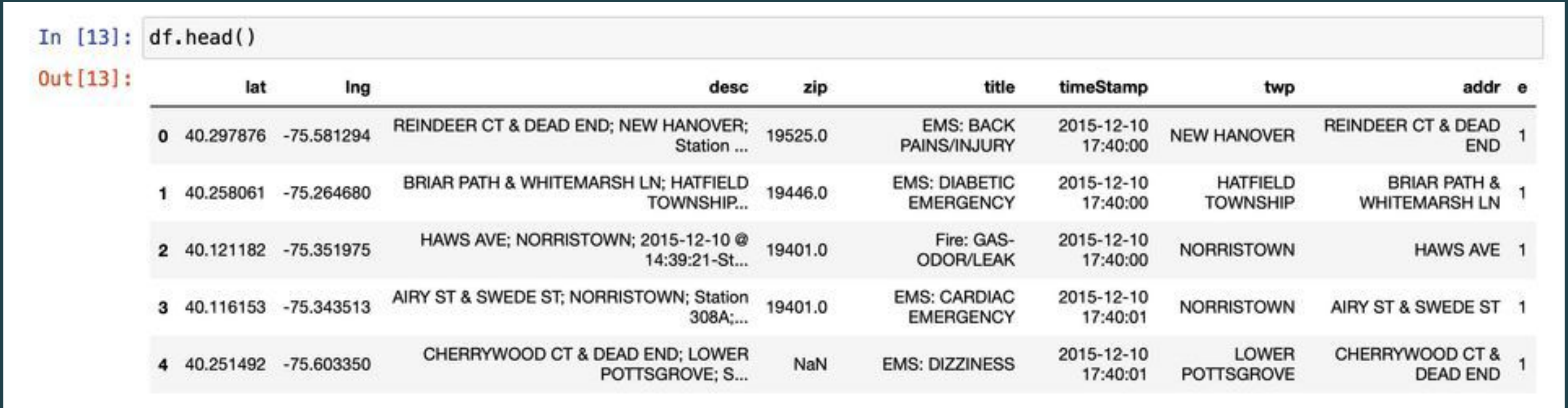
# 911 Calls Data Project

By: Keldon Morgan, Brian Paul, Philip Bernard, and Linda Pompilus

# Initial Data Set

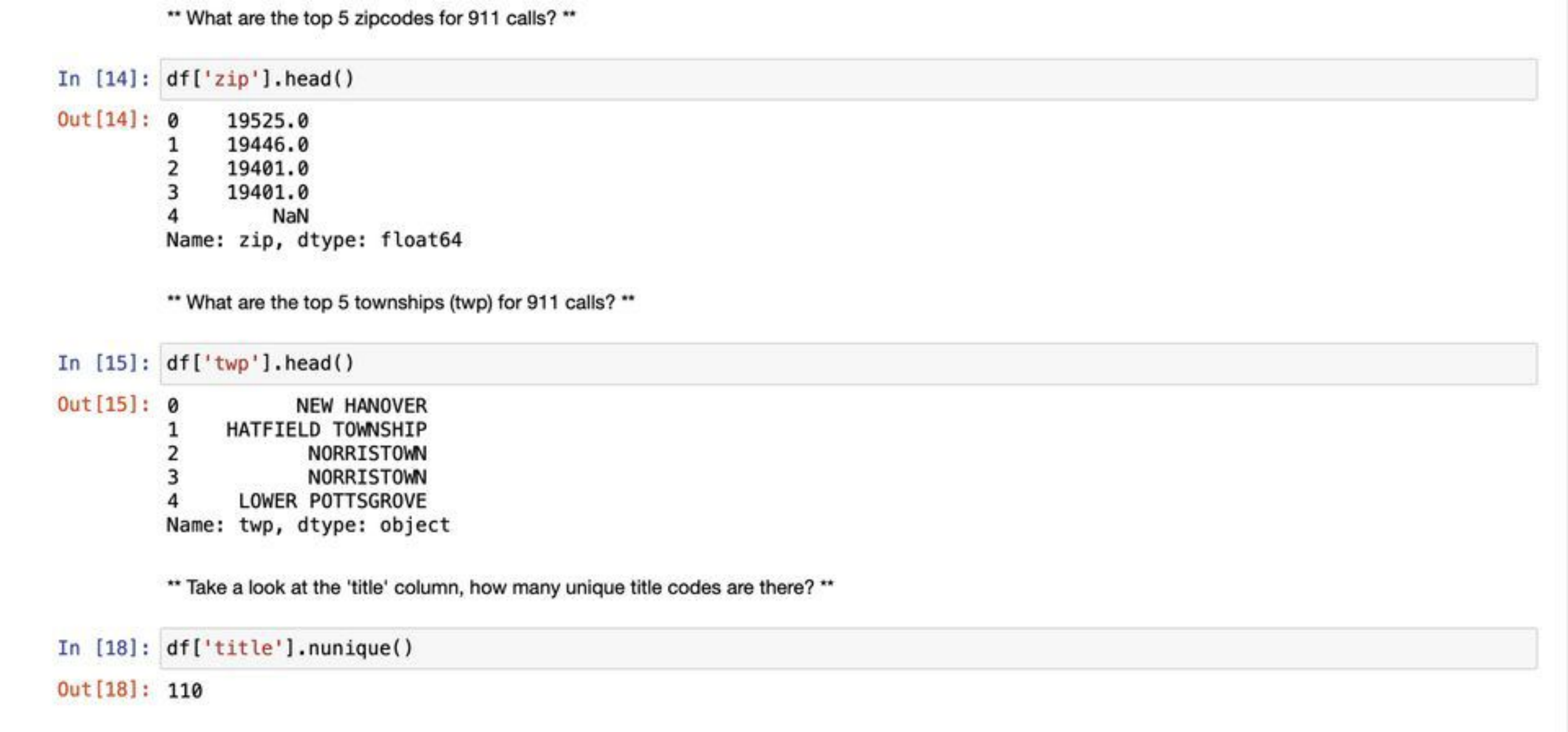


```
In [13]: df.head()
```

Out[13]:

| | 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 |

# .head()/.nunique() methods



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

```
In [14]: df['zip'].head()

Out[14]: 0    19525.0
         1    19446.0
         2    19401.0
         3    19401.0
         4        NaN
         Name: zip, dtype: float64
```

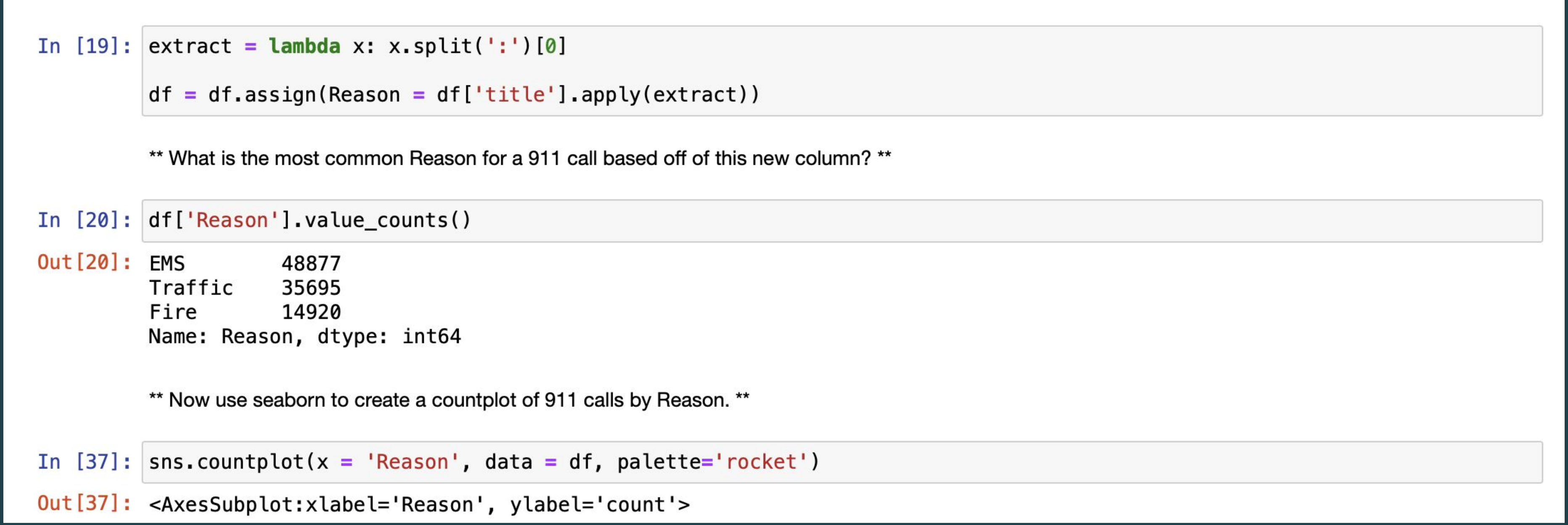
** What are the top 5 townships (twp) for 911 calls? **

```
In [15]: df['twp'].head()

Out[15]: 0          NEW HANOVER
         1    HATFIELD TOWNSHIP
         2           NORRISTOWN
         3           NORRISTOWN
         4     LOWER POTTSGROVE
         Name: twp, dtype: object
```

