Open in Colab

EDA of 911 Calls

In [ ]:
```
!gdown --id 1hfT6f4ptZ4Dg8Uhy1uXytTxUf0Ho1uO4 --output 911.csv
```

```
Downloading...
From: https://drive.google.com/uc?id=1hfT6f4ptZ4Dg8Uhy1uXytTxUf0Ho1uO4
To: /content/911.csv
18.4MB [00:00, 69.6MB/s]
```

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.set_style('whitegrid')

plt.rcParams['figure.figsize'] = (6, 4)
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarnin
g: pandas.util.testing is deprecated. Use the functions in the public API at pandas.te
sting instead.
  import pandas.util.testing as tm
```

In [ ]:
```
#Reading the data
df = pd.read_csv('911.csv')
df.head()
```

Out[ ]:

	lat	lng	desc	zip	title	timeStamp	twp	ad
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 & DEAD EI
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 WHITEMAR
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 A
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
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 17:40:01	LOWER POTTSGROVE	CHERRYWO CT & DE EI

In [ ]:
```
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
```

Start with Basic Analysis

In [ ]:
```
# Let's check out the top 5 zipcodes for calls.

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

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

In [ ]:
```
# The top townships for the calls were as follows:

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

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

In [ ]:
```
# For 90k + entries, how many unique call titles did we have?

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

Out[ ]:
```
110
```

Data Wrangling for Feature Creation

We can extract some generalised features from the columns in our dataset for further analysis.

In the *title* column, there's a kind of 'subcategory' or 'reason for call' alloted to each entry (denoted by the text before the colon).

The timestamp column can be further segregated into Year, Month and Day of Week too.

Let's start with creating a 'Reason' feature for each call.

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

Out[ ]:

	lat	lng	desc	zip	title	timeStamp	twp
99487	40.132869	-75.333515	MARKLEY ST & W LOGAN ST; NORRISTOWN; 2016-08-2...	19401.0	Traffic: VEHICLE ACCIDENT -	2016-08-24 11:06:00	NORRISTOWN
99488	40.006974	-75.289080	LANCASTER AVE & RITTENHOUSE PL; LOWER MERION; ...	19003.0	Traffic: VEHICLE ACCIDENT -	2016-08-24 11:07:02	LOWER MERION
99489	40.115429	-75.334679	CHESTNUT ST & WALNUT ST; NORRISTOWN; Station ...	19401.0	EMS: FALL VICTIM	2016-08-24 11:12:00	NORRISTOWN
99490	40.186431	-75.192555	WELSH RD & WEBSTER LN; HORSHAM; Station 352; ...	19002.0	EMS: NAUSEA/VOMITING	2016-08-24 11:17:01	HORSHAM
99491	40.207055	-75.317952	MORRIS RD & S BROAD ST; UPPER GWYNEDD; 2016-08...	19446.0	Traffic: VEHICLE ACCIDENT -	2016-08-24 11:17:02	UPPER GWYNEDD

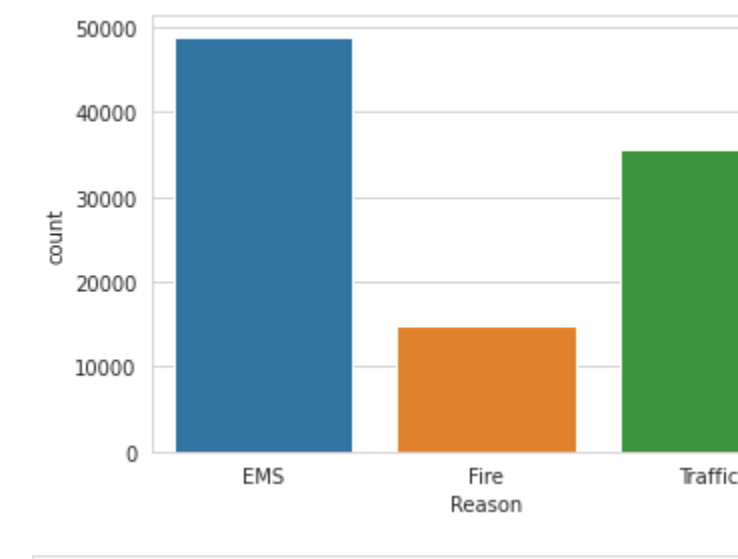
Now, let's find out the most common reason for 911 calls, according to our dataset.

