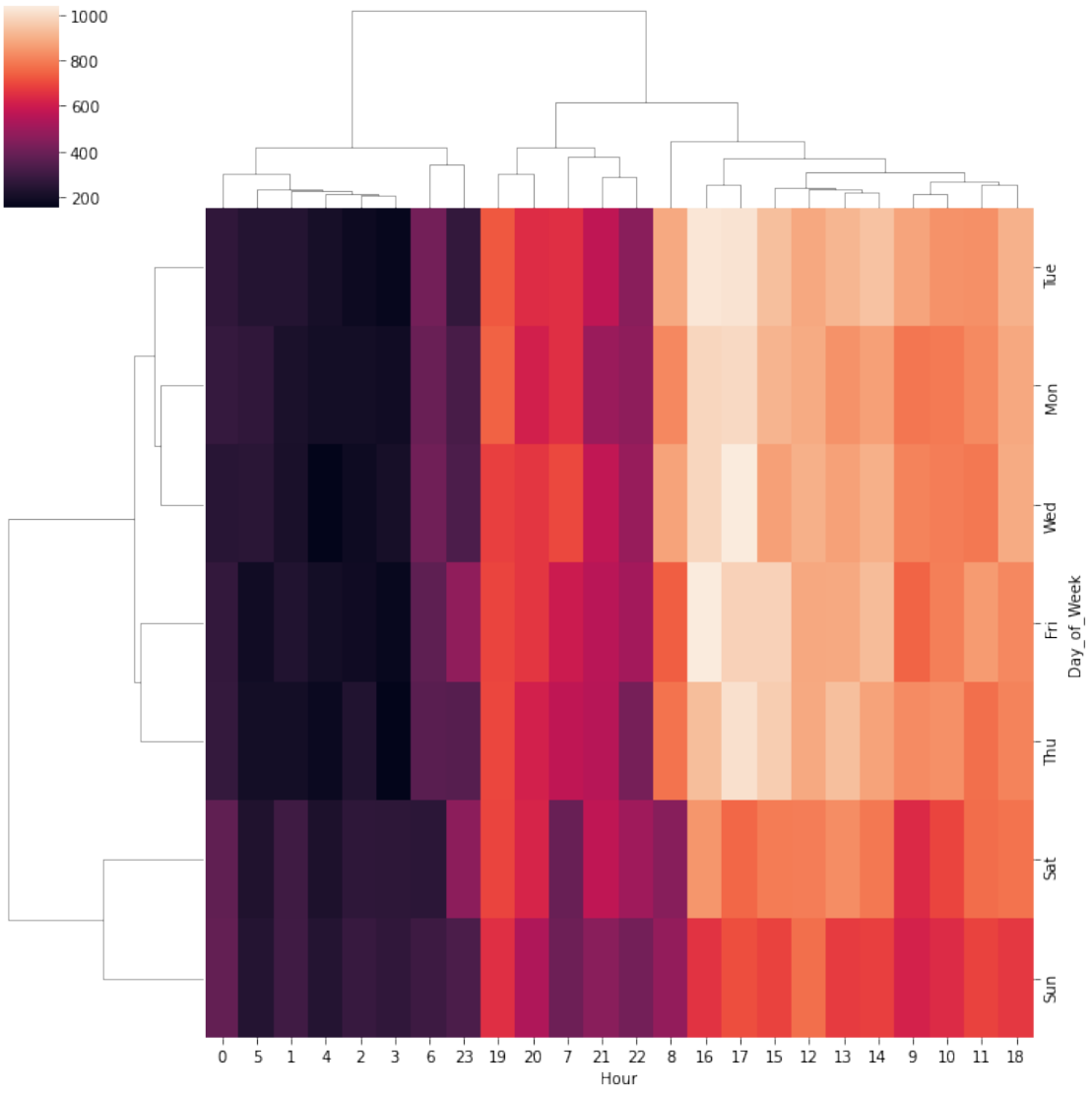
** Now create a HeatMap using this new DataFrame. **

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

[44]:
```
fig = plt.figure(figsize = (12,4))
sns.clustermap(dayhour)
```

[44]:
```
<seaborn.matrix.ClusterGrid at 0x1d124e79220>

<Figure size 864x288 with 0 Axes>
```



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

[45]:
```
wk_month = df.groupby(['Day_of_Week', 'Month']).count()['twp'].unstack()
```

[46]:
```
wk_month
```

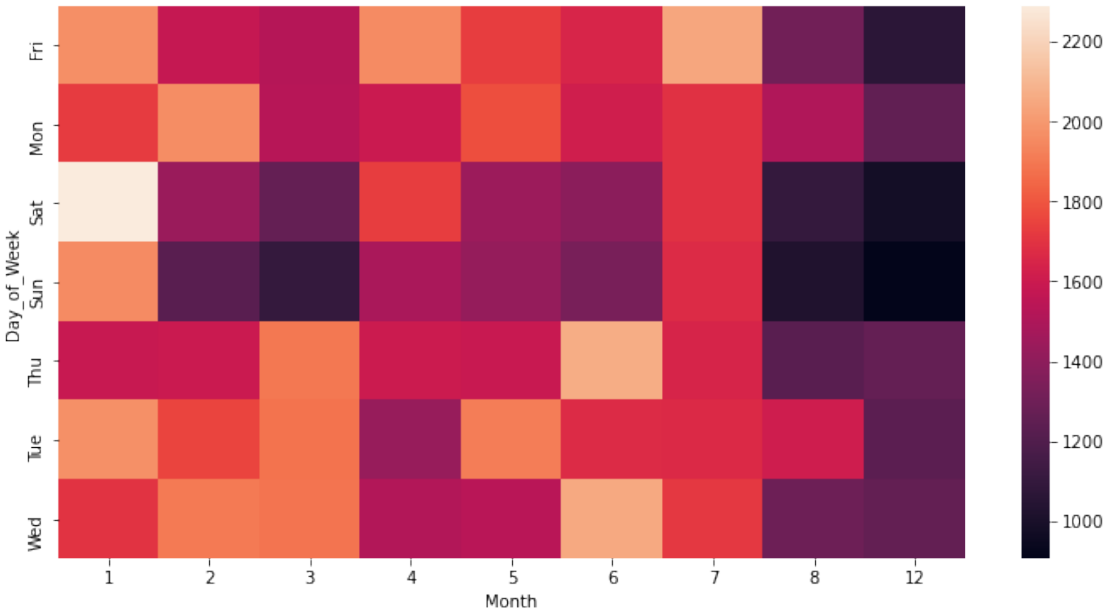
[46]:

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
Tue          1973  1753  1884  1430  1917  1673  1668  1612  1233
Wed          1699  1902  1888  1517  1538  2054  1715  1295  1260
```

[47]:
```
fig = plt.figure(figsize = (12,6))
sns.heatmap(wk_month)
```

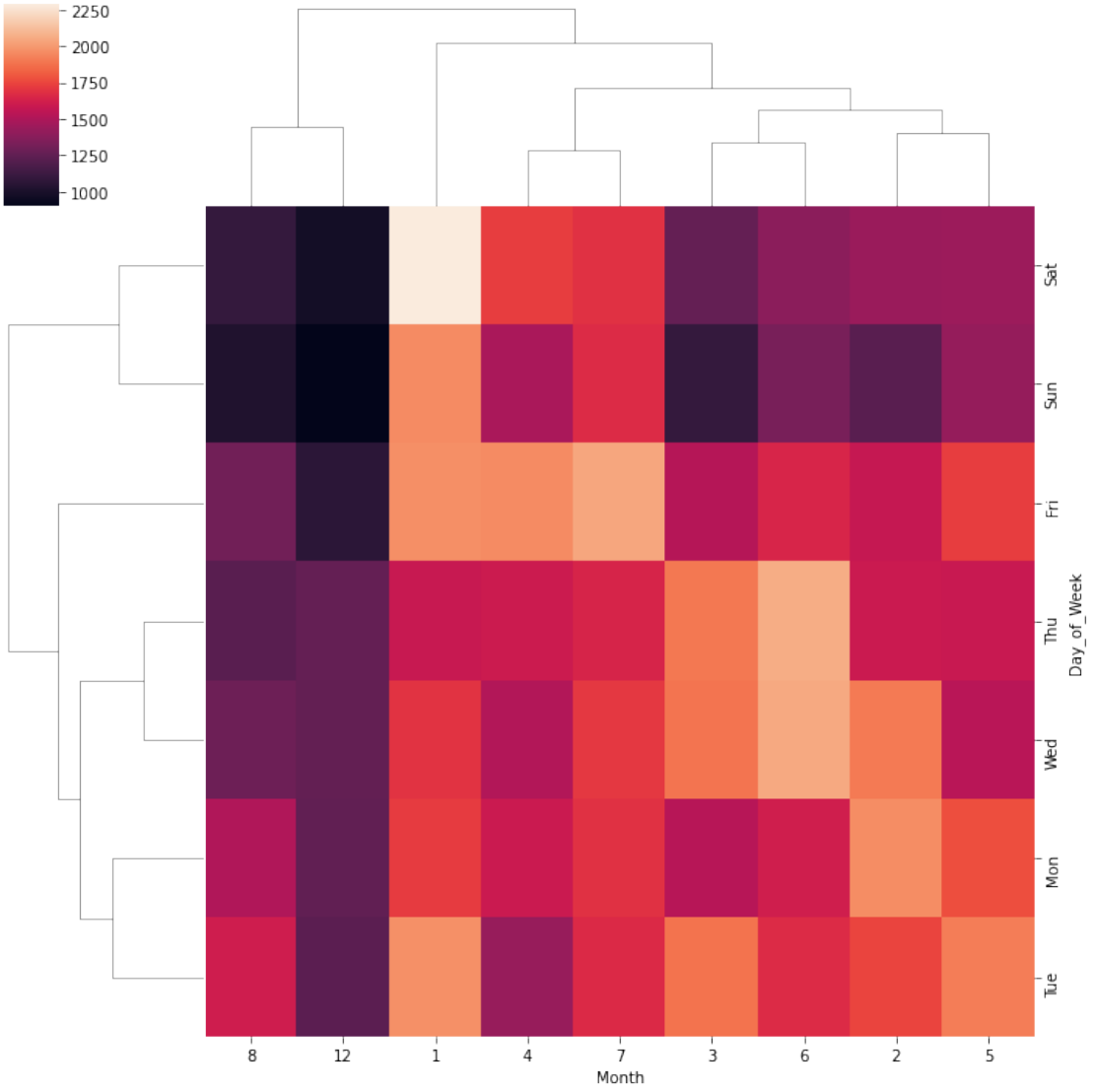
[47]: <AxesSubplot:xlabel='Month', ylabel='Day_of_Week'>



[48]:
```
fig = plt.figure(figsize = (12,4))
sns.clustermap(wk_month)
```

[48]: <seaborn.matrix.ClusterGrid at 0x1d127af1a00>

```
<Figure size 864x288 with 0 Axes>
```


2250
2000
1750
1500
1250
1000
Sat
Sun
Fri
Thu
Wed
Mon
Tue
Day_of_Week
8
12
1
4
7
3
6
2
5
Month

2 Thank you!