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

```
In [18]: df['title'].nunique()

Out[18]: 110
```

# Lambda Function/Seaborn Library



```
In [19]: extract = lambda x: x.split(':')[0]

         df = df.assign(Reason = df['title'].apply(extract))
```

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

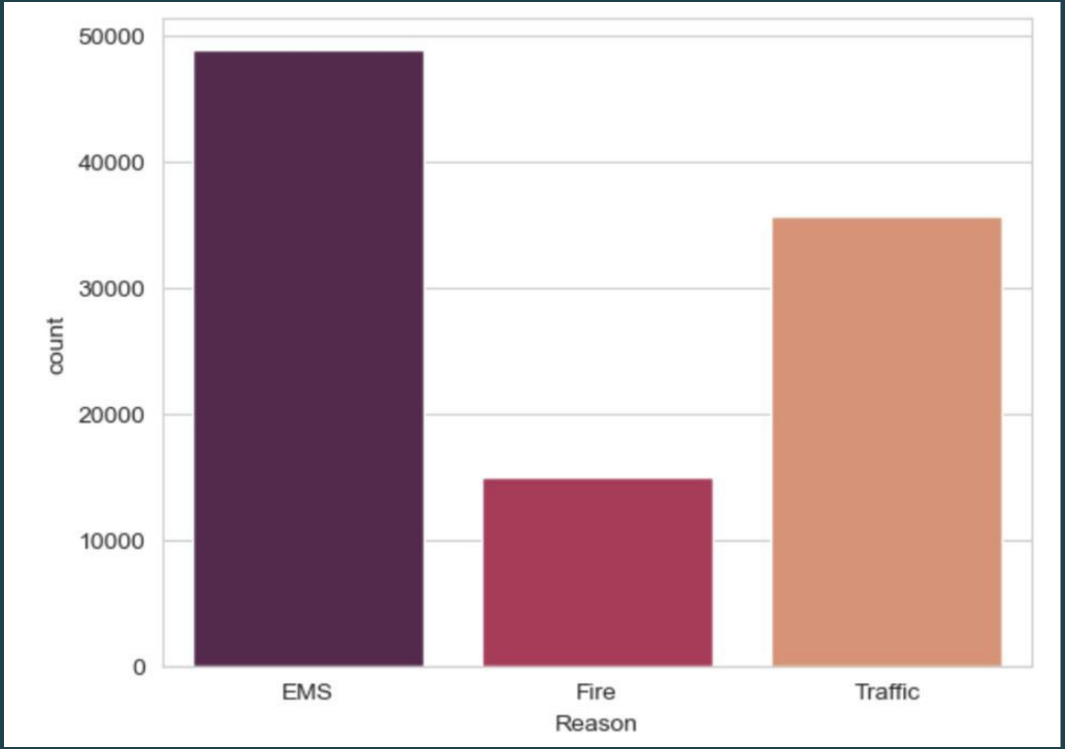
```
In [20]: df['Reason'].value_counts()

Out[20]: EMS        48877
         Traffic    35695
         Fire       14920
         Name: Reason, dtype: int64
```

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

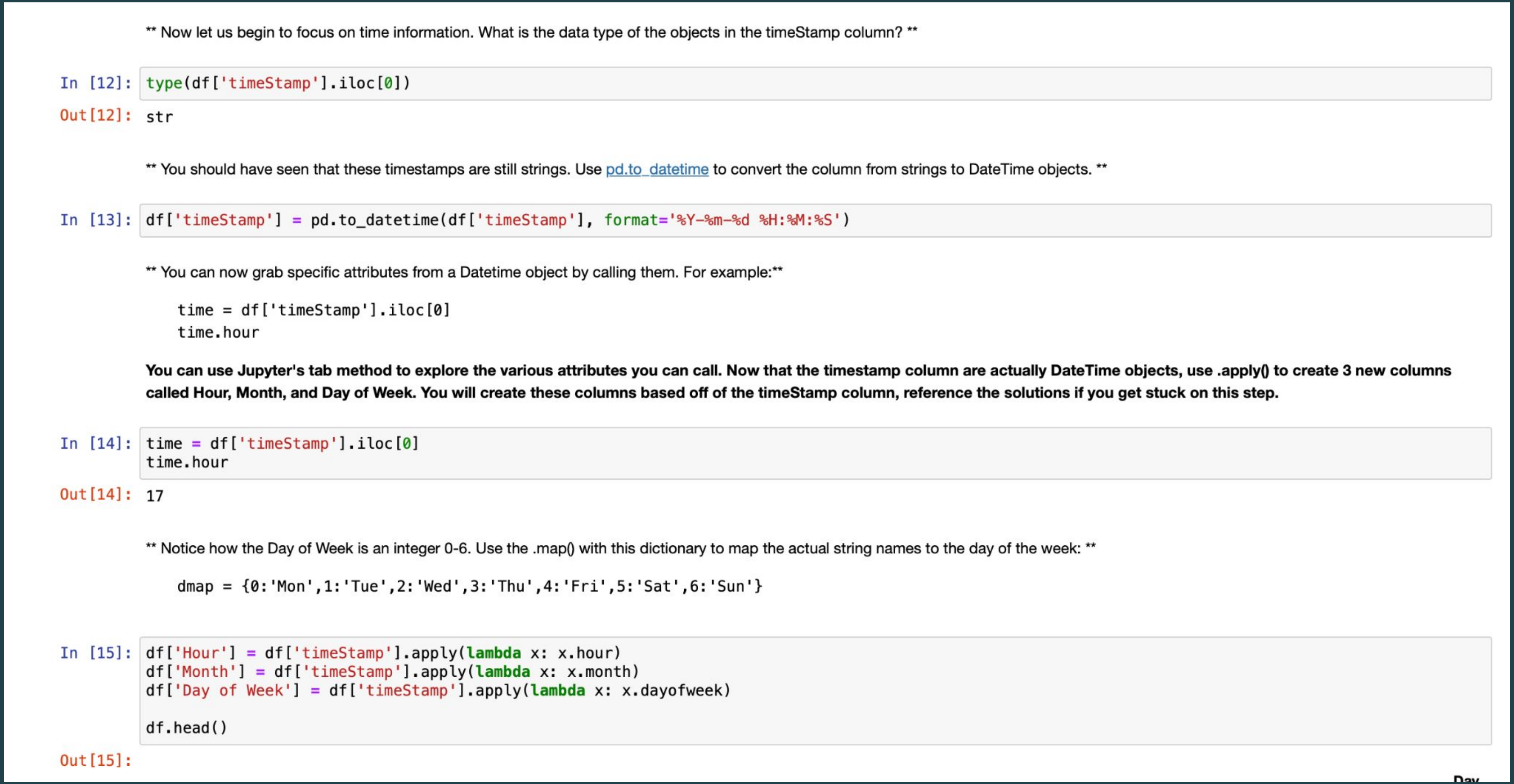
```
In [37]: sns.countplot(x = 'Reason', data = df, palette='rocket')

Out[37]: <AxesSubplot:xlabel='Reason', ylabel='count'>
```