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

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

In [ ]:
```
sns.countplot(df['Reason'])
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f79db564f60>
```



In [ ]:
```
# Let's deal with the time information we have.
# Checking the datatype of the timestamp column.
type(df['timeStamp'][0])
```

Out[ ]:
```
str
```

As the timestamps are still string types, it'll make our life easier if we convert it to a python DateTime object, so we can extract the year, month, and day information more intuitively.

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

In [ ]:
```
# For a single DateTime object, we can extract information as follows.

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

print('Hour:',time.hour)
print('Month:',time.month)
print('Day of Week:',time.dayofweek)
```

```
Hour: 17
Month: 12
Day of Week: 3
```

In [ ]:
```
# let's create new features for the above pieces of information.

df['Hour'] = df['timeStamp'].apply(lambda x: x.hour)
df['Month'] = df['timeStamp'].apply(lambda x: x.month)
df['Day of Week'] = df['timeStamp'].apply(lambda x: x.dayofweek)

df.head(3)
```

Out[ ]:

	lat	lng	desc	zip	title	timeStamp	twp	add
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 C & DEAD EN
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 WHITEMARS L
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 AV

The Day of the Week is an integer and it might not be instantly clear which number refers to which Day. We can map that information to a Mon-Sun string.

In [ ]:
```
dmap = {0:'Mon',1:'Tue',2:'Wed',3:'Thu',4:'Fri',5:'Sat',6:'Sun'}
df['Day of Week'] = df['Day of Week'].map(dmap)

df.tail(3)
```

Out[ ]:

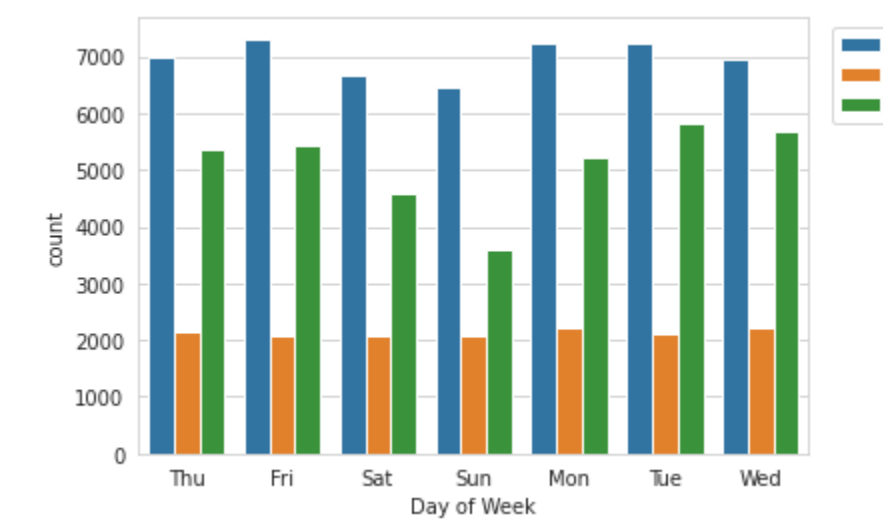
	lat	lng	desc	zip	title	timeStamp	twp
99489	40.115429	-75.334679	CHESTNUT ST & WALNUT ST; NORRISTOWN; Station ...	19401.0	EMS: FALL VICTIM	2016-08-24 11:12:00	NORRISTOWN
99490	40.186431	-75.192555	WELSH RD & WEBSTER LN; HORSHAM; Station 352; ...	19002.0	EMS: NAUSEA/VOMITING	2016-08-24 11:17:01	HORSHAM
99491	40.207055	-75.317952	MORRIS RD & S BROAD ST; UPPER GWYNEDD; 2016-08...	19446.0	Traffic: VEHICLE ACCIDENT -	2016-08-24 11:17:02	UPPER GWYNEDD

Let's combine the newly created features, to check out the most common call reasons based on the day of the week.