# 911 Phone Calls By Count/Reason

- ■ Ems
- ■ Fire
- ■ Traffic



# TimeStamp Column/Creating New Columns



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

```
In [12]: type(df['timeStamp'].iloc[0])

Out[12]: 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 [13]: df['timeStamp'] = pd.to_datetime(df['timeStamp'], format='%Y-%m-%d %H:%M:%S')
```

** 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 [14]: time = df['timeStamp'].iloc[0]
         time.hour

Out[14]: 17
```

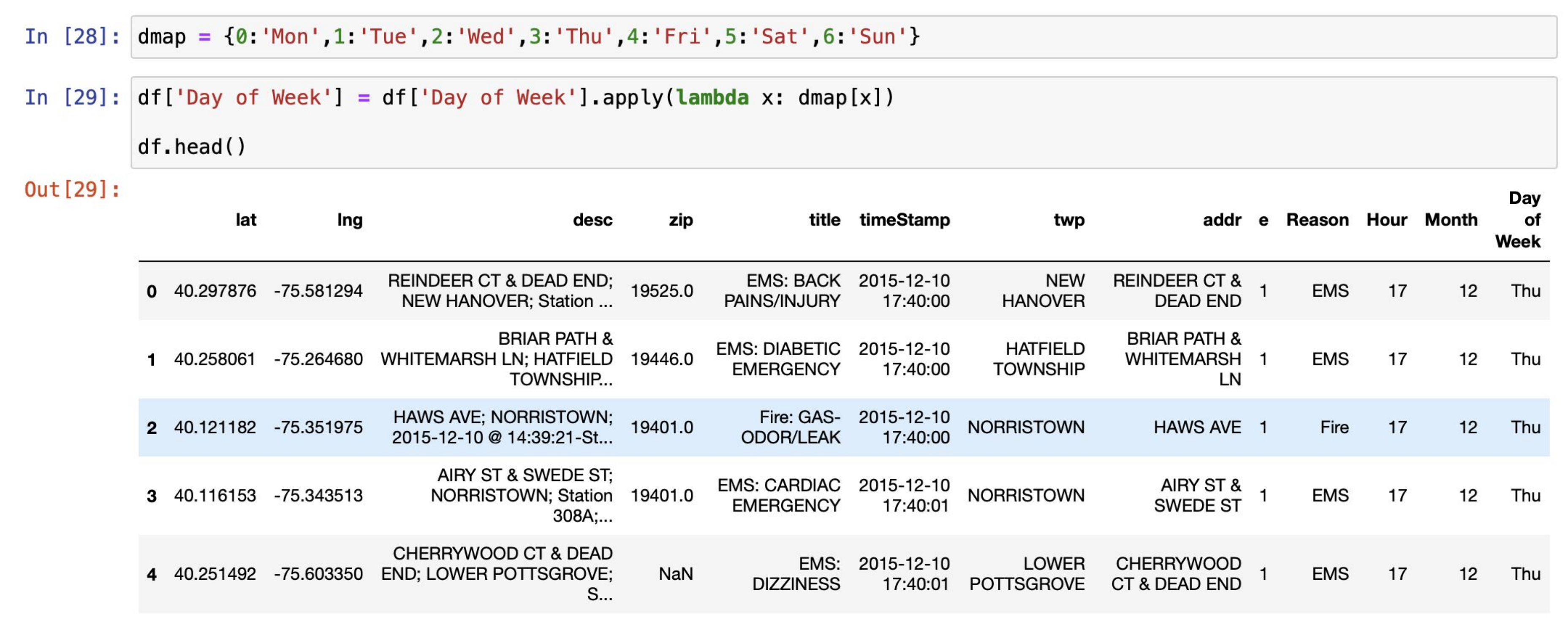
** Notice how the Day of Week is an integer 0-6. Use the .map() with this dictionary to map the actual string names to the day of the week: **

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

```
In [15]: 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()

Out[15]:
```

Day

# Dmap library w/ Lambda Function



In [28]:

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

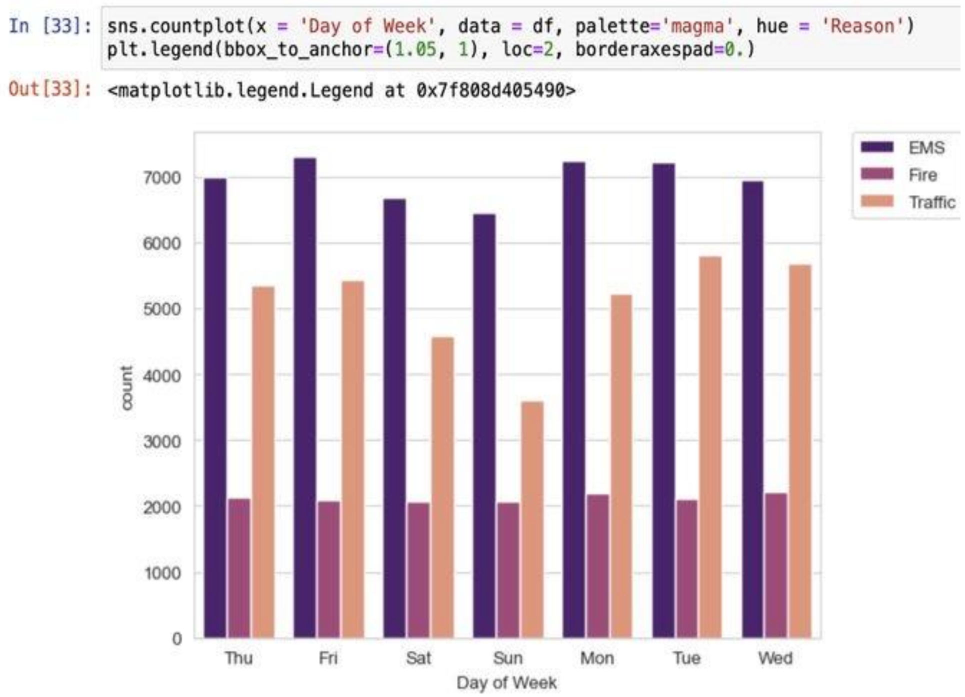
In [29]:

```python
df['Day of Week'] = df['Day of Week'].apply(lambda x: dmap[x])

df.head()
```

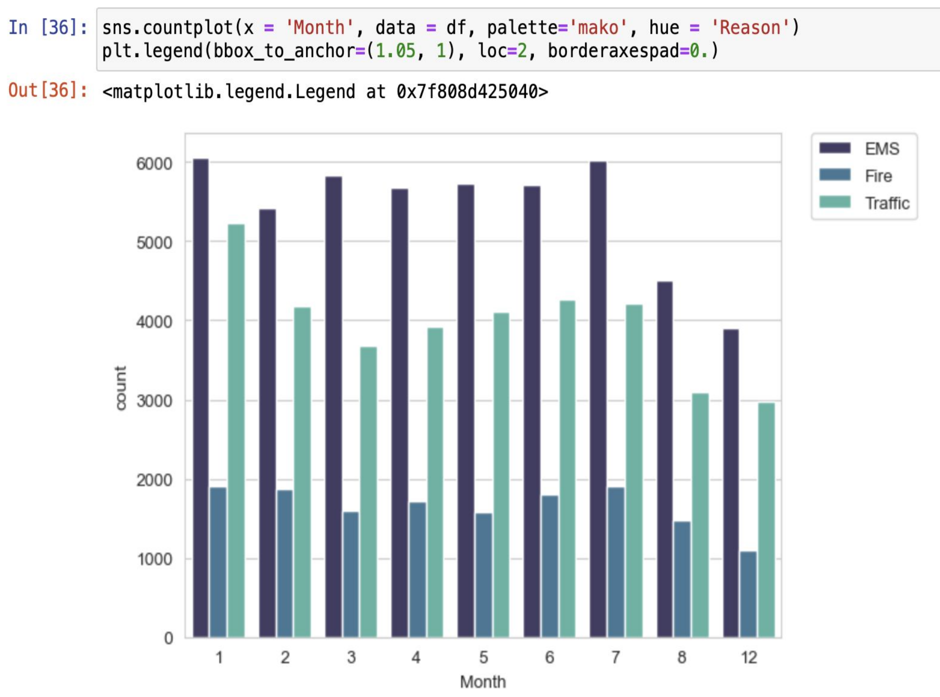
Out[29]:

| | lat | lng | desc | zip | title | timeStamp | twp | addr | e | Reason | Hour | Month | Day of Week |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | Thu |
| **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 | Thu |
| **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 | Thu |
| **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 | Thu |
| **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 | Thu |

# Comparison

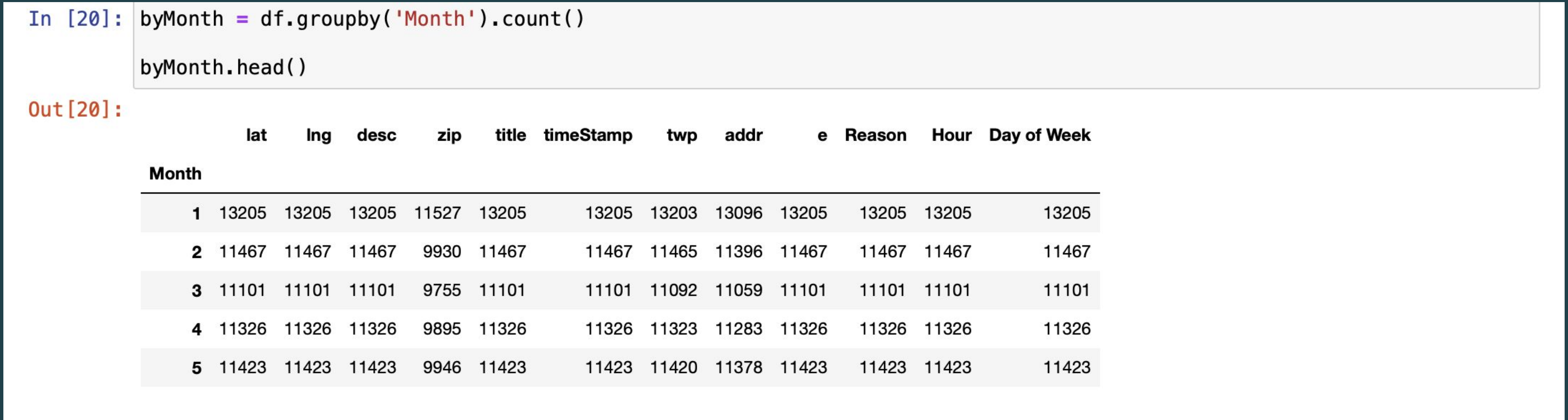


```
In [33]: sns.countplot(x = 'Day of Week', data = df, palette='magma', hue = 'Reason')
         plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

Out[33]: <matplotlib.legend.Legend at 0x7f808d405490>
```



```
In [36]: sns.countplot(x = 'Month', data = df, palette='mako', hue = 'Reason')
         plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)

Out[36]: <matplotlib.legend.Legend at 0x7f808d425040>
```

# .Count() Method

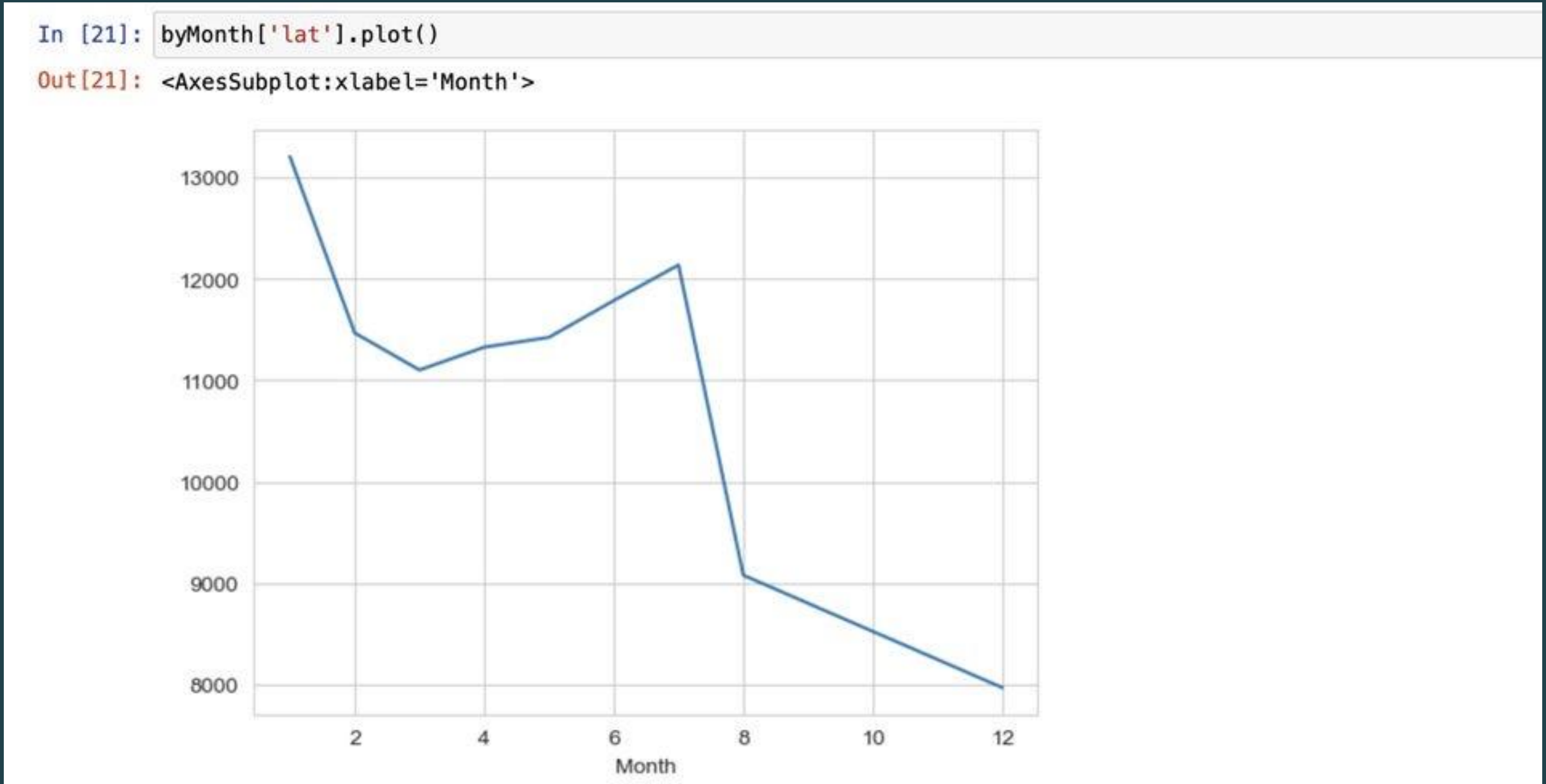


```
In [20]: byMonth = df.groupby('Month').count()