In [ ]:
```
sns.countplot(df['Day of Week'],hue=df['Reason'])

plt.legend(bbox_to_anchor=(1.25,1))
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7f79db569b70>
```

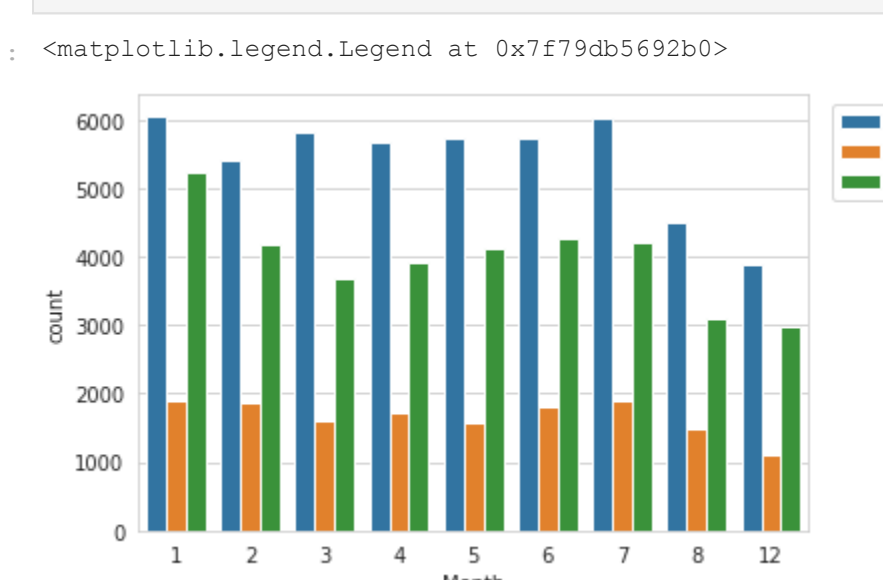


It makes sense for the number of traffic related 911 calls to be the lowest during the weekends, what's also iteresting is that Emergency Service related calls are also low during the weekend.

In [ ]:
```
sns.countplot(df['Month'],hue=df['Reason'])

plt.legend(bbox_to_anchor=(1.25,1))
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7f79db5692b0>
```

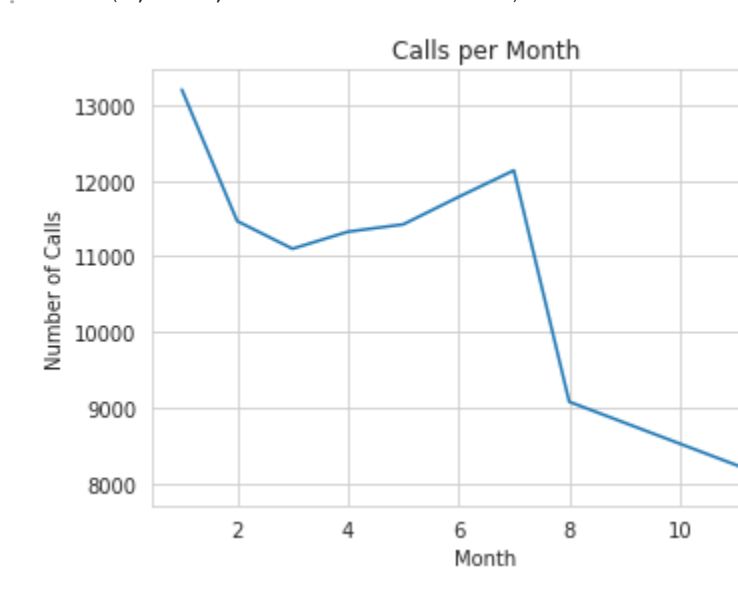


Now, let's check out the relationship between the number of calls and the month.