         byMonth.head()
```

Out[20]:

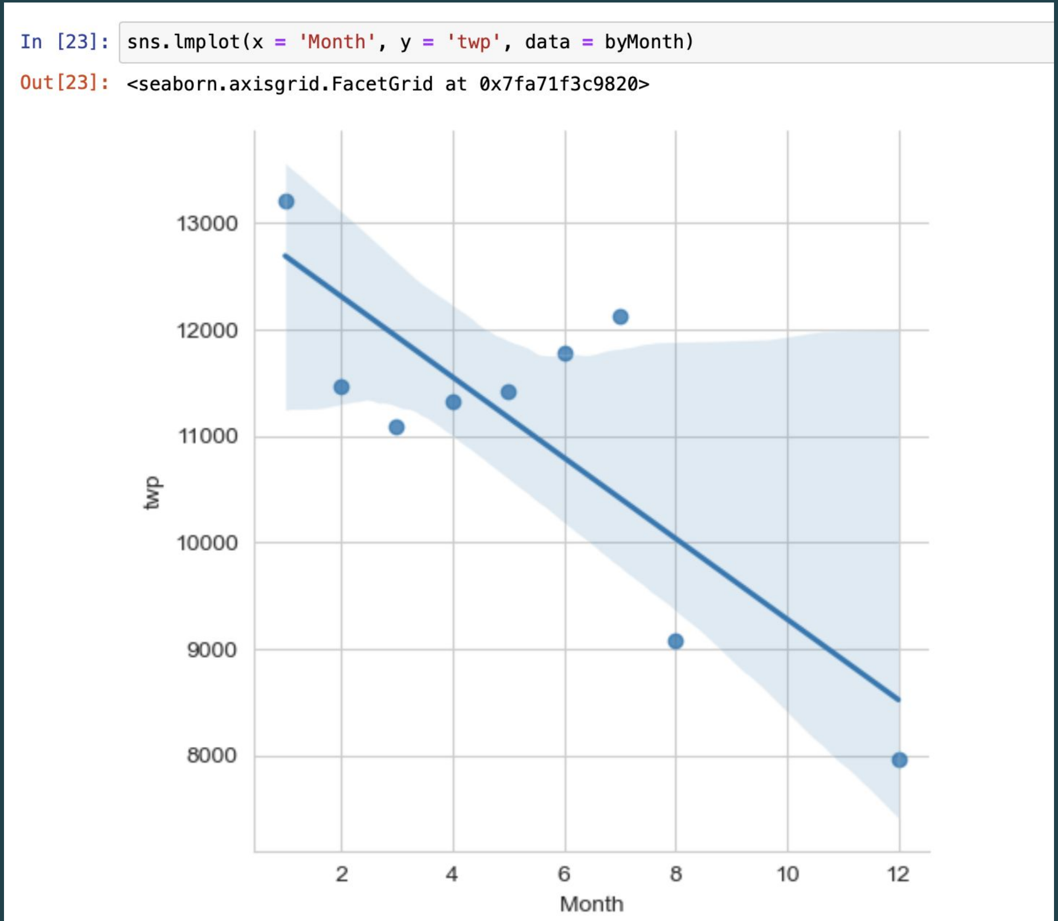
| | lat | lng | desc | zip | title | timeStamp | twp | addr | e | Reason | Hour | Day of Week |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Month** | | | | | | | | | | | | |
| **1** | 13205 | 13205 | 13205 | 11527 | 13205 | 13205 | 13203 | 13096 | 13205 | 13205 | 13205 | 13205 |
| **2** | 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 |

# Count Plot Map by Month



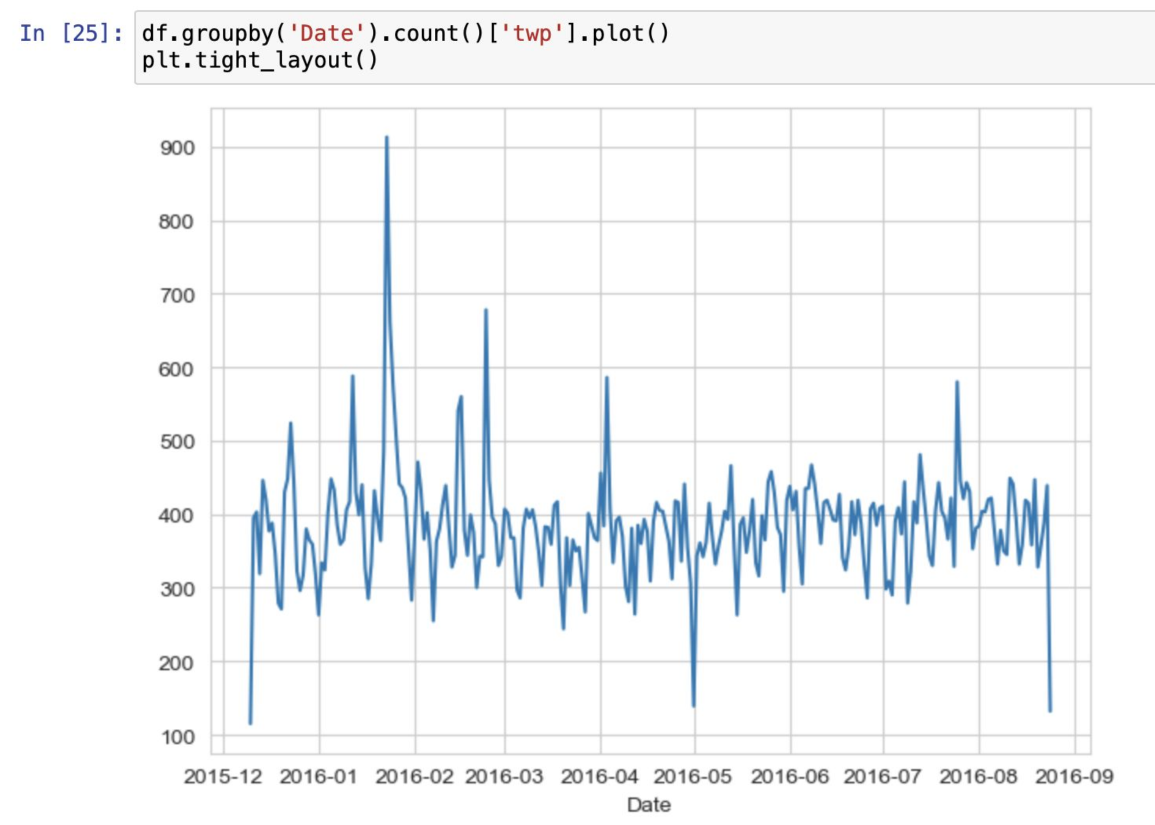
```
In [21]: byMonth['lat'].plot()

Out[21]: <AxesSubplot:xlabel='Month'>
```

# Linear fit using lmplot() from Seaborn



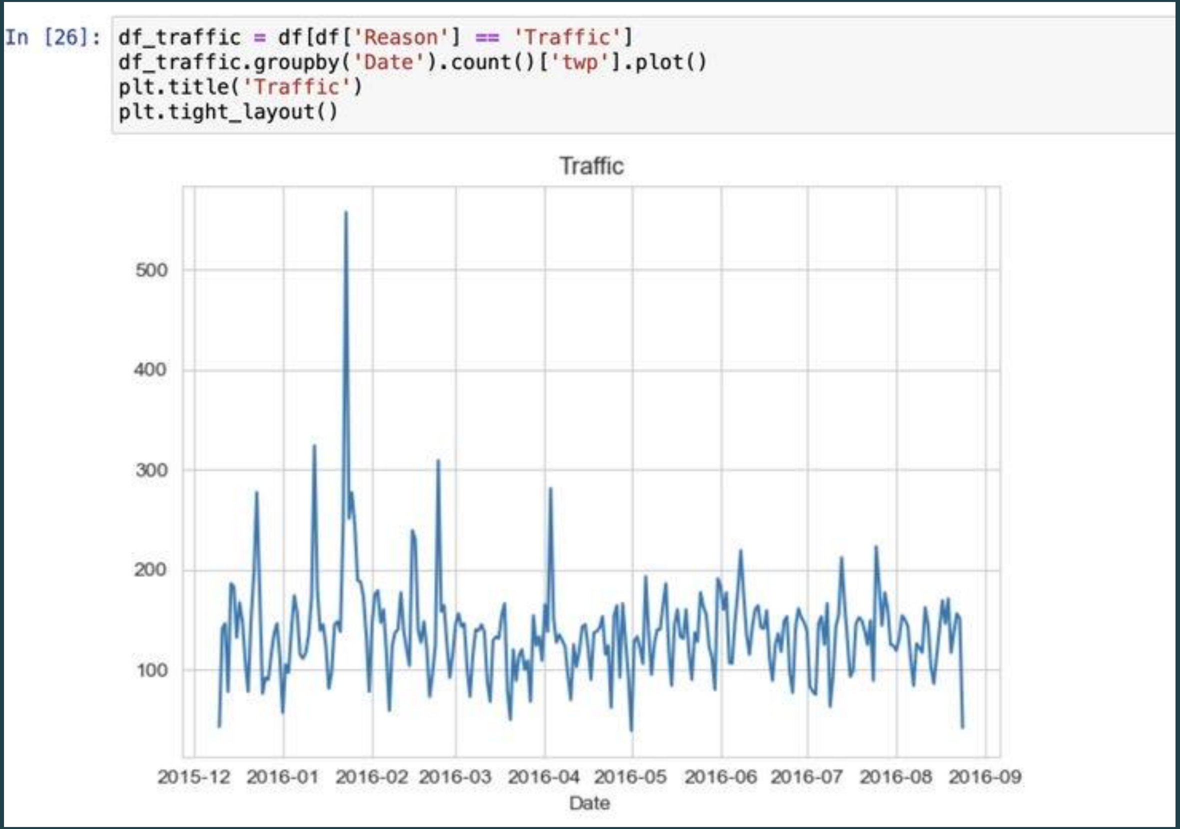
```
In [23]: sns.lmplot(x = 'Month', y = 'twp', data = byMonth)
Out[23]: <seaborn.axisgrid.FacetGrid at 0x7fa71f3c9820>
```

# Groupby Plot of 911 Calls by Date



```
In [25]: df.groupby('Date').count()['twp'].plot()
         plt.tight_layout()
```

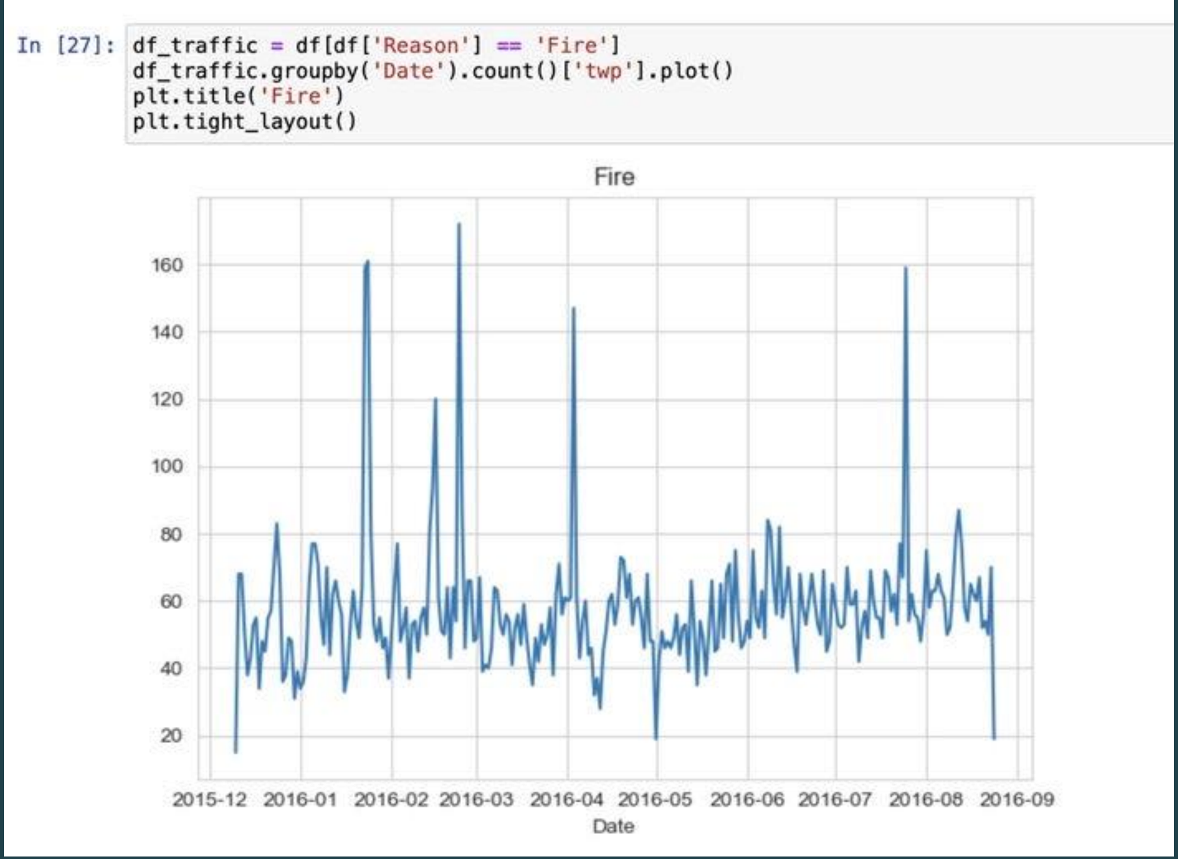
# Groupby Plot With Traffic Calls Data



In [26]:

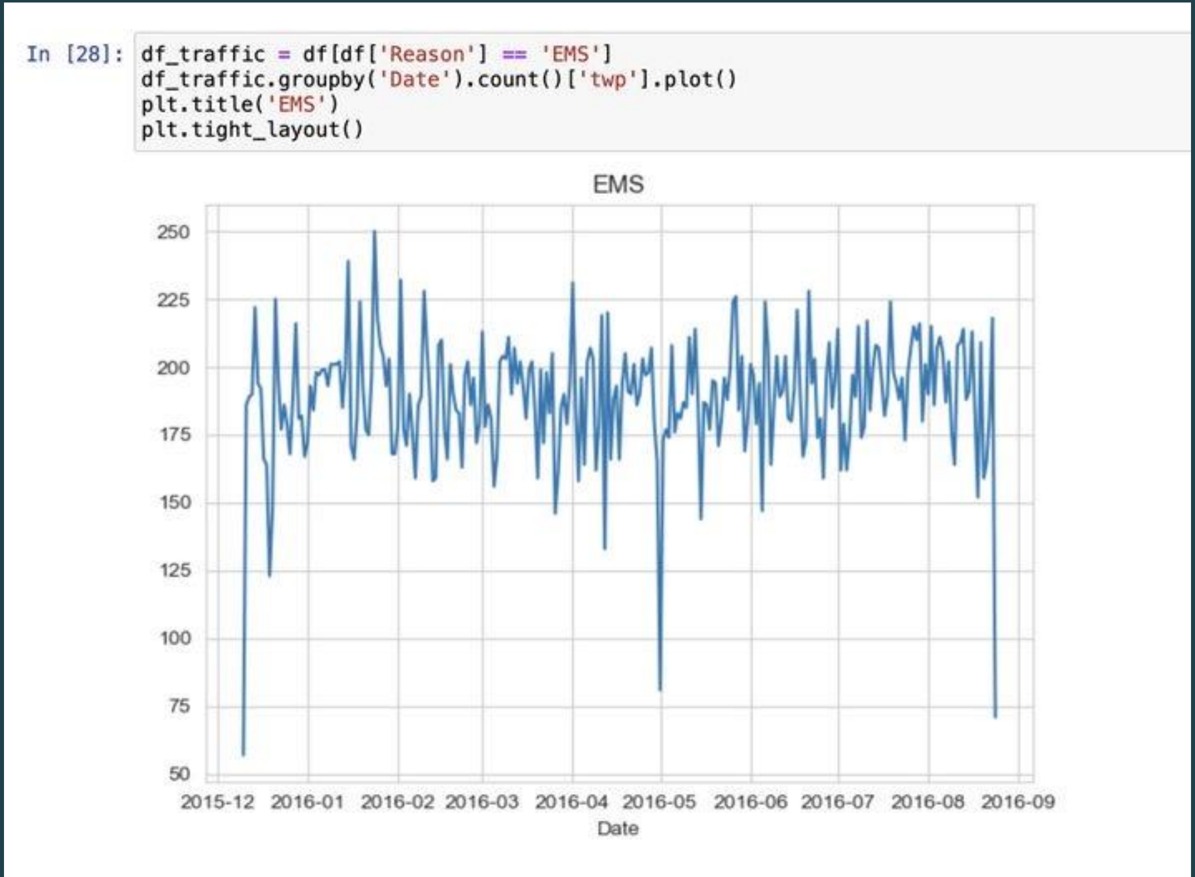
```
df_traffic = df[df['Reason'] == 'Traffic']
df_traffic.groupby('Date').count()['twp'].plot()
plt.title('Traffic')
plt.tight_layout()
```

# Groupby Plot With Fire Calls Data



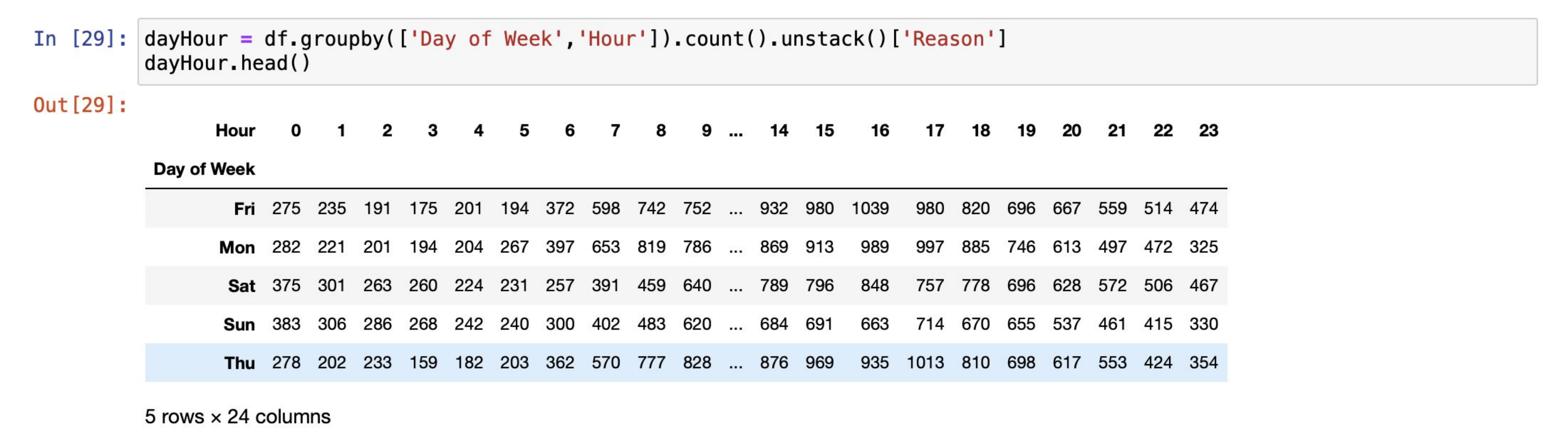
In [27]:
```
df_traffic = df[df['Reason'] == 'Fire']
df_traffic.groupby('Date').count()['twp'].plot()
plt.title('Fire')
plt.tight_layout()
```

# Groupby Plot With EMS Calls Data