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

byMonth['e'].plot.line(y='e')
plt.title('Calls per Month')
plt.ylabel('Number of Calls')
```

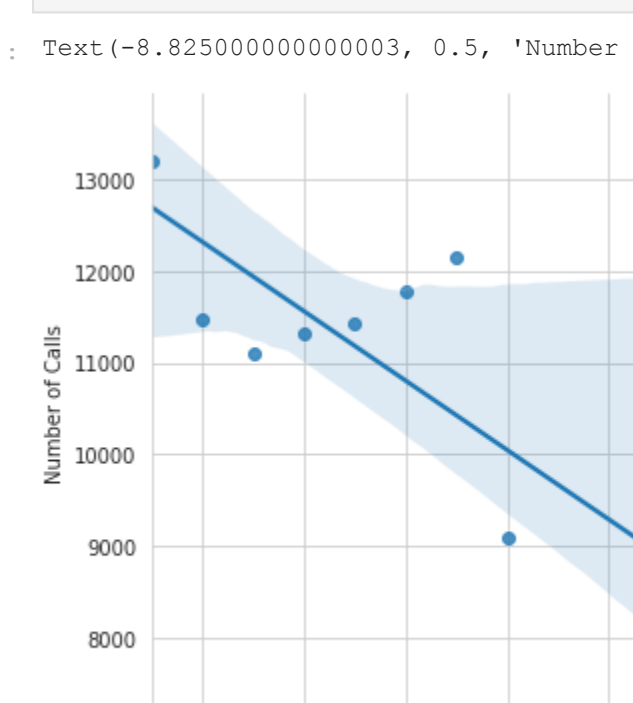
Out[ ]:
```
Text(0, 0.5, 'Number of Calls')
```



Using seaborn, let's fit the number of calls to a month and see if there's any concrete correlation between the two.

In [ ]:
```
byMonth.reset_index(inplace=True)
sns.lmplot(x='Month',y='e',data=byMonth)
plt.ylabel('Number of Calls')
```

Out[ ]:
```
Text(-8.825000000000003, 0.5, 'Number of Calls')
```



So, it does seem that there are fewer emergency calls during the holiday seasons.

Let's extract the date from the timestamp, and see behavior in a little more detail.

In [ ]:
```
df['Date']=df['timeStamp'].apply(lambda x: x.date())
df.head(2)
```

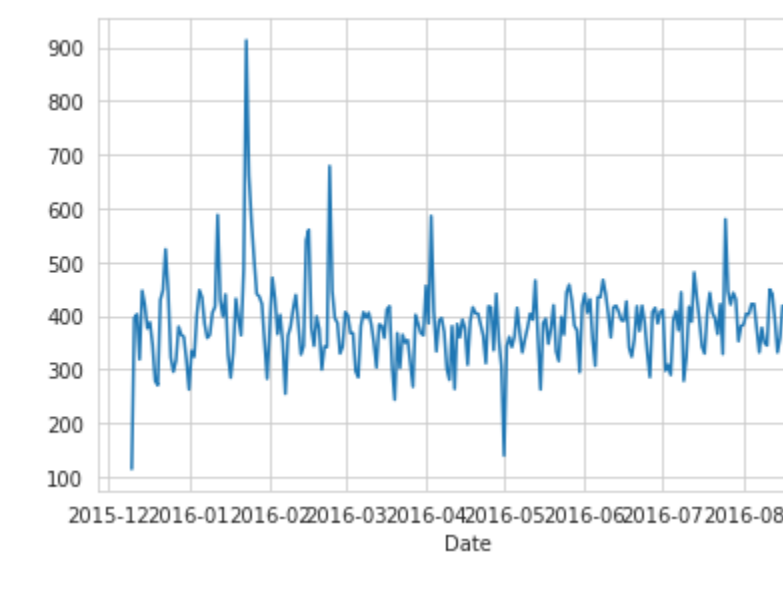
Out[ ]:

	lat	lng	desc	zip	title	timeStamp	twp	addr
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	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

In [ ]:
```
# Grouping and plotting the data:

df.groupby('Date').count()['e'].plot.line(y='e')

plt.legend().remove()
plt.tight_layout()
```



In [ ]:
```
# We can also check out the same plot for each reason separately.

df.groupby('Reason'=='Traffic','Date').count().plot.line(y='e')
plt.title('Traffic')
plt.legend().remove()
plt.tight_layout()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-22-521dedc5cb10> in <module>()
      1
----> 2 df.groupby('Reason'=='Traffic','Date').count().plot.line(y='e')
      3 plt.title('Traffic')
      4 plt.legend().remove()
      5 plt.tight_layout()

/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py in groupby(self, by, axis,
 level, as_index, sort, group_keys, squeeze, observed)
   5797         if level is None and by is None:
   5798             raise TypeError("You have to supply one of 'by' and 'level'")
-> 5799         axis = self._get_axis_number(axis)
   5800
   5801         return groupby_generic.DataFrameGroupBy(

/usr/local/lib/python3.6/dist-packages/pandas/core/generic.py in _get_axis_number(cls,
 axis)
    405             except KeyError:
    406                 pass
--> 407         raise ValueError(f"No axis named {axis} for object type {cls}")
    408
    409     @classmethod

ValueError: No axis named Date for object type <class 'pandas.core.frame.DataFrame'>
```

In [ ]:
```
df.groupby([['Reason']=='Fire'],'Date').count().plot.line(y='e')
plt.title('Fire')
plt.legend().remove()
plt.tight_layout()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-23-ffbcafdf87bb> in <module>()
----> 1 df.groupby([['Reason']=='Fire'],'Date').count().plot.line(y='e')
      2 plt.title('Fire')
      3 plt.legend().remove()
      4 plt.tight_layout()