```
In [28]: df_traffic = df[df['Reason'] == 'EMS']
         df_traffic.groupby('Date').count()['twp'].plot()
         plt.title('EMS')
         plt.tight_layout()
```

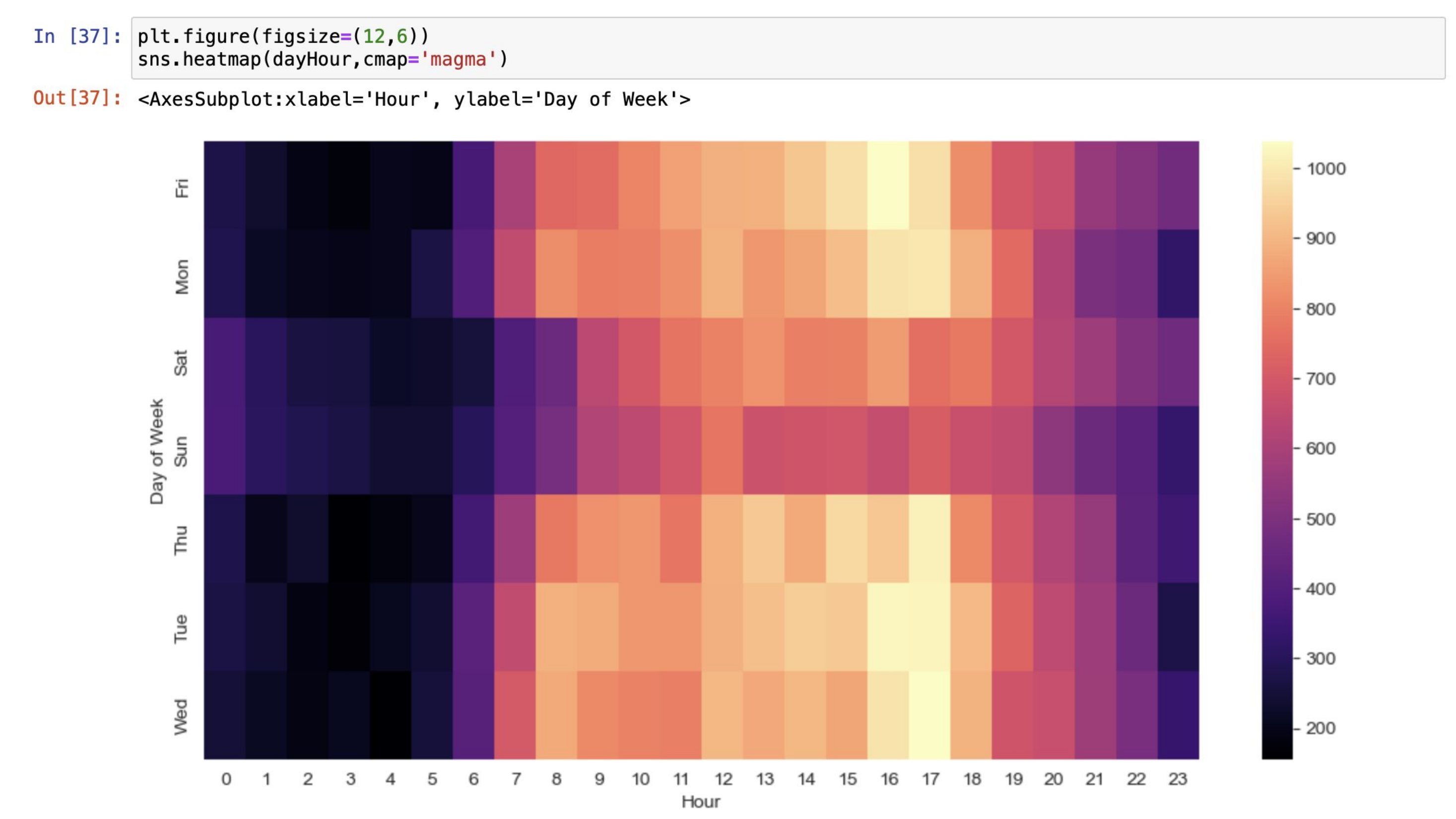
# Restructuring the DataFrame to make Heatmaps