/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py in groupby(self, by, axis,
 level, as_index, sort, group_keys, squeeze, observed)
   5797         if level is None and by is None:
   5798             raise TypeError("You have to supply one of 'by' and 'level'")
-> 5799         axis = self._get_axis_number(axis)
   5800
   5801         return groupby_generic.DataFrameGroupBy(

/usr/local/lib/python3.6/dist-packages/pandas/core/generic.py in _get_axis_number(cls,
 axis)
    405             except KeyError:
    406                 pass
--> 407         raise ValueError(f"No axis named {axis} for object type {cls}")
    408
    409     @classmethod

ValueError: No axis named Date for object type <class 'pandas.core.frame.DataFrame'>
```

In [ ]:
```
pd.groupby(df[df['Reason']=='EMS'],'Date').count().plot.line(y='e')
plt.title('EMS')
plt.legend().remove()
plt.tight_layout()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-24-150401d0fada> in <module>()
----> 1 pd.groupby(df[df['Reason']=='EMS'],'Date').count().plot.line(y='e')
      2 plt.title('EMS')
      3 plt.legend().remove()
      4 plt.tight_layout()

AttributeError: module 'pandas' has no attribute 'groupby'
```

Let's create a heatmap for the counts of calls on each hour, during a given day of the week.

In [ ]:
```
day_hour = df.pivot_table(values='lat',index='Day of Week',columns='Hour',aggfunc='cou

day_hour
```

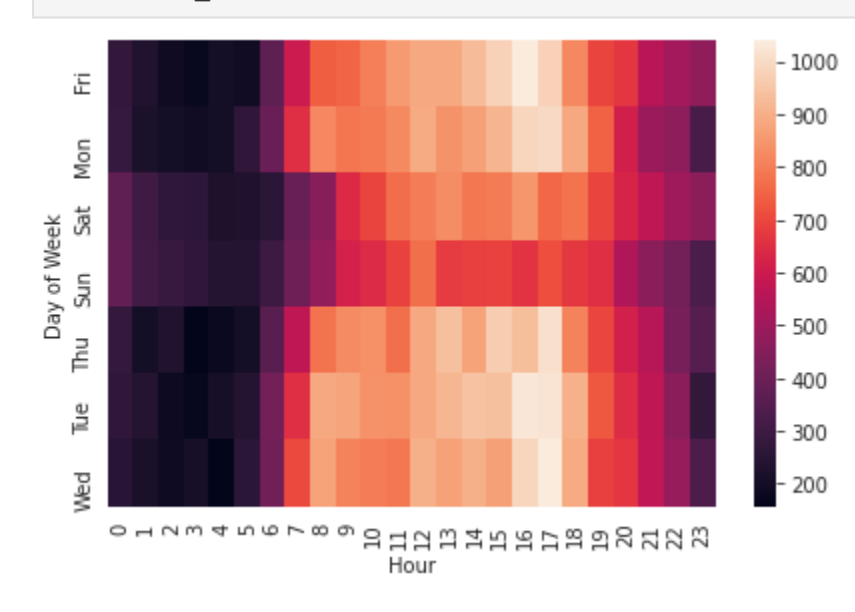
Out[ ]:

Hour	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Day of Week																		
Fri	275	235	191	175	201	194	372	598	742	752	803	859	885	890	932	980	1039	980
Mon	282	221	201	194	204	267	397	653	819	786	793	822	893	842	869	913	989	997
Sat	375	301	263	260	224	231	257	391	459	640	697	769	801	831	789	796	848	757
Sun	383	306	286	268	242	240	300	402	483	620	643	693	771	679	684	691	663	714
Thu	278	202	233	159	182	203	362	570	777	828	837	773	889	936	876	969	935	1013
Tue	269	240	186	170	209	239	415	655	889	880	840	838	887	917	943	938	1026	1019
Wed	250	216	189	209	156	255	410	701	875	808	800	789	903	872	904	867	990	1037

Now create a HeatMap using this new DataFrame.

In [ ]:
```
sns.heatmap(day_hour)

plt.tight_layout()
```



We see that most calls take place around the end of office hours on weekdays. We can create a clustermap to pair up similar Hours and Days.

In [ ]:
```
sns.clustermap(day_hour)
```

Out[ ]:
```
<seaborn.matrix.ClusterGrid at 0x7f79d8372ba8>
```