```
In [29]: dayHour = df.groupby(['Day of Week','Hour']).count().unstack()['Reason']
         dayHour.head()
```

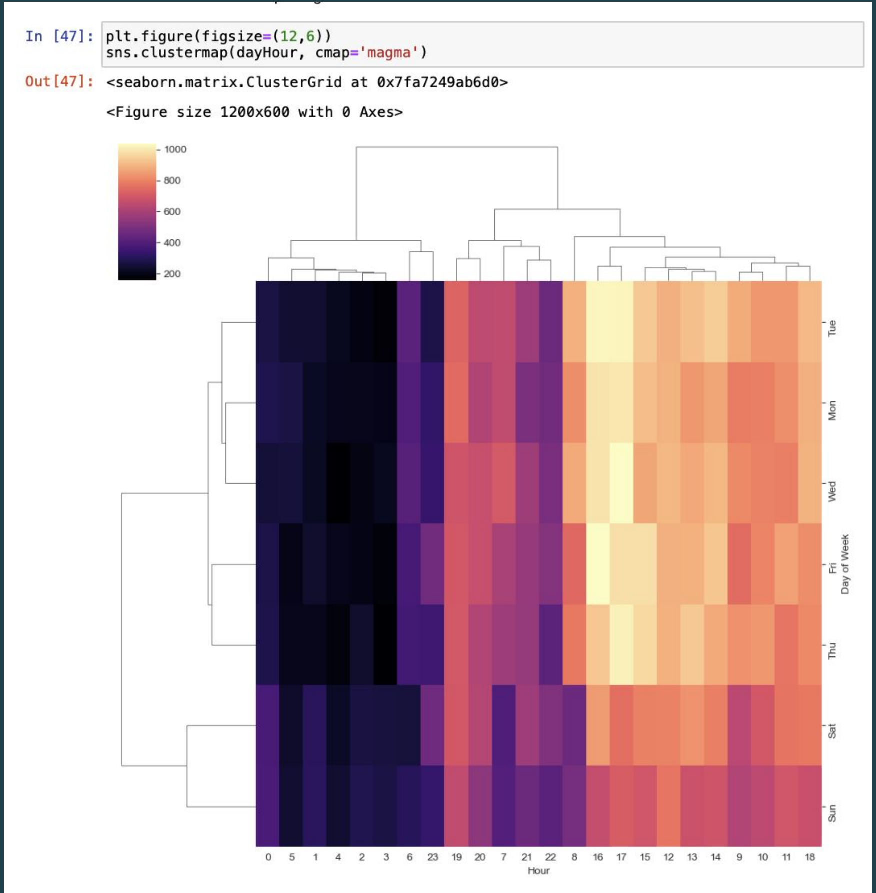
```
Out[29]:
               Hour    0    1    2    3    4    5    6    7    8    9  ...   14   15    16    17   18   19   20   21   22   23
        Day of Week
                Fri  275  235  191  175  201  194  372  598  742  752  ...  932  980  1039   980  820  696  667  559  514  474
                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
```

# Heat Map by Hour



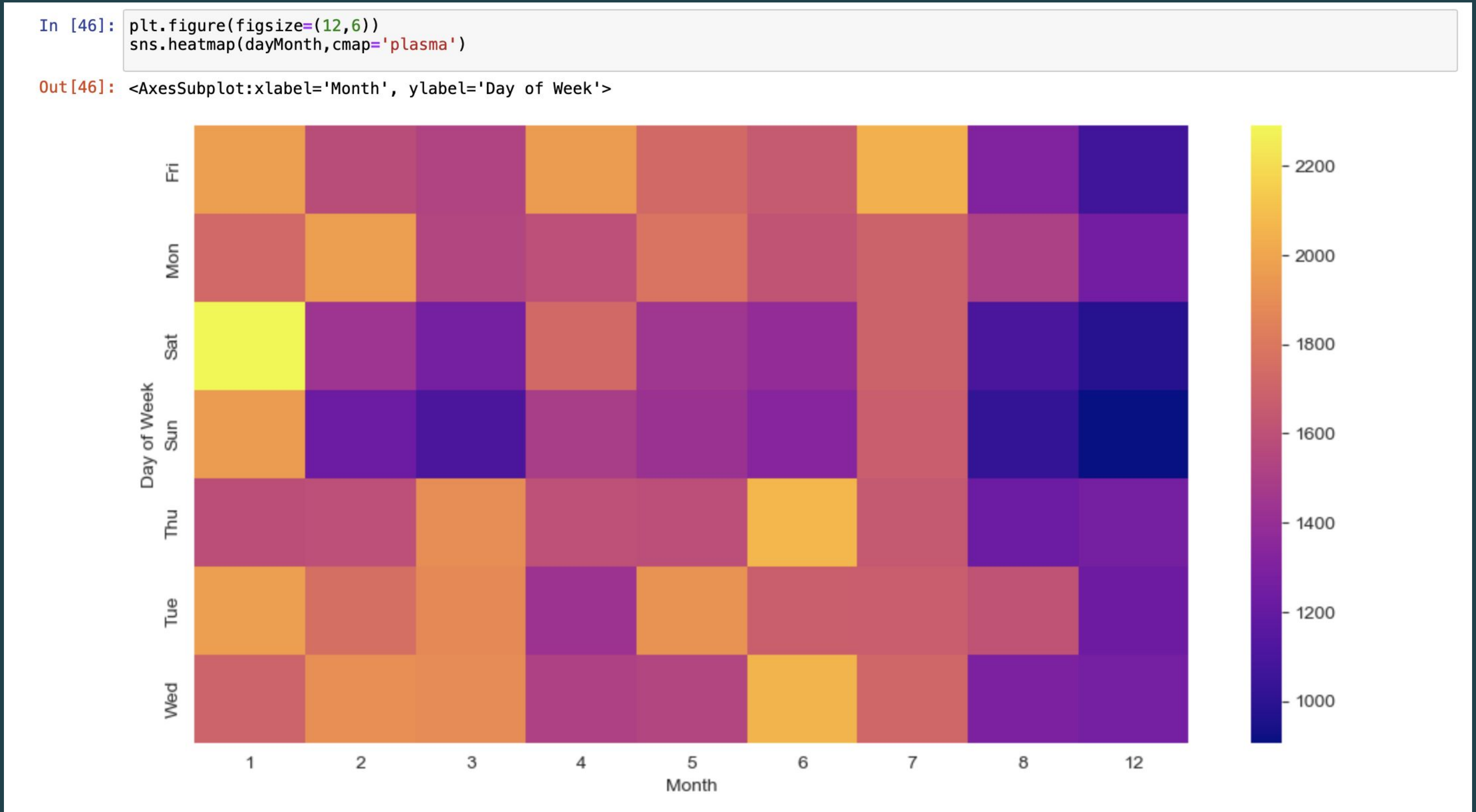
```
In [37]: plt.figure(figsize=(12,6))
         sns.heatmap(dayHour,cmap='magma')

Out[37]: <AxesSubplot:xlabel='Hour', ylabel='Day of Week'>
```

# Cluster Map by Hour



```
In [47]: plt.figure(figsize=(12,6))
         sns.clustermap(dayHour, cmap='magma')

Out[47]: <seaborn.matrix.ClusterGrid at 0x7fa7249ab6d0>

         <Figure size 1200x600 with 0 Axes>
```

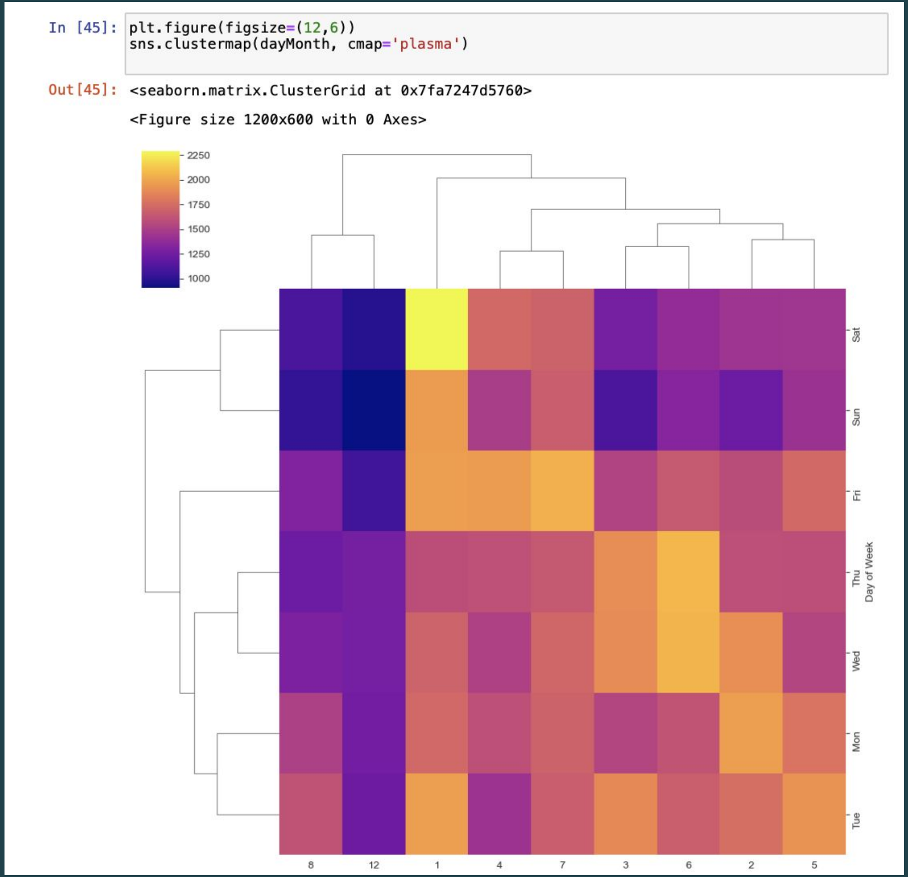
# Heat map by Month Instead of Hour



In [46]:
```
plt.figure(figsize=(12,6))
sns.heatmap(dayMonth,cmap='plasma')
```

Out[46]:
```
<AxesSubplot:xlabel='Month', ylabel='Day of Week'>
```

# Cluster Map by Month



```
In [45]: plt.figure(figsize=(12,6))
         sns.clustermap(dayMonth, cmap='plasma')
```

```
Out[45]: <seaborn.matrix.ClusterGrid at 0x7fa7247d5760>

         <Figure size 1200x600 with 0 Axes>
```